UNIVERSITÀ
DEGLI STUDI
DI UDINE

hic sunt futura

Università degli Studi di Udine
Department of Mathematics, Computer Science, and Physics
Ph.D. in Computer Science and Artificial Intelligence — XXXVIII Cycle

Chronicles on Metaheuristic Optimization

Components, Applications, and Large Language Models

Candidate

Francesca Da Ros

Supervisor

Prof. Luca Di Gaspero

Year 2026

A Teresa Sole,
per avermi insegnato che la misura del mondo
è data da ciò che amiamo.

Abstract

Metaheuristic algorithms play a central role in solving combinatorial optimization problems, yet their empirical study and reliable application remain fragmented. The community has developed a multitude of algorithms and variants, often emphasizing competitive performance over explanatory insight. Consequently, the understanding of algorithmic behavior is limited, and available tools are typically applied only to benchmark problems. On the problem side, the literature has produced numerous variants, leading to a proliferation of isolated formulations. At the same time, emerging technologies such as Large Language Models are beginning to reshape the landscape of optimization research.

This thesis addresses these challenges along four complementary directions: developing a component-level understanding of metaheuristics, extending benchmarking methodologies to real-world problems, unifying fragmented problem formulations, and exploring the integration of emerging technologies such as Large Language Models within optimization.

First, by systematically analyzing the internal components of classical methods such as Tabu Search and Simulated Annealing, the work investigate how design choices shape search dynamics and demonstrates that adaptive control mechanisms can reproduce tuned performance without manual calibration.

Second, the attention turns to a real-world application: the Oven Scheduling Problem, a parallel batch scheduling problem. The study develops theoretical lower bounds and employs Simulated Annealing and Large Neighborhood Search. Statistical analysis and visual benchmarking techniques are integrated to reveal how instance features and search trajectories jointly explain algorithmic performance.

Third, the analytical perspective is extended to the problem domain itself through a unified formulation of the Home Healthcare Routing and Scheduling Problem. We consolidate previously isolated variants and develop open tools for instance generation, analysis, translation, and validation. Additionally, transversal solution methods are introduced, with performance comparable to or exceeding that of specialized algorithms.

Fourth, the thesis explores the role of Large Language Models in optimization, systematically mapping existing research. Then, we shift the focus from what these models can do to how they *reason*. Their behavior and representational capabilities are empirically examined through direct querying and probing experiments.

Altogether, the thesis contributes to a coherent perspective on how optimization methods can be analyzed, unified, and extended, providing both conceptual clarity and practical tools for the study of complex decision problems.

Keywords. Combinatorial optimization, metaheuristics, local search, empirical analysis, large language models, oven scheduling problem, home healthcare routing and scheduling problem.

Acknowledgments

A Ph.D. is both a research journey and a life journey. Over the years, one learns to navigate algorithms and papers, but also the more fragile terrain of grief, friendship, growth, self-doubt, and resilience. The boundaries between these worlds blur, and progress often means learning to move through both at once. Sometimes this happens with clarity, sometimes simply with persistence. These acknowledgements are written from that intersection: between the road proper, shaped by guidance and collaboration, and the quieter road for the trees, where affection and love made the work, and the person behind it, possible.

First, to Luca, whose supervision combined patience with the freedom to pursue my own directions. That independence shaped how I have learned to approach research itself, mirroring his own way of doing research, with curiosity and enthusiasm. I will always remember the hours spent coding together, sharing gummy bears, or exchanging gossip, reminders that curiosity and companionship can coexist even in the most technical of tasks.

Second, to Nysret, for welcoming me in Vienna and for the many conversations that shaped both the methods and the questions in this work. His openness and generosity created a space of genuine exchange.

Third, to Andrea, whose insight and calm support have accompanied my research. Since 2019, when, as a young master's student, I decided to change my track from industrial to information engineering, his office has always been a safe place. That decision, and the trust that followed, marked the true beginning of this journey.

Fourth, to all my co-authors and co-workers. Most of you have been models of the scholar I aspire to be: thoughtful and rigorous. Others, perhaps unknowingly, revealed the pitfalls I hope to avoid. I carry both lessons with gratitude. A special thanks goes to Mimi, whose approach to research and generous way of engaging with others I deeply admire.

Fifth, to the reviewers of this manuscript, Christian Blum and Mike Preuss. I sincerely thank them for the time and attention devoted to reading a manuscript of over 400 pages, and for their encouraging evaluation of this work.

Sixth, to *quelli del primo piano* and *the ones of the third floor*. They have lit the way with humour and patience, helping me to navigate the madness of bureaucracy, deadlines, and experiments. Their laughter and kindness made even the most chaotic moments brighter. A special thank you to Eugenia, with whom I started this Ph.D.: sharing the office and our doubts, questioning our choices, and openly debating life itself made the journey richer. She is also the reason why *quelli del primo piano* exist at all: by organizing dinners and shared moments, she turned this corridor into a community.

Seventh, to David, who has offered balance, perspective, and care when research seemed to absorb everything else. His quiet steadiness has grounded me through uncertainty, and his presence has reminded me of what endures when everything else is in motion.

Finally, to my family, for the love that carried us through these years. From my parents I learned the quiet discipline of getting things done and of working hard. I thank them for

what they taught me and for how they live those lessons every day; their example made perseverance feel natural. To my brother and to Francesca, for their closeness and the sense of normality they always managed to restore, even in the hardest times. To Teresa Sole, whose presence changed us all. And to Luce, who arrived and, true to her name, brought light back into our days.

Francesca

Contents

List of Figures

List of Tables

Symbols

This list gathers the symbols used across the thesis for ease of reference. Note that problem-specific notation is defined locally within each Part of the thesis.

Problem

- $\mathcal{P}$ Optimization problem, also referred to as the problem space.
- $\mathcal{I}$ Set of existing instances of an optimization problem, also referred to as the instance space. A generic instance with x.

Search Space

- $\mathcal{S}$ Search space, i.e., the set of states (or solutions) that an algorithm possibly consider.
- $\mathcal{S}_f$ Subset of the search space containing only feasible solutions.
- s Generic state (or solution) within the search space.
- s^* State (or solution) characterized by the best quality according to an evaluation criterion.
- s_0 Initial state (or solution).

Evaluation

- obj Objective function of the optimization problem.
- F Cost function used by an algorithm to evaluate states (or solutions).
- ΔF_m Delta evaluation, i.e., the change in state (or solution) quality resulting from applying a move m.

Operators

- $\mathcal{N}$ Neighborhood operator defining the set of states (or solutions) reachable from a given state (can indicate a multi-neighborhood).
- m Generic move.
- $\boldsymbol{\sigma}$ Probability bias vector used when applying a multi-neighborhood scheme.

General Policies

C	Candidate set, i.e., a subset of the neighborhood considered for selection.
C_M	Set of moves associated with the candidate set.
τ	Threshold used in threshold-acceptance criteria.
i	Current iteration.
$i_{\max}$	Maximum number of iterations.
i_{idle}	Number of consecutive iterations without improvement.
$\mathcal{T}_{\max}$	Timout

Tabu Search

$\boldsymbol{V}$	Tabu list.
θ	Generic length of the tabu list, used in the fixed-length variant and as a reference size in other variants.
o	Index of the oldest move in the list, used in the fixed-length tabu list.
$\theta_{\min}$	Minimum tabu list length, used in the random-length variant.
$\theta_{\max}$	Maximum tabu list length, used in the random-length variant.
i_{changed}	Number of iterations since the last change of tabu length, used in the cyclic variable-length and reactive tabu lists.
i_{keeping}	Number of iterations a tabu length is kept before changing, used in the cyclic variable-length tabu list.
c	Index of the current tabu length, used in the cyclic variable-length tabu list.
$\mathcal{J}$	Number of tabu length, used in the cyclic variable-length tabu list.
$\boldsymbol{\Theta}$	Set of possible tabu tenures, used in the cyclic variable-length tabu list.
$\boldsymbol{H}$	Solution history, used in the reactive tabu list.
ω^+	Factor for increasing the tabu list length, used in the reactive tabu list.
ω^-	Factor for decreasing the tabu list length, used in the reactive tabu list.
ℓ	Cycle length, used in the reactive tabu list.
avg_ℓ	Average cycle length, used in the reactive tabu list.
ξ	Iteration threshold for detecting chaotic situations, used in the reactive tabu list.
k	Chaos counter, used in the reactive tabu list.
κ	Chaos threshold, used in the reactive tabu list.

μ Cycle-length threshold, used in the reactive tabu list.

ϕ Frequency, used in the long-term frequency-based tabu list.

Simulated Annealing

T_0 Initial temperature.

T_f Final temperature.

T Current temperature.

α Cooling rate, i.e., factor controlling the decrease of the temperature.

N_s Number of iterations per temperature level (run counter denoted as n_s).

N_a Number of accepted moves per temperature level (run counter denoted as n_a).

ρ Neighborhood acceptance ratio, i.e., $\rho = N_a/N_s$.

$R_{\max}$ Maximum number of reheats, used in simulated annealing with reheating.

T_r Reheating ratio.

β Proportion of the run spent on the first temperature decrease.

Large Neighborhood Search

$\mathcal{D}$ Set of destroy operators (a selected operator is denoted by d).

$\mathcal{R}$ Set of repair operators (a selected operator is denoted by r).

$\boldsymbol{\sigma_d}$ Probability bias vector used to select a destroy operator.

$\boldsymbol{\sigma_r}$ Probability bias vector used to select a repair operator.

Hyper-Heuristics

h Generic low-level heuristic.

Instance Space Analysis and Algorithm Selection

$\mathcal{F}$ Feature space.

$\boldsymbol{f}(x)$ Feature vector of an instance x, i.e., $\boldsymbol{f}(x) \in \mathcal{F}$.

$\boldsymbol{z}(x)$ Embedding instance x in the instance space.

$\mathcal{A}$ Algorithm space or portfolio of algorithms.

ψ Generic algorithm in the portfolio.

ψ^* Winning algorithm in the portfolio, i.e., the one achieving the best performance on a given instance.

$\mathcal{Y}$ Performance space.

$y(\psi, x)$	Performance of algorithm ψ on instance x.
ϵ	Performance tollerance, so to define *good* algorithms.
G	Subscript indicating a *good* algorithm.
B	Subscript indicating a *bad* algorithm, using the definition of ϵ.
Pr	Probability of selectiong a *good* algorithm.

Search Trajectory Networks

n	Number of locations.
n_ψ	Number of locations visited by algorithm ψ.
n^*_ψ	Number of locations visited only by algorithm ψ.
l_ψ	Average trajectory length of algorithm ψ.
r_ψ	Success rate of algorithm ψ.

Acronyms

ACO Ant Colony Optimization.
AEC Architecture, Engineering, and Construction Industry.
AI Artificial Intelligence.
ALNS Adaptive Large Neighborhood Search.
ANN Approximate Nearest Neighbor.
ANOVA Analysis of Variance.
API Application Programming Interface.
ASP Algorithm Selection Problem.
AVNS Adaptive Variable Neighborhood Search.

BBSR Benchmarking, Benchmarks, Software, and Reproducibility.
BI Best Improvement.
BPMN Business Process Model and Notation.
BPP Bin Packing Problem.
BRKGA Biased Random-Key Genetic Algorithm.

CBCTT Curriculum-based Course Timetabling.
CD Critical Difference.
CHeSC 2011 Cross Domain Heuristic Search Challenge 2011.
CLI Command Line Interface.
CO Combinatorial Optimization.
COP Combinatorial Optimization Problem.
CP Constraint Programming.
CSP Constraint Satisfaction Problem.
CVRP Capacited Vehicle Routing Problem.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.
DGM Deep Generative Model.
DPP Decap Placement Problem.
DSL Domain-Specific Language.
DT Decision Tree.

EA Evolutionary Algorithm.
EC Evolutionary Computation.
ETT Educational Timetabling.

FI First Improvement.
FTSA Fixed Temperature Simulated Annealing.

GA Genetic Algorithm.
GAI Generative Artificial Intelligence.
GCP Graph Coloring Problem.
GECCO Genetic and Evolutionary Computation Conference.
GIHH Generic Intelligent Hyper-Heuristic.
GPT Generative Pre-trained Transformer.
GRASP Greedy Randomized Adaptive Search Procedure.
GVN General Variable Neighborhood Search.

HAC Hierarchical Agglomerative Clustering.
HC Hill Climbing.
HH Hyper-Heuristic.
HHC Home Healthcare.
HHCRSP Home Healthcare Routing and Scheduling Problem.
HHSA Hyper-Heuristic-based Simulated Annealing.

ILP Integer Linear Programming.
ILS Iterated Local Search.
ISA Instance Space Analysis.

JSP Jobshop Scheduling Problem.

k-GCP k-Graph Coloring Problem.

KNN K-Nearest Neighbors.
KP Knapsack Problem.

L-GIHH Lean Generic Intelligent Hyper-Heuristic.
LAHC Late Acceptance Hill Climbing.
LB Lower Bound.
LLH Low Level Heuristic.
LLM Large Language Model.
LNS Large Neighborhood Search.
LON Local Optima Network.
LP Linear Programming.
LS Local Search.
LSRL Large State Reinforcement Learning.

MA Memetic Algorithm.
max-SAT Maximum Satisfiability Problem.
MCFP Minimum Cost Flow Problem.
MF Most Frequent.
MH Metaheuristic.
MILP Mixed Integer Linear Programming.
MIP Mixed Integer Programming.
MitL Metaheuristic in the Large.
MKP Multiple Knapsack Problem.
ML Machine Learning.
MOO Multi-Objective Optimization.

NFL No Free Lunch.
NL Natural Language.
NL4Opt Natural Language for Optimization Modeling.
NLP Natural Language Processing.
NN Neural Network.
NSGA-II Non-dominated Sorting Genetic Algorithm II.

OP Orienteering Problem.
OPL Optimization Programming Language.
OR Operations Research.
OSP Oven Scheduling Problem.
OSRM Open Source Routing Machine.

PbO Programming by Optimization.
PCA Principle Component Analysis.
PFSP Permutation Flowshop Scheduling Problem.
PRISMA Preferred Reporting Items for Systematic Reviews and Meta-Analyses.
PSO Particle Swarm Optimization.

QP Quadratic Programming.
QUOROM Quality of Reporting of Meta-analyses.

RAG Retrieval-Augmented Generation.
RF Random Forest.
RL Reinforcement Learning.
ROAR-NET Randomized Optimization Algorithms Research Network.
RQ Research Question.

SA Simulated Annealing.
SE Shannon Entropy.
SHAP SHapley Additive exPlanations.
SRP Selective Routing Problem.
STN Search Trajectory Network.
SVM Support Vector Machine.

TS Tabu Search.
TSP Traveling Salesperson Problem.

UC Use Case.
UN SDG United Nations Sustainable Development Goal.

VBP Value-Based Purchasing.
VNS Variable Neighborhood Search.
VRP Vehicle Routing Problem.

Part I

Fundamentals

Chapter 1

Introduction

In everyday life, we constantly face situations that require us to choose among competing alternatives. Some of these can be interpreted as optimization problems, where the task is to identify the best outcome according to given criteria. For instance, when exploring a new city, one may wish to plan a round trip that visits each monument exactly once while minimizing the total walking distance. Similarly, when choosing a restaurant, one often weighs the quality of the food against its price. Many such problems belong to the field of Combinatorial Optimization (CO), which studies decision problems where solutions are drawn from a discrete set and the objective is to identify the most desirable one according to some specifications. These problems are central across multiple domains and capture the structure of real-world decision-making scenarios, from logistics [63, 84] and scheduling [28] to network design [355] and resource allocation [128].

In CO, when the number of alternatives is small, the choice may be made intuitively or by simple enumeration. As the size and complexity of the problem grow, however, the space of possibilities expands combinatorially, quickly surpassing the limits of exhaustive reasoning. In such cases, algorithms are needed to tackle these problems efficiently. In many practical settings, what matters is not necessarily finding the provably optimal solution but obtaining one of sufficiently high quality within a reasonable amount of time. Heuristic algorithms are therefore widely used, trading guarantees of optimality for computational tractability. Yet, many heuristics are problem-specific and difficult to generalize across domains. To address this limitation, researchers have developed metaheuristics [199], high-level strategies for search and optimization that can be adapted to a broad range of problems [56, 439, 480]. Metaheuristics have achieved major success in optimization, with applications in fields such as public transportation [257], examination timetabling [44], scheduling [379, 381], and planning [278].

Over the years, many metaheuristic methods have been proposed and applied to thousands of optimization problems, resulting in a vast and diverse body of literature.[1] While this richness demonstrates their versatility and impact, it also raises challenges: the abundance of approaches makes systematic comparison difficult, and the proliferation of variants has sometimes obscured rather than clarified our understanding of what drives performance. This is particularly evident in the case of metaphor-based metaheuristics,

[1] At the time of writing (2025–10–28), a search for the keyword *metaheuristic* in the Scopus database yields more than 37,700 results.

which are algorithms whose design and terminology are inspired by processes observed in nature, society, or other domains [438]. In this case, what initially started with the noble intent of helping to understand the solution method soon turned into a trend of inventing ever more exotic metaphors.[2] Critical assessments have shown that such algorithms often fail to accurately reflect the processes they claim to represent, that key elements of the metaphors are modified or omitted in their implementation, or that the resulting procedures are essentially equivalent to techniques already present in the literature [26, 74, 75]. As a result, the final algorithms frequently diverge from the metaphors from which they draw inspiration, the metaphor risks confusing the reader, and obscuring the underlying method, while the process itself remains largely unexplained to the end user, undermining confidence and trust.

Beyond metaphor-driven approaches, the sheer number of algorithmic variants presents an additional challenge. New methods often introduce only minor modifications to generic metaheuristic templates, typically to better cope with a specific scenario. For instance, Franzin and Stützle [183] identified seven (7) customization points within the Simulated Annealing (SA) algorithm, each associated with between four (4) and sixteen (16) distinct versions reported in the literature. This richness often leaves practitioners with a dilemma: which algorithm should be used for a given case, and whether a different choice might have been more effective. In practice, navigating this landscape often reduces to following the most common choices in related applications, testing a few alternatives, and ultimately relying on individual biases and preferences [470].

This situation is further exacerbated by the so-called *horse-race* culture [246] that has characterized much of the optimization community. Research contributions are frequently evaluated in terms of marginal performance improvements on selected benchmark instances, rather than by the insights they provide into algorithmic behavior, generalizability, or reproducibility. As the emphasis of many works lies in outperforming competitors, comparisons among algorithms may suffer from biased experimentation and results can be influenced by external factors such as coding skills or the programming language used [470]. The coding, setup, and tuning of algorithms focused on short-term gains can yield specific improvements, but often lack longevity: when applied to new problems, or even new instances of the same problem, the entire procedure must typically be repeated [54]. In response to these limitations, a growing body of research has begun to advocate for new benchmarking methodologies that move beyond simple performance comparisons. These include visual approaches such as Instance Space Analysis (ISA) [436] and Search Trajectory Networks (STNs) [370], as well as component-based analyses [183] and meta-analyses [470].

These issues, however, are not limited to algorithms alone, but also extend to the optimization problems themselves. The literature often presents a multitude of closely related problem variants, differing only in notation, scope, or in the inclusion of minor constraints. For example, within the family of Vehicle Routing Problem (VRP), one can find variants with or without time windows, dial-a-ride formulations, pick-up and delivery constraints, time-dependent or stochastic travel times, profit-based objectives, split deliveries, release dates, and integrated formulations that combine routing with location or scheduling decisions [27]. Additional extensions consider uncertainty, the

[2] The so-called Metaheuristic Bestiary (accessed 2025–10–28) collects examples of these algorithms; e.g., Bee Colony Algorithm, Bird Migration Algorithm, etc.

adoption of new technologies, or domain-specific applications such as waste collection. These variants may reflect the requirements of specific applications, but their proliferation fragments the research landscape and makes systematic comparison across studies difficult. Consequently, both algorithm designers and practitioners are confronted with a double challenge: an overwhelming number of algorithmic choices on one hand, and a highly heterogeneous set of problem formulations on the other.

Furthermore, the design and engineering of optimization tasks and applications remain primarily human-driven. Users must convert the problem into an optimization model by defining a set of variables, constraints, and one or more objective functions, then coding and running a software solver or algorithm to find solutions. These activities are not trivial and require a certain extent of expertise. The recent advent of Large Language Models (LLMs) has opened new perspectives for optimization research.These models are trained on massive corpora of text and exhibit remarkable capabilities in reasoning, problem-solving, and code generation [171, 230, 341, 500]. Their generality has sparked growing interest in exploring whether they can support, or even replace, traditional algorithm design and application workflows. At the same time, the rapid adoption of LLMs has outpaced a rigorous understanding of their role in optimization. Fundamental issues remain concerning their explainability, the comparability of results across prompting or fine-tuning strategies, and the reproducibility of their outputs, given their inherent stochasticity and dependence on proprietary infrastructures.

1.1 Motivations

Several factors motivate this thesis. Despite the widespread use of metaheuristics, our understanding of their inner components remains limited, as most studies emphasize competitive testing over explanatory insight. At the same time, advanced benchmarking methodologies such as ISA and STNs, have been applied mainly to simplified benchmarks, with comparatively little attention devoted to real-world optimization problems. On the problem side, the literature often introduces numerous variants and formulations of essentially the same problem, fragmenting the research landscape and hindering systematic comparison. Finally, the rapid rise of Generative Artificial Intelligence (GAI), and in particular LLMs, has sparked strong interest in optimization, yet existing research remains exploratory, with limited attention to transparency and reproducibility.

1.2 Goals

This thesis pursues the following objectives:

- **Component-level understanding.** To develop systematic analyses of metaheuristic components, investigating how design choices such as tabu list strategies in Tabu Search (TS) or temperature control policies in SA influence search behavior and performance, thereby moving beyond performance-oriented competitive testing.

- **Advanced benchmarking on real-world problems.** To extend the use of advanced benchmarking methodologies, including ISA and STNs, beyond toy benchmarks,

demonstrating their value in studying a complex, real-world optimization problem, namely the Oven Scheduling Problem (OSP).

- **Unified formulations of problem variants.** To propose unifying formulations for families of related problems, supported by formal models and standardized instance representations, in order to enable systematic comparison and practical applicability. We use a relevant problem in healthcare that combines both scheduling and routing, i.e., the Home Healthcare Routing and Scheduling Problem (HHCRSP).
- **Critical exploration of LLMs in optimization.** To examine the potential and limitations of LLMs for optimization through a systematic literature review and controlled probing experiments.

1.3 Positioning within the Community

Although this thesis is not officially supported by any specific initiative other than the Università degli Studi di Udine, it is closely connected to two ongoing efforts in the research community: the Metaheuristic in the Large (MitL) initiative and the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action.[3]

The MitL initiative was introduced by Swan et al. [451] with the aim of providing reusable and extensible templates for metaheuristics, thereby enhancing both reproducibility and comparability in optimization research [450].

The ROAR-NET Cost Action[4] seeks to make randomized optimization algorithms broadly competitive in practice by identifying and reducing barriers to adoption at the scientific, technical, economic, and human levels. The Action is organized into six working groups, including those on problem modeling, benchmarking, and algorithm selection and configuration, with a strong focus on the needs of practitioners, from whom the economic value of optimization tools ultimately derives. Although this thesis has not been directly funded by ROAR-NET, the author has been actively involved in its activities, and the themes and objectives of the thesis align closely with its scope. Throughout the work, explicit indications are given where specific contributions relate to ROAR-NET Cost Action topics.

1.4 List of Publications

All research presented in this thesis is the original work of the author, carried out under the supervision of Prof. Luca Di Gaspero, with contributions from collaborators at the Università degli Studi di Udine and other national and international institutions. Portions of this work have been published in peer-reviewed journals and presented at international conferences, while additional manuscripts are currently under review. A complete list of publications produced during the three-year doctoral program is provided below in reverse chronological order.

[3] A Cost Action is an interdisciplinary network that brings researchers and innovators together to investigate a chosen topic over a period of four years.

[4] https://roar-net.eu/. Accessed 2025–10–28.

Journal Article (5)

- Ceschia, S., **Da Ros, F.**, Di Gaspero, L., Mancini, S., Maniezzo, V., Montemanni, R., Rosati, R. M., & Schaerf, A. (2025, submitted). A unified formulation for home healthcare routing and scheduling problems. *International Transactions in Operational Research*. 1–47. Journal rank: Scimago Q1 (2024).

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., Musliu, N., & Winter, F. (2025). Multi-neighborhood simulated annealing for the oven scheduling problem. *Computers & Operations Research, 177*, 106999. Journal rank: Scimago Q1 (2024).

- **Da Ros, F.**, Di Gaspero, L., Roitero, K., La Barbera, D., Mizzaro, S., Della Mea, V., Valent, F., & Deroma, L. (2024). Supporting fair and efficient emergency medical services in a large heterogeneous region. *Journal of Healthcare Informatics Research, 8*, 400–437. Journal rank: Scimago Q1 (2024).

- Maddalena, E., Mizzaro, S., Roitero, K., Viviani, M., La Barbera, D., Modha, S., Pasi, G., **Da Ros, F.**, & Soprano, M. (2024). Report on the 14th Italian Information Retrieval Workshop (IIR 2024). *SIGIR Forum, 58*, 1–13.

- Spina, D., Roitero, K., Mizzaro, S., Della Mea, V., **Da Ros, F.**, Soprano, M., Akebli, H., Falcon, A., Fasihi, M., Fiorin, A., *et al.* (2024). Report on the Hands-on PhD Course on responsible AI from the lens of an information access researcher. *SIGIR Forum, 58*, 1–61.

Conference and Workshop Papers (13)

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., Musliu, N., & Soprano, M. (2025). Search trajectory networks applied to a real-world parallel batch scheduling problem. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (pp. 68–85). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

- **Da Ros, F.**, Di Gaspero, L., & Roitero, K. (2025). Probing LLMs on optimization problems: Can they recall and interpret problem features? In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (pp. 353–371). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., & Musliu, N. (2025). Instance space analysis and algorithm selection for a parallel batch scheduling problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)* (pp. 66–83). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

- **Da Ros, F.**, & Di Gaspero, L., Kletzander, L., Lackner, M. L., Musliu, N., & Schaerf, A. (2025). Dynamic Temperature Control of Simulated Annealing using Hyper-Heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 184–194). New York, NY: ACM. Conference rank: CORE2023 A.

- Soprano, M., Maddalena, E., **Da Ros, F.**, Zuliani, M. E., & Mizzaro, S. (2025). Evaluation of crowdsourced peer review using synthetic data and simulations. In *CEUR Workshop Proceedings*.

- **Da Ros, F.**, & Di Gaspero, L. (2024). An empirical analysis of tabu lists. In *Metaheuristics International Conference, Part II* (pp. 50–64). Cham: Springer International Publishing.
- **Da Ros, F.**, Lackner, M., & Musliu, N. (2024). Theoretical lower bounds for the oven scheduling problem. In *Proceedings of the 14th International Conference on the Practice and Theory of Automated Timetabling*. Conference rank: CORE2023 C.
- **Da Ros, F.**, & Di Gaspero, L., Lackner, M. L., Musliu, N., & Winter, F. (2024). Local Search Algorithms for the Oven Scheduling Problem. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 191–194). New York, NY: ACM.
- **Da Ros, F.**, & Di Gaspero, L., Lackner, M. L., & Musliu, N. (2024). Reducing Energy Consumption in Electronic Component Manufacturing through Large Neighborhood Search. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 1706–1714). New York, NY: ACM.
- Ceschia, S., **Da Ros, F.**, Di Gaspero, L., & Schaerf, A. (2024). EasyLocal++ a 25-year Perspective on Local Search Frameworks: The Evolution of a Tool for the Design of Local Search Algorithm. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 1658–1667). New York, NY: ACM.
- **Da Ros, F.**, & Di Gaspero, L. (2023). Local Search Strategies for Multi-Objective Flowshop Scheduling: Introducing Pareto Late Acceptance Hill Climbing. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 61–62). New York, NY: ACM.
- **Da Ros, F.**, & Di Gaspero, L. (2023). Exploring the potential of JuLeS: A white box framework for local search metaheuristics. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 191–194). New York, NY: ACM.
- **Da Ros, F.**, Di Gaspero, L., La Barbera, D., Della Mea, V., Roitero, K., Deroma, L., Licata, S., & Valent, F. (2022). A multi-objective biased random-key genetic algorithm for the siting of emergency vehicles. In *Metaheuristics International Conference* (pp. 449–456). Cham: Springer International Publishing.

Pre-prints and Under Review Articles (3)

- **Da Ros, F.**, Soprano, M., Di Gaspero, L., & Roitero, K. (2024, submitted). Large language models for combinatorial optimization: A systematic review. *Submitted to a journal*. Available as preprint at https://doi.org/10.48550/arXiv.2507.03637.
- **Da Ros, F.**, Di Gaspero, L., & Roitero, K. (2025, submitted). Behavior and representation in large language models for combinatorial optimization: From feature extraction to algorithm selection. *Submitted to a journal*. Available as preprint at https://doi.org/10.48550/arXiv.2512.13374.
- Lackner, M. L., **Da Ros, F.**, & Musliu, N. (2025, submitted). Theoretical lower bounds for a parallel batch scheduling problem. *Submitted to a journal*.

1.5 Structure of the Thesis

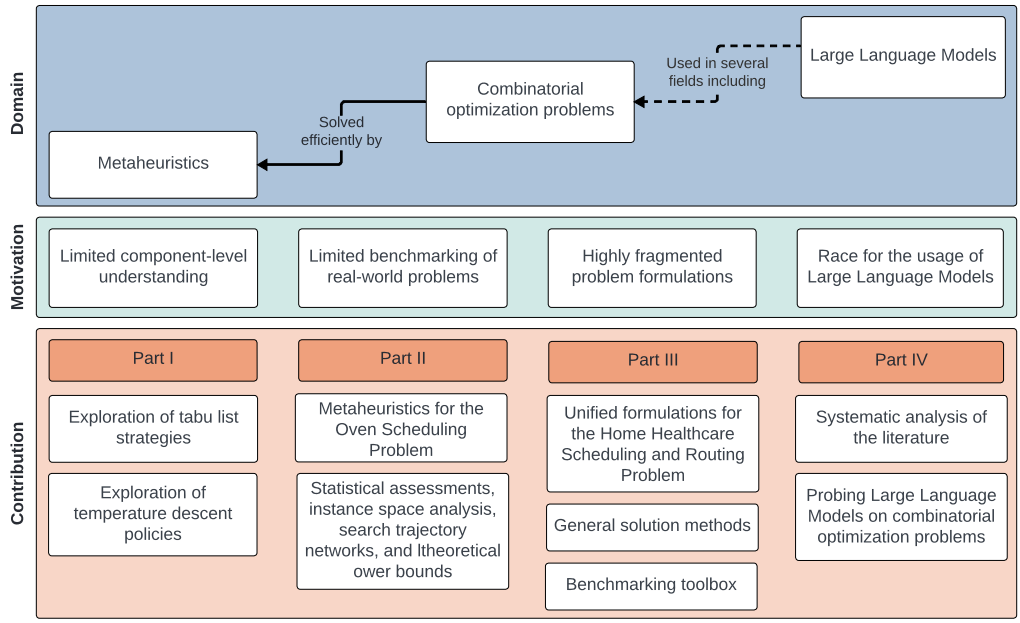


Figure 1.1: Overview of the structure of this thesis.

This thesis is organized into six parts (Part I to Part VI), as follows.

- **Part I — Fundamentals.** We set the stage for the thesis as follows. In addition to the present introduction, it includes a background chapter (Chapter 2), which summarizes the main notions and concepts adopted throughout the work, and a related work chapter (Chapter 3).

- **Part II — Metaheuristic Components.** We investigate the inner workings of metaheuristics through a component-based approach. First, we analyze TS by systematically comparing several tabu list strategies proposed in the literature, studying how they interact when all other components are kept fixed. We also contrast two different interpretations of tabu restrictions, one stricter than the other. Second, we focus on SA, and propose the use of Hyper-Heuristics (HHs) to dynamically adjust the temperature schedule, based on the observation that the balance between exploration and exploitation depends on the characteristics of the search landscape at a given point in time.

- **Part III — The Oven Scheduling Problem.** We study the OSP, a real-world parallel batch scheduling problem formulated in collaboration with an industrial partner. We develop metaheuristic methods to solve it and analyze their performance using both traditional benchmarking tools (i.e., statistical assessment) and more advanced methodologies. In addition, we consider a related problem and test the approach that achieved the majority of wins in our experiments, namely SA.

- **Part IV — The Home Healthcare Routing and Scheduling Problem.** We review the existing literature on the HHCRSP and integrate several related formulations

into a unified model. Furthermore, we standardize the surrounding ecosystem (i.e., instances, features, and evaluation criteria) necessary to correctly benchmark the problem, and we propose two general solvers that can be applied to it.

- **Part V — Large Language Models for Combinatorial Optimization.** We conduct a systematic literature review on LLMs for CO and report our findings using Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines. Building on this, we design probing experiments to assess to what extent such models can capture and exploit features of Combinatorial Optimization Problems (COPs).

- **Part VI — Concluding Remarks.** We summarize the main findings of the thesis and discuss directions for future research.

Figure 1.1 provides an overview of the thesis structure, outlining the addressed domains, underlying motivations, and the contributions of each part.

With the exception of Parts I and VI, every part follows a common sequence of chapters with a consistent internal organization. The opening chapter introduces the topic under consideration, situates it within the scientific context, highlights current limitations in the literature, and formulates specific goals and research questions. For parts that address a specific problem domain (i.e., Part III on the OSP and Part IV on the HHCRSP), this introduction is followed by a problem definition chapter. This chapter adopts a dual format: a description in natural language aligned with the standards promoted by ROAR-NET, and a formal mathematical model of the problem. In addition, a section on problem characteristics provides a rigorous treatment of instances and solutions, thereby establishing a precise foundation for theoretical discussion, empirical analysis, and comparability. Each part then includes one or more research chapters, which follow a uniform structure: an introductory section, a detailed account of the methodology, experimental setup, and results, and a concluding section that synthesizes findings, acknowledges limitations, and suggests directions for future research.

In addition to the main chapters, this thesis includes a series of appendices that complement and extend the presented research. Appendix A reports the artifacts related to this work; all materials are publicly available, and the corresponding links are provided therein. Appendix B documents the EasyLocal++ framework, a white-box metaheuristic environment employed in most of the code developments presented in this thesis. The subsequent appendices provide supplementary material specific to individual chapters, whose necessity is explicitly indicated within the manuscript:

- Appendix C details the analysis of the instances used in Chapter 5.
- Appendix D provides additional material on the OSP solution methods of Chapter 10.
- Appendix E reports the complete list of features employed in the ISA of Chapter 11.
- Appendix F describes the instance features of the HHCRSP introduced in Chapter 14.
- Appendix G presents tabular analysis of the literature reported in Chapter 17.
- Appendix H presents additional analysis w.r.t. the probing experiments of Chapter 18.

Chapter 2

Background

This chapter consolidates the methodological background, including core definitions, algorithmic procedures, and statistical tests. Its objectives are to establish the terminology that will be used consistently throughout the work, to clarify the distinction between problem-dependent and problem-independent components of metaheuristics, to provide the necessary context for Large Language Models (LLMs), and to equip the reader with the statistical toolkit required to interpret the empirical results.

Portions of this chapter draw on the author's published or submitted work, as well as didactic material she prepared over the course of her Ph.D. Regarding the latter, the section on local search algorithms (Section 2.3) is based on lecture material prepared by the author for the First ROAR-NET Cost Action training school.[1]

This chapter is organized as follows. Section 2.1 introduces core concepts in Combinatorial Optimization (CO). Section 2.2 introduces examples of Combinatorial Optimization Problems (COPs). Sections 2.3 and 2.4 present the metaheuristic methods used in this thesis. Section 2.5 summarizes background on LLMs. Finally, Section 2.6 defines the evaluation metrics, statistical procedures, and benchmarking methodologies.

2.1 Combinatorial Optimization

COPs are a class of optimization problems defined by discrete decision variables and the objective of finding one or more optimal solutions within a finite search space of solutions [325]. Many of these problems are classified as NP-hard [189, 270], meaning that, based on current knowledge, they require exponential time to be optimally solved.

Figure 2.1 outlines the steps practitioners and researchers undertake to address an optimization problem [48, 467]. We now detail the process specifically for the case of COPs.

2.1.1 Problem Modeling

The first step regards the description in Natural Language (NL) of the decision-making process or real-world issue to be tackled. Its specifications must be outlined to identify the decisions to be made, the rules that must be followed, and the goals to be achieved. Such description has also been formalized within the Randomized Optimization Algorithms

[1] https://roar-net.eu/events/first-training-school/. Accessed 2025–09–22.

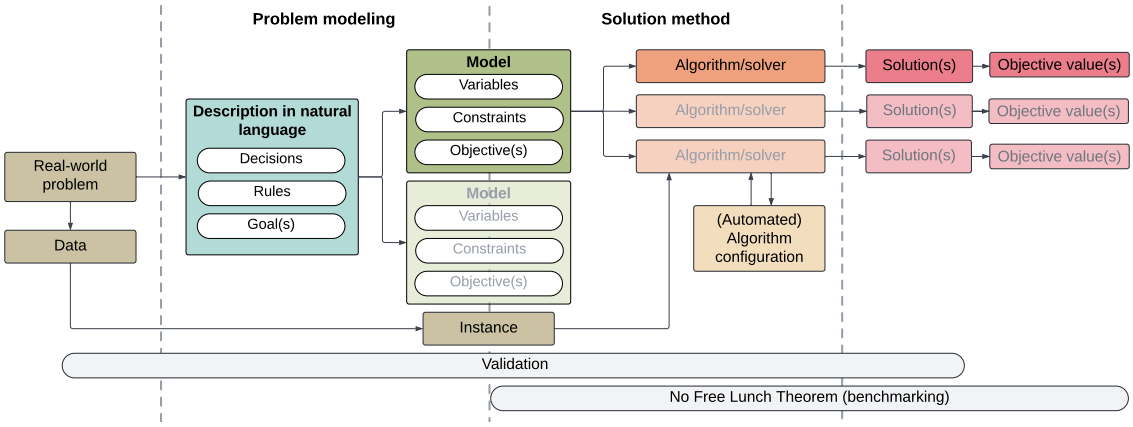


Figure 2.1: Overview of the steps for addressing an optimization problem.

Research Network (ROAR-NET) Cost Action,[2] in order to enable practitioners to easily identify the optimization task to be performed.

After framing the decision-making process, the practitioner must create a formal representation of it. A given NL description can correspond to multiple representations, called *models* or *problem formulations*. Models are articulated in terms of:

- *Variables* (or decision variables), which represent the decisions to be made;
- *Constraints*, which are restrictions on the possible values of the variables and capture problem substructures and/or business rules;
- At least one *objective function*, which assesses the quality of a solution. When one has more than one objective function, it is the case of *multi-objective optimization*.

The development of an effective problem formulation is known as *modeling*. How we define variables, objectives, and constraints determines the model type. E.g., if variables assume integer values and constraints are linear, the model is an Integer Linear Programming (ILP). Other model types, such as Mixed Integer Linear Programming (MILP) and Linear Programming (LP), handle both discrete and continuous variables, while structured constraints can be represented using Constraint Programming (CP).

While a problem formulation provides an abstract formal description of a decision process, an *instance* (or problem instance) represents a specific case of the problem defined by concrete data. A *solution* to a problem instance is an assignment of values to all the decision variables defined by the model. As mentioned before, the solution quality is evaluated through an objective function, which provides one or more objective values, also referred to as scores or costs (typically numerical values). The *search space* of an instance encompasses all possible assignments of the variables (Section 2.3.1); the subset of solutions that satisfy all constraints is called the set of *feasible* solutions. A solution which violates at least one constraint is called *infeasible*. When no feasible solution exists for an instance, resulting in a logically inconsistent instantiation of the problem model, the model instance is also called infeasible.

[2] https://github.com/roar-net/problem-statements. Accessed 2025–09–22.

2.1.2 Solution Methods

The search space is explored using *solution methods*, which are algorithms or software solvers. Solution methods for COPs can be classified based on the completeness of their search (i.e., *complete* or *incomplete* methods) and how solutions are constructed (i.e., *perturbative* or *constructive methods*). Note that models and solution methods are strictly coupled together.

Complete methods exhaustively explore the search space, ensuring that *(i)* if a solution exists, the complete method will eventually find it; *(ii)* when the search terminates, the best solution found is guaranteed to be optimal. For these reasons, complete methods are also called *exact* methods. Examples of these methods include those derived from LP, MILP, and CP [419], for which numerous widely accepted software tools exist, including `CPLEX` and `CP Optimizer` [237], `Gurobi` [208], `OR-Tools` [388], `Gecode` [190], and the solvers included in the `MiniZinc` distribution [362] together with interfaces and APIs available in common programming languages. However, given that most COPs are NP-hard, exhaustive exploration can result in an exponential increase in runtime w.r.t. the size of the problem instance. Additionally, in many real-world applications, it is not necessary to certify the optimality of a solution. Instead, obtaining a good, feasible solution in a reasonable amount of time is sufficient. Incomplete methods address such necessities by exploring the search space non-exhaustively, often using stochastic approaches. When these methods are tailored to the problem, they are called *heuristics*; when they employ problem-independent strategies, they are referred to as *metaheuristics* [333]. Examples of metaheuristics include Tabu Search (TS) [200], Simulated Annealing (SA) [183, 256], Greedy Randomized Adaptive Search Procedure (GRASP) [276], Large Neighborhood Search (LNS) [431], Genetic Algorithm (GA) [221], Biased Random-Key Genetic Algorithm (BRKGA) [313], and Ant Colony Optimization (ACO) [159]. Differently from complete methods, widely accepted tools have not been available for incomplete methods, where researchers still rely on customized coding solutions [450, 451] and on a sprout of possible frameworks and libraries [383, 432] (see also Appendix B for more contextualization on the topic of metaheuristic softwares).

In constructive methods, a solution is created step by step, starting from an empty solution with all variables unassigned, and assigning them according to specific policies (e.g., ACO). Conversely, perturbative methods start from an existing complete solution and generate new solutions by modifying some variables. Examples include methods based on the concept of neighborhood (Section 2.3).

In this thesis we mainly employ incomplete, perturbative metaheuristics (Table 2.1), specifically we use local search based methods (Section 2.3), where a single incumbent solution is iteratively modified, and selection-based hybrid methods (Section 2.4), where a metaheuristic is combined with complementary components.

2.1.3 Other Aspects

Algorithmic design choices in solvers and algorithms can be addressed through *automated algorithm configuration*, as advocated by the Programming by Optimization (PbO) manifesto [222]. Such a paradigm calls for a shift of perspective in software development, where the design of components is approached through optimization. This involves a systematic exploration of design alternatives to select a configuration for the different components via

Table 2.1: Metaheuristic solution methods employed in this thesis.

Solution method	Abbreviation	Thesis reference
Hill Climbing	HC	Chapter 10
Tabu Search	TS	Chapter 5
Simulated Annealing	SA	Chapters 6, 10 and 15
Large Neighborhood Search	LNS	Chapter 10
Hyper-Heuristics	HHs	Chapter 6

automated tools like `Irace` [316]. When the algorithms themselves are treated as parameters within an algorithm portfolio (a set of algorithms exhibiting complementary performance) this gives rise to the notion of *automated algorithm selection*, the process of choosing the most suitable algorithm for a given instance. Further discussion on algorithm configuration is provided in Section 3.2, while algorithm selection is examined in Sections 2.6.5 and 3.3.

The entire optimization pipeline undergoes various types of *validation*. First, a practitioner must ensure that the models adhere as closely as possible to the real-world scenario (i.e., validation of specifications), even though some assumptions are necessary to generalize the concepts. Then, technical validation should be applied to check code correctness, ensure adherence to the model, and evaluate its performance efficiency. Furthermore, to efficiently solve a problem, evaluating and comparing different algorithms, solvers, and models is essential. This comparison is necessary because of the No Free Lunch (NFL) Theorem [498], which states that no single algorithm is best suited for all instances of an optimization problem. Therefore, tests are conducted to determine the most effective approach for the specific problem (or problem class), ensuring that the chosen solution method is robust and efficient. The latter type of validation is also addressed as a *benchmarking* process.

2.2 Examples of Problems

CO addresses both benchmark problems, like the seminal and widely studied Maximum Satisfiability Problem (max-SAT), and real-world applications, including employee scheduling [260, 518], machine scheduling [356], educational timetabling [83], and automotive production [497], among others [453].

In this section, we describe both the benchmark and the real-world problems investigated in this thesis. Table 2.2 indicates the chapters in which each is studied.

Table 2.2: Combinatorial Optimization Problems (COPs) employed in this thesis.

Problem	Abbreviation	Thesis reference
Bin Packing Problem	BPP	Chapter 18
Graph Coloring Problem	GCP	Chapter 18, in Chapter 6 as k-GCP
Jobshop Scheduling Problem	JSP	Chapter 18
Knapsack Problem	KP	Chapter 18
Permutation Flowshop Scheduling Problem	PFSP	Chapters 5, 6 and 18
Traveling Salesperson Problem	TSP	Chapter 6
Oven Scheduling Problem	OSP	Part III
Home Healthcare Routing and Scheduling Problem	HHCRSP	Part IV

The benchmark problems used for case studies throughout the thesis are the following.

Bin Packing Problem (BPP). The BPP considers a set of items, each with its own weight, and an infinite number of bins, all with the same capacity. The objective is to minimize the total number of bins used, subject to the constraints that every item is placed in a bin, and the sum of the weights of the items in each bin does not exceed the capacity.

Graph Coloring Problem (GCP). The GCP consists in assigning a color to each node of an undirected graph such that adjacent nodes receive different colors, with the objective of minimizing the number of colors used. The smallest number of colors required to obtain a valid coloring is known as the chromatic number. The k-Graph Coloring Problem (k-GCP) is a decision or optimization variant of this problem in which the number of available colors k is fixed in advance. The task is to assign one of the k colors to each node such that no two adjacent nodes share the same color. When a valid k-coloring does not exist, the problem can be relaxed into an optimization formulation, where the quality of a solution is evaluated by the number of coloring violations, that is, the number of edges connecting nodes assigned the same color (conflicting edges).

Jobshop Scheduling Problem (JSP). The JSP involves a finite set of jobs and machines. Each job consists of a sequence of operations that must be processed on the machines in a given order, with each job-machine pair associated with a predefined processing time. The objective is to construct a feasible schedule that minimizes the makespan. A valid schedule must satisfy three main constraints. First, the prescribed order of operations within each job must be respected. Second, machines cannot process more than one operation at the same time, meaning that no overlaps are allowed. Third, operations are non-preemptive: once an operation has started on a machine, it must run to completion without being interrupted or paused.

Knapsack Problem (KP). The KP considers a finite set of items, each with a profit and a weight, and a knapsack capacity. The goal is to choose items to be included in the knapsack that maximize total profit subject to the capacity constraint. Variants of KP include the *0–1* KP, where each item can be selected at most once, the *bounded* KP, where each item can be selected at most a given number of times, and the *unbounded* KP, where there is no upper limit on the number of times each item can be selected.

Permutation Flowshop Scheduling Problem (PFSP). The PFSP consists of scheduling a set of jobs on a series of machines, where each job is defined as a sequence of tasks that must be processed in the same order on all machines. Each task has a predefined processing time. In this thesis, we focus on minimizing the *makespan*, i.e., the completion time of the last job on the final machine. In the literature, several variants of the PFSP have been proposed, addressing alternative objectives such as *tardiness* and the *flow time* [30].

Traveling Salesperson Problem (TSP). The TSP considers a list of cities and the distances between each pair of cities. The task is to find the shortest possible route that visits each city exactly once and returns to the start city (also known as *Hamiltonian cycle of minimum length*).

The real-world problems used throughout the thesis are the following.

Oven Scheduling Problem (OSP). The OSP consists of grouping components into thermally compatible batches and scheduling them on parallel ovens, subject to capacity and compatibility constraints. The objective is to minimize a weighted combination of criteria, namely the total processing time, the tardiness of jobs w.r.t. their due date, and the setup costs associated with adjusting oven temperatures.

Home Healthcare Routing and Scheduling Problem (HHCRSP). The HHCRSP consists in the assignment and routing of caregivers to patients over a planning horizon, subject to practical constraints, including skill matching, time windows, service durations, travel times, working-time limits, breaks, continuity of care, and synchronization requirements. The objective is to minimize a weighted combination of criteria that balance patient satisfaction, fairness in the distribution of working hours, and compliance with organizational policies.

2.3 Local Search Metaheuristics

Local search is not a single algorithm, but a *paradigm* on which many metaheuristics are built. It operates over a neighborhood relation defined on a search space and uses policies (e.g., exploration policy, selection policy) to traverse that space.

In this section, we first describe the key concepts of local search (Section 2.3.1) and the policies (Section 2.3.2). Second, we provide a complete overview on the local search procedure (Section 2.3.3). Then, we describe the local search methods we employ in this thesis (Sections 2.3.4 to 2.3.6).

2.3.1 Key Concepts

In an optimization problem, one is given a set of feasible solutions $\mathcal{S}_f$ and an objective function $obj : \mathcal{S}_f \to \mathbb{R}$ that quantifies the quality of each solution. In a minimization problem, the task is to find a solution $s^* \in \mathcal{S}_f$ such that $obj(s^*) \leq obj(s)$ for all $s \in \mathcal{S}_f$; in a maximization problem, the inequality is reversed. Each problem is associated with a set of existing instances $\mathcal{I}$. When applying local search to a given instance $x \in \mathcal{I}$ of the problem, the central elements are the search space $\mathcal{S}$, the cost function F, and the neighborhood relation $\mathcal{N}$.

Search space

The search space $\mathcal{S}$ is the set of admissible *states* explored by the algorithm. Each state encodes a candidate solution of instance x via some representation. We distinguish $\mathcal{S}$ from the feasible subset $\mathcal{S}_f \subseteq \mathcal{S}$ as many implementations allow $\mathcal{S}$ to include infeasible states to enhance diversification, handling them via penalties (in the cost function F) or repairs. On the contrary, $\mathcal{S}$ might also be restricted, that is it may contain a subset of all the possible solutions to the problem due to some additional constraints one might impose to reduce the number of states that an algorithm might visit, for instance in the case of symmetry breakings. In a COP, at least one optimal solution of x must be represented in $\mathcal{S}$. Together with the cost function F and a neighborhood operator N, $\mathcal{S}$ induces a fitness landscape whose properties (e.g., ruggedness, neutrality, plateaus) shape search difficulty.

A solver for a given problem specifies the search space via a state representation $s \in \mathcal{S}$. For efficiency, s is accompanied by auxiliary (intentionally redundant) structures (such as caches, indices, and partial cost aggregates) maintained under moves to support constant- or logarithmic-time updates and incremental evaluation of F. These structures do not change the problem semantics or the search space representation; on the contrary, they are bookkeeping devices kept consistent throughout the search.

Cost function

The cost function $F : \mathcal{S} \to \mathbb{R}$: assigns a scalar quality measure to each state $s \in \mathcal{S}$. The cost function of a state s is indicated as $F(s)$. In the simplest case, the cost function equals the problem objective, so that

$$F(s) = obj(s).$$

When the search space includes infeasible states or when we wish to bias the search (e.g., towards balanced solutions), we use an augmented cost

$$F(s) = obj(s) + \text{[regularization terms]} \times \text{[regularization weights]},$$

where weights are set so that, eventually, feasible solutions are preferred over infeasible ones (e.g., via large fixed or adaptive penalties).

Neighborhood relation

A neighborhood relation (or *neighborhood*) $\mathcal{N}(s)$ of state $s \in \mathcal{S}$ defines the set of states reachable by applying one admissible move m to s. Each $s' \in \mathcal{N}(s)$ is a *neighbor*. A move m applied at s maps s to s', which we write as $s \oplus m$. When evaluating $s' \in \mathcal{N}(s)$ we use the *delta cost*, which is the increment in the solution cost due to the application of the move:

$$\Delta F(s', s) \;=\; F(s') - F(s) \;=\; \Delta F_m(s) \text{ with } s' \;=\; s \oplus m.$$

(Assuming minimization in the problem modeling) a move is *improving* if $\Delta F_m(s) < 0$, *worsening* if $\Delta F_m(s) > 0$, and *neutral/sideways* if $\Delta F_m(s) = 0$. To avoid unnecessary recomputation, $F(s')$ is usually not evaluated from scratch; instead, we adopt *incremental* evaluation, computing $\Delta F_m(s)$ from the local effect of m on s.

Usually, a solver to a specific problem includes the possibilities of more than one neighborhood, i.e., a *multi-neighborhood*. A multi-neighborhood local search maintains a portfolio $\{\mathcal{N}_i\}_{i=1}^{k}$ of neighborhood operators and, at each iteration, explores one or more of them (e.g., by cycling through them or by random or adaptive selection), so that the actual neighborhood is $\mathcal{N}(s) = \bigcup_{i=1}^{k} \mathcal{N}_i(s)$. Such mechanism has shown to be effective for several problems in the literature [66, 416, 417] and is used in this thesis (e.g., in Chapter 15).

Local minima

Given a state $s \in \mathcal{S}$ and a neighborhood N, we say that s is a *(weak) local minimum* (for minimization) if

$$F(s) \;\leq\; F(s') \quad \forall\, s' \in \mathcal{N}(s).$$

It is a *strict local minimum* if

$$F(s) \;<\; F(s') \quad \forall\, s' \in \mathcal{N}(s).$$

If equality holds for some neighbors, s lies on a *plateau*. Landscape notions such as local minima and plateau are *relative to* $\mathcal{N}$: changing the neighborhood (i.e., the move set) changes which states are locally optimal. For this reason, the usage of multi-neighborhood is relevant. A local minimum is also referred to as *local optimum*.

Figure 2.2 reports a visual representation of the fitness landscape; in this case, the strict local minimum also corresponds to the global minimum.

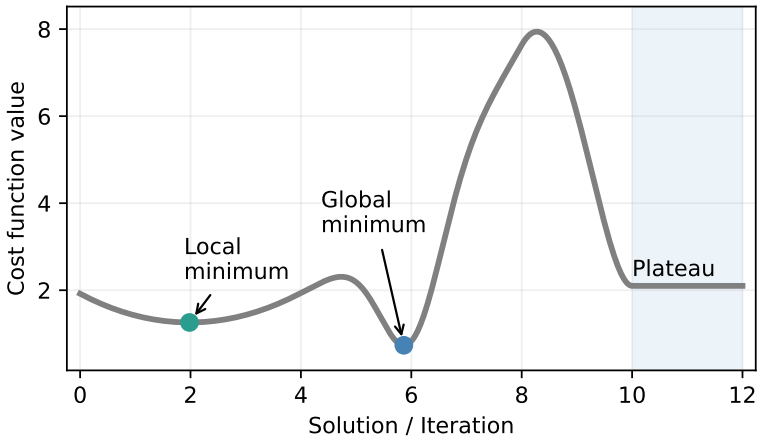


Figure 2.2: Visual representation of the fitness landscape.

2.3.2 Policies

We now formalize the control policies that coordinate the core components and steer traversal of the search space. Specifically, we consider the initialization policy, the exploration policy, the selection criterion, the acceptance criterion, and the termination criterion. Depending on the algorithm, these policies expose static or dynamic control parameters that must be specified and, where appropriate, tuned or adapted online.

Initialization policy

Because local search is inherently perturbative, it requires an initial solution $s_0 \in \mathcal{S}$ from which the search process begins. Two broad initialization regimes are commonly adopted, i.e., cold and warm starts (Table 2.3).

In a *cold start*, the initial solution is generated at random, possibly followed by a repair step to ensure feasibility. This strategy favors diversification, as it exposes the algorithm to different regions of the search space, though it often yields solutions of modest initial quality. Conversely, a *warm start* relies on an auxiliary optimization method, such as a greedy or constructive heuristic, to produce s_0. While this approach can accelerate early progress by starting from a high-quality solution, it also carries a greater risk of premature

Table 2.3: Comparison of cold and warm start initialization in local search metaheuristics.

	Cold start	Warm start
Generation source	Random generator (possibly repaired)	Auxiliary optimization method
Solution quality	Typically modest	Typically high
Search bias	Promotes diversification	Promotes intensification
Usage in this thesis	Parts II to IV	Part III

convergence. Both regimes are considered in this thesis, depending on the problem domain and experimental setting.

Exploration policy

Given the current state s, the *exploration policy* specifies which neighbors are generated and inspected before any move decision is taken. Let $\mathcal{N}(s)$ denote the full neighborhood of s and let $C(s) \subseteq \mathcal{N}(s)$ be the candidate set passed to the move-selection mechanism. Several strategies exist for constructing $C(s)$ (Table 2.4). In *full enumeration*, the entire neighborhood is evaluated, i.e., $C(s) = \mathcal{N}(s)$. Alternatively, a *candidate* or *restricted list* may be built by filtering $\mathcal{N}(s)$ according to specific criteria, such as aspiration conditions, tabu definitions, or move size restrictions. A further possibility is *sampling*, where a subset of neighbors is drawn at random to approximate the full neighborhood while reducing evaluation costs.

Table 2.4: Comparison of exploration policies for constructing the candidate set $C(s)$ in local search.

Policy	Definition	Advantages	Drawbacks
Full enumeration	$C(s) = \mathcal{N}(s)$, all neighbors are generated and evaluated.	Provides complete information; guarantees best move.	Computationally expensive.
Restricted candidate list	$C(s) \subseteq \mathcal{N}(s)$ obtained by filtering with rules.	Reduces evaluation cost; encodes problem knowledge or memory.	May discard high-quality moves; quality depends on filter design.
Sampling	$C(s) \subseteq \mathcal{N}(s)$ obtained by random sampling of neighbors.	Scalable; preserves stochastic exploration.	May miss high-quality moves; performance varies by sample size.

Selection policy

Selection policies govern how the next move to evaluate is chosen from the candidate set produced during exploration. It is convenient to reason in terms of moves rather than states. Let $\mathcal{M}(s)$ denote the set of admissible moves at state s, and let $C_M(s) \subseteq \mathcal{M}(s)$ be the subset of moves generated during the exploration stage. The corresponding candidate neighbors are then given by $s \oplus m : m \in C_M(s), \subseteq C(s)$. Several strategies for selecting a move are commonly used (Table 2.5). In the *best (steepest) improvement* strategy, the algorithm chooses

$$m^\star \in \arg \min_{m \in C_M(s)} \Delta F_m(s)$$

that is, the move yielding the greatest cost reduction. This requires evaluating all candidates in $C_M(s)$ and thus provides the strongest local progress at the expense of evaluation effort.

The *first improvement* strategy instead scans $C_M(s)$ in random order and accepts the first move satisfying $\Delta F_m(s) < 0$. This reduces the number of evaluations on average, but the outcome depends on the scanning order and in the worst case may still be as costly as steepest descent. Finally, in a *randomized choice*, the next move is sampled from a probability distribution over $C_M(s)$ (e.g., uniform), which promotes exploration but provides no guarantee of improvement.

In practice, neighborhood exploration and move selection are tightly coupled: candidates are generated and evaluated, and the next move is chosen within the same routine.

Table 2.5: Comparison of common selection policies in local search. Given a state s, let $\mathcal{M}(s)$ be the set of admissible moves, $C_M(s) \subseteq \mathcal{M}(s)$ the candidate set after exploration, and $\Delta F_m(s) = F(s \oplus m) - F(s)$ the cost difference induced by move m.

Policy	Rule	Advantages	Drawbacks
Best (steepest) improvement	Choose the move with greatest cost reduction in $C_M(s)$.	Strong local progress; deterministic.	Requires evaluating all candidates; expensive for large neighborhoods.
First improvement	Scan $C_M(s)$ in random order, return the first m with $\Delta F_m(s) < 0$.	Fewer evaluations on average; stochasticity improves diversification.	Outcome depends on scanning order; worst case as costly as best improvement.
Randomized choice	Sample a move from a probability distribution on $C_M(s)$.	Minimal evaluation cost; maximizes diversification.	No guarantee of improvement; progress may be erratic.

Acceptance policy

After a move m has been selected, the *acceptance policy* determines whether the search state is updated to $s \oplus m$ or kept as s. For minimization problems, let $\Delta F_m(s) = F(s \oplus m) - F(s)$ denote the cost difference induced by applying m. In the *improving-only* acceptance rule, a move is accepted if and only if it strictly improves the cost ($\Delta F_m(s) < 0$), while the *non-worsening* variant also allows equal-cost moves ($\Delta F_m(s) \leq 0$), thereby permitting plateaus to be traversed. More advanced schemes admit non-improving moves under controlled conditions. *Probabilistic acceptance*, as in SA, always accepts improving moves and accepts worsening moves with a probability that based on $\Delta F_m(s)$. Similarly, *threshold accepting* admits moves satisfying $\Delta F_m(s) \leq \tau$, where the threshold τ is gradually decreased over time. At the opposite extreme, an *unconditional policy* accepts every move, producing a random walk through the search space. Table 2.6 offers an overview of the acceptance policies.

Termination policy

A *termination policy* (or *stopping predicate*) specifies the condition under which the search process halts. Several types of criteria are commonly employed (Table 2.7). *Resource-based predicates* stop the search when a predefined budget is exhausted, either in terms of wall-clock/CPU time or in terms of iterations and objective evaluations. *Progress-based predicates* monitor the improvement of the current best solution, terminating after a period of stagnation or when a predefined target value (or gap to a known bound) is reached. Finally, *algorithm-specific predicates* exploit internal control parameters, such as the cooling schedule in SA, where the search stops once the temperature is below a given value.

Table 2.6: Comparison of acceptance policies in local search. We assume a minimization problem and $\Delta F_m(s) = F(s \oplus m) - F(s)$ denotes the cost difference of applying move m to solution s.

Policy	Rule	Advantages	Drawbacks
Improving-only	Accept if $\Delta F_m(s) < 0$.	Guarantees strict improvement; simple to implement.	Can stall quickly in local optima.
Non-worsening	Accept if $\Delta F_m(s) \leq 0$.	Allows plateau exploration; still monotone.	Cannot escape strict local minima.
Probabilistic acceptance	Always accept if $\Delta \leq 0$, otherwise with probability $p(\Delta)$.	Enables escape from local minima.	Requires parameter tuning.
Threshold accepting	Accept if $\Delta F_m(s) \leq \tau$, with $\tau \downarrow 0$ over time.	Enables escape from local minima.	Sensitive to threshold schedule.
Unconditional (random walk)	Always accept.	Maximizes exploration.	No guarantee of improvement; highly inefficient alone.

In practice, termination is rarely governed by a single rule; rather, multiple criteria are typically combined with a logical disjunction (e.g., time limit `OR` stagnation).

Table 2.7: Comparison of common stopping predicates in local search.

Predicate type	Definition	Advantages	Drawbacks
Time budget	Halt when wall-clock or CPU time reaches a preset limit.	Aligns with practical runtime constraints.	Depends heavily on hardware and load conditions.
Iteration budget, evaluation budget	Halt after a maximum number of iterations or evaluations.	Independent of hardware; provides predictable effort.	Does not adapt to instance difficulty or progress.
Stagnation	Halt if no improvement occurs for a given number of iterations.	Adaptive to search dynamics; avoids wasted effort.	Sensitive to parameter choice; may terminate prematurely.
Target value, bounded gap	Halt once a target objective or gap to a bound is reached.	Ensures guaranteed quality if bounds are valid.	Requires knowledge of target or bound; may be impractical in real-world settings.
Algorithm-specific	Criterion derived from internal parameters.	Natural to implement; respects algorithm design.	Not comparable across algorithms; may not reflect actual progress; may require tuning.

2.3.3 Local Search Procedure

As anticipated, local search algorithms are metaheuristics: they prescribe a general search scheme rather than a problem-specific one. A given instantiation specifies how the problem-independent components orchestrates the problem-dependant ones (Table 2.8). The first ones include the local search policies, such as the strategy for exploring the neighborhood, the rule for selecting a move, and the stopping criterion. In contrast, the problem-dependent components concern the concrete design of the search space, including the solution representation, the neighborhood relations, and the cost function with its incremental evaluation, as well as any domain-specific control parameters. This distinction is crucial: the independent components provide a reusable framework applicable across different problems, whereas the dependent ones require tailored engineering to capture the structure and constraints of the specific domain/problem under study. This philosophy underlies EasyLocal++ (Appendix B) and is further developed within the

ROAR-NET initiative.[3] Practically, to solve a new problem with a given algorithm it suffices to implement the problem-dependent pieces (solution representation, neighborhoods, incremental evaluation, etc.); the generic control mechanism (move selection, acceptance, termination, etc.) is provided by the algorithm/framework itself.

Table 2.8: Distinction between problem-independent and problem-dependent components in local search.

Problem-independent components	Problem-dependent Components
Exploration policy	State representation / search space
Selection policy	Definition of neighborhood relations
Acceptance policy	Cost function and incremental evaluation
Termination policy	Control parameters

Algorithm 1 presents the pseudocode of an abstract local search procedure. Inputs of the procedure include the problem instance; a specification of the search space $\mathcal{S}$ (i.e., the solution representation);[4] a neighborhood relation $\mathcal{N}$; and a cost function F. An initial solution s_0 is also assumed, given that local search is a perturbative methodology. Starting from $s_0 \in \mathcal{S}$, the algorithm iteratively explores the neighborhood $\mathcal{N}(s)$ of the current solution $s \in \mathcal{S}$, selects a move according to the selection policy, and applies it subject to the acceptance criterion, repeating this loop until the termination condition is met.

Algorithm 1: Pseudo-code for a general local search procedure.

Input: Problem instance, search space $\mathcal{S}$, neighborhood relation $\mathcal{N}$, cost function F, initial solution s_0.
Output: Best solution found s^*

```
// Initialize current solution s and best solution found s*
s ← s_0
s* ← s
// Proceed until the termination criterion is satisfied
while TerminationCriterion() is not met do
    // Expore the neighborhood and select the move
    m ← SelectionCriterion (ExplorationCriterion (N(s)))
    // Decide if the current solution can be modified
    if AcceptanceCriterion (s, m) is met then
        // Apply the move
        s ← s ⊕ m
        // Update the best solution if necessary
        if F(s) < F(s*) then
            s* ← s
return s*
```

[3] https://github.com/roar-net/roar-net-api-spec. Accessed 2025–09–22.

[4] Note that here and in what follows, we use *state* and *solution* interchangeably.

Algorithm 2: Pseudo-code for a multi-neighborhood Randomized Hill Climbing (RHC). In the remainder of the thesis, such implementation is referred to as Hill Climbing (HC).

Input: Problem instance, search space $\mathcal{S}$, multi-neighborhood relation $\mathcal{N} = \bigcup_{j=1}^{k} \mathcal{N}_j$, with related probability biases $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_k\}$, cost function F, total number of iterations $i_{\max}$, initial solution s_0.
Output: Best solution found s^*

```
// Initialize current solution s and best solution found s*
s ← s_0
s* ← s
// Initialize the iteration counter i
i = 0
// Proceed until the total iteration budget i_max is exhausted
while i < i_max do
    // Randomly select one neighborhood among the k possible considering σ
    N_j ← RandomNeighborhood(N, σ)
    // Randomly select one move in the selected neighborhood
    m ← RandomMove(N_j, s)
    // If the move is improving or neutral, then accept it
    if F(s ⊕ m) − F(s) ≤ 0 then
        s ← s ⊕ m
        // Update the best if necessary
        if F(s) < F(s*) then
            s* ← s
    // Update the iteration counter
    i ← i + 1
return s*
```

2.3.4 Hill Climbing

Hill Climbing (HC), also known as *Iterative Descent*, is the simplest local search scheme.[5] As with many metaheuristics, the label encompasses multiple exploration/selection policies:

- Best Improvement (BI), also known as *steepest descent*, which is about choosing m with minimal $\Delta F_m(s)$ over $\mathcal{N}(s)$;
- First Improvement (FI), which is about applying the first move encountered with $\Delta F_m(s) < 0$ under a prescribed scan order;
- Randomized Hill Climbing (RHC), which is about randomly selecting the move m.

In all cases, improving moves are always accepted (i.e., *descent* techniques); furthermore, RHC also allows the acceptance of neutral (sideways) moves to traverse plateaus (i.e., *non-ascent* technique).

Regarding termination, any criterion can be used. BI and FI terminate naturally at stagnation (i.e., at a local optimum) when no improving move exists in $\mathcal{N}(s)$ given their respective scans. By contrast, RHC samples the neighborhood; thus, certifying the absence of improving moves would require a full scan, defeating its purpose. Consequently, RHC is typically terminated by an external budget (e.g., total iterations) or a stagnation counter (e.g., a maximum number of consecutive non-improving/neutral steps).

[5] The name Hill Climbing (HC) is tied to maximization; however, throughout this thesis, we adopt the minimization convention, so *hill climbing* corresponds to descent in F.

In this thesis, HC is used in its randomized form together with a multi-neighborhood scheme (Chapter 10); for brevity, we simply refer to this implementation as HC. Algorithm 2 provides the pseudocode, using a total iteration budget $i_{\max}$ as the termination criterion. The inputs mirror those of the generic local search procedure (Algorithm 1), with the addition of a neighborhood portfolio $\mathcal{N} = \bigcup_{j=1}^{k} \mathcal{N}_j$ and associated probability biases $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_k\}$, $\sum_{j=1}^{k} \sigma_j = 1$.

Starting from an initial state s_0, we set the current solution and the best solution found so far to s_0, and initialize the iteration counter i to 0. While the iteration counter is below the threshold $i_{\max}$, we sample a neighborhood j based on $\boldsymbol{\sigma}$, and then pick a move $m \in \mathcal{N}_j(s)$ uniformly at random. If the move is neutral or improving, we accept it and thus update s; we then update the incumbent s^* whenever $F(s) < F(s^*)$, and increment i.

As discussed, the RHC used here is a non-ascent variant, since it also accepts neutral moves. This is beneficial on plateaus: neutral steps allow traversal of *flat* regions and may reach a state where an improving move becomes available. However, there are downsides: prolonged neutral wandering can inflate the time-to-improvement, and the search may cycle within a small set of neutral states. Alternative designs in the literature explicitly address these issues: TS uses memory to forbid recently visited moves/attributes and thus discourage revisits (Section 2.3.5), while SA accepts worsening moves with controlled probability, so to escape from strict local minima and plateaus (Section 2.3.6).

2.3.5 Tabu Search

Tabu Search (TS) was introduced by Glover and Laguna [200]. We now outline its main components; an in-depth characterization of TS is deferred to Chapter 5.

TS defining feature is an explicit memory structure (called *tabu list*) that forbids recently executed moves, move attributes, or even whole states [126] for a fixed or adaptive *tenure*, thereby discouraging cycling.[6] This prohibition can be overridden by an *aspiration criterion*, typically when a tabu move yields a solution better than the current incumbent.

The procedure starts from an initial solution and iterates until a termination criterion is met (typically a total iteration budget or stagnation). At each iteration, the neighborhood is explored according to a prescribed scan, with candidate moves filtered by a tabu mechanism $\mathbf{T}$ and considering the aspiration criterion. After selection among the admissible (non-tabu or aspired) candidates, TS applies the chosen move even if it worsens the current solution.

2.3.6 Simulated Annealing

Simulated Annealing (SA) is a well-established metaheuristic inspired by the annealing of metals [256], with applications in timetabling [285], scheduling [243], and routing [516].

SA defining feature is the *probabilistic acceptance of worsening moves*, meaning that worsening moves can occasionally be accepted based on a parameter called *temperature*. Usually, the temperature follows a monotone nonincreasing schedule from an initial T_0 to a final T_f: high T yields frequent acceptance of worsening moves (broad exploration), while low T enforces near-greedy behavior (intensification). Neighborhood exploration and move

[6] Note that what is declared tabu depends on the state representation and on the attributes exposed by the neighborhood (moves, move features, occasionally states or objective levels), thus it is problem- and implementation-dependant.

selection are commonly random, though alternative scans exist; likewise, many cooling schedules and acceptance criteria have been studied, e.g., geometric, linear, logarithmic, with reheating [183].

In this thesis, we adopt a classical SA in the spirit of Kirkpatrick, Gelatt, and Vecchi [256], augmented with a multi-neighborhood portfolio $\mathcal{N} = \bigcup_{j=1}^{k} \mathcal{N}_j$, the Metropolis acceptance rule, a cut-off mechanism to accelerate the early high-temperature phase (by limiting evaluations per temperature level/iteration), and a termination criterion based on the total-iteration budget $i_{\max}$.

The pseudocode is given in Algorithm 3. The temperature T starts from an initial value T_0 and decreases up to a final value T_f. The move selection occurs in two stages, as in the case of HC (Algorithm 2): first drawing the neighborhood among the k available neighborhoods according to the probability biases $\boldsymbol{\sigma}$ and then selecting a move from this neighborhood. Let $\Delta F_m(s) = F(s \oplus m) - F(s)$, under the Metropolis rule, a candidate move m is accepted if $\Delta F_m(s) \leq 0$, and otherwise with probability $\exp(-\Delta F_m(s)/T)$.

The temperature decreases according to a *geometric cooling schedule*, either after a fixed number of iterations (N_s) or after a fixed number of accepted moves ($N_a < N_s$), the latter being known as a *cut-off mechanism*. In a geometric scheme, the temperature is multiplied by a factor α at each decrease, where $0 < \alpha < 1$ denotes the *cooling rate*. The cut-off mechanism anticipates the decrease before reaching N_s whenever a threshold number of moves N_a is accepted, with $N_a < N_s$. The rationale is that if many moves are accepted, the temperature is likely too high, so reducing it earlier helps focus the search. To maintain the overall length of the schedule, the iterations saved at the current temperature are redistributed among the subsequent ones by increasing N_s accordingly. We define N_a as a proportion of N_s so that $N_a = \rho \cdot N_s$, with $0 < \rho \leq 1$. Based on $i_{\max}$, the parameter N_s is calculated as:

$$N_s = \frac{i_{\max}}{\log_\alpha \left(\frac{T_f}{T_0}\right)}$$

This approach simplifies the determination of the algorithm's duration and makes it independent of other parameters.

Among the most critical SA control parameters are the initial and final temperatures and the associated cooling scheme [183], which dictate how the temperature decreases over time. While this cooling approach can be advantageous when local minima are located near the starting point, it may hinder the ability to escape from deeper local minima encountered at lower temperatures later in the search. The literature has addressed this problem with restarting and reheating schemes [201], which temporarily increase the temperature when the search stagnates, allowing the algorithm to regain the ability to escape local optima.

Reheating is typically triggered by stagnation indicators, such as a fixed number of iterations without improvement or the attainment of a predefined lower temperature (T_f). Once activated, the temperature may be increased either by resetting it to the initial value or by applying a partial increment relative to the current state. Although reheating improves diversification and facilitates the exploration of previously inaccessible regions of the search space, it also introduces additional control parameters. These include the reheating threshold (T_r), the number of iterations allotted per reheated phase, or alternatively, the total number of reheats to perform ($R_{\max}$) within a fixed computational budget.

Algorithm 3: Pseudo-code for a multi-neighborhood Simulated Annealing (SA) with cut-off mechanism.

Input: Problem instance, search space $\mathcal{S}$, multi-neighborhood relation $\mathcal{N} = \bigcup_{j=1}^{k} \mathcal{N}_j$, with probability biases $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_k\}$, cost function F, total number of iterations $i_{\max}$, initial solution s_0, initial temperature T_0, final temperature T_f, cooling rate α, neighborhood accepted ratio ρ.
Output: Best solution found s^*

```
// Initialize current solution s and best solution found s*
s ← s_0
s* ← s
// Initialize iteration counters
i ← 0
N_s ← i_max / log_α(T_f / T_0)
N_a ← ρ · N_s
// Initialize temperature
T ← T_0
// Proceed until the total iteration budget i_max is exhausted
while i < i_max do
    // Initialize per-temperature iteration counters
    n_s ← 0
    n_a ← 0
    // Proceed until the number of iterations per temperature level or the upper bound to
     the accepted moves per temperature is not exhausted
    while n_s < N_s and n_a < N_a do
        // Randomly select one neighborhood among the k possible considering σ
        N_j ← RandomNeighborhood(N, σ)
        // Randomly select one move in the selected neighborhood
        m ← RandomMove(N_j, s)
        ΔF_m(s) ← F(s ⊕ m) − F(s)
        // Metropolis acceptance criterion
        if ΔF_m(s) ≤ 0 or RandomReal(0, 1) < exp(−ΔF_m(s)/T) then
            s ← s ⊕ m
            // Increment the counter related to the number of accepted moves
            n_a ← n_a + 1
            // Update the best if necessary
            if F(s) < F(s*) then
                s* ← s
            // Increment the counter related to the iterations per temperature
            n_s ← n_s + 1
    // Decrease the temperature based on the cooling rate α
    T ← α · T
    // Redistribute the iteration spared at this temperature level
    N_s ← N_s + (N_s − n_s) / log_α(T_f / T_0)
return s*
```

Some studies have questioned the necessity of temperature decrease, proposing Fixed Temperature Simulated Annealing (FTSA) as an alternative [113, 348]. That is, the temperature is kept constant through the search. In this context, Franzin and Stützle [182] observed that SA performs effectively when the neighborhood structures differ across various regions of the search space, while a FTSA is more suitable when the landscape is uniform. This highlights the challenge of selecting such parameters, as they depend on the

characteristics of the search landscape, which are unknown in advance.

2.4 Hybrid Metaheuristics

In this section, we first introduce LNS (Section 2.4.1) and then Hyper-Heuristics (HHs) (Section 2.4.2). Both are metaheuristics in that they describe general frameworks that can be customized to the problem at hand. In this classification, we treat them as hybrid methods, since they may incorporate other algorithms within their operations, such as destroyers or repairers in LNS and Low Level Heuristics (LLHs) in HHs.

2.4.1 Large Neighborhood Search

Large Neighborhood Search (LNS) was originally proposed by Shaw [431]. It may be viewed as a local search algorithm in which neighborhood operators act on a substantially larger part of the current solution than in traditional settings. The method alternates two phases: given a solution, LNS deliberately removes part of it (*destroy* phase) and then reconstructs it (*repair* phase).

In practice, the destroy/repair subroutines may be heuristic or exact. In this thesis, we use heuristic rules to select which portion of the solution to destroy, and exact methods to perform the repair. Typically, one defines a portfolio of destroy and repair operators, and a selection mechanism governed by probability biases (similarly to what we observed in local search with the multi-neighborhood selection). We set these biases via offline tuning, however, one might update them adaptively [7, 9, 16]. The evidence on the benefit of their dynamic setting remains mixed [470].

Algorithm 4 summarizes the LNS procedure. The inputs are: a problem instance; a set of destroy operators $\mathcal{D}$ with probability biases $\boldsymbol{\sigma}_d$; a set of repair operators $\mathcal{R}$ with probability biases $\boldsymbol{\sigma}_r$; a cost function F; and an initial solution s_0. Starting from s_0, the algorithm repeats until a termination criterion is met (in the case of this thesis, a wall-clock time limit). At each iteration, a destroy operator $d \in \mathcal{D}$ is sampled according to $\boldsymbol{\sigma}_d$ and a repair operator $r \in \mathcal{R}$ according to $\boldsymbol{\sigma}_r$; the candidate solution is $s' \leftarrow \texttt{DestroyAndRepair}(s, d, r)$, which is then evaluated through F. If the acceptance rule holds (in the case of our thesis: strictly improving or equal-cost), we set s' as the new current solution and, if necessary, we also update the best solution s^*.

2.4.2 Hyper-Heuristics

Hyper-Heuristics (HHs) are a category of high-level, problem-independent heuristic framework designed to operate on a portfolio of LLHs [70, 158, 160]; in other words, they are problem-independent heuristics that control problem-dependent heuristics. They have been applied to a variety of problems, including scheduling [345], vehicle routing [373], and timetabling [71].

Figure 2.3 shows the general concept of HHs. HHs explore the search space by iteratively selecting and applying LLHs to one or more solutions, while their only knowledge is the set of available LLHs, the execution time, and the changes in the objective function. This strict separation is called the *domain barrier* and enables high adaptability and flexibility across various problem domains. Most HHs incorporate adaptive mechanisms that dynamically

Algorithm 4: Pseudo-code for Large Neighborhood Search (LNS).

Input: Problem instance, search space $\mathcal{S}$, set of destroy operators $\mathcal{D}$, probability biases for the destroy operators $\boldsymbol{\sigma_d}$, set of repair operators $\mathcal{R}$, probability biases for the repair operators $\boldsymbol{\sigma_r}$, cost function F, initial solution s_0.
Output: Best solution found s^*

```
// Initialize current solution s and best solution found
s ← s_0
s* ← s
// Proceed until the termination criterion is satisfied
while TerminationCriterion() is not met do
    // Select a destroy operator d ∈ D based on σ_d
    d ← SelectDestroyOperator(s, D, σ_d)
    // Select a repair operator r ∈ R based on σ_r
    r ← SelectRepairOperator(s, R, σ_r)
    // Use d and r to destroy and repair the current solution s
    s' ← DestroyAndRepair(s, d, r)
    // Decide if the current solution can be modified
    if AcceptanceCriterion (s, s') is met then
        s ← s'
        // Update the best solution if needed
        if F(s) < F(s*) then
            s* ← s
return s*
```

adjust their behavior based on the limited knowledge about the current state of the search to effectively balance exploration and exploitation of the search space.

This thesis focuses on *selection* HHs [67], where the LLHs are provided as a predefined set of operators to be chosen and applied during the search process. An alternative approach would involve *generative* HHs, which aim to create new LLHs by combining basic algorithmic components. In the case of selection HHs, LLHs can be of the following type:

- **Mutation (MU)**: It applies a random change to the candidate solution, disregarding the objective value of the resulting solution.
- **Local Search (LS)**: It applies local changes to the candidate solution that improve the objective value. The resulting solution is guaranteed to be better or of the same quality as the initial one.
- **Ruin-and-recreate**: It destroys and repairs part of the solution in a LNS fashion.
- **Crossover**: It combines two solutions

In this thesis, we use as LLHs single temperature steps of SA, therefore they are inherently of LS and MU type.

HHs gained significant momentum with the Cross Domain Heuristic Search Challenge 2011 (CHeSC 2011) organized by Burke et al. [69]. This competition required participants to design HHs capable of achieving high performance across six benchmark problems. All competing HHs were developed using `HyFlex`, a software framework that separates the general HH strategies from problem-specific details [369]. The competition was won by Generic Intelligent Hyper-Heuristic (GIHH) [347], which is still competitive on several

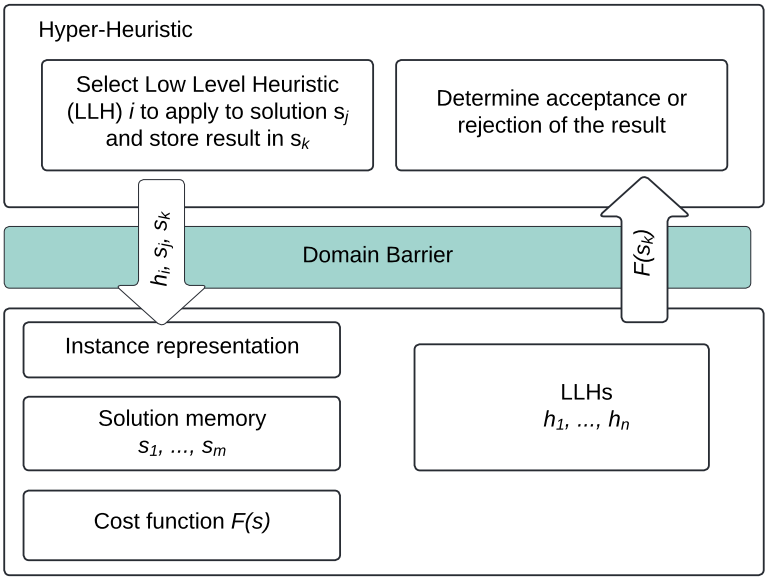


Figure 2.3: General scheme of Hyper-Heuristic (HH).

domains. A streamlined version of the winner, called Lean Generic Intelligent Hyper-Heuristic (L-GIHH), was introduced afterward [4]. Over the years, many HHs have been proposed, and successful recent entries are frequently based on Iterated Local Search (ILS) [3, 5, 6], or on Reinforcement Learning (RL) [259, 345].

2.5 Large Language Models

LLMs have fundamentally transformed the landscape of Natural Language Processing (NLP). At the core of this transformation is the Transformer architecture introduced by Vaswani et al. [476], which introduced an improved self-attention mechanism built on top of the one proposed by Bahdanau, Cho, and Bengio [33]. This mechanism allows models to weigh the importance of different words in a sentence, irrespective of their distance, leading to more contextually aware representations.

The Transformer architecture operates with an arrangement of two specified modules, an encoder and a decoder, designed to transform the input data into more abstract and general-purpose representations. Each encoder and decoder in the architecture is built from layers that perform specific functions. First, positional encoding is introduced to inject information about the order of input words into the model, which is crucial for processing sequences where the arrangement of elements carries meaning, as is common in NLP. Following positional encoding, the embedding layer converts the input tokens (i.e., individual pieces of text, typically words or sub-words) into vectors of continuous numbers. This transformation turns linguistic information into a numerical form that neural networks can process. Once the input has been encoded and embedded, it passes through the Transformer's core mechanisms, i.e., the attention layers. The encoder relies on a self-attention mechanism to independently assess and emphasize different parts of the input data. This allows the model to understand each input segment in relation to the rest of the input, thus enhancing the context awareness of the system. In parallel,

the decoder also employs a self-attention layer but focuses on generating the next output token in the sequence conditioned on the tokens generated so far. This ensures that each generated element is contextually aligned with the previous (i.e., already generated) text. Additionally, cross-attention layers in the decoder access the encoder's output to guide the generation process. These layers allow the decoder to reference the full context provided by the encoder, ensuring that each output token is a contextually appropriate continuation or response to the specific provided input sequence. This comprehensive mechanism of encoding, self-attention, and cross-attention within the Transformer allows for highly effective processing and generation of text, making it a robust model for various complex language understanding and generation tasks.

We now recall some LLM architectures (Section 2.5.1) and notions related to the analysis of LLM inner working (Section 2.5.2).

2.5.1 Architectures

Building on the innovative self-attention mechanism of the Transformer, subsequent models have been developed with specialized architectures tailored for different tasks. `ELMo` [389] defined the usage on contextual embeddings. Encoder-only models, like `BERT` [139], specialize in tasks that require a deep understanding of language context, making them ideal for applications like sentiment analysis and named entity recognition. On the other hand, decoder-only models, such as `GPT-3` [62] and `GPT-4` [64, 374], excel in generating coherent and contextually appropriate text, powering applications in creative writing and dialogue systems.

After encoder and decoder-only models, the introduction of instruction-based models marks a significant evolution in LLMs. These models are trained to follow user-provided instructions [106, 376], making them versatile tools across various tasks without needing task-specific fine-tuning. Instruction-based training involves exposing the model to various tasks during training, along with corresponding instructions, thereby enabling the model to generalize from instructions at inference time. This approach has started the development of emerging abilities in LLMs, such as zero-shot learning [263], complex reasoning strategies [50, 496], and the discovery of latent abilities [495].

Following the introduction of instruction-based models in LLMs, several state-of-the-art models have epitomized this approach, showcasing remarkable capabilities in handling a wide array of tasks directly based on user instructions. Some notable models in this field include `PaLM` [105], which represents a significant advancement in scaling up transformer-based architectures designed to perform well across diverse linguistic tasks. Its successor, `PaLM-2` [202], builds upon this foundation with improved training techniques and larger model capacities, further enhancing its ability to understand and generate nuanced text based on instructions. `PaLM-E` [162] further extends the `PaLM` series by emphasizing efficiency in energy usage and processing speed, making it a more sustainable option for deploying sophisticated NLP tasks at scale.

Another architecture is `LLaMA` [463] and its successors, `LLaMa 2` [193] and `LLaMa 3` [312], which focus on achieving high cross-task effectiveness with relatively smaller model sizes, facilitating easier deployment and lower operational costs without compromising on capability. These architectures have demonstrated significant prowess in tasks requiring

deep contextual understanding. `Mistral 7B` [239] and `Mixtral 8×7B` [240] introduce a unique approach to model training called Mixture of Experts (MoE) that involves dynamic adjustments of model parameters based on task complexity, which enhances the model's adaptability and performance across different NLP tasks. `Gemini 1.5` [457] is another innovative model that integrates dual mechanisms of understanding and generation to improve interaction dynamics in conversational AI applications. `Bard` [328] focuses on incorporating broad, encyclopedic knowledge and the ability to update its understanding in real-time, making it exceptionally useful for applications that require up-to-date information. Finally, `Claude` [25] distinguishes itself by its ethical training framework, prioritizing safety and fairness, setting a new standard in responsible AI development.

2.5.2 Large Language Models Interpretability

Despite their remarkable capabilities, the internal mechanisms by which LLMs encode and organize knowledge remain only partially understood. Two complementary approaches are commonly employed to investigate these mechanisms: *direct querying* (also referred to as *direct questioning* or *prompting*), which analyzes model behavior through carefully designed prompts, and *probing*, which examines the information embedded within the model's hidden representations.

Direct querying is a methodology used to assess the behavior of LLMs by presenting them with a prompt and analyzing their generated responses. It aims to evaluate what models know or can recall when explicitly prompted, without inspecting attention weights, internal activations, or any other internal mechanisms of the model itself. A well-known example of direct querying is the *needle in a haystack* task,[7] in which a factual statement (i.e., a short text fragment) is embedded within a long context, and the model is prompted to retrieve it *verbatim*. This methodology has been extended to other applications, such as document summarization and citation generation [272], to evaluate whether relevant information can be surfaced by the model when explicitly prompted to recall it.

Probing offers a systematic and empirically grounded framework for investigating the internal representations of LLMs. It provides a structured methodology to examine what kinds of information are captured within the models' internal mechanisms, where such information is stored, and how it is organized and retrieved across layers [248, 289, 330]. Unlike the output-based evaluation described above, which focuses on model behavior elicited through direct querying, probing operates by extracting the hidden layer activations produced as the model processes an input, and analyzing these activations using lightweight predictive models known as *probes*. This approach enables researchers to assess whether specific properties, such as syntactic, semantic, numerical, or domain-specific features, are implicitly encoded within the model's latent representations.

A typical probing setup comprises three main stages: *(i)* a controlled set of inputs is fed into a frozen LLM to collect activations from one or more internal layers of interest, typically the final layer; *(ii)* an external model (i.e., the probe) is trained on these representations to predict a target property; and *(iii)* the probe's performance is evaluated on unseen data to assess its generalization on the same property. Common probe architectures include linear classifiers and simple Multi Layer Perceptrons (MLPs), each suited to investigating

[7] https://github.com/gkamradt/LLMTest_NeedleInAHaystack/tree/main, accessed 14 Oct 2025.

different aspects of how information is encoded within LLMs [43, 255, 483]. Simpler probes, such as linear regressors, are typically used to examine whether a property is linearly accessible from the model's representations, whereas more complex, non-linear probes can explore relationships that may be distributed or non-linearly structured within the latent space [234].

Although originally developed for linguistic analysis, probing has been applied to a broad range of interpretability tasks. Several studies have examined whether hidden layers encode syntactic categories or dependency relations, showing that intermediate layers tend to capture structural information, whereas deeper layers increasingly specialize in semantic content [43, 481]. The same framework has also been extended to investigate higher-level capabilities such as discourse awareness [268], factual and relational knowledge [96], and hierarchical reasoning [19]. For instance, a probe may be trained to distinguish correct from incorrect factual triples (e.g., "Paris is the capital of France" vs. "France is the capital of Paris") or to recover the latent coordinates of entities mentioned in text, thereby revealing whether spatial or relational concepts are implicitly represented within the model. Such applications demonstrate that probing can shed light on the organization of latent knowledge in LLMs, extending well beyond surface-level linguistic features.

A key advantage of probing lies in its architecture-agnostic nature: it treats the model as a frozen feature extractor and requires neither fine-tuning nor parameter updates. Consequently, they ensure that the subsequent analysis based on the model's weights accurately reflects its learned internal state [46].

2.6 Metrics, Statistical Analysis, and Benchmarking

We focus on standard performance evaluation techniques in CO. Section 2.6.1 introduces the concept of the relative gap. Section 2.6.2 presents the set of descriptive statistics employed throughout this thesis, while Section 2.6.3 details the statistical tests used for performance comparison. Section 2.6.4 introduces the concept of Critical Difference (CD) plots for visualizing statistical comparisons. Then, Sections 2.6.5 and 2.6.6 describes the Instance Space Analysis (ISA) framework and what Search Trajectory Network (STN) are.

2.6.1 Gap

The relative gap is a performance metric used to evaluate the quality of a solution relative to a reference one. Given a solution s with an objective value $obj(s)$, and a baseline solution s_b with objective value $obj(s_b)$, the gap of s with respect to s_b is defined as:

$$\text{gap}(s) = 100 \cdot \frac{obj(s) - obj(s_b)}{obj(s_b)} \tag{2.1}$$

A gap value of 0 indicates that the two solutions have identical objective values. For minimization problems, a negative gap means that the solution s is better than the baseline s_b, while a positive gap indicates a worse solution compared to the baseline.

2.6.2 Descriptive Statistics

Descriptive statistics are used to summarize the algorithms performance across multiple runs and instances. They provide an understanding of central tendencies, variability, and overall behavior before conducting formal statistical tests. In this thesis, we use:

- **Mean.** The average objective value[8] across multiple runs. It represents the expected performance of the given algorithm.
- **Median.** The middle value in the sorted list of objective values, providing a measure of central tendency that is robust to outliers.
- **Standard Deviation.** A measure of the variability in the results, indicating the stability or robustness of the algorithm across runs.
- **Minimum and Maximum.** The best and worst observed objective values, in the case of a minimization problem.

These statistics are particularly relevant for stochastic algorithms, like metaheuristics, where variability between runs can be significant due to their random components. For exact methods, the standard deviation is trivially zero.

2.6.3 Statistical Tests

To assess whether the observed performance differences among the algorithms are statistically significant, we mainly adopt non-parametric statistical tests, i.e., the Friedman Test and the Wilcoxon Signed-Rank Test. Additionally, for what concerns the analysis reported in Chapter 18, a more structured statistical pipeline is envolved, comprising Mixed Linear Models, Likelihood-Ratio Test, Tukey's Honestly Significany Difference Test, and Wilcoxon Two One-Sided Tests for Equivalence.

Friedman Test

The Friedman test is a non-parametric equivalent of the repeated-measures Analysis of Variance (ANOVA) and is specifically designed for comparing multiple algorithms over multiple instances. It tests the null hypothesis that all algorithms perform equally, considering their average ranks across instances. For each instance, the algorithms are ranked based on their performance, with the best-performing algorithm receiving rank 1. The Friedman test then evaluates whether the observed differences in these ranks are statistically significant beyond what could be expected by chance. For example, if the test is applied with a significance level of 0.05, this means that there is a 5% probability of incorrectly rejecting the null hypothesis when it is actually true (Type I error). If the p-value resulting from the Friedman test is less than the significance level, we reject the null hypothesis and conclude that at least one algorithm performs differently from the others. The Friedman test is particularly appropriate when the assumptions of parametric tests (e.g., normality, homoscedasticity) do not hold, which is often the case in CO experiments.

[8] We use the objective value as a proxy for performance; nevertheless, alternative metrics such as the optimality gap, number of iterations, or computational time may also be considered, depending on the analysis.

Wilcoxon Signed-Rank Test

If the Friedman test rejects the null hypothesis, pair-wise post-hoc analyses are conducted using the Wilcoxon Signed-Rank Test. This is a non-parametric test used to compare two related samples, i.e., the performance ranks of two algorithms over the same set of instances. To control the family-wise error rate due to multiple pairwise comparisons, we apply Holm's correction, i.e., we adjust the p-values to maintain the overall significance level, thereby reducing the risk of Type I errors when performing multiple tests.

All in all, the Friedman test determines whether there is a statistically significant difference among the set of algorithms as a whole. When this is the case, the Wilcoxon Signed-Rank Test identifies which specific pairs of algorithms differ significantly.

Mixed Linear Models

Mixed Linear Models (MLMs), or linear mixed-effects models, extend classical linear regression by incorporating both *fixed* and *random* effects. Fixed effects represent systematic, population-level influences, while random effects capture variability due to hierarchical or repeated-measure structures in the data.

Reported results typically include estimated coefficients, standard errors, z-statistics, p-values, and 95% confidence intervals for fixed effects, along with variance estimates for random effects. Coefficients indicate the direction and strength of predictor–response relationships, with confidence intervals providing a measure of estimate precision.

Likelihood–Ratio Test

The Likelihood–Ratio Test (LRT) is a general method for comparing the goodness of fit between two *nested* statistical models, where the reduced model can be obtained from the full model by constraining one or more parameters (e.g., setting them to zero). The test evaluates whether the additional parameters in the full model significantly improve the model's ability to explain the data. A significant result indicates that the full model provides a significantly better fit, implying that the additional parameters contribute meaningfully to explaining the response variable.

Tukey's Honestly Significant Difference Test

The Tukey's Honestly Significant Difference (HSD) test is a post-hoc procedure that controls the Family-Wise Error Rate (FWER) across all pairwise contrasts among group means. The method is based on the *studentized range distribution*, which adjusts the critical value to maintain the overall significance level. By accounting for the number of comparisons being made, it ensures that the probability of committing one or more Type I errors remains within the specified threshold.

When an analysis involves categorical factors with more than two levels, a significant omnibus test (such as ANOVA or an LRT) indicates that at least one group mean differs from the others but does not identify which specific pairs differ. Tukey's HSD is therefore applied to perform post-hoc pairwise comparisons that determine where these differences occur while preserving statistical validity under multiple testing.

Wilcoxon Two One-Sided Tests for Equivalence

Traditional Null-Hypothesis Significance Testing (NHST) only determines whether an observed difference is statistically distinct from zero, but does not address whether two quantities can be considered practically equivalent. To assess practical equivalence, the Two One-Sided Tests (TOST) procedure is used. The *Wilcoxon-based TOST* is a nonparametric variant that relies on the Wilcoxon signed-rank test and is therefore robust to deviations from normality and the presence of outliers.

Given paired observations, the differences between them are tested under the composite null hypothesis that the true median difference Δ lies outside an equivalence interval: If both one-sided tests are rejected at the chosen significance level, the two samples are deemed statistically equivalent within the bounds of the considered equivalence interval.

The *Hodges–Lehmann estimator* is typically reported as a robust measure of the median difference between paired observations, providing an interpretable, distribution-free estimate of the central tendency. Together, the Wilcoxon TOST and Hodges–Lehmann estimator offer a nonparametric tool for evaluating equivalence in a robust way with respect to non-normal data and small sample sizes.

2.6.4 Critical Difference Plots

Critical Difference (CD) plots were introduced by Demšar [137] as a way to visually represent the performance of multiple algorithms over a set of instances. In a CD plot, each algorithm is positioned along the horizontal axis according to its average rank across all instances. For example, if five algorithms are being compared, each algorithm is ranked from 1 to 5 on each instance based on the cost of the solution it produces, with 1 assigned to the best-performing algorithm. The average rank across all instances is then computed for each algorithm. Algorithms whose performances are not statistically different w.r.t. pair-wise tests are considered equivalent and this is visually indicated by a thick horizontal line. Figure 2.4 reports a visual example of CD plot.

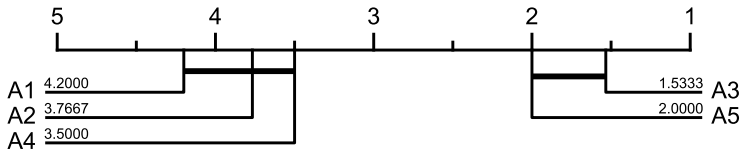


Figure 2.4: Example of a Critical Difference (CD) plot for five algorithms.

2.6.5 Instance Space Analysis

Instance Space Analysis (ISA) is a methodology developed by Smith-Miles and Muñoz [436] that builds on the Algorithm Selection Problem (ASP) introduced by Rice [411]. It represents the relationship between problem instance properties and algorithm performance. Specifically, instances are projected in a 2D space, so that one can: *(i)* Visualize the distribution and diversity of instances. *(ii)* Assess instance features. *(iii)* Identify and measure algorithm strengths and weaknesses (algorithm footprints). *(iv)* Locate areas where additional instances could offer more insights.

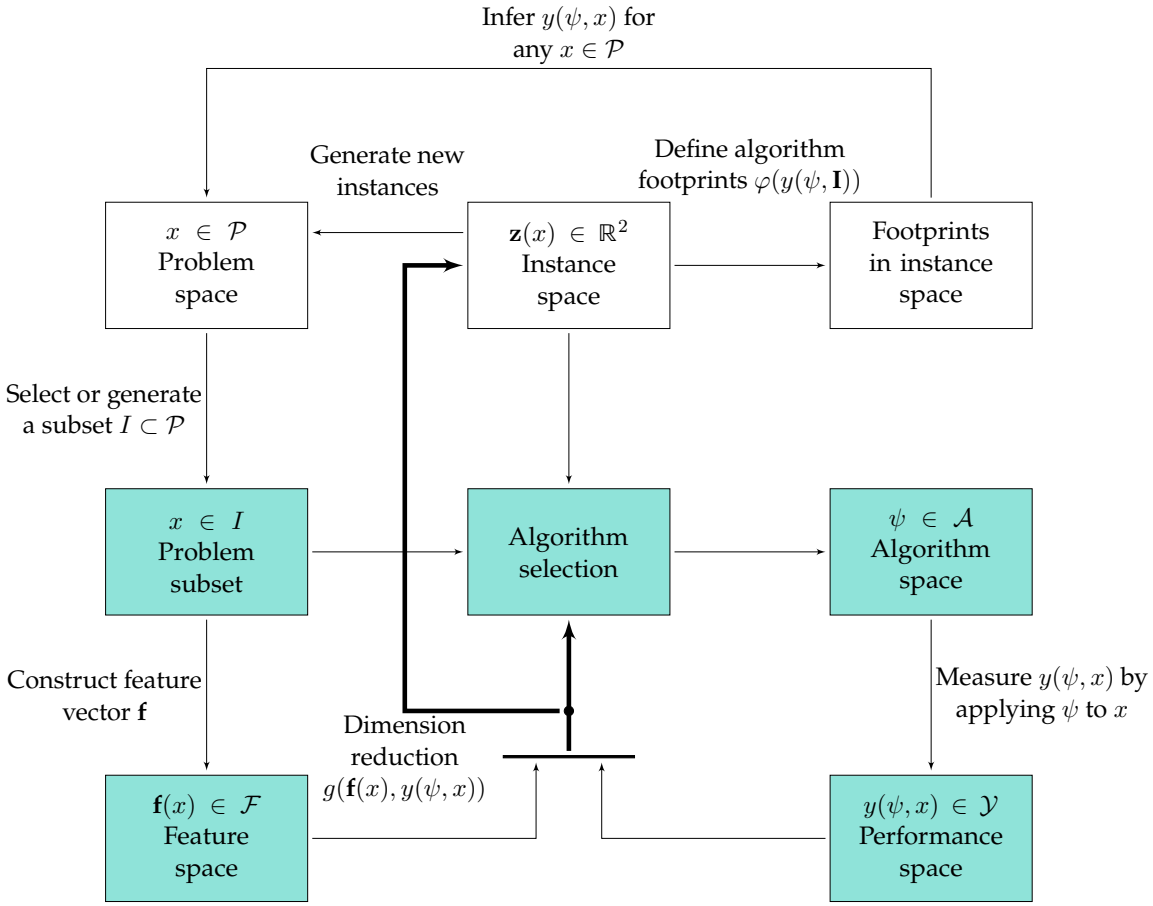


Figure 2.5: Instance Space Analysis (ISA) framework [436] extending the Algorithm Selection Problem (ASP) framework by Rice [411]. The blocks in common with Rice [411] are highlighted.

Figure 2.5 presents the general framework for developing an ISA as detailed by Smith-Miles and Muñoz [436]. The building blocks related to the work of Rice [411] are highlighted. The problem space, denoted by $\mathcal{P}$, represents all possible problem instances. However, computational results are typically only available for a subset, denoted as I. The feature space, $\mathcal{F}$, consists of vectors $\mathbf{f}$ that contain measures describing and characterizing the instances in I. These instances are solved using a set of algorithms constituting the algorithm space $\mathcal{A}$. The algorithms' performance is captured in the performance space, denoted by $\mathcal{Y}$, which is the set of metrics $y(\psi, x)$ that measure the performance of algorithm $\psi \in \mathcal{A}$ when solving instance $x \in I$. The algorithms' performance, together with the instance features, forms what is known as metadata, which is used to learn a mapping $g(f(x), y(\psi, x))$. This mapping projects an instance x from the high-dimensional feature space to a 2D space, referred to as the instance space.

ISA is closely connected to the topic of algorithm selection. Specifically, by mapping problem instances into an instance space and identifying the performance footprints of algorithms, one can gain insights into which algorithms perform best on different regions of the space. Currently, ISA represents one of the state-of-the-art tools for algorithm selection, although alternative approaches are also being explored. A historical overview of algorithm selection methods is provided in Section 3.3.

There exist off-the-shelf tools for ISA, including a web-based platform[9] and a local implementation available on GitHub.[10]

In this thesis, we adopt ISA in Chapter 11. However, an alternative, problem-indipendent form is explored in Chapter 18.

2.6.6 Search Trajectory Networks

Search Trajectory Network (STN) provides visualization of algorithms in the form of directed graphs and is used to analyze and compare the search behavior of stochastic algorithms such as metaheuristics. STNs are based on the following notions, as introduced by Ochoa, Malan, and Blum [370]:

- **Representative solution**: A solution to the problem at hand at a given point in time. It represents the state of the search algorithm (e.g., the new best solution found by SA during the search).
- **Location**: A non-empty subset of representative solutions coming from a partition of the search space (note that if no search space partitioning is applied, then the set of locations and the set of representative solutions coincide). Each representative solution belongs to exactly one location.
- **Search trajectory**: A sequence of locations created by substituting each representative solution in a sequence with its corresponding location, based on the order in which the solutions are encountered during the search.
- **Node**: A location in the search trajectory.
- **Edges**: Directed edges connecting two consecutive locations in the search trajectory. Their weight is proportional to the number of transitions between the two locations.
- **Search Trajectory Network (STN)**: The directed weighted graph constructed using the nodes, edges, and weights defined in the above points.

STN insights emerge both through visual representations and through quantitative metrics, such as the number of locations, the number of locations visited by only one algorithm, and the average length of algorithm paths, among others.

STNs are available both as a standalone[11] and a web-based tool [92]. STNs have been recently extended to the multi-objective case [280] and integrated with LLMs [90].

In this thesis, we adopt STNs in Chapter 5 to explore the internal dynamics of TS, and in Chapter 12 to assess solution methods for the OSP.

[9] https://matilda.unimelb.edu.au/matilda/. Accessed 2025–10–31.

[10] https://github.com/andremun/InstanceSpace. Accessed 2025–10–31.

[11] https://github.com/gabro8a/STNs-v2. Accessed 2025–10–06.

Chapter 3

Related Work

This chapter reviews the literature most relevant to the themes and methodologies developed throughout this thesis. Table 3.1 provides an overview of how the concepts discussed here relate to the subsequent parts of the manuscript. Portions of this chapter draw on the author's published or submitted work.

It is important to note that this chapter does not cover studies applying Large Language Models (LLMs) directly to Combinatorial Optimization (CO), but rather surveys on the topic (Section 3.8). The methodological and empirical aspects of such applications are examined in Chapter 17 (Part V), which presents a systematic literature review.

Table 3.1: Overview of how the related work sections introduced in this chapter connect to the subsequent parts and chapters of the thesis. Each entry lists the main topic addressed and the chapters where the corresponding concepts are applied or extended.

Reference	Topic	Use within the thesis
Section 3.1	Algorithm analysis, including component-based studies, Instance Space Analysis (ISA), and Search Trajectory Networks (STNs).	Chapters 5, 11 and 12.
Section 3.2	Algorithm configuration and parameter tuning, motivating adaptive approaches.	Chapter 6; offline tuning used elsewhere.
Section 3.3	Algorithm selection based on problem-dependent and problem-independent features.	Chapters 11 and 18; feature sets also used in Chapters 5, 14 and 15.
Section 3.4	Parallel Batch Scheduling and the Oven Scheduling Problem.	Part III, particularly Chapter 10.
Section 3.5	Lower bounds for Parallel Batch Scheduling Problems.	Chapter 9.
Section 3.6	Unified formulations of related optimization problems.	Part IV, particularly Chapter 14.
Section 3.7	Home Healthcare Routing and Scheduling Problems.	Part IV.
Section 3.8	Surveys on Large Language Models for Optimization.	Chapter 17.

3.1 Algorithm Analysis

In recent years, the metaheuristics community has shown growing interest in the systematic analysis of optimization algorithms, as evidenced by initiatives such as the Randomized Optimization Algorithms Research Network (ROAR-NET)[1], the Metaheuristic in the Large (MitL) [450], and the Benchmarking, Benchmarks, Software, and Reproducibility (BBSR) track at the Genetic and Evolutionary Computation Conference (GECCO).[2]

General guidance on the analysis of optimization algorithms is provided by Bartz-Beielstein et al. [37], Zufferey [526], and Hertz and Widmer [219], who outline best practices, methodologies, and common pitfalls.

Besides statistical analyses, representative techniques include Instance Space Analysis (ISA) [436], Local Optima Networks (LONs) [371], and Search Trajectory Networks (STNs) [370]. The first approach treats algorithms as monolithic entities and relates their performance to instance features [191, 446], allowing also for algorithm selection (Section 3.3). The second approach investigates the structure of problem landscapes considering the objective space [32, 55], specifically a network is constructed where each node is a local optimum and the edges are the possible transitions between them. The third approach contrasts and compares algorithmic behavior in the search space [15, 281]. Specifically a network is constructed where each node is a representative location or solution visited by a given algorithm.

More recently, researchers have started to investigate the role of individual components within metaheuristics. Ablation analyses [173] have been used to isolate and evaluate component effects [257, 340]. Franzin and Stützle [183] offered a detailed decomposition of Simulated Annealing (SA) categorizing its components and demonstrating the value of a component-based framework through automated algorithm design. Doerr et al. [156] studied the Metropolis algorithm on the CLIFF benchmark, showing that even with careful tuning, it performs worse than simple evolutionary algorithms. Other examples include the use of LONs to analyze iterated local search [129], and a meta-analysis of Adaptive Large Neighborhood Search (ALNS) by Turkes, Sörensen, and Hvattum [470], who statistically assessed the impact of adaptiveness. ALNS has also been studied from the perspective of acceptance criteria, with threshold acceptance and record-to-record travel identified as particularly effective [422]. Complementary to empirical studies, theoretical analyses have contributed to understanding metaheuristic behavior via runtime analysis [107, 138].

Considering Tabu Search (TS), early work by Di Gaspero, Chiarandini, and Schaerf [141] showed that short-term tabu lists yield similar performance in a timetabling problem. Parameter sensitivity of reactive TS has been investigated by Pellegrini et al. [385]. Danielsen and Hvattum [126] compared solution-based and attribute-based tabu search in binary integer programming, finding no statistically significant differences. Other works have explored tabu tenure policies combined with machine learning [265] and different aspiration criteria [235]. In this thesis, we extended this body of knowledge by studying the performance of different tabu list strategies and their interaction with the definition of inverse moves.

[1] https://roar-net.eu/. Accessed 2025–08–27

[2] https://gecco-2025.sigevo.org/. Accessed 2025–09–18

3.2 Automated Algorithm Configuration

As discussed in Section 2.1, one of the key activities in optimization involves *algorithm configuration*, that is, the selection of suitable values for the parameters that govern an algorithm's behavior. Over the past two decades, the literature has increasingly advocated for the automation of this process, with the goal of selecting design choices in a principled, unbiased, and reproducible manner [222].

Algorithms introduced in Sections 2.3 and 2.4, for instance, expose numerical parameters, such as the initial and final temperature in SA, the tabu tenure length in TS, or the neighbourhood selection biases in multi-neighbourhood search. However, the term *parameter* may be interpreted more broadly, encompassing not only numerical settings but also structural or procedural design decisions, as in the case of automated algorithm design [334]. In this broader sense, the automated configuration process can extend beyond tuning scalar values to constructing algorithmic frameworks themselves, thereby blurring the boundary between configuration, selection, and design. For instance, one can view portfolios of algorithms as large composite systems, where the parameter to be configured corresponds to the algorithm itself or to the strategy used to select among them (Section 3.3).

One practical way to perform algorithm configuration is through automated offline tools such as `Irace` [316] and `SMAC` [298]. These methods operate by sampling configurations within predefined parameter intervals, iteratively evaluating them across sets of training instances, and selecting those that perform best on average. While such tools have proved effective in improving performance and reducing human bias, they provide limited insight into *why* certain configurations perform well or on which types of instances they succeed. As a result, the process remains largely empirical and often detached from interpretability.

To address this limitation, Pushak and Hoos [396] investigated the properties of algorithm configuration *search landscapes*. By analysing single-parameter slices, the authors observed that configuration landscapes aggregated across instance sets tend to be unimodal and convex, suggesting the presence of relatively smooth tuning spaces. In contrast, studies such as Cleghorn and Ochoa [111] and Treimun-Costa et al. [465], which explored parameter spaces of Particle Swarm Optimization (PSO) and Genetic Algorithms (GAs) on single continuous optimization instances, revealed highly multimodal landscapes with multiple suboptimal basins. These findings highlight that the structure of configuration landscapes is problem- and instance-dependent, and that global tuning may overlook meaningful local regularities.

Focusing on CO, Bellio et al. [45] linked instance features to optimal parameter settings in SA for the Curriculum-based Course Timetabling (CBCTT), showing that no single configuration performs best across all instances. By framing tuning as a classification task, thus turning to an algorithm selection point of view, they outperformed automated tuning tools on unseen instances. Similarly, Battistutta, Schaerf, and Urli [38] treated parameter tuning for a SA applied to the Educational Timetabling (ETT) as a regression problem, learning relationships between instance descriptors and effective parameter values. More recently, Franzin and Stützle [182] examined the behavior of fixed-temperature and classical cooling variants of SA, showing that performance depends strongly on the encountered search landscape. Together, these studies suggest that dynamic and context-aware parameter adaptation during the search process represents a promising direction

beyond static offline tuning.

In this respect, the introduction of Machine Learning (ML) and Reinforcement Learning (RL) has become increasingly relevant [325]. Such techniques have been employed, for instance, to guide the selection of search operators within SA [82]. Similarly, Hyper-Heuristics (HHs) have been used in conjunction with SA to adapt the exploration of the search space by dynamically varying neighborhood structures [115, 413, 426] and more broaldy to tackle real-world problems [258, 345].

In this thesis, we use both offline and online tuning. Regarding the latter, we introduce an adaptive temperature control mechanism for SA (Chapter 6). Here, parameter adaptation is guided by a HH that reacts to the current search state, replacing fixed offline calibration with online, data-driven control.

3.3 Algorithm Selection

Within CO, algorithmic performance varies substantially across instances: an algorithm that performs well on one subset of instances may be ineffective on another, depending on factors such as instance size, structure, or constraint tightness. This phenomenon is formalized by the No Free Lunch (NFL) theorem [498], which asserts that no single algorithm can be superior across all possible problems or instances.

The challenge of selecting algorithms according to instance characteristics was systematically conceptualized by Rice [411], who defined the Algorithm Selection Problem (ASP) as the task of learning a mapping between instance features and algorithm performance. In Rice's framework, each problem instance is described by a vector of measurable characteristics, and each algorithm is evaluated by one or more performance metrics; the objective is to predict which algorithm is expected to perform best for a given instance, i.e., a supervised learning task. This formulation established the foundation for per-instance algorithm selection and inspired subsequent developments in algorithm portfolios, which are collections of complementary algorithms designed to exploit performance diversity across instances [430, 506].

Successful algorithm selection increasingly depends on quantitative instance characterization, that is, identifying features that capture structural and behavioral properties of problem instances. To support this, frameworks such as ISA [436] and tools such as MATILDA[3] have been developed, providing unified environments for feature processing, visualization, and the mapping of algorithm performance across low-dimensional instance spaces [116, 300]. For further specification, we forward the interested reader to Section 2.6.5.

Despite these advances, feature extraction still relies heavily on handcrafted, problem-specific descriptors that require substantial domain expertise. To mitigate this limitation, recent studies have explored automated feature learning. For example, Neural Networks (NNs) have been used to infer latent representations of instances directly from data [18, 386], extending also to black-box optimization problems [78].

In this thesis, we use both hand-crafted, problem specific descriptors (Chapter 11) and problem-independent ones (Chapter 18) for algorithm selection. Regarding the latter, to the best of our knowledge, we are the first to employ LLMs for this purpose, leveraging their

[3] https://matilda.unimelb.edu.au/matilda/. Accessed 2025–10–22.

representational capacity across a broader set of Combinatorial Optimization Problems (COPs) and multiple instance modalities, including `natural language`, `code-like`, and `standard` representations. Furthermore, hand-crafted, problem-specific features are also employed to characterize instances in Chapters 5, 14 and 15.

3.4 Parallel Batch Scheduling Problem

Part III deals with the Oven Scheduling Problem (OSP), a parallel batch scheduling problem. This family of problems has been extensively studied over the last three decades [180], spanning across diverse industries including manufacturing [455] and steel production [525]. Each of these variants exhibits differences to the OSP in respect to the constraints, such as the usage of a single machine [124, 339], allowance for parallelism [125, 455] and the objective, which typically is the makespan [125, 342, 485], tardiness [525], or flowtime [339].

Several solution approaches have been explored, ranging from exact methods like Constraint Programming (CP) [466] and Mixed Integer Programming (MIP) [31, 267] to metaheuristic techniques such as SA [94, 104, 125, 342], TS [77, 269, 339], GA [181], Ant Colony Optimization (ACO) [100], and ALNS [225, 504].

Although SA and an adaptive form of Large Neighborhood Search (LNS) has been previously applied to related problems, our approaches are distinct due to the incorporation of novel algorithmic components. For instance, related to the SA, we include a cut-off mechanism, an innovative combination of four neighborhood structures, and a thorough parameter setting through automated parameter tuning.

Table 3.2 provides an overview of related works. For each study, we indicate the application industry where the formulation was applied (— indicates no specific industry), the objective/objectives, whether the machine setting is parallel (denoted by a ✓), the solution method, and, for local search methods, the operators used to explore the neighborhoods.

Table 3.2: Overview of related works on Parallel Batch Scheduling Problems.

Ref.	Manufacturing	Objective	Parallel	Solution method	Local search operator
[125]	semiconductor	makespan	✓	SA	swap jobs, move jobs
[455]	composite	tardiness, batches	✓	CP, heuristic, benders decomposition, hybrid	
[525]	steel	tardiness		MIP	
[339]	semiconductor	flowtime		TS	swap batches
[94, 124]	semiconductor	makespan		SA	swap jobs
[485]	–	makespan	✓	SA, GA	swap jobs (global ordering)
[342]	–	makespan		SA	swap jobs
[466]	semiconductor	makespan	✓	arc-flow	
[31]	–	flowtime		MIP (branch & bound)	
[267]	–	tardiness	✓	MIP	
[104]	semiconductor	tardiness	✓	MIP, SA, heuristic, dynamical programming	swap jobs (global order)
[269]	pharmaceutical	flowtime, job rejection	✓	TS	swap jobs, move jobs
[77]	semiconductor	tardiness	✓	TS	move jobs, swap jobs
[181]	semiconductor	flowtime		GA	
[100]	semiconductor	makespan	✓	ACO	
[504]	–	makespan	✓	ALNS	
[225]	additive	tardiness	✓	ALNS	

Rgearding the OSP, it was first introduced by M. Lackner et al. [275] in collaboration with an industrial partner in Central Europe with the aim on one side of accomodating

industrial needs and on the other to bridge the multiple constraints arose in the literature. The authors proposed a construction heuristic, two exact formulations, one based on CP and one on Integer Linear Programming (ILP), and released a benchmark dataset of 80 instances [274], together with an instance generator, all publicly available on `Zenodo`. These instances vary in the number of jobs (10, 25, 50, 100), the number of machines (2 or 5), and the number of attributes (2 or 5).

The proposed solution methods can be summarized as follows:

Construction heuristic This dispatching rule prioritizes jobs by release date and then by due date. The algorithm starts at time 0 and, at each step, considers available machines and released jobs that remain unscheduled. The job with the earliest due date is greedily assigned to an eligible machine, and additional jobs are added to the same batch if attribute, processing-time, and capacity constraints allow it. If no job can be scheduled, time advances by one unit. Whenever all jobs are successfully scheduled, the resulting solution is feasible and its cost constitutes an upper bound on the optimal solution.

Batch-based model (exact model) This formulation assigns jobs to batches defined by their machine and processing position. Constraints are defined at the batch level, and the model seeks an optimal sequence of batches.

Representative-based model (exact model) This variant designates one representative job per batch and determines an optimal schedule for these representative jobs.

Both eaxct models were implemented in the solver-independent language `MiniZinc` [362] and in Optimization Programming Language (OPL) [218] using interval variables within `CP Optimizer`. The study evaluated several solvers, search strategies, and a warm-start strategy based on the construction heuristic. The best performance was obtained with the OPL model using representative jobs solved by `CP Optimizer`, and with the `MiniZinc` batch-position model solved by `Gurobi`. While these approaches achieved optimal solutions for instances with up to 25 jobs, they struggled with larger cases within a one-hour time limit. This highlights a persistent scalability challenge and motivates the development of heuristic and metaheuristic methods capable of producing high-quality solutions efficiently for real-world, large-scale instances.

3.5 Lower Bounds for Parallel Batch Scheduling Problems

Lower Bounds (LBs) might be usefule in practice for several reasons, such as assessing solution quality, speeding up solution methods, guiding metaheuristic search, and characterize instance difficulties. Theoretical, problem-specific LBs for batch scheduling problems have been studied in the literature. Table 3.3 summarizes these works, highlighting the objectives and characteristics of the problems addressed. Most formulations consider simplified versions of batch scheduling problems compared to the OSP and are therefore not directly applicable to the problem at hand. For example, in multi-machine scenarios [35, 57, 125, 251], the machines are often assumed to be identical, with all jobs assignable to any machine. Moreover, distinctions between job attributes (e.g., size, release dates, etc.) are typically addressed partially, rather than in a more comprehensive and realistic manner.

It is worth noting that while all formulations listed in the table incorporate varying job processing times, none address processing times constrained between a minimum and maximum, as required in the OSP (column "min/max proc. time"). In other cases, while the problem formulation includes release dates for the jobs, these are disregarded in the LB calculation [293].

Table 3.3: Literature on problem-specific Lower Bounds (LBs) for machine scheduling problems.

Ref.	Lower bound	Batching of jobs	Release dates	Due dates	Min/max proc. time	Machine eligibility	Different job size	Job families	Precedence constraints	Multiple machines	Non-identical machines
[125]	Makespan	✓	✓				✓			✓	
[262]	Makespan, total completion time	✓					✓	✓			
[294]	Batch number, tardiness	✓		✓			✓	✓			
[293]	Makespan						✓			✓	✓
[251]	Tardiness		✓	✓						✓	
[35]	Tardiness, total completion time		✓	✓						✓	
[57]	Tardiness		✓	✓					✓	✓	
Us	Tardiness, setup costs, batch number, processing time	✓	✓	✓	✓	✓	✓	✓		✓	✓

In Chapter 9, we present LBs developed specifically for the OSP; in seldom cases, we build upon and extend several existing approaches; when such extensions occur, they are explicitly mentioned. Furthermore, although the literature on LB formulations for batch scheduling problems is relatively sparse, useful insights can be drawn from related problem domains. For instance, the job-to-batch assignment component of the OSP can be modeled as a Bin Packing Problem (BPP), for which numerous LB techniques have been proposed [174, 332]. Further considerations are provided in Chapter 9.

3.6 Formulation Unifications

In most cases, so-called *unification efforts* in the literature take the form of surveys or reviews. These works often provide lists of related problems but do not constitute genuine unification attempts. Nevertheless, in the more recent literature a few examples have emerged that are more closely aligned with the approach adopted in this thesis (Part IV).

Ceschia, Di Gaspero, and Schaerf [83] proposed a survey on ETT, focusing on specific formulations and corresponding benchmark instances, and also reporting best-known results, running times, and related information. Although the format is that of a review, the work is structured as a reference paper that supports fair benchmarking and helps to reduce the fragmentation of the ETT field, where research often focuses on a single formulation in isolation. Similarly, Dursunoglu et al. [166] presented a review of Selective Routing Problems (SRPs), starting from a generic mathematical model to describe connections and similarities among different formulations, while also uncovering cases where the same

problem appeared under different names. Finally, Queiroz, Lucas, and Sörensen [397] introduced a generic tool for on-demand transportation problems, including an instance generator, instance evaluation methods, and a module for analyzing algorithmic results. Unlike works focused on a single formulation, their contribution generalizes to the broader transportation domain.

The aims of such unification efforts, and consequently their importance in CO, can be summarized as follows:

- **Reducing fragmentation of the field.** COPs are often studied in isolation, even when they belong to the same family of problems or application domain. This results in each study adopting its own notations and benchmarks, which hampers comparability and knowledge transfer.
- **Facilitating transfer of methods.** A unified perspective enables algorithms developed for one problem variant to be applied to others, supports the identification of generic solution components (e.g., neighborhoods), and opens opportunities for transfer learning across problem classes.
- **Enabling benchmarking and fair comparison.** Unified formulations allow the translation of instances across problem definitions, foster the creation of shared datasets, and ensure that algorithms can be compared under fair conditions.

In Part IV, we focus on the Home Healthcare Routing and Scheduling Problem (HHCRSP) and we propose a formulation that captures many real-world aspects that have been treated previously in the literature in a separate manner. We also propose a toolbox for the encoding of the problem and the validation of solutions, alongside two solution methods.

3.7 Home Healthcare Routing and Scheduling

The problem of scheduling Home Healthcare (HHC) services has been extensively studied over the past three decades [109, 170, 176, 204, 336].

Begur, Miller, and Weaver [42] were the first to formally introduce the problem, addressing a simplified version which they solved using a Clarke & Wright routing heuristic. One year later, Cheng and Rich [101] proposed the first Mixed Integer Linear Programming (MILP) formulation for the problem. Later approaches modeled this problem as a set covering problem. Notably, Rasmussen et al. [405] made a significant contribution by being the first to incorporate temporal dependencies between activities into their formulation. Bertels and Fahle [49], Rendl et al. [410], and Hiermann et al. [220] investigated scenarios in metropolitan settings where caregivers use public transportation networks for traveling between patients' locations, rather than using dedicated vehicles. Additionally, Rendl et al. [410] and Hiermann et al. [220] expanded this concept by incorporating transportation mode changes during trips, enabling multi-modal journeys.

Di Gaspero and Urli [151] addressed a similar setting, albeit from a different perspective. Their study examined a multi-day horizon, aiming to minimize caregivers' overtime while ensuring balanced workloads. Furthermore, they introduced the option of leaving certain patients unscheduled, which more accurately reflects overloaded systems where covering all demand with existing staff is unfeasible, suggesting the potential need for hiring temporary

additional caregivers. To tackle this problem, the authors developed a CP framework featuring specialized branching heuristics and solved the problem with a LNS algorithm.

Mankowska, Meisel, and Bierwirth [327] proposed a standardization of the problem and provided a set of benchmark instances. Furthermore, they are the first to consider synchronization constraints at patients' homes, allowing for a maximum of two services to be simultaneously performed. The goal is to minimize the total travel time, the overall tardiness, and the highest individual tardiness. By incorporating the highest tardiness component, they aim to enhance fairness among patients, ensuring that no individual experiences excessive delays compared to others. To efficiently handle the problem, they designed an Adaptive Variable Neighborhood Search (AVNS) algorithm exploiting eight neighborhood operators. The same formulation has attracted interest also from Kummer [271], Neto et al. [363], and Neto, Buriol, and Araújo [364], who designed a Biased Random-Key Genetic Algorithm (BRKGA), which outperformed the results from Mankowska, Meisel, and Bierwirth [327]. They also extend the original dataset [327] with new instances that consider realistic travel times [271]. The results provided by Kummer [271] have recently been outperformed by Ceschia et al. [80], in which the authors implemented a Multi-Neighborhood local search guided by SA. This last work also introduced a new benchmark of realistic instances, developed by considering demographic data and for which Open Source Routing Machine (OSRM) has been used to compute actual travel times on the road network. The dataset by Mankowska, Meisel, and Bierwirth [327] has been used also by Lasfargeas, Gagné, and Sioud [279], who proposed a Variable Neighborhood Search (VNS) algorithm to address a multi-period HHCRSP on derived instances, and by Montemanni, Ceschia, and Schaerf [354], who solve the problem with a compact MILP.

The research by Ait Haddadene, Labadie, and Prodhon [14] also considered patient availability time windows but differed from Mankowska, Meisel, and Bierwirth [327] by treating them as strict constraints. Their objective focused on minimizing total travel times and penalties for unsuitable caregiver-patient assignments. This work was the first to consider integrating patient preferences into caregiver assignments, a crucial practical consideration, as effective HHC depends significantly on strong patient-caregiver relationships. A MIP model, a Greedy Randomized Adaptive Search Procedure (GRASP), and Iterated Local Search (ILS) are proposed to handle this problem. The performances of the approaches have been tested on the benchmark instances by Bredström and Rönnqvist [61] for the vehicle routing and scheduling with temporal precedence and synchronization constraints, suitably adapted to fit with the HHC application, in which multiple types of services are considered. The same setting has been addressed in Masmoudi, Jarboui, and Borchani [337], where several metaheuristic approaches are presented.

Several population-based approaches have been presented in the literature. Decerle et al. [133] and Grenouilleau et al. [203] developed Memetic Algorithms (MAs), combining GA with local search procedures, whereas a model-based Evolutionary Algorithm (EA) has been proposed by [110]. Similarly to Di Gaspero and Urli [151], Grenouilleau et al. [203] considered a multi-day horizon, aiming at minimizing total travel time plus penalties for overtime, while ensuring workload balance among caregivers. Their setting is more complex than the one addressed by Di Gaspero and Urli [151], as they incorporate time-dependent travel times between patients' locations.

More recently, Xiang, Li, and Szeto [503] and Oladzad-Abbasabady et al. [372] considered

a multi-objective version of the problem, looking for a balance between operational costs and patients' and caregivers' satisfaction, exploiting a Non-dominated Sorting Genetic Algorithm II (NSGA-II) and ILS algorithms, respectively. Kordi, Divsalar, and Emami [266] also addressed the problem with a Multi-Objective VNS approach employing four objectives, i.e., total cost, emissions, workload balance, and service quality.

Rich formulations dealing with real-world constraints and features include the following works. Liu, Yuan, and Jiang [308] were the first to consider lunch breaks for caregivers. In addition, W. Liu et al. [310] considered flexible departure locations (hospital or caregiver's home) and task synchronization. Conversely, Bazirha, Benmansour, and Kadrani [40] considers multiple time windows for the patients in addition to synchronization and de-synchronization requirements. Aguiar, Ramos, and Gomes [8] analyzed a context with a limited number of vehicles relative to the number of caregivers. This constraint means that a single car should be shared by two caregivers, requiring routes that accommodate patient visits for both staff members. Yadav and Tanksale [507] considered a very rich and complex pandemic setting in which patients select the level of safety measures (in terms of contact limitations) required (based on their pathology or their health conditions) and may impose requirements on the gender and the language spoken by the operator. The goal is to maximize the number of patients served and the revenue obtained. Unfortunately, their datasets are not publicly available. Varas et al. [474] incorporated additional real-world features: caregivers are required to return to the hospital for lunch breaks, some services involve the collection of perishable biological samples that must be delivered to the hospital within a maximum time frame, and patients express multiple availabilities.

Stochastic travel and service times have been introduced by Bazirha, Kadrani, and Benmansour [41]. The problem is formulated as a two-stage stochastic programming model, in which routing and scheduling plans are provided first. At the same time, penalty costs incurred for delayed services to patients and caregivers' overtime are computed at the second stage. The authors provided a GA to solve a deterministic version of the problem. The algorithm is embedded with a Monte Carlo simulation to compute the expected second-stage costs of the first-stage solution corresponding to each chromosome in the GA. This approach has been tested on a new dataset inspired by the benchmark of Mankowska, Meisel, and Bierwirth [327], properly extended to consider stochastic travel and service times. Liu, Yuan, and Jiang [307] proposed a chance-constrained model that handles stochastic travel and service times. Specifically, they employ a route-based deterministic formulation in which a route is feasible if it respects patients' time windows in at least a minimum percentage of a set of stochastic scenarios. A similar approach was adopted by Ma et al. [323] and Zhang et al. [522]. Shahnejat-Bushehri et al. [427], instead, proposed a robust approach.

Table 3.4 summarizes the features of the recent literature on the single-period HHCRSP, with Table 3.5 reporting the abbreviations.

Table 3.4: Objectives and features of recent single-period Home Healthcare Routing and Scheduling Problem (HHCRSP) formulations.

	Objectives								Features										
Ref.	FA	OT	TA	TC	TT	UP	WT	Other	CB	CC	MD	MS	PR	PC	SY	SK	TW	UN	WR
[327]	PR		✓		✓							✓			P-S-D	H	SI-H		
[220]			✓		✓		✓	skills violation			✓			S		S	SI-S		H
[14]					✓							✓		S	P-S	H	SI-H		H
[308]				✓		✓			P								SI-H		H
[8]					✓			working time, number of caregivers	P			✓			S		SI-H		H
[133]					✓						✓	✓			S	H	SI-H		H
[307]				✓		✓				✓	✓					H	SI-S	TS	
[310]				✓		✓			P								SI-H		H
[323]	OR		✓	✓			✓			✓	✓					H	SI-S	TS	S
[427]				✓				skills violation, number of caregivers							S	S	SI-H	TS	
[503]				✓										H		H	SI-H		H
[507]								max number served patients, revenue	P		✓			H		H	SI-H		
[41]		✓	✓	✓											P-S	H	SI-H	TS	
[40]	OR						✓								S	H	SI-H		H
[110]		✓		✓			✓									H	SI-H		
[266]	OR			✓				patients satisfaction								H	SI-H		H
[372]			✓	✓		✓	✓				✓			S	P-S-D	H	SI-S		H
[474]	OR				✓		✓		H				✓	H	S		M-H		
[522]							✓	patients satisfaction		✓						H	SI-H	S	H

Table 3.5: Abbreviations of objectives and features used in Table 3.4.

Objectives			Features		
Abbr.	Description	Values	Abbr.	Description	Values
FA	Fairness	PR: patients related, OR: operator related	CB	Caregiver break	H: hospital, P: at patient location, *Empty*: no
OT	Overtime	✓: yes, *Empty*: no	CC	Chance constraint	✓: yes, *Empty*: no
TA	Tardiness	✓: yes, *Empty*: no	MD	Multi departing/arrival points	✓: yes, *Empty*: no
TC	Travel cost	✓: yes, *Empty*: no	MS	Multi service patients	✓: yes, *Empty*: no
TT	Travel time	✓: yes, *Empty*: no	PR	Perishable biological samples	✓: yes, *Empty*: no
UP	Unvisited patients	✓: yes, *Empty*: no	PC	Preference	H: hard, S: soft, *Empty*: no
WT	Waiting time	✓: yes, *Empty*: no	SY	Synchronization	P: precedence, S: simultaneous, D: disjunctive, *Empty*: no
			SK	Skill requirements	H: hard, S: soft, *Empty*: no
			TW	Time Windows	M: multiple, SI: single, H: hard, S: soft, *Empty*: no
			UN	Uncertainty	T: travel times, S: service times, *Empty*: no
			WR	Working time regulations	H: hard, S: soft, *Empty*: no

3.8 Large Language Models for Optimization and Related Surveys

Early interest in combining LLMs with CO emerged from the Natural Language for Optimization Modeling (NL4Opt) competition [403], which investigated whether Natural Language Processing (NLP) techniques could assist mathematical modeling through entity recognition and automatic problem formulation, initially focusing on Linear Programming (LP) tasks. Building upon this, subsequent research has diversified along two main directions: *model definition and formulation* and *algorithm construction*. The related works are systematically surveyed in Chapter 17 following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines, while in this section, we provide an overview of related surveys, reporting their main characteristics and highlighting the differences with respect to our work.

A limited number of surveys have addressed the intersection of LLM and CO along with related topics. It is important to remark that none of these works are systematic reviews. Fan et al. [171] presented a review of Artificial Intelligence (AI) applications in Operations Research (OR), with a focus on three specific aspects of the optimization process: conversion of data into modeling parameters, model formulations, and model optimization. Although they extensively covered various AI techniques, such as RL and NNs, the usage of LLMs was investigated only in the context of model formulation [171, Section 4]. S. Huang et al. [230] explored the topic by including not only CO, but also continuous optimization and the use of optimization for LLMs. However, when discussing LLMs for optimization, they examined a set of 20 papers and focused on a narrow range of optimization aspects, specifically algorithm generation and the usage of LLMs as search operators. Lai et al. [277] reviewed the literature on LLMs concerning their mathematical capabilities and briefly discussed their applications in CO along with math word problems, geometry problems, and theorem proving. Wang and Li [489] reported a limited amount of literature works on LLMs for OR. F. Liu et al. [304] described the application of LLMs to algorithm design in various fields, like optimization, ML, mathematical reasoning, and scientific discovery. Wu et al. [500] provided a review on the intersection of Evolutionary

Computation (EC) and LLMs. As done by S. Huang et al. [230], they also covered aspects like continuous optimization and the use of optimization for LLMs enhancement. However, their focus on EC represented only a partial view of the CO landscape and, more generally, of optimization techniques. Similarly, Yu and Liu [515] and Cai et al. [72] described the evolution of EAs to solve optimization problems from heuristic approaches to LLMs.

Our systematic review, presented in Chapter 17, differ from the aforementioned surveys in the following ways:

1. We review the topic using a rigorous methodology to identify, screen, include, and analyze relevant literature works. We report the process using PRISMA 2020.
2. We limit our focus to the usage of LLMs within CO, thus excluding other types of optimization, as well as the use of optimization within LLMs, nor, more generally, the use of LLMs for mathematical problem-solving.
3. We address the entire optimization process, not focusing solely on specific parts.
4. We considered a broader set of optimization techniques and algorithms, not just evolutionary techniques.
5. We described datasets, frameworks, tools, and metrics that could enhance applications of LLMs within CO.

Part II

Metaheuristic Components

Chapter 4

Introduction

Despite the ubiquity of metaheuristics, our understanding of their inner workings remains limited. Most studies prioritise aggregate performance comparisons between complete algorithms, making it difficult to isolate the causal contribution of individual components or their interactions. A component-wise perspective is therefore essential to disentangle how design choices influence search dynamics and performance.

Among the many building blocks of metaheuristics, *memory* and *control mechanisms* play a particularly central role. In Tabu Search (TS), for instance, a rich variety of tabu list management strategies has been proposed (fixed vs. adaptive, short- vs. long-term), yet no consensus or systematic empirical comparison exists. As a result, it remains unclear whether observed differences stem from the tabu list policy itself or from co-varying design choices. Systematic behavioral analyses, e.g., via Search Trajectory Networks (STNs), remain relatively rare, limiting insight into how tenure schemes shape trajectory structure, stagnation patterns, or improvement phases.

A similar situation arises in Simulated Annealing (SA), where temperature control governs the balance between diversification and intensification. Classical schedules introduce multiple interdependent parameters (e.g., initial temperature, cooling rate, and reheating policy), and current practice largely depends on tuning-centric approaches that are computationally costly and prone to overfitting. Fixed or preset schedules often generalise poorly across problems (or even across regions of the same landscape) while reheating adds further parameters and complexity. These challenges motivate a broader shift from per-instance tuning toward adaptive, online control, where parameter updates respond dynamically to the encountered landscape.

4.1 Goals

Our goal to develop systematic, component-based analyses of metaheuristics and their interactions, aiming at a mechanism-level understanding. We focus on two widely used local-search metaheuristics: TS and SA.

For TS, we isolate the contribution of the tabu list by comparing strategies ranging from fixed to adaptive and from short- to long-term, while holding all other components fixed; we also contrast two inverse-move definitions (strict vs. relaxed).

Regarding SA, classical schedules involve multiple interdependent parameters (e.g.,

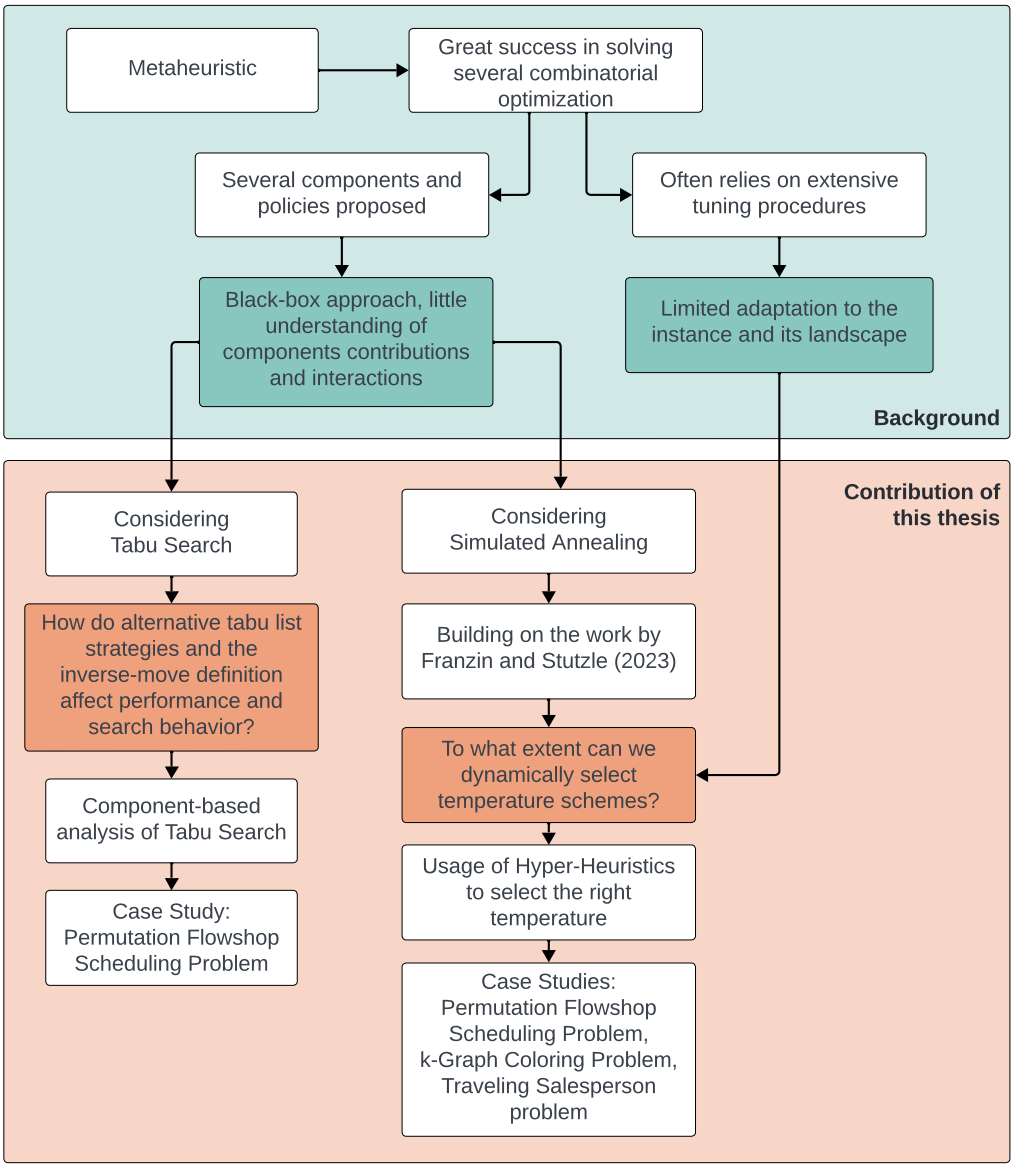


Figure 4.1: Schematic representation of the goals and content of this part of the thesis.

initial temperature, cooling rate, reheating policy, plateau length) that are typically set via *tuning-centric* protocols (e.g., automatic configuration). This is costly, sensitive to the sampled instances and budget, and risks conflating algorithm design with calibration. Building on Franzin and Stützle [182], which shows that SA and Fixed Temperature Simulated Annealing (FTSA) exhibit landscape-dependent performance, we therefore investigate *adaptive, in-run* temperature control.

We now formalize the goals of this part of the thesis in form of Research Questions (RQs):

RQ1 How do alternative tabu list strategies (from fixed to adaptive, short- to long-term) and the inverse-move definition (strict vs. relaxed) affect performance and search behavior when all other components are held fixed? Do they interact?

RQ2 To what extent can we exploit an informed external control to dynamically adapt the

temperarure schedules in SA?

To address **RQ1**, we adopt a component-based perspective on TS and systematically analyse tabu list management strategies while holding all other components fixed (Chapter 5). Using the Permutation Flowshop Scheduling Problem (PFSP) as a case study, we evaluate five mechanisms under two (2) alternative inverse-move definitions (strict vs. relaxed). Our methodology combines large-scale experimental evaluation with behavioral diagnostics, including STNs, to provide both performance-level (black-box) and mechanism-level (white-box) insights. The investigation addressing this RQ is presented in Chapter 5 and refers to and extends the following published work:

- **Da Ros, F.**, & Di Gaspero, L. (2024). An empirical analysis of tabu lists. In *Metaheuristics International Conference, Part II* (pp. 50–64). Cham: Springer International Publishing.

To address **RQ2**, we explore dynamic temperature control for SA in a problem-independent manner. We develop a Hyper-Heuristic-based Simulated Annealing (HHSA), which is a framework in which a selection Hyper-Heuristic (HH) chooses, at run time, a temperature from a discrete set while FTSA acts as the low-level heuristic. The goals are to reduce reliance on heavy tuning and to adapt the exploration–exploitation balance to local landscape characteristics rather than follow a single preset schedule. We evaluate the approach across PFSP, Graph Coloring Problem (GCP), and Traveling Salesperson Problem (TSP). The investigation addressing this RQ is presented in Chapter 6 and refers to the following work:

- **Da Ros, F.**, & Di Gaspero, L., Kletzander, L., Lackner, M. L., Musliu, N., & Schaerf, A. (2025). Dynamic Temperature Control of Simulated Annealing using Hyper-Heuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference* (pp. 184–194). New York, NY: ACM. Conference rank: CORE2023 A.

Figure 4.1 provides an overview of the structure of this part of the thesis. Background is highlighted in blue, while the original contributions of this thesis are marked in orange.

4.2 Structure

This part of the thesis is organized as follows. Chapter 5 analyzes the impact of different tabu list tenures. Chapter 6 investigates dynamic temperature setting for SA.

Chapter 5

Tabu List Strategies

Over the years, dozens of metaheuristic methods have been proposed and applied to thousands of optimization problems [65, 83, 365, 475], resulting in a vast and diverse body of literature. While this richness demonstrates the versatility and impact of metaheuristics, it also raises important challenges: the abundance of approaches makes systematic comparison difficult, and the proliferation of variants has sometimes obscured rather than clarified what drives performance. In practice, navigation of this landscape often reduces to following common choices in related work, limited testing of alternatives, and reliance on individual preferences.

Furthermore, comparative studies frequently report that one algorithm *outperforms* some other ones. For instance, an article might affirm that a multi-neighborhood Simulated Annealing (SA) with a geometric cooling scheme and cut-off mechanism delivers better performance than an Adaptive Large Neighborhood Search (ALNS) with three (3) destroy and two (2) repair operators, yet rarely does the article clarify which components actually drive the observed performance: was it the cooling scheme? The cut-off? The choice of the neighborhoods? Or simply parameter settings? Moreover, comparisons might be influenced by differences in implementation practices and by incomplete reporting of methodological aspects [315, 470].

Only recently the literature has begun to address such component-level questions directly: for example, whether the temperature descent in SA is actually beneficial [182], or whether the adaptive layer of an ALNS genuinely improves performance [470], or what is the role of the tabu list in the overall efficacy of Tabu Search (TS) (as in the case of this chapter).

TS [200] is one of the earliest and most widely applied local search metaheuristics.[1] Over time, numerous variants have been developed, differing in tabu list structures, neighborhood strategies, and aspiration criteria. Yet, despite its success, the contribution of individual components to its effectiveness remains poorly understood, and few systematic conclusions have been drawn regarding their role [126, 235].

Building on the component-based perspective of Franzin and Stützle [183], this paper shifts the focus from viewing TS as a monolithic algorithm to analyzing the contribution of its individual components, with particular attention to the tabu list. Specifically, we:

- Systematically analyze TS in terms of components and evaluate a set of short-term

[1] On 2025–09–20, according to Scopus, 3,724 articles include *tabu search* in the title, and 13,327 in the title, abstract, or keywords.

and long-term tabu list management strategies;

- Study their impact on search dynamics and performance in the Permutation Flowshop Scheduling Problem (PFSP) [175], a benchmark problem well established in the literature and relevant to related domains [494];
- Combine black-box white-box analysis via Search Trajectory Networks (STNs) [370], therefore capturing the dynamics of the search process.

This chapter is structured as follows. Section 5.1 details our methodology. Section 5.2 presents the experimental setup and Section 5.3 reports the results. Section 5.4 concludes the chapter and outlines future research directions.

5.1 Methodology

Our objective is to investigate the role of the tabu list component in shaping the overall efficacy of TS algorithms. Specifically, we aim to assess how different tabu list management strategies influence both solution quality and search dynamics. To this end, we adopt a component-based perspective: the general structure of TS is kept fixed, while the tabu list mechanism is systematically varied. This allows us to isolate and analyze the contribution of short-term, long-term, and dynamic memory strategies within a controlled framework.

We first provide a component-based overview of TS (Section 5.1.1) and then we focus on a set of tabu list strategies from the literature (Section 5.1.2).

5.1.1 Component-based Tabu Search

Tabu Search (TS) is a local search-based metaheuristic introduced by Glover and Laguna [200]. At each iteration, the traditional TS evaluates a subset of the current neighborhood and selects the solution with the best cost function value, even if this entails accepting a worsening move. This mechanism facilitates escaping from local minima but also creates the risk of cycling among previously visited solutions. To address this, TS employs a tabu list that records recently applied moves (or recently visited solutions) and forbids their inverses; prohibitions may nevertheless be lifted if an aspiration criterion is satisfied.

We decompose TS into seven (7) components, of which two (2) are problem-dependent and five (5) are algorithm-specific. A generic outline of the algorithm is reported in Algorithm 5, where each component is highlighted in Smallcaps.

The problem-dependent components are:

- Initial Solution: the procedure used to construct a starting point for the search.
- Neighborhood: the mechanism for generating candidate solutions from the current one; together with the neighborhood, the definition of the Inverse of a move is also required if the Tabu Strategy does not consider prohibitions on solutions.

The problem-independent components, which govern the general behavior of the metaheuristic algorithm, include:

- Termination Criterion: the rule determining termination of the algorithm.

- Exploration Criterion: the strategy for scanning the neighborhood.
- Tabu Strategy: the core mechanism of TS, designed to prevent cycling by forbidding recently applied moves or recently visited solutions (Section 5.1.2).
- Aspiration Criterion: a rule that allows overriding tabu restrictions.
- Acceptance Criterion: the rule for updating the current solution, which may include the acceptance of non-improving moves to facilitate escape from local optima.

Algorithm 5: Component-based formulation of Tabu Search (TS). The components we have identified are written in Smallcaps.

Input: A problem instance, a Neighborhood $\mathcal{N}$, an Initial Solution s_0, control parameters
Output: Best solution found s^*
Initialize the incumbent solution $s \leftarrow s_0$
Initialize the best solution found during the search $s^* \leftarrow s_0$
Initialize the Tabu Strategy considering the control parameters
while *Termination Criterion is not met* **do**
  Choose a move m in the Neighborhood of s based on the Exploration Criterion and accounting for the Tabu Strategy and the Aspiration Criterion
  if m *meets the Acceptance Criterion* **then**
    $s \leftarrow s \oplus m$
    if s *improves over* s^* **then**
      $s^* \leftarrow s$
  Update the Tabu Strategy considering the control paramters
return s^*

5.1.2 Tabu List Strategies

We next describe the tabu list strategies considered in this study. All of them are well-established in the literature and are presented here in their general form, without problem-specific adaptations. For ease of reference, each strategy is denoted as **TS-**, where $-$ is an incremental index starting from 1.

The simplest tabu strategy is a fixed-length list (**TS1**), in which forbidden moves are stored in a list of constant size θ. Each move remains tabu for θ iterations, and the list is updated in a circular fashion. Algorithm 6 describes the procedure for determining whether the current move is considered tabu, whereas Algorithm 7 outlines the update process for TS1.

Algorithm 6: Scan of a the tabu list

Input: Current move m, Tabu list **V**
for *move* t *stored in* **V** **do**
  if Inverse(m,t) **then**
    return True
return False

Algorithm 7: Update of a fixed-length tabu list (TS1).

Input: Current move m, size of the tabu list θ, tabu list $\mathbf{V}$, index of the oldest move o
if *size of* $\mathbf{V}$ *is less than* θ **then**
  Push m to the back of $\mathbf{V}$ // Tabu list is not full yet
else
  Insert m at position o in $\mathbf{V}$ // Overwrite the oldest move
  Update $o \leftarrow (o + 1) \bmod \theta$ // Advance oldest index

A common alternative is a random-length list (**TS2**), introduced by Gendreau, Hertz, and Laporte [194]. Here, each move is assigned a tabu tenure drawn uniformly at random between two predefined bounds, $\theta_{\min}$ and $\theta_{\max}$. Consequently, the tabu list stores each move together with the iteration number at which it is scheduled for removal and is updated as reported in Algorithm 8.

Algorithm 8: Update of a random-length tabu list (TS2).

Input: Current move m, current iteration i, tabu list $\mathbf{V}$, minimum and maximum tabu tenures $\theta_{\min}$ and $\theta_{\max}$
Draw a random tabu tenure $\theta \sim U(\theta_{\min}, \theta_{\max})$
Insert m into $\mathbf{V}$ with removal iteration $i + \theta$
for *move* t *in* $\mathbf{V}$ **do**
  if *removal iteration of* t *equals* i **then**
    Remove t from $\mathbf{V}$ // The move is no longer tabu

A further variant is the cyclic variable-length list (TS3), proposed by É. D. Taillard [454]. In this approach, the tabu list length remains fixed for a given number of iterations, denoted by i_{keeping}. After every i_{keeping} iterations, the list length is updated to a new value selected from a predefined set of candidate lengths, $\mathbf{\Theta} = \theta_1, \ldots, \theta_{\mathcal{J}}$, where $|\mathbf{\Theta}| = \mathcal{J}$. The values in $\mathbf{\Theta}$ are used cyclically to determine the current list length. Algorithm 9 outlines the corresponding update procedure.

Algorithm 9: Update of a cyclic variable-length tabu list (TS3).

Input: Current move m, current iteration i, tabu list $\mathbf{V}$, iterations since last tenure change i_{changed}, current tenure index c, set of possible tenures $\mathbf{\Theta}$, interval i_{keeping} controlling tenure changes
Set current tenure $\theta \leftarrow \mathbf{\Theta}[c]$
Insert m into $\mathbf{V}$ with removal iteration $i + \theta$
for *move* t *in* $\mathbf{V}$ **do**
  if *removal iteration of* t *equals* i **then**
    Remove t from $\mathbf{V}$ // expired move
if $i_{changed} > i_{keeping}$ **then**
  Update $c \leftarrow (c + 1) \bmod |\mathbf{\Theta}|$ // Advance to next tenure
  Reset $i_{\text{changed}} \leftarrow 0$
else
  Increment $i_{\text{changed}} \leftarrow i_{\text{changed}} + 1$

Reactive tabu list strategies (**TS4**) were first introduced by Battiti and Tecchiolli [39].

In these strategies, the length of the prohibition period is dynamically adjusted through feedback mechanisms during the search. Specifically, TS4 employs a classical tabu list with an initial size of 1, which can be increased by a factor ω^+ or decreased by a factor ω^- depending on the observed search behavior. In addition, the algorithm maintains a history of visited solutions, recording when and how often each solution is encountered, to further guide the adjustment process. The update of the data structures proceeds as follows:

- Cycle detection: If a cycle is detected (i.e., a previously visited solution is revisited), the following actions are taken:
  - If the same solution has been repeated more than ξ times, a variable named *chaos* is incremented. When the chaos level exceeds a threshold κ, an escape mechanism is triggered to diversify the search. The escape mechanism empties the solution history and performs a given number of random moves on the current solution.
  - If the same solution has been found within a number μ of iterations, the tabu tenure is increased by factor ω^+. At the same time, the average cycle length is updated according to a moving average rule:

    $$\text{avg}_\ell \leftarrow 0.9 \cdot \text{avg} + 0.1 \cdot \ell,$$

    where ℓ is the cycle length of the solution (number of iterations since the solution was last met).
- Cycle length condition: Otherwise, if the average cycle length is less than the number of iterations since the last time there was a change, the tabu tenure is decreased by factor ω^-.

Algorithm 10 outlines the procedure used to update TS4. In addition to the tabu list $\mathbf{V}$, an auxiliary data structure $\mathbf{H}$ is maintained to record, for each solution encountered, the iteration at which it was last visited and the total number of times it has been encountered.

Another option is to employ a long-term frequency-based strategy (**TS5**), which relies on a transition measure [200, p. 94]. A transition measure records how frequently a given attribute (e.g., a move or swap) occurs across the solutions visited along the search trajectory. Accordingly, TS5 maintains a frequency table that counts how often each move (or its relevant attribute) has been applied. A move is declared *tabu* whenever its relative frequency exceeds a predefined threshold ϕ. Algorithm 11 outlines the procedure used to scan the frequency table, while Algorithm 12 describes how the table is updated.

The literature also proposes alternative strategies that we do not consider here, as they require problem- or instance-specific customization. For example, Blöchliger and Zufferey [53] introduced a Fluctuation-Of-the-Objective (FOO) scheme in which the tabu tenure adapts to variations in the objective function. In this case, the degree of fluctuation is measured as the difference between the minimum and maximum objective values observed within a sliding window, making the tenure instance-dependant.

In our study, tabu status is assigned to the attributes of moves. However, alternative approaches exist in the literature where tabu is defined on the objective values (e.g., fixed-length tabu list applied to the objective value [195]) or even on entire solutions [93]

Algorithm 10: Update of a reactive tabu list (TS4).

Input: Current move m, current solution s, current iteration i, increase and decrease factors for the tabu tenure ω^+ and ω^-, solution history $\mathbf{H}$, tabu list $\mathbf{V}$, iteration threshold ξ, cycle length threshold μ, chaos counter k, chaos threshold κ, current average cycle length avg_ℓ, iterations since last tenure change i_{changed}

```
Insert m into V
Increase i_changed ← i_changed + 1
if s ∉ H then
    Insert s into H with visited times = 1 and last visited iteration = i
else
    Compute cycle length ℓ ← i − last visited iteration of s
    Increment visited times of s
    Update last visited iteration of s to i
    if visited times of s > ξ then
        Increment chaos counter k ← k + 1
        if k > κ then
            Perform a sequence of random moves          // Escape from chaotic region
            Reset memory structures and related counters
            return
    if ℓ < μ then
        Increase the tabu tenure by a factor ω+
        Update the average cycle length avg_ℓ using ℓ
        Reset i_changed ← 0
if i_changed > avg_ℓ then
    Decrease the tabu tenure by a factor ω−
Maintain V at the current tabu tenure length
```

5.2 Experimental Setup

We first report the case study we selected (Section 5.2.1), then we provide some technical details (Section 5.2.2) and references to the tuning process (Section 5.2.3).

5.2.1 Case Study

To investigate the behavior of the tabu list strategies, we use the PFSP. Section 2.2 reports the problem definition, while the remainder of this section discusses both the problem-dependant components and the instances.

Problem-dependant Components

A solution to the PFSP is represented by the permutation of jobs that defines their processing order on the machines. The search process starts from a randomly generated permutation. We adopt the Swap neighborhood, where a move consists of exchanging the positions (p_a and p_b) of two jobs (a and b) within the sequence.

We use two variations of the inverse-move concept. In the first variation, denoted as **IN1**, a move is declared tabu if it involves relocating both jobs a and b. In the second, denoted as **IN2**, a move is tabu if it relocates at least one of the two jobs.

Algorithm 11: Check of the transition-measure table (TS5).

Input: Current move m, current iteration i, frequency table $\mathbf{V}$, threshold ϕ
if $m \in \mathbf{V}$ **then**
  Compute relative frequency $r \leftarrow \frac{\mathbf{V}[m]}{i}$ // Occurrences over total iterations
  if $r > \phi$ **then**
    return True // The move is tabu
return False // The move is admissible

Algorithm 12: Update of a frequency-based tabu list (TS5).

Input: Current move m, frequency table $\mathbf{V}$
if $m \in \mathbf{V}$ **then**
  Increment the counter of m in $\mathbf{V}$; // Increase occurrence count
else
  Create a new entry for m in $\mathbf{V}$ with frequency 1 ; // First occurrence of the move

Figure 5.1 illustrates these concepts through a toy example involving five jobs (indexed from 0 to 4). The figure depicts the solution representation, the swap neighborhood, and the application of the tabu status under IN1 and IN2. The solution is encoded as a permutation of jobs, i.e., an array of length 5 (equal to the number of jobs), whose elements correspond to job indices. In the example, the move takes the job at position $p_a = 0$ (job 1) and swaps it with the job at position $p_b = 2$ (job 0). The inverse types operate as follows: IN1 prohibits subsequent moves that involve the same pair of jobs, i.e., jobs 1 and 2, being moved together again. In contrast, IN2 forbids any move that alters the position of at least one of these two jobs. For example, a move $\langle 0, 0, 3, 2\rangle$ that swaps the positions of job 0 and job 2 would be admissible under IN1, but prohibited under IN2.

Instances

We use both benchmark instances from the literature and newly generated instances.

For the literature benchmarks, we consider the widely used sets by Reeves [406], Heller [217], and E. Taillard [452]. In the Reeves set, only odd-numbered instances are used, since the even-numbered ones are derived these ones. Across these families, the number of jobs ranges from 20 to 200, while the number of machines varies between 5 and 20.

To complement these benchmarks, we generate additional instances using the instance generator[2] originally introduced by Strassl and Musliu [446] in their Instance Space Analysis (ISA) of the Jobshop Scheduling Problem (JSP). We adapt this generator to the PFSP setting, by removing the shuffling of the machine order. In total, we generate 1,000 instances, of which 970 are used for tuning and 30 are set aside for testing (randomly selected). The generated instances cover a broader range of characteristics than those of the literature: the number of jobs varies from 20 to 500, the number of machines from 5 to 40, and the processing times are drawn from diverse distributions, including uniform (1–99 and 1–200), binomial, and negative binomial.

[2] https://github.com/strassl/jssp-instances/releases/tag/v1.0. Accessed 2025–10–16.

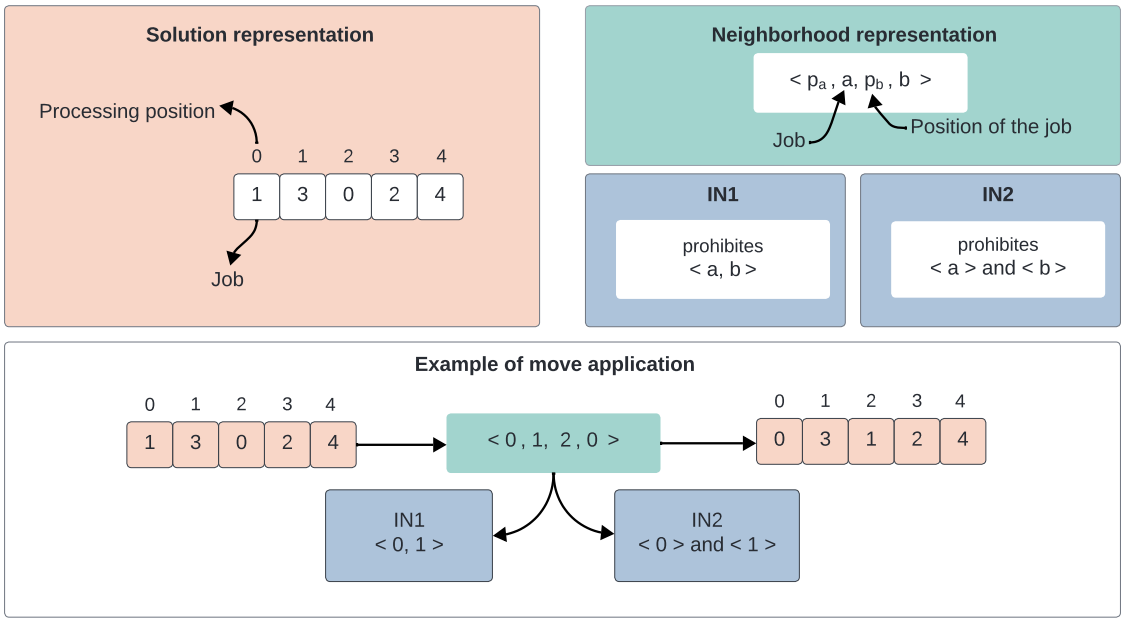


Figure 5.1: Implementation of problem-related components considering a Permutation Flowshop Scheduling Problem (PFSP) instance with 5 jobs (0–4).

In total, 970 instances are used for tuning, while 163 instances form the validation set. Among the latter, 30 are generated instances.

Figure 5.2 shows the instance space of the instances in a two-dimensional Principle Component Analysis (PCA) projection, where tuning, literature, and generated instances are distinguished. Details on the feature extraction and dimensionality reduction are provided in Appendix C.

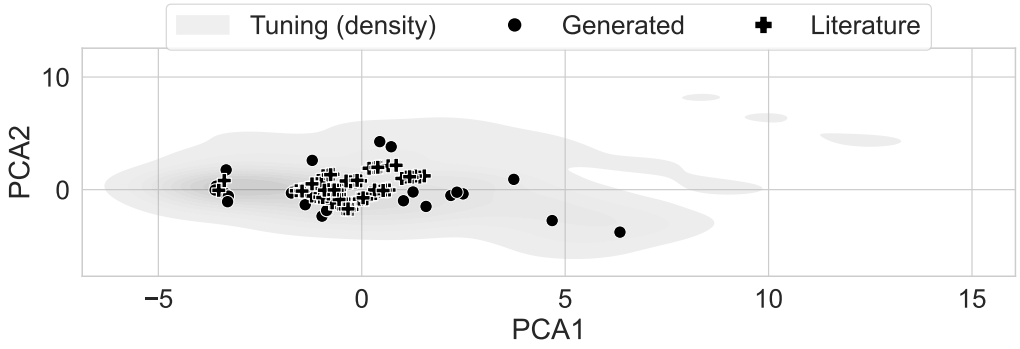


Figure 5.2: Instance space.

5.2.2 Technical Details

All experiments are run on a dual-socket machine equipped with two Intel Xeon Platinum 8368 CPUs (2.4 GHz, 38 cores each) and 512 GB of RAM (8×64 GB RDIMM).

The code is implemented in C++ using the EasyLocal++ v.4 and compiled with `Clang++18`. The use of a software framework [59, p. 143] is essential, as it enables systematic code reuse

and ensures a fair comparison among algorithmic variants: the general implementation of TS relies on the same problem encoding and local search components across all experiments.

In this study, we vary only the tabu list management strategies, while keeping the remaining components of TS fixed:

- The neighborhood is explored exhaustively, such that the best admissible move is always selected from the candidate set.
- For aspiration, we adopt the aspiration by objective rule, whereby a move on the tabu list may be accepted if it yields a solution better than the best found so far.
- The termination criterion is based on stagnation: the search halts after 10,000 consecutive non-improving (idle) iterations (i_{idle}). In addition, we enforce a runtime limit $\mathcal{T}_{\max}$ of 7,200 seconds.

The problem-dependent component, on the other hand, are instead specified in the case study description (Section 5.2.1).

To mitigate stochastic variability, each algorithmic configuration (inverse-tabu list variant) is executed 15 independent times per instance. For a subset of instances (the ones we generate), we also record detailed information about the search trajectory, including: the current solution and its cost, whether the best solution is updated, the type of accepted move, the neighborhood size, and the distribution of moves across categories (improving, worsening, tabu, etc.).

5.2.3 Parameter Tuning

Each of the strategies described in Section 5.1.2 depends on one or more parameters that must be carefully tuned to ensure both fairness and performance. To this end, we employ `Irace` v. 4.2 [316]. We perform independent tuning procedures for every combination of inverse move (IN1 and IN2) and tabu search strategy (TS1 to TS5). Each procedure is allocated 1,000 experiments. Each individual experiment is terminated after 3,600 seconds or when it reaches 10,000 idle iterations.

Table 5.1 summarizes the parameters, while Table 5.2 reports the corresponding search ranges and final values. A dash (—) in the "Range" column indicates that the parameter value was fixed through preliminary experiments.

An additional note is necessary for TS3. The tuning was conducted by setting first the number $\mathcal{J}$ of tabu list tenures, and subsequently their tenure $\{\theta_1, \ldots, \theta_{\mathcal{J}}\}$.

5.3 Results

This section summarizes the main results. First, we analyze performance across the entire benchmark set (Section 5.3.1). We then focus on a subset of instances to study the search behavior (Section 5.3.2) and the corresponding STNs (Section 5.3.3).

For convenience, Table 5.3 reports an overview of the abbreviations used for tabu search strategies and inverse definition.

Table 5.1: Description of the parameters required by each tabu list strategy.

Tabu list	Parameter	Description
TS1	θ	Number of iterations a move is considered tabu
TS2	$\theta_{\min}$	Minimum number of iterations a move is considered tabu
	$\theta_{\max}$	Maximum number of iterations a move is considered tabu
TS3	i_{keeping}	Number of iterations a tabu length is kept
	$\mathcal{J}$	Number of tabu tenures
	$\{\theta_1, \ldots, \theta_{\mathcal{J}}\}$	Lengths of tabu tenures
TS4	ω^-	Decrease factor
	ω^+	Increase factor
	κ	Chaos
	ξ	Iteration threshold
	μ	Maximum cycle length
TS5	ϕ	Frequency of the move to be labeled as tabu
Overall	i_{idle}	Total number of idle iterations.
	$\mathcal{T}_{\max}$	Timeout in seconds.

5.3.1 Performance Analysis

We address the performance comparison of the different tabu list strategies and inverse types. For each combination of instance, tabu list strategy, and inverse type, we compute the minimum and the average of the best cost over 15 runs, and normalize these values by the instance's best cost. Values closer to 1.0 indicate higher-quality solutions.

The distributions are reported in Figure 5.3. The y-axis shows the normalized cost, while the x-axis indicates the tabu list strategy, with the inverse types distinguished by color. Each point indicates one instance, i.e., each boxplot summarizes 163 points. The analysis is complemented by Table 5.4 that reports the number of unique wins and the rank of each tabu list strategy and inverse type. Results are ordered by decreasing number of unique wins. In brackets, we report whether the aggregation across instances is based on the minimum or average cost.

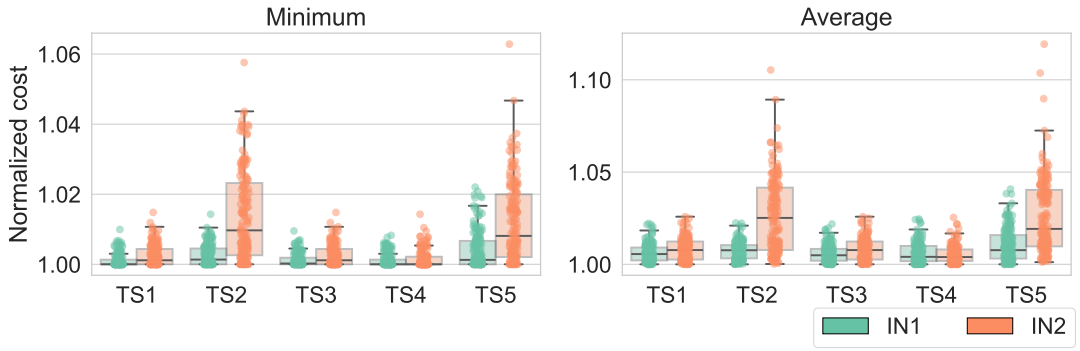


Figure 5.3: Solution quality comparison across tabu list strategies and inverse types. Costs are normalized with respect to the best found, so that values closer to 1.0 indicate better performance.

When focusing on the best performance across runs (minimum cost over 15 runs), TS1-IN1 and TS4-IN1 emerge as the strongest configurations. TS1-IN1 achieves 18 unique wins, while TS4-IN1 dominates in a similar number of cases (16). Overall, IN1 tends to

Table 5.2: Tuning ranges and final parameter values.

Tabu List	Parameter	Range	IN1	IN2
TS1	θ	1–100	30	10
TS2	$\theta_{\min}$	1–100	6	8
	$\theta_{\max}$	1–100	43	56
TS3	i_{keeping}	—	100	100
	$\mathcal{J}$	1–5	5	1
	$\{\theta_1, \ldots, \theta_{\mathcal{J}}\}$	1–100	{11, 34, 20, 8, 98 }	{ 10 }
TS4	ω^-	0.10–0.90	0.37	0.68
	ω^+	1.20–3.00	2.75	2.98
	κ	1–10	9	7
	ξ	1–10	8	5
	μ	1–100	66	96
TS5	ϕ	0.01–0.99	0.01	0.04
Idle iterations	i_{idle}	—	10,000	10,000
Timeout (tuning)	$\mathcal{T}_{\max}$	—	3,600	3,600
Timeout	$\mathcal{T}_{\max}$	—	7,200	7,200

Table 5.3: Nomenclature overview.

Abbr.	Definition	Abbr.	Definition
TS1	Fixed-sized	IN1	Both jobs
TS2	Random-based	IN2	At least one job
TS3	List of sizes		
TS4	Reactive		
TS5	Frequency-based		

outperform IN2, suggesting that the first inverse strategy is more suitable across tabu list variants. The exception is TS4, where the difference between inverses is less pronounced, indicating that its reactive nature makes it less sensitive to the inverse definition.

However, when considering the average cost across runs, the dominance among tabu list strategies becomes less accentuated. This suggests that while some configurations are capable of occasionally reaching the best solutions, their reliability is not consistently superior across repeated runs.

At the other end of the spectrum, TS2 and TS5 combined with IN2 consistently report the worst performance. A possible explanation is that the randomness in tabu tenure length (TS2) interacts poorly with the stricter definition of inverse moves in IN2, leading to weak guidance of the search. Similarly, the frequency-based mechanism in TS5 may reduce the ability of the search to escape local optima when combined with IN2.

We conduct a statistical analysis to validate our findings, considering both the minimum and the average solution quality. Across all instances, the strategies differ significantly in normalized solution quality (Friedman test, p-value < 0.001, with significance level 0.05). Figure 5.4 reports the results of the pairwise statistical comparison, where each entry

Table 5.4: Overview of the number of unique wins and the rank of each tabu list strategy and inverse type. The analysis unique wins considers all 163 instances, using the minimum cost achieved per strategy and inverse type, whereas the average rank reports the results considering the average and the minimum. Results are ordered by decreasing number of unique wins.

Tabu list	Inverse	Unique wins (min)	Avg. rank (min)	Avg. rank (avg)
TS1	IN1	**18**	**3.50** (1)	3.74 (4)
TS4	IN1	16	3.66 (2)	3.62 (3)
TS4	IN2	10	3.80 (4)	**3.13** (1)
TS3	IN1	8	3.75 (3)	3.46 (2)
TS1	IN2	8	4.65 (5)	4.46 (5)
TS2	IN1	5	4.84 (6)	4.46 (5)
TS2	IN2	2	7.72 (9)	7.49 (7)
TS5	IN2	1	7.50 (8)	8.09 (8)
TS5	IN1	0	5.59 (7)	6.55 (6)

corresponds to a p-value. Values highlighted in red denote pairs that, after applying a post-hoc Wilcoxon signed-rank test with Holm's correction (significance level 0.05), are not statistically different. Overall, some tabu list strategies perform similarly, with no significant differences among the leading configurations (TS1, TS4, and TS3 with IN1). In contrast, the weakest performers are consistently TS2-IN2 and TS5 with both inverse strategies, which are significantly outperformed by the best approaches.

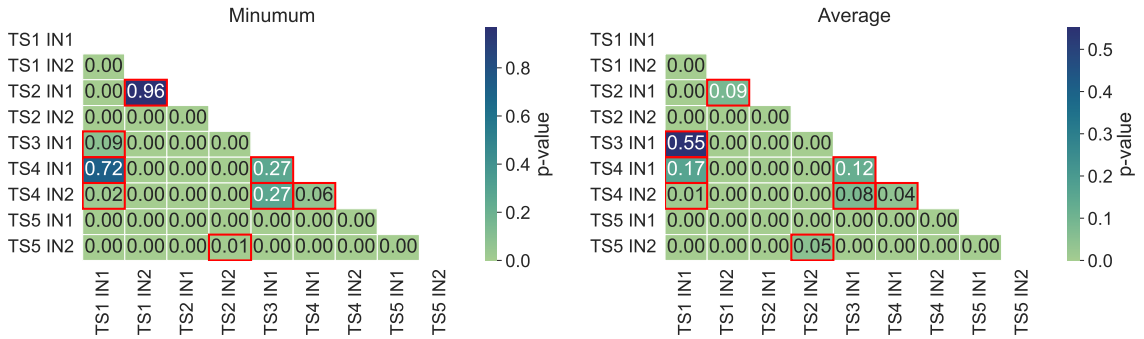


Figure 5.4: Pairwise statistical analysis of tabu search strategies based on normalized solution quality across all instances. Each cell reports the p-value of a post-hoc Wilcoxon signed-rank test (significance level 0.05). Entries highlighted in red denote pairs of strategies that are not statistically different after Holm's correction.

5.3.2 Search Behavior

To analyze the search behavior of the tabu strategies and inverse types, we consider the iterations to stagnation, the search landscape in terms of quality of accepted move w.r.t. the current solution, and the usage of the aspiration criterion.

For these analyses, we focus on the 30 generated instances, as they provide greater diversity within the instance space.

Iteration to Stagnation

We define stagnation as the iteration after which no further improvements of the best solution found (in the current run) are observed. Figure 5.5 reports a total of 30 (instances) $\times$ 15 (seeds) points per boxplot. The y-axis reports the iterations, while the x-axis indicates the tabu list strategy, with the inverse types distinguished by color.

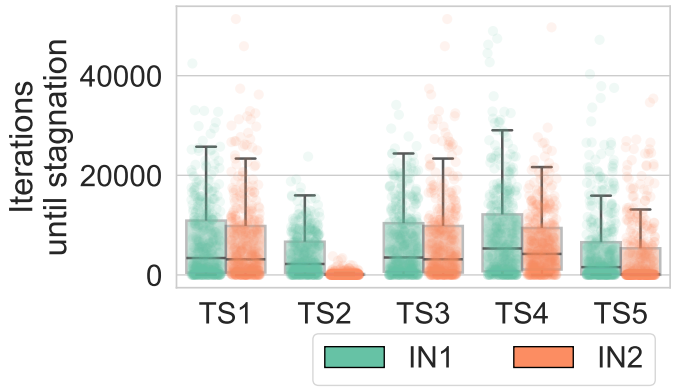


Figure 5.5: Average number of iterations until stagnation.

Regarding the best performing strategies, TS1 and TS3 show similar behavior under IN1 (with IN2, this similarity is due to the parameter settings). In contrast, TS4-IN1 typically stagnates later on average, suggesting that it maintains exploration pressure and continues to generate improving solutions for longer.

The underperforming combinations (e.g., TS2-IN2 and TS5-IN2) stagnate much earlier than the others. A likely explanation is that the randomness in the tabu tenure length of TS2 interacts poorly with the stricter IN2 inverse definition, providing inconsistent guidance and leading the search to exhaust its potential prematurely. Similarly, the frequency-based mechanism of TS5, when coupled with IN2, may become overly restrictive, reducing the search ability to diversify and escape local optima.

Search Landscape

To analyze the search landscape with respect to the prohibition mechanism, we consider the acceptance of different move types during the search. For each run, we compute the proportion of improving, worsening, and neutral moves with respect to the cost of the current solution. These proportions serve as a proxy for the balance between intensification and diversification, as well as plateau navigation. By averaging these proportions across runs and instances, we obtain a behavioral profile for each tabu list strategy and inverse type. This analysis allows us to contrast how the prohibition mechanism shapes the effective exploration of the search landscape and explains differences in performance observed in the aggregate results.

Figure 5.6 shows the distribution of move types across tabu list strategies and inverse definitions. The y-axis reports the proportion of moves in percentage, while the x-axis distinguishes the algorithmic configurations. Colors indicate improving (marine green, bottom proportion), worsening (orange, mid proportion), and neutral (light blue, upper proportion) moves, with bars stacked so that each column sums to 100%.

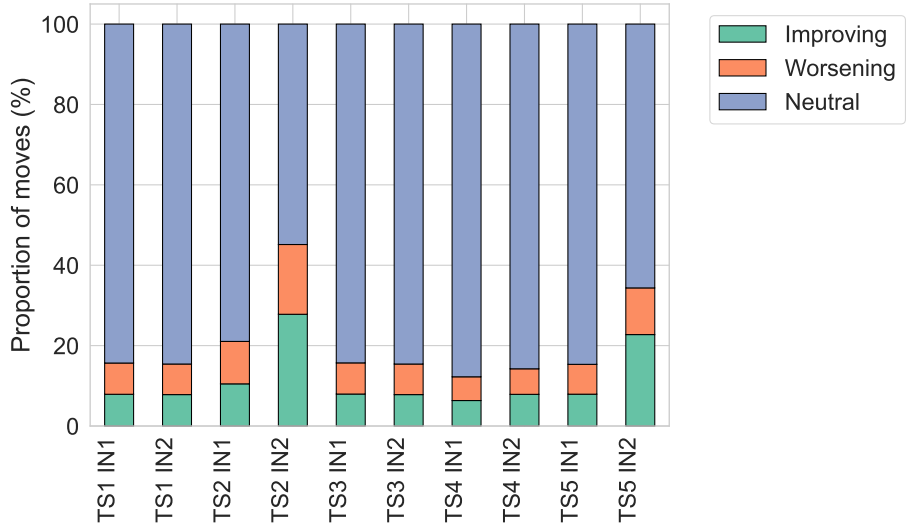


Figure 5.6: Distribution of improving (below, in marine green), worsening (center, in orange), and neutral (upper, in light blue) moves across tabu list strategies and inverse types. Values represent average proportions per instance, aggregated over 15 runs.

Overall, the configurations that achieved the best performance (i.e., TS1, TS3, and TS4) exhibit broadly similar proportions of move types. In contrast, the weaker strategies (TS2–IN2 and TS5–IN2) display a markedly different profile, with disproportionately higher improving and worsening moves and limited capabilities of navigating the neutral areas. This restrictive acceptance behavior is consistent with their shorter effective runs.

Aspiration Criterion Usage

As reported in Section 5.1.1, an aspiration criterion allows for overriding the prohibition mechanism. In our case, a tabu move is accepted if it yields a solution better than the current best, thus relaxing the prohibition mechanism when it is overly restrictive.

Each time the best solution is improved, we record whether the accepted move was admitted through the aspiration criterion. Figure 5.7 summarizes these results: the y-axis indicates the proportion of aspired moves (in %), while the x-axis distinguishes tabu list strategies, with inverse types differentiated by color.

The results reveal clear differences between inverse definitions. IN2 consistently triggers the largest number of aspiration calls (up to 40.6%), indicating that this inverse is overly restrictive and would otherwise risk discarding improving information. By contrast, IN1 relies on aspiration the least, with a maximum of only 12.6% for TS2. Interestingly, TS4 makes comparatively few aspiration calls under both inverses, suggesting that its adaptive tenure mechanism is effective at preventing overly restrictive situations.

5.3.3 Search Trajectories

To study the search dynamics of the analyzed algorithms, we employ Search Trajectory Network (STN), a recently introduced tool for comparing the internal dynamics of algorithms on specific problem instances [370], as described in Section 2.6.6. The implementation of STNs follows five main steps (Figure 5.8):

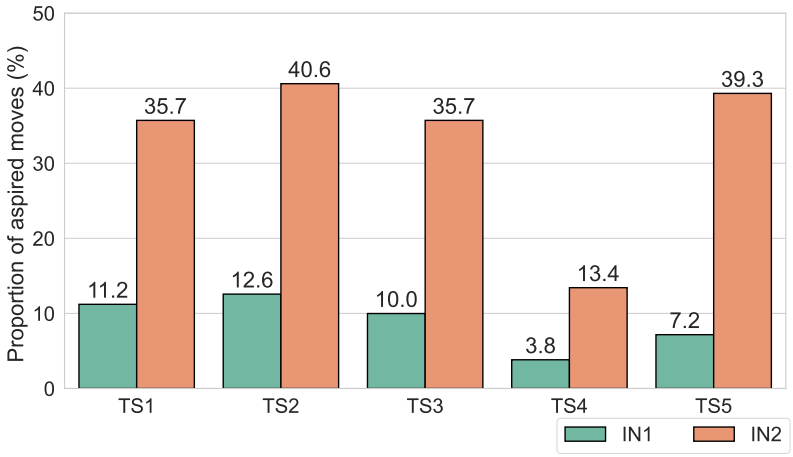


Figure 5.7: Proportion of improving moves (over the best cost so far) accepted via the aspiration criterion. Values are aggregated by average over 15 runs.

1. **Sampling process:** defines how representative solutions are tracked. In our case, we log every improvement in the best solution encountered during the search.
2. **Solution encoding:** defines how solutions are represented. We provide the STN with the array representation described in Section 5.2.1, converting job indices into bitstrings so that all solutions have the same dimension.
3. **Search space partitioning:** defines how representative solutions are mapped to locations. Here, each location corresponds to a unique solution.
4. **STN representation:** defines how the STN graphs are generated. We used the publicly available tool `STNs-v2`.[3]
5. **STN metrics:** defines the metrics used to analyze the trajectories. In our case, we employ the metrics listed in Table 5.5, extracted from the run logs using `Python`.

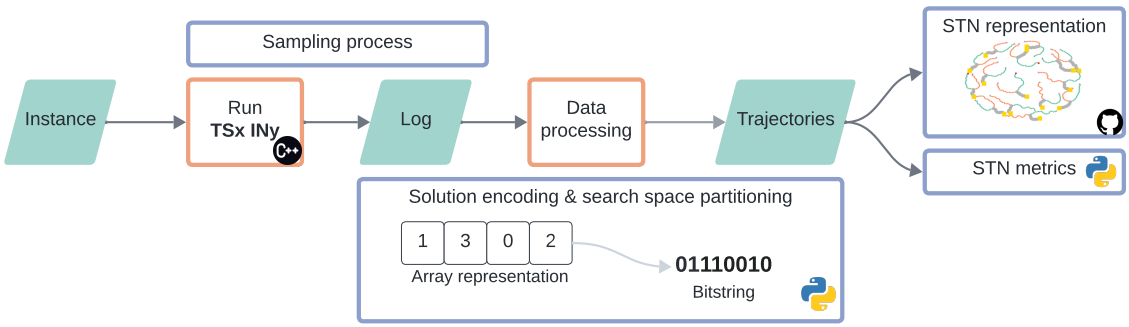


Figure 5.8: Workflow for implementing Search Trajectory Networks (STNs) given an instance.

Since the tool is instance-based and can only analyze one instance at a time, we select a subset of six instances. We report the aggregated values of the STNs metrics by inverse type (Table 5.6) and by tabu strategy (Table 5.7).

[3] https://github.com/gabro8a/STNs-v2, commit: 5eb42b4

Table 5.5: Definition of the Search Trajectory Network (STN) metrics.

Metric	Definition
n	Total number of locations.
n_ψ	Number of locations visited by algorithm ψ.
n^*_ψ	Number of locations visited exclusively by algorithm ψ.
l_ψ	Average trajectory length, i.e., the number of nodes visited by algorithm ψ within one run.
r_ψ	Success rate (%) of algorithm ψ, i.e., the proportion of runs in which algorithm ψ ends in a location with the best cost.

Table 5.6: Aggregated values for Search Trajectory Network (STN) metrics considering the inverse type. Values are averaged across instances and tabu list strategies.

Metric	IN1	IN2	Total
n	—	—	1060.5
n_ψ	734.7	701.9	—
n^*_ψ	358.6	325.8	—
l_ψ	49.0	46.8	—
r_ψ (%)	42.4	35.3	—

For space constraints, Figure 5.9 illustrates the STN graphs for a single representative instance, namely **gen0896**, with results shown separately for each inverse type (Figure 5.9a reports IN1 and Figure 5.9b reports IN2). The left panel presents the STN in the standard Kamada–Kawai layout [249], while the right panel adopts a hierarchical format in which the vertical axis corresponds to the objective value.

For IN1 (Figure 5.9a), most trajectories initially overlap but then diverge as the search progresses, each strategy following its own path once the tabu tenure reaches its full regime. In contrast, for IN2 (Figure 5.9b), the stricter inverse definition causes the majority of trajectories to collapse into a single shared path (shown in gray). The only strategy that maintains additional exploration under IN2 is TS4, which continues to generate distinct steps beyond the common trajectory.

5.4 Conclusions

We first summarize the chapter (Section 5.4.1), then discuss its limitations (Section 5.4.2), and finally outline future research directions (Section 5.4.3).

5.4.1 Content

This study considered a systematic component-based analysis of tabu list strategies in TS, focusing on their role in the PFSP and considering two tabu moves definitions. By combining performance evaluation with behavioral analyses and search trajectory networks, we isolated the contribution of different short-term, long-term, and dynamic memory mechanisms. Our findings highlight several key points:

- **Effectiveness of simple vs. adaptive lists:** Fixed-length lists (TS1) and reactive strategies (TS4) consistently rank among the top performers. This suggests that both

Table 5.7: Aggregated values for Search Trajectory Network (STN) metrics considering the tabu list strategy. Values are averaged across instances and inverse types.

Metric	TS1	TS2	TS3	TS4	TS5	Total
n	—	—	—	—	—	1709.4
n_ψ	752.1	644.9	760.25	745.5	688.7	—
n^*_ψ	116.6	204.0	140.2	347.1	255.5	—
l_ψ	50.1	43.0	50.7	49.7	45.9	—
r_ψ (%)	46.1	33.3	47.2	40.6	21.1	—

straightforward prohibitions and adaptive feedback mechanisms can provide a strong balance between intensification and diversification.

- **Importance of inverse definition:** The stricter inverse type (IN2) often restricts the search excessively, leading to premature stagnation and heavy reliance on aspiration. In contrast, IN1 enables broader exploration and more stable performance across tabu list variants.
- **Weakness of random and frequency-based policies:** Randomized tenures (TS2) and frequency-based prohibitions (TS5) systematically underperform, particularly when combined with IN2. Their search dynamics reveal early stagnation, narrow exploration, and limited use of neutral moves, which reduces their ability to escape local optima.
- **Dynamics captured through STNs:** The trajectory analysis showed that inverse definitions have a strong structural impact: under IN1, strategies diverge and explore distinct regions of the search space, whereas under IN2, trajectories collapse onto a common path, reducing diversification. TS4 is the notable exception, maintaining exploration pressure even under stricter conditions.

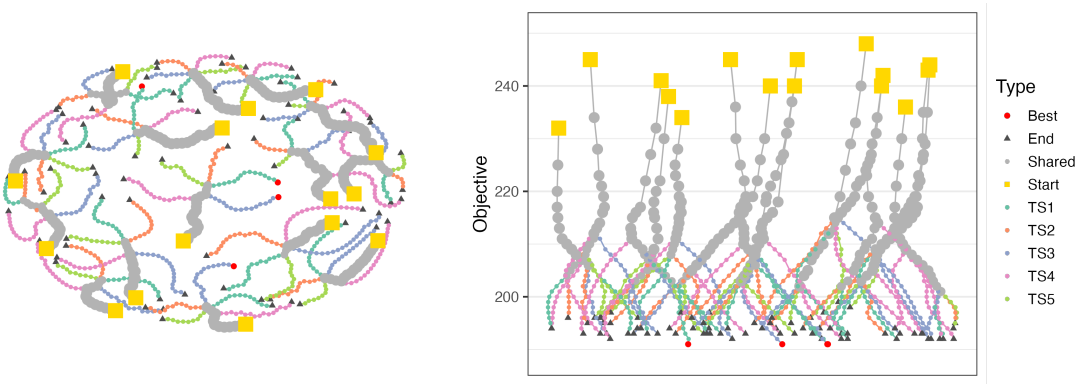
Together, these results strengthen the case for studying metaheuristics at the component level, moving beyond horse-race comparisons of algorithms. They also show that apparently minor design choices, such as how inverse moves are defined, can have a substantial impact on both performance and search dynamics.

5.4.2 Limitations

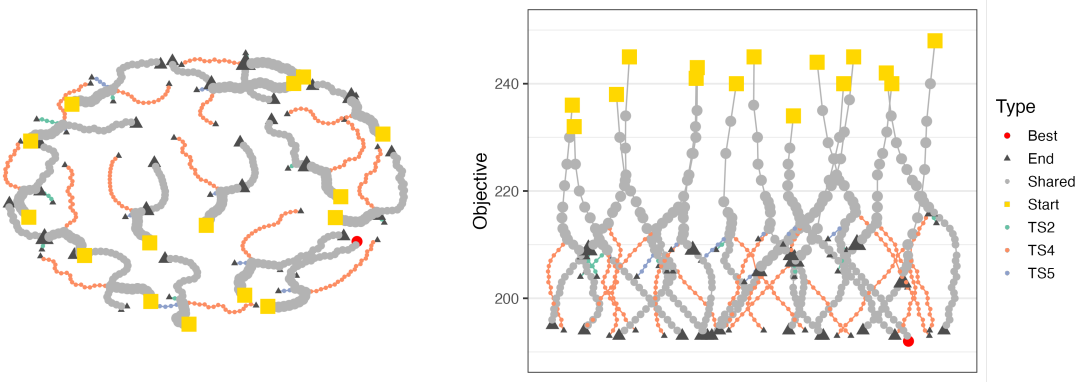
This work presents several limitations. First, the analysis could be extended to additional components of TS, such as neighborhood exploration or move acceptance strategies, to better capture their interplay with tabu list management. Second, the experimental scope is restricted to a single problem (the PFSP) which limits the generalizability of the findings to other problem families and landscape topologies. Finally, although behavioral metrics and search trajectory analysis provided valuable insights, these could be complemented by landscape-based measures (e.g., neutrality, autocorrelation, or evolvability) to deepen the interpretation of observed dynamics.

5.4.3 Future Work

Future research may extend this work in several directions:



(a) IN1



(b) IN2.

Figure 5.9: Search Trajectory Networks (STNs) for instance **gen0896**.

- **Beyond tabu lists:** analyzing other components of Tabu Search (e.g., aspiration, exploration, or acceptance rules) in a similar controlled fashion.
- **Cross-problem generality:** testing whether the observed patterns hold across other problem domains and landscape structures.
- **Integration with automated design:** coupling component-level analyses with automated algorithm configuration to better understand synergies among components.

Chapter 6

Dynamic Temperature Setting of Simulated Annealing

One of the most prominent examples of stochastic local search algorithm is Simulated Annealing (SA), which simulates the annealing process in metallurgy [256]. It relies on a temperature parameter that gradually decreases throughout the search, transitioning from a phase of exploration to one of exploitation.

To enhance diversification and escape local optima, Simulated Annealing (SA) can also employ mechanisms such as reheating or restarts, temporarily increasing the temperature during stagnation. Alternatively, temperature may be held constant throughout the search, leading to the so-called Fixed Temperature Simulated Annealing (FTSA) [113, 348]. Despite its effectiveness, SA remains highly sensitive to parameter settings, particularly the temperature schedule and cut-off, which often require extensive tuning [157, 222, 246]. Moreover, optimal configurations frequently depend on instance-specific characteristics [36, 45], potentially requiring multiple tuning campaigns.

Research has shown that dynamic parameter control often outperforms static ones [168], as different phases of the search process benefit from distinct parameter settings [155]. Dynamically adapting algorithmic behavior based on real-time feedback from the search process is therefore key to improving performance and generalization. Hyper-Heuristics (HHs) offer a promising avenue for such adaptive control. They are high-level, domain-independent strategies that manage a portfolio of Low Level Heuristics (LLHs), aiming to automatically exploit the strengths of each heuristic across diverse scenarios [158, 160].

We investigate whether HHs can be leveraged to dynamically adapt the temperature schedule of SA in a problem-independent manner, thus eliminating the need for manual tuning and prior knowledge about instance characteristics. We introduce a novel approach called Hyper-Heuristic-based Simulated Annealing (HHSA), which treats each FTSA configuration as a LLH, enabling an adaptive controller to switch between different temperature strategies during the search. The contributions of this chapter are as follows:

- **Design of the HHSA framework**: We propose a modular architecture where temperature control is delegated to an adaptive HH strategy managing a portfolio of FTSAs. This design allows the search process to dynamically adjust its behavior across time without tuning, and without being tied to any specific problem domain.

- **Comprehensive evaluation across domains and strategies**: We benchmark HHSA on three optimization problems. We also evaluate three distinct HH strategies: Generic Intelligent Hyper-Heuristic (GIHH), Lean Generic Intelligent Hyper-Heuristic (L-GIHH), and Large State Reinforcement Learning (LSRL).
- **In-depth comparative and ablation analysis**: We assess HHSA against tuned and untuned versions of SA, including a reheating variant. Our analysis includes statistical testing, ablation of individual LLHs, and comparison with a random control baseline. These experiments provide insight into the contribution of adaptive temperature control and the relevance of specific components.
- **Development of a reusable software platform**: We implement HHSA as a new problem domain in the `HyFlex` framework, tightly integrated with the native C++ EasyLocal++ library (Appendix B). This ensures consistency between SA and LLHs, both in terms of data structures and execution model.

This chapter is structured as follows. Section 6.1 details the HHSA framework. Section 6.2 presents the experimental setup and Section 6.3 reports the results. Section 6.4 concludes the chapter and outlines future research directions.

6.1 Methodology

Our goal is to dynamically steer the temperature behavior of the SA process throughout the search. To achieve this, we adopt a control framework grounded in HHs, which operates over a portfolio of LLHs, each implemented as a FTSA. This design enables adaptive control over the trade-off between exploration and exploitation, as the HH can select the most appropriate fixed-temperature strategy in response to the evolving search dynamics. We denote this overall approach as HHSA. Figure 6.1 reports its general scheme.

In the remainder of this section, we first detail the LLHs (Section 6.1.1) and then the HH schemes (Section 6.1.2).

6.1.1 Low Level Heuristics

Each LLH implemented within our framework is instantiated as a FTSA procedure. In this setup, we refer to Algorithm 3 (page 26 Section 2.3.6), where the temperature remains constant throughout the search, i.e., $T = T_0 = T_f$. As a result, operations related to temperature updates and reallocation of unused iterations are omitted. Algorithm 13 reports the pseudocode for a generic FTSA; for consistency, we use the notions introduced to describe SA. Each FTSA executes up to a maximum of N_s iterations. However, as in a standard SA temperature step, the heuristic may terminate earlier if the number of accepted moves exceeds $N_a = \rho N_s$. The parameter ρ defines this acceptance threshold and is dynamically adjusted by the HH based on its control over search intensity, expressed through the *depth of search* parameter.

Since the effect of temperature on move acceptance is influenced by the scale of the cost function, we normalize the objective values (in this case, cost function and objective value correspond). Specifically, $F(s)$ is mapped to the interval $[0, 1]$ for all $s \in \mathcal{S}$, independently of the problem domain. This allows us to adopt a common temperature range,

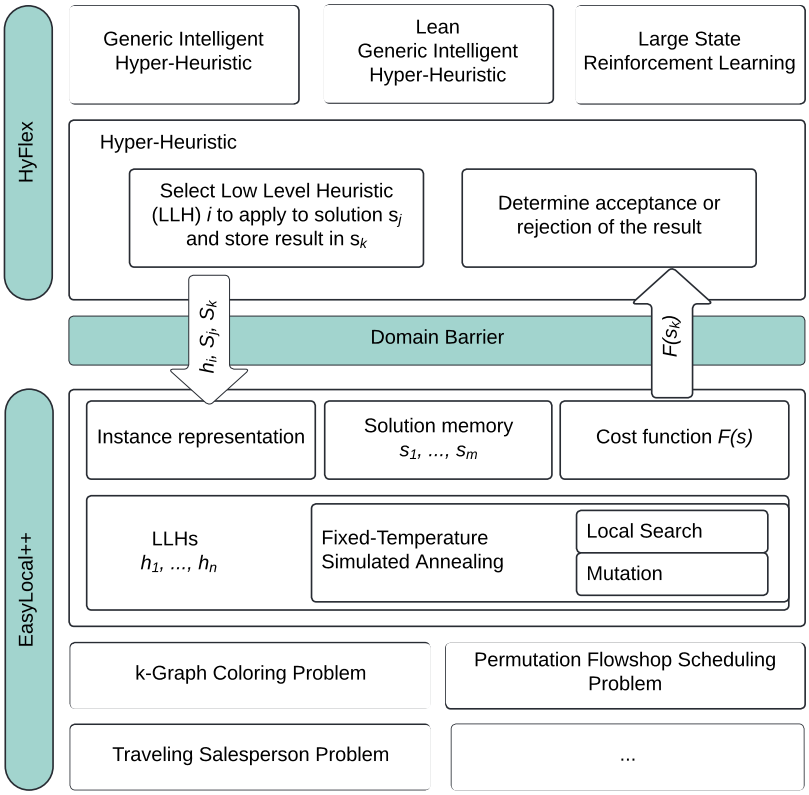


Figure 6.1: Conceptual overview of the Hyper-Heuristic (HH) framework and the proposed Hyper-Heuristic-based Simulated Annealing (HHSA) framework.

i.e., $[0.00001, 1.0]$, across different problems. In this way, temperature settings remain meaningful and comparable without requiring domain-specific tuning.

6.1.2 Hyper-Heuristic Schemes

We compare and evaluate three state-of-the-art HHs:

- **Generic Intelligent Hyper-Heuristic (GIHH)** (also known as AdapHH) [347]: This HH is the winner of the Cross Domain Heuristic Search Challenge 2011 (CHeSC 2011). It operates on a single incumbent solution (with an auxiliary pool for crossover) and dynamically maintains active subsets of LLHs tailored to different search stages. The quality of each LLH is evaluated based on its ability to generate new solutions and to improve solution quality. Under-performing heuristics are subjected to a time-based tabu mechanism and may eventually be excluded permanently. The framework also incorporates relay hybridization, learning effective heuristic sequences through a linear reward-inaction scheme. To improve adaptive exploration, a threshold-acceptance mechanism is used, allowing non-improving moves up to a dynamic threshold that increases if progress stalls; excessive threshold growth triggers a restart. The *depth of search* is continuously adapted through a reward-penalty strategy.
- **Lean Generic Intelligent Hyper-Heuristic (L-GIHH)** [4]: This HH is derived from GIHH via Accidental Complexity Analysis. It streamlines the control structure of

Algorithm 13: Pseudo-code for a Fixed Temperature (FTSA) with cut-off mechanism.

Input: Problem instance, search space $\mathcal{S}$, neighborhood relation $\mathcal{N}$, cost function F, total number of iterations N_s, initial solution s_0, temperature T, neighborhood accepted ratio ρ.
Output: Best solution found s^*

```
// Initialize current solution s and best solution found
s ← s_0
s* ← s
// Initialize counters and parameters
n_s ← 0
n_a ← 0
N_a ← ρ · N_s
// Proceed until the number of iterations or the upper bound to the accepted moves is not exhausted
while n_s < N_s and n_a < N_a do
    // Randomly select one move
    m ← RandomMove(N, s)
    ΔF_m(s) ← F(s ⊕ m) − F(s)
    // Metropolis acceptance criterion
    if ΔF_m(s) ≤ 0 or RandomReal(0, 1) < exp(−ΔF_m(s)/T) then
        s ← s ⊕ m
        // Increment the counter related to the number of accepted moves
        n_a ← n_a + 1
        // Update the best if necessary
        if F(s) < F(s*) then
            s* ← s
    // Increment the iteration counter
    n_s ← n_s + 1
return s*
```

GIHH by removing components that did not significantly contribute to performance, such as phase-based tabu mechanisms, parameter oscillations, and several forms of dynamic parameter adaptation. However, the threshold-acceptance mechanism, including its restart conditions, was retained due to its demonstrated effectiveness. Literature comparisons show that L-GIHH matches or slightly surpasses GIHH in performance while substantially reducing algorithmic complexity.

- **Large State Reinforcement Learning (LSRL)** [259]: This HH formulates the control problem as a Reinforcement Learning (RL) task, inspired by earlier approaches such as those reported by Mischek and Musliu [346]. Here, the HH acts as an agent, heuristic selection is treated as an action, the problem instance serves as the environment, and rewards are computed based on improvements in objective value and runtime. LSRL employs a state representation built from 15 features, capturing aspects of the current search state and recent heuristic performance, including their success rates. It operates on a single incumbent solution, aside from a pool for crossover, and utilizes solution chains guided by the Luby sequence, i.e., frequent short heuristic chains and rarer long ones, accepting any improving move, and reverting to the best previous solution if necessary. Tile coding is used for approximating the value function, and exploration is managed through an ϵ-greedy policy adapted with ideas

from Iterated Local Search (ILS). Restarts are triggered when no improvement occurs over a specified window. LSRL has shown competitive performance compared to GIHH and L-GIHH across several real-world domains [258].

The choice to focus on these HHs is motivated by their demonstrated success across diverse problem domains, and by the fact that they do not impose rigid, phase-based ordering of LLHs as many ILS-based HHs do.

6.2 Experimental Setup

We standardize our terminology as follows. The fine-tuned SA is denoted SA_t, and its untuned baseline SA_{nt}. When equipped with reheating, we write SAR for the tuned variant and SAR_{nt} for the untuned one. We use HHSA for the general HH-based framework, and we refer to specific HHs by name (GIHH, L-GIHH, and LSRL). A selector that chooses the next LLH uniformly at random is denoted HH_r.

We now detail the case studies (Section 6.2.1), algorithmic configurations (Section 6.2.2), implementation and hardware setup (Section 6.2.3), and experimental plan (Section 6.2.4).

6.2.1 Case Studies

To characterize the behavior of HHSA across various domains of combinatorial optimization, we consider three well-established benchmark problems, namely k-Graph Coloring Problem (k-GCP), Permutation Flowshop Scheduling Problem (PFSP), and Traveling Salesperson Problem (TSP). These problems are selected for their diversity in domain, structure, and search landscape. Their implementations either represent the current state-of-the-art or employ neighborhoods known to be highly effective. It is important to note that our goal is not to claim superiority of HHSA or SA over specialized solvers for these problems. Rather, our focus is on assessing the comparative performance of HHSA against SA variants. While problem definitions appear in Section 2.2, in this section we only detail the specific problem-dependant components.

k-Graph Coloring Problem

A solution to the k-GCP is encoded as an array, where each index represents a node and the corresponding value denotes the assigned color. The search begins from a randomly generated solution and explores the neighborhood using the Recolor operator, which selects a conflicting node and assigns it a new color. The cost is normalized by dividing the number of conflicting edges by the total number of edges in the graph.

Permutation Flowshop Scheduling Problem

Solutions to the PFSP are encoded as permutations, i.e., the job execution order. The search process begins with a random permutation and explores the neighborhood using the Swap operator, which exchanges the positions of two jobs in the schedule. The cost is normalized by dividing the makespan by the sum of the longest processing times per machine.

Traveling Salesperson Problem

Solutions to the TSP are encoded as permutations representing the visit order of the cities, i.e., a tour. The search begins from a randomly generated tour and explores the neighborhood using the classical 2-Opt operator [117]. This move removes two edges from the tour and reconnects the resulting segments by reversing the subsequence between the removed edges. The cost is normalized by dividing the total travel distance by the sum of the maximum outgoing edge weights from each node.

6.2.2 Algorithm Configurations

We now report details related to the algorithmic configuration of both HHSA and SA.

Hyper-Heuristic-based Simulated Annealing Configuration

The FTSA used within the HHSA framework operates without requiring any parameter tuning. For its configuration, we define five fixed temperatures linearly spaced across the interval $[0.00001, 1]$, specifically $\{0.00001, 0.25,0.5, 0.75, 1\}$. This results in a total of 10 LLHs, comprising five mutation-based (MU) and five local search (LS) operators, each associated with a different temperature level. Table 6.1 reports the acceptance probabilities at different levels of T. Each LLH is executed for a maximum of 1,000 iterations (N_s), consistent with previous work in the HH literature [258]. The cut-off ratio parameter ρ is initially set to 0.2 and may be dynamically adjusted by the HH via the depth of search mechanism, with an upper limit of 0.4. This cut-off mechanism allows early termination of the current temperature level if a sufficient number of moves (ρN_s) have already been accepted, thereby reallocating effort to more promising regions of the search space.

Table 6.1: Acceptance probability $p = \exp(-\Delta F/T)$ for selected ΔF and temperatures T used in Hyper-Heuristic-based Simulated Annealing (HHSA).

$T \setminus \Delta$	0.001	0.01	0.05	0.10	0.20	0.50	0.75	1.00
0.00001	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
0.25	99.6%	96.1%	81.9%	67.0%	44.9%	13.5%	5.0%	1.8%
0.50	99.8%	98.0%	90.5%	81.9%	67.0%	36.8%	22.3%	13.5%
0.75	99.9%	98.7%	93.6%	87.5%	76.6%	51.3%	36.8%	26.4%
1.00	99.9%	99.0%	95.1%	90.5%	81.9%	60.7%	47.2%	36.8%

Simulated Annealing Configuration

SA, both with and without reheating, depends on a broad set of parameters (Table 6.2) that require careful tuning. For this purpose, we employed `Irace` v.4.2.0 [316]. Each tuning run was allocated a budget of 8,000 experiments, with the running time per instance scaled according to the instance size (Section 6.2.4 reports a how this scaling is conducted).

We also consider a non-tuned version of SA (with and without rehearting, SAR_{nt} and SA_{nt}, respectively), where the parameters are set to reasonable default values within the selected ranges. The purpose is twofold: *(i)* we aim to validate the necessity of the costly tuning process; *(ii)* non-tuned represents the opposite end of the configuration spectrum, where no parameter optimization is applied, serving as an upper bound for comparison.

Table 6.2: Simulated Annealing (SA) parameters description. We indicate with ✓ in the "SAR" column those paramters that are used just within the reheating version of SA.

Param.	SAR	Description
T_0		The initial temperature.
T_f		The final temperature at which the search terminates, in the case of SA, or at which reheating is required, in the case of SAR.
α		Cooling rate that determines how quickly the temperature decreases during the search.
ρ		Neighbors accepted ratio, which is the target ratio of accepted neighbor solutions to evaluated neighbors.
$R_{\max}$	✓	Maximum number of reheats.
T_r	✓	Temperature reheat ration as proportion of T_0, so that $T_r \in [0, 1]$.
β	✓	Proportion of the run spent on the first temperature decrease.

Table 6.3: Parameters of Simulated Annealing without reheating.

		SA_t			
Parameter	Range	k-GCP	PFSP	TSP	SA_{nt}
T_0	0.20000–1.00000	0.93099	0.32089	0.39204	1.00000
T_f	0.00001–0.00010	0.00002	0.00001	0.00001	0.00001
α	0.95000–0.99000	0.97645	0.98149	0.97813	0.95000
ρ	0.20000–0.40000	0.21137	0.23285	0.36624	0.20000
exp	8,000	7,994	7,994	7,998	—

Table 6.3 reports the parameter ranges and selected values for SA_t, together with the corresponding SA_{nt} settings. Similarly, Table 6.4 shows the parameters for SAR and its untuned counterpart SAR_{nt}. The ranges are comparable to those used for HHSA. For completeness, we indicate the tuning budget allocated to each procedure ("exp").

Table 6.4: Parameters of Simulated Annealing with reheating.

		SAR			
Parameter	Range	k-GCP	PFSP	TSP	SAR_{nt}
T_0	0.2–1.0	0.63444	0.7303	0.35042	1.0
T_f	0.00001–0.00010	0.00001	0.00001	0.00001	0.00001
α	0.95–0.99	0.95838	0.9894	0.35042	0.95
ρ	0.2–0.4	0.31015	0.24387	0.2293	0.2
$R_{\max}$	1–5	1	3	1	3
T_r	0.05–1.0	1.0	0.3	0.1	0.10
β	0.10–0.50	0.2	0.3	0.10	0.50
exp	8,000	7,991	7,993	7,991	—

6.2.3 Technical Details

The experiments were run on a machine equipped with 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8×64GB RDIMM. The problem-related specifications, LLHs, and SA were implemented in C++ using EasyLocal++ v.3. The code was compiled with `Clang++`.

6.2.4 Experimental Plan

All experiments are conducted on publicly available datasets. To maintain fairness, each algorithm was evaluated on a common timeout per instance, where the timeout was scaled proportionally to the instance size. To activate the cut-off (and therefore allow some sort of temperature adaptation based on how the run proceeds), we set a maximum number of iterations based on preliminary experiments (i.e., average number of iterations a SA can perform in the given amount of time). For each benchmark problem, we randomly selected 50 test instances and 20 tuning instances. It is important to emphasize that the tuning phase was applied exclusively to SA with and without reheating, whereas HHSA is designed to function without explicit tuning (Section 6.2.2). Table 6.5 summarizes the datasets and their references, the source code used, the formulas applied to scale the timeouts, and the resulting average timeout durations (expressed in seconds).

Table 6.5: Instances and timeout details.

		Timeout	
Case study	Instances	Formula	Time [s]
k-GCP	dimacs[a]	$0.0075 \cdot colors \cdot nodes$	113.17
PFSP	[452]	$0.0075 \cdot jobs \cdot machines$ [183]	8.15
TSP	[408]	$0.0075 \cdot nodes$	22.34

[a] https://mat.tepper.cmu.edu/COLOR/instances.html. Accessed 2025–01–29.

For each algorithm and instance, we perform 10 independent runs using different random seeds to account for stochastic variability. The comparison is based on the best objective values (all these problems are minimization problems, so the lower the better) and considering of the average gap. In the case the denominator is zero, then we adjust it so that $\text{gap}(s) = (F(s) - F(s_b)) \cdot 100$, with s being the solution under analysis, F the cost function, and s_b the reference solution.

We then perform a ranking-based evaluation, where algorithms are ranked according to their performance, assigning rank 1 to the best-performing method, rank 2 to the second-best, and so on (i.e., the lower the better). Ties are handled by assigning the average of the ranks that would have been assigned to the tied methods. Finally, we use also statistical tests.

6.3 Results

In this section, we first analyze the performance of HHSA w.r.t. SA (Section 6.3.1) and then we investigate HHSA building blocks (Section 6.3.2).

6.3.1 Benchmarking Hyper-Heuristic-based Simulated Annealing

To benchmark the performance of HHSA, we compare it against two baselines: SA_t and SAR. Our evaluation focuses on the average gap relative to the best solution found across 10 independent runs.

Figure 6.2 summarizes the results, separated by problem domain, HH, and baseline. The upper section of the figure shows comparisons against SA_t, while the lower section refers

to SAR. The left portion displays results for the k-GCP, the center for the PFSP, and the right for the TSP. The x-axis indicates the specific HHSA variant under evaluation, and the y-axis reports the average objective gap. Note that the scale of the y-axis varies by problem domain to accommodate the different magnitudes of the gaps.

In both the k-GCP and PFSP domains, the performance gap between HHSA and SA_t is relatively small, particularly when compared to the TSP. For k-GCP, both SA_t and SAR slightly outperform the HHSA variants, although the average gap remains under 1%, and the differences are less marked in the case of SAR. In PFSP, the average gap is close to zero across all methods, making it difficult to clearly distinguish which approach performs best. Conversely, in the TSP domain, all HHSA variants consistently outperform both SA baselines, highlighting the advantage of adaptive temperature selection in this setting.

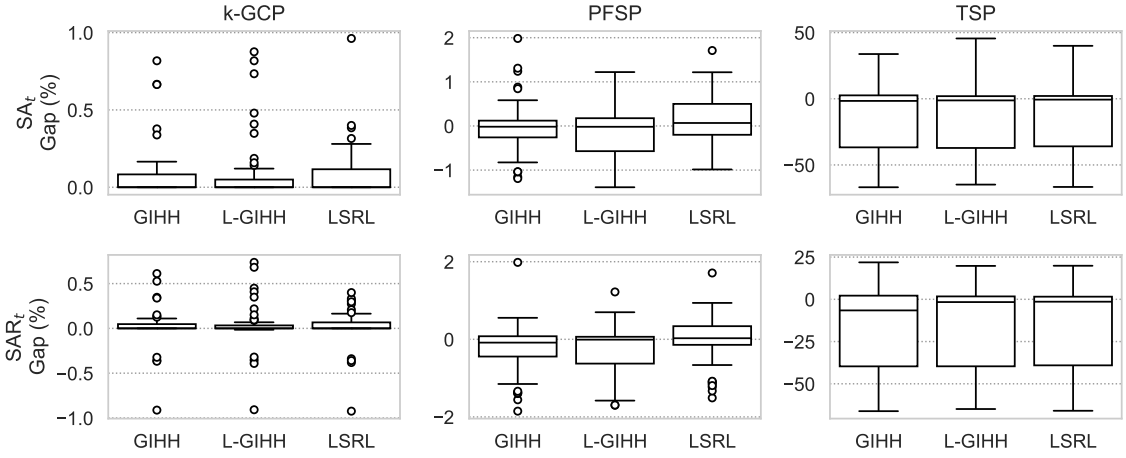


Figure 6.2: Gap (%) between any Hyper-Heuristic-based Simulated Annealing (HHSA) and the tuned version of Simulated Annealing (SA_t) or the tuned version of Simulated Annealing with reheating (SAR). SA_t and SAR are used as baselines.

To support our observations, we further analyze the results by ranking the methods and applying statistical significance tests, as reported in Table 6.6. The left part of the table reports the comparison w.r.t. SA_t, whereas the right part regards SAR. The analysis is organized by problem domain (*Prob.*) and algorithm (*Alg.*). The algorithm marked with $\star$ serves as the baseline for both the *Gap* calculation and the statistical comparison. The *Sign.* column indicates whether the difference is statistically significant (✓: yes, ✗: no), with the better-performing algorithm shown in parentheses. Out of the 18 statistical tests conducted to compare HHSA variants with SA baselines, results show that in 6 cases (33%) the performance is statistically indistinguishable. In 8 cases (44%), HHSA significantly outperforms the corresponding SA variant, while in the remaining 4 cases (22%), the SA variants yield better results. Table 6.7 complements this analysis by reporting the rankings across all five algorithms (i.e., both SA variants and the three HHSA methods). For both PFSP and TSP, the top two ranked methods belong to the HHSA family. Conversely, as in earlier observations, SA variants exhibit better performance in the case of k-GCP.

To further validate the adaptability of HHSA, we compare its performance against SA_{nt} and SAR_{nt} — two baseline variants configured without parameter tuning. This analysis serves to illustrate the potential benefits of using HHSA in scenarios where tuning is impractical or unavailable. The results, summarized in Table 6.8, show that in most cases,

Table 6.6: Comparison between SA_t (left) and SAR (right) against GIHH, L-GIHH, and LSRL.

Prob.	Alg.	Rank	Gap	p-value	Sign.	Alg.	Rank	Gap	p-value	Sign.
k-GCP	SA_t ⋆	1.90	—	—	—	SAR ⋆	2.16	—	—	—
	GIHH	2.59	0.083	<0.001	✓ (SA)	GIHH	2.52	0.018	0.114	✗
	L-GIHH	2.72	0.094	<0.001	✓ (SA)	L-GIHH	2.60	0.029	0.126	✗
	LSRL	2.79	0.082	<0.001	✓ (SA)	LSRL	2.72	0.017	0.159	✗
PFSP	SA_t ⋆	2.54	—	—	—	SAR ⋆	2.58	—	—	—
	GIHH	2.19	-0.061	0.232	✗	GIHH	2.20	-0.236	0.031	✓ (HHSA)
	L-GIHH	2.03	-0.109	0.291	✗	L-GIHH	2.01	-0.284	0.024	✓ (HHSA)
	LSRL	3.24	0.144	0.019	✓ (SA)	LSRL	3.21	-0.031	0.352	✗
TSP	SA_t ⋆	2.69	—	—	—	SAR ⋆	2.69	—	—	—
	GIHH	2.44	-16.034	0.015	✓ (HHSA)	GIHH	2.42	-17.798	0.003	✓ (HHSA)
	L-GIHH	2.51	-15.105	0.016	✓ (HHSA)	L-GIHH	2.51	-17.119	0.004	✓ (HHSA)
	LSRL	2.36	-15.564	0.018	✓ (HHSA)	LSRL	2.38	-17.527	0.005	✓ (HHSA)

Table 6.7: Average rank of each algorithm across the three problem domains. Lower ranks indicate better performance.

Alg.	k-GCP	PFSP	TSP
SA_t	2.35	2.83	3.18
SAR	2.71	3.29	3.20
GIHH	3.22	2.60	2.87
L-GIHH	3.30	2.43	2.92
LSRL	3.42	3.85	2.83

HHSA significantly outperforms the untuned SA variants. The main exception is observed in the k-GCP domain, where the performance is comparable. This is explained by the fact that the default parameters used in SA_{nt} closely resemble the configuration found through tuning. Overall, in 3 out of 18 cases (17%), the untuned SA variants outperform HHSA; in 5 cases (28%), their performance is statistically equivalent; and in the remaining 10 cases (56%), HHSA variants deliver superior results.

Table 6.8: Comparison between SA_{nt} (left) and SAR_{nt} (right) against GIHH, L-GIHH, and LSRL.

Prob.	Alg.	Rank	Gap	p-value	Sign.	Alg.	Rank	Gap	p-value	Sign.
k-GCP	SA_{nt} ⋆	1.93	—	—	—	SAR_{nt} ⋆	2.21	—	—	—
	GIHH	2.59	0.084	<0.001	✓ (SA)	GIHH	2.49	0.013	0.231	✗
	L-GIHH	2.7	0.096	<0.001	✓ (SA)	L-GIHH	2.58	0.025	0.199	✗
	LSRL	2.78	0.084	<0.001	✓ (SA)	LSRL	2.72	0.013	0.149	✗
PFSP	SA_{nt} ⋆	3.01	—	—	—	SAR_{nt} ⋆	3.19	—	—	—
	GIHH	2.04	-0.315	<0.001	✓ (HHSA)	GIHH	1.99	-0.414	<0.001	✓ (HHSA)
	L-GIHH	1.86	-0.363	<0.001	✓ (HHSA)	L-GIHH	1.82	-0.462	<0.001	✓ (HHSA)
	LSRL	3.09	-0.11	0.632	✗	LSRL	3.0	-0.21	0.137	✗
TSP	SA_{nt} ⋆	2.65	—	—	—	SAR_{nt} ⋆	2.87	—	—	—
	GIHH	2.48	-16.392	0.015	✓ (HHSA)	GIHH	2.36	-19.621	<0.001	✓ (HHSA)
	L-GIHH	2.49	-15.29	0.009	✓ (HHSA)	L-GIHH	2.45	-18.808	0.001	✓ (HHSA)
	LSRL	2.38	-15.774	0.015	✓ (HHSA)	LSRL	2.32	-19.228	0.002	✓ (HHSA)

6.3.2 Components Evaluation

We analyze the role of the LLHs by tracking the calls made by each HH to each LLH across all runs and instances. We then study frequency, combination, and importance, and compare the HH-based approach to a random one. A general observation is that the gap measures may appear close to zero in several cases. However, one must also consider their statistical significance, as the observed gaps are likely influenced by the adaptive nature of the algorithms.

Relative Frequencies

Figure 6.3 shows the relative frequency of each LLH applied during the search, broken down by problem domain, HH, and LLH type. The y-axis reports the percentage of times each LLH is selected, with scale ranges varying across problem domains to reflect differences in distribution. Across all problems, the most frequently selected LLHs are those operating at low temperatures and using the LS mechanism. However, depending on the characteristics of the instance and the underlying landscape, HHSA may also utilize MU or higher-temperature LLHs. For instance, in the k-GCP, which features a landscape dominated by large plateaus [331], all HHSA variants make notable use of MU strategies and higher temperatures to enhance exploration. Conversely, in the case of TSP, where improving moves are more abundant, pure exploitation through low-temperature LS operators is generally sufficient. A consistent pattern observed across all HHs is the dominant use of FTSA operators configured with the lowest temperature, suggesting a strong bias toward intensification. Higher-temperature operators are used much less frequently, and their usage does not reveal any consistent preference across domains. When comparing the behavior of the different HHs, we note that LSRL employs MU operators less frequently than both GIHH and L-GIHH.

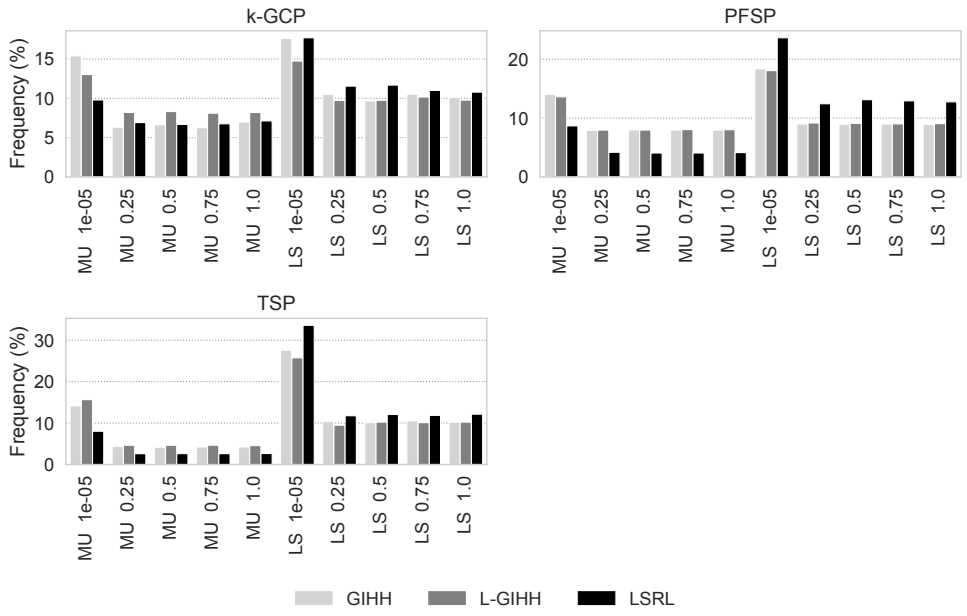


Figure 6.3: Frequency (%) of Low Level Heuristics (LLHs) per type, i.e., Mutation (MU) and Local Search (LS), and temperature

Although the aggregated data suggests that the HHSA variants tend to apply LLHs in a similar manner, this trend does not consistently hold at the instance level. Due to space limitations, we do not analyze each instance individually; instead, we present a representative example. Figure 6.4 illustrates the temporal distribution of LLHs calls for a specific TSP instance (`fl1577`), showing how the selected temperature levels evolve over the course of the search. To ensure comparability across runs, completion is expressed as a percentage of the total runtime, with each LLH call mapped to its relative point in the execution timeline. The figure reveals that higher-temperature operators are more frequently applied during the early phases of the search. In this example, GIHH intermittently returns to higher temperatures throughout the run, whereas L-GIHH abandons them earlier. Although no explicit restarts were triggered for this instance, each HH demonstrates a distinct dynamic, occasionally reintroducing higher temperatures in short bursts that resemble implicit reheating phases.

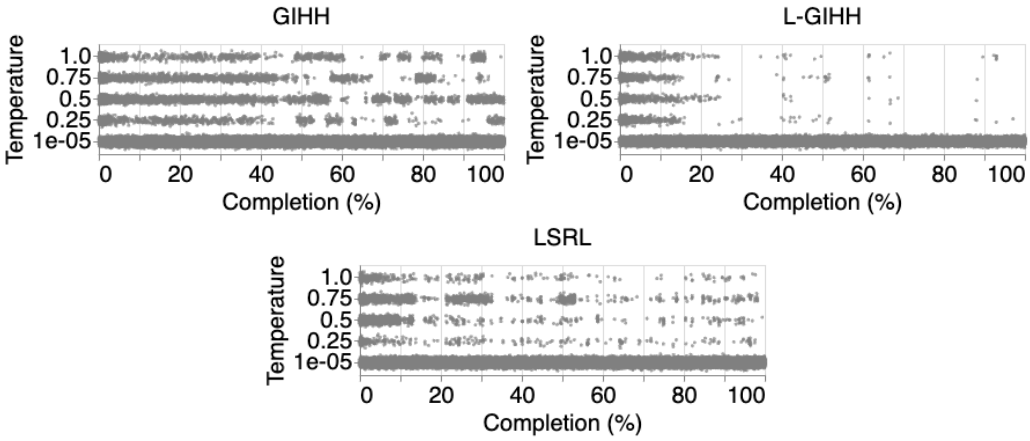


Figure 6.4: Example of temperature scheme for a Traveling Salesperson Problem (TSP) instance.

Combination of Low-Level Heuristics

In adaptive HHs, the effectiveness of the search process is shaped not only by the selection of individual LLHs, but also by the sequence in which they are applied. These sequences define the overarching application policy and can reveal how the controller adapts over time. While frequency analysis offers insight into the general usage patterns of HHSA, it is important to recognize that LLHs are often used in structured, temporally-dependent combinations rather than in isolation.

To explore these interaction patterns, we analyze bigrams—pairs of consecutive LLHs applications—capturing the local transitions between heuristics during search. Figure 6.5 presents bigram frequencies for each problem domain and HHSA variant. Each cell represents the relative frequency with which a specific LLH (next) follows another (previous).

Across all domains, the most pronounced pattern is the frequent repeated use of the LS LLH at the lowest temperature, as reflected by the dark square in the bottom-right corner. Both GIHH and L-GIHH tend to alternate between MU and LS operators at the lowest temperature, whereas LSRL shows a preference for switching among different LS heuristics. Diagonal patterns—indicative of repeated use of the same LLH—are especially visible in k-GCP and TSP, suggesting stronger short-term heuristic commitment in these

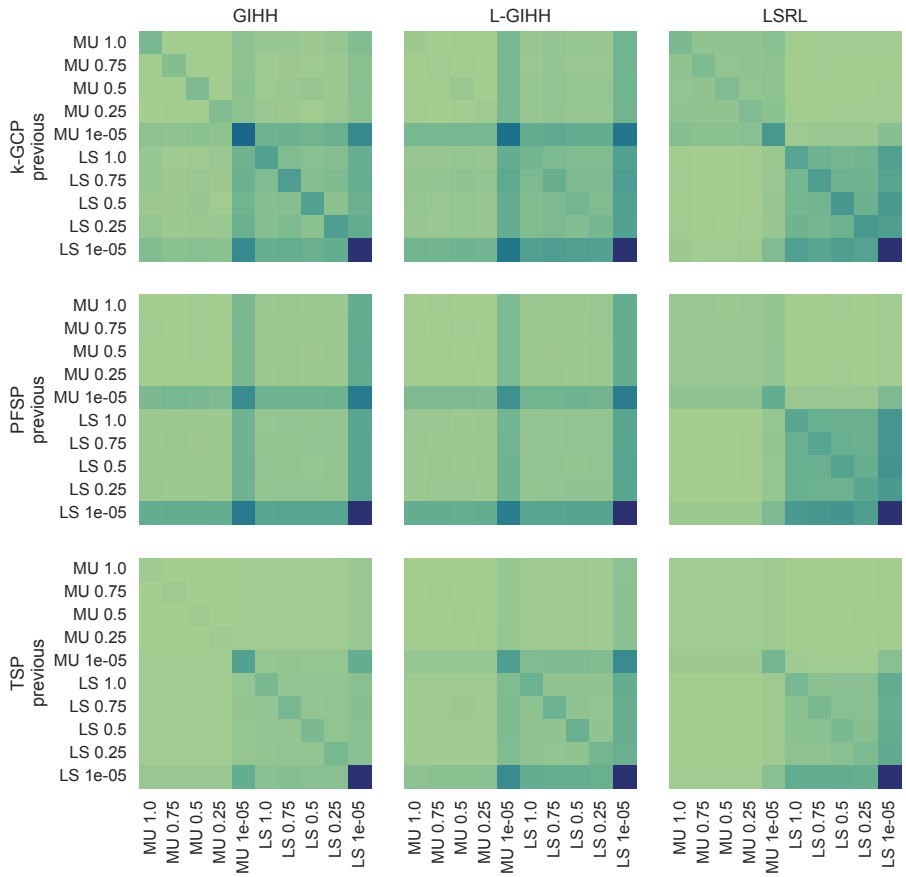


Figure 6.5: Transition patterns between Low Level Heuristics (LLHs).

domains. Although similar transition structures are observed across domains, their relative intensities vary, reflecting domain-specific differences in HH behavior.

Importance of Low-Level Heuristics

The previous sections examined how frequently different LLHs are selected by each HH and the patterns in which they are combined. However, low selection frequency does not necessarily indicate that a given LLH lacks importance—its occasional use may still yield a significant impact on solution quality. This raises two key questions: how does HHSA perform when using only LS LLHs, given that MU operators are rarely selected, and to what extent does each fixed temperature contribute to performance? To address these questions, we perform an ablation study in which all MU LLHs are removed from the portfolio. This enables us to isolate their contribution and assess their influence on search dynamics and final solution quality. It is worth noting that well-designed HHs should be capable of autonomously identifying and down-weighting underperforming heuristics, while also learning to balance the use of LS and MU LLHs effectively through their adaptive control mechanisms.

Table 6.9 presents the ablation analysis concerning the less frequently used MU LLHs. In most cases, both types of LLHs, i.e., MU and LS, are necessary for optimal performance: in 6 out of 9 comparisons (67%), the removal of MU leads to a statistically significant

performance degradation. In two cases within the TSP domain and one in PFSP, no statistically significant difference is observed, indicating that the exclusion of MU LLHs does not negatively affect performance. This sult aligns with the earlier frequency analysis, which showed that MU operators are used sparingly in TSP, suggesting their limited contribution in that domain and HHs learning abilities.

Table 6.9: Comparison between GIHH, L-GIHH, and LSRL with and without the mutation LLHs.

Prob.	Alg.	LLHs	Rank	Gap	p-value	Sign.
k-GCP	GIHH	LS + MU ⋆	1.37	—	—	—
		LS	1.63	0.04	0.003	✓ (LS + MU)
	L-GIHH	LS + MU ⋆	1.40	—	—	—
		LS	1.60	0.03	0.042	✓ (LS + MU)
	LSRL	LS + MU ⋆	1.33	—	—	—
		LS	1.67	0.2	0.002	✓ (LS + MU)
PFSP	GIHH	LS + MU ⋆	1.22	—	—	—
		LS	1.78	0.30	<0.001	✓ (LS + MU)
	L-GIHH	LS + MU ⋆	1.11	—	—	—
		LS	1.89	0.30	<0.001	✓ (LS + MU)
	LSRL	LS + MU ⋆	1.50	—	—	—
		LS	1.50	0.01	0.834	✗
TSP	GIHH	LS + MU ⋆	1.34	—	—	—
		LS	1.66	0.19	0.017	✓ (LS+MU)
	L-GIHH	LS + MU ⋆	1.4	—	—	—
		LS	1.6	-0.70	0.141	✗
	LSRL	LS + MU ⋆	1.6	—	—	—
		MU	1.4	-1.23	0.052	✗

To complement the ablation analysis, we also evaluate the standalone performance of each individual LLH, running them independently outside the HH framework. The goal of this comparison is to assess whether HHSA is capable of outperforming its constituent components. Demonstrating superior performance relative to the best individual LLHs would provide further evidence for the effectiveness of collaborative management and adaptive selection within the HH framework. To do so, we run five FTSA considering the five temperatures used in the HHSA. Tables 6.10 to 6.12 present the comparative analysis between HHSA and the individual FTSA configurations. In most cases, the HHSA algorithm outperforms the fixed-temperature variants, with a few exceptions. In two cases from the k-GCP domain and two from the PFSP, HHSA performs on par with the best-performing low-temperature FTSA. However, when considering ranks, only in one case a low-temperature FTSA outperforms HHSA.

Effectiveness of the Adaptive Strategy

To assess the importance of adaptiveness and learning within HHSA, we compare the performance of the employed HHs against a random-based baseline, denoted as HH_r. This baseline selects a random LLH at each iteration, without any adaptive control. The goal of this experiment is to determine whether the effectiveness of HHSA stems simply from

Table 6.10: Comparison at different temperatures for the k-Graph Coloring Problem (k-GCP).

Alg.	Rank	Gap	p-value	Sign.
GIHH⋆	1.83	—	—	—
$\text{FTSA}_{1.0}$	4.10	3.30	<0.001	✓ (GIHH)
$\text{FTSA}_{0.75}$	4.15	3.28	<0.001	✓ (GIHH)
$\text{FTSA}_{0.50}$	4.13	3.28	<0.001	✓ (GIHH)
$\text{FTSA}_{0.25}$	4.13	3.27	<0.001	✓ (GIHH)
$\text{FTSA}_{0.00001}$	2.66	0.05	0.023	✓ (GIHH)
L-GIHH ⋆	1.88	—	—	—
$\text{FTSA}_{1.0}$	4.10	3.29	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.75}$	4.15	3.28	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.50}$	4.13	3.28	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.25}$	4.13	3.26	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.00001}$	2.61	0.04	0.099	✗
LSRL⋆	1.91	—	—	—
$\text{FTSA}_{1.0}$	4.10	3.30	<0.001	✓ (LSRL)
$\text{FTSA}_{0.75}$	4.15	3.29	<0.001	✓ (LSRL)
$\text{FTSA}_{0.50}$	4.13	3.28	<0.001	✓ (LSRL)
$\text{FTSA}_{0.25}$	4.13	3.27	<0.001	✓ (LSRL)
$\text{FTSA}_{0.00001}$	2.58	0.05	0.117	✗

introducing temperature variation, or from doing so strategically, i.e., at the right time and in the right context, through the HH's ability to learn and respond to the underlying fitness landscape. Table 6.13 summarizes the results of this analysis. With the exception of one case (PFSP with LSRL) results indicate that the full HHSA configuration consistently outperforms its ablated counterparts. Moreover, the differences across methods are statistically significant in all but that one case. These findings confirm that the HHs are effectively adapting to the problem characteristics, leveraging the available LLHs to guide the search process more efficiently.

6.4 Concluding Remarks

Section 6.4.1 summarizes the chapter, Section 6.4.2 discusses its limitations, and Section 6.4.3 outlines directions for future research.

6.4.1 Content

We investigated a HH approach for dynamically controlling the temperature parameter in SA, aiming to remove costly parameter tuning while maintaining high performance. The framework combines LLHs derived from FTSA under the guidance of HHs and is implemented within a modular environment bridging `HyFlex` and EasyLocal++. Results show that HHs effectively learn to favor beneficial LLHs while discarding less useful ones. In two of three domains, HHSA matched or outperformed finely tuned SA, and the analysis indicates that HHs apply heuristics in coherent sequences, revealing emergent heuristic-application policies.

6.4.2 Limitations

While the results are encouraging, this study presents some limitations. First, the current LLHs portfolio does not account for varying degrees of mutation intensity within the

Table 6.11: Different temperatures for the Permutation Flowshop Scheduling Problem (PFSP).

Alg.	Rank	Gap	p-value	Sign.
GIHH$\star$	1.58	—	—	—
FTSA$_1$	4.63	9.32	<0.001	✓ (GIHH)
FTSA$_{0.75}$	4.57	9.22	<0.001	✓ (GIHH)
FTSA$_{0.50}$	4.25	9.16	<0.001	✓ (GIHH)
FTSA$_{0.25}$	4.55	9.3	<0.001	✓ (GIHH)
FTSA$_{0.00001}$	1.42	-0.09	0.055	✗
L-GIHH$\star$	1.54	—	—	—
FTSA$_1$	4.63	9.37	<0.001	✓ (L-GIHH)
FTSA$_{0.75}$	4.57	9.27	<0.001	✓ (L-GIHH)
FTSA$_{0.50}$	4.25	9.21	<0.001	✓ (L-GIHH)
FTSA$_{0.25}$	4.55	9.36	<0.001	✓ (L-GIHH)
FTSA$_{0.00001}$	1.46	-0.04	0.128	✗
LSRL$\star$	1.7	—	—	—
FTSA$_1$	4.63	9.09	<0.001	✓ (LSRL)
FTSA$_{0.75}$	4.57	8.99	<0.001	✓ (LSRL)
FTSA$_{0.50}$	4.25	8.93	<0.001	✓ (LSRL)
FTSA$_{0.25}$	4.55	9.08	<0.001	✓ (LSRL)
FTSA$_{0.00001}$	1.3	-0.29	<0.001	✓ (FTSA)

move operators, limiting the granularity of adaptability. Second, the setup includes only homogeneous LLHs, each corresponding to a FTSA with fixed iteration budgets, which may restrict flexibility in search dynamics. Third, the SA parameters are fully delegated to the HH layer, which itself requires parameter tuning; this two-level parameterization may introduce additional complexity and dependencies.

6.4.3 Future Work

Several directions can extend the present work. One way is to introduce mutation operators with adjustable intensity, enabling finer control of search diversification and intensification. Another is to expand the portfolio with heterogeneous LLHs, such as variable-length runs or complete SA executions, to increase flexibility in search strategies. Finally, future research could explore transferring learned control strategies across instances or domains, paving the way for cross-domain generalization.

Table 6.12: Different temperatures for the Traveling Salesperson Problem (TSP).

Alg.	Rank	Gap	p-value	Sign.
GIHH$\star$	1.33	—	—	—
FTSA_{1}	4.66	>100.00	<0.001	✓ (GIHH)
$\text{FTSA}_{0.75}$	4.82	>100.00	<0.001	✓ (GIHH)
$\text{FTSA}_{0.50}$	4.4	>100.00	<0.001	✓ (GIHH)
$\text{FTSA}_{0.25}$	4.12	>100.00	<0.001	✓ (GIHH)
$\text{FTSA}_{0.00001}$	1.67	28.6	<0.001	✓ (GIHH)
L-GIHH$\star$	1.32	—	—	—
FTSA_{1}	4.66	>100.00	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.75}$	4.82	>100.00	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.50}$	4.4	>100.00	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.25}$	4.12	>100.00	<0.001	✓ (L-GIHH)
$\text{FTSA}_{0.00001}$	1.68	26.7	<0.001	✓ (L-GIHH)
LSRL$\star$	1.32	—	—	—
FTSA_{1}	4.66	>100.00	<0.001	✓ (LSRL)
$\text{FTSA}_{0.75}$	4.82	>100.00	<0.001	✓ (LSRL)
$\text{FTSA}_{0.50}$	4.4	>100.00	<0.001	✓ (LSRL)
$\text{FTSA}_{0.25}$	4.12	>100.00	<0.001	✓ (LSRL)
$\text{FTSA}_{0.00001}$	1.68	27.82	<0.001	✓ (LSRL)

Table 6.13: Comparison between GIHH, L-GIHH, and LSRL and HH_{r}.

Prob.	Alg.	Rank	Gap	p-value	Sign.
k-GCP	$\text{HH}_{\text{r}}\star$	3.01	—	—	—
	GIHH	2.17	-0.085	<0.001	✓ (HHSA)
	LGIHH	2.36	-0.073	<0.001	✓ (HHSA)
	LSRL	2.46	-0.085	0.002	✓ (HHSA)
PFSP	HH_{r}	2.78	—	—	—
	GIHH	2.17	-0.186	0.018	✓ (HHSA)
	LGIHH	1.92	-0.234	0.003	✓ (HHSA)
	LSRL	3.13	0.019	0.443	✗
TSP	HH_{r}	2.97	—	—	—
	GIHH	2.4	-8.116	0.010	✓ (HHSA)
	LGIHH	2.41	-6.705	<0.001	✓ (HHSA)
	LSRL	2.22	-7.321	<0.001	✓ (HHSA)

Part III

Oven Scheduling Problem

Chapter 7

Introduction

Energy production and use are among the main drivers of global warming and the climate crisis.[1] Improving energy efficiency therefore emerges as one of the most cost-effective mitigation measures, particularly in industrial sectors.[2] This priority is reflected in the United Nations Sustainable Development Goals (UN SDGs), specifically Goal 9 (*Build resilient infrastructure, promote inclusive and sustainable industrialization, and foster innovation*) and Goal 12 (*Ensure sustainable consumption and production patterns*).

The urgency is evident: without action, annual electricity consumption is projected to rise sharply across applications such as material processing, lighting, refrigeration, air conditioning, and the operation of electric motors and transformers. This growing demand is fueled by rising income levels, population growth, and the expansion of high-tech industries. Yet many policies on energy efficiency remain weak, outdated, or non-existent.[3]

Among the most energy-intensive sectors is semiconductor fabrication [98], where specialized heat-treatment ovens (Figure 7.1) consume large amounts of energy during the component hardening process (Figure 7.2). In 2021 alone, the industry consumed 149 billion kWh of electricity, enough to power a metropolis of more than 25 million people for an entire year [488].

A practical way to reduce this high energy use is to group compatible components (or jobs) and process them in batches, thus maximizing efficient use of resources. However, deciding how to assign jobs to batches and schedule them across multiple ovens involves complex constraints. This problem is known in the literature as (parallel) batch scheduling problm [180]. Within this family, the Oven Scheduling Problem (OSP) has recently been introduced as a real-world, NP-hard parallel batch scheduling problem in the context of the semiconductor industry [275]. The OSP aims to group compatible components into batches and determine an optimal processing schedule across available ovens. Jobs in the OSP are characterized by features such as minimum and maximum processing times, due dates, and sizes. Ovens, in turn, operate within specific time slots and have limited capacity. The objective combines three components: the cumulative processing time of batches, the number of tardy jobs, and the cumulative setup costs.

While the OSP builds on constraints and objectives familiar in scheduling research, it

[1] https://www.unep.org/topics/energy. Accessed 2025–10–04.

[2] https://sdgs.un.org/2030agenda. Accessed 2025–10–04.

[3] https://united4efficiency.org. Accessed 2025–10–04.

Figure 7.1: Reflow oven typically used in the semiconductor industry (photograph by Nelatan, distributed under a CC BY-SA 3.0 license).

represents the first formulation that integrates a broad range of real-world requirements into a cohesive model. Thus, it establishes a testbed that more faithfully mirrors industrial conditions and moves beyond simplified benchmark.

The problem has so far been tackled mainly with exact approaches, such as Constraint Programming (CP) and Integer Linear Programming (ILP) [275]. These methods offer theoretical rigor but face two major limitations. First, they struggle to solve large instances optimally as job counts grow, which is a critical issue, since industrial settings often involve hundreds of jobs. Second, in practice, timely decision-making is essential: it is often preferable to obtain good solutions quickly than optimal ones after long computations. Yet current exact methods typically require up to an hour of runtime (i.e., the given timeout), which severely limits their usability in time-sensitive applications. This motivates the exploration of scalable heuristic and metaheuristic approaches for the OSP.

7.1 Goals

The central goal of this part of the thesis is to develop efficient and scalable methods for solving the OSP, particularly suited for large and realistic problem instances where exact methods fall short. Equally important is the rigorous characterization of these methods, ensuring they are not only effective in practice but also well understood in terms of their underlying dynamics. This entails examining how different algorithmic components influence solution quality, analyzing performance relative to instance characteristics, and exploring the structure of the search space.

At the time these studies were conducted, the application of advanced analysis techniques, such as Instance Space Analysis (ISA) and Search Trajectory Networks (STNs), to non-benchmark, industrially relevant problems was still very limited. Yet such analyses are crucial for demonstrating the practical relevance of optimization methods to potential industry collaborators. Accordingly, the contributions of this work extend beyond the parallel batch scheduling community, offering methodological value to the broader

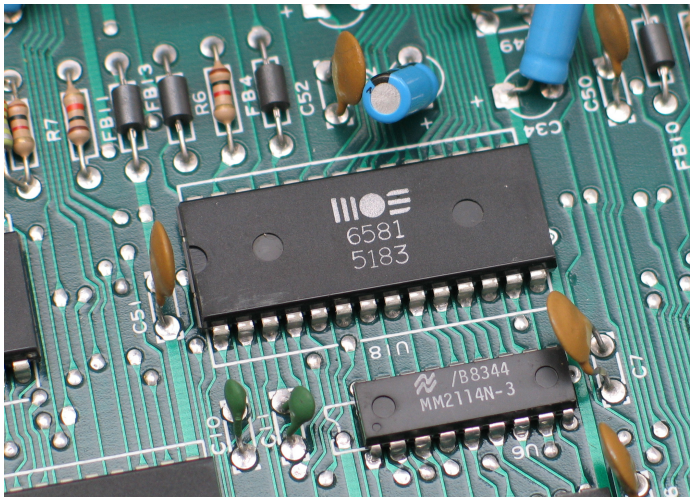


Figure 7.2: Example of component, i.e., a MOS 6581 sound chip from a Commodore 64 main board (photograph by Christian Taube, distributed under a Creative Commons Attribution-Share Alike 2.5 Generic license).

benchmarking and algorithm analysis domains.

These goals can be formulated as the following Research Questions (RQs):

RQ1 Can exact methods be improved through theoretical insights?

RQ2 Can exact methods performance be enhanced by integrating heuristic approaches?

RQ3 How do classical metaheuristics perform on the OSP?

RQ4 What are the key features of the OSP that characterize its structure, and which features best capture the difficulty of an instance?

RQ5 How can we effectively characterize the search spaces of OSP instances?

To address **RQ1**, we develop and analyze theoretically derived Lower Bounds (LBs). These bounds incorporate the problem's core constraints, including machine eligibility, machine availability, job processing times, and batch-dependent setup costs. The investigation addressing this RQ is presented in Chapter 9 and refers to the following works:

- **Da Ros, F.**, Lackner, M., & Musliu, N. (2024). Theoretical lower bounds for the oven scheduling problem. In *Proceedings of the 14th International Conference on the Practice and Theory of Automated Timetabling*. Conference rank: CORE2023 C.
- Lackner, M. L., **Da Ros, F.**, & Musliu, N. (2025, submitted). Theoretical lower bounds for a parallel batch scheduling problem. *Submitted to a journal*.

To address **RQ2**, we design a tailored Large Neighborhood Search (LNS) approach for the OSP. The method introduces specialized destroy and repair strategies, alongside parameter tuning, with the explicit goal of handling large instances intractable for exact solvers. The investigation addressing this RQ is presented in Chapter 10 and refers to the following work:

- **Da Ros, F.**, & Di Gaspero, L., Lackner, M. L., & Musliu, N. (2024). Reducing Energy Consumption in Electronic Component Manufacturing through Large Neighborhood Search. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 1706–1714). New York, NY: ACM.

To address **RQ3**, we develop a multi-neighborhood Simulated Annealing (SA) algorithm. This approach consistently outperforms CP, ILP, and even our proposed LNS method on large instances, while achieving competitive results on smaller ones. It also provides good results on a related batch scheduling problem (Appendix D). The investigation addressing this RQ is presented in Chapter 10 and refers to the following work:

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., Musliu, N., & Winter, F. (2025). Multi-neighborhood simulated annealing for the oven scheduling problem. *Computers & Operations Research, 177*, 106999. Journal rank: Scimago Q1 (2024).

To address **RQ4**, we perform an ISA using a set of 165 instance features. Following the methodology of Smith-Miles and Muñoz [436], we apply dimensionality reduction and clustering to explore the instance space. This analysis is complemented by algorithm selection techniques, where predictive models confirm the ability of certain features to discriminate between algorithmic behaviors. The investigation addressing this RQ is presented in Chapter 11 and refers to the following work:

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., & Musliu, N. (2025). Instance space analysis and algorithm selection for a parallel batch scheduling problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)* (pp. 66–83). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

To address **RQ5**, we apply STNs to a subset of problem instances. This analysis compares the search behavior of both the SA and LNS algorithms, providing insights into their dynamics and their interactions with the underlying landscape structure The investigation addressing this RQ is presented in Chapter 12 and refers to the following work:

- **Da Ros, F.**, Di Gaspero, L., Lackner, M. L., Musliu, N., & Soprano, M. (2025). Search trajectory networks applied to a real-world parallel batch scheduling problem. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)* (pp. 68–85). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

Figure 7.3 provides an overview of the structure of this part of the thesis. Background is highlighted in blue, while the original contributions of this thesis are marked in orange.

7.2 Structure

This part of the thesis is structured as follows. Chapter 8 presents a detailed description of the OSP. Chapter 9 discusses the theoretical and empirical development of LBs for the OSP. Chapter 10 introduces the two solution methods developed in this thesis: LNS and SA, respectively. Chapter 11 reports on the ISA conducted to understand instance-level characteristics and algorithm performance. Finally, Chapter 12 focuses on the landscape analysis, using STNs to study the search behavior of SA and LNS.

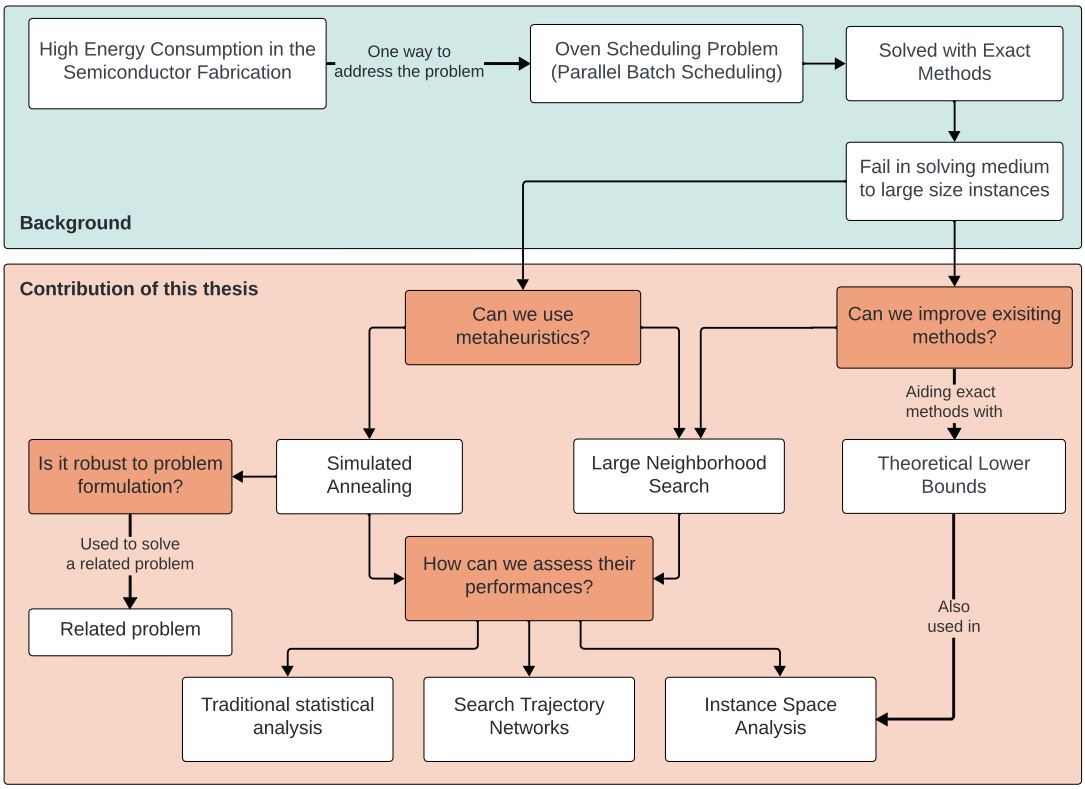


Figure 7.3: Schematic representation of the goals and content of this part of the thesis.

Chapter 8

Problem Definition

The Oven Scheduling Problem (OSP) is a real-world scheduling problem involving parallel batch processing with machine eligibility, availability constraints, and setup times. This problem originates from a collaboration with an industrial partner in Central Europe and reflects the challenges commonly encountered in high-throughput manufacturing industry.

To make the problem accessible to both technical and non-technical audiences, we provide two complementary perspectives: a lay description, which offers an intuitive overview of the problem's logic and practical implications, and a formal definition, which rigorously defines the problem using the mathematical and algorithmic structures needed for modeling and solving it.

Providing a lay description of the problem aligns with recent initiatives such as those promoted by the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action, which emphasize the importance of understanding how both academic and industrial users formulate and interact with optimization problems. These efforts advocate for bridging formal models with real-world requirements by making problem descriptions accessible, supporting user-driven formulation, and enabling broader adoption of optimization techniques. In this context, offering an intuitive explanation of the problem complements the formal definition, helping to contextualize the problem structure, clarify user needs, and support the development of tools and interfaces that promote practical and effective optimization workflows.

This chapter is organized as follows. Section 8.1 presents a lay description of the problem, followed by its formal definition in Section 8.2. Section 8.3 introduces the formal representation of the problem in terms of instance and solution structures.

8.1 Lay Problem Definition

In this section, we report a lay natural language description of the OSP, starting with entities (Section 8.1.1), followed by the optimization goal (Section 8.1.2), and concluding with the operational rules (Section 8.1.3).

8.1.1 Entities

The main entities of the OSP are the followings: a set of *machines* (or *ovens*), a set of *jobs* (or *components*), a set of *attributes*, representing the temperatures at which jobs need to be

processed, and the length of the *scheduling horizon*, representing the total time available.

Each machine is characterized by its *initial attribute* (temperature), a set of *availability* intervals indicating when it can operate, and a *maximum processing capacity*. Each job is characterized by its *release time*, *due date*, *minimum* and *maximum processing time*, specific *attribute* (temperature), *size*, and a subset of *eligible machines* for where it can be processed. Changing the oven temperature to match a job attribute incurs economic and time costs, represented by matrices describing *setup costs* and *setup times* for switching settings.

8.1.2 Optimization Goal

The aim of the OSP is to group compatible jobs into *batches* and devise a processing *schedule* on the set of machines. The optimization goal of the OSP consists of three parts: minimizing the cumulative processing time of the batches across machines, minimizing the cumulative setup costs associated with these batches, and minimizing the count of tardy jobs. The objective function is a combination of these components based on industry needs (e.g., in some cases the first component might be more important than the other two, or vice versa).

8.1.3 Operational Rules

A feasible solution to the OSP must satisfy the following operational rules:

- Each job can be processed by exactly one batch on one of its eligible machines.
- Each batch must be processed within the scheduling horizon.
- All jobs in one batch have the same start and end time.
- All jobs assigned to one batch must share the same attribute.
- The batch size must not exceed the capacity of the machine it is assigned to.
- A batch may not start before the release time of any job in that batch.
- The processing time of each batch must lie between the minimum and maximum processing times of all assigned jobs. Preemption is not allowed.
- Batches on the same machine may not overlap, and setup times must be accounted for between consecutive batches.
- Both the batch processing time and the preceding setup time must lie entirely within one of the machine's availability intervals.

8.2 Formal Problem Description

In this section, we report the mathematical formulation of the OSP, in the form of the Integer Linear Programming (ILP) as proposed by M. Lackner et al. [275]. Note that some of the notation has been modified w.r.t. the original version to facilitate readability. First, we describe the instance parameters (Section 8.2.1), then we report the decision variables (Section 8.2.2) and constraints (Section 8.2.3), and finally we conclude with the objective function (Section 8.2.4).

8.2.1 Instance Parameters

An instance of the OSP consists of a set $\mathcal{M} = \{1, \ldots, k\}$ of machines, a set $\mathcal{J} = \{1, \ldots, n\}$ of jobs, a set $\mathcal{A} = \{1, \ldots, a\}$ of attributes, and the length $l \in \mathbb{N}$ of the scheduling horizon. Furthermore, for each machine, the scheduling horizon is divided into a maximum of I availability intervals, with the remaining time representing the downtimes when the machines are unavailable. Table 8.1 reports the parameters of an instance of the OSP.

Table 8.1: Parameters of an instance of the Integer Linear Programming (ILP) model for the Oven Scheduling Problem (OSP).

Param.	Domain	Description
k	$\mathbb{N}$	Number of machines, so that $\mathcal{M} = \{1, \ldots, k\}$
n	$\mathbb{N}$	Number of jobs, so that $\mathcal{J} = \{1, \ldots, n\}$
a	$\mathbb{N}$	Number of attributes, so that $\mathcal{A} = \{1, \ldots, a\}$
l	$\mathbb{N}$	Length of the scheduling horizon, consisting of timeslots $[0, l]$
c_m	$\mathbb{N}$	Maximum capacity of machine $m \in \mathcal{M}$
$\alpha_m^{\text{initial}}$	$\mathcal{A}$	Initial attribute of machine $m \in \mathcal{M}$
$\sigma_{m,i}^{\text{start}}$	$[0, l]$	Start of the i-th interval on machine $m \in \mathcal{M}$ for $i \in [1, I]$
$\sigma_{m,i}^{\text{end}}$	$[0, l]$	End of the i-th interval on machine $m \in \mathcal{M}$ for $i \in [1, I]$
τ_j^{start}	$[0, l]$	Release date of job $j \in \mathcal{J}$
τ_j^{end}	$[0, l]$	Due date of job $j \in \mathcal{J}$
λ_j^{min}	$[0, l]$	Minimal processing time of job $j \in \mathcal{J}$. The instance minimal processing time is indicated as $\min_T$
λ_j^{max}	$[0, l]$	Maximal processing time of job $j \in \mathcal{J}$. The instance maximal processing time is indicated as $\max_T$
α_j	$\mathcal{A}$	Attribute of job $j \in \mathcal{J}$
s_j	$\mathbb{N}$	Size of job $j \in \mathcal{J}$
$\mathcal{E}_j$	$\mathcal{M}$	Eligible machines of job $j \in \mathcal{J}$
ST	$\mathcal{A} \times \mathcal{A} \to \mathbb{N}$	Matrix of setup times. The maximum setup time is indicated with $\max_{\text{ST}}$
SC	$\mathcal{A} \times \mathcal{A} \to \mathbb{N}$	Matrix of setup cost incurred. The maximum setup cost is indicated with $\max_{\text{SC}}$

Table 8.2: Decision variables for the Integer Linear Programming (ILP) formulation of the Oven Scheduling Problem (OSP).

Var.	Domain	Description
$X_{m,b,j}$	$\{0, 1\}$	1, if job j is processed in batch $B_{m,b}$, i.e., in batch number b on machine m; 0, otherwise. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n] \quad \forall j \in \mathcal{J}$
$S_{m,b}$	$[0, l]$	Start time of batch $B_{m,b}$. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$
$P_{m,b}$	$[0, \max_T]$	Processing time of batch $B_{m,b}$. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$
$I_{m,b,i}$	$\{0, 1\}$	1, if batch number b on machine m is processed during the machine's i-th availability interval; 0, otherwise. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n] \quad \forall i \in [1, I + 1]$.
$A_{m,b}$	$[0, a]$	Attribute of batch $B_{m,b}$. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$.
$T_{m,b,j}$	$\{0, 1\}$	1, if job j in batch $B_{m,b}$ is late; 0, otherwise. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n] \quad \forall j \in \mathcal{J}$.
$st_{m,b}$	$[0, \max_{\overline{ST}}]$	Setup time between batch $B_{m,b}$ and the following batch. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$.
$sc_{m,b}$	$[0, \max_{\overline{SC}}]$	Setup cost between batch $B_{m,b}$ and the following batch. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$.
$E_{m,b}$	$\{0, 1\}$	1, if batch $B_{m,b}$ is empty; 0, otherwise. Defined $\forall m \in \mathcal{M} \quad \forall b \in [1, n]$.

8.2.2 Decision Variables

The formulation relies on modeling batches by their processing order on machines. For this reason, we consider the following additions to the instance parameters. In

the worst possible case, there will be n batches (one batch for each job). Thus, a set $\mathcal{B} = \{B_{1,1}, \ldots, B_{1,n}, \ldots, B_{k,1}, \ldots, B_{k,n}\}$ models up to n batches for machines 1 to k. To break symmetries, it is enforced that batches are ordered in the machines based on their starting times, with empty batches scheduled at the end.

To handle empty batches, an additional attribute with value 0 is defined and the setup time matrix and the setup costs matrix are extended to $\overline{ST}$ and $\overline{SC}$, respectively, so that no time or cost is incurred when moving from/to an empty batch. Additionally, we add an availability interval $[l, l]$ to each machine so that empty batches can be scheduled on this interval (the maximum number of intervals becomes $I + 1$).

Three sets of decision variables are employed to model the OSP. First, we introduce a set of binary variables $X_{m,b,j}$ encoding whether job j is assigned to batch $B_{m,b}$. Jobs that are scheduled in the same batch are processed simultaneously, i.e., have the same start and end time. Therefore, the integer decision variables $S_{m,b}$ and $P_{m,b}$ are used to encode the start time and the processing time of batch $B_{m,b}$. Additionally, also some auxiliary variables are defined to simplify the problem modeling. Table 8.2 lists all decision variables.

8.2.3 Constraints

The ILP formulation of the OSP involves the following constraints. Each job is assigned to exactly one batch:

$$\sum_{m \in \mathcal{M}} \sum_{b \in [1,n]} X_{m,b,j} = 1 \qquad \forall j \in \mathcal{J}$$

Each job is assigned to exactly one of its eligible machines:

$$\sum_{m \in \mathcal{E}_j} \sum_{b \in [1,n]} X_{m,b,j} = 1 \qquad \forall j \in \mathcal{J}$$

Batches may not start before the release date of any job in the batch:

$$S_{m,b} \geq \tau_j^{\text{start}} \cdot X_{m,b,j} \qquad \forall m \in \mathcal{M}, b \in [1,n], j \in \mathcal{J}$$

Batch processing times must lie between the minimal and maximal processing time of all assigned jobs:

$$\begin{aligned} &\lambda_j^{\min} \cdot X_{m,b,j} \leq P_{m,b} \quad \wedge \\ &P_{m,b} \leq \lambda_j^{\max} \cdot X_{m,b,j} + \max_T \cdot (1 - X_{m,b,j}) \qquad \forall m \in \mathcal{M}, b \in [1,n], j \in \mathcal{J} \end{aligned}$$

Batches on the same machine may not overlap and setup times must be considered:

$$S_{m,b} + P_{m,b} + st_{m,b} \leq S_{m,b+1} \qquad \forall m \in \mathcal{M}, b \in [1, n-1]$$

Batches and the preceding setup times must lie within one machine availability interval:

$$\begin{aligned} &\sigma_{m,i}^{\text{start}} \cdot I_{m,b,i} \leq S_{m,b} \quad \wedge \\ &S_{m,b} \leq \sigma_{m,i}^{\text{end}} \cdot I_{m,b,i} + l \cdot (1 - I_{m,b,i}) \qquad \forall m \in \mathcal{M}, b \in [1,n], i \in [1, I+1] \end{aligned}$$

$$\sum_{i\in[1,I+1]} I_{m,b,i} = 1 \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$\sigma^{\text{start}}_{m,i} \cdot I_{m,b,i} \leq S_{m,b} - st_{m,b-1} \qquad \forall m \in \mathcal{M}, b \in [2,n], i \in [1,I+1]$$
$$\sigma^{\text{start}}_{m,i} \cdot I_{m,1,i} \leq S_{m,1} - \overline{ST}(\alpha^{\text{initial}}_m, A_{m,1}) \qquad \forall m \in \mathcal{M}, i \in [1,I+1]$$
$$S_{m,b} + P_{m,b} \leq \sigma^{\text{end}}_{m,i} \cdot I_{m,b,i} + l \cdot (1 - I_{m,b,i}) \qquad \forall m \in \mathcal{M}, b \in [1,n], i \in [1,I+1]$$

The total size of each batch may not exceed the assigned machine's capacity:

$$\sum_{j\in\mathcal{J}} s_j \cdot X_{m,b,j} \leq c_m \qquad \forall m \in \mathcal{M}, b \in [1,n]$$

Jobs in one batch must have the same attribute:

$$a_j \cdot X_{m,b,j} \leq A_{m,b} \quad \wedge$$
$$A_{m,b} \leq a_j \cdot X_{m,b,j} + a \cdot (1 - X_{m,b,j}) \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}, b \in [1,n]$$

The helper variables for tardy jobs are set as follows:

$$T_{m,b,j} \leq X_{m,b,j} \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}, b \in [1,n]$$
$$S_{m,b} + P_{m,b} \leq (X_{m,b,j} - T_{m,b,j}) \cdot (\tau^{\text{end}}_j - l) + l \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}, b \in [1,n]$$
$$S_{m,b} + P_{m,b} + (1 - T_{m,b,j}) \cdot (l+1) > \tau^{\text{end}}_j \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}, b \in [1,n]$$

Setup times and costs between batches on the same machine are defined as follows:

$$st_{m,b} = \overline{ST}(A_{m,b}, A_{m,b+1}) \qquad \forall m \in \mathcal{M}, b \in [1,n-1]$$
$$sc_{m,b} = \overline{SC}(A_{m,b}, A_{m,b+1}) \qquad \forall m \in \mathcal{M}, b \in [1,n-1]$$

The decision variables for empty batches are set as follows:

$$\sum_{j\in\mathcal{J}} X_{m,b,j} \geq 1 - E_{m,b} \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$X_{m,b,j} \leq 1 - E_{m,b} \qquad \forall j \in \mathcal{J}, m \in \mathcal{M}, b \in [1,n]$$
$$S_{m,b} \geq l \cdot E_{m,b} \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$P_{m,b} <= \max_T \cdot (1 - E_{m,b}) \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$E_{m,b} <= I_{m,b,I+1} \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$A_{m,b} <= a \cdot (1 - E_{m,b}) \qquad \forall m \in \mathcal{M}, b \in [1,n]$$
$$E_{m,b} <= E_{m,b+1} \qquad \forall m \in \mathcal{M}, b \in [1,n-1]$$

8.2.4 Objective Function

We aim to minimize the cumulative processing time of the batches (p), the number of tardy jobs (t), and the cumulative setup cost between the batches (sc). Specifically:

$$p = \sum_{m\in\mathcal{M}} \sum_{b\in[1,n]} P_{m,b}$$

$$sc = \sum_{m \in \mathcal{M}} \overline{SC}(s_m, A_{m,1}) + \sum_{m \in \mathcal{M}} \sum_{b \in [1,n-1]} sc_{m,b}$$
$$t = \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}} \sum_{b \in [1,n]} T_{m,b,j}$$

In order to combine the three objective components using a linear combination, p, sc, and t need to be normalized to $\tilde{p}$, $\tilde{sc}$ and $\tilde{t} \in [0, 1]$. The normalization is as follows:

$$\tilde{p} = \frac{p}{avg_t \cdot n} \text{ with } avg_t = \left\lceil \frac{\sum_{j \in \mathcal{J}} \lambda_j^{\min}}{n} \right\rceil$$
$$\tilde{sc} = \frac{sc}{\max(\max_{\overline{SC}}, 1) \cdot n}$$
$$\tilde{t} = \frac{t}{n}$$

The three normalized components are combined in the objective function (obj) as follows:

$$\text{obj} = \frac{w_p \cdot \tilde{p} + w_{sc} \cdot \tilde{sc} + w_t \cdot \tilde{t}}{w_p + w_{sc} + w_t} \quad \in [0, 1] \subset \mathbb{R} \tag{8.1}$$

where the weights w_p, w_{sc} and w_t take integer values considering three Use Cases (UCs):

UC-1: The focus is on respecting the due dates of the jobs, thus weights are set as follows: $w_p = 4$, $w_{sc} = 1$, and $w_t = 100$.

UC-2: The focus is on reducing the processing time of the batches, thus weights are considered in a lexicographic manner so that $w_p = w_t \cdot (n + 1)$, $w_{sc} = 1$, and $w_t = n \cdot \max_{SC} + 1$.

UC-3: The focus is both on respecting the due dates and on reducing the processing time of the batches, thus weights are set as follows: $w_p = 2$, $w_{sc} = 1$, and $w_t = 2$.

8.3 Representation of Problem Characteristics

In this section, we first present the details of the instance representation (Section 8.3.1), followed by a description of the solution representation (Section 8.3.2).

8.3.1 Instance Representation

To illustrate the OSP, we consider a randomly generated instance, where we need to schedule 5 jobs ($n = 5$) over 2 machines ($k = 2$), considering 2 attributes ($a = 2$) over a scheduling horizon of 60 time units ($l = 60$). The instance parameters related to the machines are as follows.

m	1	2
c_m	4	3
$\alpha_m^{\text{initial}}$	2	1
$[\sigma_{m,1}^{\text{start}}, \sigma_{m,1}^{\text{end}}]$	[0, 37]	[2, 30]
$[\sigma_{m,2}^{\text{start}}, \sigma_{m,2}^{\text{end}}]$	[40, 60]	[40, 60]

The instance parameters related to the jobs are as follows.

j	1	2	3	4	5
$\mathcal{E}_j$	1	1	1		
			2	2	2
τ_j^{start}	7	5	5	6	8
τ_j^{end}	22	20	19	22	20
λ_j^{min}	12	10	11	10	11
λ_j^{max}	15	13	12	15	15
s_j	2	2	2	3	2
α_j	1	1	2	2	2

The instance parameters related to setup times and costs are as follows.

$$\text{ST} = \begin{pmatrix} 2 & 3 \\ 1 & 5 \end{pmatrix} \qquad \text{SC} = \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix}$$

The instances provided in the original work were available in three formats: `dzn`, `dat`, and `json`. While the content across these formats was equivalent, the structure and data representation varied. For example, the `json` files expressed dates and times using UNIX timestamps, whereas the `dzn` and `dat` formats represented time in discrete units ranging from 0 to the end of the scheduling horizon.

In this thesis, we adopt the `dzn` format for both the literature instances (Section 3.4) and those newly generated (Chapter 11). This choice is motivated by the following considerations: *(i)* `dzn` is supported by libraries in most common programming languages, facilitating parsing and integration; *(ii)* it is human-readable and easy to interpret; *(iii)* it is independent of timestamp conventions, avoiding ambiguity in time representation. Listing 8.1 presents the structure of a `dzn` instance for the OSP. In addition to the core instance parameters, the file also includes auxiliary information such as normalization factors for the objective value and supporting data useful for defining variable domains, such as the minimum job duration and the maximum setup cost.

Listing 8.1: Scheme of a `dzn` instance for the Oven Scheduling Problem (OSP).

```
l=..;
a=..;
setup_costs=[|..|..|];
setup_times=[|..|..|];
```

```
m=..;
min_cap=[..];
max_cap=[..];

initState=[..];
s=..;
m_a_s = [|..,|..|];
m_a_e = [|..,|..|];
n=..;
eligible_machine = [{..},];
earliest_start=[..];
latest_end=[..];
min_time=[..];
max_time=[..];
size=[..];
attribute=[..];

upper_bound_integer_objective=..;
mult_factor_total_runtime=..;
mult_factor_finished_toolate=..;
mult_factor_total_setuptimes=..;
mult_factor_total_setupcosts=..;
running_time_bound=..;
min_duration=..;
max_duration=..;
max_setup_time=..;
max_setup_cost=..;
```

8.3.2 Solution Representation

Figure 8.1 illustrates a feasible solution for this instance. Each machine is described by a timeline, where the x-axis represents time, while the y-axis represents the capacity of the machine. The maximum capacity of each machine $m \in \mathcal{M}$ (c_m) is marked with a dashed line, and the machines' unavailability periods are hatched rectangles. Each rectangle represents a job $j \in \mathcal{J}$: the height corresponds to its size (s_j) and the color indicates its attribute (α_j). Batches are represented by stacked jobs, such as job 1 and job 2. The width of each block represents the processing time of the batch (e.g., in the case of job 1 and job 2, the minimal processing time of the batch corresponds to the maximum between $\lambda_1^{\min}$ and $\lambda_2^{\min}$, which is $\lambda_2^{\min} = 12$; note that $\lambda_1^{\min}$ is highlighted with a dashed line). In this solution, the running time of the ovens is $p = 44$, the number of tardy jobs is $t = 2$ (jobs 3 and 5 finish after their due date), and the setup costs are $sc = 4$. This solution is optimal with respect to the objective function defined in Equation (8.1) and referring to UC-1.

While the solution representation described above is well-suited for human interpretation, as it offers a clear visualization of durations and scheduling constraints, it is not machine-readable. In the original work, no standard format was specified for solution output. As a result, solutions were typically reported in the default print format of the respective solution methods, often including both decision and auxiliary variables.

In this thesis, we standardize solution output by adopting a structured `json` format. This choice facilitates automated parsing, reproducibility, and integration with analysis tools. Listing 8.2 presents an example of a solution file in `json` format. For each job, the file

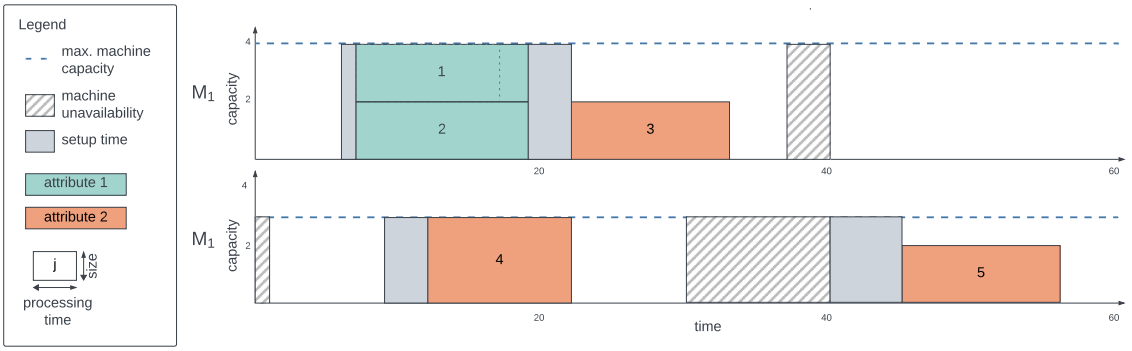


Figure 8.1: Visual representation of a feasible solution for the Oven Scheduling Problem (OSP).

specifies the assigned batch (i.e., its position within the machine's processing sequence), the machine to which the job is allocated, the job's start time, and its associated processing time. Such a representation is general enough to accommodate potential relaxations of certain constraints. For example, while the current model assumes that all jobs within a batch must start and finish simultaneously, this structure could be adapted to support alternative formulations.

Listing 8.2: Scheme of a `json` solution file for the Oven Scheduling Problem (OSP).

```
{
    "batch_for_job": [...],
    "machine_for_job": [...],
    "start_time_per_job": [...],
    "processing_time_per_job": [...]
}
```

Chapter 9

Theoretical Lower Bounds

In practice, it is often preferable to obtain near-optimal solutions quickly rather than exact ones. However, evaluating their quality is difficult without reliable baselines, particularly when exact methods are unavailable or fail to provide tight Lower Bounds (LBs). This is the the case of Oven Scheduling Problem (OSP) where exact approaches found optimal solutions for 38 of 80 benchmark instances and rarely did so for larger real-world cases. Problem-specific, efficiently computable LBs are therefore valuable: they support solution quality assessment, strengthen exact methods by bounding variable ranges, and assist metaheuristics through stopping criteria or search guidance.

Although theoretical LBs exist for batch scheduling problems (Section 3.5), they cannot be directly applied to the OSP, as they overlook key constraints and objectives.Thus, in this chapter, we present and evaluate a theoretical approach to computing LBs for the objectives of the OSP and considering elements neglected from the current literature.

Our primary contributions are as follows:

- **Thereotical LBs**: We introduce a procedure to compute theoretical LBs for the three objective components of the OSP and related aspects, more specifically for the number of batches, the total oven runtime, the number of tardy jobs, and the total setup costs. Our approach differs from the existing literature as it considers machine eligibility and compatibility of processing times. Moreover, a Vehicle Routing instance is solved in order to compute a LB on the setup costs and a Minimum Cost Flow instance is solved to bound the tardiness.

- **Evaluation of the LBs**: We comprehensively evaluate the tightness of our calculated LBs, encompassing the overall cost function and its individual components. We differentiate between instances where an optimal solution is available and those where it is not. Notably, for larger instances, our calculated LBs provide a small gap w.r.t. the optimal solution value and very often outperform the LBs generated by commercial solvers (when the optimal solution value is not known).

- **Integration of LBs in existing methods**: We integrate the LBs into exact solution approaches and uncover that the calculation times are greatly reduced, and new optimality proofs and best solutions are found.

In a subsequent chapter (Chapter 11), these LBs will be used to characterize instance complexity in a Instance Space Analysis (ISA).

This chapter is structure as follows. Section 9.1 introduces the theoretical LBs; Section 9.2 details the experimental setup; Section 9.3 presents the results; and Section 9.4 offers concluding remarks.

9.1 Methodology

We now present the theoretical LBs for each component of the objective function and for the number of batches. The examples refer to the instance of Section 8.3.1.

9.1.1 Lower Bound on the Number of Batches and Processing Time

Since jobs can only be combined in a batch if they share the same attribute, bounds on the number of batches required are calculated independently for all attributes. For a given attribute $r \in \mathcal{A}$, we denote by b_r the number of batches in a feasible solution and by p_r the minimal cumulative processing time of batches consisting of jobs with attribute r.

Bound based on Machine Capacities and Job Sizes

Due to the machine capacity constraints, a bound on the required number of batches is

$$b_r \geq \left\lceil \frac{\sum_{j \in \mathcal{J}: \alpha_j = r} s_j}{\max_{m \in \mathcal{M}} \{c_m\}} \right\rceil, \tag{9.1}$$

as stated by Koh et al. [262]. This corresponds to the minimal number of batches required if we assume that jobs can be split into smaller jobs of unit size and that all jobs can be scheduled on the machine with the largest machine capacity. This bound can be tightened by applying bounds that have been developed in the Bin Packing literature. A first possibility for improvement is to distinguish between "large" and "small" jobs (similarly to Damodaran and Vélez-Gallego [125] and Li et al. [293]). Large jobs are those jobs that are so large that they cannot accommodate any other jobs in the same batch and thus need to be processed in a batch of their own. All other jobs are referred to as small jobs. For a given attribute r, the sets of large jobs J_r^l and small jobs J_r^s with attribute r are as follows:

$$J_r^l = \Big\{ j \in \mathcal{J} : \alpha_j = r, s_j + s_i > \max_{m \in \mathcal{E}_j}(c_m) \quad \forall i \in \mathcal{J} \text{ with } i \neq j \text{ and } a_i = r \Big\}$$

$$J_r^s = \{ j \in \mathcal{J} : \alpha_j = r \} \setminus J_r^l$$

Instead of the bound in equation (9.1), we thus have the tighter bound:

$$b_r \geq |J_r^l| + \left\lceil \frac{\sum_{j \in J_r^s} s_j}{\max_{m \in \mathcal{M}} \{c_m\}} \right\rceil. \tag{9.2}$$

Another, slightly different possibility for improvement, is to apply bounds that have been developed by Martello and Toth [332] for the Bin Packing Problem, in particular the

bound L_2 given in Theorem 2.2 therein. Given any integer k with $0 \leq k \leq \frac{1}{2}C$ where $C = \max_{m\in\mathcal{M}}\{c_m\}$ and an attribute $r \in \mathcal{A}$, the following sets are defined (the notation from [332] is adapted to the OSP setting):

$$\begin{aligned} J_r^1(k) &= \{j \in \mathcal{J} : \alpha_j = r, s_j > C - k\} \\ J_r^2(k) &= \{j \in \mathcal{J} : \alpha_j = r, C - k \geq s_j > C/2\} \\ J_r^3(k) &= \{j \in \mathcal{J} : \alpha_j = r, C/2 \geq s_j > k\}\,. \end{aligned}$$

Then it holds that

$$\begin{aligned} b_r(k) = |J_r^1(k)| + |J_r^2(k)| + \\ + \max\left(0, \left\lceil \frac{\sum_{j\in J_r^3(k)} s_j - \left(|J_r^2(k)|C - \sum_{j\in J_r^2(k)} s_j\right)}{C} \right\rceil\right) \end{aligned} \tag{9.3}$$

is a LB on the number of batches with jobs of attribute r. This is because all jobs in $J_r^1(k) \cup J_r^2(k)$ have a size greater than $C/2$ and require a separate batch.

In the following, we refine these bounds from the literature by considering the machine eligibility constraints and the compatibility of processing times.

Refinement of the Bound for Small Jobs based on Machine Eligibility

Considering all small jobs of attribute $r \in \mathcal{A}$, we further distinguish them between those that can be processed on several machines and those with a single eligible machine. Given a machine $i \in \mathcal{M}$, we use the following notation:

$$b_{r,i} = \frac{\sum_{j\in J_r^s:\mathcal{E}_j=\{i\}} s_j}{c_i}, \text{ and } cap_i = (\lceil b_{r,i}\rceil - b_{r,i}) \cdot c_i$$

i.e., $\lceil b_{r,i}\rceil$ is the minimal number of batches with small jobs that need to be processed on machine i and cap_i is the total remaining capacity in these batches.

To schedule the small jobs of attribute r, we proceed as follows:

- Small jobs that need to be processed on a specific machine are scheduled on this machine.
- The remaining small jobs are used to fill up the previously created batches.
- If there are still jobs left, we assume that they can be split into unit-size jobs and can be scheduled on the machine with maximal capacity, creating b_r^* additional batches.

The bound in equation (9.2) can thus be tightened as follows:

$$\begin{aligned} b_r \geq b_r^E &:= |J_r^l| + \sum_{i\in\mathcal{M}} \lceil b_{r,i}\rceil + b_r^*, \text{ where} \\ b_r^* &= \left\lceil \frac{\max\left(0, \sum_{j\in J_r^s:|\mathcal{E}_j|>1} s_j - \sum_{i\in\mathcal{M}} cap_i\right)}{\max_{m\in\mathcal{M}}\{c_m\}} \right\rceil \end{aligned} \tag{9.4}$$

In order to calculate a LB on the cumulative batch processing time p_r of these batches, note that all large jobs are processed in batches of their own which run for their respective minimal processing times. Thus

$$p_r \geq \sum_{j \in J_r^l} \lambda_j^{\min} + p_r^E, \tag{9.5}$$

where p_r^E denotes the minimal cumulative processing time of batches consisting of small jobs with attribute r. A bound for p_r^E can be calculated as follows:

- For every machine i with $b_{r,i} > 0$, create the collection of minimal processing times of small jobs that need to be processed on i; create the sum of the $\lceil b_{r,i} \rceil$ smallest elements from this collection.
- From the collection of minimal processing times of small jobs that can be processed on multiple machines, create the sum of the b_r^* smallest elements.
- Among all small jobs, pick the one with the largest minimal processing time. The batch containing this job will necessarily have this job's minimal processing time. In the previous two sums, one can thus replace the overall largest processing time with this value.

Refinement of the Bin Packing Bound based on Machine Eligibility

In a similar fashion, the bound in equation (9.3) can be refined taking into account machine eligibility constraints. For this purpose, we denote by $J_{r,i}^2(k)$, respectively $J_{r,i}^3(k)$, the set of jobs in $J_r^2(k)$, respectively $J_r^3(k)$, that can only be processed on machine $i \in \mathcal{M}$. In analogy to the bound in equation (9.4), the bin packing bound $b_r(k)$ from equation (9.3) can be tightened as follows:

$$\begin{aligned}
b_r &\geq \max_{0 \leq k \leq \frac{1}{2}C} b_r^E(k), \text{ where} \qquad (9.6)\\
b_r^E(k) &:= |J_r^1(k)| + \sum_{i \in \mathcal{M}} \left(|J_{r,i}^2(k)| + b'_{r,i} \right) \\
&\quad + \left\lceil \frac{1}{C} \max \left(0, \sum_{\substack{j \in J_r^2 \cup J_r^3, \\ |\mathcal{E}_j| > 1}} s_j - \sum_{i \in \mathcal{M}} (c_i b'_{r,i} - cap_{r,i}) \right) \right\rceil, \\
b'_{r,i} &:= \max \left(0, \left\lceil \frac{cap_{r,i}}{c_i} \right\rceil \right), \\
cap_{r,i} &:= \sum_{j \in J_{r,i}^3(k)} s_j - \sum_{j \in J_{r,i}^2(k)} (c_i - s_j)
\end{aligned}$$

A LB on the batch processing times is given by

$$p_r \geq \max_{0 \leq k \leq C/2} \left(\sum_{j \in J_r^1(k)} \lambda_j^{\min} + p_r^E(k) \right), \tag{9.7}$$

where $p_r^E(k)$ can be obtained analogously to the bound p_r^E in equation (9.5).

Refinement of the Bound for Small Jobs based on Compatible Job Processing Times

Two jobs i and j with respective minimal and maximal processing times $\lambda_i^{\min}, \lambda_j^{\min}$ and $\lambda_i^{\max}, \lambda_j^{\max}$ may only be combined in a batch if the intervals of their processing times have a non-empty intersection:

$$[\lambda_i^{\min}, \lambda_i^{\max}] \cap [\lambda_j^{\min}, \lambda_j^{\max}] \neq \emptyset. \tag{9.8}$$

This compatibility relation between jobs can be represented with the help of a *compatibility graph* $G = (V, E)$, where V is the set of all jobs $\mathcal{I}$ and $(i, j) \in E$ if and only if the jobs i and j have compatible processing times. In this graph, a batch forms a (not necessarily maximal) clique. The problem of solving an OSP instance with unit-sized jobs, i.e. $s_j = 1$ for all $j \in \mathcal{J}$, and a single machine with capacity c is thus equivalent to covering the nodes of the compatibility graph with the smallest number of cliques with size no larger than c. Let us therefore consider the following special case of the OSP:

> *OSP*: Given a set of jobs $\mathcal{I}$ of unit size defined by their minimal and maximal processing times, i.e. $j = [\lambda_j^{min}, \lambda_j^{max}]$ and $s_j = 1$ for all $j \in \mathcal{I}$, and a single machine with capacity $c \in \mathbb{N}$, how many batches do we need at least in order to process all jobs if jobs can only be processed in the same batch if the compatibility condition* (9.8) *is fulfilled?*

Several variants of this problem have been studied in the literature, e.g. by Finke et al. [179]; the variant that we are interested in corresponds to their problem (P2). Solving the problem OSP* will allow us to obtain LBs for the OSP: Indeed, as in equation (9.1), we obtain LBs on the number of batches required and their processing times if we assume that jobs can be split into smaller jobs of unit size and that all jobs can be scheduled on the machine with largest machine capacity.

Covering the nodes of a graph with the smallest number of cliques with size no larger than c is NP-complete for arbitrary graphs, but solvable in polynomial time for interval graphs. A simple greedy algorithm to solve this problem is provided by Finke et al. [179] and referred to as the algorithm GAC (greedy algorithm with compatibility). By adapting the order in which jobs are processed by the GAC algorithm, we obtain an algorithm that minimizes both the number of batches and the cumulative batch processing time. We call this algorithm GAC+.

Algorithm GAC+. Consider the set of jobs $\mathcal{I}$ in non-increasing order $j_1, j_2, \ldots, j_n$ of their minimal processing times $\lambda_j^{\min}$, breaking ties arbitrarily. Construct one batch per iteration until all jobs have been placed into batches. In iteration i, open a new batch B_i and label it with the first job j^* that has not yet been placed in a batch. Starting with $j^* = [\lambda_{j^*}^{\min}, \lambda_{j^*}^{\max}]$, place into B_i the first c not yet scheduled jobs j for which $\lambda_{j^*}^{\min} \in [\lambda_j^{\min}, \lambda_j^{\max}]$ (or all of them if there are fewer than c).

For a set $\mathcal{I}$ of unit-sized jobs and a maximum batch size $c \in \mathbb{N}$, we denote by $GACb(\mathcal{I}, c)$ the number of batches returned by the GAC+ algorithm above. Similarly, let $GACp(\mathcal{I}, c)$ denote the minimal processing time returned by the GAC+ algorithm for this instance.

Theorem 1. *For any given set of unit-sized jobs $\mathcal{I}$ and for any given constant $c \in \mathbb{N}$, Algorithm GAC+ solves the problem OSP*, i.e., $GACb(\mathcal{I}, c)$ is the minimum number of batches required*

under the condition that a batch may not contain more than c jobs. Moreover, the cumulative batch processing time $GACp(\mathcal{I}, c)$ is minimal.

By slight abuse of notation, for a set $\mathcal{J}$ of jobs with arbitrary job sizes, let $GACb(\mathcal{J}, c)$ denote the number of batches returned by the GAC+ algorithm when replacing every job $j \in \mathcal{J}$ with s_j identical copies of unit-sized jobs. Similarly, let $GACp(\mathcal{J}, c)$ denote the minimal processing time returned by the GAC+ algorithm for this instance. With this notation, we obtain the following bounds:

$$b_r \geq |J_r^l| + b_r^C, \tag{9.9}$$
$$\text{with } b_r^C = GACb(J_r^s, \max_{m \in \mathcal{M}}\{c_m\}),$$
$$p_r \geq \sum_{j \in J_r^l} \lambda_j^{\min} + p_r^C, \tag{9.10}$$
$$\text{with } p_r^C = GACp(J_r^s, \max_{m \in \mathcal{M}}\{c_m\}).$$

Proof of Theorem 1. We follow the proof of Theorem 4 in [179], extending it to include the minimization of the cumulative batch processing time and adapting it to our variant of the algorithm. The proof is by induction over the number of jobs and the induction start with a single job is trivial.

Let us start with a simple observation about the minimum number of batches and the minimal batch processing time. For this, let $b(\mathcal{I}, c)$ denote the minimum number of batches required to schedule all jobs in $\mathcal{I}$ under the condition that a batch may not contain more than c jobs. Similarly, let $p(\mathcal{I}, c)$ denote the minimal cumulative batch processing time in any schedule of all jobs in $\mathcal{I}$. Then these two functions are monotonous in $\mathcal{I}$, i.e.:

$$b(\mathcal{I}, c) \geq b(\mathcal{I} \setminus \{j\}, c) \tag{9.11}$$
$$\text{and } p(\mathcal{I}, c) \geq p(\mathcal{I} \setminus \{j\}, c), \text{ for every } j \in \mathcal{I}.$$

For the induction step, consider a sequence of batches $\mathcal{B} = (B_1, B_2, \ldots, B_b)$ constructed by the algorithm GAC+ for $\mathcal{I}$, B_1 being the first batch constructed by the algorithm and $p \in \mathbb{N}$ being the cumulative batch processing time of $\mathcal{B}$. Let the label of B_1 be the job $i = [\lambda_i^{\min}, \lambda_i^{\max}]$, i.e., $\lambda_i^{\min} = \max_{j \in \mathcal{I}} \lambda_j^{\min}$ and the processing time of B_1 is equal to $\lambda_i^{\min}$. For the set of jobs $\mathcal{I} \setminus B_1$, the algorithm constructs the batch sequence $B_2, \ldots, B_b$ (see the definition of GAC+). By the induction hypothesis we know that $B_2, \ldots, B_b$ is optimal for $\mathcal{I} \setminus B_1$, i.e., $b(\mathcal{I} \setminus B_1, c) = |\mathcal{B}| - 1 = b - 1$ and $p(\mathcal{I} \setminus B_1, c) = p - \lambda_i^{\min}$. It thus suffices to show that there exists a batch sequence of minimal length and with minimal batch processing time that contains the batch B_1.

Let O_1 be the batch containing i in an optimal sequence of batches $\mathcal{O}$ and let us choose $\mathcal{O}$ such that the size of the intersection $|O_1 \cap B_1|$ is maximal. We will prove that $O_1 = B_1$.

First note that $i \in O_1$ implies that $\lambda_i^{\min} \in [\lambda_j^{\min}, \lambda_j^{\max}]$ for all jobs $j \in O_1$: $\lambda_j^{\min} \leq \lambda_i^{\min}$ since $\lambda_i^{\min}$ is maximal and $\lambda_i^{\min} \leq \lambda_j^{\max}$ since every job $j \in O_1$ needs to be compatible with i. Thus the processing time of batch O_1 is equal to $\lambda_i^{\min}$.

We now distinguish two cases: $|B_1| < c$ and $|B_1| = c$, where c is the maximum batch size. If $|B_1| < c$, the batch B_1 contains all neighbors of i in the compatibility graph G corresponding to the set of jobs $\mathcal{I}$. Since O_1 is a clique containing i, it follows that $O_1 \subseteq B_1$.

Then by monotonicity (as stated in equation (9.11)), we have

$$\begin{aligned} |\mathcal{B}| - 1 &= b(\mathcal{I} \setminus B_1, c) \\ &\leq b(\mathcal{I} \setminus O_1, c) = |\mathcal{O}| - 1 \\ \text{and } p - \lambda_i^{\min} &= p(\mathcal{I} \setminus B_1, c) \\ &\leq p(\mathcal{I} \setminus O_1, c) = p(\mathcal{I}, c) - \lambda_i^{\min}, \end{aligned}$$

which proves that $\mathcal{B}$ is optimal both in terms of the number of batches and in terms of the cumulative processing time.

For the case $|B_1| = c$, we assume towards a contradiction that there exists a job j so that

$$j = [\lambda_j^{\min}, \lambda_j^{\max}] \in B_1 \setminus O_1$$

This implies that there must also exist a job $k = [\lambda_k^{\min}, \lambda_k^{\max}] \in O_1 \setminus B_1$. (As before, $O_1 \subset B_1$ would imply that $\mathcal{B}$ is optimal. However, the job j could have been added to O_1 without having an impact on the number of batches or the batch processing time required by $\mathcal{O}$. This however is a contradiction to the choice of $\mathcal{O}$). From the definition of the algorithm, we know that B_1 consists of the first c jobs containing $\lambda_i^{\min}$. Therefore, $j < k$ and $\lambda_j^{\min} \geq \lambda_k^{\min}$. Moreover, as noted earlier, we know that $\lambda_i^{\min} \in k = [\lambda_k^{\min}, \lambda_k^{\max}]$ and thus $[\lambda_j^{\min}, \lambda_i^{\min}] \subseteq k$.

We then define $O_1' \coloneqq (O_1 \setminus \{k\}) \cup \{j\}$ and redefine the batch $O \in \mathcal{O}$ that contains j as $O' \coloneqq (O \setminus \{j\}) \cup \{k\}$. Both these batches fulfill the compatibility constraint for the processing times: O_1' does so because $\lambda_i^{\min}$ is contained in j and in all jobs in O_1 and O' does so because all jobs that are compatible with j are also compatible with k (If job s is compatible with j, this means that $s = [\lambda_s^{\min}, \lambda_s^{\max}] \cap [\lambda_j^{\min}, \lambda_i^{\min}] \neq \emptyset$, since $\lambda_i^{\min}$ is maximal among all minimal processing times. On the other hand, we already noted that $[\lambda_j^{\min}, \lambda_i^{\min}] \subseteq k$ and thus $s \cap k \neq \emptyset$, which means that s and k are compatible.) As for the processing times of the batches, both batches O_1 and O_1' have the processing time $\lambda_i^{\min}$ as they contain job i. For the batch O', we have replaced the job i with a job with smaller or equal processing time ($\lambda_i^{\min} \geq \lambda_k^{\min}$). Thus the processing time of batch O' is smaller or equal to the batch processing time of O. We have thus produced another optimal sequence of batches $\mathcal{O}' = \mathcal{O} \setminus \{O_1, O\} \cup \{O_1', O'\}$. However, $|O_1' \cap B_1| > |O_1 \cap B_1|$ which is in contradiction to the choice of $\mathcal{O}$. This finishes the proof. □

Overall Bound on the Number of Batches and the Minimal Cumulative Processing Time

Combining the previously established bounds, we obtain:

$$\begin{aligned} b &\geq \sum_{r=1}^{a} \max\left(|J_r^l| + \max(b_r^E, b_r^C), \max_{k \in [0, \frac{C}{2}]} b_r^E(k) \right), \\ p &\geq \sum_{r=1}^{a} \max\Bigg(\sum_{j \in J_r^l} \lambda_j^{\min} + \max(p_r^E, p_r^C), \\ &\qquad \max_{k \in [0, \frac{C}{2}]} \Big(\sum_{j \in J_r^1(k) \cup J_r^2(k)} \lambda_j^{\min} + p_r^E(k) \Big) \Bigg), \end{aligned}$$

where b_r^E is defined in equation (9.4), b_r^C in equation (9.9) and $b_r^E(k)$ in equation (9.6), the procedure to calculate p_r^E and $p_r^E(k)$ is described right after equation (9.5) and p_r^C is defined in equation 9.10.

Example

For the example instance, the simple bin packing bound from Equation (9.2) that distinguishes between small and large jobs already delivers the best possible result. Indeed, it establishes the optimal number of batches: for attribute 1 both small jobs 1 and 2 can be processed within one batch, i.e., $b_1 = 1$; for attribute 2, jobs 4 and 5 are large and require a batch of their own, and the remaining small job 3 also requires one batch, i.e., $b_2 = 3$; 4 batches are thus required in total. As for the cumulative batch processing time, we have:

$$\begin{aligned} p \geq & \max(\lambda_1^{\min}, \lambda_2^{\min}) + \lambda_4^{\min} + \lambda_5^{\min} + \lambda_3^{\min} \\ = & 12 + 10 + 11 + 11 = 44 \end{aligned}$$

which corresponds to the optimal value for this instance.

In order to exemplify the calculation of the refined LBs, let us consider a larger random instance with ten jobs and two machines. The respective machine capacities are $c_1 = 18$ and $c_2 = 20$; the relevant job parameters are given as follows:

j	1	2	3	4	5	6	7	8	9	10
$\mathcal{E}_j$	M_1	M_1		M_1	M_1		M_1	M_1		M_1
	M_2	M_2	M_2		M_2	M_2	M_2		M_2	M_2
$\lambda_j^{\min}$	11	10	19	19	10	19	11	50	19	11
$\lambda_j^{\max}$	11	50	19	19	11	50	50	50	19	50
s_j	18	16	17	2	6	19	11	11	4	14
α_j	2	2	2	1	2	2	2	2	1	1

The values of the LBs for the number of batches required and the cumulative batch processing times are summarized in Table 9.1. We explain their calculation in what follows. The sets of large jobs are $J_1^l = \emptyset$ and $J_2^l = \{1, 2, 3, 6\}$, we thus need 4 batches for the large jobs of attribute 2 and none for attribute 1. The processing times for large batches are given by the minimal processing times of the large jobs and contribute $11 + 10 + 19 + 19 = 59$ to the cumulative batch processing time.

For the processing time of small jobs, we exemplify the calculation of the bounds in equation 9.4 based on eligible machines for attribute 1 and the one of the bound based on compatible processing times for attribute 2. For attribute 1, we have three small jobs (4, 9, and 10) of which job 4 can only be processed on machine 1 and job 9 only on machine 2. Two different batches are thus required for these jobs. Since the cumulative remaining machine capacity ($2 \cdot \max\{c_m\} - (s_4 + s_9) = 40 - (2 + 4) = 34$) is sufficient to accommodate job 10 with $s_{10} = 14$, these two batches suffice. In this case, the runtime of the two batches

is given by the minimal runtime of the two jobs 4 and 9, and is equal to 38 in total. For the bounds $b_1(k)$

As for attribute 2, the small jobs are 5, 7, and 8. Their respective intervals of possible processing times are $[10, 11]$, $[11, 50]$ and $[50, 50]$. In the compatibility graph, there is therefore an edge between nodes 5 and 7 and between nodes 7 and 8. To follow algorithm GAC+, we sort the list of jobs in decreasing order of their minimal processing times: $(8, 7, 5)$. A first batch with a processing time of 50 is created for job 8. The remaining capacity in this batch is $20 - 11 = 9$ (assuming that it is assigned to the machine with maximal capacity). We thus proceed in the list of jobs and add 9 of the 11 units of job 7 to this batch. For the remaining 2 units of job 7, a new batch with processing time 11 is created. We can add the entire job 5 to this batch. In total, two batches with a cumulative processing time of 61 are needed for the small jobs of attribute 2.

The bounds in equations (9.6) and (9.7) yield the same results. For attribute 1, the maximum values is obtained for $k = 0, 1$ and 2 where we have $b_1(k) = 2$ and $p_1(k) = 38$. For attribute 2, the maximum values are obtained for $k = 3$ where we have $b_2(3) = 6$ and $p_2(3) = 120$. To be more precise:

$$\begin{aligned} b_2(3) =& |\underbrace{\{1,6\}}_{=J_2^1(3)}| + |\underbrace{\{8\}}_{=J_{2,1}^1(3)}| + |\underbrace{\{3\}}_{=J_{2,2}^1(3)}| \\ &+ \left\lceil \frac{s_2 + s_5 + s_7 - 10}{20} \right\rceil \\ =& 4 + \left\lceil \frac{33 - 10}{20} \right\rceil = 6. \end{aligned}$$

Table 9.1: Lower bounds for the number of batches and the cumulative batch processing time for an example instance with 10 jobs. Best results per attribute are highlighted in bold font.

		attribute		
	bound type	1	2	$\sum$
number of batches	simple (9.1)	1	5	8
	b_r^E(9.4)	**2**	**6**	
	b_r^C(9.9)	1	**6**	
	$\max b_r(k)$ (9.6)	**2**	**6**	
processing times	large jobs + p_r^E (9.5)	**38**	119	158
	large jobs + p_r^C (9.10)	19	**120**	
	$\max \sum \lambda_j^{\min} + p_r(k)$ (9.7)	**38**	**120**	

9.1.2 Lower Bound on the Setup Costs

Using the LB on the number of batches obtained in Section 9.1.1, a LB on the setup costs can be computed by solving a Vehicle Routing Problem (VRP). This is done by creating one location per batch required and one vehicle per machine. Machine eligibility restrictions correspond to restrictions on which vehicles may visit which locations and the cost of

traveling between locations is given by the setup costs between batches.

The VRP [127] is a classical combinatorial optimization problem that generalizes the Traveling Salesperson Problem (TSP). Given a fleet of vehicles and a set of customers at different locations, the task is to find an optimal, i.e., cost-minimal, set of routes such that all customers are visited exactly once.

In this section, we first introduce the additional notation, then we outline how to convert an OSP instance to a VRP one, and, finally, we provide an example.

Preliminaries

In order to create a VRP instance from an OSP instance, the matrix of setup costs SC first needs to be checked and possibly adapted. Indeed, it might be the case that the setup costs do not fulfill the triangle inequality. If this is the case, it can be advantageous, in terms of minimizing the setup costs, to introduce additional batches and one cannot derive a LB on the setup costs by using the minimum number of batches required. Thus, for every pair of attributes $r, s \in \mathcal{A}$, we calculate the adapted setup costs SC$'$ as follows:

$$\text{SC}'(r, s) = \min_{t \in \mathcal{A}} \text{SC}(r, t) + \text{SC}(t, s),$$

i.e., $\text{SC}'(r, s) = \text{SC}(r, s)$ if the triangle inequality is fulfilled for attributes r and s.

From the procedure described in Section 9.1.1, the minimum number of batches b_r required per attribute $r \in \mathcal{A}$ is retrieved. Moreover, the set of eligible machines for each of these batches is stored and denoted by $\mathcal{E}_{i,r} \subseteq \mathcal{M}$ for $r \in \mathcal{A}$ and $1 \leq i \leq b_r$.

Creation of the Vehicle Routing Problem Instance for an Oven Scheduling Problem Instance

The VRP instance is constructed in the following way.

- One location is created per batch required: A node labeled $B_{r,i}$ is introduced for every attribute $r \in \mathcal{A}$ and $1 \leq i \leq b_r$.
- One vehicle v_m per machine $m \in \mathcal{M}$ is created.
- For every vehicle v_m, a depot D_m is created. The vehicle v_m must start and end its route at its depot D_m. The depot D_m is connected to those locations $B_{r,i}$ for which it holds that $m \in \mathcal{E}_{i,r}$.
- An edge is present between every pair of distinct locations. The cost of traveling from location $B_{r,i}$ to location $B_{s,j}$ is equal to $\text{SC}'(r, s)$.
- The cost of traveling from depot D_m to location $B_{r,i}$ (if the edge is present) is equal to $\text{SC}'(\alpha_m^{\text{initial}}, r)$, where $\alpha_m^{\text{initial}}$ is the initial attribute of machine $m \in \mathcal{M}$. No costs are incurred when traveling from a location to a depot.

An optimal solution of the resulting VRP instance, i.e., a cost-minimal route that visits all locations exactly once, clearly is a LB on the setup costs of the considered OSP instance.

Example of a Vehicle Routing Problem Instance and its Optimal Solution

As an example, the VRP instance corresponding to the OSP instance described in Section 8.3.1 is given in Figure 9.1. Note that the setup costs for this example fulfill the triangle inequality and thus SC $=$ SC′. From Section 9.1.1, one obtains that at least one batch is necessary for attribute 1 and three batches are necessary for attribute 2. We have: $b_1 = 1$ and $\mathcal{E}_{1,1} = \{M_1\}$ as well as $b_2 = 3$ with $\mathcal{E}_{2,1} = \{M_2\}$, $\mathcal{E}_{2,2} = \{M_1, M_2\}$ and $\mathcal{E}_{2,3} = \{M_2\}$. A VRP instance with two vehicles, two depots, and four locations is thus created. The following routes correspond to an optimal solution with cost $= 3$: D_1–$B_{1,1}$–D_1 for vehicle v_1 (cost 2), D_2–$B_{2,1}$–$B_{2,3}$–$B_{2,2}$–D_2 for vehicle v_2 (cost 1).

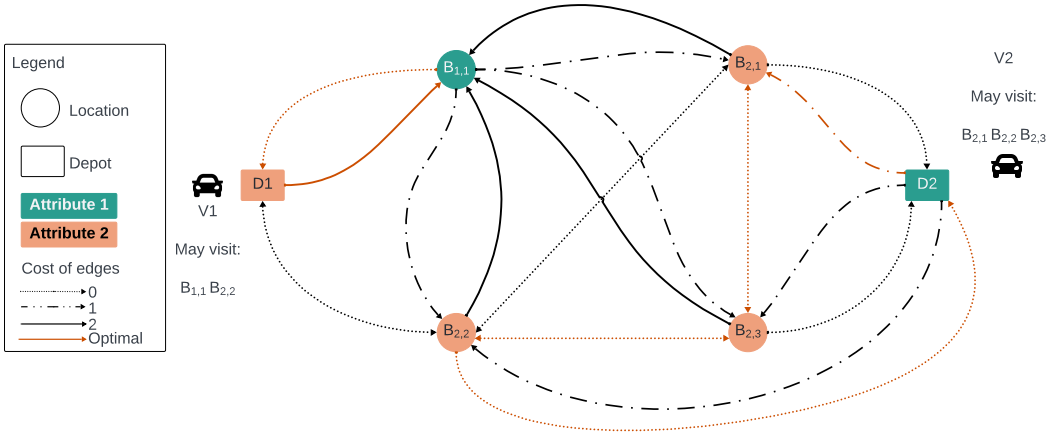


Figure 9.1: Example of the Vehicle Routing Problem (VRP) instance used to compute a Lower Bound (LB) on the setup costs for the Oven Scheduling Problem (OSP) instance described in Section 8.3.1. An optimal route for the two vehicles of cost 3 is given by following the edges marked in green.

9.1.3 Lower Bound on the Number of Tardy Jobs

We compute the LB on the number of tardy jobs by solving a Minimum Cost Flow Problem (MCFP). This is a network optimization problem where each edge in the graph is assigned a unit cost for transporting material across it and a maximal possible flow. The objective is to determine the flow (i.e., which amount to send across each edge and this amount can be a real number) with the least total cost while satisfying specific constraints [118]. The MCFP includes special nodes, referred to as supply nodes and demand nodes: Supply nodes add a positive amount (supply) to the flow, which could represent production at that node; Demand nodes subtract a negative amount (demand) from the flow, which could represent consumption at that node. The flow in the network must adhere to the flow conservation rule: At each node, the total flow out of the node minus the total flow into the node must equal the supply or demand at that node. The MCFP aims to transport material from supply nodes to demand nodes in such a way that the total transportation cost is minimized while ensuring the flow conservation rule is met at every node.

Dealing with tardiness using MCFP has been put forward also by Baptiste, Jouglet, and Savourey [35], however, their approach is different as they divide the jobs into unary jobs and address a basic version of batch scheduling problem. This approach could not be directly applied to the OSP because, given the size of our instances, the problem size could

explode to intractable sizes.

In this section, we first introduce the additional notation and how to convert an instance of the OSP to one for the MCFP together with an example. Finally, we report an alternative way to calculate a LB on t.

Preliminaries

The minimal setup time for each job $j \in J$ is indicated as $\min_j^{\text{ST}}$, more formally:

$$\min_j^{\text{ST}} = \min_{r \in \mathcal{A}} \text{ST}(r, \alpha_j)$$

where $\alpha_j \in \mathcal{A}$ is the attribute of job j.

The minimal setup and processing time indicated as π_j is calculated for each job $j \in J$ as

$$\pi_j = \lambda_j^{\min} + \min_j^{\text{ST}}$$

For each machine $m \in \mathcal{M}$, each machine availability interval $[\sigma_{m,i}^{\text{start}}, \sigma_{m,i}^{\text{end}}]$ is divided into sub-intervals of equal length. The length of each sub-interval is given by the minimal π_j considering the jobs compatible with that interval, the last sub-interval possibly being shorter than the others. The determination of the minimal π_j takes into account release times, minimum processing times, sizes, and machine eligibility. These sub-intervals are denoted by the triple (m, i, k), with m being the machine, i the interval, and k the sub-interval. The start and end of each sub-interval are given by $[\beta_{m,i,k}^{\text{start}}, \beta_{m,i,k}^{\text{end}}]$.

Creation of the Minimum Cost Flow Problem Instance for an Oven Scheduling Instance

Given the previously introduced, notation the MCFP instance is constructed as follows.

- Each job j is a supply node with a supply equal to s_j, that is the size of job j.
- Each sub-interval (m, i, k) is considered as a demand node with a demand equal to C_m, that is the capacity of machine m. In our MCFP instance, nodes that are neither supply nor demand nodes are not present.
- Regarding the conservation of flow, the introduction of subintervals generates more capacity than required, resulting in an imbalance between the total job sizes and the demand. To address this, we introduce a dummy node to absorb the excess capacity and ensure balance.
- There is an edge between job j and sub-interval (m, i, k), if the jobs can start within the sub-interval $[\beta_{m,i,k}^{\text{start}}, \beta_{m,i,k}^{\text{end}}]$ and it fits within the interval $[\sigma_{m,i}^{\text{start}}, \sigma_{m,i}^{\text{end}}]$; more formally:

$$\tau_j^{\text{start}} - \min_j^{\text{ST}} \leq \beta_{m,i,k}^{\text{start}}$$
$$\rho_{j,m,i,k} + \lambda_j^{\min} \leq \sigma_{m,i}^{\text{end}}$$

  where $\rho_{j,m,i,k}$ is the earliest possible start of job j within the sub-interval (m, i, k), that is

$$\rho_{j,m,i,k} = \max(\tau_j^{\text{start}}, \beta_{m,i,k}^{\text{start}} + \min_j^{\text{ST}})$$

- If there is an edge between job j and sub-interval (m, i, k), the maximum flow is equal to s_j, that is the size of the job. The unit cost of this edge is 0 if the job is not tardy, whereas it is equal to $1/s_j$ if the job is late, that is $\rho_{j,m,i,k} + \lambda_j^{\min} > \tau_j^{\text{end}}$. If the entire job j is assigned to the sub-interval (m, i, k), then cost is either 0, if the job is not tardy, or 1, if the job is tardy.

An optimal solution of the resulting MCFP instance, i.e., a cost-minimal flow that allows all jobs to be linked to a sub-interval, is a LB on the MCFP of the considered OSP instance.

Example of an Minimum Cost Flow Problem Instance and its Optimal Solution

We now consider the example reported instance and calculate a LB on the number of tardy jobs.[1] For each job $j \in \mathcal{J}$, we retrieve its $\min_j^{\text{ST}}$ and thereof calculate its π_j.

j	$\min_j^{\text{ST}}$	π_j
1	1	13
2	1	11
3	3	14
4	3	13
5	3	14

For each interval, we detect which jobs are compatible and report the minimal π_j, which is then used to compute the sub-intervals.

m	i	compatible jobs	min π_j
1	1	(1,2,3)	11
1	2	(1,2,3)	11
2	1	(3,4,5)	13
2	2	(3,4,5)	13

Subsequently, the sub-interval nodes have the following start and end times.

m	i	k	$\beta_{m,i,k}^{\text{start}}$	$\beta_{m,i,k}^{\text{end}}$
1	1	1	0	11
1	1	2	11	22
1	1	3	22	33
1	1	4	33	37
1	2	1	40	51
1	2	2	51	60
2	1	1	2	15
2	1	2	15	28
2	1	3	28	30
2	2	1	40	53
2	2	2	53	60

[1] In the example demonstration, we omit to report the dummy job.

Eventually, the edges connect each job j with the sub-intervals they can be accommodated into. For each edge, we report the earliest possible start of the job within that sub-interval, the unit cost, and the flow value w.r.t. the optimal solution.

m	i	k	j	$\rho_{j,m,i,k}$	unit cost	flow
1	1	1	1	7	0	1
1	1	1	2	5	0	2
1	1	1	3	5	0	1
1	1	2	1	12	0.5	0
1	1	2	2	12	0.5	0
1	1	2	3	14	0.5	0
1	1	3	1	23	0.5	0
1	1	3	2	23	0.5	0
1	1	3	3	25	0.5	0
1	2	1	1	41	0.5	1
1	2	1	2	41	0.5	0
1	2	1	3	43	0.5	0
2	1	1	3	5	0	1
2	1	1	4	6	0	0
2	1	1	5	8	0	2
2	1	2	3	18	0.5	0
2	1	2	4	18	0.33	3
2	1	2	5	18	0.5	0
2	2	1	3	43	0.5	0
2	2	1	4	43	0.33	0
2	2	1	5	43	0.5	0

Figure 9.2 illustrates the optimal solution of the MCFP for the given example. Jobs are depicted as circles, while sub-intervals are represented as rectangles. All potential edges are displayed, with those selected in the optimal solution highlighted: light blue edges indicate no associated cost, whereas yellow edges signify a cost, indicating that tardiness has occurred. The optimal cost for the MCFP is $\lceil 1.49 \rceil = 2$, thus 2 jobs are always late.

An Alternative Way to Calculate the Lower Bound on the Number of Tardy Jobs

An alternative, yet much simpler way, to calculate a LB on the number of tardy jobs is to schedule every job independently in a batch of its own on the first eligible machine available and compute the resulting completion time. If the completion time is after the job's latest end date, this will be a tardy job in every solution.

However, since it consistently produces lower-quality LBs compared to the method based on MCFP, its results are omitted from the experimental analysis (Section 9.3), and this section is included solely for completeness.

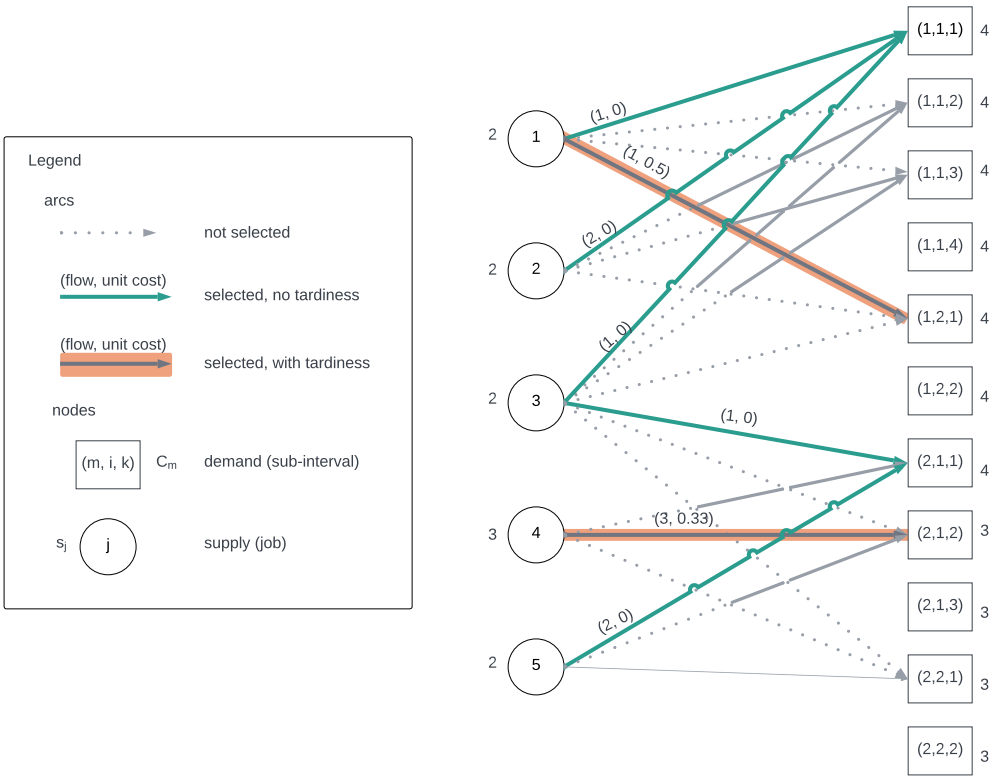


Figure 9.2: Example of the Minimum Cost Flow Problem (MCFP) instance used to compute a Lower Bound (LB) on the number of tardy jobs for the Oven Scheduling Problem (OSP) instance described in Section 8.1. The cost-optimal flow is achieved by selecting the marked edges with the respective flows. This corresponds to 2 tardy jobs.

9.1.4 Computational Complexity of the Lower Bound Calculations

The LBs on the total processing time as well as the minimum number of batches required can be calculated in polynomial time. The (refined) bin packing bounds can clearly be computed in polynomial time, the respective formulas being provided explicitly. Moreover, the refined bound for small jobs based on compatible job processing times is calculated using the polynomial time algorithm GAC+. This algorithm has the same computational complexity as the GAC algorithm presented by Finke et al. [179].

The LB on the setup costs is calculated by solving a corresponding VRP instance. The VRP is known to be NP-hard since it generalizes the TSP [250]. It can therefore not be expected that this LB can be computed efficiently for arbitrarily large instances of the OSP. However, note that the size of the VRP instances is determined by the minimum number of batches required for the corresponding OSP instance. As a consequence, this number might be significantly smaller than the number of jobs.[2]

The LB on the tardy jobs is computed by solving a MCFP instance. The MCFP can be solved in polynomial time using the network simplex algorithm [375].

[2]For the benchmark instances considered in Section 9.2, the number of nodes in the corresponding VRP networks is, on average, equal to 36.69% of the number of jobs (ranging from 3.89% to 90.00%). All generated VRP instances can be solved to optimality in less than one second on average. Detailed timing results are reported in Table 9.2.

9.2 Experimental Setup

In this section, we detail the experimental setup used to evaluate the LBs and to integrate them into existing solution methods. We first describe the benchmark instances used (Section 9.2.1), and then report the computing environment and implementation details (Section 9.2.2).

9.2.1 Instances

We consider the set of instances introduced by M. Lackner et al. [275] (Section 3.4). In addition, we generate 40 new instances with up to 500 jobs while preserving the same structural characteristics as the original set (i.e., number of machines and attributes). These new instances serve to evaluate scalability under more demanding conditions.

In total, our benchmark set comprises 120 instances, which vary along three key dimensions: the number of jobs (10, 25, 50, 100, 250, and 500), the number of machines (2 or 5), and the number of job attributes (2 or 5). In analyses that require aggregation by instance size, we categorize the benchmark as follows: 40 small instances with 10 or 25 jobs, 40 medium instances with 50 or 100 jobs, and 40 large instances with 250 or 500 jobs.

9.2.2 Technical Details

All experiments are executed on a machine equipped with 2x Intel Xeon CPU E5–2650 v4 (12 cores @ 2.20GHz, without hyperthreading). The calculation of the theoretical LBs and the construction heuristic are implemented in C#. The VRP and MCFP models used to determine bounds on the setup costs and the number of tardy jobs are implemented using `CP-SAT` from `OR-Tools` 9.11 in C#. For both models, the solver was run until an optimal solution was found. Table 9.2 reports a complete breakdown of the timings.

Table 9.2: Breakdown of timings for the calculation of the lower bounds.

Calculation	Avg (s)	Std (s)
All (obj)	5.39	12.81
— Processing time (p) and batch number bound (b)	2.55	5.99
— Setup cost bound (sc)	0.32	0.31
— Tardy jobs bound (t)	2.47	10.22
Construction heuristic	2.81	10.38

For the exact methods, we considered the following configurations:

- "cpopt": the interval-variable model with representative jobs, solved using CP Optimizer (CPLEX Studio 22.11);
- "mzn-gurobi": the MiniZinc model with batch positions, solved with Gurobi 10.0.1 via MiniZinc 2.8.2;
- "cpopt-WS" and "mzn-gurobi-WS": the corresponding variants warm-started with the construction heuristic, as in the original work of M. Lackner et al. [275].

Each method is run with a timeout of 1 hour per instance. The original code is available on Zenodo [274].

9.3 Results

In this section, we first report the results of the experiments performed to evaluate the quality of the theoretical LBs (Section 9.3.1). Then, we showcase how these bounds can be applied to enhance existing solution methods (Section 9.3.2).

9.3.1 Evaluation of Lower Bounds

Our objective is to assess the tightness of the proposed LBs, both for the overall cost (obj) and for the individual components (t, p, sc) as well as the number of batches (b). To this end, we run the exact models to optimize either the overall cost or one objective component at a time across the entire benchmark set. We distinguish between instances where the optimal solution cost is known and those where it is not.

For instances where the optimum is known, we compute the LB relative gap w.r.t. this optimum. The results are summarized in Figure 9.3. For the combined objective obj, the gap is below 10% in 60.98% of the instances (25 out of 41). Among the components, the tightest bounds are obtained for tardiness t (below 10% in 75.31% of the instances, 61 out of 81), while more room for improvement remains for processing time p (below 10% in 60.47% of the instances, 26 out of 43), batches b (below 10% in 47.76% of the instances, 32 out of 67), and setup costs sc (below 10% in 53.45% of the instances, 31 out of 58). These findings suggest that the bounds are relatively tight also for larger instances where the optimum is unknown.

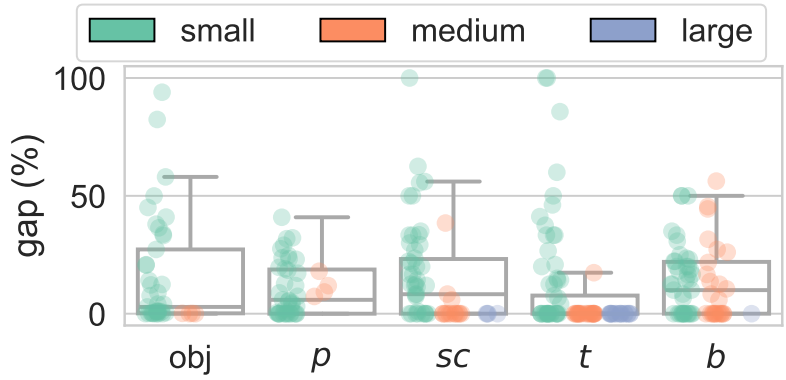


Figure 9.3: Percentage gap between the known optimum and the calculated lower bounds for the overall cost (obj) and the individual components (p, sc, t) as well as the number of batches (b).

For instances where the optimum is not known, we compare the problem-specific LBs against the best dual bounds obtained by CP Optimizer and Gurobi ("cpopt", "cpopt-WS", "mzn-gurobi", "mzn-gurobi-WS"). For each objective as well as the number of batches, we record whether the theoretical bounds are tighter ("Theor.́"), the solver bounds are tighter ("Solv.́"), or the bounds are equal. Table 9.3 summarizes the results. Overall, the theoretical bounds dominate the solver bounds in most cases, particularly for larger instances. This

dominance is most pronounced for processing time p, setup costs sc, and the number of batches b, where the theoretical bounds consistently outperform the solver ones.

Table 9.3: Comparison of the quality of the theoretical lower bounds ("Theor.") and the best lower bounds retrieved by the exact methods ("Solv."), considering those instances where the optimal solution cost is not known. Totals are highlighted in bold font.

Lower bound Type		Small (#)	Medium (#)	Large (#)	**Total** (#)
obj	Theor.	0	22	40	**62**
	Solv.	3	14	0	**17**
	Equal	0	0	0	**0**
	Total	**3**	**36**	**40**	**79**
p	Theor.	1	30	40	**71**
	Solv.	0	6	0	**6**
	Equal	0	0	0	**0**
	Total	1	36	40	**77**
sc	Theor.	1	25	36	**62**
	Solv.	0	0	0	**0**
	Equal	0	0	0	**0**
	Total	**1**	**25**	**36**	**62**
t	Theor.	0	1	21	**22**
	Solv.	0	10	0	**10**
	Equal	0	6	1	**7**
	Total	**0**	**17**	**22**	**39**
b	Theor.	0	7	39	**46**
	Solv.	0	5	0	**5**
	Equal	0	2	0	**2**
	Total	**0**	**14**	**39**	**53**

9.3.2 Application of Lower Bounds

We now investigate how these bounds can support exact methods. First, we use the LBs and the construction heuristic as a quick estimate for the optimal solution, then analyze the impact of incorporating theoretical LBs into exact solvers, and finally assess solution quality with respect to the derived LBs.

Quick Estimate of the Optimal Solution Cost

The construction heuristic reported in Section 3.4 and used to warm start the exact solvers always produces a feasible schedule for our benchmark instances, and its solution cost therefore serves as an upper bound on the optimal cost.

To estimate the optimal solution cost, we compute the relative gap between the theoretical LB and the cost of the solution generated by the heuristic. Figure 9.4 shows the results for the combined objective obj. The heuristic rarely produces optimal solutions (1 out of 120 instances), and in many cases, the gap between this upper bound and the LB is large. Nonetheless, for 37 instances across all sizes the relative gap is below 1%, including some

large instances for which no solver could prove optimality. For 59 instances, this gap is below 10% (i.e., 22 instances with a gap within 1% and 10%).

These findings indicate that, within a short computation time (2.81 s on average), the construction heuristic combined with the theoretical LBs provides tight estimates of the optimal solution cost for a notable share of the benchmark set, and reasonably good estimates for nearly half of the instances.

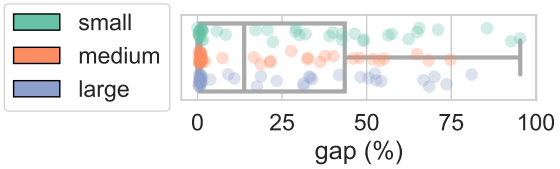


Figure 9.4: Relative bound gap (%) between the upper bound found by the construction heuristic and the theoretical Lower Bound (LB) per instance considering the overall cost (obj).

Aiding Exact Methods with Theoretical Bounds

A recommended practice for building efficient exact models is to tightly restrict and bound the domains of variables (see, e.g., the MiniZinc guide[3] on efficient modeling). Tighter bounds improve algorithmic efficiency by narrowing the search space, enabling faster convergence or earlier identification of infeasible regions. When solving the OSP with the exact methods, the our LBs can be computed in a preprocessing step and passed to the model as input. In this way, the variables corresponding to each objective component and to the aggregated objective can be bounded from below. Providing the exact methods with the LBs could help the solvers in two ways: first, the bounds could help close the optimality gap, thus speeding up optimality proofs; second, the bounds could help prune the search space and allow the solvers to find better solutions within the time limit.

We now evaluate whether supplying exact methods with these bounds improves their performance. Specifically, we conduct experiments where the objective function and its components are bounded from below by the calculated theoretical values. Table 9.4 summarizes the results by solution method. The table reports: the number of instances for which the optimal solution, when known, was reached ("optimal"); the number of instances for which a feasible solution was found ("solved"), the number of instances for which the method could prove optimality ("proven opt"); the number of instances for which the best solution was found ("best"); the number of instances for which the best LB could be found ("best lower bound"); the average run time ("avg rt") and its standard deviation ("std rt") in seconds. Note that for the number of best solutions found and of best LBs found, the comparison is made among a single solution method, i.e., comparing results obtained when no non-trivial bounds are provided ("none") and when lower bound ("LB") are provided. The statistics regarding runtime are calculated for the subset of instances for which the respective solution method could prove optimality when it was not provided with bounds (meaning that instances for which the time-out was reached are not included). The majority of solution methods, namely "mzn-gurobi", "cpopt", and "cpopt-WS", demonstrate improved performance and solution quality when LBs are

[3] https://www.minizinc.org/doc-2.5.5/en/efficient.html. Accessed 2025–09–27.

incorporated. For "mzn-gurobi-WS"[4], one less instance could be solved when LBs are provided, but overall better solutions and better LBs can be found. Moreover, for all analyzed solution methods, the presence of bounds facilitates the discovery of improved LBs by the commercial solvers, thus contributing to closing the optimality gap.

Table 9.4: Comparison of the results obtained with exact methods with and without the inclusion of bounds. Numbers in brackets indicate the improvement. Best results per solution method and performance parameters are highlighted in bold font.

Solution method	Bounds incl. in model	Optimal (#)	Solved (#)	Proven opt. (#)	Best (#)	Best LB (#)	Avg rt (s)	Std rt (s)
mzn-gurobi	None	40	64	31	52	41	429.5	860.6
	LB	**41** (+1)	**77** (+13)	**36** (+5)	**74** (+22)	**68** (+27)	**108.2**	203.2
mzn-gurobi-WS	None	**41**	**89**	34	57	41	764.3	1217.9
	LB	**41** (+0)	88 (-1)	**35** (+1)	**86** (+29)	**82** (+41)	**354.5**	862.2
cpopt	None	**39**	**114**	28	73	28	**18.4**	34.1
	LB	**39** (+0)	**114** (+0)	**33** (+5)	**86** (+13)	**120** (+92)	23.1	63.7
cpopt-WS	None	38	**120**	28	77	28	**17.6**	30.0
	LB	**40** (+2)	**120** (+0)	**33** (+5)	**95** (+18)	**120** (+92)	20.3	50.0

Table 9.5 offers a comprehensive comparison of overall best results. The inclusion of bounds enabled the methods to deliver 3 new optimality proofs, to find 27 better solutions, and to improve the dual bounds for 72 instances. Additionally, the computational time reduces when bounds are utilized compared to when they are not.

Table 9.5: Overall comparison of the best results per instance achieved with exact methods without the inclusion of bounds and with the inclusion of bounds.

Bounds included	Optimal (#)	Solved (#)	Proven opt. (#)	Best (#)	Best LB (#)	Avg rt (s)	Std rt (s)
No	41	120	38	75	43	486.8	1075.6
Yes	41 (+0)	120 (+0)	41 (+3)	102 (+27)	115 (+72)	91.6	212.8

Measuring Solution Quality

In practical settings, it is often sufficient to obtain solutions of good quality within a short time, even if they are not provably optimal. To benchmark such solutions, we use the best available LBs. For each instance, we compare the best solution obtained by the exact methods with the best LB, chosen as the maximum among the theoretical bounds and those retrieved by the exact methods. The relative gap between these two values is then calculated. Results are summarized in Table 9.6. Overall, 41 instances achieve a gap of 0%. Among the remaining instances, the average gap is 15.40% with a standard deviation of 20.07% and a median of 3.97%. As expected, many small instances are solved to optimality. For medium and large instances, the methods deliver solutions with a relative gap of at most 1% in 50% of the cases (40 out of 80). For most of the remaining instances, the gap exceeds 5%, indicating room for improvement in both solution quality and bound strength.

[4]The warm-start data provided to Gurobi only contains values for a subset of the decision variables. The

Table 9.6: Number of instances by percentage gap between the best solution found by any of the exact methods and the best lower bounds.

		Gap (%)			
	total (#)	=0 (#)	0 to 1 (#)	1 to 5 (#)	> 5 (#)
Small	40	37	0	0	3
Medium	40	4	18	1	17
Large	40	0	18	3	19
Total	**120**	**41**	**36**	**4**	**39**

9.4 Concluding Remarks

In this concluding section, we distill the content of the chapter (Section 9.4.1), discuss limitations (Section 9.4.2), and outline future directions (Section 9.4.3).

9.4.1 Content

In this study, we introduced a procedure for calculating theoretical LBs for the Oven Scheduling Problem which can be calculated within a couple of seconds even for large instances. The experimental evaluation demonstrated their quality, in particular for large instances with 50 jobs or more. Moreover, we showcased their practical utility by incorporating them into exact methods, where our LBs can help to find better solutions, deliver more optimality proofs, and find high-quality solutions in a shorter time. Together with the upper bounds delivered by a construction heuristic, the LBs can also be used to calculate a quick estimate of the optimal solution cost. For roughly one-third of the benchmark set, the relative gap between this upper and the LB is less than 1%. Such an estimate can be useful in practical settings, since it can help to decide whether it pays off to use more time-consuming solution methods to solve a specific instance or whether the solution provided by the construction heuristic is already close enough to the optimum.

9.4.2 Limitations

Despite the encouraging results obtained on the considered benchmark set, we acknowledge the following limitations:

- The approach has been evaluated on a single use case. While this scenario reflects real-world applications, other problem variants or operational settings may exhibit different structural properties and deserve dedicated investigation.
- Part of the computational effort stems from reducing the original problem to well-known, yet still NP-hard, subproblems. In particular, the size of the derived VRP instances is bounded by the number of batches, which may be substantially larger than in the benchmark instances considered here, potentially leading to increased solution times.

solver thus needs to complete the partial solution and, for "mzn-gurobi-WS", fails to do so for large instances.

- Although the proposed lower bounds play a key role in guiding and strengthening the solution process, they should not be regarded as a substitute for effective solution methods. Their practical value is instead reflected in the ability to obtain improved best-known solutions, significantly reduce optimality gaps, and provide three new proofs of optimality for previously open instances, while high-quality feasible solutions remain essential for tackling larger or more challenging instances.

9.4.3 Future Work

In future work, we aim to:

- **Bound-aware local search:** Extend the use of LBs within metaheuristics by designing adaptive mechanisms, like dynamically reweighting neighbourhoods, pruning moves whose best-case gain cannot close the bound gap, and using gap dynamics to trigger intensification/diversification.
- **Stronger and broader bounds:** Tighten the current LBs and generalise them to alternative objectives and machine environments.
- **Explainability:** Leverage LBs to inform decision-makers about solver outputs by providing near-optimality certificates (residual gap), gap-based stopping guidance, and qualitative attributions that clarify why further improvement is likely or unlikely.

Chapter 10

Solution Methods

While the Oven Scheduling Problem (OSP) has been addressed using Constraint Programming (CP) and Integer Linear Programming (ILP) methods (Section 3.4), a significant challenge remains: only a limited number of benchmark instances with more than 100 jobs can be solved optimally. This highlights the need for methods that improve solution quality, particularly since real-world instances often involve many jobs.

For this reason, we propose to apply Large Neighborhood Search (LNS) and Simulated Annealing (SA) to the OSP. More specifically, this chapter makes the following contributions:

- **Algorithms development**: Development and fine-tuning of LNS and SA approaches for the OSP, with a particular focus on the large benchmark instances for which existing exact methods fail to find optimal solutions within practical time limits.
- **Comparative analysis**: A comparative analysis against state-of-the-art results, showing that our approaches achieve superior performance on many large instances.
- **SA component analysis**: A thorough analysis of the components of SA, which emerged as the preferable solution method. We investigate the contribution of individual components (i.e., the metaheuristic scheme, the initial solution, and the neighborhood structures) in achieving high-quality results. In particular, an ablation study on the neighborhoods confirms that the MoveJob neighborhood is essential, as its removal can reduce performance by nearly 50%.
- **Dashboards**: user-friendly dashboards. We bridge the gap between research and practice by developing user-friendly dashboards that allow decision-makers to easily interact with our optimization tools and visualize results in the form of Gantt charts. Due to space limitations, these contributions are reported in Appendix D.
- **Robustness across problems**: A comparative analysis of SA on a related problem. Also in this case, due to space limitations, we report this contribution in Appendix D.

This chapter is structured as follows. Section 10.1 reports the solution methods, i.e., the problem-dependent component of the methods we employ. Section 10.2 describes the experimental setup, followed by a discussion of the results in Section 10.3. Finally, Section 10.4 provides concluding remarks.

10.1 Methodology

We now present the problem-dependent components of the methods proposed for the OSP, namely LNS (Section 10.1.1) and SA (Section 10.1.2). We also describe a Hill Climbing (HC) algorithm (Section 10.1.3), which later used as a baseline for benchmarking SA. General information on the algorithmic schemes can be found in Chapter 2.

10.1.1 Large Neighborhood Search

Solution Representation

The core solution representation concerns the scheduling of batches on the machines. Formally, this is encoded as a jagged array, where each row corresponds to a machine, each column indicates the batch position, and each entry specifies the set of jobs assigned to that batch, as well as precomputed information such as starting times, processing times, and waiting times. For efficiency, the multi-dimensional array is dimensioned exactly to fit the number of batches required by each machine in the current solution.

This representation is complemented by additional data structures that facilitate efficient computations required by the destroy operators. For example, we maintain references to batches with identical attributes.

Initial Solution

As initial solution, we use the construction heuristic introduced by M. Lackner et al. [275] and detailed in Section 3.4.

Destroy Operators

The destroy operators aim to select a subset of batches to be removed from the current solution, thereby allowing for modification of job grouping and scheduling order. The size of the sub-problem δ, that is the number of batches to be removed from the current solution, is specified to the destroy operator.

We propose three ways to select the δ batches to remove:

- Random Batches (RB): Select δ random batches across all possible machines.
- Random Batches from One Machine (RBM): Select δ random batches and remove them from the schedule of one random machine. This approach capitalizes on the potential for improved scheduling opportunities.
- Random Batches with the Same Attribute (RBA): Select δ random batches that share the same attribute. This method aims to exploit potentially compatible jobs to identify new batching possibilities.

Repair Operators

Once a set of batches is removed from the current solution, the repair mechanism solves the sub-instance that arises. Specifically, we employ the following strategy: Batches on machines not affected by the destruction are entirely fixed, preventing any addition of jobs

or modification of their schedules. For batches scheduled on machines involved in the destruction but not directly affected, jobs can be added (provided the processing time of the batch remains unchanged) and their start times adjusted while ensuring adherence to the precedence order.

As for the repair operator, we use the `CP Optimizer` model referred by M. Lackner et al. [275], as it is regarded as the most efficient exact method for the OSP. In the experimental evaluation, we utilize Optimization Programming Language (OPL) with two relative optimality tolerances: 5% (OPL-5) and 0.001% (OPL-001). This allows us to alternate between a strategy that prioritizes resolution velocity over ensuring optimality (OPL-5) and one that operates vice versa (OPL-001).

Subproblem Size

An essential parameter for LNS is the size δ of the sub-problem, which significantly influences the effectiveness of both the destroy and repair operators. In our system, the destroy operators are quick and straightforward to apply. However, the complexity of the repair operation depends on δ and the characteristics of the instances themselves.

Based on preliminary experiments, we set the minimum destruction size at $\delta_{\min} = 3$. This size can be solved in just a few seconds, ensuring rapid computation while still allowing for substantial improvements in the solution. However, if too many idle iterations occur without improvement, indicating a potential local minimum, δ can be increased up to $\delta_{\max} = 7$. Eventually, the repair operators are equipped with a cutoff of 2 minutes to maintain efficiency.

The update of δ is as follows: Initially, δ is set to its starting value $\delta_{\min}$. Its value is increased by 1 until reaching $\delta_{\max}$ whenever the previous improvement was more than $n_{\max}$ iterations ago. When a new improvement is found, δ is set to the initial value $\delta_{\min}$.

10.1.2 Simulated Annealing

Solution Representation

We represent a solution to the OSP as an array $\mathbf{s}$ sized to match the number of jobs. For each job j, the array stores the pair $s_j = (m, b)$, where m refers to the machine the job is assigned to and b is the position at which the job is processed on that machine. The pair (m, b) is referred to as batch.

Figure 10.1 presents an example of array $\mathbf{s}$. For instance, job 5 is assigned with the pair $(1, 2)$ meaning that it is processed by machine 1 in the second batch. Job 7 is assigned with the same pair, meaning that job 5 and job 7 are processed in the same batch.

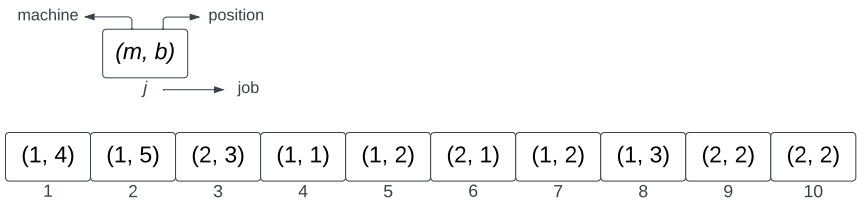


Figure 10.1: Example of array $\mathbf{s}$ for an instance with 10 jobs and 2 machines.

The schedule for each machine m is then deterministically constructed from $\mathbf{s}$ so that each batch is scheduled as early as possible, taking into account setup times, processing times, release dates, and the machine's availability intervals.

The proposed local search procedure permits the violation of a single constraint: some jobs can be scheduled after the scheduling horizon. This violation is accounted for in the objective function by adding the number of jobs scheduled beyond the scheduling horizon multiplied by a very high weight (such "big M"-constants are calculated based on the instance in such a way that an unfeasible solution will have obj > 1).

Although the vector $\mathbf{s}$ is sufficient to represent and construct the solution, it is supplemented by a set of redundant data structures that are used to speed up computations. For instance, the batch organization is represented using a multi-dimensional array. Figure 10.2 shows the same example as Figure 10.1, but from the perspective of batches: each row represents a different machine, each column indicates the batch position and the content specifies the jobs belonging to each batch (in the form of set). Note that, for efficiency reasons, this multi-dimensional array is dimensioned exactly to fit the number of batches required by each machine in the current solution. Additionally, even if not strictly necessary as the timings can be computes deterministically with an at-the-earliest procedure, we store also all the information related to the start times, end times, etc.

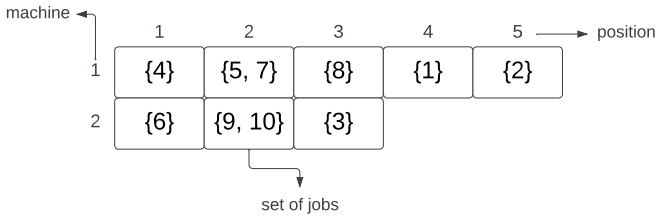


Figure 10.2: Example of batch organization using a jagged multi-dimensional array for an instance with 10 jobs and 2 machines.

Initial Solution

We consider two initialization strategies. The first leverages the construction heuristic introduced by M. Lackner et al. [275], as detailed in Section 3.4. The second strategy, designed to assess the contributions of the first one in the success of SA, involves generating a random initial solution. In this approach, jobs are randomly selected from the set of available ones and assigned to a randomly chosen feasible machine. Unlike the heuristic method, this strategy does not group compatible jobs into shared batches; instead, each batch contains exactly one job.

Neighborhood Relations

We propose a multi-neighborhood approach that makes use of four neighborhoods:

- Swap: The move Swap(m, b, p) swaps the batch in position b with the one in position p on machine m, where $b < p$ (for breaking symmetries).
- NewUnaryBatch: The move NewUnaryBatch(j, m, b) move creates a new batch at position b in machine m containing only job j. The batches already present in m

with a position equal to or greater to b are moved after the newly introduced batch. A prerequisite to this move is that the job j must be compatible with machine m. Furthermore, suppose the current batch of job j is composed only of j. In that case, its future position should be other than the current one (i.e., another machine m w.r.t. the current one or, if the machine is the same, then the position must be different).

- NewBatch: The move NewBatch$(\mathcal{J}, m_1, b, m_2, p)$ moves the set of jobs $\mathcal{J}$ (with $|\mathcal{J}| > 1$) from the batch at position b on machine m_1 to a new batch at position p on machine m_2. The batches already present in m_2 with a position equal to or greater to p are moved after the newly introduced batch. Machine m_2 must be compatible with all jobs in the set $\mathcal{J}$. Since these jobs were already in the same batch, their compatibility is ensured. The NewBatch neighborhood extends NewUnaryBatch.

- MoveJob: The move MoveJob(j, m, b) inserts job j in the existing batch at position b of machine m. A prerequisite to this move is that job j must be compatible with machine m and with the jobs already present in the existing batch. MoveJob differs from NewBatch and NewUnaryBatch because these two create new batches, while MoveJob acts exclusively on existing batches and eventually can end up reducing the number of batches (i.e., in the case where j originally belongs to a unary batch).

Figure 10.3 presents some visual examples of the above-described moves considering the same instance detailed in Figures 10.1 and 10.2. Figure 10.3a reports an example of Swap move. Specifically, it swaps the batch in position 2 with the batch in position 5 of machine 1. This results in changes to the processing order of the jobs but does not alter the composition of the batches. Consequently, considering the components of the objective function, this move will affect the component related to the number of tardy jobs (t) and the component related to setup costs (sc). Figure 10.3b illustrates an example of the NewUnaryBatch move. In this example, job 5, previously processed at position 2 on machine 1 together with job 7, is now processed on its own at position 3 on machine 2. This results in changes to both the composition of the batches and the processing order of the existing (unmodified) batches. Consequently, all three objective components are affected. Figure 10.3c reports an example of MoveJob move. We move job 5 that was previously processed at position 2 on machine 1; it is now processed together with job 3 at position 3 of machine 2. As observed for the NewUnaryBatch, this move also modifies both the composition of the batches and the processing order of the other batches, thus affecting all three objective components.

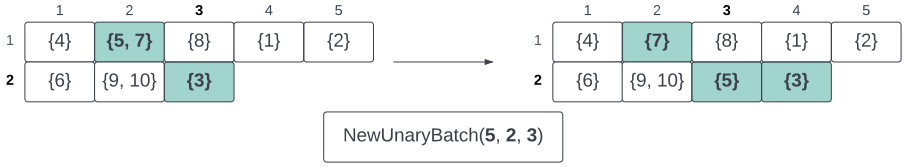
Although similar neighborhoods have been applied to related problem formulations (e.g., the Swap move is among the most commonly employed strategies when dealing with permutation-based problems [269, 379]), to the best of our knowledge, they have never been used in combination. This combined approach is a key factor in effectively solving the OSP and, more broadly, the parallel batch scheduling problem (Appendix D).

We now describe the selection of a move from the union multi-neighborhood composed as

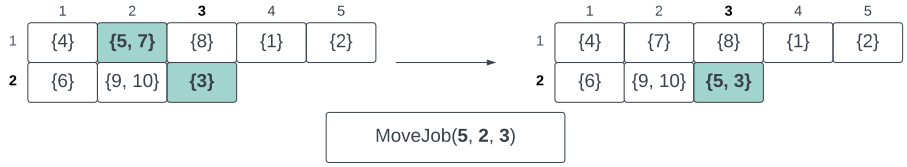
$$\text{Swap} \cup \text{NewUnaryBatch} \cup \text{NewBatch} \cup \text{MoveJob}$$

Four real-valued parameters are defined: σ_{S}, σ_{NUB}, σ_{NB}, and σ_{MJ}, with $\sigma_{\text{S}} + \sigma_{\text{NUB}} + \sigma_{\text{NB}} + \sigma_{\text{MJ}} = 1$. These parameters represent the probability of selecting each neighborhood and need to be tuned along with the other SA parameters. At each iteration, a specific

(a) SWAP(m, b, p) move.



(b) NEWUNARYBATCH(j, m, b) move.



(c) MOVEJOB(j, m, b) move.

Figure 10.3: Example of neighborhoods.

neighborhood is chosen based on the above-described fixed probability biases. Subsequently, the move from the selected neighborhood is extracted in a random fashion. For all the neighborhoods, the generation of a random move involves drawing attributes one at a time while adhering to specific preconditions. This approach is employed to guard against the possibility of entering infinite loops, particularly when an attribute has an empty domain.

10.1.3 Hill Cimbing

Simple local search methods, such as HC, are known to suffer from premature convergence as they get trapped in local optima. Nevertheless, they remain useful for benchmarking purposes, particularly when assessing the added value of more sophisticated metaheuristics. In this work, we consider random-based HC as a baseline to benchmark the performance of SA. To ensure a fair comparison, we adopt the same problem-dependent components used in SA, modifying only the underlying search strategy. This allows us to isolate the effect of the metaheuristic itself and understand whether the observed performance gains are attributable to the broader search framework or the problem-specific components.

10.2 Experimental Setup

Our objective is to evaluate whether SA and LNS can effectively overcome the limitations observed in exact methods approaches when applied to the OSP. This section outlines the experimental setup used in our analysis. We begin by introducing the methodology adopted to compare the different solution methods (Section 10.2.1). We then present details of the parameter tuning (Section 10.2.2), the benchmark instances used in the experiments (Section 10.2.3), and the technical implementation details (Section 10.2.4).

10.2.1 Comparison Methodology

In this part of the thesis, we compare the performance of four solution methods: ILP, CP, LNS, and SA. For LNS and SA, we further investigate the impact of their internal components to better understand their contribution to performance. In the case of SA, the comparison includes also another local search metaheuristic, HC.

To account for the stochastic nature of SA and LNS, each experiment is repeated 10 times per instance and use case, using different random seeds. For each instance and algorithm, the solution with the best objective value is selected for analysis.

We use the Friedman test with a significance level of $\alpha = 0.05$ to assess whether observed performance differences among the algorithms are statistically significant. If the null hypothesis is rejected, we perform post hoc pairwise comparisons using the Wilcoxon signed-rank test with Holm's correction ($\alpha = 0.05$). To complement the statistical analysis, we visualize the relative rankings of the algorithms using Critical Difference (CD) plots. In addition, we evaluate performance using relative gaps, computed with respect to a baseline solution (depending on the analysis case).

10.2.2 Prameter Tuning

In this section, we outline the tuning procedures we conducted to set the parameters of LNS, SA, and HC. All tuning procedures are conducted on `Irace` v.3.5 [316].

Large Neighborhood Search

The LNS algorithm depends on several parameters, including the probabilities of selecting each destroy operator (σ) and each repair operator (ρ).

We perform three independent tuning procedures, i.e., one for each use case. Each tuning procedure is allocated a budget of 1,000 experiments, with a maximum runtime of 30 minutes per experiment. Table 10.1 summarizes the parameters.

Table 10.1: Parameter tuning for the Large Neighborhood Search (LNS) by Use Case (UC)

Name	Description	Range	Value		
			UC-1	UC-2	UC-3
σ_{RB}	Bias of RandomBatches destroyer	0.0–1.0	0.4	0.4	0.4
σ_{RBM}	Bias of RandomBatchesSameMachine destroyer	0.0–1.0	0.3	0.2	0.3
σ_{RBA}	Bias of RandomBatchesSameAttribute destroyer	0.0–1.0	0.3	0.4	0.3
σ_{OPL-5}	Bias of Opl-5 repairer	0.0–1.0	0.2	0.2	0.9

Simulated Annealing

Given that three use cases are considered, each with different weights of the objective components, and given that each neighborhood has a different impact on these components, three separate tuning procedures are conducted. Each tuning procedure is granted with a total of 30,000 experiments, with the SA having a stopping criterion fixed at $i_{\max} = 1{,}000{,}000$. Furthermore, the final temperate is fixed ($T_f = 0.01$) after a set of preliminary runs of SA. Details on the parameter ranges and their final values are presented in Table 10.2.

Table 10.2: Parameter tuning for the Simulated Annealing (SA) by Use Case (UC).

Name	Description	Range	Value		
			UC-1	UC-2	UC-3
T_0	Start Temperature	50–200	199	115	178
α	Cooling rate	0.98–0.99	0.99	0.98	0.99
ρ	Neighborhood accepted ratio	0.10–0.40	0.31	0.24	0.29
σ_{S}	Bias of Swap neighborhood	0.00–1.00	0.09	0.26	0.07
σ_{NUB}	Bias of NewUnaryBatch neighborhood	0.00–1.00	0.14	0.11	0.4
σ_{NB}	Bias of NewBatch neighborhood	0.00–1.00	0.41	0.47	0.07
σ_{MJ}	Bias of MoveJob neighborhood	0.00–1.00	0.36	0.16	0.46

Hill Climbing

Unlike SA, the HC algorithm depends solely on the parameters governing neighborhood selection, which still require tuning. Since HC is used exclusively for the component analysis, we performed tuning only for one Use Case (UC), namely UC-3. The tuning procedure was allocated a total budget of 30,000 experiments. Details regarding the parameter ranges and the final selected values are reported in Table 10.3.

Table 10.3: Parameter tuning for Hill Climbing (HC) for the third Use Case (UC-3).

Name	Description	Range	Value
σ_{S}	Bias of Swap neighborhood	0.00–1.00	0.29
σ_{NUB}	Bias of NewUnaryBatch neighborhood	0.00–1.00	0.38
σ_{NB}	Bias of NewBatch neighborhood	0.00–1.00	0.07
σ_{MJ}	Bias of MoveJob neighborhood	0.00–1.00	0.26

10.2.3 Instances

This analysis considers both the publicly available instances from M. Lackner et al. [275] and those introduced in Section 9.2.1. The benchmark set comprises 120 instances, varying by the number of jobs (10, 25, 50, 100, 250, or 500), the number of machines (2 or 5), and the number of attributes (2 or 5). For simplicity, the instances are categorized based on the number of jobs: 40 small instances (10 and 25 jobs), 40 medium instances (50 and 100 jobs), and 40 large instances (250 and 500 jobs). For tuning purposes, we use 50 additional instances with characteristics similar to those in the test set.

To analyze the scalability of SA (Section 10.3.2), a new set of 21 instances has been generated, i.e., 7 instances have 1,000 jobs, 7 have 2,500 jobs, and 7 have 5,000 jobs.

10.2.4 Technical Details

All experiments are executed on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8×64GB RDIMM. Each algorithm is executed on a single thread.

Exact Methods

To ensure a fair comparison, CP and ILP are run on the same machine used for the SA and LNS experiments. The experiments maintain the original timeout periods from previous works, granting CP and ILP an execution time of 1 hour. The ILP is solved with `Gurobi` v.11 [208], whereas the CP model use `CP Optimizer` v.22.1.0 [273].

Large Neighborhood Search

We implement the LNS structure and the destroy operators using `Python` v.3.10.12; the repair operator is run with `CP Optimizer` v.22.1.0. The LNS algorithm is equipped with a timeout of 1 hour. To ensure a fair comparison w.r.t. the exact methods, we also conduct experiments where exact methods are run for more time. These experiments are in line with the others obtained in this section and are reported in Appendix D.

Simulated Annealing and Hill Climbing

The SA and HC algorithms are implemented in `C++` using EasyLocal++ v.3 (Appendix B). The code is compiled using `Clang++15`.

The SA algorithm is equipped with a stopping criterion of $i_{\max} = 1,000,000$ iterations, an overall timeout of 1 hour, and a maximum of $10,000$ iterations with no improvements. We conducted additional experiments by allowing our SA algorithm to run for more iterations ($i_{\max}$ =5,000,000, with $50,000$ idle iterations) to determine if this would yield better performance. The experiments reported that allowing more iterations did not produce better results.

The HC algorithm is equipped a stopping criterion of $i_{\max} = 1,000,000$ iterations and a maximum of $10,000$ iterations with no improvements, and an overall timeout of 1 hour.

Methodological Considerations on Stopping Criteria

It is important to note that the time limits applied to the different algorithms are not uniform, reflecting the differences in their nature. For the exact methods (ILP and CP), the timeouts are inherited directly from the original publication [275].

In contrast, metaheuristic approaches such as SA and LNS follow fundamentally different paradigms. Exact methods aim to either prove optimality or exhaustively search constrained spaces, whereas metaheuristics are designed to explore the solution space heuristically, often trading guaranteed optimality for scalability and flexibility in handling larger or more complex instances. For SA, the stopping condition is not strictly time-based but rather driven by stagnation, i.e., the search terminates when no further improvements are observed over a sustained period and with an overall number of iterations to safeguard from incredibly long run and allow for machine indipendent settings of parameters. On the other hand, LNS was assigned a fixed time budget. SA never reached the timeout and always terminated the run before. To address potential concerns that LNS benefits disproportionately from this extended runtime, we conducted additional experiments with adjusted time limits to assess the sensitivity of its performance relative to the computational budget (Appendix D). Results were not different from what we observe in this chapter.

This distinction underscores the challenge of establishing perfectly comparable conditions across fundamentally different solution methodologies, different programming language implementation, etc. Nonetheless, our experimental design seeks to strike a balance between fairness and reflecting the typical use cases and practical behaviors of each algorithm class. Such complex topic is now of relevance within Working Group 6 of the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action.

10.3 Results

In this section, we first compare the solution methods (Section 10.3.1) and then we analyze the components of the one that perform the best (i.e., SA Section 10.3.2). For additional analysis on LNS, please refer to Appendix D.

10.3.1 Comparison of Solution Methods

We analyze the results by first considering the small instances and then the medium to large ones.

Results on Small Instances

CP and ILP typically yield good results for small problem instances. The exact methods are able to prove the optimality of solutions for almost all instances within this category (but only for two larger ones). The proposed metaheuristic algorithms match the solution quality of these methods, as confirmed by the Friedman test, which shows no significant difference in performance among the algorithms. Notably, they are able to find optimal solutions for instances where the optimal result is known (though, naturally, they cannot formally prove this).

Results on Medium and Large Instances

We now compare our SA with CP, ILP, and LNS considering medium instances and large instances (where CP and ILP tends to struggle).

The null hypothesis of the Friedman test is rejected for all use cases and instance types, as the p-value is considerably lower than the fixed significance level α (in all cases p-value $\leq 10^{-12}$). This indicates that the differences in algorithm performance are statistically significant. Therefore, we may proceed by comparing the algorithms using CD plots (Figure 10.4). In five out of the six combinations of use cases and instance sizes, SA outperforms all other methods, with no other method showing comparable performance from a statistical perspective. The only case where LNS outperforms SA is in UC-2 for the medium-sized instances. In 3 out of 6 cases, LNS delivers better results than the exact methods, while in the remaining cases, its performance is comparable.

We now proceed to analyze the relative gap of each method with respect to the best known solution across all approaches (ILP, CP, LNS, and SA). Figure 10.5 presents the results, broken down by use case (UC) and instance size. One notable observation is that the gap for ILP is often so large that it visually compresses the performance of the other methods in the figure. This is primarily due to the fact that ILP fails to find feasible

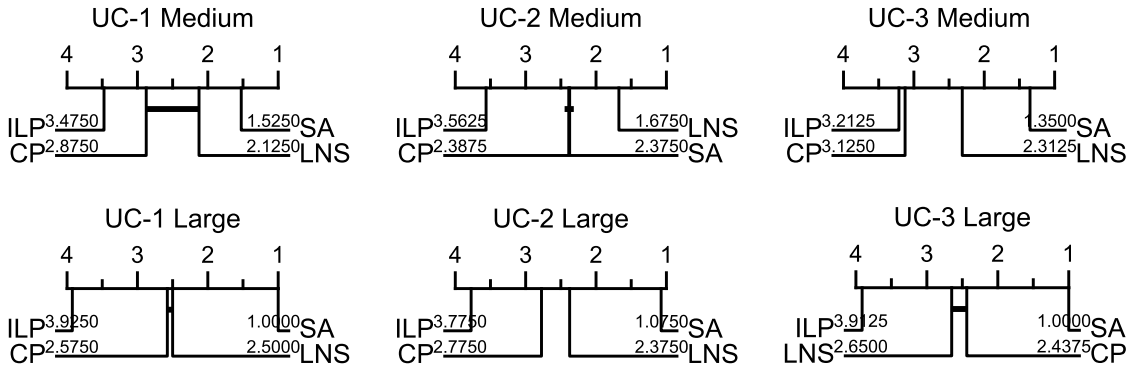


Figure 10.4: Critical Difference (CD) plots ranking Simulated Annealing (SA), Integer Linear Programming (ILP), Constraint Programming (CP), and Large Neighborhood Search (LNS). The analysis is conducted by Use Case (UC) and by instance size (medium and large).

solutions for many instances and this is set to an objective value of 1.0. To provide a clearer quantitative comparison, we also report the average relative gaps in Table 10.4.

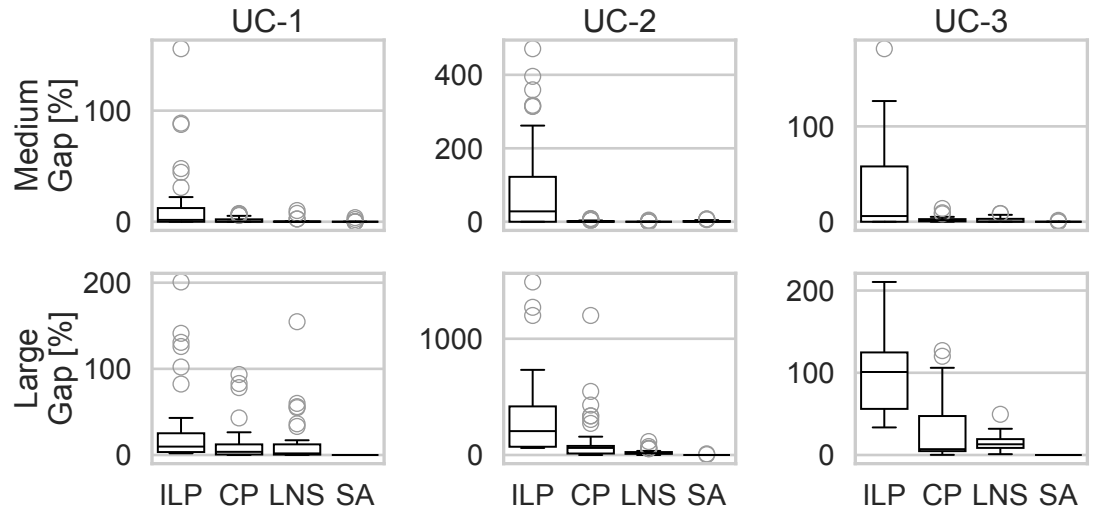


Figure 10.5: Comparison of Integer Linear Programming (ILP), Constraint Programming (CP), Large Neighborhood Search (LNS), and Simulated Annealing (SA) w.r.t. the best results by Use Case (UC) and instance size (medium and large).

The previous statistical results made it clear that SA outperforms the other algorithms in the majority of the cases. In order to quantify the improvement made by SA per instance, the relative gap is employed. For every instance, the best objective cost retrieved by SA is compared with the best solution found by any of the other methods (that is the baseline solution, i.e., a negative gap implies that SA improves w.r.t. the other methods). For UC-1, the average gap is -3.53% (± 8.07); For UC-2, the average gap is -6.23% (± 10.5); for UC-3, the average gap is -4.90% (± 5.76) (see Figure 10.6 for a comprehensive overview). Table 10.5 reports the number of instances per use case and instance size per gap class. For large instances, SA achieves better results than the best solution found from the state-of-the-art methods (for all the large instances in UC-1 and UC-3, and for the large

Table 10.4: Comparison of Integer Linear Programming (ILP), Constraint Programming (CP), Large Neighborhood Search (LNS), and Simulated Annealing (SA) w.r.t. the best results by Use Case (UC) and instance size (medium and large). The comparison is in terms of average gap w.r.t. the best known results.

UC	Size	Average gap [%]			
		ILP	CP	LNS	SA
UC-1	Medium	14.35 ± 31.40	1.45 ± 2.31	0.64 ± 2.00	0.15 ± 0.66
	Large	29.1 ± 46.61	12.85 ± 22.77	14.87 ± 9.29	0.00 ± 0.00
UC-2	Medium	93.53 ± 132.70	1.41 ± 2.37	0.23 ± 0.85	1.34 ± 2.2
	Large	316.14 ± 343.56	120.16 ± 217.65	23.27 ± 22.53	0.32 1.62
UC-3	Medium	35.05 ± 45.84	2.24 ± 2.84	1.72 ± 2.54	0.04 ± 0.23
	Large	94.66 ± 41.08	27.58 ± 36.11	14.87 ± 9.29	0.00 ± 0.00

majority in UC-2 gap $< 0\%$). Overall, considering both medium and large instances, an improvement of more than 1%, could be made in 48.33% of all cases. Nevertheless, in the case of UC-2, LNS is capable of delivering better results than those of SA.

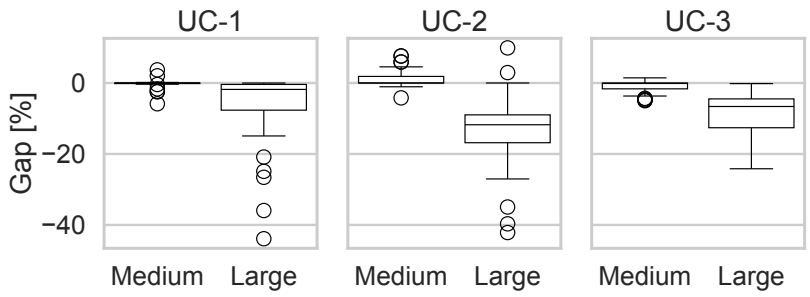


Figure 10.6: Comparison of Simulated Annealing (SA) w.r.t. the state-of-the-art results by Use Case (UC) and instance size (medium and large).

10.3.2 Simulated Annealing Components

This section analyzes the impact of various components on the final results, including the metaheuristic scheme, the initial solution, and the neighborhoods.

The analyses involve UC-3, since it is the use case with the most balanced set of weights. Furthermore, the analysis focuses on medium to large instances as these are the cases where SA clearly demonstrates its superiority compared to other algorithms.

Metaheuristic Scheme

This section compares SA with another closely related metaheuristic, HC. The motivation behind this comparison lies in understanding whether the metaheuristic scheme of SA significantly contributes to finding high-quality solutions for the OSP. HC is chosen for this analysis as it represents a simplified version of SA, where a move is accepted only if it yields an improvement or a solution of the same quality.

Table 10.5: Comparison of Simulated Annealing (SA) w.r.t. the state-of-the-art results by Use Case (UC) and instance size (medium and large) in terms of number of instances.

UC	Size	Total	Gap				
			$[-\infty;-1]$	$(-1;0)$	$=0$	$(0;1)$	$[1;+\infty]$
UC-1	Medium	40	4	14	16	4	2
	Large	40	22	18	0	0	0
UC-2	Medium	40	2	5	9	10	14
	Large	40	37	0	1	0	2
UC-3	Medium	40	12	9	16	2	1
	Large	40	39	1	0	0	0

The results of the Friedman test indicate that the algorithms perform differently for medium and large instances. Figure 10.7 reports the related CD plots, considering as a reference point for the state-of-the-art methods LNS. The diagram highlights that SA delivers superior performance w.r.t. HC, which is however better than LNS.

Figure 10.7: Critical Difference (CD) plots ranking the Simulated Annealing (SA) w.r.t. Hill Climbing (HC). The analysis also includes Large Neighborhood Search (LNS) as a reference point.

The relative gap is calculated using HC as the baseline, so as to measure the improvement achieved by SA (Table 10.6). SA consistently outperforms HC, with a growing relative gap as the problem size increases. This indicates that SA is more effective for tackling the increased complexity of larger problems. Overall, the results suggest that the larger the instances, the greater the advantage of using SA over HC.

Table 10.6: Comparison of Simulated Annealing (SA) w.r.t. Hill Climbing (HC) results by instance size in terms of number of instances.

Size	Total	Gap				
		$[-\infty;-1]$	$(-1;0)$	$=0$	$(0;1)$	$[1;+\infty]$
Medium	40	9	12	15	4	0
Large	40	8	28	1	2	1

Initial Solution

The proposed SA employs the construction heuristic by M. Lackner et al. [275] as the initial solution. While in Chapter 9 we demonstrated that such an algorithm represents together with the theoretical Lower Bounds (LBs) a good estimate to the optimal solution cost, the present analysis wants to understand its utility for the SA approach. To this extent, the

results of SA that uses the construction heuristic are compared with those of SA that uses the random initial solution (obtained as described in Section 10.1.2). The analysis also includes the initial solution values, obtained through the construction heuristic and the random procedure. The calculation time of the initial solutions is comparable (in both cases they are less than 1 second on average).

The results of the Friedman test indicate that the algorithms perform differently, as the null hypothesis is rejected for both instance sizes (p-value $\leq 10^{-12}$). Consequently, Figure 10.8 shows the CD plots, which visually highlight these performance differences. For both instance sizes, there is no statistical evidence to suggest that the four algorithms perform equally. As expected, both SA algorithms outperform the initial solutions, as they use the latter as starting points to explore the search space. SA methods initialized with random solutions consistently perform worse than those using solutions generated by a construction heuristic. This is partly because random solutions do not facilitate grouping jobs into batches, which impacts the component of the cost function related to processing time (p). On the other hand, while the SA algorithm starting from a construction heuristic performs better than one starting from a random solution, their rankings are relatively close, although they still show statistically significant differences.

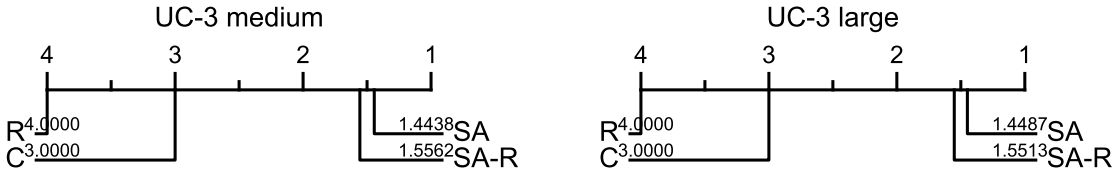


Figure 10.8: Critical Difference (CD) plots ranking the Simulated Annealing (SA) w.r.t. a Random Solution (R), a Constructive Solution (C), SA run using a random solution as starting point (SA-R).

The improvement of SA compared to its initial solution (obtained through the construction heuristic) is evaluated using the gap and the initial solution as a baseline. Figure 10.9 reports the results differentiating by instance size. In all cases, SA consistently improves the initial solution ranging from -35.35% to -2.83% (with an average of $-14.28\% \pm 6.88$).

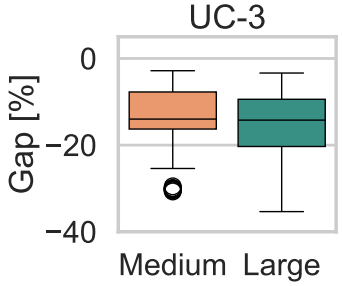


Figure 10.9: Comparison of Simulated Annealing (SA) w.r.t. the initial solution obtained through the construction heuristic. The comparison is conducted by differentiating by instance size.

Neighborhoods

Franzin and Stützle [183] pointed out that neighborhoods are often one of the keys to SA success. For this reason, this section examines how each neighborhood contributes to the final solution. An ablation analysis is conducted by studying how the SA performs when using all but one neighborhood [173]. To do so, a zero probability bias is associated to the removed neighborhood, while the remaining ones are scaled proportionally to keep the same mutual ratios among them. The SA version without a given neighborhood is indicated with NoNeighborhoodName.

The p-value is less than 10^{-12} for both instance sizes, indicating that the algorithms do not perform equally. Figure 10.10 presents the results in the form of CD plots. In all cases, the SA algorithm using all neighborhoods outperforms any version that excludes one neighborhood. Among the variants using all but one neighborhood, a clear ranking emerges: excluding the MoveJob neighborhood results in the poorest performance, highlighting the importance of aggregating jobs into existing batches for the success of SA. Conversely, the neighborhood responsible for creating new batches with multiple jobs has the least impact, although it still contributes to the overall effectiveness of SA explorations.

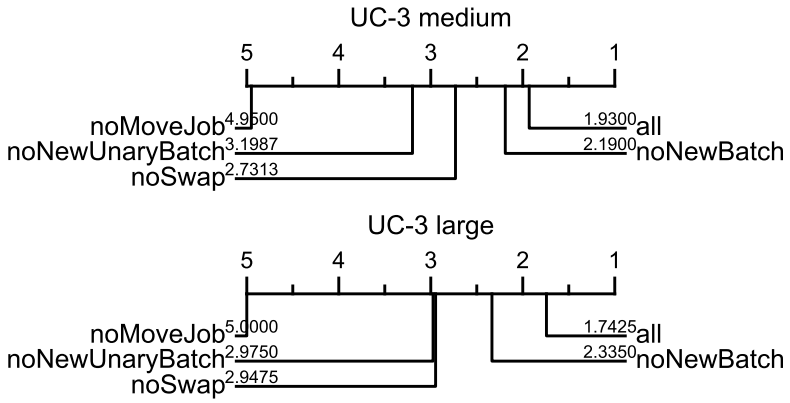


Figure 10.10: Critical Difference (CD) plots ranking the Simulated Annealing (SA) that uses all four neighborhoods (indicated with ALL) w.r.t. SA using all but one the neighborhoods. Note that in the case of large instances, the NoNewUnaryBatch and NoSwap are statistically equal, but the thick line indicating this is too little to be observed.

The results are analyzed using the gap metric, with the SA algorithm employing all neighborhoods as the baseline (Figure 10.11). In all cases, the average gap is greater than zero, confirming that the SA algorithm with all neighborhoods delivers superior results. Specifically, when considering the algorithm without the MoveJob neighborhood, the gap can be as high as nearly 45%, indicating that this move is crucial for good performance. However, for the other three neighborhoods, while with no statistically significant performance difference, in a few cases, employing just three out of four moves could potentially improve the results.

During preliminary experiments, other neighborhoods common for (parallel) batch scheduling problems were also tested (for instance, inverting the order of the batches or inserting a batch in a new position of the schedule). However, further analysis demonstrated that they were not useful for the OSP, and removing them provided better performance.

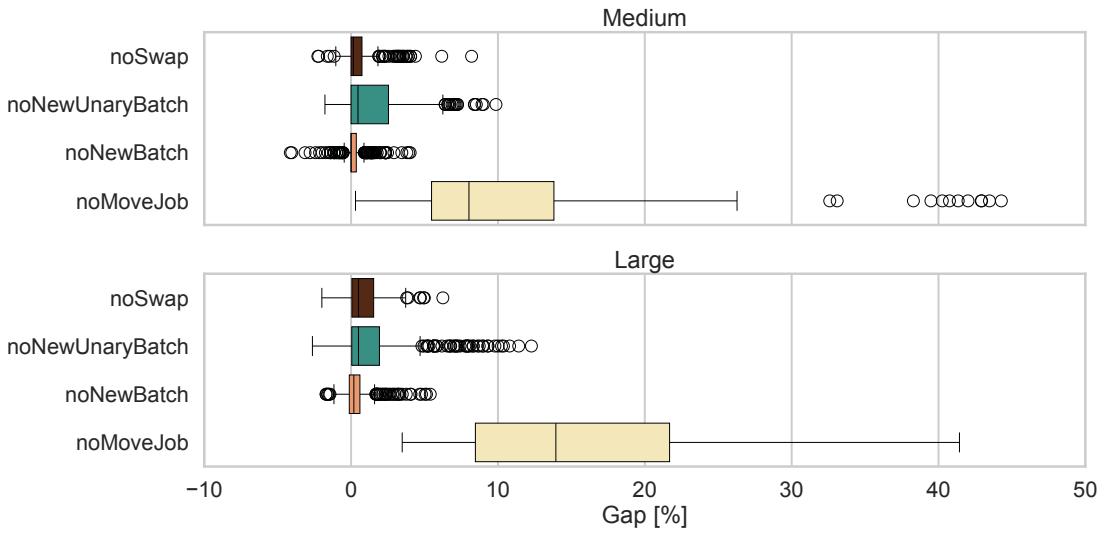


Figure 10.11: Comparison of Simulated Annealing (SA) that uses all four the neighborhoods w.r.t. SA using all but one the neighborhoods.

Scalability to Larger Instances

A set of larger instances is generated to test the scalability of SA. The analysis involves UC-3 and considers LNS as a baseline.

The statistical test confirms that the two approaches do not perform equally, with SA yielding consistently better results than LNS. The latter is often unable to improve the results over the initial solution. In all instances, the gap is consistently $< -1\%$. The average gap is -11.55% ($\pm$5.13) (Figure 10.12).

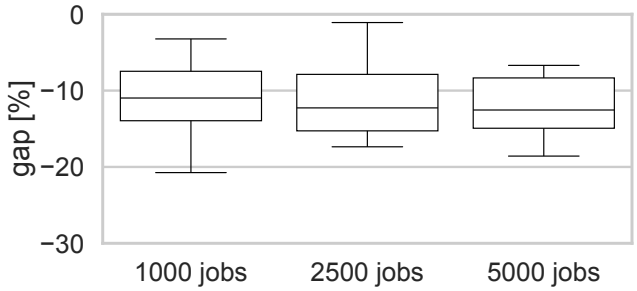


Figure 10.12: Evidence of scalability of Simulated Annealing (SA) toward larger instances.

10.4 Concluding Remarks

We now summarize the contributions of this chapter (Section 10.4.1), then discuss its limitations (Section 10.4.2), and finally outline possible directions for future research (Section 10.4.3).

10.4.1 Content

This chapter focused on the development and evaluation of metaheuristic approaches for the OSP, presenting an innovative combination of four neighborhoods within a SA framework and a LNS approach based on three destroy operators. The relevance of the OSP from an industrial perspective is clear, as it addresses complex scheduling needs that are critical for improving operational efficiency and enhancing the sustainability of semiconductor manufacturing processes. The proposed methods not only outperform state-of-the-art techniques on large problem instances but also match their performance on smaller ones, demonstrating robustness across different problem scales.

Among the approaches, SA delivered the best results. Its performance is driven by a novel combination of neighborhood structures that, to the best of our knowledge, has not previously been explored in the literature, even in the broader context of batch scheduling problems. This innovation enables more effective exploration of the search space, leading to higher-quality solutions. Our in-depth analysis further highlighted the critical role of both the initial solution construction and the four neighborhoods, each of which positively contributes to solution quality. In particular, the construction heuristics for generating the initial solution proved instrumental in consistently outperforming alternative variants.

10.4.2 Limitations

The main limitation of this study lies in the inherent difficulty of comparing methods that are fundamentally different in nature, such as exact approaches, local search techniques, and hybrid metaheuristics. This challenge complicates the establishment of fully fair benchmarks across methodological paradigms.

10.4.3 Future Work

Promising avenues for future research include the exploration of larger and more diverse sets of instances, as well as extensions to related problem formulations. Another direction concerns the integration of reinforcement learning techniques to dynamically adjust the probabilities of applying specific neighborhoods. This is motivated by the observation that, in a subset of instances, three out of four neighborhoods produced promising results, suggesting that adaptive selection strategies could further enhance performance.

Chapter 11

Instance Space Analysis and Algorithm Selection

In Chapter 10, traditional analyses of stochastic algorithms, such as gap analysis and Critical Difference (CD) plots, have identified two algorithms as the most effective for solving the Oven Scheduling Problem (OSP): Simulated Annealing (SA) and Large Neighborhood Search (LNS). Both have shown to deliver high-quality solutions across a range of instances, yet their performance can vary depending on the specific instance characteristics. Given the significant impact that scheduling quality can have on overall industry efficiency, selecting the most suitable algorithm for a given problem instance becomes essential. One promising methodology for achieving this is the Instance Space Analysis (ISA) [436], which allows for a comprehensive evaluation of algorithms across a diverse range of problem instances. ISA connects algorithms' performance to instance characteristics, identifying areas where each algorithm excels or struggles.

In this chapter, we analyze the instance space of the OSP (Use Case (UC)-2) and characterize the performance patterns of SA and LNS within the ISA framework. We also investigate the viability of automated algorithm selection. The main contributions are:

- **New instances**: A dataset of 1396 instances spanning a broader range of variations than those available in the literature for related problems.
- **Feature design**: A set of 165 features that characterise parallel batch scheduling instances; of these, 7 are selected by ISA as most informative.
- **Instance space characterisation**: Identification of regions in the instance space where one algorithm outperforms the other and an analysis of algorithm performance with respect to instance features, highlighting strengths and weaknesses.
- **Portfolio via algorithm selection**: An automated algorithm selection portfolio using four Machine Learning (ML) models, each yielding performance that surpasses either SA or LNS alone.

This chapter is organised as follows. Section 11.1 presents the ISA instantiation for the OSP; Section 11.2 details the experimental setup; Section 11.3 reports the results; and Section 11.4 offers concluding remarks.

11.1 Methodology

In this section, we detail how ISA is instantiated for the OSP. Specifically, Section 11.1.1 defines the problem subset; Section 11.1.2 describes the algorithms considered; Section 11.1.3 specifies the performance space; Section 11.1.4 enumerates the instance features; and Section 11.1.5 explains the construction of the instance space. Section 11.1.6 summarises the complementary algorithm selection experiments used to validate the ISA findings. We remind the reader that general information on ISA are available in Section 2.6.5.

11.1.1 Problem Subset

Our ISA includes a total of 2190 instances: 794 coming from the literature and real-world cases, and 1396 newly generated specifically for this work. Note that the analysis presented in this chapter are concerned with UC-2.

Existing Instances

We include in our ISA the 80 original instances for the OSP from the work by M. Lackner et al. [275] (`OSP-1`, see also Section 3.4). We include 40 larger instances (`OSP-2`) and 25 tuning instances (indicated as `TU`, see Section 9.2 and Section 10.2). We also consider 49 real-world instances from an industrial partner (`RL`) and 600 instances related to a similar problem [125] (`DA`, see also Appendix D).

Newly Generated Instances

We generate 1396 additional instances (indicated as `NEW`) using the random instance generator described by M. Lackner et al. [275]. These new instances serve two objectives: *(i)* Better cover the instance space. *(ii)* Train the models used for algorithm selection. To ensure the diversity of this new instance set, the parameter values chosen for the generation cover a broader range than those of the existing instances. These parameter values include the number of jobs, machines, and attributes, as well as maximum processing times, machine capacity limits, etc. The generation process is repeated multiple times until no (large) areas of the instance space remain empty.

11.1.2 Algorithm Space

We analyze the two state-of-the-art algorithms for the OSP, which are the LNS algorithm and the SA algorithm reported in Chapter 10.

11.1.3 Performance Space

As the primary performance metric for our ISA, we employ the value of the objective function obj (Equation 8.1), where lower values indicate better performance. In addition, following the approach suggested by Smith-Miles and Muñoz [436], we apply two supplementary methods to analyze the results. First, we consider an algorithm to be *good* for a given instance if its performance is within $(1 + \epsilon)$ times the best performance on that instance. Formally, this is expressed by

$$\widehat{\text{obj}}_{\psi} \leq 1 + \epsilon$$

where

$$\widehat{\text{obj}}_{\psi}(x) = \frac{\text{obj}_{\psi}(x)}{\min_{\psi' \in \mathcal{A}} obj_{\psi'}(x)}$$

is the scaled cost of algorithm ψ on instance x, relative to the best known cost for x. We set $\epsilon = 0.01$ (i.e., a 1% gap). Moreover, an algorithm is considered a *winner* for an instance if it is among the best-performing algorithms, meaning it achieves the best performance with $\epsilon = 0$. If an algorithm is the sole winner for an instance, it is classified as a *unique winner*.

11.1.4 Feature Space

We propose 165 features to characterise the instances: 115 direct descriptors, 9 features derived from theoretically computed Lower Bounds (LBs), and 41 graph-based descriptors. To the best of our knowledge we are the first to systematically incorporate theoretically computed LBs into ISA and to employ a compatibility-graph construction tailored to the OSP. Regarding the latter, graph representations have appeared in ISA before [116, 446].

We now provide an abridged list of features; the complete list is reported in Appendix E.

Direct Features

Direct features can be extracted directly from the instances through simple counts or examination of instance entities, possibly involving only minor calculations. For example, we include the number of jobs, the number of machines, the number of operations, calculated as the product between the number of jobs and the number of machines, the number of jobs per machine considering the concept of eligible machines, the job sizes, and the normalization weights for each cost component.

Lower Bounds as Features

The feature set includes theoretical LBs for the OSP (Chapter 9) and related metrics: the bound on the number of batches, LBs on cost components, the upper bound on processing time reduction from batching, and the time to compute these bounds.

Graph Features

Graph features are based on the undirected job compatibility graph. In this graph, each node corresponds to a job, and arcs connect pairs of nodes (jobs) that can be processed together under the following conditions: *(i)* They share the same attribute. *(ii)* They can be processed on at least one common machine. *(iii)* Their processing times are compatible. *(iv)* Their release and due dates are compatible. Given that job tardiness is already considered in the objective function, we also introduce a relaxed version of the graph where condition *(iv)* is omitted. For each graph (complete and relaxed), we extract features such as the number of edges, density, and isolated nodes.

Key Statistical Figures

Some features are represented by a single value, while others require aggregation to capture their statistical distribution. In such cases, we compute the minimum ($\min$), maximum

($\max$), range ($range = \max - \min$), mean, quartiles ($q1, q2, q3$), and standard deviation (std). For instance, the total number of jobs is a single value, whereas the jobs' sizes are aggregated into average, standard deviation, etc.

11.1.5 Instance Space Construction

We know detail how the instance space is constructed in terms of how the features are processed, how the instance space boundaries are constructed, and how strengths and weaknesses areas of the algorithms are detected. Finally, we also provide tecnical details useful for reproducibility of the results.

Feature Processing

First, the feature values are preprocessed by *(i)* Bounding outliers to the median plus/minus five times the inter-quartile range. *(ii)* Normalizing the values so they are normally distributed by applying a Box-Cox transformation and a z-score standardization.

For each algorithm, the most relevant features are identified by calculating their Pearson correlation with the algorithm's performance, retaining only those with a correlation above a minimum threshold of 0.45. The full set of features is then grouped using k-means clustering (with $k = 7$ clusters, a maximum of 1000 iterations, and 100 repeats), based on similarity in Pearson correlation values. This grouping allows us to select a representative feature from each cluster, simplifying and enhancing the effectiveness of subsequent visualizations. The representative feature is selected by initially projecting the features onto a 2D space using a Principle Component Analysis (PCA). Next, a Random Forest (RF) model (with 100 trees) is applied to classify each instance as either *easy* or *hard* for each algorithm. Finally, a Genetic Algorithm (GA) is employed to guide the feature selection process, identifying the combination of features that produces the most accurate RF predictions (see [436, Section 3.2.2]). The selected features are projected into a 2D space according to:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} c_1^1 & c_2^1 \\ \dots & \\ c_1^k & c_2^k \end{pmatrix}^{\mathbf{T}} \cdot \begin{pmatrix} \text{feature}_1 \\ \dots \\ \text{feature}_k \end{pmatrix} \tag{11.1}$$

The coefficients $c_1^1, \dots, c_2^k$ are determined numerically by solving the problem defined in [436, Section 3.2.3]. The 2D projection serves as the primary tool for exploring the instance space.

Instance Space Boundaries

The coverage of the instance space is assessed by identifying two boundaries: an experimental and a theoretical one. The experimental boundary is defined by the convex hull of the point cloud, representing the area with empirical evidence of existing instances. The theoretical boundary, in contrast, is derived from the hypercube formed using the minimum and maximum values of each selected feature. This hypercube is refined by eliminating improbable feature combinations: for any two features feature_1 and feature_2 with a correlation grater than 0.75, vertices containing $\min(\text{feature}_1)$ and $\max(\text{feature}_2)$, or $\min(\text{feature}_2)$ and $\max(\text{feature}_1)$ are removed. The same pruning method is applied to

features with strong negative correlations. The remaining vertices are projected into the 2D space using the previously determined coefficients, and their convex hull defines the theoretical boundary. Note that this procedure yields an approximate boundary, which may be a rough estimate of the true extent of the space.

Strengths and Weaknesses of Algorithms

The instance space is divided into regions of strengths and weaknesses for each algorithm, enabling automated recommendations. The ISA procedure trains a separate Support Vector Machine (SVM) for each algorithm using normalized coordinates as inputs.

We use SVM with a polynomial kernel and automatic tuning via 10-fold stratified cross-validation. This approach predicts the best algorithm for new instances, prioritizing the model with the highest precision, and if needed, the one with the highest average performance. Instances without a clear winner are marked as having no strong preference.

While SVM provides useful insights, the analysis should be complemented by considering the concept of an *algorithm footprint*, which represents the generalized area of the instance space where the algorithm performs well or best. ISA constructs the algorithm footprint by employing Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to identify high-density clusters of instances where each algorithm performs well. For each identified cluster, it generalizes the convex hull concept to form a polygon that tightly encloses all points within the cluster. We set a purity threshold of 0.75, meaning that at least 75% of all instances where the algorithm shows good or best performance must be enclosed within the footprint. These footprints offer a more nuanced assessment of algorithm performance, providing metrics such as the footprint area, density (calculated as the number of instances within the footprint per unit area), and purity.

11.1.6 Algorithm Selection

Algorithm selection aims to predict the best algorithm (the winner) for a given instance, treating it as a classification problem where each class represents one algorithm. This contrasts with the ISA approach, which builds separate models for each algorithm and combines predictions, resolving ties using precision and performance probability. We apply our approach to validate the ISA outcomes and explore algorithm portfolio's capabilities.

Dataset

We focus on instances with a unique winner ($\epsilon = 0.0$), excluding ties as misclassifying a tie still yields an optimal result. For experiments validating ISA results, we use only the features identified by ISA. In contrast, for algorithm selection, we use the full feature set, following related research [116, 446].

Classification Algorithms

The following standard classification algorithms were used for algorithm selection (for the parameter tuning ranges, please refer to Section 11.2):

- Most Frequent (MF), always predicting the most frequent class as a baseline (i.e., SA).

- RF, with 170 trees for tests related to ISA validation and 200 trees for the test related to the algorithm selection experiments, reflecting the use of two datasets.
- Decision Tree (DT), using the Gini criterion, a maximum depth of 8, minimum samples to split set to 14, and minimum samples at a leaf set to 10.
- K-Nearest Neighbors (KNN), with 5 neighbors.
- SVM, with a linear kernel and regularization parameter equal to 1.

Performance Measures

We evaluate the models using:

- **Accuracy**, the proportion of correct classifications;
- **Recall**, the proportion of actual positives correctly identified;
- **Precision**, the proportion of true positives among all positive predictions.

We consider the SA class as the positive class. These metrics are also applied to assess ISA SVM models.

SHapley Additive exPlanations

To validate the results of the ISA, we train a RF model using the 7 features selected by the ISA procedure and analyze their influence on the model's predictions using SHapley Additive exPlanations (SHAP) [320]. SHAP is a game-theoretic approach for interpreting ML model outputs, where SHAP values quantify each feature's impact by assessing its contribution to the overall prediction [167].

11.2 Experimental Setup

The experimental setup is as follows:

Solution Methods The algorithms' executions are scheduled on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8×64GB RDIMM. We run each algorithm 5 times with a different seed (i.e., from 42 to 46). The algorithm's implementation and tuning are the same as the one reported in the related sections. Both methods are stopped with a 1-hour time limit or after 10,000 idle iterations.

Feature Extraction Direct features are extracted employing the `numpy` (v.1.26.4) `Python` package [213]. Lower bounds are extracted with the same setting reported in Chapter 9. Graph-based features are extracted using the `NetworkX` (v.3.1) `Python` package [209]

Instance Space Construction The instance space is constructed using the ISA implementation available from the public repository[1] on a MacBook Air M3 with 16GB of memory and macOS Sonoma 14.3. All parameters not explicitly mentioned within

[1] `https://github.com/andremun/InstanceSpace`, mirrored on Zenodo [359].

the related description (Section 11.1) are set to default values. To enhance result visualization, the ISA images are reproduced using the `Python` package `seaborn` (v. 0.13.2) [492].

Classification Algorithms All models are implemented using `scikit-learn` (v.1.5.1) [384] and evaluated via 10-times iterated stratified 10-fold cross-validation, using accuracy as the primary metric. Experiments are run on the same Mac Book described before. Parameter tuning was performed for the specified parameters, with default settings otherwise. More specifically, tuning of the classification algorithm is conducted using a randomized search, with a maximum of 100 iterations. When the number of combinations of configurations is lower than the maximum number of iterations, the procedure corresponds to an exhaustive search. The following parameter ranges have been tuned for the models:

- DT: The criterion is chosen between gini, entropy, or log loss; the maximum depth of the tree is chosen in the range from 1 to 20; the minimum number of samples required to split an internal node is chosen in the range from 2 to 20; the minimum number of samples required to be at a leaf node is chosen in the range from 1 to 20.
- RF: The number of trees is chosen in the range from 100 to 500 (step width of 10).
- KNN: The number of neighbors is chosen in the range from 3 to 15.
- SVM: The kernel is chosen between linear, rbf, or poly; the gamma parameter (when needed) is chosen from the values 0.1, 1, 10, or 100; the regularization parameter is chosen from the values 0.1, 1, 10, 100, or 1000.

SHapley Additive exPlanations SHAP values are obtained using the `shap` (v.0.46.0) `Python` package [319].

11.3 Results

In this section, we first outline the results linked to the ISA (Section 11.3.1) and the those of the traditional algorithm selection (Section 11.3.2).

11.3.1 Instance Space Analysis

This section reports the results of the ISA, starting with the identification of the most important features. Second, we report the distribution of the instances in the instance space, distinguishing by origin and highlighting the instance space boundary. We also analyze the algorithm footprints and the distribution of the features.

Projection Matrix

The projection matrix computed by the ISA procedure and used to map the processed features into the 2D space is as follows:

$$\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} 0.1342 & 0.4788 \\ -0.2584 & 0.6512 \\ -0.44 & 0.6504 \\ -0.48 & 0.3313 \\ -0.3202 & 0.5603 \\ -0.4475 & 0.3603 \\ 0.9907 & 0.6723 \end{pmatrix}^{\mathbf{T}} \cdot \begin{pmatrix} \text{number of jobs} \\ \text{number of jobs per machine (q3)} \\ \text{number of operations} \\ \text{normalization weight processing time} \\ \text{time to compute lower bounds} \\ \text{lower bound number batches} \\ \text{upper bound reduction processing time} \end{pmatrix} \tag{11.2}$$

Four out of seven features selected are connected to the size of the instance: the *number of jobs*, the third quartile of the number of jobs per machine (*number of jobs per machine (q3)*), the *number of operations*, and the lexicographic weight of the processing time (w_p, indicated by *normalization weight processing time*). The remaining three features pertain to LBs: *time to compute lower bounds*, *lower bound number batches*, and *upper bound reduction processing time*. Notably, no features related to compatibility graphs were selected.

Distribution of Instances

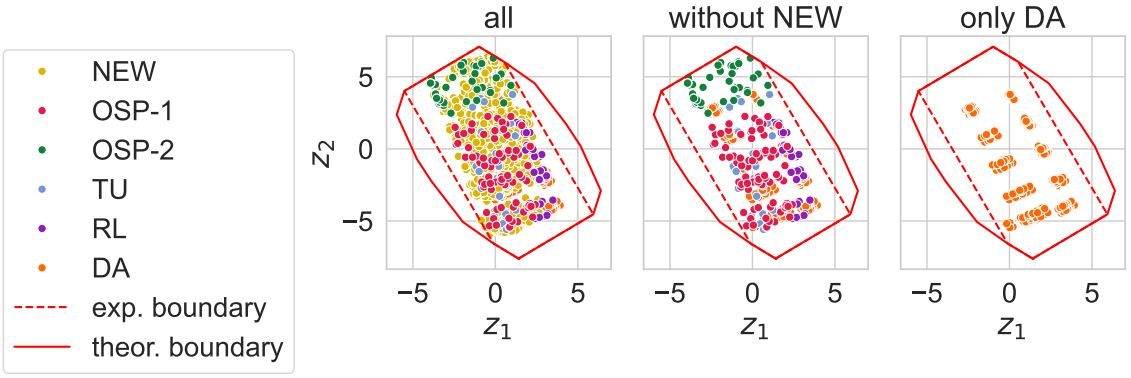


Figure 11.1: Instance space organized by the sources of the instances.

Figure 11.1 shows the projection of the instances within the instance space, organized by source. The left plot displays all instances (2190), the middle plot shows only those previously used in the literature (794), and the right one only `DA` instances (600). Theoretical boundaries ("theor. boundary") are marked with a continuous red line, while experimental boundaries ("exp. boundary") are indicated by a dashed red line. The middle plot reveals that the existing instances only cover certain regions of the instance space. In particular, the upper-left part of the instance space is only sparsely covered. The `NEW` instances, generated for this study, cover the entire instance space, showing the ability of the generator [275] to produce instances with diverse characteristics.

Among the existing instances, the `OSP-1` set is distributed across four lines between $-2.4 < z_1 < 2.4$ and $z_2 < 2.3$; these lines roughly correspond to the number of jobs (10, 25, 50, and 100) used in instance creation. In contrast, the `OSP-2` instances occupy the upper portion of the space ($z_2 \geq 2.5$). The `TU` set spans regions that overlap with both `OSP-1` and `OSP-2`, which can be expected since `TU` instances were specifically generated to

exhibit similar characteristics. The RL set shares the same z_2 range as OSP-1 but is more concentrated toward the right side of the instance space ($z_1 > 1.3$). Lastly, the DA instances form distinct clusters due to their generation process, where processing times, job sizes, and ready times were sampled from discrete uniform distributions [125].

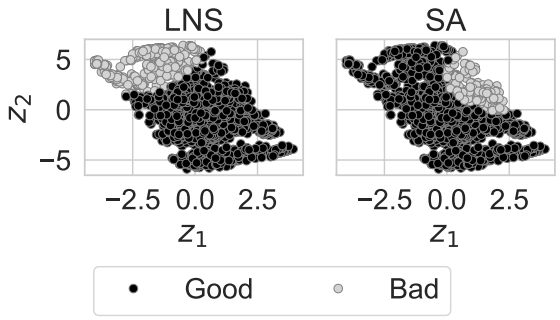


Figure 11.2: Algorithm footprint of predicted Support Vector Machines (SVMs) performance of Large Neighborhood Search (LNS) and Simulated Annealing (SA).

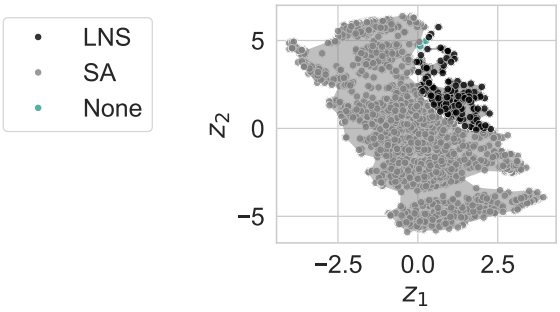


Figure 11.3: Recommended algorithms from Support Vector Machine (SVM) prediction models.

Algorithm Footprints

For each algorithm, a SVM model is trained to predict regions where the algorithm is expected to perform well ($\epsilon = 0.01$ w.r.t. the winner). Figure 11.2 illustrates predictions for LNS (left) and SA (right). Instances where the algorithm performs well are cyan, while poor performance is shown in magenta. Both algorithms perform well in the lower and mid-left regions. SA has strengths in the top and top-left region, which is weak for LNS, while LNS is strong in the mid-right, where SA is outperformed. The clear distinction, especially in the topmost areas, suggests the benefit of an algorithm portfolio approach.

Figure 11.3 integrates the SVM models into a unified algorithm selection, with LNS in cyan and SA in magenta. Ties are resolved by choosing the SVM model with higher precision, assigning the lower region to SA. In a few instances (3, labeled 'None' in black), neither algorithm is recommended due to limited statistical support, possibly requiring longer computation times or a hybrid approach.

Table 11.1 presents the performance metrics of the SVM models. The high precision values confirm that when an SVM model predicts good performance for an algorithm on a given instance, the prediction is reliable. Additionally, the high recall levels indicate

Table 11.1: Support Vector Machine (SVM) performance metrics.

	Pr	Accuracy	Precision	Recall
LNS	0.709	0.899	0.879	0.994
SA	0.815	0.869	0.894	0.952
Portfolio	0.898	—	—	—

that the model is unlikely to overlook instances where the algorithm performs well. The probability of correctly selecting a good algorithm (i.e., $\Pr = \Pr(\widehat{\text{obj}}_{\psi}(x) \leq 1 + \epsilon)$, $\epsilon = 0.01$) for a given instance is 0.709 when always using LNS and 0.815 when always using SA. The combination of the models in the algorithm portfolio increases this probability to 0.898.

Table 11.2 presents the footprint of the algorithms based on their actual performance, rather than the simulated data from the SVM models. It lists the normalized values for the areas, densities, and purities of the footprints, categorized for both good (subscript G) and best (subscript B) performance. In the best performance category, ties are resolved randomly, as no definitive criterion (such as precision, which is used for breaking ties in the combined SVM prediction) is available to determine the superior choice.

Focusing on the best performance, both algorithms exhibit densities greater than 1, indicating that their regions of strong performance are denser than average. While there is considerable overlap in the areas where both algorithms perform well, each also has unique regions where it outperforms the other. Notably, the extent of these exclusive areas is nearly equal for both algorithms.

Table 11.2: Footprint metrics on the actual outcomes (not simulated from Support Vector Machine, SVM) regarding Large Neighborhood Search (LNS) and Simulated Annealing (SA).

	area_G	density_G	purity_G	area_B	density_B	purity_B
LNS	0.87	0.94	0.88	0.17	1.05	0.81
SA	0.74	1.03	0.93	0.18	1.10	0.98

Distribution of the Features

Given that each algorithm has distinct strengths and weaknesses, our goal is now to explore how instance properties influence algorithm performance. Figure 11.4 shows the distribution of selected features across the instance space, with colors representing normalized values: cyan shades indicate smaller values, and magenta shades indicate larger ones. Most features shift from high to low values top-down, except for the *normalization weight of processing time* and the *lower bound to the number of batches* (top-left to bottom-right) and the *upper bound to the reduction of processing time* (right to left). Instances easily solved by both algorithms generally have fewer jobs and operations, while SA performs better on larger instances, making instance size a key factor. Conversely, LNS is superior when there is more potential for processing time reduction and a smaller lower bound on the number of batches.

To validate these results, we trained a RF model with the 7 selected features and analyzed their influence using SHAP values. The model achieved an average accuracy of 0.831 ± 0.003, recall of 0.866 ± 0.004, and precision of 0.859 ± 0.005. Figure 11.5 shows the SHAP values

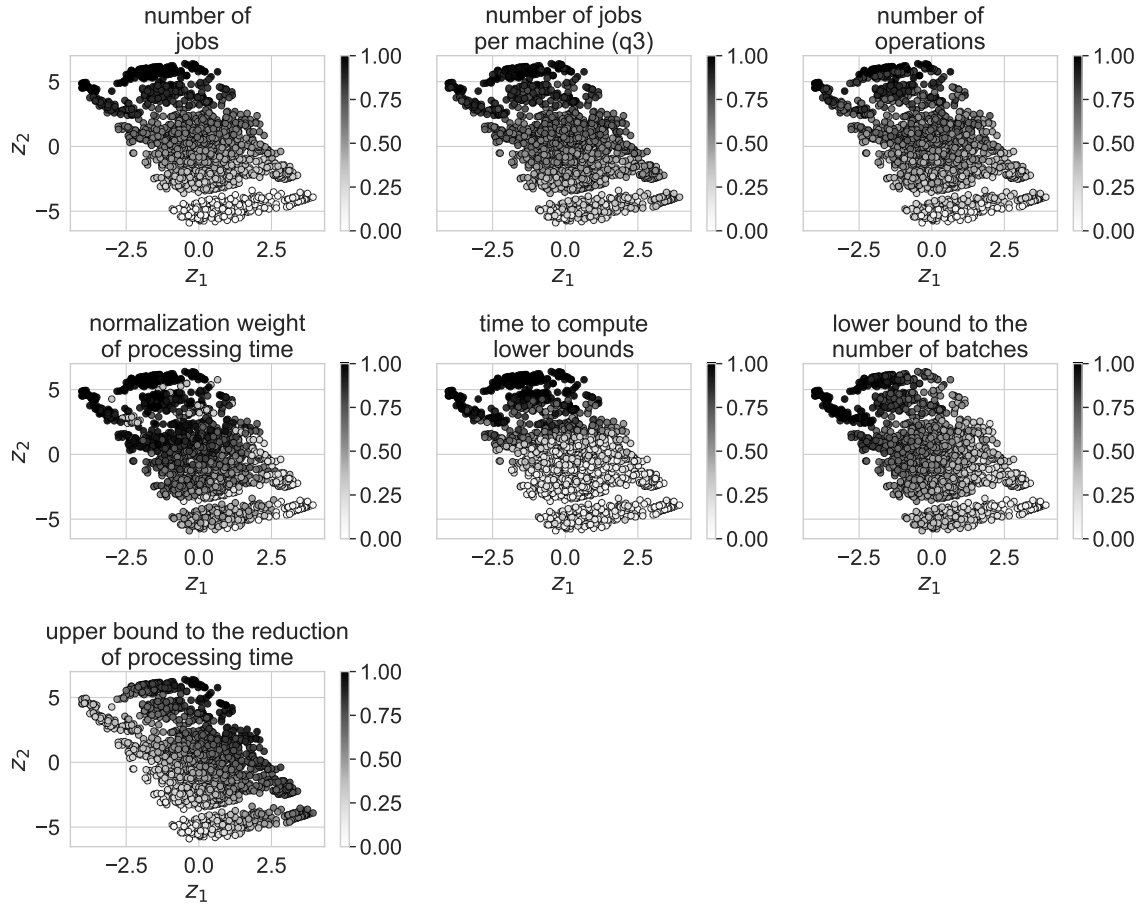


Figure 11.4: Normalized feature distribution across instance space.

for SA predictions (with symmetric results for LNS). Feature values are color-coded, pink for higher values and blue for lower. The x-axis represents the SHAP values. These results align with ISA findings. For example, lower *upper bound reduction of processing time* values favor SA, while higher values favor LNS. Likewise, a higher number of jobs is linked to SA, and fewer to LNS. This relatively clear distinction between feature values associated with SA and LNS is promising for feature-based algorithm selection. Indeed, the well-defined regions in the instance space suggest that predictive models can reliably guide algorithm selection decisions.

11.3.2 Algorithm Selection

Figure 11.6 shows the results of the algorithm selection, displaying accuracy, recall, and precision scores. Referring to accuracy and precision, all models outperform the MF model, with RF and SVM emerging as the top two options. The recall value for the MF model is 1, by definition, and is included here just for completeness. More specifically, RF achieves the highest accuracy at 0.855 ± 0.003, closely followed by SVM at 0.851 ± 0.003. For recall, RF led with 0.899 ± 0.004, while SVM followed at 0.886 ± 0.004. Finally, in terms of precision, SVM performed best at 0.871 ± 0.003, with RF slightly behind at 0.868 ± 0.003. Examining the algorithm selection process through the value of the objective function, MF shows an average gap of 1.5%. In contrast, all ML models achieve an average gap of less than 1%: RF

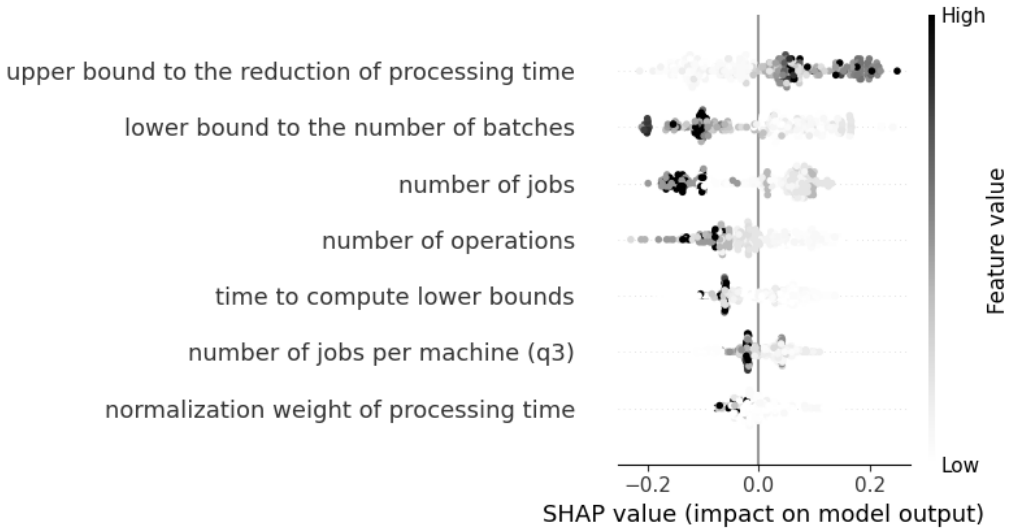


Figure 11.5: SHapley Additive exPlanations (SHAP) values for Random Forest predicting the best algorithm (positive class: Simulated Annealing, SA).

at 0.3%, DT at 0.4%, SVM at 0.4%, and KNN at 0.6%.

These results indicate that using a combination of algorithms, selected through a suitable classification model (i.e., an algorithm portfolio), is more effective than consistently using MF (i.e., the most frequent winner, that is SA). The differences in accuracy and precision are minimal, with SVM and RF emerging as equally strong options, both significantly improving performance over the baseline.

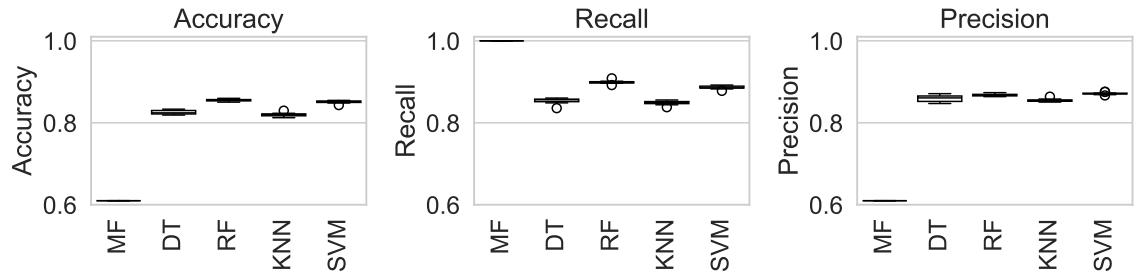


Figure 11.6: Performance of the algorithm portfolio, distinguishing by performance metric and classification algorithm. We consider the Simulated Annealing (SA) as the positive class.

11.4 Concluding Remarks

In this concluding section, we summarize the content of the chapter (Section 11.4.1), discuss limitations (Section 11.4.2) and outline future research directions (Section 11.4.3).

11.4.1 Content

We presented a comprehensive study of the OSP, evaluating the performance of two state-of-the-art algorithms, LNS and SA, using the ISA framework. The analysis spanned across 2190 instances described by 165 features. The results highlighted the strengths and

weaknesses of each algorithm and emphasized the importance of identifying a collection of relevant features that define the instance space effectively. The introduction of novel instances (1396 instances) allowed us to achieve better coverage of the instance space, facilitating a more accurate understanding of the algorithms' behavior. By discovering the most relevant features, we could map regions in the projected space where algorithms performed differently. This distinction justifies an algorithm portfolio approach that, for any given instance, selects either LNS or SA, based on the instance's feature values. The results obtained from four well-known ML models confirmed the clear advantage of such a portfolio approach since it significantly enhances the performance compared to the use of any single algorithm.

11.4.2 Limitations

This study has the following limitations:

- **Scope:** We target a single-stage, deterministic OSP with non-preemptive parallel batching, capacity limits, machine eligibility, availability calendars, and sequence/family-dependent setups. Results may not transfer to multi-stage, stochastic, or re-entrant settings, or to alternative batch-time definitions.
- **Feature design:** The feature set is hand-engineered. LB-derived features inherit assumptions from the underlying relaxations and may be less informative under some circumstances.
- **ISA methodology:** Although ISA is well established, conclusions are sensitive to projection/clustering choices and their hyper parameters, which were selected via manual tuning.
- **Portfolio learning:** Algorithm-selection models were trained/validated on the available benchmarks; out-of-distribution generalization to new industrial datasets is untested. Labels reflect empirical winners under a fixed budget and may change with different cost models or stability criteria; we did not assess calibration or decision-theoretic utility (e.g., risk-sensitive selection).

11.4.3 Future Work

Future research will explore the instance space for other variants of the OSP, such as those involving different objective weight settings (like UC 1 and 3). Additionally, the feature set could be expanded with more informative graph-based features, which have the potential to enrich the landscape of the instance space further. These enhancements could provide deeper insights into algorithm performance and reveal new opportunities for improving algorithm selection strategies.

Chapter 12

Search Trajectory Networks

Comparing different solution approaches for an optimization problem is essential to find the most suitable algorithm for a given class of instances. To this end, it is common for optimization practitioners to collect quantitative data on the algorithm's behavior (e.g., best objective function value, number of iterations, time, etc.) and to display them in tables, box plots, and line plots. While these analyses are useful, they fail to represent the complete behavior of the algorithms and focus only on the final step of the optimization process.

Recently, Search Trajectory Networks (STNs) have emerged as a powerful tool for enhancing the interpretability and explainability of algorithmic outcomes [370]. STNs offer a graphical representation of algorithmic behavior by representing the trajectories taken by algorithms in the search space as a directed graph. These networks are particularly useful for visualizing and analyzing the paths that search algorithms traverse when applied to the same problem instance.

Considering the Oven Scheduling Problem (OSP), classical analyses of stochastic algorithms, such as gap analysis and Critical Difference (CD) plots (Chapter 10), coupled with Instance Space Analysis (ISA) (Chapter 11), demonstrated that Large Neighborhood Search (LNS) and Simulated Annealing (SA) are the most successful algorithms. However, these findings are primarily based on evaluating the final outcomes, with a limited exploration of how these algorithms behave within the search space.

This chapter aims to analyze the behavior of two state-of-the-art algorithms, SA and LNS, in solving a real-world parallel batch scheduling problem using STNs. Through this analysis, we contribute to the study of algorithms in parallel batch scheduling by providing deeper insights into the solution landscape of problem instances and its relationship with algorithm performance. Additionally, we highlight the value of STNs in algorithm analysis, with the goal of encouraging their adoption as a standard practice.

This chapter is structured as follows. Section 12.1 introduces STN methodology and Section 12.2 details its instantiation for the OSP. Section 12.3 reports the results and Section 12.4 offers concluding remarks.

12.1 Methodology

We now outline the methodology followed in this work (Figure 12.1). We apply two state-of-the-art algorithms for the OSP (Section 12.1.1) on a subset of problem instances

(Section 12.1.2). Representative solutions are collected (Sections 12.1.3 and 12.1.4) and STNs are constructed in form of graphs and described through specific metrics (Section 12.1.5). Furthermore, locations are identified through search space partitioning (Section 12.1.6). We remind the reader that additional details on STN are available in Section 2.6.6.

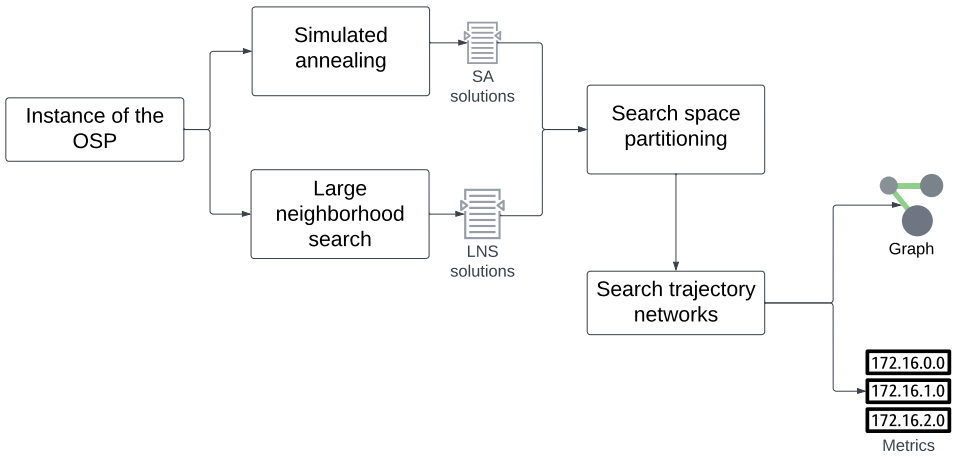


Figure 12.1: Experimental pipeline used in this chapter.

12.1.1 Algorithms

We consider the two state-of-the-art algorithms for the OSP (SA and LNS, see Chapter 10).

12.1.2 Instances

We consider the OSP benchmark dataset originally proposed by M. Lackner et al. [275] (Section 3.4), together with the larger instances introduced in Chapter 9, for a total of 120 instances. From these, we randomly sample 10 instances. Their characteristics are reported in Table 12.1, along with the algorithm achieving the best objective value (labelled "Winner"). As a crude proxy for combinatorial difficulty, we approximate the search-space size by $\text{jobs}^{\text{jobs}+\text{machines}}$.

Table 12.1: Characteristics of the selected instances. Graphical Search Trajectory Networks (STNs) for instances marked with a "◇" are presented in Figure 12.3.

	Code	Jobs	Machines	Attributes	Horizon	Search space size	Winner
	`inst_45`	50	2	2	15,403	$\approx 10^{89}$	LNS
	`inst_50`	50	2	5	3,021	$\approx 10^{89}$	LNS
◇	`inst_61`	100	2	2	1,267	$\approx 10^{204}$	LNS
	`inst_62`	100	2	2	10,304	$\approx 10^{204}$	SA
	`inst_70`	100	2	5	32,861	$\approx 10^{204}$	LNS
	`inst_72`	100	5	2	3,748	$\approx 10^{210}$	SA
◇	`inst_77`	100	5	5	1,135	$\approx 10^{210}$	SA & LNS
	`inst_79`	100	5	5	29,816	$\approx 10^{210}$	SA
	`inst_82`	250	2	2	8,224	$\approx 10^{605}$	LNS
◇	`inst_91`	250	5	2	7,605	$\approx 10^{612}$	SA

12.1.3 Sampling Process

The first step in implementing STNs is to track the representative solutions and their transitions. In our approach, we log any changes in the best solution encountered. We avoid using frequency-based logging (e.g., recording after a given number of steps) since the recording process itself does not impact the algorithm's runtime. In fact, our as SA method is iteration-based and LNS logging procedures require less than 0.0001 seconds to register a new solution. Additionally, frequency-based logging risks missing important information, as only a subset of the solutions may be captured using this approach.

12.1.4 Solution Encoding

A solution to the OSP is represented by two arrays of size equal to the number of jobs. The first array stores, for each job j, the machine m on which it is processed. The second array stores, for each job j, the position b in the schedule of machine m at which it is processed. This representation is sufficient because processing times, batch start times, and other elements of the schedule can be calculated deterministically based on the job's processing order, assigned machine, and the machine's availability and setup times, following an earliest-possible scheduling procedure. This is how both SA and LNS deal with solutions when evaluating their objective value.

STNs require all solutions to be in string format, with all the strings having the same length. For this reason, we concatenate the above-mentioned arrays in a single data structure and convert the numbers to bitstrings.

Figure 12.2 illustrates an example of the solution encoding for a random instance with 10 jobs, 2 machines, and 2 attributes. At the top, we display a human-readable solution representation in the form of a Gantt chart for both machines. The horizontal axis represents time, while the vertical axis denotes machine capacity. The machine's maximum capacity is indicated by a dashed red line, and unavailable time intervals are shown in black. Each job is represented by a rectangle, where the color corresponds to the attribute and the height corresponds to the job's size. Jobs processed in the same batch are stacked, so their combined height reflects the batch's total size. Jobs within a batch begin and end processing simultaneously, with the processing time set by the longest minimum processing time among the jobs. The middle section of Figure 12.2 encodes the solution using two arrays: one specifying the machine assigned to each job and another indicating the processing order. Finally, these two arrays are concatenated into a single array, with colors distinguishing the machine assignment segment from the processing order segment. The bottom portion of Figure 12.2 illustrates this concatenated array in its bitstring form.

Preliminary experiments considering additional elements in the solution representation (e.g., the processing start time, processing duration, etc.) provided results comparable to this minimal solution representation and are therefore not included here.

12.1.5 Search Trajectoriy Networks Analysis

The STN analysis includes two parts, the graphical representation and the metrics. Regarding the first, we interchangeably use networks generated by the Kamada-Kawai [249] and Fruchterman-Reingold [186] algorithms. Both algorithms produce force-directed layouts

with a roughly even distribution of nodes, minimized edge crossings, nearly uniform edge lengths, and preserved symmetries.

Considering the latter, we evaluate the following set of STN metrics:

- **Nodes**: Number of nodes.
- **Edges**: Number of edges.
- **Best**: Number of nodes with the best-found evaluation.
- **Ends**: Number of nodes at the end of the trajectories other than the nodes with the best evaluation.
- **Shared**: Number of nodes visited by more than one algorithm.
- **Strength**: Incoming weighted degree of best nodes normalized by the total number of runs.

We do not consider the number of nodes at the beginning of the trajectories (*initial*) since all algorithms start from the same initial solution.

12.1.6 Search Space Partitioning

STNs built solely from representative solutions (where each location corresponds to a unique solution) often become cluttered and hard to interpret. This issue is especially pronounced in real-world problems where algorithmic paths explore a large number of solutions. To address this, search space partitioning can be employed. This involves mapping similar solutions to a smaller set of locations, based on some measure of proximity or similarity between solutions. In this work, we consider two partitioning strategies, one based on Shannon Entropy (SE) [370] and the other on Hierarchical Agglomerative Clustering (HAC) [89].

The core idea of SE-based search space partitioning is to merge solutions by removing variables from the search space. The procedure is as follows (see [370, Section 5.4] for theoretical details): First, the SE [428] of each decision variable is calculated, based on the values they assume in the list of representative solutions so that variables that present the same values in multiple solutions have lower SE values; this means that they are not so informative. Second, decision variables with lower SE values are removed. The number of variables removed is based on a percentage threshold set by the user.

HAC-based search partitioning is a bottom-up approach. The implementation uses single-linkage and is extended w.r.t. traditional HAC with a mechanism controlling the size (i.e., the percentage of all solutions a cluster can maximally contain) and volume (i.e., the percentage of the covered search space volume the solutions of a cluster span) of the clusters. Both these control values are set by the user (as a percentage w.r.t. the number of representative solutions, see [89, Section 3] for theoretical details). The procedure is as follows: Initially, each data point (i.e., representative solution) is considered as a cluster of size 1. Based on a distance measure that can be specified by the user, the two closest clusters are determined and, if possible in terms of resulting cluster volume and size, are merged into a new cluster. This step is repeated iteratively.

12.2 Experimental Setup

The experimental setup is as follows:

Algorithms The experiments are executed on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8×64GB RDIMM. Each algorithm is executed on a single thread 5 times, each time with a different seed (i.e., from 42 to 46). Both methods are stopped with a 1-hour time limit or after 10,000 idle iterations. The parameters are set as in Chapter 10.

Representations and Metrics To plot STN graphs and extract the metrics, we used the web tool available at the public `GitHub` repository `stnweb`.[1] The tool was run on a MacBook Air M3 with 16GB of memory and macOS Sonoma 14.3.

Search Space Partitioning When dealing with partitioning, we cannot choose the same values for reduction, volume size, or cluster size for all instances as they present different properties (e.g., number of solutions, similarity between solutions, etc.) [370]. In the case of SE, we consider a reduction between 50% and 90% (with 10% steps). In the case of HAC, we consider the Euclidian distance[2] and cluster size and volume between 1% and 5% (with 1% steps). For each instance, the choice was made through a full factorial analysis of the possible combinations, choosing the setting with the most expressive visualization.

12.3 Results

In this section, we report the results of the experimental analysis, considering the metrics and the visual representation as described in the previous section.

Table 12.2 reports the STN metrics distinguishing the instances (label "Code") and the search space partitioning strategy (label "Partitioning", the percentages between brackets correspond to the chosen settings of the related parameters).

In all instances, STNs with no space partitioning (indicated with "—") do not identify areas explored by both algorithms, as the number of shared nodes is always 1 (since all runs start at the same initial solution). When applying SE, the number of nodes decreases slightly; however, the aggregation into locations happens only within representative solutions of the same algorithm (i.e., only representative solutions explored by SA or only representative solutions explored by LNS, therefore *shared*=1). We also conducted experiments increasing the partitioning size (up to 99%), but the results were not different w.r.t. those of lower partitioning percentages. Conversely, when applying HAC as a search partitioning strategy, it is possible to reduce significantly the number of locations and to identify locations visited by both algorithms: for all instances, the number of shared nodes is greater than 1. The difference in behavior between the two partitioning strategies may be attributed to the overestimation inherent in the solution representation using bitstrings. Specifically, HAC shows greater robustness to the zero-padding of strings, while SE is more sensitive.

[1] `https://github.com/camilochs/stnweb`. Accessed 2025–01–03

[2] The only other available option in the `stnweb` tool is the Manhattan distance, which is not suitable in this a case.

Table 12.2: Search Trajectory Network (STN) metrics for the Oven Scheduling Problem (OSP). The STNs for instances marked with a "◇" are presented in Figure 12.3.

Code	Partitioning	Nodes	Edges	Shared	Best	End	Strength
	—	361	360	1	1	9	0.1
`inst_45`	SE (90%)	358	357	1	1	9	0.1
	HAC (1%,1%)	131	184	12	1	9	0.1
	—	276	275	1	5	5	0.5
`inst_50`	SE (90%)	275	274	1	5	5	0.5
	HAC (1%,1%)	139	177	14	4	4	0.7
	—	836	835	1	3	7	0.3
◇ `inst_61`	SE (90%)	830	829	1	3	7	0.3
	HAC (1%,1%)	113	246	10	3	5	0.8
	—	1169	1168	1	1	9	0.1
`inst_62`	SE (90%)	1163	1162	1	1	9	0.1
	HAC (1%,1%)	118	242	15	1	8	0.4
	—	1789	1788	1	1	9	0.1
`inst_70`	SE (90%)	1765	1764	1	1	9	0.1
	HAC (2%,2%)	59	173	4	1	6	0.4
	—	828	827	1	5	5	0.5
`inst_72`	SE (90%)	824	823	1	5	5	0.5
	HAC (1%,1%)	119	180	13	3	5	1.1
	—	837	836	1	5	5	0.5
◇ `inst_77`	SE (90%)	833	832	1	5	5	0.5
	HAC (1%,2%)	121	215	10	3	4	1.4
	—	950	949	1	1	9	0.1
`inst_79`	SE (90%)	940	939	1	1	9	0.1
	HAC (1%,1%)	169	220	9	1	8	0.6
	—	553	552	1	1	9	0.1
`inst_82`	SE (90%)	553	552	1	1	9	0.1
	HAC (1%,1%)	117	247	7	1	9	0.1
	—	1391	1390	1	1	9	0.1
◇ `inst_91`	SE (90%)	1385	1384	1	1	9	0.1
	HAC (2%,2%)	57	104	3	1	9	0.4

Considering the number of best nodes, four instances (`inst_50`, `inst_61`, `inst_72`, and `inst_77`) have more than one best solution. Only in one case (`inst_61`), all best solutions belong to different locations of the search space (i.e., before and after the HAC partitioning the number of best solutions does not decrease). In the other three cases, the best solutions do not belong to a single location either, since the number of best nodes is always greater than 1 after partitioning. This is an indicator of a multi-modal landscape with high-quality solutions scattered across it. The strength values of best solutions are higher in the case of search space partitioning with HAC, meaning that the best nodes have many connections with other nodes. In this case, we can assume that the location where the best nodes lies is linked to several other locations and is therefore relatively easy to reach.

Due to space limitations, we present only the network representations for a subset of the selected instances (Figure 12.3). The networks are constructed with distinct colors, shapes, and sizes to convey specific information:

- Each algorithm is represented by a unique color; the trajectories of SA appear in gray,

while those of LNS appear in a lighter shade of gray.

- Nodes visited by both algorithms are indicated by black circles.
- Nodes representing the best solutions are indicated by white circles.
- Starting nodes are represented by black squares.
- End nodes are represented by black triangles.
- The size of a node is proportional to the number of visits paid to that node.

The selected instances present different characteristics: one where one algorithm outperforms the other, but both algorithms visit the locations of the best solutions according to HAC space partitioning (`inst_61`); one where both algorithms reach optimal solutions (`inst_77`); one where one algorithm is superior and only it visits the best solution locations (`inst_91`). We consider the networks without search space partitioning (Figure 12.3, images A, C, and E) and those partitioned using HAC (Figure 12.3, images B, D, and F). Results partitioned with SE are omitted, as it does not identify shared locations.

In all cases, SA trajectories appear longer than those of LNS: this is justified by the algorithm procedures themselves, since SA randomly explores the search space with small and quick changes, whereas LNS considers sub-problems iteratively, performing fewer but larger changes on the current solution.

HAC partitioning highlights that SA tends to visit more representative solutions within the same locations (i.e., the nodes are larger); on the contrary, LNS tends to visit representative solutions in different locations of the space. The majority of the shared locations are within the initial part of the search, this is justified by the fact that both algorithms start from the same solution (i.e., both algorithms construct the initial solution using the algorithm proposed by M. Lackner et al. [275], see also Section 3.4). Finally, the end nodes also appear to be visited multiple times (i.e., they are larger in size than other nodes), suggesting that the algorithms may become trapped in sub-optimal regions, where good but not optimal solutions are located.

12.4 Concluding Remarks

In this concluding section, we summarize the content of the chapter (Section 12.4.1), discuss limitations (Section 12.4.2), and outline future research directions (Section 12.4.3).

12.4.1 Content

In this work, we applied STNs to analyze the behavior of two state-of-the-art algorithms, SA and LNS, for solving a real-world parallel batch scheduling problem, the OSP. We considered a subset of 10 instances taken from the literature and represented STNs with and without search space partitioning. Considering the search partitioning strategies, the HAC can uncover the behavior of the algorithms better, as it is able to detect shared locations between the algorithms.

The search space of the instances is extensive (up to 1,391 unique visited nodes) and displays signs of multi-modality with high-quality solutions appearing in different space

locations. While the algorithms explore different areas within the space, most shared locations occurring early in the search; SA frequently revisits the same locations multiple times, whereas LNS visits fewer representative solutions, typically situated in distinct locations. This observation indicates that it may be beneficial to combine both algorithms in a portfolio or hybrid method.

12.4.2 Limitations

Despite offering valuable insights into the dynamics of metaheuristics, STNs have notable limitations. First, constructing interpretable networks is labour-intensive and only partially automatable: assessing search partitioning typically relies on visual inspection, which is time-consuming and restricts analyses to small subsets of instances. Second, while search-space partitioning is essential, especially in large spaces, it introduces an additional modelling layer; conclusions can be sensitive to the partitioning method and its hyperparameters, so STNs are not an off-the-shelf tool. Third, although HAC often yields coherent partitions, it scales poorly to large solution sets because it requires computing and storing the full pairwise distance matrix.

12.4.3 Future Work

Future work proceeds along two lines. First, we will complement the present analysis with Local Optima Networks (Local Optima Networks (LONs)) to contrast global landscape connectivity with the transition patterns captured by STNs.

Second, we will extend the STN study along different directions:

- **Scalability of clustering**: Evaluate how HAC performs on larger solution sets and possibily approximate the results via heuristic methods such as Approximate Nearest Neighbor (ANN).
- **Representation effects**: Compare alternative schedule/trajectory representations and measure their impact on clustering and the induced STNs.
- **Semantic alignment**: Quantify the agreement between STN clusters and human-readable descriptors of the solutions.

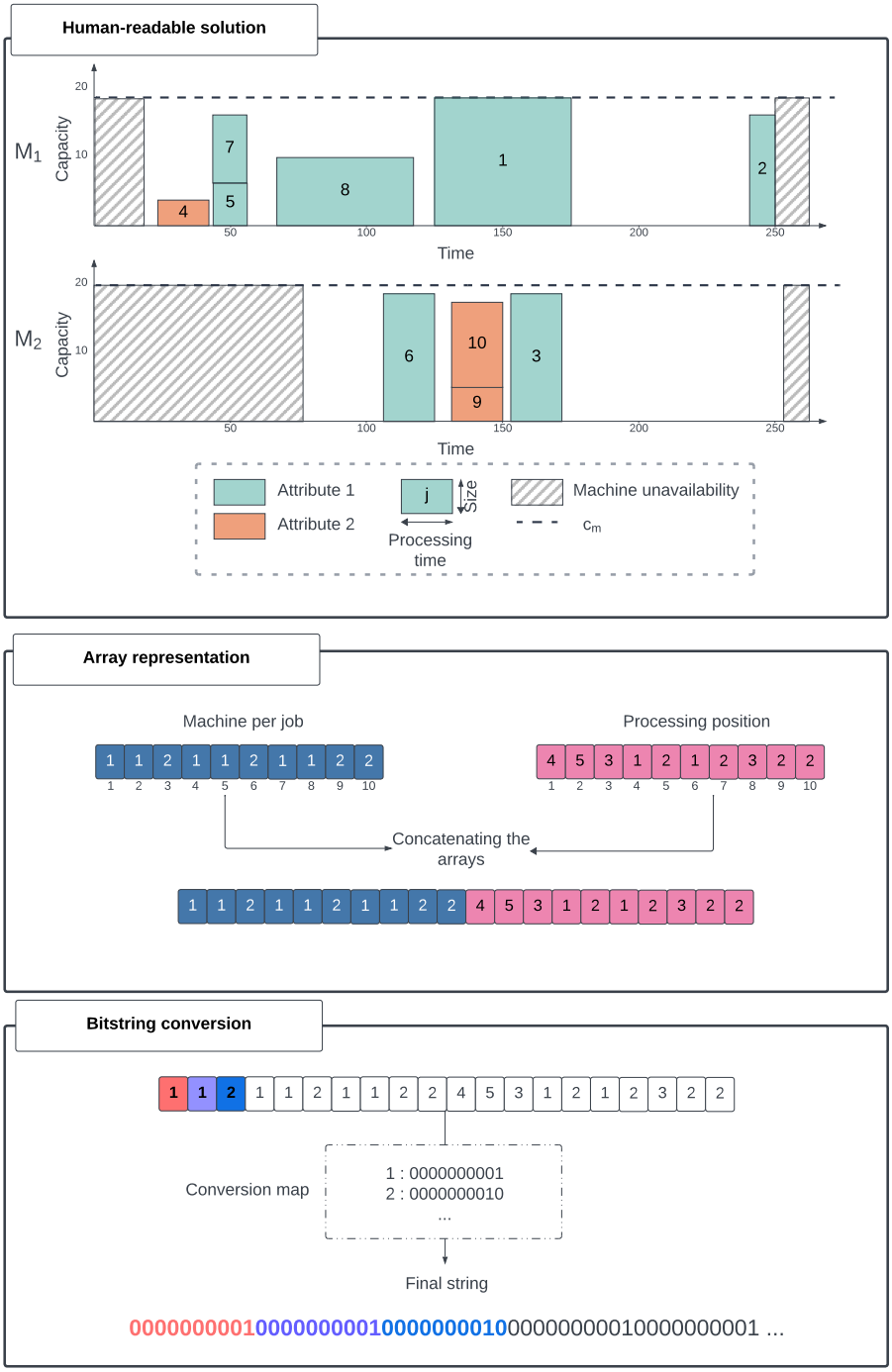


Figure 12.2: Solution representation for building Search Trajectory Networks (STNs) for the Oven Scheduling Problem (OSP).

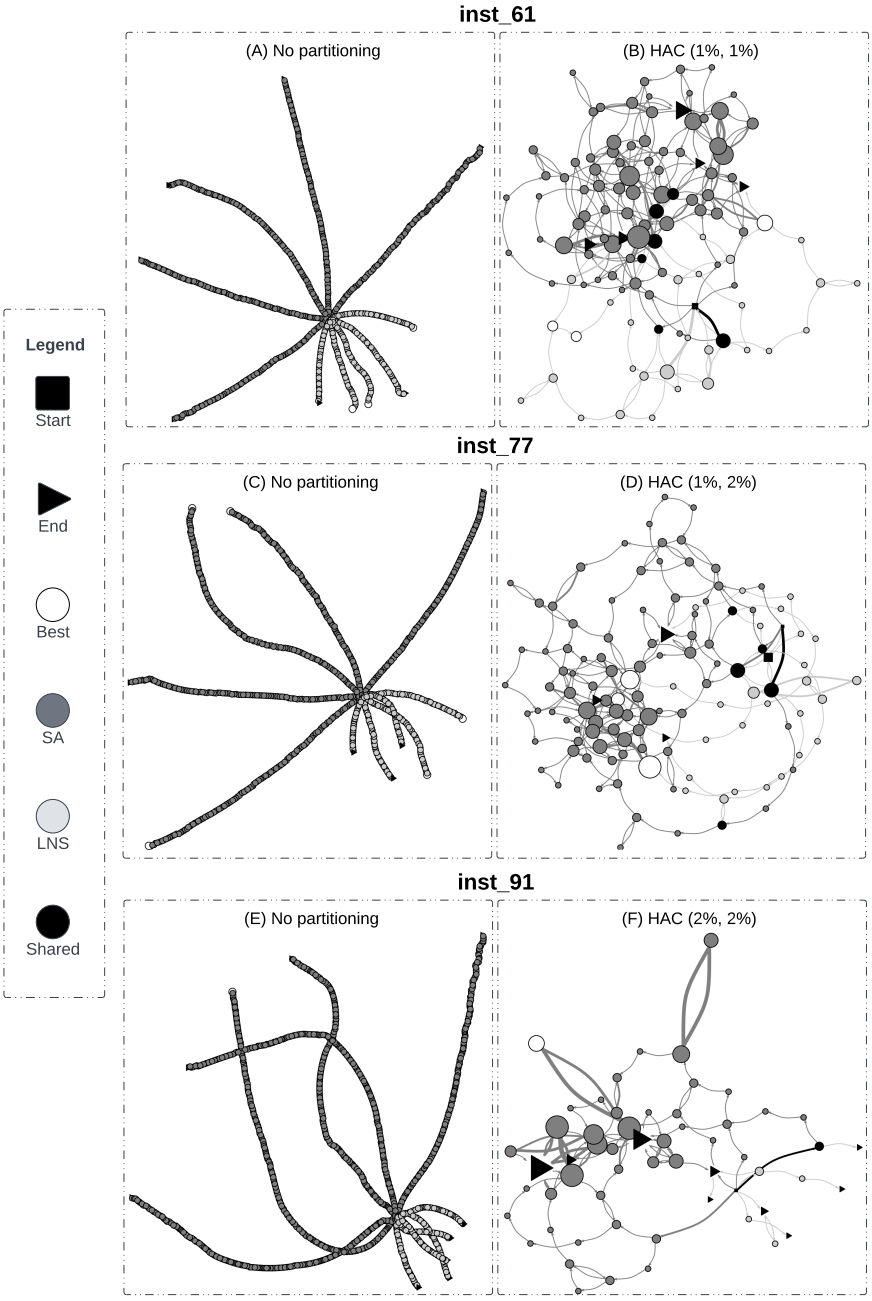


Figure 12.3: Search trajectory networks (STNs) for selected instances of the Oven Scheduling Problem (OSP).

Part IV

Home Healthcare Routing and Scheduling Problem

Chapter 13

Introduction

Home Healthcare (HHC) constitutes an essential component of modern healthcare systems. It provides patients with personalized, accessible, and continuous care in the comfort of their homes, extending services beyond traditional medical facilities such as hospitals and nursing homes. It is particularly beneficial for seniors, individuals with disabilities, and those recovering from illness or surgery, as it includes services like medical care (e.g., nursing, physical therapy, medication administration), assistance with activities of daily living (e.g., bathing, meal preparation), and emotional support through companionship. Compared to institutional care, HHC is typically more cost-effective [196], alleviates pressure on hospitals and family caregivers, and enables individuals to receive care in familiar environments that support their autonomy and well-being.

The operational model of HHC involves specialized caregivers visiting patients' homes to provide services before moving on to subsequent appointments (Figure 13.1). Unlike conventional healthcare scheduling [79, 478], this setting requires the simultaneous coordination of *who* performs the service, *when* it occurs, and *how* caregivers travel between patients. The resulting Home Healthcare Routing and Scheduling Problem (HHCRSP) integrates routing and scheduling decisions into a single optimization problem, and has attracted attention in the Operations Research (OR) literature [170, 176, 187].

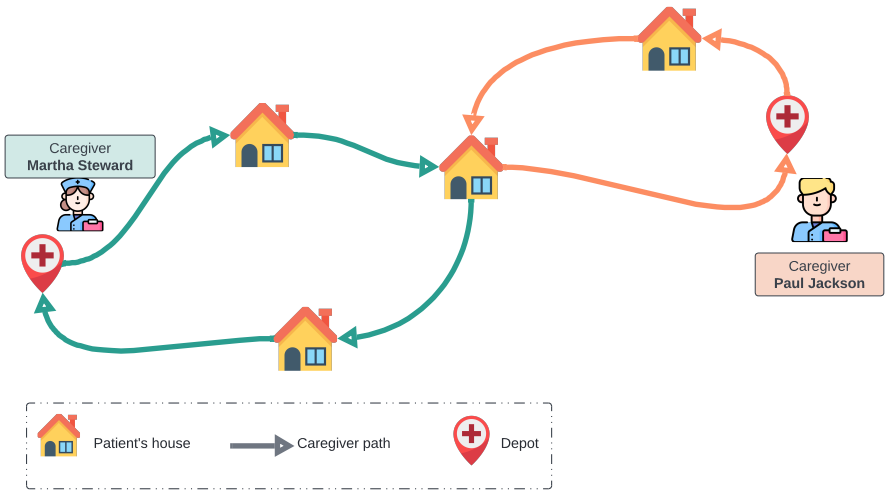


Figure 13.1: Naive representation of the operational model of Home Healtchare (HHC).

Over the past decade, several HHCRSP formulations have been proposed, yet relatively few capture multiple real-world characteristics simultaneously, limiting their applicability in practice. Moreover, the scarcity of publicly available datasets, often due to privacy concerns, has hindered reproducibility and systematic comparison across studies. Among the available models, the formulation of Mankowska, Meisel, and Bierwirth [327] has become a reference [80, 279, 354, 363, 364], partly because it is accompanied by a public benchmark dataset. State of the art methods are a Simulated Annealing (SA) algorithm by Ceschia et al. [80] and a Mixed Integer Linear Programming (MILP) by Montemanni, Ceschia, and Schaerf [354]

While Mankowska, Meisel, and Bierwirth [327] formulation has provided a valuable foundation for benchmarking, it still omits several key aspects of practice, including multiple caregiver departure points, shift structures and overtime, optional patient visits, and caregiver–patient incompatibilities. Other formulations in the literature consider some of these aspects individually, but not within a unified framework, and their corresponding datasets are often unpublished or inaccessible.

13.1 Goals

This work aims to extend the specifications from Mankowska, Meisel, and Bierwirth [327] by incorporating the most significant and practically relevant features proposed across various HHCRSP formulations in the literature. We present a unified framework that positions the existing specifications as specific subclasses, while enriching them with additional entities and concepts that address a broader spectrum of real-world constraints. To support this enhanced formulation, we provide a comprehensive collection of artificial yet realistic instances that facilitate rigorous benchmarking within this unifying framework. More programmatically, this part of the thesis pursues four goals:

1. **Unified formulation**: Extend the specifications of Mankowska, Meisel, and Bierwirth [327] into a single, backward-compatible HHCRSP formulation that captures the most practically relevant constraints reported in the literature.

2. **Reproducible infrastructure**: Provide a standardized, machine- and human-readable `json` schema for both instances and solutions together with an instance generator, feature extractor, and solution validator to enable fair, repeatable benchmarking.

3. **Benchmark coverage**: Translate existing datasets to the unified schema and release new, diverse instances that broaden coverage of realistic problem characteristics.

4. **Methodological assessment**: Generalize and extend state of the art solution methods to the unified formulation and evaluate their performance across the expanded instance space.

These goals can be formulated as the following Research Questions (RQs):

RQ1 **Unified formulation and schema adequacy:** Can the proposed unified formulation subsume Mankowska, Meisel, and Bierwirth [327] and other widely used variants while preserving backward compatibility with public benchmarks? Does the proposed schema, together with the generator, feature extractor, and validator,

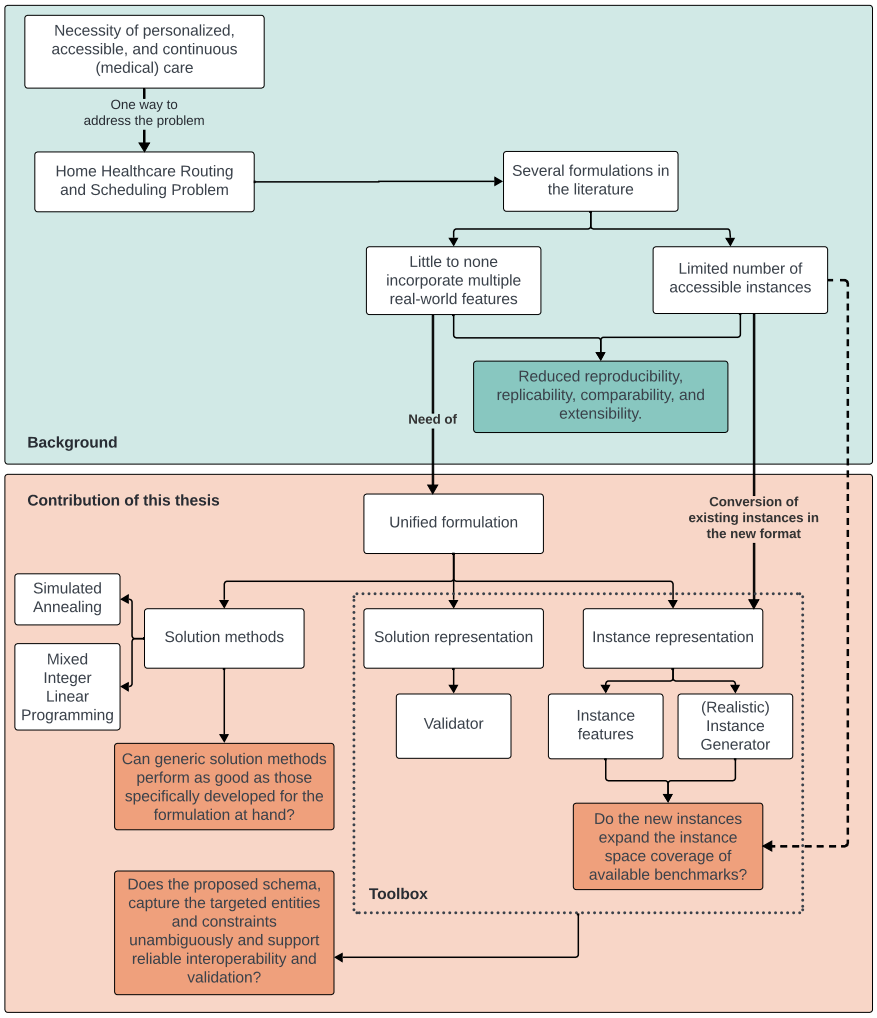


Figure 13.2: Schematic representation of the goals and content of this part of the thesis.

capture the targeted entities and constraints unambiguously and support reliable interoperability and validation?

RQ2 **Instance-space coverage:** Do the new instances expand coverage of the instance space relative to existing benchmarks (i.e., increase coverage and diversity in the defined feature space)?

RQ3 **Generic vs. specialised methods:** Can general-purpose solution approaches perform as well as specialized algorithms tailored to specific formulations?

To address **RQ1**, we introduce a unified HHCRSP formulation (natural language specification and mathematical model; Chapter 14), define standardized `json` input/output schemas, provide tooling (generator, feature extractor, validator), and translate existing benchmarks into this format.

To address **RQ2**, we analyze the instance space of both translated and newly generated datasets to quantify coverage and diversity (Chapter 14).

To address **RQ3**, we generalize and extend state of the art methods to the unified formulation and evaluate them on translated and new instances, enabling like-for-like comparisons with specialised methods reported in prior work where available (Chapter 15).

The investigation addressing these RQs are presented originally presented as part of the teaching material of the author of this thesis for the first ROAR-NET Training School and as published material in:

- Ceschia, S., **Da Ros, F.**, Di Gaspero, L., Mancini, S., Maniezzo, V., Montemanni, R., Rosati, R. M., & Schaerf, A. (2025, submitted). A unified formulation for home healthcare routing and scheduling problems. *International Transactions in Operational Research*. 1–47. Journal rank: Scimago Q1 (2024).

Considering the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action, the work presented in this chapter is relevant to two working groups: *Working Group 1* (Problem Modelling and User Experience) and the *Working Group 6* (Benchmarking).

Figure 13.2 provides an overview of the structure of this part of the thesis. Background on the HHCRSP is highlighted in blue, while the original contributions of this thesis are marked in orange.

Pre-claimer. While this work addresses a longstanding obstacle in optimization research, i.e., the proliferation of formulations and input formats across closely related variants, it also has limitations. Our feature set is anchored to the specification of Mankowska, Meisel, and Bierwirth [327] and to characteristics that can be derived from public (or requestable) datasets; as such, it cannot exhaust the breadth of requirements encountered in industrial deployments. Although our unified formulation already incorporates more than ten additional real-world characteristics, further context-specific aspects will inevitably arise. However, to accommodate this diversity, the framework is *modular by design*. The unified formulation decomposes entities and constraints into independent components; the `json` schema uses optional, typed sections; and the instance generator, feature extractor, and validator expose plug-in interfaces. This enables practitioners to incorporate new aspects (e.g., additional resources, precedence relations, alternative objectives, or stochastic arrivals) without breaking backward compatibility, and to extend the feature set as new signals become available.

13.2 Structure

This part of the thesis is organized as follows. Chapter 14 introduces the problem, outlining the instance and solution representations together with the reproducibility toolbox. Chapter 15 presents the solution methods and their empirical evaluation.

Chapter 14

Problem Definition

The literature on Home Healthcare Routing and Scheduling Problem (HHCRSP) includes numerous formulations, each emphasizing different constraints and objectives. However, relatively few formulations integrate multiple real-world features simultaneously, limiting practical applicability. Compounding this, problem data and benchmark instances are seldom publicly available (largely due to privacy concerns) hindering replication, comparability, and cumulative progress.

This chapter presents a unified, modular, and extensible formulation of the HHCRSP. The formulation positions prominent specifications as special cases while exposing optional components for additional constraints and objectives. We also define canonical instance and solution representations and provide an instance generator. As in the case of the Oven Scheduling Problem (OSP) (Chapter 8) and in compliance with guidelines by the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action, we provide two complementary perspectives: a lay description, which offers an intuitive overview of the problem's logic and practical implications, and a formal definition, which rigorously defines the problem.

The contributions of this chapter are:

- **Unified, modular formulation.** A single model that subsumes the basic HHCRSP and incorporates multiple real-world features previously treated in isolation.
- **Standardised representations.** A modular `json` schema for instances and solutions, with clear semantics and validation rules.
- **Benchmark translation**: Conversion of existing instances into the unified schema.
- **Instance generation and coverage analysis.** An instance generator and a new dataset; a comparative analysis via Principle Component Analysis (PCA) demonstrating coverage of, and extension beyond, existing instance spaces.
- **Evaluation framework.** A complete, reproducible infrastructure in `Python` for loading, validating, and evaluating instances and solutions.

This chapter is organised as follows. Section 14.1 introduces an informal (lay) description of the problem, Section 14.2 presents its formal definition, Section 14.3 defines the instance and solution representations, and Section 14.4 outlines the toolbox architecture, describing how the instance-format converter, generator, and related components interconnect.

14.1 Lay Problem Definition

In this section, we define the essential entities of the HHCRSP model (Section 14.1.1), extend them with additional elements that reflect the operational complexities of real-world settings (Section 14.1.2), and conclude by categorizing the cost components and constraint violations that shape the objective function (Section 14.1.3).

14.1.1 Essential Entities

Essential entities constitute the core elements upon which all problem variants are built, regardless of their specific operational contexts or additional constraints.

Patients

Patients are the recipients of Home Healthcare (HHC) services, each characterized by a set of attributes that may influence service delivery planning. Each patient is associated with a specific geographic position (i.e., their home), which is the destination for caregivers' visits. The spatial distribution of patients, combined with the underlying road and transportation networks, determines travel times between locations.

The patients' temporal availability is represented by at least one time window, depending on personal schedules, preferences, or medical requirements. Time windows establish constraints within which services must be delivered.

Services

Services represent the specific care activities that must be delivered to patients, spanning a broad spectrum of medical, therapeutic, and supportive interventions. Each service is characterized by its nature and purpose, ranging from clinical procedures such as medication administration or wound care to supportive functions like personal hygiene assistance or rehabilitation exercises. The diversity of services reflects the multidimensional nature of HHC delivery and requires the appropriate matching of patient needs with caregiver capabilities.

Several parameters define the temporal dimension of services. Service duration specifies the time required to complete the activity, which may vary based on patient condition, service complexity, or caregiver experience. For time-sensitive services, such as insulin administration or collecting biological samples, precise timing is critical to ensure clinical efficacy or specimen viability. Patients often require multiple services, creating interdependencies that must be carefully coordinated. Services may be subject to precedence relationships, where certain activities must precede others to maintain care continuity and effectiveness. The synchronization of services, mainly when multiple caregivers must work in tandem, introduces additional complexity to the scheduling problem.

Caregivers

Caregivers are the workforce responsible for delivering HHC services to patients. Each caregiver has a unique professional profile defined by their qualifications (also called specializations or competencies). Qualifications are essential in HHC planning, as not all

caregivers can perform all services. For instance, a nurse caregiver specialized in wound care may not be qualified to conduct physical therapy, while a physiotherapist caregiver lacks the credentials to administer certain medications.

Each caregiver typically operates within defined working time shifts, establishing their availability throughout the planning horizon. These shifts may incorporate mandatory lunch breaks and allow for overtime under the limits of labor regulations.

Mobility aspects further characterize caregivers within the HHC system. Each caregiver begins and ends their workday at designated locations, which may be their home, a healthcare facility, or another fixed point. Their movement between patient locations is governed by available transportation modes, which influence travel times and operational costs. In this formulation, we consider only one transportation mode, i.e., individual cars.

Scheduling Horizon

The scheduling horizon represents the temporal framework within which HHC operations are planned and executed. In our setting, we restrict it to a single day and time granularity based on minutes.

Core Feasibility Criteria

The essential feasibility criteria for a solution to the HHCRSP problem are as follows. A service cannot be provided by a caregiver who is not qualified for it. This ensures that patients receive care only from caregivers with the appropriate skills and certifications required for their medical needs. Moreover, a service cannot start before the patient's time window begins. In case of early arrival, the caregiver has to wait until the time window starts. This temporal constraint respects patient availability and preferences while ensuring service delivery occurs when patients are prepared to receive care.

14.1.2 Additional Elements

We extend the essential entities with additional elements that capture the operational complexities of real-world HHC settings and bridge the gap between the academic formulations and the practice.

Time Windows

Patient availability is defined through either a *single* or *multiple time windows*. When patients have multiple time windows and require multiple services, as in Bazirha, Benmansour, and Kadrani [40] and Bazirha, Kadrani, and Benmansour [41], all services must be delivered within a single selected time window, rather than distributed across different periods.

Time window enforcement can be approached in two primary ways: the standard approach requires only that the service begins within the designated time window, allowing the service to potentially extend beyond the window's end [327]; alternatively, both the beginning and end of the service must occur within the specified time frame (*strict time windows* [40, 41]).

Synchronization

For patients necessitating multiple services, the delivery pattern may take various forms to accommodate clinical requirements and operational constraints. Services may be provided *simultaneously*, with multiple caregivers attending to the patient at the same time; *sequentially*, where one service must follow another; or *independently*, allowing services to be scheduled without interdependence.

Sequential double-service patients require a specific minimum and maximum time gap between services. This constraint expresses a relationship between services; e.g., a medication must be administered at least a certain time after a meal but before a specific elapsed time to ensure therapeutic efficacy.

Work Shift

The caregivers' schedule is structured around defined *shifts*, which may either consist of a complete workday (full-time) or segmented periods (part-time). Each caregiver begins their service route from a designated location at or after their shift's start and is expected to return by the shift's end. When a caregiver's return time extends beyond their scheduled shift end, this constitutes *overtime* (Section 14.1.3).

Caregiver Break

Caregiver schedules must accommodate rest periods, particularly during extended shifts. Caregivers working through midday require a *lunch break* that must begin within a specified time window and last for a predetermined duration. These breaks may occur near client locations or at designated facilities, with any necessary movement incorporated into the overall travel time calculations.

Departure and Arrival Points

In practical HHC operations, it is unrealistic to assume that all caregivers begin their shifts at a central office location. Consequently, we allow caregivers to start their routes from either the central healthcare facility or their residences and subsequently reach their arrival point upon shift completion.

Patients / Caregivers Relationships

The relationship between caregivers and patients extends beyond simple service provision, including interpersonal dynamics that can impact care quality and patient satisfaction. These relationships are expressed in the form of:

- **Incompatibility**: Specific patient-caregiver pairings may be unsuitable due to various personal or medical factors. For instance, a nurse with a cat allergy would be unable to provide care in a home with feline pets, or a patient might request caregivers of a specific gender for personal care services as a firm requirement. These restrictions are included through specific patient-caregiver combinations declared as *incompatible*, thereby preventing these assignments in the resulting schedule.

- **Preferences**: Beyond strict incompatibilities, the quality of home healthcare is significantly influenced by patient preferences for particular caregivers. These preferences may stem from various factors, e.g., communication style, and are defined as sets of preferred caregivers for a given patient.

Optional patients

Often, real-world cases involve situations where the available caregiver capacity cannot satisfy the total patient demand. Within this context, we distinguish between two patient categories based on service urgency. *Mandatory* patients must receive all scheduled services within the current planning horizon without exception. Conversely, *optional* patients may have their care postponed to subsequent planning periods when service capacity is insufficient. Partial service delivery is not possible: a patient is adequately deemed served when all their required services have been successfully scheduled.

14.1.3 Cost Components and Violations

In a general formulation of the HHCRSP, various operational aspects can be incorporated either as *hard* constraints that must be satisfied for a solution to be feasible or as *soft* constraints that may be violated with associated penalty costs in the objective function. The following cost components and potential constraint violations capture operational efficiency metrics and quality-of-service factors. The specific implementation of these elements as either hard constraints or weighted penalty terms depends on organizational priorities and operational requirements. Section 14.3 illustrates specific modeling choices and provides a flexible framework that accommodates various approaches to handling these constraints.

- **Travel time:** represents the duration caregivers need to move between locations. It includes all movement within a caregiver's route, including travel from the starting location to the first patient, between consecutive patients, and from the final patient back to the ending location. This component directly influences operational costs through fuel consumption, vehicle wear, and productive time utilization.

- **Waiting time:** occurs when a caregiver arrives at a patient's location before their time window begins. During this period, the caregiver must wait until the earliest permissible service start time, resulting in unproductive time that nonetheless counts toward their work hours. This component represents inefficiency in schedule coordination between caregiver availability and patient time windows.

- **Tardiness:** measures the delay in service delivery beyond the scheduled or promised time. This can occur when a caregiver arrives (in relaxed time window scenario) or completes (in strict time window scenarios) a service after the end of the time window. Tardiness negatively impacts patient satisfaction and may compromise care quality for time-sensitive services.

- **Overtime:** represents work performed beyond a caregiver's shift duration. It occurs when a caregiver's return time to their ending location exceeds their shift end time.

This component typically incurs premium labor costs and may contribute to caregiver fatigue and decreased job satisfaction.

- **Idle time:** represents the sum of all time periods within a caregiver's shift that are not used for patient service, travel, or required lunch break. It is calculated as the total shift duration minus the combined service, travel, and lunch time. Specifically, idle time includes three components: waiting time that occurs when a caregiver arrives early at a patient location; the time interval between shift start and departure to the first patient; and the period between returning to the ending location and the conclusion of the shift (only when this return occurs before the shift ends). This measure captures all non-productive periods within a caregiver's scheduled shift and represents potential inefficiency in resource utilization. It also serves as an indicator of fairness when considering its distribution across caregivers. Significant disparities in idle time may lead to unequal work experiences, where some caregivers face fragmented or underutilized shifts, impacting job satisfaction and perceived equity.

- **Unscheduled optional patients:** quantifies the penalty for not providing service to optional patients within the current planning horizon. While these patients can have their care postponed, each postponement potentially impacts their satisfaction.

- **Unmet preferences:** measures violations of patient preferences regarding their caregivers. When a patient is assigned a non-preferred caregiver despite having expressed specific preferences, this component captures the resulting patient dissatisfaction or potential negative impact on care effectiveness.

- **Ignored incompatibilities:** measures violations of defined incompatibility relationships between patients and caregivers. Similar to unmet preferences, these violations result in patient dissatisfaction, but typically with more severe consequences for the care relationship.

- **Qualifications requirements:** measures violations for assigning caregivers to services for which they lack the necessary qualifications. Such assignments compromise care quality, patient safety, and regulatory compliance.

- **Missed lunch break:** represents the penalty incurred when a caregiver's schedule fails to accommodate the required lunch break within the designated time window. As this affects caregiver well-being and may violate labor regulations, the associated cost reflects regulatory compliance risks and potential negative impacts on caregiver performance and retention.

- **Workload balance:** is a fairness-oriented cost component that considers the deviation of individual caregiver's working time with respect to the average working time among all caregivers. This component promotes an equitable distribution of work.

Table 14.1 categorizes the various cost components according to the stakeholders primarily affected by them. Each component is attributed to one or more of three key stakeholder groups, i.e., caregivers, patients, and the healthcare organization itself. In addition, each component is linked to one or more operational objectives, classified into three categories:

Table 14.1: Attribution of cost components to stakeholders and classification by operational goals. The column "MILP" reports a reference to the cost components of the formal model described in Section 14.2.

		Stakeholder			Operational goals		
Cost component	MILP	Caregiver	Patient	Organization	Efficiency	Fairness	Well-being
Travel time	(CC1)			✓	✓		
Overtime	(CC4)	✓			✓		✓
Tardiness	(CC2),(CC3)		✓			✓	✓
Waiting time	(CC5)			✓	✓		
Idle time	(CC6)			✓	✓	✓	
Unscheduled optional patients	(CC10)		✓		✓		✓
Unmet preferences	(CC8)		✓				✓
Ignored incompatibilities		✓	✓				✓
Qualifications requirements			✓				✓
Missed lunch break	(CC9)	✓					✓
Workload balance	(CC7)	✓				✓	

efficiency, which relates to overall costs and resource usage; fairness, which concerns the equitable distribution of workload and service among individuals (both, caregivers and patients); and individual well-being, which reflects the quality of experience or outcomes for each person involved, particularly focusing on patient satisfaction and caregiver comfort. Specifically, patients' and caregivers' preferences are handled through multiple modeling choices, which are briefly discussed in Section 14.2.3.

In Section 15.3.3, we provide some managerial insights gained by ad-hoc experiments with alternative weights for the different cost components, specifically tailored to shed light on the various trade-offs among different stakeholders and operational objectives.

14.2 Formal Problem Definition

Following the conceptual framework introduced in Section 14.1, we present a rigorous mathematical formulation of the HHCRSP. Section 14.2.1 introduces the required notation, Section 14.2.2 specifies the objective function and constraints, and Section 14.2.3 details the treatment of preferences.

14.2.1 Preliminaries and Notation

Each caregiver $v \in V$ can perform only a subset of the services in S. This is specified by the binary parameter γ_{vs}, which equals 1 if caregiver v is qualified to provide service $s \in S$, and 0 otherwise. In addition, each caregiver $v \in V$ is associated with a location σ_v, representing the point where the caregiver both starts and ends their working shift. The working shift of caregiver v is defined by the time interval $[\alpha_v, \beta_v]$. A set $V_{lb} \subseteq V$ of caregivers is assigned a time window for a lunch break.

We define the node set $C'_0 = C' \cup \{0\}$, which includes the locations of all patients and an additional node 0, used as a placeholder to represent the start and end location of the caregiver under consideration. For any two nodes $i, j \in C'_0$, the travel time between them is denoted by d_{ij} when both i and j correspond to patient locations (i.e., $i, j \neq 0$). If one of the indices is 0, the placeholder is replaced by the actual starting/ending location σ_v of the relevant caregiver.

The set $C'_{opt} \subseteq C'$ represents the optional patients. To account for lunch breaks, an additional set $C^{lb} = \{c_v^{lb} : v \in V_{lb}\}$ is included in C'_{opt}, where each element corresponds to an artificial patient associated with a caregiver. These artificial patients are defined so that their service time matches the duration of the lunch break, their time windows coincide with the allocated lunch break interval, and they can only be served by their respective caregiver. Furthermore, they are assumed to be located at distance zero from all patients and depot locations, and their service must strictly begin within the assigned time window, without any delay.

The set of patients C' is divided into those requiring a single service C_1 and those requiring two distinct services C_2. For C_2 patients, two different caregivers must provide the services. For each patient $i \in C'$, let $S_i \subseteq S$ denote the set of services required by that patient. By assumption, each patient requires at least one and at most two services, that is, $1 \leq |S_i| \leq 2$. For technical convenience, we define $S_0 = S$. Furthermore, let $V_i \subseteq V$ be the set of caregivers qualified to provide at least one of the services in S_i, taking into account the patient–caregiver incompatibilities specified in the input.

For each patient $i \in C_2$ who requires two services, we impose both a minimum and a maximum separation time ($\delta_i^{\min}$ and $\delta_i^{\max} \geq \delta_i^{\min}$) between the two services. Specifically, if the first service begins at time t, then the second must begin within the interval $[t + \delta_i^{\min}, t + \delta_i^{\max}]$. A precedence relation is therefore established between the two services. In the special case where $\delta_i^{\min} = \delta_i^{\max} = 0$, the two services must start simultaneously. Moreover, it is assumed that the two services required by a double-service patient must be carried out by different caregivers.

Given a patient $i \in C'$ and a required service $s \in S_i$, the corresponding service duration is denoted by p'_{is}. Patients may also specify a set of preferred caregivers P_i; if no preference is given, P_i is assumed to include all caregivers qualified to provide service to i. Each patient i is further associated with a non empty set of alternative time windows $T_i = \{[e_i^1, l_i^1], \ldots, [e_i^k, l_i^k]\}$, which indicate the intervals during which service must begin. If a caregiver arrives before the opening time e_i^k, they are required to wait until the time window starts. All services assigned to the same patient must take place within the same time window.

The problem addresses the spatial and temporal planning of caregivers' tours to ensure that patients receive the required treatments, while minimizing a multi-component objective function. The specific cost components and their weights can be flexibly adjusted to different operational contexts, allowing the formulation to capture most variants found in the literature through suitable parameter choices.

The cost components we consider are the following:

(CC1) Total time distance travelled by the caregivers.

(CC2) Total tardiness (delay) in treatments over all patients.

(CC3) Maximum tardiness in treatments over all patients.

(CC4) Total overtime carried out by the caregivers.

(CC5) Total waiting time at patients' locations spent by the caregivers.

(CC6) Maximum idle time over all caregivers.

(CC7) Total absolute deviation of working time of caregivers with respect to the average working time.

(CC8) Total penalty paid due to patients visited by caregivers different from the preferred ones.

(CC9) Total penalty accumulated by caregivers who are skipping their lunch breaks.

(CC10) Total penalty accumulated by not visiting optional patients.

While our formulation provides flexibility in defining the objective function, certain aspects described in Section 14.1.3 are treated as hard constraints rather than penalty terms. Specifically, patients/caregiver incompatibilities and qualification requirements are mandatory restrictions that must be satisfied in any feasible solution. On the other hand, other aspects, such as caregiver late arrivals with respect to patient's time windows, overtime, and missed lunch breaks, are accounted for as penalties.

Table 14.2: Sets of the Mixed Integer Linear Programming (MILP) model.

Symbol	Reference set	Description
C'		Original patients.
V		Caregivers.
$S = S_0$		Services.
C'_0	$C' \cup \{0\}$	Locations of the original patients plus a placeholder location for the caregivers.
V_{lb}	$V_{lb} \subseteq V$	Caregivers assigned with lunch break.
C^{lb}	V_{lb}	Artificial patients modeling the lunch breaks.
C'_{opt}	$C'_{opt} \subseteq C'$	Optional patients considering the initial set of patients.
C_1	$C_1 \subseteq C'$	Patients requiring one service.
C_2	$C_2 \subseteq C'$	Patients requiring two services.
C_3	$C_3 \subseteq C$	Set with the same cardinality of C_2, used to divide the double-services of C_2.
C	$C' \cup C_3$	All patients real and fictitious, including those with divided double-service and the artificial ones used to map the lunch breaks.
C_0	$C \cup \{0\}$	Locations of the patients plus a placeholder location for the caregivers.
C_{opt}	$C_{opt} \subseteq C$	Optional patients.
V_i	$V_i \subseteq V, i \in C$	Caregivers that can provide services to patient i.
T_i	$i \in C$	Time windows in which patient i is available.
S_i	$S_i \subseteq S, i \in C$	Services required by patient i.
P_i	$P_i \subseteq V, i \in C$	Caregivers preferred by patient i
C^v	$\{i \in C : v \in S_i\}, v \in V$	Patients that might be served by caregiver v
C^v_0	$C^v_0 \cup \{0\}, v \in V$	Locations of the patients that might be served by caregiver v plus a placeholder location for the caregiver.

14.2.2 Model

The model is an extension of that originally proposed in Montemanni, Ceschia, and Schaerf [354] and Montemanni [353].

Table 14.3: Parameters of Mixed Integer Linear Programming (MILP) model.

Symbol	Reference set	Description
p'_{is}	$i \in C'$, $s \in S$	Duration of service s for patient i.
p_i	$i \in C$	Duration of service for patient i.
e_i^k	$i \in C$, $k \in \{1, \ldots, \lvert T_i \rvert\}$	Start of time window k of patient i.
l_i^k	$i \in C$, $k \in \{1, \ldots, \lvert T_i \rvert\}$	End of time window k of patient i.
$\delta_i^{\min}$	$i \in C$	Min. time separation between the services of patient i (if double service)
$\delta_i^{\max}$	$i \in C$	Max. time separation between the services of patient i (if double service)
σ_v	$v \in V$	Location of caregiver v.
α_v	$v \in V$	Start time of the working shift of caregiver v.
β_v	$v \in V$	End time of the working shift of caregiver v.
γ_{vs}	$v \in V$, $s \in S$	1, if caregiver v can perform service s, 0 otherwise.
d_{ij}	$i, j \in C_0$	Travel time between generic nodes i and j.
λ_i		Weight of cost component (CCi).

By observing that in the model we have no benefit associated with the concept of patients requiring two visits, a simplified model can be derived by duplicating the nodes of the set C_2, those associated with patients requiring two services. Formally, we define $C = C' \cup C_3$ where C_3 has the same cardinality of C_2 and for each node $i \in C_2$ there is exactly one node $j \in C_3$, representing both services required by the same patient. Notice that, when $i \in C'_{opt}$, then the new node j is inserted in $C_{opt} \subseteq C$ (C_{opt} is the update of C'_{opt} with the fictitious second services). A precedence $c_i \prec c_j$ is also imposed between the two nodes to connect them. Notice that if the two services associated with c_i and c_j have to be executed simultaneously, the precedence is fictitious and only indicates the relation between the nodes. As a consequence of this transformation, the processing time p_i can now be directly associated with each node $i \in C$. The sets defining the problem are redefined consequently as $C_0 = C \cup \{0\}$, $C^v = \{i \in C : v \in S_i\}$ and $C_0^v = C^v \cup \{0\}$, where the definition of each S_i and V_i are updated to the new set of nodes C.

The problem can be formulated as a Vehicle Routing Problem with Soft Time Windows featuring a complex objective function and additional constraints regarding: *(i)* the assignment of service providers to each node, *(ii)* strict temporal constraints about visit times for node pairs linked to the same customer (i.e., patient), *(iii)* management of idle and waiting periods, overtime, and tardiness considerations. The weighting coefficients λ_i that determine the relative importance of different cost components are not fixed parameters but rather depend on the specific problem instances, providing the necessary flexibility to adapt the cost function to different scenarios and requirements.

The variables of the new model are defined as follows (see also Table 14.2):

- x_{ij}^v is a binary variable equal to 1 if caregiver $v \in V$ visits patient $j \in C^v$ immediately after patient $i \in C^v$, and 0 otherwise; in particular, $x_{ii}^v = 1$ denotes that patient i is not visited by caregiver v;

- t_i denotes the starting time of the visit at patient $i \in C$ (possibly greater than the

Table 14.4: Variables of the Mixed Integer Linear Programming (MILP) model.

Symbol	Domain	Description
x_{ij}^{v}	$\{0,1\}$	1, if caregiver v visits patient j after patient i; 0, otherwise.
t_i	$[0,\infty]$	Start time of the visit at patient i.
r_{ik}	$\{0,1\}$	1, if patient i is visited during their time window k.
z_i	$[0,\infty]$	Tardiness in the execution of the service for patient i.
$D_{\max}$	$[0,\infty]$	Maximum tardiness.
v_i	$\{0,1\}$	1, if optional patient i is skipped; 0, otherwise.
w_i	$[0,\infty]$	Waiting time spent at patient i.
a_v	$[0,\infty]$	Active working time of caregiver v.
s_v	$[0,\infty]$	Deviation of the working time of caregiver v w.r.t. the average.
y_v	$[0,\infty]$	Extra time worked by caregiver v.
b_v	$[0,\infty]$	Transportation time of caregiver v from the patients before and after the lunch break.
$H_{\max}$	$[0,\infty]$	Maximum idle time.

arrival time at the location of patient i);

- v_i is a binary variable taking value 1 if the optional patient $i \in C_{opt}$ is skipped, 0 otherwise;

- r_{ik} is a binary variable that takes value 1 if patient $i \in C$ is visited during time window $k \in \{1, \ldots, |T_i|\}$;

- a_v denotes the active working time of caregiver $v \in V$, calculated as service time plus transportation time and lunch time, if applicable;

- s_v denotes the absolute deviation of the working time of caregiver $v \in V$ with respect to the average working time of all caregivers;

- z_i denotes the tardiness (delay) in the execution of treatment at patient $i \in C$ with respect to the end of the time window l_i of the patient;

- y_v denotes extra time worked by caregiver $v \in V$ beyond β_v, the theoretical end of their shift;

- b_v denotes the transportation time spent by a caregiver $v \in V_{lb}$ to move from the patient immediately before the lunch break to that immediately after. The variable takes the value 0 if the lunch break is skipped;

- w_i denotes the waiting time at patient i, when i is not the first patient visited by a caregiver;

- $D_{\max}$ quantifies the maximum tardiness experienced when visiting patients;

- $H_{\max}$ quantifies the maximum idle time experienced by caregivers.

The mathematical model is presented below. For clarity, we have organized the constraints into separate sections, each containing the formal presentation and explanatory comments.

Objective Function

$$\begin{aligned}\min \lambda_1 \left(\sum_{i \in C_0'} \sum_{j \in C_0, j \neq i} d_{ij} \sum_{v \in V_i \cap V_j} x_{ij}^v + \sum_{v \in V_{lb}} b_v \right) + \lambda_2 \sum_{i \in C} z_i + \\ + \lambda_3 D_{\max} + \lambda_4 \sum_{v \in V} y_v + \lambda_5 \sum_{i \in C} w_i + \lambda_6 H_{\max} + \lambda_7 \sum_{v \in V} s_v + \\ + \lambda_8 \sum_{i \in C} \sum_{v \in V, v \notin P_i} (1 - x_{ii}^v) + \lambda_9 \sum_{v \in V_{lb}} x^v_{c_v^{lb} c_v^{lb}} + \lambda_{10} \sum_{i \in C_{opt}} v_i \end{aligned} \tag{14.1}$$

The objective function (14.1) optimizes the problem by combining the ten targets defined in Section 14.2.3, following the same order as in the presentation. The weights λ_i (with $i = 1, \ldots, 10$) are instance-dependent and written in the input file. The optimization model is subject to the constraints described in the following sections.

Routing Constraints

$$\sum_{j \in C_0^v} x_{ji}^v = 1 \qquad v \in V, i \in C_0^v \tag{14.2}$$

$$\sum_{v \in V : i \in C^v} (1 - x_{ii}^v) \underbrace{+ v_i}_{\text{if } i \in C_{opt}} = 1 \qquad i \in C \tag{14.3}$$

$$\sum_{j \in C_0^v, j \neq i} x_{ji}^v = \sum_{j \in C_0^v, j \neq i} x_{ij}^v \qquad v \in V, i \in C_0^v \tag{14.4}$$

$$x_{00}^v = 1 \Rightarrow x_{ii}^v = 1 \qquad v \in V, i \in C_0^v \tag{14.5}$$

Equations (14.2) state that for each caregiver v and each location i, exactly one x variable must be active. Constraints (14.3) state that exactly one caregiver must visit each compulsory patient, while optional patients can be skipped. Constraints (14.4) are flow-conservation constraints to impose, together with the timing constraints defined later, feasibility for the tours of the caregivers. Constraints (14.5) state that if a caregiver does not operate (x_{00}^v), then no patient can be assigned to their route.

Time-related Constraints

$$r_{ik} = 1 \Rightarrow z_i \geq t_i - l_i^k \qquad i \in C, k \in \{1, \ldots, |T_i|\} \tag{14.6}$$

$$x_{i0}^v = 1 \Rightarrow y_v \geq t_i + p_i + d_{i\sigma_v} - \beta_v \qquad v \in V, i \in C_0^v \tag{14.7}$$

$$x_{0j}^v = 1 \Rightarrow t_j \geq \alpha_v + d_{\sigma_v j} + w_j \underbrace{+ b_v}_{\text{if } v \in V_{lb} \wedge j = c_v^{lb}} \qquad v \in V, j \in C^v \tag{14.8}$$

$$x_{ij}^v = 1 \Rightarrow t_j = t_i + p_i + d_{ij} + w_j \underbrace{+ b_v}_{\text{if } v \in V_{lb} \wedge j = c_v^{lb}} \qquad v \in V, i, j \in C^v, i \neq j \tag{14.9}$$

$$\delta_i^{\min} \leq t_j - t_i \leq \delta_i^{\max} \qquad i, j \in C : i \prec j \tag{14.10}$$

Inequalities (14.6) define the tardiness at each patient i as the eventual excess over l_i^k of

the starting time of service at patient i during time window k. Constraints (14.7) define the eventual overtime of each caregiver v as the arrival time back at the ending location, described as the starting time at the last patient i plus the operating time at i plus the travel time from i to σ_v, minus the theoretical end of the shift β_v. Notice that the constraint is activated only if $x^v_{i0} = 1$. Such a conditional expression can be easily implemented in modern Mixed Integer Linear Programming (MILP) solvers through indicator constraints ([236, 387]). This is also true for similar constraints in the remainder of the model. Equations (14.8) take care of timing for the first patient of each caregiver. In particular, the starting time at the first patient j of caregiver v is given by the start time α_v of the shift of v plus the eventual waiting time at the base w_v plus the travel time from the base to the first patient j. We assume that a caregiver waits at their starting location σ_v rather than traveling to the first patient and waiting there. As a result, any waiting time at the first patient is disregarded. The corresponding constraint is enforced only when $x^v_{0j} = 1$. Constraints (14.9) take care of timing in case of the move of a caregiver from one patient to another, or back to the base. The starting time at the patient j of caregiver v is given by the start time t_i at patient i plus the service time at patient i plus the travel time from patient i to patient j plus the eventual waiting time w_j at patient j. The constraint is activated only if $x^v_{ij} = 1$. Constraints (14.10) impose the hard time-windows associated with double-visits to the same patient (here split into virtual patients i and j).

Double-Service and Lunch-Break Constraints

$$x^v_{ii} + x^v_{jj} \geq 1 \qquad v \in V, i, j \in C^v_0 : i \prec j \qquad (14.11)$$

$$\Big((x^v_{ic^{lb}_v} = 1) \wedge (x^v_{c^{lb}_v j} = 1)\Big) \Rightarrow b_v = d_{ij} \qquad v \in V_{lb}, i, j \in C^v_0 \qquad (14.12)$$

Inequalities (14.11) force each caregiver to do at most one service[1] at the same patient in need of a double visit (here split into virtual patients i and j). Constraints (14.12) set the value of variable b_v, expressly inserted to collect the travel time around the possible lunch-break according to the patients visited right before and right it. Namely, if the artificial patient c^{lb}_v is visited in between patients i and j by the caregiver v, then the variable b_v, expressly inserted to collect the travel time between the lunch-time customers, is assigned to d_{ij}. The value of b_v will be used when calculating the active working time of caregiver v in constraints (14.13).

Working Time Constraints

$$a_v = \sum_{i \in C^v} \left(p_i(1 - x^v_{ii}) + d_{\sigma_v i} x^v_{0i} + d_{i\sigma_v} x^v_{i0} + \sum_{j \in C^v, j \neq i} d_{ij} x^v_{ij} \underbrace{+b_v}_{\text{if } v \in V_{lb}} \right) \qquad v \in V \qquad (14.13)$$

[1] We recall that x^v_{ii} is 1 when patient i is *not visited* by caregiver v.

$$s_v \geq a_v - \frac{\sum_{k \in V} a_k}{|V|} \qquad v \in V \tag{14.14}$$

$$s_v \geq \frac{\sum_{k \in V} a_k}{|V|} - a_v \qquad v \in V \tag{14.15}$$

Constraints (14.13) define the total active working time of caregiver $v \in V$, calculated as the sum of service time at patients, the traveling times from the starting point and to the ending point, the travel times among customers, and the lunch break in case it is taken. Inequalities (14.14) and (14.15) calculate, for each caregiver, the absolute deviation of the associated working time with respect to the average one. Specifically, given a caregiver v, inequality (14.14) captures the excess of working time with respect to the average value among all the caregivers, while inequality (14.15) captures its deficiency.

Time Window Constraints

$$\sum_{k \in \{1,\dots,|T_i|\}} r_{ik} \underbrace{+v_i}_{\text{if } i \in C_{opt}} = 1 \qquad i \in C \tag{14.16}$$

$$r_{ik} = r_{jk} \qquad i,j \in C : i \prec j, k \in \{1,\dots,|T_i|\} \tag{14.17}$$

$$r_{ik} = 1 \Rightarrow t_i \geq e_i^k \qquad i \in C, k \in \{1,\dots,|T_i|\} \tag{14.18}$$

$$r_{ik} = 1 \Rightarrow t_i \leq l_i^k \qquad i \in C, k \in \{1,\dots,|T_i|\} \tag{14.19}$$

Equalities (14.16) impose that for each mandatory patient, exactly one time window has to be selected, or none for optional patients. Equalities (14.17) force the same time window for all the services requested by the same patient. Inequalities (14.18) and (14.19) constrain the visit time at each patient based on the time window selected to service them.

Maximum Value Definition Constraints

$$D_{\max} \geq z_i \qquad i \in C \tag{14.20}$$

$$H_{\max} \geq \beta_v + y_v - \alpha_v - \sum_{i \in C^v} d_{\sigma_v i} x_{0i}^v + (p_i + d_{i\sigma_v}) x_{i0}^v + \sum_{i \in C^v} \sum_{j \in C^v, j \neq i} (p_i + d_{ij}) x_{ij}^v - b_v \qquad v \in V \tag{14.21}$$

Constraints (14.20) assign the maximum tardiness at patients, $D_{\max}$. Analogously, constraints (14.21) are used to assign the maximum idle time among caregivers, $H_{\max}$. The idle time for each caregiver $v \in V$ is defined as the length of the working shift of v, eventually extended by overtime y_v, minus the time spent traveling or operating by the caregiver.

14.2.3 A Note on Preferences

Given the importance of patient-centered care and staff well-being, our model incorporates features that bring the formulation closer to real-world home healthcare operations. Unlike Mankowska, Meisel, and Bierwirth [327], which neglects patient–caregiver preferences, our model considers them as follows:

- Incompatibilities: for each patient i, the set of compatible caregivers V_i (Table 14.2) allows to define hard constraints that exclude infeasible assignments.
- Preferences: for each patient i, the set of preferred caregivers P_i (Table 14.2) allows to define soft constraints whose violations are penalized in the objective function via cost component (CC8).

Consequently, preferences are represented on a three-level scale: *(i)* incompatibilities (hard exclusions), *(ii)* preferred assignments (soft constraints; violations penalized), and *(iii)* neutral assignments (neither preferred nor incompatible, incurring no reward or penalty) This approach is more practical than fine-grained scoring schemes (e.g., Ait Haddadene, Labadie, and Prodhon [14]).

The model also incorporates staff preferences, which were absent in prior works:

- Home-Based Routes: caregivers may start and end their routes at home rather than at the central office, represented by σ_v in Table 14.3 and in the extended set of locations by the signpost $\{0\}$.
- Working Hours: shift types specify preferred working hours, represented by α_v and β_v in Table 14.3 and accounted for in Constraints (14.8) and in cost component (CC4).

14.3 Representation of Problem Characteristics

We first present a unified representation for both HHCRSP instances and solutions (Sections 14.3.1 and 14.3.2), followed by a set of instance features designed to capture structural differences and analyze their distribution in the instance space (Section 14.3.3). We then describe the adaptation of benchmark instances from the literature to ensure compatibility with this representation (Section 14.3.4), and conclude with a newly generated dataset, created through a dedicated instance generator, to explore specific aspects of the problem (Section 14.3.5).

14.3.1 Instance Representation

Problem instances are essential for combinatorial optimization, modeling real-world scenarios, and evaluating solution methods [397]. For the HHCRSP, the accessibility to well-defined and diverse instances faces several challenges: varying problem formulations with different objectives and constraints, lack of a standardized format for instances and solutions, and limited availability to the research community due to privacy concerns.

Despite growing research interest, existing HHCRSP datasets remain fragmented with heterogeneous formats, creating two significant limitations: difficulty accommodating

evolving healthcare requirements and regulatory changes, and complications in implementing and comparing solution approaches across studies. Text-based formats also introduce fragility issues due to a lack of error-checking mechanisms.

We propose a unified `json`-based representation for HHCRSP to address these challenges. This format is widely used, human-readable, and accommodates new problem-specific features as formulations evolve. `json` files are easily parsed by most programming languages, ensuring compatibility with different solver implementations. While we provide a dedicated semantic validator (Section 14.4), users can independently verify syntactic compliance against a `json` Schema [390] specification.

The core structure of an HHCRSP instance is shown in Listing 14.1. This format captures the main components of the problem: a list of patients (`patients`), a list of caregivers (`caregivers`), and a list of services (`services`) that caregivers can provide and patients may require. Additionally, the instance includes information about the departure and arrival points for caregiver routes (`terminal_points`).

The instance also provides a distance matrix (`distances`) reporting the travel times in an array of arrays to enhance usability. Although this information is redundant, since patient and terminal point locations are specified using latitude and longitude, it ensures that all solvers operate with the same distances.

Finally, the `metadata` field contains supplementary information that, while not directly part of the instance, is essential for ensuring comparability, facilitating benchmarking, and preserving relevant details. For example, it records the weights of the cost components, either as numerical values for soft constraints or as the string `HARD` otherwise, and includes details about the instance generator when the instance is synthetically produced.

Listing 14.1: General instance structure.

```
{
  "patients": [...],
  "caregivers": [...],
  "services": [...],
  "terminal_points": [...],
  "distances": [...],
  "metadata": {...},
}
```

Each patient in the `patients` list is represented by a unique identifier and a set of attributes. The example in Listing 14.2 illustrates these attributes, including the list of required services (`required_services`), where each item specifies the service and its estimated duration. This duration this may differ from the default duration for that service type (Listing 14.5). Additionally, the patient's preferred time windows for receiving care are recorded under the `time_windows` field, where time is expressed in units (e.g., minutes) relative to the beginning of the planning horizon. The instance also specifies the patient's preferred and incompatible caregivers through the `preferred_caregivers` and `incompatible_caregivers` fields. The `optional` attribute indicates whether the patient is mandatory or not. Finally, the geographical coordinates are provided along with the `distance_matrix_index`, which maps the location to an entry in the distance matrix (`distances`).

Listing 14.2: Patient specification.

```
{
  "id": "p2",
  "required_services": [ { "service": "s3", "duration": 30 } ],
  "time_windows": [ { "start": 60, "end": 300 } ],
  "location": [ 13.3011, 38.1034 ],
  "distance_matrix_index": 4,
  "incompatible_caregivers": [ "c2" ],
  "preferred_caregivers": [ "c1", "c4" ],
  "optional": false
}
```

Each caregiver in the `caregivers` list is represented by a unique identifier and a set of attributes. The example in Listing 14.3 illustrates these attributes. This identifier may also appear in the `preferred_caregivers` and `incompatible_caregivers` fields of the patients. The `abilities` field specifies the services the caregiver is qualified to perform. The departure and arrival points are identified by their respective indices, as defined in the corresponding fields in the related section (`terminal_points`, Listing 14.4 reports the structure). The `working_shift` field specifies the caregiver's working hours.

Listing 14.3: Caregiver specification.

```
{
  "id": "c1",
  "abilities": [ "s3" ],
  "departing_point": "d1", "arrival_point": "d1",
  "working_shift": { "start": 0, "end": 600 }
}
```

Listing 14.4: Terminal point (i.e., departing/arrival point) specification.

```
{
   "id": "d0",
   "distance_matrix_index": 0,
   "location": [ 13.3468, 38.1109 ]
}
```

Each service in the `services` list is represented by a unique identifier, its type, and a default duration (which may differ from the duration required by a specific patient's needs). Listing 14.5 reports an example. The distinction between the identifier and its type allows for grouping related services under broader categories. For example, under the type "medical care", services such as physiotherapy or wound care can be included, while the type "daily activity" may include tasks like bathing or house cleaning. The identifier is referenced in the `required_services` list of patients and the `abilities` field of caregivers.

Listing 14.5: Service specification.

```
{
    "id": "s0",
    "type": "t0",
    "default_duration": 45
}
```

The generic instance can be customized by adding additional fields, such as specifications for lunch breaks (Listing 14.6). Similarly, the patient, caregiver, and service components can be adapted to include or exclude specific features depending on the problem formulation. For example, if the formulation accounts for lunch breaks, each caregiver will include a flag indicating whether they are entitled to such a break (`lunch_break`); conversely, if the formulation does not consider optional patients, the corresponding field can be omitted. This flexible structure allows the instance representation to accommodate many problem variants without altering the core format. By selectively including or excluding fields, the same schema can represent basic and extended formulations.

Listing 14.6: Lunch specification.

```
"lunch_breaks": { "start": 180, "end": 360, "min_duration": 30 }
```

14.3.2 Solution Representation

A solution to an HHCRSP instance consists of the following decisions: *(i)* The route each caregiver follows (i.e., the order in which patients are visited). *(ii)* The arrival time, departure time, and service provided by each caregiver at each patient in their route. *(iii)* If lunch breaks are considered, each caregiver's lunch break location, and start and end time.

A common solution representation ensures consistent evaluation, making it easier to compare different approaches. It also improves interpretability for practitioners, aiding in debugging, model implementation, and facilitating reproducibility, as solutions can be stored, analyzed, and shared in a standardized manner. Therefore, we encode the solution by employing the `json` format as in the case of the instance (Section 14.3.1). The core structure of an HHCRSP solution is shown in Listing 14.7. This format captures the routes (`routes`), along with optional fields such as cost components and overall cost (both in terms of total cost and the number of violations). While these additional fields are not part of the solution itself, they facilitate comparability, benchmarking, and debugging.

Listing 14.7: General solution structure.

```
{
   "routes": [...],
   "cost_components": {...},
   "cost": { "cost": 2706, "violations": 0 }
}
```

Each item in the `routes` list represents a caregiver and includes their identifier (`caregiver_id`) along with the list of locations they visit (`locations`). Each location entry specifies the service performed, the corresponding place (i.e., the patient), and the temporal details of the visit (i.e., the start and end time of the service are required). If a lunch break is scheduled, it is recorded as a service labeled `lunch-break`.

Listing 14.8: Route specification.

```
{
  "caregiver_id": "c2",
```

```
  "locations": [
    ...
    { "arrival_time": 430,  "start_time": 430, "departure_time": 460, "patient
        ": "p7", "service": "s1" },
    ...
  ]
}
```

While the representation above enables easy benchmarking and explicitly reports all solution characteristics, inspecting solutions visually may also be beneficial. A solution to the HHCRSP can be effectively visualized as a Gantt chart, as put forward by Ceschia et al. [80]. In this representation, reported in Figure 14.1 each item on the y-axis corresponds to either a patient or a terminal point, while the x-axis represents the scheduling horizon. The activities of each caregiver are distinguished by different colors, including the lunch break (abbreviated as LB). Additionally, patient availability times are shown in gray to provide a clear visual reference.

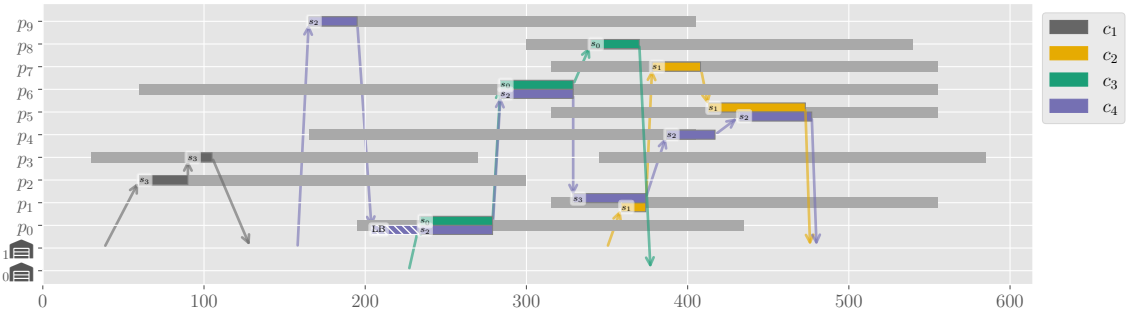


Figure 14.1: Example of visualization for one solution to the Home Healthcare Routing and Scheduling Problem (HHCRSP).

14.3.3 Instance Features

Algorithms should be tested on diverse benchmark instances to reveal their strengths and weaknesses [436]. The key rationale is that similar instances are unlikely to provide new insights or meaningful additional knowledge.

To analyze HHCRSP instances, we define a set of 272 features, either directly available from the instance data or requiring only minimal computation. Examples include the number of patients and caregivers, the proportion of patients requiring multiple services, caregiver shift durations, and the degree of overlap between caregiver shifts and patient availability. Some features are single-valued, while others require aggregation to capture their statistical distribution, such as the minimum ($\min$), maximum ($\max$), or range (range $= \max - \min$). For instance, patient time-window lengths are aggregated across patients.[2] The complete list of features is provided in Appendix F.

In this work, these features are used to evaluate the capabilities of our generator (Section 14.3.5). In future research, they could support Instance Space Analysis (ISA) [436], algorithm selection, and other analytical tasks.

[2]The feature count (272) also includes the aggregated measures.

14.3.4 Existing Instances Conversion

Among the limited publicly available datasets, we selected and translated into our unified format those provided by Bazirha, Benmansour, and Kadrani [40], Bazirha, Kadrani, and Benmansour [41], Ceschia et al. [80], Hiermann et al. [220], Kummer [271], and Mankowska, Meisel, and Bierwirth [327]. These datasets were chosen because they either align closely with the formulation by Mankowska, Meisel, and Bierwirth [327], introduce additional real-world extensions to previous formulations, or include many instances. We exclude from our conversion the single instance provided by Varas et al. [474], due to its limited scope, and the dataset by Liu, Yuan, and Jiang [307], as it addresses a stochastic formulation of the problem, which falls outside the scope of our analysis. Note that the datasets by Bazirha, Kadrani, and Benmansour [41], Ceschia et al. [80], Kummer [271], and Mankowska, Meisel, and Bierwirth [327] refer to the same formulation. Whereas Bazirha, Benmansour, and Kadrani [40] report also the possibility for patients to express more than one availability time window, and Hiermann et al. [220] consider optional patients, incompatibilities between patients and caregivers, and multimodal transportation modes. However, the publicly available dataset is limited: it contains only 4 randomly generated instances (with realistic travel times based on the city of Vienna, Austria). Although their original study examined real-world instances, these remain unavailable due to confidentiality constraints.

Table 14.5 presents publicly available datasets for the single-period HHCRSP. Each column corresponds to a dataset, while each row represents a specific characteristic. Additionally, we report the number of instances in each dataset, whether they incorporate real-time distances and population density (indicated with a ✓), the file format, and their availability. For Kummer [271] and *Us* we report both the number of validation instances (160 and 24, respectively), and their total number (15040 and 465, respectively).

Table 14.5: Publicly available datasets for single period HHCRSP.

Characteristic	Mankowska et al. [327]	Hiermann et al. [220]	Liu et al. [307]	Kummer [271]	Bazirha et al. [41]	Bazirha et al. [40]	Varas et al. [474]	Ceschia et al. [80]	Us
Instances [#]	70	4	30	160/15040	42	40	1	30	24 / 465
Patients [#]	10–300	419–424	30, 40, 50	10–400	10–52	10–200	141	44–378	10–700
Caregivers [#]	3–40	493–518	5, 7, 9	3–40	3–10	3–40	17	7–51	2–124
Services [#]	6	5	30, 40, 50	6	6	6	155	2–4	4–18
Double-service patients [%]	30	55–77 *		30	0–32	0–60**	11	3–48	0–100
Time windows [min]	120	25–385	60–120	120	120	120	120–300	60–180	20–480
Variable service durations		✓	✓	✓	✓	✓	✓	✓	✓
Real time distances		✓		✓			✓	✓	✓
Real population							✓	✓	✓
File format	`txt`	`dm, conf`	`c++`	`txt`	`xlsx`	`xlsx`	`csv`	`json`	`json`
Availability	[a]	[b]	[c]	[d]	[e]	[f]	[g]	[h]	[i]

*Patients with 3 services are present (0–2%)
**Patients with 3 services are present (0–9%)
[a] https://prodlog.wiwi.uni-halle.de/forschung/research_data/hhcrsp/. Accessed 2025–10–01.
[b] https://www.ac.tuwien.ac.at/research/problem-instances/. Accessed 2025–10–01.
[c] https://drive.google.com/file/d/1ppfHc0AZfb2HpJmvCEsxRpwEiLNE4f5b/view. Accessed 2025–10–01.
[d] https://github.com/afkummer/hhcrsp-dataset-2021. Accessed 2025–10–01.
[e] On request. Translated at [i]
[f] On request. Translated at [i]
[g] On appendix of the article and on request
[h] https://github.com/iolab-uniud/hhcrsp. Accessed 2025–10–01.
[i] see Appendix A

Converting Mankowska, Meisel, and Bierwirth [327] and Kummer [271] instances proved easy due to their simple textual data format. This format uses basic placeholders (e.g., `nbNodes` for number of nodes) followed by either a single scalar value, space-separated values for vectors, or line-separated sequences for matrices.

Translating Ceschia et al. [80] instances proved straightforward since these were generated

using an earlier version of our generator

As for Bazirha, Kadrani, and Benmansour [41] instances, translation was straightforward regarding problem formulations; however, the main challenge lay in the format itself. Their dataset comprises six groups (A through F) of seven instances each, with each group provided in a separate Excel file. Each instance occupied a distinct tab within its file (seven tabs per file), with properties arranged in layered tables. The Bazirha, Benmansour, and Kadrani [40] dataset presented similar format challenges. It contains four groups with 9 instances each, plus 4 additional very large instances.

Translating Hiermann et al. [220] instances required several assumptions to align with our specifications. Each instance was split across three files: car time distance matrix, public transportation time distance matrix, and entity details (caregivers, patients, services). We standardized timing data by multiplying values by 5, as specified in the original paper's compendium. The four randomly generated instances had no mandatory services, so all were treated as optional. For patients requiring multiple independent services, we represented each service as a separate patient. We explicitly defined incompatibilities through strict relations in patients' descriptions, while marking compatible caregivers as preferred. This approach allows patients to be served by caregivers without all the required qualifications, with appropriate cost penalties. We excluded this dataset from our analysis since our solution methods currently support only one distance matrix (with multiple transportation modes planned for future work).

14.3.5 Instance Generator and New Dataset

While in the literature many formulations exist that include a limited set of real-world aspects each, this is not the case for the instances: there exist only a limited number of datasets that are available, which are based on real-world features (like time distances), and that employ some of the realistic aspects reported in our formulation. For this reason, we need to have an instance generator that *(i)* is modeled with real-world features; *(ii)* can account for all the realistic aspects; and *(iii)* is highly flexible and customizable.

The generator takes several inputs to define the characteristics of an instance. These inputs include *geographic information*, such as the target city and the radius within which patients are located; *service-related parameters*, including the type and the total number of services per each type; *patient characteristics*, such as the total number of patients, the proportion of single- and double-service patients (distinguishing between independent, synchronous, and sequential services), the share of patients with incompatible or preferred caregivers, optional patients, and those requiring multiple time windows; *caregiver parameters*, including shift durations, lunch break specifications, and the configuration of departure and arrival points (e.g., whether they are fixed, flexible, or different for start and end locations); *temporal settings*, such as the temporal horizon, the interpretation of time windows (whether they should be considered as strict or not), time window durations, service durations (with bias options for selection), the temporal gap between consecutive services, and the granularity of time windows. Additionally, the generator allows customization of *cost components*, enabling users to define their mix and weights. Only two assumptions are made: *(i)* A patient can require at most two services. *(ii)* A patient can express at most two preferred time-windows. Nevertheless, the generator allows for a high degree of customization,

enabling users to model different scenarios. In fact, certain aspects can be omitted if not required, e.g., lunch break specifications or setting the proportion of single-service patients to zero. This flexibility ensures that instances can be tailored to different problem formulations, regulatory constraints, and practical considerations.

The generator operates in two sequential phases. First, it generates geographic information (patient and departing/arrival locations along with travel times), followed by other entities such as temporal distribution and patient characteristics. The generator leverages real-world city networks and population patterns for spatial distribution modeling. It employs GEOSTAT Census 2021 datasets[3] to represent population data in specific geographical areas (at 1 km^2 resolution), while Open Source Routing Machine (OSRM) [322] calculates travel times using actual road networks and a car as a means of transportation.[4] Geographic data can be downloaded from these reference sites when required, optimizing resource usage. All other aspects are randomly generated according to user-specified parameters provided through command line inputs and configuration files.

The enhanced generator builds upon the one proposed by Ceschia et al. [80], with both versions integrating actual road travel times and population distribution data to create more realistic models. However, our generator enhances flexibility by exposing all adjustable parameters to the user, allowing for greater customization. Additionally, it provides a comprehensive representation of real-world aspects that were neglected before.

We generate 465 new instances, whose characteristics are reported in Table 14.5, alongside those of the existing datasets. For the experiments reported in Chapter 15, we randomly select 24 instances.

The new dataset is designed to include all the realistic characteristics described in this chapter. The weights of the cost components are assigned based on a predefined range (note that a user can generate instances with their weight setting and mix). We utilize a Hammersley point set [210] to systematically sample weight values from the following ranges. Travel time, total tardiness, total waiting time, and total extra time weights range from 1 to 10, with a step size of 1. The highest tardiness weight varies between 0 and 5, with a step of 1. The highest idle time weight is assigned within the broader range of 0 to 100, while caregiver preferences range from 0 to 20, with a step of 1. The weight for optional patients is selected from 100 to 200 in increments of 10, whereas the penalty for missed lunch breaks follows a range of 50 to 100, also increasing in steps of 10. This approach ensures a diverse set of weight configurations, balancing different aspects of the problem and stressing the algorithms differently.

With these new instances, we aim to establish the following key points:

1. Our generator can produce instances comparable to those in existing literature, particularly those exhibiting realistic characteristics (e.g., realistic travel times).

2. It can produce instances that address gaps in current datasets.

3. It supports additional features described in problem formulations for which no publicly accessible instances are currently available.

[3] https://ec.europa.eu/eurostat/web/gisco/geodata/grids. Accessed 2025–04–08.

[4] https://project-osrm.org. Accessed 2025–04–08.

While point 3 is inherently achieved through the generator's configurability (e.g., enabling lunch breaks, optional patients, etc.), the first two require a more detailed investigation. To address these, we adopt the following methodology: We group the existing datasets based on the problem formulations they reflect and perform a PCA on each group. We then project our newly generated instances onto the resulting PCA space to visually and quantitatively evaluate how they align with and differ from the existing datasets.

A PCA [197] is a dimensionality reduction approach that converts high-dimensional datasets into lower-dimensional representations while maximizing the retention of variance in the original data [95]. It identifies the principal components, i.e., orthogonal directions along which the data varies the most, allowing for a more interpretable visualization and analysis of datasets. To perform the analysis, we use the `pca` v.2.0.9 `Python` package.[5]

The examination of existing datasets w.r.t the underlying formulations is essential, as certain features are exclusive to particular models (e.g., the instances by Ceschia et al. [80] lack optional patients. In contrast, those by Hiermann et al. [220] include them). Thus, these features are assigned a default value of 0 in datasets that do not incorporate them.

Figure 14.2 displays two PCA analyses: the left plot shows instances from Kummer [271] and Ceschia et al. [80], whereas the right plot presents those from Hiermann et al. [220]. Literature instances are shown as dark scatter points, while our generated instances are visualized through density distributions across the space.

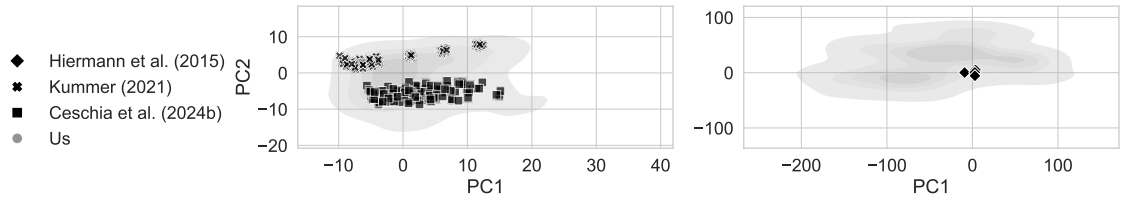


Figure 14.2: Principle Components Analysis (PCA) of existing datasets by problem formulation.

In the datasets by Kummer [271] and Ceschia et al. [80], four principal components account for 80% of the variance. These components are mainly linked to the size of the patients' time windows and include the total length of all patient time windows (i.e., the sum of the duration of all the time windows), the minimum time window duration, the average number of services required per patient, and the median patient availability (calculated as the total duration of all time windows assigned to a patient). In contrast, the variability in the instances from Hiermann et al. [220] is primarily explained by two principal components, mainly composed of the following features: the total number of time windows and the median duration of caregiver shifts.

In all cases, our instances span the feature space of the existing datasets. It is important to note that this approach projects our instances onto a simpler feature space, not only because of the dimensionality reduction, but also because the literature instances do not account for some aspects. Consequently, we exclude from the PCA some characteristics present in our generated instances that are absent in the existing datasets.

[5] `https://erdogant.github.io/pca`. Accessed 2025–04–08.

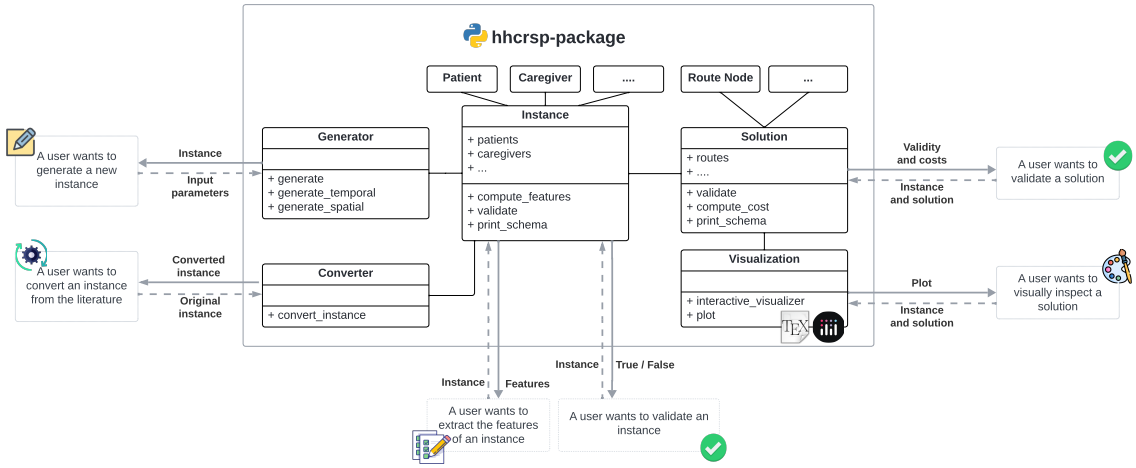


Figure 14.3: General framework of the Home Healthcare Routing and Scheduling Problem (HHCRSP) toolbox.

14.4 Toolbox

We developed a `Python` package for handling HHCRSP instances and solutions, providing a comprehensive tool for instance generation, validation, and assessment.

The package includes the generator for creating problem instances, a validator for verifying both instances and solutions in terms of format and validity (e.g., assessment of solutions in relation to the problem formulation at hand), and additional utilities to facilitate analysis and benchmarking, such as the calculation of the cost components, the extraction of the features, the visualization and inspection of solution, and the conversion of existing literature instances. Each element in the instance and solution is defined as a separate class, e.g., a class defines the patient, a class defines the caregiver, etc. All the components and methods can also be accessed through Command Line Interface (CLI), ensuring ease of use for different workflows. Figure 14.3 reports the general scheme of the toolbox and the different tasks a user can perform.

Chapter 15

Solution Methods

Given the multitude of formulations proposed for the Home Healthcare Routing and Scheduling Problem (HHCRSP), a variety of solution methods have been developed, each tailored to the specific assumptions, constraints, and data structures of its underlying model. These methods typically exploit problem-specific simplifications or topological characteristics, which limits their applicability to other formulations or practical contexts. However, in real-world operations—where problem features evolve and data vary across care providers—more general and adaptable solution methods are required.

In this chapter, we adapt, generalize, and extend two state-of-the-art approaches — a Simulated Annealing (SA) metaheuristic [80] and a Mixed Integer Linear Programming (MILP) solver [354] — to the unified HHCRSP formulation proposed in Chapter 14. Both methods were originally developed for the narrower variant introduced by Mankowska, Meisel, and Bierwirth [327]. We extend them to accommodate the additional real-world features captured by our unified framework, such as multiple caregiver departure points, shift constraints and overtime, optional visits, and caregiver–patient incompatibilities. We then conduct an extensive empirical evaluation of these enhanced methods. The assessment covers both translated instances from existing benchmarks and newly generated instances representing the extended formulation. Through this evaluation, we analyze the performance and scalability of each approach, explore their sensitivity to model components, and derive managerial insights from ablation studies on the cost-function composition. The contributions of this chapter are threefold:

- **Generalized solution methods:** extensions of the SA and MILP approaches to the unified HHCRSP formulation.
- **Comprehensive experimental evaluation:** performance assessment on both existing and newly generated instance sets.
- **Managerial and methodological insights:** analysis of cost-function components and their influence on solution quality and computational effort.

This chapter is structured as follows. Section 15.1 outlines the solution methods employed to solve the unified version of the HHCRSP. Section 15.2 details the experimental setup and Section 15.3 reports the results. Finally, Section 15.4 draws some conclusions.

15.1 Methodology

To solved the HHCRSP, we use MILP (Section 15.1.1) and SA (Section 15.1.2). Both of them were originally proposed in the literature [80, 354]; nevertheless, adapting them to the full feature set required *(i)* a revised model, *(ii)* novel neighborhood relations, and *(iii)* a redesigned earliest-time computation.

15.1.1 Mixed Integer Linear Programming

To solve the problem using the model introduced in Section 14.2, we formulated the model in the syntax required by a constraint satisfaction solver and processed it using a black-box solver engine. Specifically, we used `OR-Tools CP-SAT`.

15.1.2 Simulated Annealing

In this section, we outline the problem-dependant characteristics of our SA, namely the solution representation together with the initial solution and the neighborhood relations.

Solution Representation

The search space is defined by two components:

- Π, a global permutation of patients (i.e., a sequence without repetitions), and
- Θ, the assignment of each patient to one or two caregivers.

Assignments are restricted to *feasible* caregivers, meaning those possessing the required skills for the corresponding service. In the extended formulation, as for caregivers without the necessary qualification, we exclude a-priori incompatible caregivers from the pool of possible assignments. All other additional features contribute to the cost function that guides the local search toward lower-cost solutions. An important consideration concerns optional patients and their potential exclusion from scheduling. While preserving the global permutation structure, unscheduled optional patients are placed at the end of the sequence, thereby excluding them from route construction.

From Π and Θ, the caregivers' routes and corresponding service times are deterministically constructed. Service start times are computed as early as possible, taking into account the beginning of caregivers' shifts, patients' availability windows, synchronization, and, when relevant, scheduled lunch breaks.

Technically, Π and Θ are sufficient to represent the search space. Nevertheless, they are complemented by a set of redundant data structures introduced to speed up computations. Among these, we include the inverse of Π (i.e., the position of each patient in the ordering) and the route of each caregiver. Note that we do not simply use the route of each caregiver, as in many applications of Vehicle Routing Problem (VRP), as it would be more difficult to deal with the concepts of synchronization.

Figure 15.1 provides an example of a solution representation. The example considers five patients (encoded with numbers from 1 to 5) and two caregivers (encoded as 1 and 2). For the sake of simplicity, optional patients are not reported in this example. The main and redundant data structures are distinguished by color.

We use this example to illustrate the construction of caregivers' routes. Starting from Π, the first patient to be served is patient 2, who requires both caregivers; therefore, patient 2 appears in the first position of both routes. Next, patient 5, served only by caregiver 1, is inserted after patient 2 in caregiver 1's route, and so on. Service times are computed as previously described and are served in related data structures, so to not have to recompute them everytime they are needed.

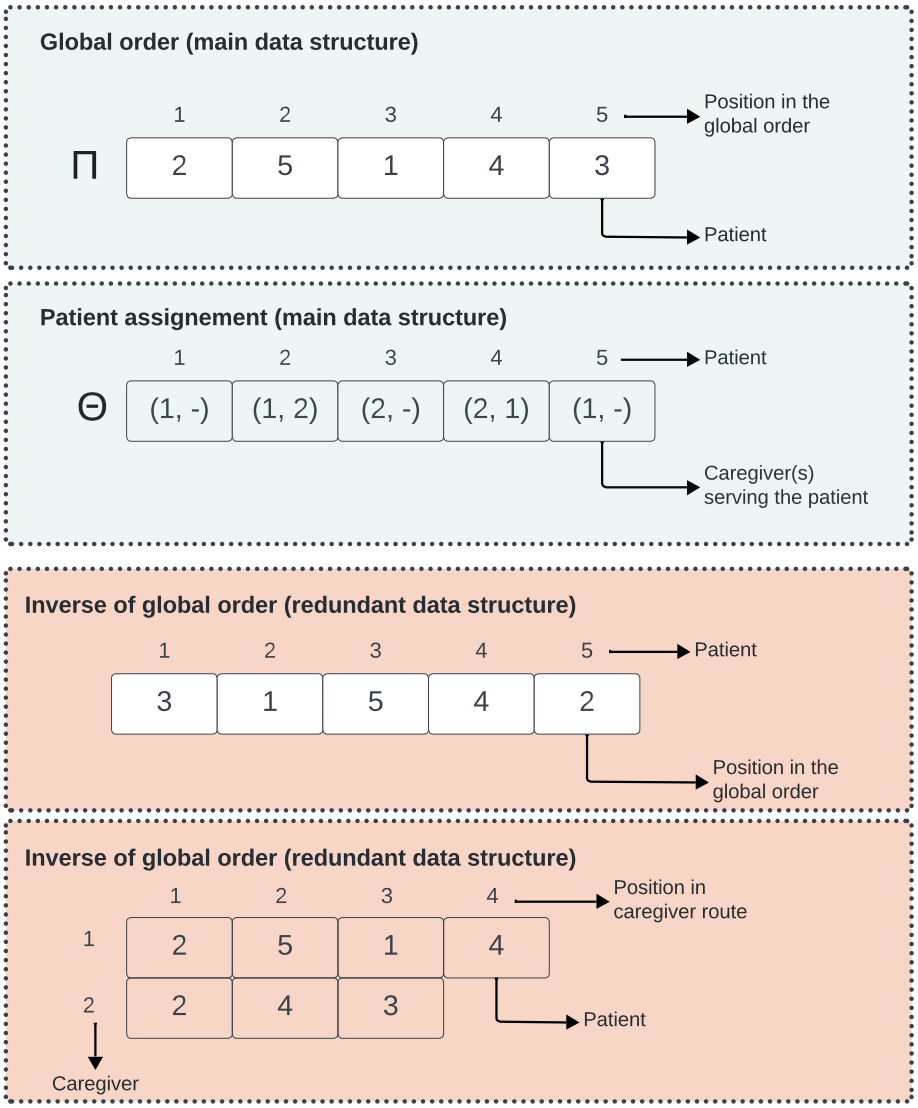


Figure 15.1: Example of solution representation for the Simulated Annealing (SA) used to solve the Home Healthcare Routing and Scheduling Problem (HHCRSP).

Initial Solution

The initial solution is generated randomly, consisting of a random permutation of patients and a random assignment of caregivers. Specifically, a random permutation of patients is first generated; then, for each patient, one or two feasible caregivers (depending on the patient's requirements) are selected uniformly at random.

Neighborhood Relations

Our SA considers the union of four neighborhoods:

- MovePatient: It repositions and reassign a patient.
- SwapPatients: It swaps two patients in terms of global position and caregivers.
- InRouteSwap: It swaps two patients within the same caregiver.
- FlipPatient: It inserts or removes optional patients from the schedule

The first three neighborhood relations (MovePatient, SwapPatients, and InRouteSwap) are applied exclusively to scheduled patients, i.e., those positioned in the head portion of the sequence. On the other hand, the solver detects the possible absence of optional patients, and in that case, the rate of FlipPatient is set to 0 throughout the entire search. Similarly, if certain features are not used, the corresponding cost component is assigned a weight of 0, and the search procedure does not compute its value. In addition, the auxiliary data structures necessary for their computation are not updated, saving computational time.

15.2 Experimental Setup

We evaluate the proposed methods using a defined experimental setup that includes the benchmark instances (Section 15.2.1), parameter-tuning procedure (Section 15.2.2), and general technical details (Section 15.2.3).

15.2.1 Instances

We perform some experimental analysis with the following datasets: *(i)* Bazirha, Kadrani, and Benmansour [41]; *(ii)* Bazirha, Benmansour, and Kadrani [40]; *(iii)* our newly generated dataset. We do not consider those by Hiermann et al. [220] given their limited scope. Comparisons on the datasets of the original formulation of Ceschia et al. [80], Kummer [271], and Mankowska, Meisel, and Bierwirth [327] are already reported by Ceschia et al. [80] for SA and by Montemanni, Ceschia, and Schaerf [354] for the MILP model, respectively. Therefore, we do not repeat them here.

15.2.2 Parameter Tuning

SA relies on a set of parameters that must be carefully tuned. Given that the datasets we analyze deal with different constraints and objectives, and that the instances present different characteristics, we perform three tuning procedures. That is, one for each dataset, i.e., *(i)* Bazirha, Kadrani, and Benmansour [41]; *(ii)* Bazirha, Benmansour, and Kadrani [40]; *(iii)* our newly generated dataset. For the first two datasets, we used the three-modal neighborhood SA, which uses MovePatient, SwapPatients, and InRouteSwap. For our new dataset, we employed quadri-modal SA, which also adds the FlipPatient neighborhood to the SA algorithm. This neighborhood was not employed in solving other datasets since they did not consider optional patients.

The tuning procedures were conducted using `Irace` [316], with a budget of 10,000 experiments. Based on preliminary experiments, the SA was granted in each case 10,000,000

evaluations. Table 15.1 presents both the parameter search ranges explored during tuning and the final parameter values. Note that the MovePatient probability bias is not included as a parameter since it is implicitly calculated as 1 minus the sum of the other probability biases.

Table 15.1: Parameter ranges and selected values for the Simulated Annealing (SA).

Parameter	Search range	Final value		
		Bazirha et al. [41]	Bazirha et al. [40]	New dataset
Start temperature	[10.0, 30.0]	20.803	90.1175	25.7795
Min temperature	[0.1, 1.0]	0.501	0.8035	0.5442
Cooling rate	[0.985, 0.995]	0.986	0.9932	0.9873
Neighbors accepted ratio	[0.07, 0.15]	0.122	0.0793	0.1181
SwapPatients rate	[0.0, 0.2]	0.101	0.1225	0.1677
InRouteSwap rate	[0.0, 0.2]	0.075	0.0858	0.0205
FlipPatient rate	[0.0, 0.1]	—	—	0.0915

15.2.3 Technical Details

The MILP model is implemented using `OR-Tools CP-SAT` v.9.12 [387]. The SA algorithm is implemented in `C++` using EasyLocal++ v.3 (Appendix B) and has been compiled using the `GNU g++` v. 13.3.

Experiments on the MILP model are conducted on an Intel Core i7 12700F machine; running times have been normalized to the machines used by Bazirha, Benmansour, and Kadrani [40]. Experiments on SA ran on an Ubuntu Linux 24.04 machine with 4 Intel i7–7700 (3.60 GHz) physical cores, hyper-threaded to 8 virtual cores. A single virtual core has been dedicated to each experiment.

15.3 Results

This section reports results on existing (Section 15.3.1) and newly generated instances (Section 15.3.2), and concludes with managerial insights on the impact of cost components on the final outcomes (Section 15.3.3).

15.3.1 Existing Instances

In this section, we compare our solution methods on the formulations and datasets introduced by Bazirha, Kadrani, and Benmansour [41] and Bazirha, Benmansour, and Kadrani [40], the other available datasets our formulation can fully model. In Bazirha, Kadrani, and Benmansour [41], the authors allocated a time limit of 4 hours, whereas in Bazirha, Benmansour, and Kadrani [40], they reduced it to 2 hours on the same machine (Intel i7–7600U 2.80 GHz CPU and 16 GB of RAM).

Bazirha, Kadrani, and Benmansour [41] proposed a two-stage stochastic model that minimizes transportation cost in the first stage, and the expected value of the recourse actions penalizing tardiness and overtime in the second stage. In their deterministic model, the authors focused solely on minimizing transportation costs. Thus, we adopted the

same objective function for our comparison. They introduced two datasets: dataset $\mathcal{SS}$ that includes only patients with a single service, and dataset $\mathcal{MS}$ with patients requiring multiple services. We excluded from our experimental analysis dataset $\mathcal{SS}$ because MILP models could easily reach all optimal solutions in Bazirha, Kadrani, and Benmansour [41]. Dataset $\mathcal{MS}$ consists of 3 groups ($\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$) with an increasing number of patients (10, 25, 50) and caregivers (3, 5, 10).

Table 15.2 presents comparative results of the different solution methods. Best values (optimal when available) are highlighted in bold.

We distinguish between the two different MILP models considered, i.e., the one from Bazirha, Kadrani, and Benmansour [41] and the new one we introduce, by the name of the solver used, `CPLEX`[236] and `CP-SAT`, respectively. For the MILP models, we report the Lower Bound (LB), the Upper Bound (UB), and the CPU time. For `CP-SAT`, the CPU time has been normalized to the machine used by Bazirha, Kadrani, and Benmansour [41]

In the case of the stochastic methods, each instance was executed 10 times. We report the average value (Avg), the best value (UB), and the average running time in seconds (t) over 10 executions. Our SA was granted 10M iterations.

Table 15.2: Comparative results the $\mathcal{MS}$ dataset by Bazirha, Kadrani, and Benmansour [41].

	Bazirha et al. [41]									Bazirha et al. [40]			Us					
	MILP (CPLEX)			GVNS			GA			SA			MILP (CP-SAT)			SA		
	LB	UB	t [s]	Avg	UB	t [s]	Avg	UB	t [s]	Avg	UB	t [s]	LB	UB	t [s]	Avg	UB	t [s]
$\mathcal{D}1$	769.0	**769**	1.6	**769.0**	**769**	<1	**769.0**	**769**	0.1	**769.0**	**769**	3.1	769	**769**	0.1	**769.0**	**769**	8.6
$\mathcal{D}2$	872.0	**872**	1.5	**872.0**	**872**	<1	**872.0**	**872**	0.1	**872.0**	**872**	2.5	872	**872**	0.1	**872.0**	**872**	8.5
$\mathcal{D}3$	709.0	**709**	1.8	**709.0**	**709**	<1	709.8	**709**	0.2	709.0	**709**	3.6	709	**709**	0.2	**709.0**	**709**	8.6
$\mathcal{D}4$	938.0	**938**	1.6	**938.0**	**938**	<1	**938.0**	**938**	0.1	**938.0**	938	2.3	938	**938**	0.2	**938.0**	**938**	8.6
$\mathcal{D}5$	777.0	**777**	1.6	**777.0**	**777**	<1	**777.0**	**777**	0.2	**777.0**	**777**	3.3	777	**777**	0.1	**777.0**	**777**	8.6
$\mathcal{D}6$	588.0	**588**	1.5	**588.0**	**588**	<1	594.2	**588**	0.1	**588.0**	**588**	2.8	588	**588**	0.1	**588.0**	**588**	8.6
$\mathcal{D}7$	609.0	**609**	1.5	**609.0**	**609**	<1	**609.0**	**609**	0.1	**609.0**	**609**	2.9	609	**609**	0.1	**609.0**	**609**	8.6
$\mathcal{E}1$	1317.0	**1317**	18.5	1336.1	**1317**	5.9	1382.7	**1317**	6.1	1355.0	**1317**	12.9	1317	**1317**	13.8	1330.9	**1317**	12.7
$\mathcal{E}2$	1361.0	**1361**	8.1	1391.7	1374	5.8	1407.8	1387	5.8	1385.0	**1361**	12.5	1361	**1361**	14.9	1480.6	1384	12.9
$\mathcal{E}3$	1338.0	**1338**	7.2	1357.5	**1338**	5.6	1419.1	1353	6.4	1367.5	**1338**	14.1	1338	**1338**	4.5	1362.3	**1338**	12.7
$\mathcal{E}4$	1150.0	**1150**	5.6	1159.4	**1150**	4.6	1207.2	1171	5.6	1151.4	**1150**	15.2	1150	**1150**	10.1	1150.3	**1150**	12.9
$\mathcal{E}5$	1246.0	**1246**	3.6	1268.6	**1246**	4.8	1286.9	1247	6.4	1250.0	**1246**	12.0	1246	**1246**	4.1	1254.2	1254	13.0
$\mathcal{E}6$	1251.0	**1251**	11.3	1259.3	**1251**	5.2	1286.7	**1251**	6.8	1268.6	**1251**	12.9	1251	**1251**	10.2	1253.6	**1251**	12.9
$\mathcal{E}7$	1145.0	**1145**	5.1	1147.6	**1145**	3.3	1155.1	**1145**	4.7	1145.3	**1145**	12.7	1145	**1145**	1.7	1159.1	**1145**	12.8
$\mathcal{F}1$	1754.0	**1754**	1971.0	1903.0	1808	276.6	2169.4	2069	113.8	1858.3	1780	46.8	1754	**1754**	760.6	1856.5	1796	20.1
$\mathcal{F}2$	1668.7	1864	14400.0	1933.6	1832	186.3	2166.2	1989	119.7	1876.4	1832	38.6	*1749	**1828**	14400.0	1890.3	1841	19.9
$\mathcal{F}3$	1685.0	1731	14400.0	1822.7	1813	205.2	2033.5	1940	129.1	1759.2	1731	43.1	1674	**1726**	14400.0	1769.1	1734	19.9
$\mathcal{F}4$	1721.7	1962	14400.0	1955.3	1908	231.2	2193.5	2105	96.9	1937.1	1908	47.4	*1883	**1883**	14400.0	1956.3	1930	20.0
$\mathcal{F}5$	1793.0	2011	14400.0	2179.0	2040	273.5	2502.5	2296	139.6	2037.1	2015	51.3	*1941	**2009**	14400.0	2103.8	2044	19.8
$\mathcal{F}6$	1808.0	**1808**	2207.0	1853.1	1818	209.6	2189.0	2035	148.7	1840.7	1820	42.1	1808	**1808**	4738.0	1854.5	1835	20.1
$\mathcal{F}7$	1708.1	**1730**	14400.0	1783.1	1738	196.1	2098.8	2012	113.6	1772.5	1740	45.7	*1730	**1730**	2801.4	1775.3	1748	20.1
Avg	1248.0	1282.4	3630.9	1314.9	1287.6	115.3	1417.5	1360.9	43.1	1298.4	1280.8	20.4	1135.6	**1276.6**	3141.0	1307.6	1287.1	13.8

Comparing the exact methods, it is evident that `CP-SAT` outperforms `CPLEX` in terms of both quality and computational time. In most cases (14 out of 21), `CP-SAT` was able to find the optimal solution in a shorter time. It also improved the lower bounds for four instances ($\mathcal{F}1$, $\mathcal{F}4$, $\mathcal{F}5$, $\mathcal{F}7$), marked with an asterisk, and certified the optimality for instance $\mathcal{F}3$. `CP-SAT` provided the best solution even for larger instances of group $\mathcal{F}$, although it required a long running time.

When focusing on the metaheuristic methods, it is important to note that SA algorithms, both the one by Bazirha, Benmansour, and Kadrani [40] and ours, clearly outperformed General Variable Neighborhood Search (GVN) and Genetic Algorithm (GA) by Bazirha, Kadrani, and Benmansour [41].

Comparing the two versions of SA, we can conclude that their performances are comparable. Specifically, the SA algorithm presented by Bazirha, Kadrani, and Benmansour [41] found 13 out of 21 best results, while our SA achieved 12 out of 21. The average value of our SA is 2.4% worse than the best solution, compared to a gap of 1.71% for the SA by Bazirha, Kadrani, and Benmansour [41]. However, the average running time of Bazirha, Kadrani, and Benmansour [41] is 20.4 seconds, while for our SA is 13.8 seconds on comparable machines.

Bazirha, Benmansour, and Kadrani [40] employed `CPLEX` and a SA algorithm. Furthermore, the authors generated three datasets, namely *MSMTW*, *Small*, and *Large*. Datasets *MSMTW* has three groups ($\mathcal{J}$, $\mathcal{K}$ and $\mathcal{L}$) with 10, 25 and 50 patients respectively. All patients in this dataset have two time windows. Dataset *Small* consists of the nine instances of group $\mathcal{M}$, each with 10 patients, several double-service patients ranging from 20% to 60%, and several time windows per patient varying from 1 to 3. Finally, dataset *Large* (group $\mathcal{N}$) includes only four instances that assign to each patient three time windows and up to three services. We excluded instances $\mathcal{N}$100s and $\mathcal{N}$200s because they involve patients requiring three services, and we focused only on instances $\mathcal{N}$100p and $\mathcal{N}$200p with 100 and 200 patients.

Table 15.3 shows the comparative results between our solution methods and those proposed by Bazirha, Benmansour, and Kadrani [40] in their formulation and datasets. In their paper, the workload balance deviation s_v is rounded up to $\lceil s_v \rceil$ for each single caregiver; we follow the same approximation for the comparison.

It is worth mentioning that our SA approach is not customized to this specific formulation and its constraints and features. For example, we do not exploit the fact that time windows are strict, and we allow the search not to meet them, but penalize this case with a high artificial cost.

Observing the results, all methods find optimal solutions for small instances of group $\mathcal{J}$ and group $\mathcal{M}$. In instances of group $\mathcal{K}$ (25 patients), `CP-SAT` could still identify numerous best solutions (4 out of 9) within 2 hours of CPU time. In contrast, `CPLEX` could find only one best solution. Regarding the SA algorithms, they display similar performances on average. For instances of group $\mathcal{L}$ (with 50 patients), the exact methods struggle; in particular, `CPLEX` could not deliver any solution within two hours, while `CP-SAT` could find a solution for 5 instances, albeit not of good quality. Comparing the two metaheuristic methods on this set, our SA outperforms on average the one by Bazirha, Benmansour, and Kadrani [40], obtaining several best solutions (i.e., 7 out of 9) in a shorter running time (always less than 23.5 seconds).

Finally, our SA, if granted more iterations (100,000,000) to have comparable running times, obtained better results on the larger instances ($\mathcal{N}$100p and $\mathcal{N}$200p), outperforming Bazirha, Benmansour, and Kadrani [40] by an average of 25% in the objective function.

15.3.2 New Instances

We now consider our newly generated dataset. Out of the complete set (465 instances), we randomly select 24 instances for demonstration purposes. `CP-SAT` was granted a maximum of 3600 seconds, whereas SA was granted 10,000,000 iterations (an average of 67.9 seconds).

Table 15.4 reports both the main features of the instances and the results we obtained.

Table 15.3: Comparative results on the dataset by Bazirha, Benmansour, and Kadrani [40].

	Bazirha et al. [40]					Us					
	MILP (CPLEX)		SA			MILP (CP-SAT)			SA		
	UB	t [s]	Avg	UB	t [s]	LB	UB	t [s]	Avg	UB	t [s]
$\mathcal{J}1$	**161.0**	3.7	161.8	**161.0**	6.3	161.0	**161.0**	16.5	**161.0**	**161.0**	8.9
$\mathcal{J}2$	**130.5**	12.9	**130.5**	**130.5**	5.2	130.5	**130.5**	49.6	**130.5**	**130.5**	9.2
$\mathcal{J}3$	**102.5**	6.0	105.3	**102.5**	6.0	102.5	**102.5**	10.0	**102.5**	**102.5**	9.1
$\mathcal{J}4$	**311.0**	6.0	**311.0**	**311.0**	4.9	311.0	**311.0**	16.7	**311.0**	**311.0**	9.8
$\mathcal{J}5$	**160.5**	13.6	**160.5**	**160.5**	4.7	160.5	**160.5**	13.9	**160.5**	**160.5**	9.4
$\mathcal{J}6$	**160.0**	10.4	**160.0**	**160.0**	4.0	160.0	**160.0**	32.8	**160.0**	**160.0**	9.6
$\mathcal{J}7$	**246.5**	5.7	**246.5**	**246.5**	4.9	246.5	**246.5**	8.3	**246.5**	**246.5**	7.9
$\mathcal{J}8$	**57.5**	6.6	**57.5**	**57.5**	4.8	57.5	**57.5**	0.7	**57.5**	**57.5**	9.5
$\mathcal{J}9$	**165.0**	32.5	166.1	**165.0**	4.2	165.0	**165.0**	144.6	205.0	**165.0**	9.6
$\mathcal{K}1$	177.0	7200.0	143.4	116.0	25.6	38.0	99.0	7200.0	93.0	**77.0**	14.4
$\mathcal{K}2$	82.5	7200.0	136.1	110.0	25.5	57.5	**64.5**	7200.0	98.3	72.0	14.0
$\mathcal{K}3$	118.5	7200.0	154.0	103.5	24.7	52.5	131.5	7200.0	122.6	**90.0**	14.3
$\mathcal{K}4$	**381.0**	7200.0	478.0	385.0	20.5	84.0	398.0	7200.0	441.3	409.5	14.7
$\mathcal{K}5$	219.5	7200.0	282.0	216.5	20.9	63.0	**191.5**	7200.0	225.7	199.5	15.0
$\mathcal{K}6$	286.0	7200.0	303.6	244.0	18.5	82.5	291.0	7200.0	268.8	**219.0**	15.0
$\mathcal{K}7$	294.0	7200.0	270.0	**165.5**	18.7	42.5	288.0	7200.0	238.3	182.0	15.1
$\mathcal{K}8$	76.5	7200.0	150.6	94.5	20.1	39.0	**61.5**	7200.0	114.6	82.5	14.9
$\mathcal{K}9$	201.5	7200.0	158.4	121.5	18.9	40.5	**98.0**	7200.0	139.5	123.5	14.9
$\mathcal{L}1$	—	7200.0	200.6	**152.0**	89.0	0.0	335.5	7200.0	188.0	169.0	22.1
$\mathcal{L}2$	—	7200.0	163.2	125.5	80.1	—	—	7200.0	129.0	**90.0**	22.1
$\mathcal{L}3$	—	7200.0	186.7	147.5	86.9	—	—	7200.0	158.3	**121.5**	22.2
$\mathcal{L}4$	—	7200.0	434.4	**403.0**	81.4	—	—	7200.0	520.6	438.0	22.3
$\mathcal{L}5$	—	7200.0	334.1	264.0	85.5	0.0	440.5	7200.0	309.6	**202.0**	22.4
$\mathcal{L}6$	—	7200.0	574.4	504.0	82.4	—	—	7200.0	598.5	**429.0**	22.5
$\mathcal{L}7$	—	7200.0	280.8	218.0	87.2	0.0	291.0	7200.0	280.2	**183.5**	23.3
$\mathcal{L}8$	—	7200.0	216.6	187.0	85.1	0.0	281.0	7200.0	177.2	**128.5**	23.1
$\mathcal{L}9$	—	7200.0	258.9	183.5	79.5	0.5	217.5	7200.0	227.4	**137.5**	23.4
$\mathcal{M}1$	**223.0**	2.8	**223.0**	**223.0**	3.0	223.0	**223.0**	1.5	**223.0**	**223.0**	8.7
$\mathcal{M}2$	**144.0**	6.8	**144.0**	**144.0**	4.3	144.0	**144.0**	3.7	**144.0**	**144.0**	8.8
$\mathcal{M}3$	**8.5**	8.8	**8.5**	**8.5**	4.2	8.5	**8.5**	4.6	**8.5**	**8.5**	8.8
$\mathcal{M}4$	**162.5**	2.2	**162.5**	**162.5**	4.7	162.5	**162.5**	15.0	**162.5**	**162.5**	9.0
$\mathcal{M}5$	**146.5**	14.9	**146.5**	**146.5**	6.8	146.5	**146.5**	20.0	**146.5**	**146.5**	9.3
$\mathcal{M}6$	**51.0**	25.9	51.7	**51.0**	7.0	51.0	**51.0**	26.6	55.0	**51.0**	9.2
$\mathcal{M}7$	**424.5**	202.0	428.0	**424.5**	5.4	424.5	**424.5**	1504.2	**424.8**	**424.5**	10.6
$\mathcal{M}8$	**51.0**	436.9	70.0	**51.0**	6.6	51.0	**51.0**	208.7	**51.2**	**51.0**	10.6
$\mathcal{M}9$	**29.5**	32.0	33.8	**29.5**	6.6	29.5	**29.5**	1554.3	29.6	**29.5**	10.7
$\mathcal{N}100$p	—	7200.0	99.3	81.0	269.8	—	—	7200.0	77.2	**57**	673.7
$\mathcal{N}200$p	—	7200.0	242.8	215.5	1409.1	—	—	7200.0	197.6	**138**	673.7
Avg		3811.3	206.2	178.2	71.7			3885.0	194.3	**165.4**	40.1

Regarding the features, the first five columns report the number of patients (C'), caregivers (C), services ($\sum_{i \in C'} |S_i|$), the percentage of double-service patients ($|C_2|/|C'|$), and optional patients ($|C'_{opt}|/|C'|$). The dataset exhibits significant diversity, comprising instances from 10 to 450 patients. The proportion of patients requiring two services varies from 0 % to 100%, as in the case of the optional patients.

CP-SAT optimally solved instances up to 15 patients (i-116 and i-134) in a short CPU time. For instances with 25 patients, it still achieves the best values for three out of four cases in 1 hour; results of SA in these instances are, on average, worse by 13%, but with a much

Table 15.4: Results on the new dataset.

	Features					MILP (CP-SAT)			SA		
	$\|C'\|$	$\|V\|$	$\sum_{i\in C'}\|S_i\|$	$\|C_2\|/\|C'\|$	$\|C'_{opt}\|/\|C'\|$	LB	UB	t [s]	Avg	UB	t [s]
i-054	200	30	220	10%	10%	—	—	3600.0	69904.4	**65188**	70.7
i-077	100	17	120	20%	20%	—	—	3600.0	17114.5	**16164**	48.6
i-083	70	11	98	40%	0%	7349	15725	3600.9	15828.3	**12831**	40.3
i-100	25	5	35	40%	0%	14538	**14744**	3600.0	17916.6	15119	25.1
i-116	10	4	13	30%	60%	17117	**17117**	0.1	18303.5	17393	13.6
i-126	200	32	260	30%	20%	—	—	3600.0	49607.6	**47475**	86.9
i-134	15	3	17	13%	60%	15616	**15616**	6.4	15820.0	15820	16.0
i-164	200	27	240	20%	0%	—	—	3600.0	61539.7	**58873**	83.9
i-167	100	13	109	9%	100%	—	—	3600.0	16818.2	**13805**	40.0
i-175	450	63	539	20%	0%	—	—	3600.0	34314.7	**34014**	186.3
i-185	340	62	442	30%	20%	—	—	3600.0	26839.8	**26591**	137.6
i-219	100	18	130	30%	20%	—	—	3600.0	11535.0	**11086**	51.4
i-235	25	6	44	76%	48%	1913	**4702**	3600.0	7694.3	6021	21.9
i-247	25	5	32	28%	48%	18881	**19492**	3600.0	21383.0	21186	24.7
i-250	190	26	217	14%	30%	—	—	3600.0	29697.0	**28639**	85.4
i-263	250	42	324	30%	20%	—	—	3600.0	38049.8	**36933**	111.8
i-272	170	26	194	14%	50%	—	—	3600.0	27974.6	**27672**	74.2
i-316	25	5	50	100%	0%	9063	10300	3600.0	11510.3	**10196**	32.5
i-360	200	35	300	50%	20%	—	—	3600.0	49530.6	**46205**	89.7
i-369	75	13	97	29%	20%	16402	26346	3600.0	26947.4	**25225**	41.8
i-406	100	18	130	30%	50%	—	—	3600.0	26197.6	**25260**	43.1
i-414	70	11	98	40%	0%	24660	37633	3600.0	35982.0	**33457**	38.4
i-416	400	46	400	0%	0%	—	—	3600.0	42150.7	**41009**	131.6
i-446	75	13	111	48%	20%	15965	28872	3600.0	21768.3	**21463**	49.3

shorter running time of ≈ 26 seconds. CP-SAT can provide a feasible solution for up to 75 patients within the time limit; however, SA typically achieves an average improvement of about 13% in approximately 40 seconds.

Comparing the two methods is not possible for larger instances because CP-SAT could not provide any solution in 1 hour. We can observe that the running time of SA increases proportionally to the number of patients and that the deviation between average and UB values is more relevant when there are many double-service patients (instances i-083, i-100, i-235, and i-316) and optional patients (instance i-167).

We observe from Tables 15.2–15.4 that the running time for both our methods increases significantly with the complexity of the formulation. For example, for SA for instances with 25 patients it goes from 13, to 15, to 26 seconds, approximately.

15.3.3 Managerial Insights

In this section, we frame the analysis through the lens of Value-Based Purchasing (VBP), a policy instrument increasingly used by Ministries and Departments of Health of different countries to realign incentives across patients, providers, and payers. VBP shifts reimbursements and incentives away from mere activity volume toward outcome- and experience-based metrics, thus rewarding hospitals and other healthcare organizations for delivering high-quality, patient-centered care, while containing cost growth [252]. For example, in 2022, the Centers for Medicare & Medicaid Services (USA) launched the Ex-

panded Home Health Value-Based Purchasing Model,[1] whose primary goal is to "provide incentives for better quality care with greater efficiency". Comparable initiatives are now emerging in Europe with the adoption of VBP broadly in healthcare related topics.[2] These programmes underscore the need to analyze the trade-offs among stakeholder objectives that our model addresses.

Therefore, we decided to consider the perspectives of the different stakeholders, and investigate how individual cost components behave when the optimization is steered toward a single stakeholder or operational goal. More specifically, we adopted the objective-function breakdown presented in Table 14.1, which separates cost components by stakeholder (patient, caregiver, organization) and by operational goal (efficiency, fairness, well-being). For every instance, we reran our SA under the six scenarios, each time considering only one cost group (patient, caregiver, organization, efficiency, fairness, well-being) while keeping all other algorithmic parameters identical to the ones previously outlined. We conducted 10 runs $\times$ 24 instances $\times$ 6 scenarios = 1,440 experiments.

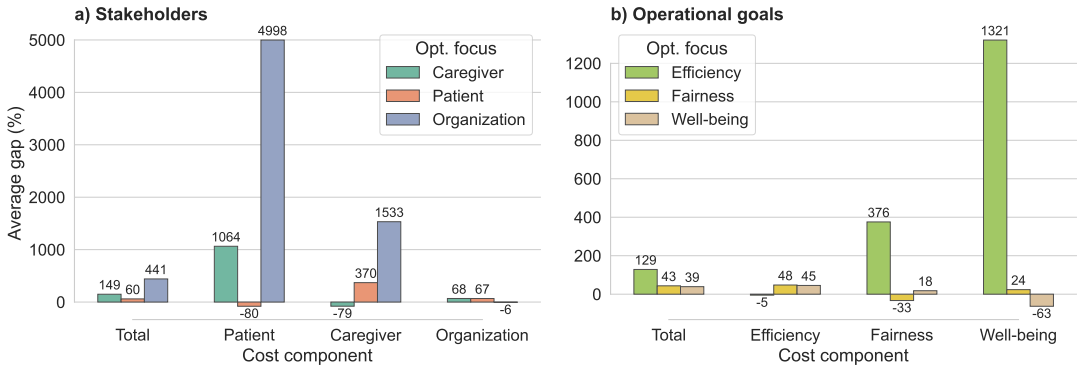


Figure 15.2: Average gap (%) relative to the fully-optimized solution when the optimization is carried out for different stakeholder groups (left) and operational objectives (right)

Figure 15.2 shows the results of these analyses through bar charts. Different colors represent results for various stakeholders (left) or operational goals (right). To compare, we calculated the average percentage gap as the difference between the cost from a single cost group and the total cost, divided by the total cost. A negative value indicates that focusing on a specific group leads to an improvement in the value of the corresponding cost group. It could be noticed that, when we adopt a patient-centered perspective and ignore the soft constraints related to caregivers and organization, the total cost increases only moderately (+60%) compared to other cases, which show increases of +149% for caregivers and +441% for organization. The cost components related to patients decrease by 80%, while costs for the organization and caregivers worsen by 67% and 370%, respectively. Focusing on caregivers, their related cost components decrease by 79%, whereas costs related to the organization increase by 68%, and those related to patients rise by 1,064%. Unexpectedly, the worst results are observed from the organization's perspective. Specifically, the reduction in organization costs is only 6%, while costs for patients and caregivers increase by 4,998% and 1,533%, respectively. This analysis suggests adopting a patient perspective, in line with

[1] https://www.cms.gov/priorities/innovation/innovation-models/expanded-home-health-value-based-purchasing-model. Accessed 2025–07–22

[2] https://www.sanidad.gob.es/gabinete/notasPrensa.do?id=6475. Accessed 2025–07–22

recent trends that push healthcare institutions towards the creation of a patient-centered healthcare delivery system [350].

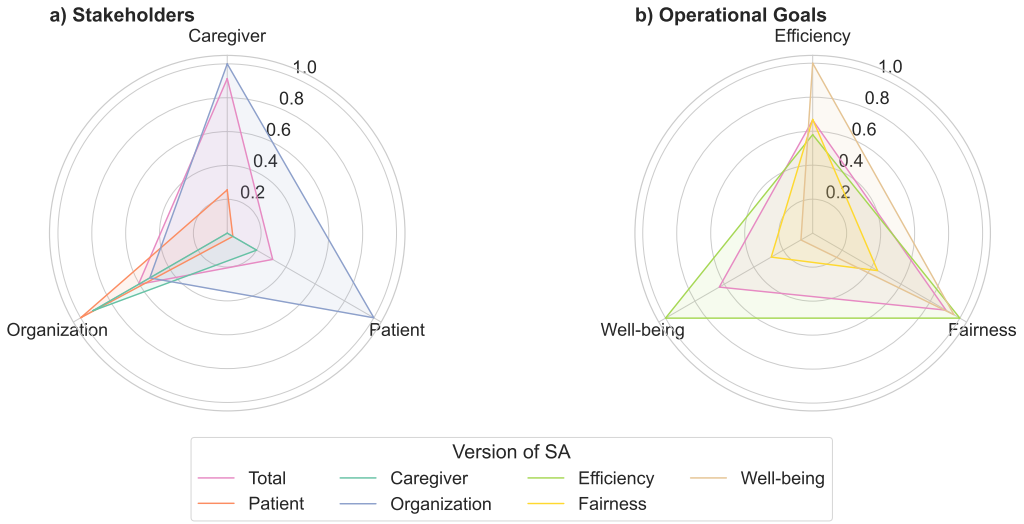


Figure 15.3: Radar plot for instance i-083.

Figure 15.3 contrasts the optimization strategies for a representative instance, i.e., i-083. Each axis reports the normalized cost for one stakeholder (left) or operational goal (right), where smaller values denote better performance. Considering the stakeholders, the organization-focused optimization attains the lowest organizational cost (≈ 0.5, as one would expect), but does so at the expenses of patients and caregivers (both ≈ 1.0), and eventually has the highest mean cost (≈ 0.8). Caregiver-focused optimization almost eliminates caregiver cost (≈ 0), keeps patient cost low (≈ 0.2), yet pushes the organizational cost up to ≈ 0.95 These large spreads among objectives signal a strong imbalance. Patient-focused optimization excels in patient metrics (≈ 0.05) and performs well on caregiver metrics (≈ 0.25), but incurs the highest organizational costs (1.0). Joint optimization of the full cost components offers a more balanced compromise across all three criteria with a relatively low standard deviation (≈ 0.25), suggesting it provides a fairer solution that does not significantly sacrifice any single stakeholder's priorities at the expense of others. Regarding the operational goals, efficiency and well-being clearly trade off, while fairness holds the lowest average cost (≈ 0.4). As expected, the efficiency goal's performance aligns with those obtained when focusing on the organization perspective. Additionally, it is confirmed that SA achieves the most balanced solutions among all three criteria (with a standard deviation ≈ 0.14).

To isolate the influence of incompatibility and preference constraints on the total cost, we conducted an additional experiment in which these two constraint classes were removed while all other hard and soft constraints remained active. Across the test set, on average, 41% of patients express preferences (range 0–75%) and 3% of patient–caregiver pairings exhibit incompatibilities (range 0–15%). Eliminating both incompatibilities and preferences yielded a mean improvement of less than 1%, rising to roughly 6% when the focus is restricted to patient-related cost components. This outcome is unsurprising: without incompatibilities and preferences, the solver can form routes with lower tardiness. Because such constraints reflect real-world clinical requirements and stakeholders' wishes, and

because their omission delivers only a marginal cost saving, we believe their explicit inclusion in the formulation is well justified.

15.4 Concluding Remarks

This section summarizes the main content of the chapter (Section 15.4.1), discusses its limitations (Section 15.4.2), and outlines directions for future research (Section 15.4.3).

15.4.1 Content

In this chapter, we adapted to this general problem two search methods that already proved to be at a state-of-the-art level for the basic formulation by Mankowska, Meisel, and Bierwirth [327], namely a compact MILP model and a SA algorithm. We provided results obtained by these methods on both available and novel datasets. Both methods found results at the same level or better than the solution method specifically developed for the given formulation. The novelty is, on the one hand, an extension of existing methods. On the other hand, we believe it constitutes an important unification effort.

15.4.2 Limitations

It should be noted, however, that some of the features proposed in the literature have been left out of the current framework and therefore of the solutio methods. The main ones are:

- **Multi-day**: When planning across multiple days, additional challenges emerge, including maintaining continuity of care and balancing caregiver workloads between days. Multi-day planning can be approached as a series of sequential single-day problems, adjusting preferences and patient status (optional/mandatory) between consecutive days. However, solving the entire planning horizon simultaneously typically yields superior overall solutions.

- **Uncertainty/stochasticity**: Since real-world healthcare operations have some inherently non-deterministic aspects, stochastic elements will better reflect their nature. For example, variability in service times, travel delays, and patient availability is inevitable in practice. Robust optimization models that accommodate unexpected events and dynamic rescheduling mechanisms would significantly enhance the practical applicability of our solution approaches.

- **Multi-modal**: Currently, we assume that all caregivers use the same means of transportation (i.e., cars), and consequently that the traveling time between two locations is a single value. It is generally possible to use alternative means exhibiting different travel times, costs, and environmental impacts (e.g., employing public and private transportation). This extension would add a new set of decision variables representing the modeling choices for the transportation modality of each route.

- **Perishable biological samples and medicines**: When collecting biological samples that could degrade, caregivers must return to the hospital or a clinical analysis lab within a short timeframe after service. Similarly, caregivers must promptly travel

from the hospital/pharmacy to the patient when delivering perishable medicines, avoiding unnecessary intermediate stops [474].

15.4.3 Future Work

In future work, we plan to incrementally add all the features mentioned in Section 15.4.2, trying to blend them smoothly within the current framework and without compromising the ability to solve the overall problem with a single general solution method.

Part V

Large Language Models

Chapter 16

Introduction

While successful applications of Combinatorial Optimization (CO) abound, the design and engineering of optimization tasks remain largely human-driven. Practitioners must translate informal problem descriptions into formal models and subsequently implement, configure, and run solvers or algorithms. These steps are non-trivial, knowledge-intensive, and often repeated across similar problems, making the development pipeline both slow and expertise-dependent.

Recent advances in Large Language Models (LLMs) have brought interest in automating or assisting parts of this pipeline [171, 230, 341, 500] as they can process Natural Language (NL) and produce structured artefacts. For instance, LLMs can support tasks such as translating textual descriptions into formal models [215, 238] or generating solver-related code [283, 467]. These capabilities suggest a potential shift from manual model formulation to interactive, language-based optimization engineering.

Yet, this emerging research area remains fragmented and immature. The rapid proliferation of studies has produced a highly heterogeneous landscape: tasks, instance representations, and evaluation protocols vary widely, while common baselines and reporting standards are often absent. Moreover, most work focuses on what LLMs can do, rather than how they behave as reasoning or optimization agents. Mechanism-level analyses, common in other domains such as geospatial or scientific reasoning [131], are still scarce within CO. As a result, the community lacks a principled understanding of when, why, and under which representations LLMs can reliably support combinatorial reasoning.

16.1 Goals

Our aim in this part is twofold: *(i)* to map the state of the art on the use of LLMs in CO; and *(ii)* to assess what information about a *given instance* LLMs actually encode and can make available for downstream optimization tasks. More programmatically, this part of the thesis answers the following Research Questions (RQs):

RQ1 What is the current landscape of LLMs in CO and in Combinatorial Optimization Problems (COPs)?

(**a**) How are LLMs currently being applied to COPs?

(**b**) Which stages of the optimization pipeline are supported (e.g., modelling, instance/feature extraction, solver guidance, parameter control, analysis)?

(**c**) Which LLM architectures and training paradigms are most commonly used and reported as effective for CO?

(**d**) Which application domains are employing LLMs within CO?

(**e**) What are the main empirical and methodological trends?

(**f**) What research directions emerge from gaps and inconsistencies in the literature?

RQ2 Given an instance of a COP, what do LLMs *know* and how does this depend on the representation?

(**a**) To what extent can LLMs identify salient structural or numerical characteristics of CO instances from their representations?

(**b**) Can representations extracted from the hidden layers of LLMs serve as a proxy to predict instance features?

(**c**) Can representations extracted from the hidden layers of LLMs serve as an effective proxy for per-instance algorithm selection, and how does their predictive power compare to that of traditional handcrafted features?

To address **RQ1**, we perform a systematic investigation of the intersection between LLMs and CO. We conduct a comprehensive search across Scopus and Google Scholar, initially retrieving over 2,000 publications. Each study is evaluated against four (4) inclusion and four (4) exclusion criteria concerning its language, research focus, publication year, and type. After screening, 103 studies are retained for analysis. We classify the selected studies into semantic categories and thematic topics, thereby offering a consolidated view of current trends and gaps. The review process follows the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines to ensure transparency and reproducibility. The investigation addressing this RQ is presented in Chapter 17, which relies on the following work:

- **Da Ros, F.**, Soprano, M., Di Gaspero, L., & Roitero, K. (2024, submitted). Large language models for combinatorial optimization: A systematic review. *Submitted to a journal*. Available as preprint at https://doi.org/10.48550/arXiv.2507.03637.

In **RQ2**, we investigate how LLMs represent and reason about COPs. We design a dual methodological framework combining *direct querying*, to analyze what models can explicitly articulate about optimization instances, and *probing*, to examine what information is implicitly encoded in their hidden representations. The study focuses on a representative set of problems and a set of different instance representations. We analyze whether LLMs can identify relevant instance features, whether these features are represented in LLMs hidden layes, and whether these latent representations can be used as predictors for per-instance algorithm selection. The detailed methodology and results addressing this research question are presented in Chapter 18, which draws on the following work:

- **Da Ros, F.**, Di Gaspero, L., & Roitero, K. (2025). Probing LLMs on optimization problems: Can they recall and interpret problem features? In *International Conference*

on the Applications of Evolutionary Computation (Part of EvoStar) (pp. 353–371). Cham: Springer Nature Switzerland. Conference rank: CORE2023 B.

- **Da Ros, F.**, Di Gaspero, L., & Roitero, K. (2025, submitted). Behavior and representation in large language models for combinatorial optimization: From feature extraction to algorithm selection. *Submitted to a journal*. Available as preprint at https://doi.org/10.48550/arXiv.2512.13374.

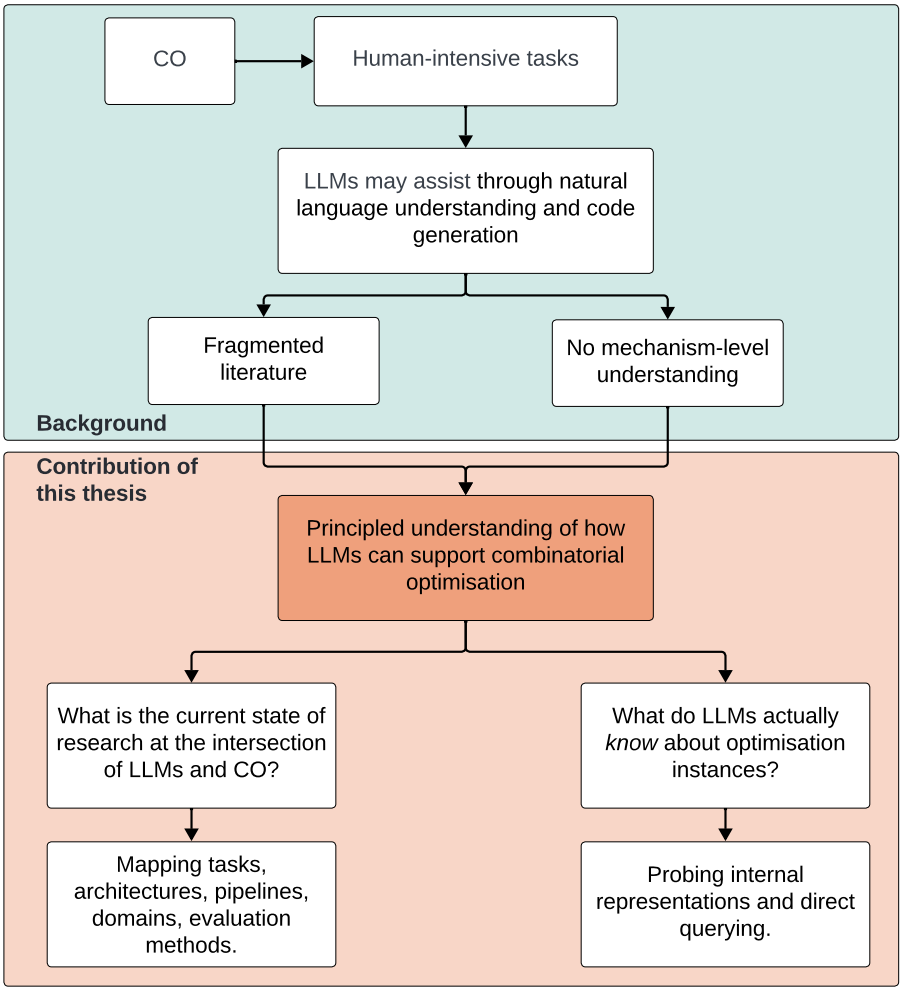


Figure 16.1: Overview of the goals and content of this part of the thesis. The abbreviation CO stands for Combinatorial Optimization, whereas LLM stands for Large Language Model.

16.2 Structure

This part of the thesis is organized as follows. Chapter 17 presents a systematic review of the literature on the use of LLMs in CO. Chapter 18 then introduces probing and algorithm-selection experiments designed to assess the extent to which LLMs can capture features of COPs.

Chapter 17

Systematic Literature Review

The rapid advancement in Artificial Intelligence (AI) and in particular in Natural Language Processing (NLP) has led to a rapid increase in the capabilities and applications of Large Language Models (LLMs), resulting in a proliferation of scholarly works and models being developed. While highlighting the increasing activity in the field, this multitude of studies has created a complex body of knowledge that is challenging to navigate. Looking specifically at the application of LLMs to Combinatorial Optimization (CO), the academic literature is fragmented, with existing works characterized by diverse methodologies, areas of applications, and findings.

This systematic review aims to consolidate the current state-of-the-art in LLMs applied to CO. We identify, screen, analyze, and systematically organize the literature to clarify the topic and identify strategic directions for ongoing and future research efforts. Through this examination, we seek to understand the capabilities of LLMs in addressing complex optimization tasks and to explore the evolving trends and directions in this field. By systematically synthesizing and analyzing existing research, this review aims to provide a structured understanding of how LLMs are employed in CO and offer insights that can inform future research in the field. The process is reported following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines.

While exploring the reverse relationship between LLMs and CO (i.e., applying CO techniques to enhance LLMs) is also a valid research direction, we deliberately chose not to include it in this survey. This is primarily because such applications essentially use well-established CO methods in a new, albeit challenging, domain. Instead, we focus on the novel and emerging contributions of LLMs to the practice of CO.

Our review emphasizes CO rather than other paradigms, such as continuous optimization. This decision is also motivated by the inherent differences between the two paradigms: continuous optimization typically relies on mathematical models or black-box ones, whereas CO encompasses a broader and more diverse set of problem domains (e.g., scheduling, assignment, routing, permutation-based problems) characterized by intricate substructures and complex constraints. This diversity makes the application of LLMs to CO potentially more impactful, given the richness and complexity of the CO landscape.

Although this section surveys related work, it is conducted as a systematic literature review and therefore follows an experimental layout. We adopt the PRISMA methodology (Section 17.1). The experimental setup is described in Section 17.2. Results are presented in

Section 17.3. Section 17.4 offers a synthesis and conclusions.

Appendix G complements this chapter by reporting the PRISMA checklists, complete descriptions of the included studies, and a detailed tabular analysis of the findings.

17.1 Methodology

To ensure methodological rigour, we follow the PRISMA guidelines, detailed in Section 17.1.1. Section 17.1.2 defines the terminology adopted throughout the chapter.

17.1.1 PRISMA

The Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines are a well-known framework for reporting systematic reviews. The current version was proposed in 2020 [378] and builds upon the PRISMA 2009 formulation [352]. This methodology evolved from the earlier Quality of Reporting of Meta-analyses (QUOROM) guidelines [351], which were firstly introduced in 1999. This evolution over time has broadened the scope of the guidelines. While QUOROM primarily focused on improving reports of meta-analyses in clinical trials, PRISMA addresses systematic reviews that evaluate the effects of health interventions more broadly. These guidelines are sufficiently general to apply to reports of systematic reviews that evaluate other types of interventions [297, 499], systematic reviews with objectives beyond evaluating interventions [216, 418], and systematic reviews not related with the medical field [361, 437]. PRISMA can be used to report systematic reviews that involve result synthesis, such as meta-analyses and other statistical methods. It is also helpful for reviews identifying only a single eligible study and for mixed-methods approaches.

At its core, PRISMA is composed of four main elements: the statement [378], the explanation and elaboration document [377], the checklist,[1] and the flow diagram.[2] The statement introduces the purpose of the methodology. The checklist consists of 27 items across 7 sections, providing guidelines for writing a systematic review report. It should be used alongside the explanation and elaboration document, which offers additional reporting guidance for each item. The diagram shows the flow of information through the review phases.

17.1.2 Terminology

Most of the terminology used within PRISMA resources [377, 378] derives from its original field of application, i.e., health-related interventions, which may be confusing for researchers from other disciplines. We particularly refer to the following definitions [378]:

- **Study**: Experiment including a defined group of participants and one or more interventions and outcomes.
- **Report**: Paper providing information about a particular study. Multiple reports may refer to the same study.

[1] https://www.prisma-statement.org/prisma-2020-checklist. Accessed 2025–10–01.

[2] https://www.prisma-statement.org/prisma-2020-flow-diagram. Accessed 2025–10–01.

- **Record**: Title or abstract (or both) of a report indexed in a database or website.
- **Outcome**: Measurement event for participants in a study.
- **Result**: Combination of a point estimate and precision measurement for an outcome.

Since our systematic review covers a broader scope than health-related interventions, where we expect to find only individual eligible studies rather than multiple studies referring to a given experimental design, we join the definitions of *study* and *report*, using *study* to refer to papers in general. This approach apply also to the definitions of *result* and *outcome*. We consider the term *record* to encompass titles and abstracts of papers. For example, the PRISMA flow diagram explicitly distinguishes between records screened and reports sought for retrieval. We adjust this to refer only to records as we define them.

17.2 Experimental Setup

We now report the experimental setting of our literature review by first describing the overall context (Section 17.2.1) and then by detailing all the phases (Sections 17.2.2 to 17.2.4)

17.2.1 Process

Our systematic review includes studies up to the end of 2024. The literature collection process, conducted according to PRISMA, involves three main activities (Figure 17.1): identification, screening, and inclusion.

While the inclusion activity simply involves reporting the number of studies included in the systematic review, the task of identifying and screening records is more complex. Figure 17.2 reports a Business Process Model and Notation (BPMN) diagram [368] describing how we conducted these activities. Initially, all four (4) authors[3] worked together to define the keywords for searching records, establish eligibility criteria for study inclusion/exclusion, and select the databases for record identification. Subsequently, one (1) author sent queries to the selected databases to retrieve records and performed data cleaning to remove duplicates and records without authors. Then, the same author screened the remaining records by reviewing titles, abstracts, and publication years in order to apply an initial filter. The resulting list was then augmented through citation tracking [114]. After this phase, three (3) authors independently read the full text of one-third of the studies each. They then decided whether to include each study based on the eligibility criteria. Subsequently, all authors collectively cross-checked the studies read by the other authors. The author not initially involved in full-text reading performed the final screening and resolved conflicts.

17.2.2 Identification

On January 7, 2025, we searched two (2) popular databases to identify records: Scopus and Google Scholar. Scopus is an extensive, multidisciplinary database of peer-reviewed literature, while Google Scholar allows for broader searches, including pre-prints.

To identify studies addressing the usage of LLMs within CO, we used the following set of keywords: “large language models”, “generative artificial intelligence”, “GPT”,

[3] In this chapter, we intend as authors the ones involved in the publication from which the study draw from.

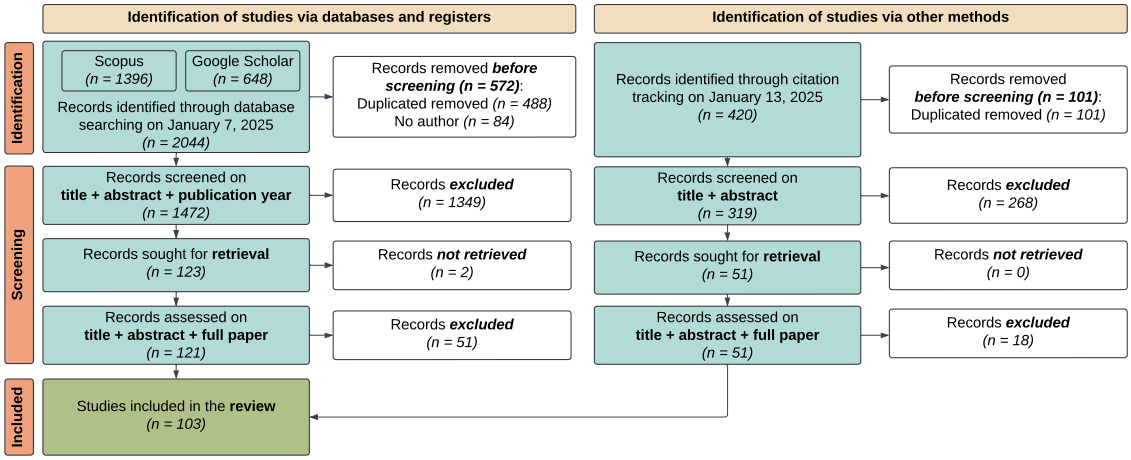


Figure 17.1: Literature identification, screening, and inclusion activities conducted following the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines.

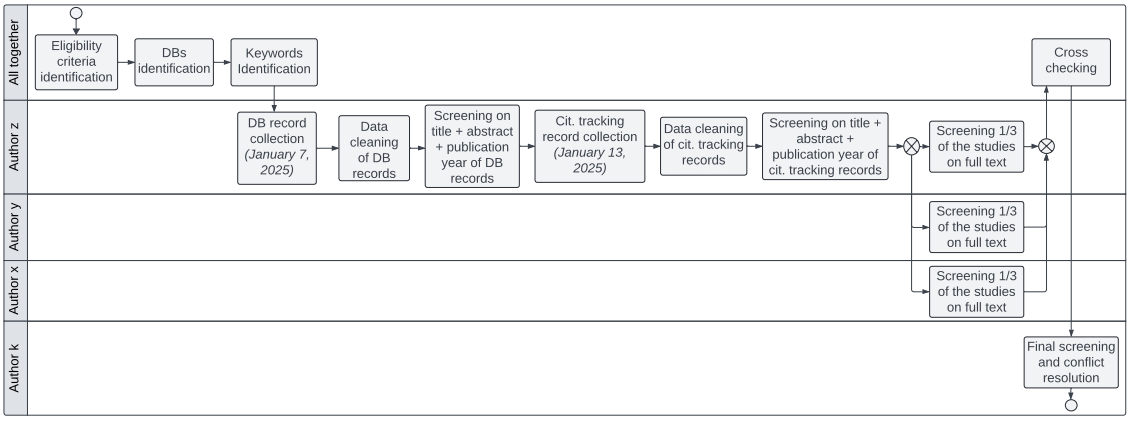


Figure 17.2: Business Process Model and Notation (BPMN) diagram describing the identification and selection activities of records in the literature collection process.

"optimization", "combinatorial optimization", "mathematical formulations", "metaheuristics", "constraint programming", "integer programming", "integer linear programming", "NL4Opt", "Ner4Opt". These keywords were combined into 14 search queries, reported in Table 17.1 along with the number of matches found for each query. Besides the self-explanatory queries (i.e., those including the relevant LLM and CO terms like "combinatorial optimization"), we also include the keywords "NL4Opt" and "Ner4Opt" in our Google Scholar search. These terms relate to a recent competition focused on extracting the meaning and formulation of an optimization problem from its textual description. Note that Google Scholar only provides an estimated number of matches and that we used the advanced query functionality on Scopus to restrict the search to study titles, abstracts, and keywords using the `TITLE-ABS-KEY(...)` construct. Using these queries, we retrieved approximately 648 records from Google Scholar and 1,396 from Scopus, for a total of 2,044 records. This list required further processing because both databases contain duplicates. We found 269 duplicate records from Google Scholar, 196 from Scopus, and 23 shared records, totaling to 488 duplicates. Another case to consider is a small subset of records that lack an author string and should, therefore, be removed from the list of 2,044 records.

Specifically, we identified 4 records without an author string on Google Scholar and 84 records on Scopus. When combining the two lists, the number of distinct records without an author string is 84, as all four records found on Google Scholar are also present on Scopus. The final number of distinct records from Google Scholar and Scopus is 1,472.

Table 17.1: Queries used to retrieve records from Scopus and Google Scholar.

Scopus		Google Scholar	
Query	Count	Query	Count
"large language models" and "optimization"	944	"large language models" and "combinatorial optimization"	334
"GPT" and "optimization"	394	"large language models" and "constraint programming"	104
"large language models" and "constraint programming"	10	"large language models" and "mathematical formulations"	106
"large language models" and "integer programming"	14	"large language models" and "metaheuristics"	88
"large language models" and "integer linear programming"	15	"NL4Opt"	11
"generative artificial intelligence" and "combinatorial optimization"	0	"Ner4Opt"	5
"large language models" and "combinatorial optimization"	14		
"GPT" and "combinatorial optimization"	5		

17.2.3 Screening

We define 8 eligibility criteria to screen the list of studies referred by the 1,472 records retrieved. Specifically, we define 4 inclusion (INCL) criteria and 4 exclusion (EXCL) criteria:

INCL1 The study is written in English.

INCL2 The study focuses primarily on the usage of LLMs in the field of CO.

INCL3 The study has been published in 2016 or later.

INCL4 The study is a research article, a case report, a technical note, a narrative review, a systematic review, a position paper, a pictorial essay, or a PhD Dissertation.

EXCL1 The study is written in languages other than English.

EXCL2 The study focuses on optimization in the context of LLMs, solely on CO, solely on LLMs, or on optimization different from CO, such as Continuous Optimization.

EXCL3 The study has been published in 2015 or earlier.

EXCL4 The study is not a research article, a case report, a technical report, a narrative review, a systematic review, a position paper, a pictorial essay, or a PhD Dissertation.

In summary, we focus on how LLMs are used in optimization, specifically targeting studies that address the subfield of CO (INCL2); consequently, we exclude those taking the opposite perspective (EXCL2). Note that some of the exclusion criteria may appear

redundant w.r.t. the inclusion ones; this is due to a rigorous application of the PRISMA guidelines in reporting on our activity of review.

The formulation of these criteria is based on the concept of " primary focus" on the use of LLMs in the field of CO. According to our criteria, a study focusing on the use of LLMs in Combinatorial Optimization Problems (COPs) should either:

1. Describe the usage of LLMs within one of the tasks of the overall optimization process.
2. Address specific aspects of a COP.
3. Propose improvements or new approaches for a given optimization paradigm.
4. Outline use cases, perspectives, and/or potential future applications.
5. Involve specific LLMs and, if applicable, provide metrics and/or reference datasets.

We believe that this notion of " primary focus" allows us to consider not only research papers but also reviews and surveys, which are equally important as they may provide general directions and perspectives (INCL2 and INCL4). This also helps clarify the rationale behind EXCL2 and EXCL4; if any of the components described above is missing, we exclude the study from consideration. For example, let us hypothesize a study on how an LLM could enhance an evolutionary procedure for solving a COP; this study would align with our inclusion criteria. Conversely, consider a study proposing the use of a metaheuristic to generate prompts for LLMs; such a study would not meet our criteria and would be excluded. Concerning the criteria for the publication year (INCL3/EXCL3), we selected 2016 as the earliest publication year to ensure comprehensive coverage for our systematic review. This choice aligns with key developments in LLMs, particularly the "Attention Is All You Need" paper [476], which introduced the Transformer architecture in 2017 (Section 2.5).

Regarding the types of publications considered, we have identified a subset that aligns with our objectives (INCL4). We therefore exclude all other publication types not specified therein (EXCL4).

We screened the identified records (1,472) according to our eligibility criteria. We removed 68 records because they were published earlier than 2016 (INCL3/EXCL3). A total of 1,227 records tackled solely LLMs, solely optimization, or a type of optimization different from CO. Additionally, we excluded 47 records that addressed optimization in the context of LLMs (INCL2/EXCL2). We also removed 7 records because they referred to studies of the wrong type (EXCL4). In total, we removed 1,349 records, resulting in a final count of 123 records.

On January 13, 2025, we performed citation tracking [114] on the 123 records that remained after the initial screening. Specifically, we traced all the citations received by the collected studies referred to in the records, identifying a total of 420 records. We then checked for duplicates in this set, identifying 101 duplicates. Thus, we ended up with 319 distinct records from citation tracking.

We manually addressed each record, evaluating them according to our criteria as we did previously. Among the 319 distinct records, we found that 251 tackled solely LLMs, solely optimization, or a type of optimization different from CO (EXCL2). Additionally, 11 records referred to studies that tackled optimization in the context of LLMs. We excluded 5

records because they referred to studies of the wrong type (EXCL4) and 1 study because it was not written in English (EXCL1). In total, we removed a total of 268 records, leaving us with 51 records. By examining these citations, we uncovered additional highly related or complementary sources w.r.t. the initial set of records, thereby broadening and enriching our literature review.

In summary, combining the records retrieved through databases (123) and those through citation tracking (51), we had a total of 174 records.

17.2.4 Inclusion

We read the full text of each of the 174 studies referred to by the records obtained after the screening activity to decide which ones to include in our systematic review. This involved a more thorough evaluation of each study concerning our eligibility criteria. For instance:

- Amarasinghe et al. [21] proposed to automate the formulation of optimization problems from Natural Language (NL) descriptions, using fine-tuned LLMs to generate modular code for complex real-world business optimization scenarios. In this study, LLMs are used to enhance the formulation of a COP, thereby satisfying our inclusion criteria.

- Pluhacek et al. [392] used an LLM to identify and decompose six well-performing swarm algorithms for continuous optimization. Despite the involvement of LLMs, we exclude this study because it focuses exclusively on continuous optimization problems (EXCL2).

- Wang et al. [490] proposed a survey on the usage of Deep Generative Models (DGMs) for COPs. We exclude this study since DGMs are not LLMs (EXCL2).

Thus, the number of studies included in our review is 103 (i.e., 2 studies were not retrieved and 69 were not included as per inclusion/exclusion criteria, 174 = 103 + 2 + 69).

Table 17.2 summarizes the characteristics of the included studies. Please note that when feasible, the characteristics reported are those of the peer-reviewed version of the studies, i.e., if a study was previously published in a pre-print form and later in a peer-reviewed version, we consider the latter here. The majority (82 studies, 80%) were published in 2024. The remaining studies were published in 2023 (17 studies, 16%) and in 2022 (4 studies, 4%). Considering their identification approach, 70 (68%) were retrieved from databases; additionally, we included 33 (32%) studies referred to by records found through citation tracking. Regarding the publication venues, approximately 50 studies (49%) were disseminated through pre-print distribution services such as arXiv. The remaining studies were peer-reviewed, with 25 (24%) published in journals and 28 (27%) in conference proceedings. Focusing on the type of each study, as specified in our inclusion criteria (INCL4), 84 of them (approximately 82%) are research studies addressing specific research questions and/or experimental settings. Additionally, we included 11 (10%) literature reviews, one (1%) technical report, and 7 (7%) position papers, as these provide perspectives, guidelines, or future directions.

Table 17.2: Overview of the 103 studies included in our review.

Year	Total	Identification		Venue			Publication Type			
		DB	Cit. Track.	Journal	Conf.	Pre-print	Research	Review	Report	Position
2022	4	3	1	0	1	3	4	0	0	0
2023	17	16	1	3	4	10	15	0	1	1
2024	82	51	31	22	23	37	65	11	0	6
Total	103	70	33	25	28	50	84	11	1	7

17.3 Results

We now classify and discuss the 103 studies in terms of optimization process (Section 17.3.1), LLMs (Section 17.3.2), benchmark datasets (Section 17.3.3), and application domain (Section 17.3.4). Finally, the analysis include also the non-experimental literature (Section 17.3.5).

17.3.1 Optimization Process

Overall 86 out of the 103 studies (83%) are concerned with the optimization process. The omitted studies provide perspective and general direction on the topic (Section 17.3.5). In the counts related to the number of studies per step or task, a study addressing multiple tasks within the same step is counted only once for that step. For instance, Li, Zhang, and Mak-Hau [290] addresses both entity recognition and model creation within the problem modeling step. Therefore, it is counted as 1 in the problem modeling step. If a study addresses multiple steps, it is counted as 1 in each step. For instance, Tsouros et al. [467] addresses activities both in the problem modeling and solution method steps, thus it is counted as 1 in each of them. Most of the studies (64 out of 103) focus on activities related to the solution method, representing 63% of the total. The problem modeling task follows closely, with 38 studies, accounting for 37%. A limited number of studies are related to benchmarking (7, 7%) and validation (9, 9%). The optimization process could benefit from the application of LLMs in several ways, as advocated by Wasserkrug et al. [493], who argues for their integration throughout the entire process. While such a holistic approach is possible, only 24 studies (23%) address multiple steps across the pipeline.

Problem Modeling

Problem modeling aims to translate problems, expressed in NL, into mathematical and computational models. This step is foundational, as the quality of the models influence the success of optimization.

A specialized understanding of the domain in which the COP is located is crucial for effective modeling and for ensuring realistic and applicable solutions. Traditionally, human experts are necessary in this phase as they bring deep knowledge of industry-specific rules, regulations, etc. In such context, LLM can help by extracting implicit knowledge from the real-world data and integrating it into solution processes [88, 132, 311, 524] and to generate solutions [119, 130, 245]. Li et al. [295] integrates the domain knowledge, with both entity recognition and the generation of solution code.

In the given problem context, it is crucial to identify the key entities and their interrelations to properly model variables, constraints, and objectives. LLMs can be particularly useful for recognizing and labeling these elements within a NL description of the problem [154, 486]. This capability can enhance the accessibility and usability of the models, enabling non-domain experts to solve significant problems across various industries. Such activity is frequently paired with the extraction of domain knowledge [17, 469], the creation of the model [12, 140, 215, 238, 290, 366, 367, 402], and toward code generation [233, 295]. Note that Obata et al. [367] also addresses solution generation. As mentioned, LLMs can produce optimization models [188, 407] and assist users in doing so, as evidenced, for instance, by the conversational agents developed by Abdullin et al. [1]. The generation of an optimization model is often accompanied by the production of the related code [228, 244, 283, 505]. Several studies have jointly addressed entity recognition, model creation, and code generation [10, 11, 211, 241, 247, 358, 467, 509, 519], with Michailidis, Tsouros, and Guns [343] also addressing solution generation.

A total of 38 studies focused on problem modeling, including the identification of domain knowledge (11 studies), entity recognition (25 studies), and model creation (25 studies).

Solution Methods

Solution methods refer to strategies and algorithms used to find solutions to COPs. The choice of the solution method is linked to the problem model and is deemed critical, as it determines the efficiency, scalability, and accuracy of the results. Thus, optimization researchers' expertise is essential to select the right strategy and to design algorithms and their components. LLMs have been adopted in the design of solution methods, especially through *code generation*; there exists one case where LLMs have been used for algorithm selection [501]. Complete algorithms have been designed starting from NL description of solution methods for Constraint Programming (CP) [21, 479] and Mixed Integer Linear Programming (MILP) [20, 287, 292]. F. Liu et al. [303] and Stein, Vermetten, and Bäck [444] used LLMs to generate heuristic algorithms within an evolutionary framework, with the latter also including in the loop parameter tuning using an automated tool (i.e., `SMAC` [298]). On the contrary, a few studies implemented a LLM-based parameter tuning [198, 282, 335, 472]. Ye et al. [512] and F. Liu et al. [302] developed heuristics for a hyper-heuristic framework. Romera-Paredes et al. [415] introduced FunSearch, a method for discovering new heuristics for the online bin packing problem, achieving improvements over widely used baselines. Following, many other studies tackled the topic of generating code for heuristics [99, 303, 305, 448, 510, 511, 514, 520]. Mao et al. [329] integrated LLMs into mutation and crossover operators for Genetic Algorithms (GAs). Most of the generated code is written in `Python` (30, note that overall 35 studies use `Python`).

Generating solutions that satisfy the constraints and are diverse enough is complex and sometimes even unfeasible [477]. LLMs have been asked to directly *produce solutions* [58, 103, 169, 198, 207, 226, 227, 232, 242, 288, 301, 309, 349, 360, 409, 414, 421, 464, 491, 508, 513]. Such an approach is useful, for instance, in metaheuristics, where complete solutions are required as a starting point or to be mutated during the evolutionary process, and in population-based optimization algorithms, where a large number of (possibly diverse) solutions are required at each iteration. Finally, code generation and solution generation

have been tackled together by 2 works [60, 253].

A total of 64 studies focused on solution methods, encompassing code generation (36 studies), solution generation (28 studies), parameter tuning (4), and algorithm selection (1).

Validation

The *validation* step ensures that the model, the code, and the produced solutions align with the problem requirements and meet end-users' needs. This process often requires substantial human input, as domain experts are typically involved in an iterative feedback loop to critically review the models and solutions to verify that all specifications have been adequately addressed. As LLMs have been proven to perform fairly well in such tasks [424], they have also been applied to optimization. Huang, Shi, and Sukhatme [233] employed an LLM to validate solutions in an end-to-end framework (spanning from problem modeling to technical validation). Conversely, Hao Chen and Li [212] acted at the model level, employing a LLM agent to validate the LP/MILP models. Specifically, they integrated LLMs in diagnosing inconsistent (i.e., over-constrained) MILP optimization models, trying to isolate the minimal subset of linear constraints (including variable bounds) that makes the model instance infeasible. Validation has also been addressed by other studies, reported also in other steps of the optimization process [169, 211, 288, 358, 505, 519]. Note that we do not address bug checks in LLM-generated code as proper technical validation since we assume such checks are inherently part of that activity (if not state otherwise). A total of 9 study focused on validation (5 on solution validation and 6 on model validation).

Models and Algorithms Types

Since the optimization process can be clearly outlined in terms of tasks and steps, it is crucial to recognize that modeling choices, solution methods, and validation procedures are closely interconnected. This section presents an overview of the algorithms and solvers considered in the studies. Please refer to the sections related to the single steps.

Considering the underlying model and algorithm type, the majority of studies (19) focus on Linear Programming (LP) and MILP (18). Other exact methods involve CP (7) and Integer Linear Programming (ILP) (2). Heuristic and metaheuristic algorithms account for 20 studies; among them, 9 focus on heuristics, 3 focuses on Hyper-Heuristics (HHs), 3 on GAs, and 3 on Evolutionary Algorithms (EAs). Furthermore, 3 study explores the capabilities of LLM w.r.t. multi-objective CO [301]. One study deals with SAT, even though LLMs are used to generate heuristics for their solution [448].

One reason for the prominence of LP in these studies is the Natural Language for Optimization Modeling (NL4Opt) competition. The primary objective of this competition was to assess whether models could generalize to unseen problem domains, with an emphasis on two sub-tasks: named entity recognition for LP components and LP model generation [403] (see also Section 17.3.3 for a description of the related dataset).

Benchmarking

CO benchmarking involves evaluating and comparing the performance of algorithms and techniques, so to determine how effective a solution method is and, potentially, understand

the reasons behind its performance. Conventional comparison methods involve gathering numerical data from algorithm runs and reporting them in the form of tables and boxplots, alongside considerations of computational times and statistical tests (such as the Friedman test) to account for stochasticity. For decades, researchers have emphasized the need for additional tools to better understand the behavior of optimization algorithms [56]. One such tool is related to Search Trajectory Networks (STNs), a graph-based tool for the visualization of metaheuristic behavior [370]. While these visualizations are very informative, they require prior knowledge to be produced (e.g., parameter setting) and interpreted (e.g., which algorithm is superior). To bridge this gap, Chacón Sartori, Blum, and Ochoa [91] enriched STNs with LLM-generated explanations and suggestions.

The visual investigation of solutions is enabled by Da et al. [119], who also allows solution validation. The visual capabilities of LLM are also explored by Elhenawy et al. [169], who directly employ them to solve problems, presenting instances in visual form.

A relevant topic in CO is explainability of results as a way for enhancing confidence and trust in the solutions. Interesting questions include identifying which solution components most significantly influence the final result, among others. Answering to these questions requires in-depth knowledge of the application domain, the solution method, and the specific instance being addressed. To reduce this human-intensive task, Kikuta et al. [254] proposed a post hoc LLM-based explanation framework aimed at clarifying the decision-making process for Vehicle Routing Problems (VRPs). Similarly, also other study included an explainability level [226, 414].

A total of 7 studies focused on benchmarking (3 on visual analysis and 4 on explainability).

17.3.2 Large Language Models

The 70 LLMs used in the 103 studies are based on 22 different architectures. Despite the diverse capabilities of the architectures employed, most LLMs have been used in CO to enhance problem modeling and solution methods. In the following, we provide a description of how LLMs are used within the included studies, considering their architectures, general tasks, and related activities. Note that the numbers for activities may not sum to those at the main task level, as a given LLM can be used for multiple purposes.

Architectures

13 out of the 22 architectures focus on generative approaches. Introduced in 2022, `GPT-3.5` [62] is particularly effective for tasks requiring fluent text generation. `GLM` [164] emerged as a balanced approach to generative tasks, leveraging both bidirectional and autoregressive capabilities.

During 2023, several architectures were released: `LLaMa 2` [193], optimized for efficiency and suitable for tasks involving scalability and quick inference times; `PaLM 2` [202], which performs strongly in scenarios requiring comprehension and contextual text generation; `Mistral` [239], designed for efficient inference and competitive performance in text generation, making it suitable for resource-constrained deployments; and `Qwen` [398], focusing on general-purpose text generation with multilingual support and fine-tuned reasoning capabilities. In the same year, `GPT-4` [374] appeared as a particularly strong model for fluent text generation, alongside `LLaMa 3` [312], which is optimized for instruction-following and

structured text understanding. DeepSeek [134] also adopts generative approaches with a particular focus on the Chinese language, while Claude 3.5 [24] excels in dialogue-based applications and knowledge-intensive tasks. Qwen2 [399] extends its predecessor with multilingual capabilities and refined reasoning. Cohere's Command-R+ [13] is optimized for Retrieval-Augmented Generation (RAG), improving factual accuracy and knowledge recall. Finally, Yi [461] is tailored for diverse text generation tasks, integrating efficient training strategies for improved performance.

A total of 62 studies rely on these architectures, namely 24 on GPT-3.5 (2022), 2 on GLM (2022), 9 on LLaMa 2 (2023), 3 on PaLM 2 (2023), 2 on Mistral (2023), 3 on Qwen (2023), 43 on GPT-4 (2023), 7 on LLaMa 3 (2024), 6 on DeepSeek (2024), 7 on Claude 3.5 (2024), 1 on Qwen2 (2024), 1 on Cohere (2024), and 1 on Yi (2024).

Textual input is handled by 5 architectures. T5 [401] (2019) treats all text-based tasks as text-to-text problems, offering significant flexibility across a range of natural language processing applications. BART [286] (2019) is particularly effective at transforming textual input into structured outputs, making it suitable for complex textual tasks. UnixCoder [206] (2022) is an architecture specialized in programming-related tasks, such as code generation, code summarization, and code translation. Additionally, LLaMa 2 [193] (2023) and its successor, LLaMa 3 [312] (2024) provide LLMs optimized for instruction-following (e.g., LLaMa 3-70B-Instruct) and are fine-tuned for tasks requiring structured text understanding. A total of 9 studies rely on these architectures: 3 on T5 (2019), 3 on BART (2019), 1 on UnixCoder (2022), 1 on LLaMa 2 (2023), and 1 on LLaMa 3 (2024).

4 architectures focus on multimodal data. All these architectures (or their multimodal models) were introduced in 2024. Gemini emphasizes both multimodal capabilities and generative approaches, making it ideal for tasks that require integrating and generating text, images, and possibly other forms of data. GPT-4-Vision-Preview extends the GPT-4 [374] architecture by incorporating visual input processing, allowing it to handle tasks involving image understanding and text-image reasoning. Mixtral [240] leverages an ensemble of models designed to handle multimodal tasks, such as text-to-image and image-to-text conversions. OpenCoder [229] is optimized for code-related tasks and supports multimodal inputs, particularly focusing on enhancing software development workflows with a combination of textual and structural data processing. A total of 9 studies rely on these architectures: 5 on Gemini, 2 on Mixtral, 1 on GPT-4, and 1 on OpenCoder.

Some of the LLMs analyzed in the selected studies have been deprecated or replaced by newer versions. OpenAI's Text-Davinci-003 and Text-Davinci-Edit-001, based on GPT-3.5, have been phased out in favor of GPT-4-Turbo and GPT-4o. Google rebranded Bard as Gemini, transitioning from PaLM 2 to the Gemini 1.x and 2.x series. Meta's LLaMa 2-13B has been largely replaced by LLaMa 3, while OpenAI's GPT-4 evolved into GPT-4o, incorporating multimodal capabilities. Similarly, Mixtral-8×7B-Instruct-v0.1, Qwen, and DeepSeek have advanced to Mixtral, Qwen2, and DeepSeek-V2, respectively. Anthropic's Claude 3.5 is also expected to be succeeded by newer iterations.

Given the varying performance of LLMs across architectures, access to their source code remains a key issue to be analyzed. Most high-performing models, including OpenAI's GPT-3.5 and GPT-4, Google's PaLM 2 and Gemini, as well as Anthropic's Claude 3.5 and Cohere's Command-R+, are closed source and require paid access. Mixtral, while open-weight, is commercialized via paid APIs, and DeepSeek V2 adopts a more restrictive

licensing model. In contrast, Meta's LLaMa 2 and LLaMa 3 are open source under specific licenses, while Mistral AI offers both open-weight and commercial models. Qwen and Qwen2 provide open-source variants alongside proprietary versions. Fully open-source models include BART, T5, and certain versions of LLaMa, Mistral, StarCoder, and OpenCoder (the latter two specialized for code generation). Given the evolving nature of access policies and licensing conditions, these constraints may influence model selection in research.

We count separately both specific implementations designed for conversational purposes and versions of the same model updated to a specific timestamp, such as GPT-4-0613 (referring to a version of GPT-4 released or fine-tuned on June 13, 2023). This approach aims to enhance clarity and improve reproducibility. However, there is no guarantee of consistent responses even when using different versions of the same models [184, 261], especially since chat completions are not deterministic by default.[4]

When using LLMs in a research setting, several key aspects must be addressed to ensure both reliable and reproducible results. Documenting the exact prompts used is crucial, as LLMs are highly sensitive to prompt variations, which can lead to significantly different outcomes [22, 51]. Additionally, detailed reporting on the model version, configuration, and any fine-tuning is essential for accuracy and replicability. Notably, a limited number of studies (7) that conducted experiments in this area did not provide any details about the LLM employed [140, 311, 343, 360, 367, 407, 472]. Understanding the training data used in an LLM can help identify potential data leakage, biases, or knowledge gaps that might affect results [34]. Moreover, documenting any additional steps, such as data preprocessing or post-processing of model outputs, is critical for ensuring that findings can be replicated across different studies and datasets.

Finally, when evaluating LLMs for COPs, there is no universally adopted metric. Accuracy is common, but its meaning depends on the task: sometimes a custom "score" is defined, while other metrics are domain-specific. Most studies instead assess LLMs via the objective values of their solutions. As noted in Chapter 2, each problem has its own objective function (e.g., minimizing travel distance in Traveling Salesperson Problem (TSP), or tardiness in Permutation Flowshop Scheduling Problem (PFSP)). Since the optimum is not always known, solution quality is often judged relative to the best-known result, using the optimality gap (see Equation 2.1).

Problem Modeling

A total of 40 LLMs out of 70 (57%) have been used for problem modeling. Among them, 19 LLMs have focused on enhancing *model creation*.

A common approach involves creating models from unstructured NL. Some of these approaches address optimization problems more broadly, using GPT-3.5-Turbo (an optimized variant of GPT-3.5 for faster performance and lower cost [505]), GPT-3.5-turbo-0613, GPT-4-0613, and GPT-4 itself [244], as well as LLaMa 2-7b, a version of LLaMa 2 with 7 billion parameters [12], and T5-Base, the base model built on the T5 architecture [215]. Additional optimized or specialized variants, including GPT-4o, an improved multimodal version of GPT-4 [10], Qwen1.5-14B, DeepSeek-V2 [509], Mistral-7B, DeepSeek-Math-7B,

[4] https://platform.openai.com/docs/advanced-usage/reproducible-outputs. Accessed 2024–08–26.

LLaMa 3-8B, and Qwen2.5-7B [228], have also been used for broad optimization tasks, leveraging improved reasoning capabilities and efficiency.

Instruction-tuned models such as Code Llama-Instruct and Zephyr-7b-beta, a 7B-parameter model designed for instruction-following, have been used for structured task execution [358]. Furthermore, both GPT-4o and Claude 3.5 Sonnet have been employed in a zero-shot fashion to generate problem formulations directly from user input, thus reducing the need for extensive prompts or examples [211]. Other approaches focus on specific formulations. For instance, LLMs have been used to automate the generation of LP models with ChatGPT-3.5 [290], GPT-4 [1], and the BART architecture, including both BART-Base [188, 402] and BART-Large variants [188, 238]. Furthermore, ChatGPT-3.5 has also been employed in MILP problems [21], as have Bard, a conversational model built on the PaLM-2 architecture [290], and LLaMa 3-70B [247]. Instruction-tuned Code Llama-Instruct and Zephyr-7b-beta have likewise been applied to Quadratic Programming (QP) tasks by translating user directives into valid constraints and objective functions.

Concerning *entity recognition*, 22 LLMs have been used to identify and extract specific elements from model representations. Specifically, GPT-4 and Text-Davinci-003, based on GPT-3.5 and optimized for a wide range of tasks, have been used to translate user input into constraints that the underlying CP model can process [283]. This approach has also been applied to LP and MILP problems using GPT-3.5 [11, 519], GPT-4 [519], ChatGPT-4 [17], LLaMa 3-70B [10, 247, 358], ChatGPT-4o [10, 211, 241], and Claude 3.5 Sonnet [211]. Meanwhile, CodeLlama-Instruct and Zephyr-7b-beta have also been used for QP problems. ChatGPT-3.5 and Bard have been used exclusively for MILP [290], while T5-Base has been employed for LP [215]. LLMs have been applied to general problems, GPT-3.5-Turbo-0163, GPT-4, LLaMa 2-7b, LLaMa 2-13b, and LLaMa 3-70B [12, 119], GPT-4o-mini [295], BART-Base, BART-Large [238, 402], and Gemini 1.0 Pro [233].

There are 12 studies dealing with extracting *knowledge*.GPT-4 and GPT-4-0163 have been used to extract general domain knowledge in financial planning [132, 247]. GPT-4o, Claude 3.5 Opus, Command-R+, and Mixtral-8×2B have been applied to a network problem [88], and GPT-4o has also been used in robotics [349]. LLaMa2-7B, LLaMa2-13B, GPT-3.5, and GPT-4 have been employed to model knowledge for the VRP problem [119]. Moreover, ChatGPT-4, has been used to model domain knowledge as knowledge graphs [469].

Solution Methods

LLMs has been employed in 60 out of 70 (85%) works for what concerns the solution method task. 23 LLMs have been used for *solution generation*. One approach involves creating candidate solutions for well-known COPs, such as a specific class of the VRP, by leveraging both GPT-based models like GPT-4 [11], GPT-4-Turbo, and GPT-4o [253], and LLaMa-based or T5-based models, including LLaMa 2 and T5-Base [103]. Another approach uses LLMs to combine problem descriptions and previously generated solutions within a meta-prompt processed by ChatGPT-3.5-Turbo, PaLM 2-L, PaLM 2-L-IT, and text-bison [508].

The multimodal capabilities of GPT-4-Vision-Preview and ChatGPT-4o have also been employed to generate solutions through visual prompts [169, 232]. Moreover, ChatGPT-4 has been adopted for finance-related solutions [414] or automated sequence planning in

robot-based assembly [513]. Other LLMs used for domain-specific problems include GPT-4 for travel planning [130], program scheduling [245], and traffic simulation [119]. LLaMa 2-7b and LLaMa 2-13b have also been applied to traffic simulation. Claude 3.5 Sonnet has been used in molecular biology [409], and ChatGPT-4-Turbo together with ChatGPT-4o-mini has been used to coordinate computation in a graph reasoning scenario [226]. GPT-3.5-Turbo has found utility in industry-related problems [505], while GPT-4-Turbo, GPT-4o, Gemini 1.5 (Pro and Flash), and Gemma 2 27B have been employed in planning tasks [58, 349]. Furthermore, in the context of GAs, GPT-3.5-Turbo-0613 has been used to select parent solutions from the current population [309] and to perform mutation and crossover [329].

Additionally, 35 LLMs leverage approaches for *code generation*. GPT-3.5, GPT-4, GPT-4o, and Qwen (LoRA Fine-Tuned), which is a version of Qwen fine-tuned using LoRA [224], have been used to generate code specifically for LP, Mixed Integer Programming (MIP), and MILP approaches to COPs [11, 244, 292, 519], whereas CodeLlama-Instruct and Zephyr-7b-beta have been applied to both MILP and QP [358]. Various methods for automating the generation of heuristic algorithms rely on a range of LLMs, each optimized for different aspects of code generation. For example, CodeLlama, a code-oriented variant of LLaMa 2 [520], and StarCoder, trained on extensive code-related datasets, are fine-tuned for specific programming tasks, while DeepSeek-LLM-7B-Base, GPT-3.5-Turbo, and GPT-4-Turbo are optimized for speed and efficiency [247, 301–303, 305, 415, 510, 512, 514]. GPT-3.5-Turbo-0613 has likewise been employed for generating crossover and mutation implementations in Python [329] within EA contexts. Multimodal models, such as GPT-4o, and advanced reasoning models like Claude 3.5 Opus and Claude 3.5 Haiku, have also been utilized in heuristic generation tasks, and DeepSeek-Coder along with its updated DeepSeek-Coder-V2 focus on high-performance code synthesis. Similarly, GLM-3-Turbo, OpenCoder-8B-Instruct, and Yi-34b-Chat provide structured, optimized code generation [305, 511]. For large-scale problem-solving, models such as Gemini 1.0 Pro and Codey, both built on PaLM 2, have shown strong performance in complex coding scenarios, while the latest iterations of foundation models (e.g., LLaMa 3-70B, LLaMa3-70B-Instruct, LLaMa 3.1-8B, GPT-3.5-Turbo, GPT-4o-Mini, and Qwen-Turbo) have been refined for specialized programming applications [305, 511, 520]. Other approaches harness a self-reflection mechanism to directly generate executable Python code from natural language, exemplified by GPT-4 and Gemini 1.0 Pro [233]. A comprehensive code-generating framework for business optimization has also been developed using CodeT5-finetuned_CodeRL [284], a model that employs reinforcement learning on top of CodeT5 [21]. Additionally, Text-Davinci-Edit-001 has been employed to generate MiniZinc-specific representations [20], while Text-Davinci-003 and GPT-4 have produced Gurobi-based code for supply chain optimization [287]. GPT-3.5-Turbo, Mistral-7B, DeepSeek-Math-7B, LLaMA 3-8B, and Qwen2.5-7B have been broadly used for industry-related problems [228, 505], and a specialized version called GPT-4o-2024-08-06 has been introduced to optimize code for SAT solvers [448]. Moreover, GPT-4o and Claude 3.5 Sonnet have addressed supply chain and robot logistics tasks [211], and ChatGPT-4o-mini has been applied to multi-agent solution design [295].

Moving to *parameter tuning*, we identified 5 applications of LLMs. ChatGPT-4o has been adopted to model user preferences by adjusting an optimizer's weights [198], whereas GPT-3.5, GPT-4, Gemini, and Le Chat (the chatbot interface for Mistral models) have been

used to set metaheuristics parameters [335].

A single application aimed at enhancing *algorithm selection*: `UnixCoder`, a code representation model designed for programming tasks, has been used to extract features linked to the underlying optimization algorithms [501].

Validation

We found 10 applications of LLMs out of 70 (14%), in the context of validation for optimization models. In particular, 7 LLMs have been involved in *solution validation*. `GPT-4` and `Gemini 1.0 Pro` have been used to validate solutions generated for the VRP [233], while `GPT-3.5`, `LLaMa 2-7b`, `LLaMa 2-13b`, and `ChatGPT-4o` have been applied to broader validation tasks [119, 169]. Additionally, `Qwen (LoRA Fine-Tuned)` has been leveraged for solution validation [519].

As for *model validation*, 9 LLMs have played a role. `GPT-4` has been employed to diagnose infeasible MILP models through interactive sessions [212], while `GPT-3.5-Turbo` has been used to validate models built from natural language [505]. Similarly, `GPT-3.5`, `GPT-4`, and `Qwen (LoRA Fine-Tuned)` have also been applied in this context [212, 519], with `GPT-4o` and `Claude 3.5 Sonnet` reported for model validation tasks [211]. In addition, code-focused models have been utilized to ensure correctness: `CodeLlama-Instruct` and `Zephyr-7b-beta` have been adopted to verify that the generated code remains consistent with the original model formulations [358].

Benchmarking

As for benchmarking, 9 out of 70 (14%) LLMs have been used to enhance it. Specifically, 8 LLMs use *visual analysis*, as they are integrated, for instance, with a tool designed to visualize the behavior of various algorithms applied to specific instances of a COP [91]. The models used by Chacón Sartori, Blum, and Ochoa [91] include `Mixtral-8×7B-Instruct-v0.1`, which is optimized for instructional and guided tasks, `GPT-4-Turbo`, and `Tulu-v2-dpo-7b`, a fine-tuned version of `LLaMa 2` that was trained on a mix of publicly available, synthetic, and human-generated datasets using a parametrization of the RLHF algorithm known as Direct Preference Optimization [400]. Furthermore, `GPT-3.5`, `GPT-4`, `LLaMa 2-7b`, and `LLaMa 2-13b` have been employed to visualize solutions for a domain-specific problem [119], while `ChatGPT-4o` has been used to interpret visual inputs in lieu of purely numerical data [169]. We identified 5 applications aimed at improving *explainability*. `ChatGPT-4` has been employed to describe the decision-making process for the VRP [254], while `ChatGPT-4-Turbo`, `ChatGPT-4o`, `ChatGPT-4o-mini`, and `Claude 3.5 Sonnet` have been used for instances of both the TSP and mTSP [119, 198, 226, 409].

Supporting Platforms

While reviewing the studies considered for inclusion, we came across two platforms used by Chacón Sartori, Blum, and Ochoa [91] to support the design of their approach. Although these platforms are general-purpose and facilitate the use of LLMs broadly rather than specifically in the context of CO, we believe it is beneficial to describe them briefly.

Chatbot Arena [102] is an open platform for evaluating LLMs based on human preferences. It offers access to over twenty LLMs, including both proprietary and open-source options,

and provides a leaderboard to compare results. Chat2Vis [324], on the other hand, focuses on generating visualizations directly from natural language text. It uses various LLMs and shows that a set of proposed prompts offers a reliable approach to rendering visualizations from natural language queries, even when they are highly misspecified or underspecified.

17.3.3 Benchmark Datasets

The methodologies proposed in the studies have been evaluated using well-known CO instances, related problem suits (e.g., MIPLIB[5] and ASLIB),[6] or datasets specifically developed for the case of LLMs. Considering this latter point, 29 studies assess the efficacy and efficiency of leveraging LLM within CO with specific benchmark datasets developed for this purpose.

The most frequently used benchmark dataset is the LPWP, also referred to as the NL4Opt dataset, which has been employed in 16 studies [1, 11, 12, 154, 188, 215, 228, 238, 241, 290, 343, 366, 402, 486, 505, 519]. Initially introduced by Ramamonjison et al. [402] and later expanded by Li, Zhang, and Mak-Hau [290], this dataset has been used in the NL4Opt competition. The original dataset includes 4,216 NL problem declarations derived from 1,101 LP problems across six different domains. The extension enhances the dataset by introducing new problem descriptions and constraint types, such as logic constraints and binary variables. The second most used benchmark dataset is the ComplexOR dataset [505], which has been employed in 3 studies [11, 241, 505] and has been introduced to complement the NL4Opt dataset. The dataset consists of NL descriptions of 37 problems across different domains, sourced from academic studies and real-world scenarios, encompassing 25 LP formulations and 12 MILP formulations.As ComplexOR also the NLP4LP dataset [10, 11] has been used in 3 studies [10, 11, 241]. It encompasses NL descriptions of 67 problems across various domains and including both LP (54) and MILP (13) formulations. This same dataset has been enriched to up to more than 200 optimization problems by AhmadiTeshnizi et al. [10]. C. Huang et al. [228] proposed OR-Instruct, a pipeline for creating synthetic data for optimization data. To test the capabilities of such a pipeline, they created IndustryOR, which has been used in 2 studies [228, 241]. As the name suggests, it focuses on industrial problems (a total of 100 real-world problems from eight industries) covering LP, ILP, MILP, and non-linear programming. X. Huang et al. [231] introduced Mamo, a dataset for LP modeling. It includes 652 easy and 211 complex LP problems, each paired with its corresponding optimal solution, sourced from various academic materials. Mamo has been used in 2 studies [228, 241]. Luo et al. [321] introduced GraphInstruct, a dataset comprising 21 reasoning problems on the topic of graphs and networks, including COPs. GraphInstruct has been used in 2 studies [226, 295]. Similar to this dataset, there is Talk Like A Graph [172], LLM4DyG [523], GraphViz [97], NLGraph [484], and GNN-AutoGL [295] (note that these datasets have been used only by one study [295]).

The remaining datasets have been used exclusively in one out of the retrieved studies, many times this being the study proposing the dataset on the first place. Amarasinghe et al. [21] introduced the AI-copilot-data dataset, which includes 100 NL descriptions of problems within the production scheduling domain. Huang, Shi, and Sukhatme [233] introduced the

[5] https://miplib.zib.de/

[6] https://www.coseal.net/aslib/

homonym dataset,[7] which comprises 80 NL descriptions of routing problems, including variants for single-robot and multi-robot routing. Almonacid [20] introduced a homonym dataset, which comprises 10 NL instructions for the creation of MiniZinc models involving discrete variables and arrays of discrete variables, thus ILP problems. Hao Chen and Li [212] introduced the OptiChat dataset, which comprises 63 infeasible (i.e., inconsistent) MILP model formulations across various domains. This dataset was derived from feasible models expressed through the Python library Pyomo [214] and sourced from a collection of libraries and textbooks. The formulations were created by modifying one or more model parameters (e.g., minimum inventory, demand, maximum capacity) or adding constraints (e.g., maximum cost, minimum demand for a particular product) so to make the instances infeasible (i.e., their set of feasible solutions is empty). Lawless et al. [283] introduced two datasets, Safeguard and Code Generation. They both are strictly connected to the framework the authors proposed. The Safeguard dataset is a binary classification dataset designed to evaluate whether the system at hand (i.e., a system that integrates CP and LLMs to schedule meetings in a company) has sufficient data sources (e.g., information related to the meeting attendees) to handle given NL constraints. The Code Generation dataset contains a collection of NL constraints that can be translated into Python code using the data structures available in the system. An example of such constraints is "The team has a no-meeting policy on {WEEKDAY}". This dataset aims to test the capability of generating executable Python code that satisfies the specified constraints. While employing NL4Opt and other existing dataset from the Machine Learning (ML) community, Michailidis, Tsouros, and Guns [343] also introduced a homonym CP-based benchmark. The dataset was built using 18 CP problems from a university-level CP modelling course. Z. Yang et al. [509] introduced OptiBench and ReSocratic-29k. OptiBench includes 816 real-world optimization problems spanning multiple domains, focusing on linear and mixed-integer programming and introducing a graph-based evaluation method to assess model correctness. ReSocratic-29k consists of 29,000 optimization problem demonstrations, generated through a reverse synthesis approach that first constructs step-by-step formulations before back-translating them into NL questions. These datasets provide a targeted benchmark for assessing and improving LLMs in formulating and solving optimization tasks. Borazjanizadeh et al. [60] introduced SearchBench, a dataset encompassing five problem categories and 1,107 instances, primarily focused on puzzles and general combinatorics. Each problem type corresponds to a well-known COP, but the constraints have been slightly modified to reduce the likelihood that LLMs encountered identical problems during their training phase. Mostajabdaveh et al. [358] presented a new dataset to complement the existing NL4Opt and ComplexOR benchmarks, aiming to provide less structured input and more complex optimization scenarios. This benchmark includes problems related to LP, MILP, and QP. Unlike NL4Opt, which expects a formal model as output, and ComplexOR, which requires Python code, this dataset outputs solutions in Zimple code. Similarly, J. Zhang et al. [519] enriched the NL4Opt dataset with English and Chinese problem descriptions. D. Ju et al. [247] introduced a dataset of 173,700 training and 21,800 test samples related to travel planning.

Finally, a different dataset is ORQUA [357]. It is designed to evaluate the extent of CO

[7] We use the term "homonym" when the authors did not provide a specific name for the dataset.

knowledge in LLMs. Given a problem description, the dataset assesses the model's ability to understand and identify, for example, the appropriate type of mathematical modeling the problem corresponds to.

Table 17.3: Examples of mathematical reasoning tasks from various benchmark datasets.

Dataset	Task	Example
GSM8K [112]	Arithmetic Word Problem	Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
MultiArith [223]	Multi-Step Arithmetic	John has 3 apples. He buys 5 more apples and then eats 2. How many apples does he have left?
AquA [299]	Probability	From a pack of 52 cards, two cards are drawn together at random. What is the probability of both the cards being kings?
BIG-Bench [440]	Logical Reasoning	If a snail climbs a 10-meter pole at a rate of 2 meters per day but slides back 1 meter each night, how many days will it take to reach the top?
BIG-Bench Hard [449]	Combinatorial Reasoning	A school has 6 different clubs. Each student must join exactly 2 clubs. How many unique pairs of clubs can be formed?

Similar to the platforms discussed in Section 17.3.2, we also identified general-purpose datasets in the reviewed studies. These datasets are valuable for developing LLMs-based approaches to optimization, particularly by providing math-related problems expressed in unstructured natural language which are useful especially for problem modeling (e.g., handling addition, multiplication, and data structures like sets). These resources can support researchers and practitioners in designing new applications of LLMs in CO. We briefly outline their characteristics and report some examples in Table 17.3. Notably, all of these resources were used by Ahmed and Choudhury [12], while GSM8K was also employed by C. Yang et al. [508].

GSM8K [112] consists of 8,500 linguistically diverse grade-school human-generated math word problems. Unlike simpler arithmetic datasets like AddSub [223] (395 problems) and SingleOp [264] (562 problems), GSM8K requires multi-step reasoning. Its emphasis on stepwise numerical reasoning aligns with CO, where problems often require sequential computations and recursive decision-making.

MultiArith [223] consists of around 600 multi-step arithmetic problems that require sequential operations (e.g., addition, subtraction, and multiplication). It was partially generated from existing math problems and structured for ML applications. Its focus on structured numerical reasoning makes it relevant to CO techniques such as branch-and-bound and constraint satisfaction, which rely on constructive decision-making.

AquA [299] presents 100,000 human- and machine-generated algebraic word problems in a multiple-choice format along with rationales explaining each answer. It requires the computation of correct answers and their selection from distractors. Many COPs involve symbolic reasoning and equation solving, making AquA useful for assessing a model's ability to handle algebraic structures and optimization constraints.

BIG-Bench [440] is a large-scale benchmark evaluating language models across more than 200 tasks, covering mathematics, reasoning, linguistics, commonsense understanding, and code generation. The dataset was created through a combination of human-designed tasks and algorithmically generated problem sets, allowing for a broad evaluation of logical reasoning and heuristic problem-solving. This makes it valuable for studying how models approach optimization-based decision-making. A more challenging subset, BIG-Bench

Hard [449], focuses on 23 particularly difficult tasks from BIG-Bench that remain unsolved or challenging for state-of-the-art models. Many of these challenges relate to CO, where finding optimal solutions in complex search spaces is a key difficulty.

17.3.4 Application Domains

CO has been applied to a wide range of problems across various domains, and over the years, these problems have been formalized into well-known prototypical models, such as the TSP, the VRP, and the Capacited Vehicle Routing Problem (CVRP) in the field of routing. Among the 103 studies, 64 explicitly address problems within specific domains or problem formulations, which we overview in this section. In this analysis, literature reviews that provide information on the use of LLM without offering actual implementations are excluded, such as those by Fan et al. [171], Wu et al. [500], and Zhao et al. [524]. Exceptions apply to reviews like the one by Saka et al. [421] that also verified the application of LLMs in CO within the construction domain.

The majority of these studies (26) focus on routing problems, particularly within the contexts of the TSP [99, 169, 227, 253, 301–303, 305, 311, 335, 444, 447, 508, 510, 512, 520], VRP [103, 119, 198, 232, 233, 253, 254, 305, 502, 512], orienteering [512], and traveling [130, 247, 288]. While the first three COPs are well-known in the CO field, the latter refers to the modeling of the traveling activity (i.e., visiting a new city) as a COP. It cannot be considered as a TSP variant in all cases, as it is not given that the goal is to find a Hamiltonian path. Two other domains that have garnered significant attention are scheduling/planning (14 studies) and networks and graphs (10). In the first domain, studies have focused on variants of well-known benchmark problems, like the PFSP [21, 302] and planning [58, 140, 211, 242, 349, 367, 380, 491, 513], as well as specific scheduling problems, like server [511] and meeting/conference [245, 283] scheduling. Also for what concerns the Network and Graph domains, studies focused on general network design [226, 295, 314, 469, 514], but also on classical problems like coloring [335], social networks Chacón Sartori et al. [88], critical node identification [329], and path finding [60, 253].

The remaining studies focused on packing (9), combinatorics (4), engineering (3), finance (3), bioinformatics (3), supply chain (1), and strings (1).

A limited number of studies (9) explore multiple problem domains and formulations to demonstrate the capabilities of LLMs across various contexts [60, 99, 302, 335, 415, 444, 510, 512, 520]. Additionally, Khan and Hamad [253] tackled a diverse set of problems, which can all be reduced to graph-based ones, while studies like the one by Lawless et al. [282] inherently address multiple problem domains as they are based on library of problems (initially not thought for applications in the field of LLMs), like the MIPLIB dataset.

17.3.5 Positions from Non-experimental Literature

Among the 103 selected studies, 11 are literature reviews, 7 are position papers, and 1 is a technical report.

Fan et al. [171], Wu et al. [500], S. Huang et al. [230], Lai et al. [277], Cai et al. [72], and F. Liu et al. [304] presented literature reviews on the topic of LLM and optimization or on strictly related fields, such as algorithm design. Saka et al. [421], Zhao et al. [524], Wu et al. [502], Pallagani et al. [380], Sui et al. [447], and Long et al. [314] presented literature

reviews that deal with CO in specific application domains (i.e., engineering, planning, VRP, planning and scheduling, TSP, and network optimization, respectively) and in some cases involve (preliminary) studies and experiments with LLM.

Wasserkrug et al. [493] put forward a position paper advocating for the usage of an LLM within optimization considering all the optimization steps. Similarly, Tsouros et al. [467] proposed a LLM-aided optimization pipeline in the context of CP modeling. Freuder [185] highlighted the potential of LLMs in facilitating the discussion between optimization and domain experts. Srivastava and Pallagani [441], Yu and Liu [515], and Ustyugov [472] presented insights on the usage of LLMs within EAs, planning-like tasks, and automated MILP configuration.

The only technical report identified is by Ramamonjison et al. [402]. The report includes insights and comparisons related to the NL4Opt challenge.

17.4 Concluding Remarks

In this concluding section, we distill the content of the chapter (Section 17.4.1), discuss limitations (Section 17.4.2), and outline future directions for extending both this systematic literature review and current practice in the field (Section 17.4.3).

17.4.1 Content

In this systematic review, we examined the application of LLMs to CO. Our study summarizes the literature leveraging the PRISMA 2020 guidelines. To our knowledge, this is the first attempt to comprehensively study the application of LLMs to CO and COP.

Out of over 2,000 collected publications, we included 103 studies in our analysis. These studies were classified based on the task LLMs performed within the optimization process, the implementation details of the LLMs, the most commonly used datasets, and the application domains where LLMs have been employed in CO thus far. Additionally, we highlighted the caveats associated with using LLMs in the field of optimization, outlined future research directions, and exposed the limitations of the present review.

17.4.2 Limitations

While our systematic review offers valuable insights into the use of LLMs in CO, it has some limitations. First, research on LLMs and their application to optimization is advancing rapidly, with new studies emerging continuously. Consequently, while this systematic review attempts to be as comprehensive as possible, it inevitably cannot cover all existing studies. Also, systematic reviews depend on data gathered from other studies, meaning the quality and bias level of the evidence in a systematic review are directly tied to those of the data sources [163]. Therefore, we acknowledge that following PRISMA guidelines might have led to the inclusion of too many studies that report positive outcomes or successful applications of LLM over those that do not, potentially overstating the effectiveness or applicability of LLM in this field due to inherent selection bias.

Then, our review focuses solely on how LLMs can be applied to CO. An equally significant aspect not covered in this study is how CO techniques can be employed to enhance LLMs, which is dual to our main focus. By not addressing this aspect, which can be pursued

in future work, our review may miss relationships and interdependencies between the two research fields. Examples of such works include Pan et al. [382], who experiment with using Metaheuristics (MHs) to engineer LLM prompts. More specifically, they use, among other methods, Hill Climbing (HC) and Simulated Annealing (SA), to discover and learn new efficient prompts. Another example is the work by Singla, Singh, and Kukreja [433], who model the trade-off between response quality and inference cost of LLM as a bi-objective combinatorial optimization problem. Additionally, this study focuses on CO, disregarding other types of optimizations, such as continuous optimization. Examples of such works are those by Pluhacek et al. [392, 393].

We intentionally included a significant number of pre-prints alongside peer-reviewed publications (Section 17.2.3). Such a decision is driven by the fast-paced nature of research in the field of LLMs, where discoveries and advancements are rapidly shared through pre-prints before formal publication (see, for example, Devlin et al. [139]). Pre-prints allow for timely access to the latest research, innovations, and discussions and are widely used within the NLP community. We take no position on the content of pre-prints found on platforms like arXiv; instead, we include these documents to maximize the recall of our systematic review. Eventually, our systematic review includes studies based on closed-source LLMs, like ChatGPT. The proprietary nature of these systems often restricts access to their full methodologies and inner workings, which creates significant challenges for understanding and replicating the reported findings. Despite these downsides, we include works dealing both with open- and closed-source LLMs to maximize the recall of our systematic review.

17.4.3 Future Work

In this section, we first outline possible future work on the topic of LLMs for CO and then on this literature review.

On Large Language Models for Combinatorial Optimization

While various aspects of the optimization process have been addressed in the existing literature, certain areas still call for further exploration:

- **Metaheuristics**: The adoption of LLMs in MH frameworks has been limited and scattered. Future research could investigate how LLMs can be used to dynamically adjust MH strategies, optimizing parameters (as anticipated in some studies [198, 282, 335]) or switching strategies based on the current state of the search. This could enhance the flexibility and effectiveness of MHs. Future studies could explore how LLMs might expand local search neighborhoods by suggesting structures or transition operators.

- **Algorithm Selection**: Utilizing LLMs to explore the search space of algorithm selection might be winning (as anticipated by Wu et al. [501]) and LLMs could help pinpoint scenarios where current solvers are less effective. Additionally, LLMs could be used to generate instances specifically designed to test the strengths and weaknesses of these solvers, leading to improved performance insights.

- **Synthetic Instance Generation**: LLMs could be leveraged to create synthetic instances that replicate the complexities of real-world problems or that are specifically crafted to challenge existing algorithms. This approach has been anticipated in the IndustryOR dataset [228].
- **LLMs behavior w.r.t. NLP problem description**: While LLM capabilities w.r.t. visual representations in CO has been addressed [169], it remains unclear to what extent LLMs adjust their responses depending on the input style thus representing a promising research area.
- **Evaluation Protocol**: Evaluating the performance of LLMs on COPs remains challenging due to several factors, e.g., problems can have multiple representations, various encoding methods, and often lack known optimal solutions. While some efforts exist, a promising research direction involves developing standardized evaluation protocols and identifying suitable metrics. Preliminary studies on this direction have been conducted by Stein et al. [443, 445]. With much of the state-of-the-art work being conducted relying on proprietary models such as ChatGPT, future studies will also focus on developing methods for assessing and reporting on such models to improve the transparency and replicability of the presented studies. This could involve creating frameworks for sharing results that do not compromise proprietary data but still provide sufficient detail for academic reproducibility/replicability (as also advocated by many researchers in the optimization field [450, 451]).
- **Agent-based System and COPs**: Recent studies have explored how LLMs can function as autonomous agents [487]. While some of the studies already use an ensemble of LLMs, a promising research direction would be to investigate thoroughly how these autonomous agents can be effectively utilized within CO.
- **Ethical and bias considerations**:As LLMs are integrated into optimization frameworks, research should also focus on ensuring that these technologies are used responsibly. This includes considering the environmental impact of the resources required by LLMs and developing more sustainable approaches to large-scale optimization. Future research should examine the potential biases in LLM-generated optimization solutions. More generally, strategies for ensuring fairness as well as incorporating ethical considerations into the optimization process could be explored.

On the Review

The research on LLMs is rapidly evolving, thus continuously identifying areas for further research is crucial. We plan to periodically update our systematic review, as advocated by the PRISMA guidelines [378]. This process ensures that our review remains relevant in this fast-moving field and allows us to update those studies initially polished as pre-prints (thus solving one of the limitations of our work, Section 17.4.2). Future research will also account for the aspects neglected by this chapter, such as the integration of LLMs into optimization paradigms other than CO and for the dual aspect of optimization used for enhancing LLMs.

Chapter 18

Direct Querying and Probing of Large Language Models

Recent developments in Large Language Models (LLMs) have demonstrated their capacity to process text, generate executable code, and generalize across domains [338]. These capabilities have prompted researchers to explore how LLMs can support optimization tasks, from model formulation to heuristic generation and algorithmic analysis (Chapter 17). While many of these studies have produced encouraging results, they primarily assess the *functional performance* of the models (i.e., what they can do) without investigating their *representational understanding*, that is, what they internally learn about problem structure and the algorithmic behavior.

Understanding how LLMs represent optimization problems is particularly relevant. If LLMs encode meaningful information about problem structure or algorithmic performance, they could serve as general-purpose feature extractors for tasks such as algorithm selection, performance prediction, or hybrid optimization frameworks that combine learning and search. Conversely, if their internal representations lack such structure, this would highlight their current limitations.

In this study, we are concerned with Combinatorial Optimization (CO). We investigate the extent to which LLMs capture information about Combinatorial Optimization Problems (COPs), focusing on whether such information is reflected in their internal representations or can be accessed through direct querying, with particular attention to instance features and per-instance algorithm selection. Specificaly, we want to understand *(i)* if LLMs can identify salient structural or numerical characteristics of CO instances from their representations; *(ii)* if LLMs can encode information about CO instances, and if this information is accessible; and *(iii)* if the representations extracted from the hidden layers of LLMs can serve as effective instance features for per-instance algorithm selection. To tackle this matter, we combine *direct querying* and *probing* analyses. In the first part of our study, we evaluate whether models exhibit awareness of instance-level features typically indirectly associated with algorithmic performance, and whether such features can be explicitly and implicitly retrieved from instance representations. In the second part, we use probing to examine whether the internal representations can act as surrogates for handcrafted features traditionally obtained through explicit feature engineering. We conduct experiments across four (4) representative COPs, using publicly available instance sets and algorithm

portfolios from prior Instance Space Analysis (ISA) studies. Our evaluation spans multiple feature complexities, instance representations, and probing configurations, providing a comprehensive perspective on the relationship between explicit and implicit representations in LLMs

The main contributions of this chapter are as follows:

- We introduce an experimental framework for evaluating what LLMs learn about COPs through direct querying and probing.
- We provide the first large-scale analysis of how LLMs encode structural, numerical, and algorithmic information across multiple domains and input representations.
- We release the dataset, code, and prompts to foster reproducibility and further exploration at the intersection of LLMs and optimization (Appendix A).

This chapter is structured as follows. Section 18.1 outlines the methodology we followed, then Section 18.2 details the experimental setup. Section 18.3 report the empirical results and Section 18.4 draws some conclusions. Appendix H reports additional tabular results.

18.1 Methodology

In this section, we present the methodology adopted to investigate how LLMs represent and process COPs, with the broader goal of assessing whether these models encode information that can support algorithm selection. Specifically, our analysis is structured around two (2) complementary perspectives: *(i)* the *explicit reasoning abilities* of the models, i.e., how they respond to targeted prompts; and *(ii)* their *implicit representations*, i.e., the extent to which internal layers encode information about problem instances and their properties. To address these perspectives, we employ a twofold experimental strategy: *(i) Direct Querying*, wherein models are explicitly prompted to infer instance-level features; and *(ii) Probing*, wherein the LLM internal activations are analyzed to assess whether features or algorithmic performance signals are implicitly represented.

We begin by outlining the CO setting, including the problems considered, the instance representations, the feature sets, and the algorithm portfolios (Section 18.1.1). We then describe the direct querying procedure (Section 18.1.2), followed by two (2) probing analyses: the first focusing on the representation of problem features (Section 18.1.3), and the second on information relevant to algorithm selection (Section 18.1.4). The overall structure of our methodology is summarized in Table 18.1, which provides an overview of the specific objectives, approaches, and evaluation procedures associated with each experimental component.

18.1.1 Combinatorial Optimization Concepts

In this section, we recall the main concepts and definitions underlying CO to establish a common terminology and clarify how these notions are instantiated in this study.

Each COP defines a general problem structure that is instantiated through specific *instances*, which provide the concrete input data. For example, the Graph Coloring Problem (GCP) is defined on an undirected graph $G = (V, E)$, where V denotes the set of nodes and

Table 18.1: Overview of the methodological components of this study. Each component addresses a specific research goal and employs a dedicated evaluation strategy.

Goal	Methodology	Evaluation	Section
Assess whether LLMs can explicitly identify problem features from instance representations	**Direct Querying**: models are prompted to extract features directly from the instance	Comparison between model-predicted and ground-truth features	Section 18.1.2
Determine whether internal representations of LLMs encode problem features	**Feature Probing**: regressors are trained on last-layer activations to predict instance features	Comparison between predicted and ground-truth features	Section 18.1.3
Determine whether internal representations of LLMs encode information predictive of algorithm performance	**Algorithm Selection Probing**: classifiers are trained on last-layer activations to predict the per-instance best algorithm	Classification accuracy, comparison with baseline methods	Section 18.1.4

E the set of edges connecting pairs of nodes. The task is to assign a color to each node such that adjacent nodes receive different colors, yielding a valid coloring that minimizes the total number of colors used. An instance of the GCP therefore specifies the graph topology, i.e., the nodes and edges that define G. Instances are typically stored in standardized formats and parsed into data structures suitable for algorithmic processing.

For example, GCP instances are available in the DIMACS format,[1] an abridged description of which is reported below:

> In this format, nodes are numbered from 1 up to n. There are m edges in the graph.
> Files are assumed to be well-formed and internally consistent: node identifier values are valid, nodes are defined uniquely, exactly m edges are defined, and so forth.
> Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character c. There is one problem line per input file.
> The problem line must appear before any node or arc descriptor lines. The problem line has the following format: p FORMAT NODES EDGES. The lower-case character p signifies that this is the problem line. The FORMAT indicates the graph format. The NODES field contains an integer value specifying the number of nodes in the graph. The EDGES field contains an integer value specifying the number of edges in the graph.
> There is one edge descriptor line for each edge in the graph, each with the following format: e u v. The lower-case character e signifies that this is an edge descriptor line. For an edge (u, v) the fields u and v specify its endpoints. Each edge (u, v) appears exactly once in the input file and is not repeated as (v, u).

An instance in this representation for the GCP is illustrated below:

```
p edge 100 1902
e 1 2
e 1 6
e 1 10
...
```

In the remainder of this work, we refer to this standardized format as `standard`. In addition to it, we also consider two (2) alternative instance representations: natural language

[1] http://archive.dimacs.rutgers.edu/. Accessed 2025–10–14.

descriptions (`natural language`) and code-like formulations derived from the MiniZinc modeling language (`code-like`). The same instance introduced above can be expressed in `natural language` form as follows:

> The instance is named name. The graph has 100 nodes and 1902 edges. There is an edge between node 1 and node 2. There is an edge between node 1 and node 6. There is an edge between node 1 and node 10. ...

The correspondent `code-like` format is:[2]

```
int: n = ...;
set of int: V = ...;
int: num_edges = ...;
array[1..num_edges, 1..2] of V: E = ...;
```

Problem instances can be further characterized through *features*, i.e., measurable attributes that capture structural or statistical properties of an instance and are often problem-specific. Features enable comparisons across instances and support tasks such as ISA and algorithm selection. In this study, features are employed in two (2) complementary ways:

1. **Feature extraction and interpretation**: we investigate whether, and to what extent, LLMs can identify or reconstruct relevant instance features directly from the instance representations. Features are grouped by their extraction complexity[3]: *(i) directly extractable features*, which appear explicitly in the instance (○); *(ii) low-effort features*, which require minimal computation, such as counts or extrema (◓); and *(iii) high-effort features*, which involve aggregation or derived calculations, such as averages (●). Table 18.2 reports examples of features used for the GCP.

2. **Baseline for algorithm selection**: we use both the above features and additional descriptors defined in prior ISA studies (e.g., Smith-Miles and Bowly [434]) as inputs to traditional per-instance algorithm selection models. These baselines serve as reference points against which we compare models that instead rely on LLM hidden-layer representations.

Table 18.2: Graph Coloring Problem features.

Name	Description
`feat_nodes`	The total number of nodes in the graph.
`feat_edges`	The total number of edges in the graph.
`feat_degree_1`	The degree of node 1 in the graph, representing the number of connections per node.
`feat_density`	The density of the graph, calculated as the ratio of 2 times the number of edges to the number of nodes times the number of nodes minus 1.
`feat_ratio_1`	The ratio of the number of nodes to the number of edges in the graph.
`feat_ratio_2`	The ratio of the number of edges to the number of nodes in the graph.
`feat_degree_mean`	The average degree of the nodes in the graph, representing the mean number of connections per node.
`feat_degree_max`	The maximum degree of the nodes in the graph, indicating the highest number of connections any node has.
`feat_degree_min`	The minimum degree of the nodes in the graph, indicating the lowest number of connections any node has.

[2] https://www.hakank.org/minizinc/coloring_ip.mzn. Accessed 2025–10–15.

[3] Extraction complexity is defined relative to the LLM, reflecting how difficult a feature is for the model to infer.

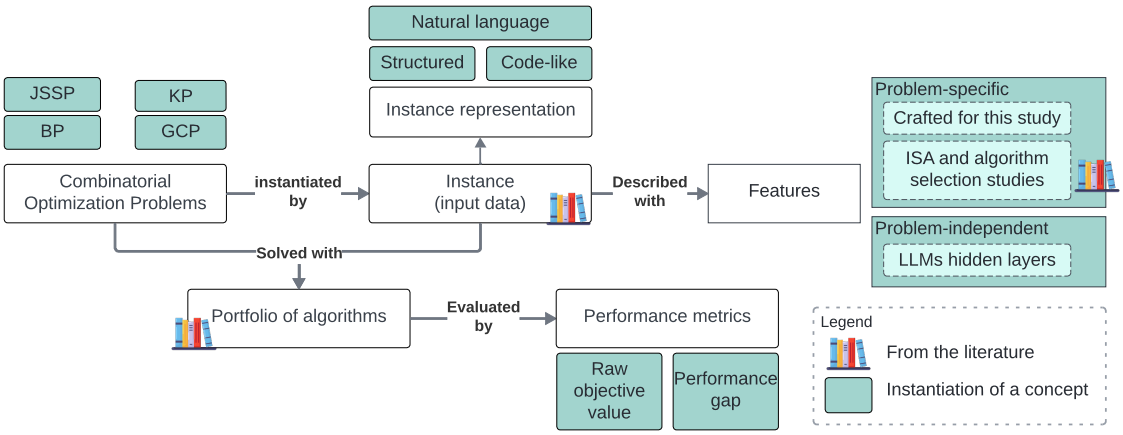


Figure 18.1: Conceptual structure of the case studies and their instantiation in this work.

COPs are typically solved by *algorithms* that explore the search space to identify high-quality solutions. In this study, we adopt an *algorithm portfolio* perspective, considering sets of algorithms with complementary performance across instances. Algorithmic performance is evaluated by comparing the objective values obtained on each instance. For illustration, in the case of the GCP, we consider two (2) graph-coloring heuristics, DSATUR (degree saturation) and MAXIS (maximal independent set), which have been demonstrated to exhibit complementary strengths and weaknesses through ISA [434]. The performance metric is the raw objective value, namely the number of colors in a valid coloring, computed on the same set of benchmark instances used in that study [434].

Figure 18.1 provides a schematic overview of this methodological framework, illustrating how the main CO concepts are organized and instantiated across the case studies considered in this work. We consider Bin Packing Problem (BPP), Graph Coloring Problem (GCP), Jobshop Scheduling Problem (JSP), anf Knapsack Problem (KP). Specifically, for each problem, we retrieve benchmark instances from prior ISA studies to ensure diversity in structural characteristics. Each instance is expressed through different representations, either structured (e.g., `standard` or `code-like`) or natural language (`natural language`). Moreover, each instance is characterized by two (2) complementary types of features. The first are *handcrafted features*, conceived, designed, and extracted through manually written code, as in traditional algorithm selection analyses. The second are *automatically derived features*, obtained directly from LLM hidden-layer activations without the need for explicit feature engineering. Quantitative characterization of the instances relies on these feature sets, while algorithm portfolios comprise methods identified in the literature as exhibiting complementary performance, evaluated through standard metrics such as raw objective value and performance gap.

Table 18.3 summarizes the main characteristics of the case studies considered in this work. For each COP, we report the number of benchmark instances, the number of algorithms included in the portfolio, and the number of features considered (— means no feature of that type). The column ISA refers to the study from which the instances, features labelled in the remainder of the paper as ISA, and algorithm portfolios were originally derived. The columns “Standard description” and “Code-like” indicate the source of the standardized and code-like representations, respectively.

Table 18.3: Summary of the considered problems.

Problem	Instances	Algorithms	Features for direct querying and probing						ISA	Standard description	Code-like
			List	Type		Compl.					
				Int	Float	○	◓	●			
BP	8,815	2	Bin capacity, items, maximum/minimum/average item weight	4	1	2	2	1	[306]		
GCP	8,278	2	Nodes, edges, graph density, ratio of the number of nodes to the number of edges in the graph and vice versa, degree of the first node, maximum / minimum / average degree	5	4	2	—	7	[434]	[153]	
JSP	4,467	12	Jobs, machines, operations, maximum / minimum / average duration	5	1	2	3	1	[446]		[344]
KP	5,300	3	Capacity, maximum / minimum / average weight/profit, weight / profit / efficiency of the first item, average efficiency	7	4	1	7	3	[435]		[344]

18.1.2 Direct Querying

Direct querying is employed to assess the extent to which LLMs can identify or describe relevant characteristics of COPs based on given instance representations. This approach focuses on externally observable behavior. Prompts are constructed according to the Reasoning–Task–Format (RTF) framework to ensure consistent task framing and output structure across all models. Each prompt comprises three (3) components:

- **Reasoning preamble:** defines the assumed role/domain expertise;
- **Task description:** specifies the reasoning objective or operation to be performed;
- **Format specification:** constrains the response to a predefined structure, ensuring syntactic validity and facilitating automated evaluation.

The prompt template used in this study is presented below. It shows how the reasoning preamble, task description, and format specification are combined.

> You are given an instance of a combinatorial optimization problem: `[full-problem-name]`.
> This is not a coding task. Do not return any code.
> Extract the numeric value of the following feature from the instance:
>
> - Feature name: `[feature-name]`
> - Feature description: `[feature-description]`
> - Expected type: `[feature-type]`
>
> The instance is provided here: """
> `[instance]`
> """
>
> Instructions:
>
> - Return a `json` object only, with a single field "`value`", i.e., '{"`value`": ...}'.
> - The "value" field should contain the numeric value of the required feature and should be of the expected type (i.e., `[feature-type]`).
> - If the value is unknown/undeterminable, return '{"`value`": `null`}'.
> - No explanations, no extra fields.
>
> Answer:

Additionally, the LLM is equipped with a structured decoding layer,[4] which guides the generation process and constrains the output to follow a predefined `json` schema. This mechanism ensures both syntactic validity and type consistency with the feature being predicted. The template of the schema is reported below, where `[feature-type]` indicates the expected data type of the target feature (e.g., `int` or `float`).

```
{
  "type": "object",
  "additionalProperties": false,
  "required": ["value"],
  "properties": {
    "value": { "anyOf": [ { "type": "[feature-type]" }, { "type": "null" } ] }
  }
}
```

The correctness of model-generated features is evaluated by comparing them against ground-truth values computed from the corresponding instances, using both exact-match and error-based criteria.

The overall setup of the direct querying experiment is depicted in Figure 18.2. For each problem, the LLM receives a prompt as per template above defined, that reports an instance represented in one of three (3) formats together with a textual instruction specifying the target feature. The output conforms to the structured decoding format previously introduced. The direct querying procedure is repeated for all features defined for each problem. Predicted feature values are then compared with ground-truth values obtained by running handcrafted feature extractors on the same instance.

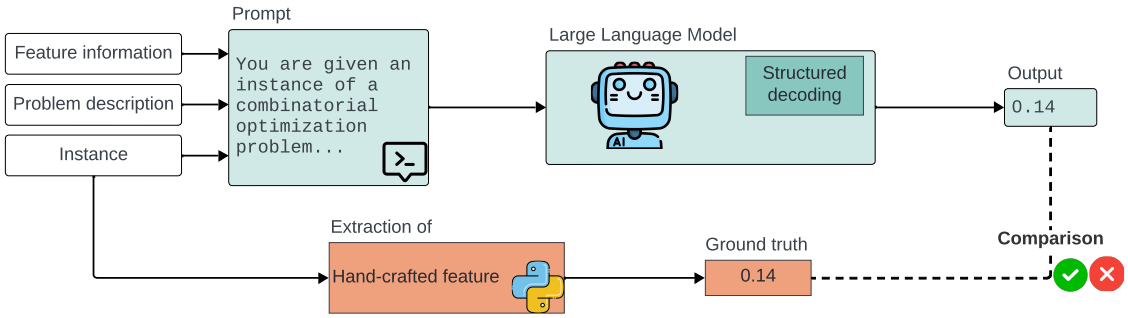


Figure 18.2: Overview of the direct querying pipeline.

18.1.3 Probing on Feature Extraction

Probing is employed to investigate whether LLMs encode information about the structural and numerical characteristics of CO instances. In contrast to direct querying (Section 18.1.2), which analyzes externally observable behavior, probing examines the latent representations learned by the model, providing a complementary perspective on how instance properties are internally encoded.

For each combination of problem and instance representation, every instance is passed through the model without triggering text generation. Hidden-layer activations are

[4] https://blog.vllm.ai/2025/01/14/struct-decode-intro.html. Accessed 2025–10–17.

extracted from the final layer, and a single vector representation for each instance is obtained by applying three (3) common pooling strategies: *mean*, *max*, and *last* (see, e.g., Tang and Yang [456]). The *mean* strategy computes the average of the last hidden-layer activations across all tokens; the *max* strategy takes the maximum activation along each representation dimension across tokens; and the *last* strategy uses the activation vector corresponding to the End-Of-Sequence (EOS) token, i.e., the model final prediction signal indicating the termination of generation. These pooled vectors provide compact, contextualized embeddings of each instance, which are then used as input to downstream probing models to assess whether the model internally encodes instance-level structural and numerical characteristics.

Each pooled vector is then paired with the ground-truth feature values corresponding to the same instance, forming a dataset for supervised probing. For each feature and representation, we train a separate regression model (probe) to assess whether the feature can be accurately decoded from the hidden representations, thereby quantifying the extent to which such information is linearly accessible within the model latent space.

To assess both the accessibility and the complexity of the information encoded in the model representations, we employ three (3) types of probes:

- **Linear probe:** a standard linear regressor used as a baseline. It follows a conventional preprocessing pipeline including feature standardization and removal of zero-variance components. High performance with this probe indicates that the feature is linearly decodable from the hidden representation.

- **Simple probe:** a one-hidden-layer Multi Layer Perceptron (MLP). The input corresponds to the pooled hidden representation (dimension 3072), followed by a hidden layer of 128 neurons with Rectified Linear Unit (ReLU) activation and a single scalar output predicting the target feature value. This probe can capture mild non-linear relationships and serves to test whether limited model capacity improves decoding accuracy.

- **Complex probe:** a LightGBM regressor designed to capture richer, non-linear dependencies. It is used to assess whether additional model capacity yields substantial accuracy gains, which would suggest that the information is encoded in a non-linearly separable manner within the representation.

The evaluation protocol mirrors that adopted for direct querying (Section 18.1.2), i.e., comparing predicted and ground-truth feature values through exact-match and error-based metrics.

The same probe types is subsequently extended to evaluate whether algorithmic performance signals, instead of instance features, are implicitly represented within the hidden states of the model (Section 18.1.4). Naturally, in this case we will have a classification task.

18.1.4 Probing on Algorithm Selection

We further extend the probing analysis to examine whether the information encoded in the pooled layers of LLMs can support downstream tasks, specifically per-instance algorithm selection. In this setting, probing is formulated as a classification problem, where

the objective is to identify, for each instance, the algorithm expected to achieve the best performance based on its latent representation.

The setup for this experiment mirrors that of the feature-probing analysis, employing the same probe types, with the key distinction that the target variable is now categorical—that is, the label of the algorithm achieving the best performance—rather than continuous. Each LLM-derived representation is paired with the ground-truth label of the best-performing algorithm for the corresponding instance, forming a dataset for supervised classification. A classifier probe is then trained to determine whether the identity of the winning algorithm can be decoded from the model hidden representations.

Performance is evaluated in terms of *set-aware accuracy*, which accounts for instances where multiple algorithms achieve identical best performance (ties). During training, one of the tied algorithms is randomly selected as the target label, whereas during evaluation, a prediction is considered correct if the classifier selects any of the algorithms attaining the best performance for that instance. A prediction is thus counted as correct if it matches any of the true *winning* algorithms for that instance.

To preserve balanced distributions across instances with different sets of winning algorithms, stratified sampling is performed over the powerset of possible labels (excluding the empty set). Each subset of algorithms defines a stratum, ensuring similar proportions of label sets in the training and evaluation splits.

To contextualize the results, we include comparisons against three (3) baselines:

- **Most frequent winner:** a naive baseline that always predicts the algorithm most frequently achieving the best performance in the training set.
- **ISA-based classifier:** a model of identical complexity trained on instance features derived from previous ISA studies.
- **Handcrafted feature classifier:** a model of identical complexity trained on the handcrafted features employed in the preceding experiments.

Classifier performance is evaluated under a stratified k-fold cross-validation protocol. All classifiers are trained and tested on identical data partitions, ensuring that each fold exposes the models to the same training and test instances. These partitions are also reused in the subsequent mixed linear analysis to establish paired comparisons of model effects, enabling a direct assessment of whether LLM-derived representations achieve comparable predictive power to traditional ISA-based and handcrafted features.

The overall probing workflow is illustrated in Figure 18.3. Each instance is processed by the LLM, from which hidden representations are extracted and pooled to obtain fixed-length embeddings. These embeddings are paired with ground-truth labels to construct datasets for regression and classification probing. The regression probes test whether instance features can be decoded from the latent space, while the classification probes assess whether algorithmic performance patterns are implicitly captured within it.

18.2 Experimental Setup

All experiments are conducted using the `Llama-3.2-3B-Instruct` model, selected for its extended context window, which is essential given the long token sequences produced

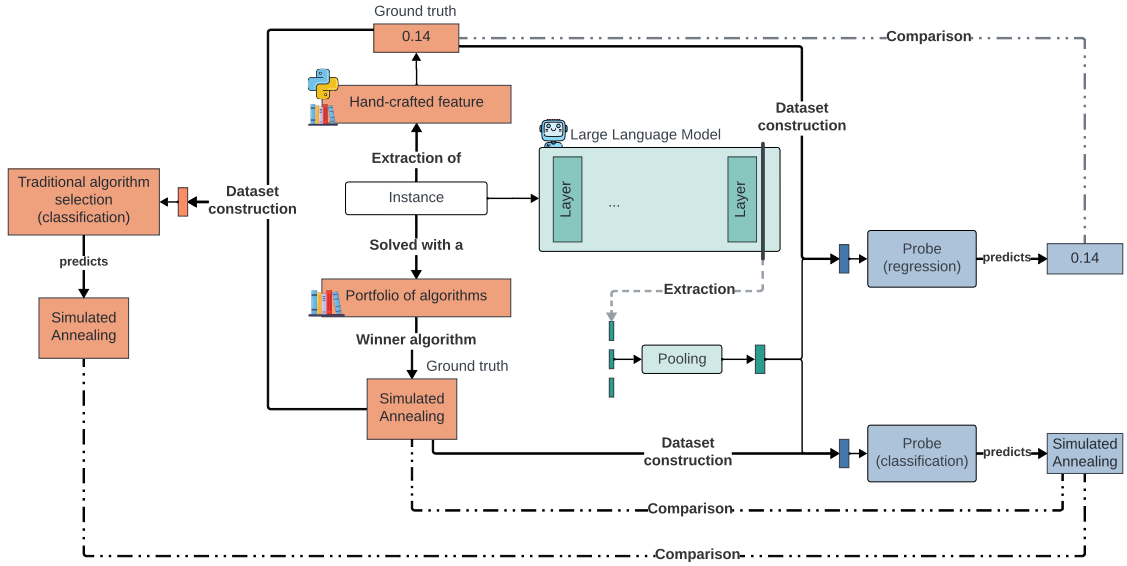


Figure 18.3: Overview of the probing process.

by all instance representations, particularly the `natural language` representation. Other available models were not suitable due to their shorter context limitations. Moreover, commercial LLMs were deliberately avoided to ensure full reproducibility of the research and to contain costs, which would otherwise preclude large-scale experimentation of the kind reported in this study.

Figure 18.4 presents the token distributions for the four COPs, highlighting in different colors the three instance representations. Overall, the context length of the `Llama-3.2-3B-Instruct` model (131,072 tokens) is not exceeded for any of the problems considered, allowing the entire input to be processed without truncation. This ensures that all instance information is preserved and no input modification is required.

The model is accessed via the `Hugging Face` interface[5] and executed using the `VLLM` library,[6] which provides improved robustness and efficiency over the standard `transformers` implementation. `VLLM` enables effective parallelization of both text generation in direct querying and hidden-state extraction in probing, allowing efficient processing of large batches and long input contexts. Hidden activations are fully extracted through `VLLM` and aggregated using the pooling strategies described in Section 18.1.3.

To ensure reproducibility, all experimental data and configurations are managed with `dvc`, while the source code is version-controlled through `git`. Experiments are executed on an NVIDIA DGX Station equipped with four NVIDIA A100 GPUs (80 GB each). The full experimental set comprises 26,860 instances, each evaluated under three distinct representations. Separate runs are performed for direct querying, which involves token generation, and probing, which is computationally faster. The total runtime amounts to approximately two weeks for direct querying and one week for probing. Post-processing tasks, including regression and classification analyses, are carried out offline on a MacBook Pro (Apple M4, 32 GB RAM, 4+6 cores). Random seeds, problem sets, and replicate blocks

[5] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct. Accessed 2025–10–30.

[6] https://docs.vllm.ai/en/latest/. Accessed 2025–10–30.

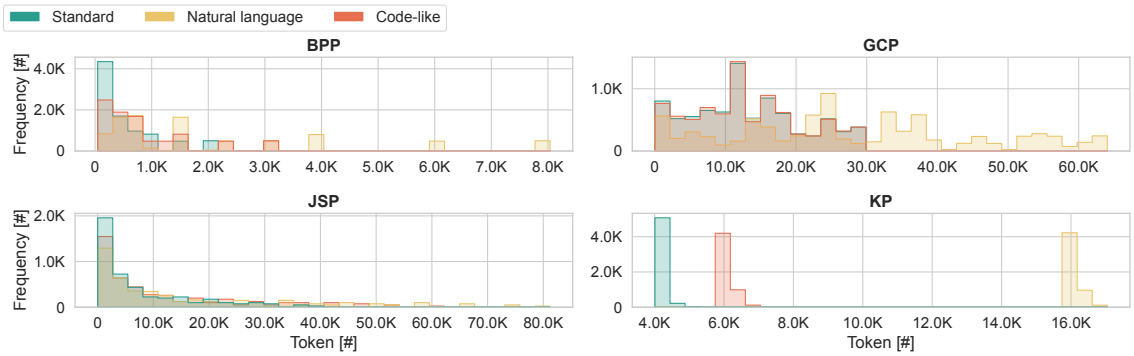


Figure 18.4: Number of tokens per problem and type of instance representation.

are fixed across all runs to ensure consistent experimental conditions. The complete `git` repository, including configuration files, experimental scripts, and processing pipelines, will be released as an open repository upon paper acceptance.

The probing models are implemented using `scikit-learn`, with `LinearRegression` for regression, `MostFrequentClassifier` and `LogisticRegression` for classification, and `PyTorch` with the `skorch` adapter for the MLP probes applied to both tasks. Furthermore, `LightGBM` is used for regression and classification alike. The `scikit-learn` framework was selected for its interoperability and consistency, allowing a unified experimental pipeline across all probe types. An *ad hoc* adapter class was developed for the set-aware classification task to ensure the correct computation of its evaluation metric within the same workflow.

Regarding the algorithm portfolios, the results are drawn from the original ISA studies (also referenced in Table 18.3), together with the corresponding instances and features labeled as ISA features. The ground-truth features employed in the experiments presented in Sections 18.1.2 and 18.1.3 are computed by dedicated feature-extraction scripts developed in `Python`. The handcrafted feature extractors are included in the project code base and integrated within a data preparation pipeline that processes the benchmark instances and computes the relevant features for the downstream LLM-related experiments.

The experiments are organized as follows. In the direct querying setting, the prompt template is instantiated with each problem instance in its given representation (for all three representations), together with the target feature, its description, and the structured decoding schema guiding the LLM generation. Results are collected for every problem–feature pair, yielding a total of 611,037 experiments, each executed independently within the LLM pipeline (Figure 18.2). In the probing setting (pipeline reported in Figure 18.3), only the instance representation is provided to the network, resulting in 80,580 experiments. The last hidden state is extracted, pooled, and used to train regression models on a 70%/30% train–validation split, with a dedicated regressor trained for each feature. In the algorithm selection setting, both the pooled last hidden states—covering all combinations of representation and pooling strategy—and the manually designed features (handcrafted or ISA-based) are used as classifier inputs. Depending on the number of algorithms considered (see Table 18.3, third column), either binary or multi-class models are trained using a stratified k-fold cross-validation scheme ($k = 5$) for each problem. Overall, the GPU-intensive tasks amount to 691,617 LLM executions, while the regression and classification models were trained offline on the laptop computer, comprising 1,833,111

Table 18.4: Direct querying metrics by problem and representation. ○, ◓, ● indicate features of increasing extraction complexity: directly extractable, low-effort computation, and high-effort computation, respectively. With —, we indicate that no features of such type are present. Except for the MAE, in all other case the value reported is a proportion w.r.t. the total.

		MAE			Equals			Within 1%			Within 5%		
		○	◓	●	○	◓	●	○	◓	●	○	◓	●
BPP	`code-like`	34.82	70.85	133.32	0.96	0.46	0.00	0.96	0.51	0.05	0.96	0.54	0.22
	`natural language`	2.98	27.33	158.88	1.00	0.90	0.00	1.00	0.90	0.02	1.00	0.91	0.08
	`standard`	471.22	197.49	102.77	0.44	0.23	0.00	0.44	0.25	0.05	0.44	0.28	0.23
GCP	`code-like`	12.56	—	81.31	1.00	—	0.01	1.00	—	0.04	1.00	—	0.09
	`natural language`	0.00	—	179.94	1.00	—	0.02	1.00	—	0.08	1.00	—	0.13
	`standard`	910.47	—	22.89	0.54	—	0.01	0.54	—	0.02	0.55	—	0.05
JSP	`code-like`	0.00	446.61	9.59	1.00	0.44	0.12	1.00	0.45	0.18	1.00	0.49	0.34
	`natural language`	0.42	541.31	5.97	0.98	0.37	0.14	0.98	0.37	0.24	0.98	0.41	0.47
	`standard`	20.53	643.63	18.08	0.49	0.10	0.01	0.49	0.11	0.05	0.49	0.14	0.21
KP	`code-like`	9,331.44	583,906.78	381.53	0.97	0.19	0.00	0.97	0.21	0.02	0.97	0.24	0.03
	`natural language`	3,968.89	6,984.98	259.40	0.99	0.38	0.00	0.99	0.46	0.03	0.99	0.50	0.09
	`standard`	261,452.48	24,747.88	341.84	0.06	0.11	0.00	0.06	0.15	0.01	0.06	0.19	0.02

regression and 322,320 classification experiments.

18.3 Results

This section reports the results of the experimental analysis conducted to evaluate the behavior and representational properties of LLMs when applied to COPs. The experiments follow the methodology introduced in Section 18.1, encompassing three (3) complementary analyses: direct querying, feature probing, and algorithm selection probing. Each analysis focuses on a distinct aspect of model behavior: respectively, explicit feature identification, implicit feature encoding, and predictive modeling of algorithmic performance.

18.3.1 Direct Querying

We begin the experimental analysis by examining the outcomes of the direct querying experiments. Table 18.4 summarizes the results across the four COPs, grouped by feature complexity: ○ for features directly extractable from the instance representation, ◓ for those requiring low-effort computations (e.g., counts or extrema), and ● for features involving more complex calculations or aggregations. Performance is assessed using multiple metrics: Mean Absolute Error (MAE), computed as the average absolute deviation from the ground truth; the proportion of exact matches; and the proportions of predictions within 1% and 5% deviation from the ground truth values. These latter metrics provide a more sensitive view of near-accurate predictions, complementing the MAE analysis.

Overall, the direct querying experiments show that LLMs can reliably infer simple instance features directly from textual or code-like representations, while struggling with features requiring even minimal computation. Performance is highest for ○ features across all problems, with accuracy approaching perfect exact matches, but declines notably for ◓ and ● features, indicating limited numerical and procedural reasoning capabilities as expected [420].

Among representations, `natural language` consistently yields the best results, followed

by `code-like`, whereas `standard` inputs perform markedly worse. This suggests that LLMs rely primarily on semantic and contextual cues rather than structural or symbolic encodings. Generation stability is high, with non-null responses produced in over 99.8% of all queries, indicating consistent model behavior across problems and representations.

To investigate whether these limitations stem from the models reasoning behavior or from the nature of their internal representations, the next analysis turns to feature probing. This approach aims to determine whether the information underlying more complex features is nevertheless implicitly encoded within the latent space, even if it cannot be explicitly retrieved through direct querying.

18.3.2 Probing on Feature Extraction

We now turn to probing to examine whether LLMs implicitly encode structural and numerical information about problem instances within their internal representations. In contrast to direct querying, which evaluates explicit reasoning ability, feature probing tests whether such information can be linearly or non-linearly decoded from hidden-layer activations, offering a deeper insight into the model representational structure.

We compute MAE values for each problem, representation, and feature complexity level, comparing the performance of *direct querying* and *probing*. These results are reported in Appendix H and summarized in Figure 18.5, which illustrates the difference in MAE between the two approaches across all configurations. In these plots, negative values indicate that probing yields lower error than direct querying, whereas positive values denote configurations in which direct querying performs better.

The comparative analysis of MAE values across problems and feature complexity levels reveals a complementary relationship between direct querying and probing. For high-computation features, probing exhibits a clear advantage, reflecting a greater capacity to capture and exploit the complex relational information embedded within the problem representations. Conversely, direct querying performs better for extraction or low-computation features, particularly with the `natural language` representation. This finding confirms and extends the observations from the direct querying analysis, showing that the method remains most effective when retrieving explicit, easily accessible descriptors that require minimal or no transformation.

Beyond the MAE, the approximation of each approach was further assessed by measuring the proportion of predictions falling within 1% of the ground truth. The corresponding results, shown in Figure 18.6, compare the performance of the different probing configurations. For completeness, the figure also includes results for direct querying, with the analysis distinguishing among feature complexity, problem type, and representation.

The findings confirm and extend the trends observed in the MAE analysis. Direct querying remains more effective for the `natural language` representation and simple extraction tasks, achieving agreement rates (within 1%) above 0.9 across all problems. In contrast, no single probing configuration emerges as a universal best performer, although the `MLPRegressor` generally underperforms relative to the other probes.

These results indicate that direct querying and probing capture distinct yet complementary aspects of the feature space. The former excels with explicit extractions and structurally simple descriptors, whereas the latter leverages higher-level, computed features.

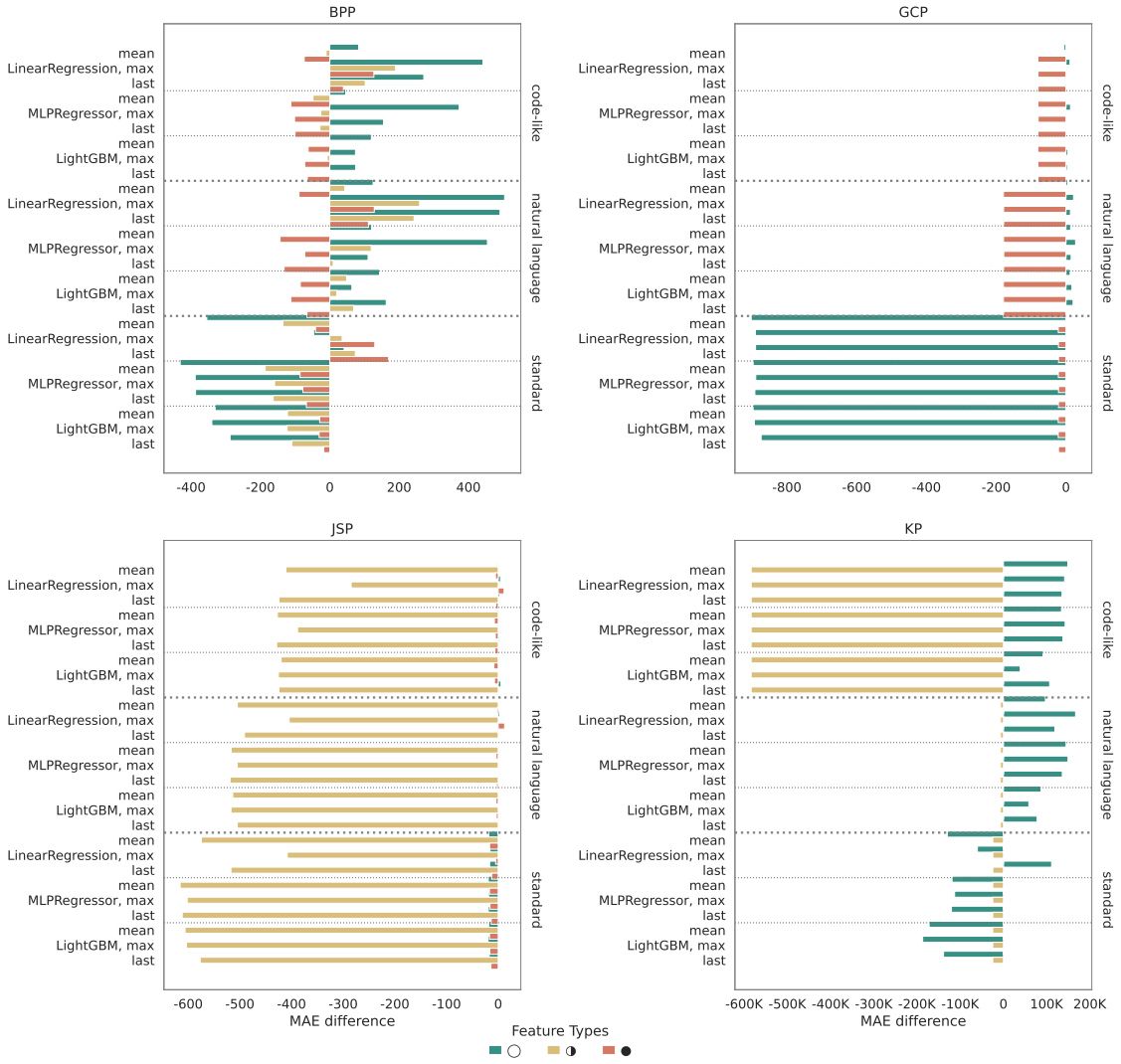


Figure 18.5: Difference in MAE of Different Prompting with respect to the Direct Querying.

18.3.3 Probing on Algorithm Selection

While the previous analyses show that certain features cannot be explicitly extracted through prompting, it remains an open question whether the information required for such features is nevertheless encoded within the model hidden representations. To explore this, we assess whether these latent embeddings can match the effectiveness of handcrafted features in the key downstream task of algorithm selection. A complete summary of the statistical analyses performed below is reported in Appendix H.

Comparative Statistical Analysis of Classifier Performance

This subsection presents a comparative statistical analysis of the classifiers employed in the algorithm selection task, aiming to determine whether their predictive performances differ significantly. The analysis serves as a preliminary validation step to ensure that the

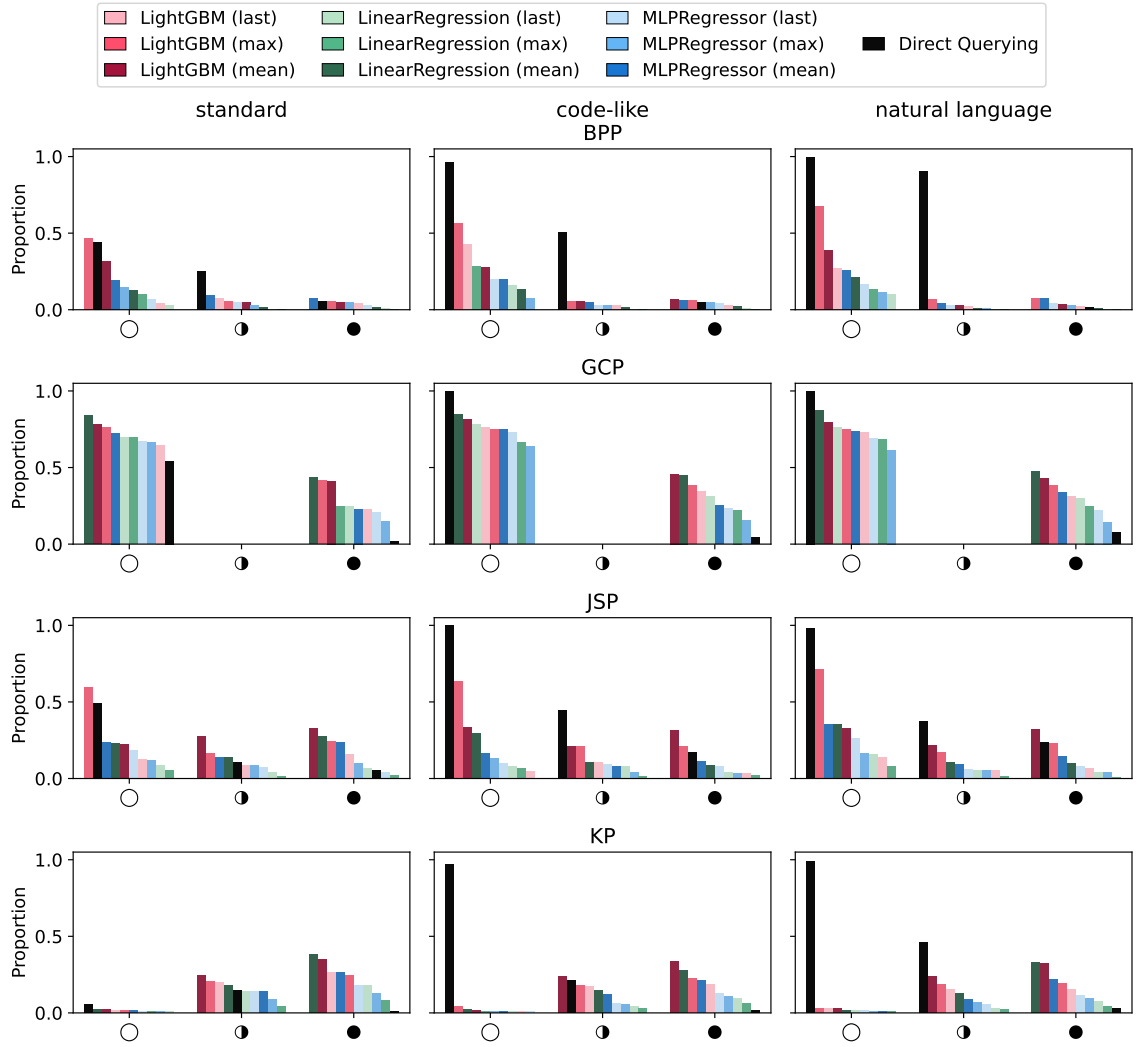


Figure 18.6: Probing and Direct Querying results within 1% to the Ground Truth. We distinguish by feature complexity (grouped bars), problem (rows), and problem representations (columns).

choice of classifier does not confound the interpretation of subsequent results. To this end, multiple classifiers are evaluated under identical data partitions.

To assess the influence of classifier type on prediction accuracy,[7] we fitted a linear mixed-effects model with the block (*problem*, *replicate*) as a random intercept to account for repeated measurements across problem instances. The model included the classifier as a fixed factor and was estimated via maximum likelihood. This formulation enables testing for systematic performance differences among classifiers while controlling for variability attributable to specific problems.

The overall effect of the classifier was highly significant (likelihood-ratio test, $p < 0.001$),[8] indicating that classifier choice substantially affects predictive accuracy. The estimated fixed effects, expressed relative to the reference classifier (`MostFrequentClassifier`), show that both `LogisticRegression` and `LightGBM` achieved significantly higher accuracies than

[7]For these analyses the term accuracy has to be intended in the set-aware sense presented in Section 18.1.4

[8]For lightness in notation, we indicate with p the p-value.

the baseline ($+0.192$ and $+0.149$ points, respectively; $p < 0.001$ for both), whereas the `MLPClassifier` exhibited a small but statistically significant decrease in accuracy (-0.011 points; $p = 0.044$). The random-effect variance associated with the problem grouping ($\sigma^2 = 0.012$) indicates moderate between-problem variability, suggesting that classifier effects are largely consistent across different problem types.

Figure 18.7 shows the distribution of classifier accuracies for each problem type. Overall, the results confirm the trends identified by the mixed-effects model: the `LightGBM` and `LogisticRegression` classifiers consistently outperform the `MostFrequentClassifier` baseline across all problems, whereas the `MLPClassifier` yields comparable or slightly lower performance.

Performance differences are also problem-dependent. For example, all classifiers perform relatively well on BPP, while the spread of accuracies is larger for JSP and KP, indicating greater variability in problem difficulty and model stability. This pattern aligns with the non-zero between-problem variance estimated by the mixed-effects model.

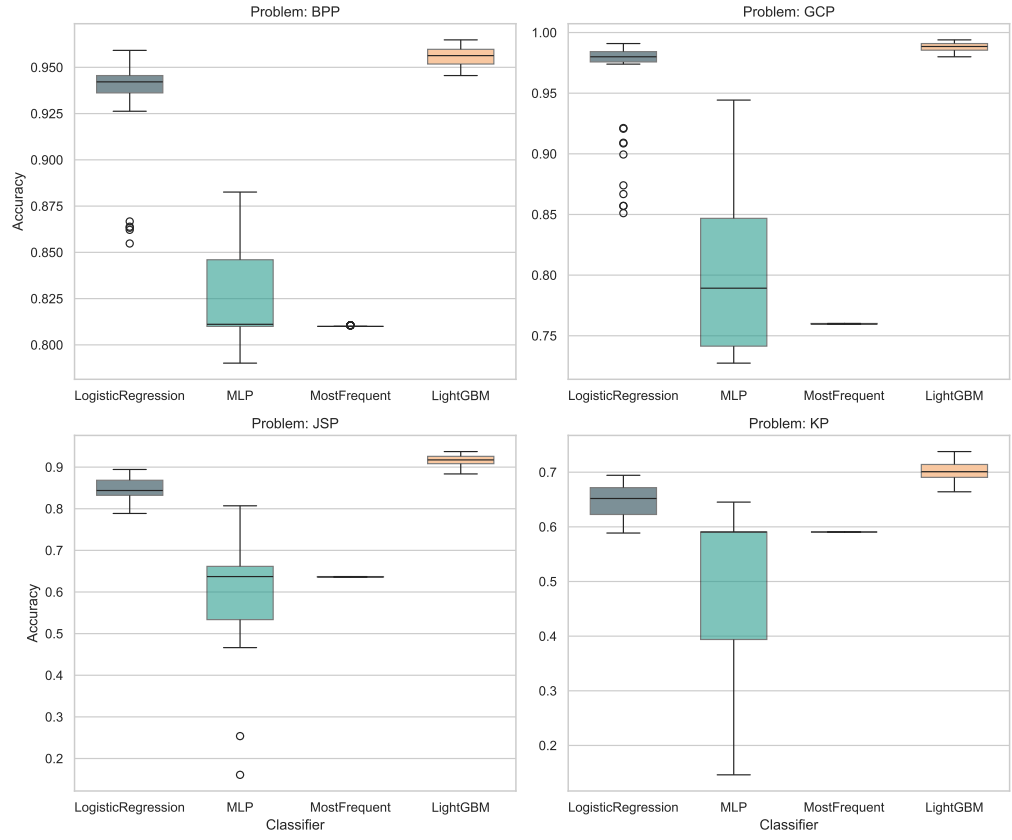


Figure 18.7: Classification accuracy by problem and classifier. Each panel shows the mean and variability of the accuracy achieved by the evaluated classifiers across the benchmark problems in the different probing settings.

Given these results, the two underperforming classifiers, i.e., `MostFrequentClassifier` and `MLPClassifier`, are excluded from the subsequent analyses to focus on models that demonstrate stable and competitive predictive behavior. This ensures that the remaining comparisons more accurately reflect the representational capacity of the features rather than limitations of the classification models themselves.

Comparative Analysis of ISA-Based and Handcrafted Feature Sets

We now compare the predictive behavior of the ISA-based and handcrafted feature sets to assess whether they yield systematically different outcomes in the algorithm selection task. While ISA-based features are designed as comprehensive descriptors and may include or extend the handcrafted ones, it is nevertheless important to evaluate whether this broader coverage translates into superior predictive power. Establishing this comparison also provides a consistent baseline for the subsequent analysis, where the predictive power and robustness of manually engineered features are contrasted with those derived from LLM hidden representations.

To examine whether the predictive behavior of the two baseline feature sets differs systematically, we fit a linear mixed-effects model with *classifier* as a fixed factor and *block*, defined as the (*problem*, *replicate*) pair, as a random intercept. The analysis identified a significant main effect of the specific baseline employed (LRT $\chi^2(1) = 40.0$, $p < 0.001$). The handcrafted feature baseline achieved a mean accuracy of 0.786 (95% CI $[0.737, 0.836]$), while the ISA-based representation yielded an increase of 0.031 points (95% CI $[0.023, 0.039]$). The model revealed a significant main effect of classifier, with `LightGBM` yielding an increase of accuracy of 0.091 points (95% CI $[0.083, 0.099]$) with respect to `Logistic Regression`. The estimated between-problem variance ($\sigma^2 = 0.013$) suggests, also in this case, moderate heterogeneity but consistent improvement across problems.

Complementary tests provide a nuanced view of this effect. A pairwise Tukey comparison on unadjusted data did not detect a statistically significant difference, reflecting the variability introduced by individual problems. Nonetheless, a non-parametric Wilcoxon-based Two One-Sided Tests (TOST) procedure with an equivalence margin of 0.01 failed to demonstrate equivalence ($p \approx 1$ for the upper bound), indicating that the two feature sets cannot be considered practically interchangeable. Overall, the ISA-based features offer a modest yet consistent advantage over the handcrafted baseline, supporting their use as the primary reference in subsequent analyses.

Analysis of Representation and Pooling Effects on Probing Performance

To investigate how different combinations of instance representation and pooling strategy influence probing performance, a full-factorial analysis is conducted to evaluate main and interaction effects between the three representations (`standard`, `code-like`, and `natural language`) and the three pooling strategies (*mean*, *max*, and *last*) on classifier accuracy. The analysis controls for problem-specific variability to isolate systematic effects attributable to representation and pooling choices, providing insight into how these factors shape the extractable information from LLM hidden activations.

We fitted a linear mixed-effects model including *classifier* as a fixed factor and *block* (i.e., the pair (*problem*, *replicate*)) as a random intercept to assess how different combinations of representation and pooling activations influence classifier performance in the algorithm selection task. The model evaluated the main and interaction effects of the two factors, while accounting for systematic differences between classifiers. Likelihood ratio tests indicated significant main effects of both representation ($\chi^2(6) = 21.32$, $p = 0.0016$) and pooling ($\chi^2(6) = 26.61$, $p < 0.001$), whereas the representation $\times$ pooling interaction was not significant ($\chi^2(4) = 6.77$, $p = 0.15$). A main effect of classifier was also observed, with

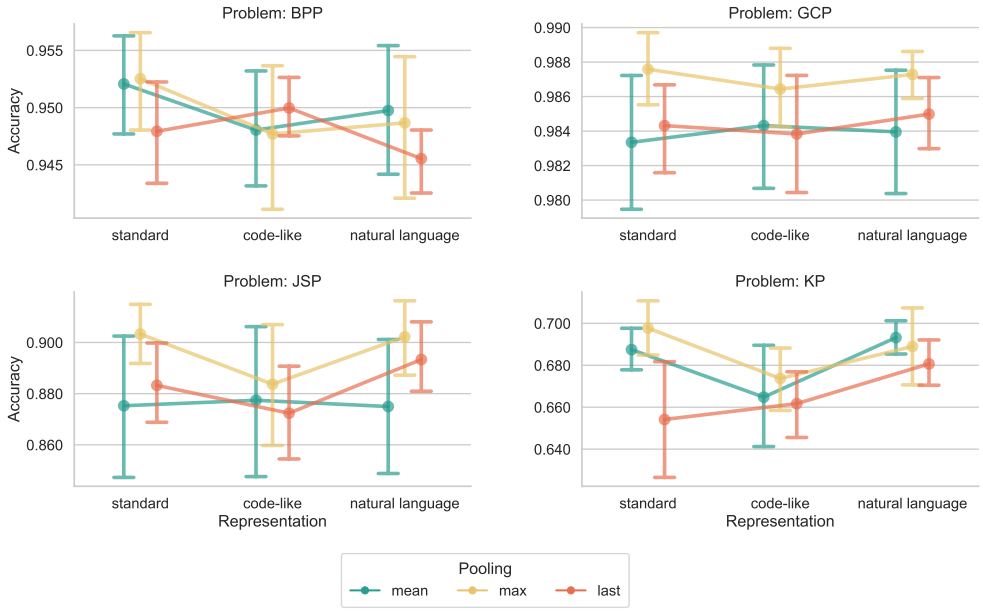


Figure 18.8: Effects of representation and pooling in the full factorial experiment. Each panel shows the mean accuracy (95% CI) for combinations of representation and pooling, averaged across problems and classifiers.

`LightGBM` outperforming `Logistic Regression` by approximately 0.03 accuracy points ($p < 0.001$). These results suggest that classifier choice and the representation-related factors contribute independently to performance, without evidence of interactions.

The estimated coefficients indicate that the `natural language` representation performs on par with the `standard` representation (difference < 0.01, $p = 0.81$), confirming that representation has no practical influence overall. Although the global likelihood-ratio test for the representation factor was significant ($\chi^2(6) = 21.32$, $p = 0.0016$), pairwise contrasts revealed no systematic advantage for any specific representation. The between-problem variance ($\sigma^2 = 0.014$) still indicates moderate heterogeneity, suggesting that these patterns are largely consistent across problems.

On the other hand, among pooling strategies, only *max* pooling showed a statistically significant advantage over the *mean* baseline ($+0.011$, $p = 0.005$), while the effect of *last* pooling was not significant (-0.007, $p = 0.064$). Although the magnitude of the improvement is modest, this suggests a slight benefit from aggregating activations via the maximum rather than the mean.

Figure 18.8 illustrates the mean accuracies (with 95% confidence intervals) obtained for each problem under different combinations of representation and pooling strategy. Across all problems, *max* pooling consistently provides a slight advantage over the other two pooling methods, regardless of the underlying representation. The effect of representation, however, is less consistent and appears to depend on the specific problem. This variability suggests that the influence of representation may interact with problem characteristics, warranting further investigation on a broader set of COPs, a direction that lies beyond the scope of the present study.

Overall, the analysis confirms that while pooling strategies exert statistically detectable

effects, their practical impact on accuracy is minor. Classifier choice remains the dominant determinant of performance, underscoring the robustness of LLM-derived embeddings. Nevertheless, for the final analysis we adopt a single configuration consisting of *max* pooling and the `standard` representation to provide a consistent basis for comparison with the baseline models.

Comparative Analysis of ISA and LLM-Derived Representations for Algorithm Selection

In this final analysis, we compare the algorithm selection performance obtained using traditional ISA-based features with that achieved from LLM-derived hidden-layer activations. The aim is to determine whether information encoded within LLMs can match or exceed the predictive power of handcrafted instance descriptors in identifying the per-instance best algorithm. The comparison is conducted between the best-performing configurations identified in the previous analyses: the ISA-based feature set and the LLM-based setup using *max* pooling with the `standard` representation. In both cases, the `LightGBM` classifier is employed, as it demonstrated superior predictive performance across earlier experiments.

Although the fixed effect of representation (ISA vs. LLM) is statistically significant and positive ($+0.008$, $p < 0.001$), the estimated between-problem variance ($\sigma^2 = 0.012$) exceeds the fixed-effect magnitude, indicating that the improvement provided by ISA is smaller than the variability observed across problems. A non-parametric Wilcoxon-based TOST, applied to the ranked paired differences, confirmed equivalence at 0.01. The Hodges–Lehmann 95% confidence interval for the median paired difference was $[-0.0039, 0.0244]$, with a median difference of 0.0054, entirely within the ± 0.01 equivalence region. Overall, these results suggest that although the tiny effect of ISA is statistically detectable, it is practically negligible, and that ISA and LLM can be considered indistinguishable in predictive performance within a few hundredths of accuracy.

We conclude this analysis by comparing the average accuracies achieved by ISA and LLM in the algorithm selection task across the different problems. Figure 18.9 illustrates this comparison, showing that the most challenging problem for LLM is KP, which likely explains the lack of statistical equivalence observed in the previous tests. Unfortunately, we are not able to perform a meaningful within-problem analysis, since the number of replicates per problem does not allow for reliable conclusions. However, an informal inspection suggests that for KP the equivalence margin is around ± 0.015, indicating that future studies should consider including problems with a similar structure. Nevertheless, the differences remain within a few hundredths, suggesting that overall performance remains comparable.

18.4 Concluding Remarks

We now summarize the contributions of this chapter (Section 18.4.1), its limitations (Section 18.4.2), and possible directions for future research (Section 18.4.3).

18.4.1 Content

We investigated how LLMs represent COPs and whether these internal representations can support downstream decision-making tasks. Through a combination of *direct querying*

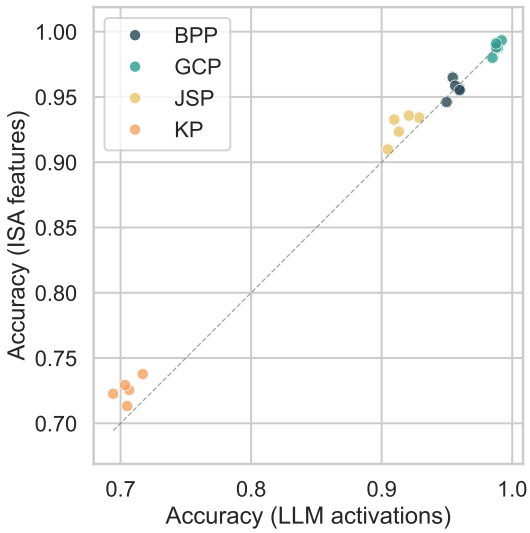


Figure 18.9: Comparison of Accuracies in Algorithm Selection between ISA features equipped `LightGBM` and LLM activation-based features under `standard` representation.

and *probing* analyses, we examined both the explicit reasoning capabilities and the implicit knowledge encoded within the models across four (4) benchmark problems and three (3) types of instance representations. Two (2) tasks were considered: *feature extraction*, assessing whether models can recover meaningful structural and numerical characteristics from problem instances, and *per-instance algorithm selection*, testing whether internal representations can predict the best solver for each instance.

Our results revealed that:

- The direct querying experiments (Section 18.3.1) show that LLMs can infer simple, explicitly stated instance features from `natural language` and `code-like` representations. For low-complexity features, the MAE is moderate and predictions fall within 1% of the ground truth value in most cases, indicating a moderate surface-level comprehension. However, performance deteriorates markedly for features requiring even limited computation or aggregation, confirming that current LLMs have difficulty performing quantitative reasoning beyond explicit extraction.

- The probing analyses (Section 18.3.2) reveal that LLMs partially encode structural and numerical information about CO instances in their internal representations. Such information can be partially recovered through supervised decoding. In general, probing achieves lower errors with respect to direct querying on complex features.

- Through an extensive experimental campaign (Section 18.3.3), we find that LLM-extracted latent representations effectively serve as instance features surrogates for per-instance algorithm selection. Across all problems, max pooling offers a consistent but modest advantage, while the effect of instance representation depends on the problem type. When compared with traditional ISA-based features, LLM-derived embeddings achieve statistically comparable predictive performance, demonstrating that latent representations can match the effectiveness of human-defined descriptors.

Notably, the classifier choice remains critical: the best-performing model, `LightGBM`, successfully captures non-linear relationships among both LLM-derived and human-defined features, as shown by its better predictive performance.

The latter findings reveal an important trade-off. While human-defined features demand substantial domain expertise and manual engineering, they are computationally inexpensive and inherently more interpretable. In contrast, LLMs achieve comparable predictive power with minimal feature design effort but at a significantly higher computational cost for their extraction and with a considerable loss of interpretability. This contrast highlights a broader question for future research: how to balance interpretability, reproducibility, human effort, and energy efficiency when integrating LLMs.

18.4.2 Limitations

The main limitation of this study is that the experimental analysis is restricted to a single LLM family and four benchmark COPs, which, although representative, do not capture the full diversity of optimization settings.

A further caveat concerns the chosen LLM: depending on the adopted classification and on the time at which the experiments were conducted, the selected model may be regarded as a *small* language model. While this characterization is inherently time-dependent and debatable, extending the analysis to additional and more recent LLMs would be a natural and valuable direction for future work (Section 18.4.3).

18.4.3 Future Work

Possible future work aims in tackling the limitations of this study, that is extending this investigation to other model architectures, larger instance scales, and real-world optimization problems. Furthermore, recent progress in structured reasoning and tool-augmented generation with LLMs offers promising opportunities to strengthen direct querying methods.

Another promising avenue lies in integrating LLM-derived activations into broader ISA pipelines, using them not only for algorithm selection but also for tasks such as instance clustering, performance landscape modeling, and exploratory analysis of problem spaces.

Part VI

Conclusions

Chapter 19

Conclusions

This thesis set out to address the inherent challenges in the empirical study, design, and reliable application of metaheuristic algorithms for Combinatorial Optimization (CO). It pursued four (4) complementary directions: the analysis of metaheuristic components, the usage of advanced benchmarking methodologies, the unification of proliferating problem formulations, and the integration of emerging technologies within optimization workflows.

This final chapter offers a comprehensive overview of the main findings, contributions, and perspectives of the thesis. While each research chapter concluded with its own discussion and outlook, here we integrate those results to provide a comprehensive view of the research programme as a whole. We begin with an overview of the thesis content (Section 19.1) and contributions (Section 19.2), followed by a discussion of overarching limitations (Section 19.3) and directions for future work (Section 19.4).

19.1 Content

This thesis comprises six (6) parts in total, with Part I and Part VI (the present one) respectively introducing and concluding the work. In what follows, we focus on the four (4) research parts that form the methodological and experimental core of the thesis.

19.1.1 Metaheuristic Components

Table 19.1: Mapping of research questions (RQs) to chapters, employed methodologies, and summarized findings w.r.t. Part II (Goal — Component-based Analysis).

Research Question	Reference	Methodology	Answer
RQ1: How do alternative tabu tenure strategies and inverse move definitions affect performance and search behavior?	Chapter 5	Systematic comparison of 5 tabu tenures and 2 inverse definitions on PFSP	Fixed and reactive lists achieve similar quality; stricter inverses lead to convergent trajectories.
RQ2: To what extent can informed external control dynamically adapt temperature schedules in SA?	Chapter 6	HH dynamic control of SA temperature	Adaptive control matches tuned SA in the majority of the cases.

Part II examined the fundamental building blocks of metaheuristic algorithms and how their design choices and interactions influence search dynamics, thereby addressing the first goal of this thesis: *component-level understanding*. To this end, we focused on two of the most established local search metaheuristics, Tabu Search (TS) and Simulated Annealing (SA), and systematically analyzed their internal mechanisms through controlled, component-based experimentation. Table 19.1 reports an overview of the methodologies and related Research Questions (RQs).

In Chapter 5, we decomposed TS into seven (7) elementary components and investigated how different tabu list strategiesand inverse move definitions affect search behavior on the Permutation Flowshop Scheduling Problem (PFSP). The experiments, conducted over 160 validation instances, revealed that despite their conceptual differences, the simplest (fixed-length) and the most adaptive (reactive) tabu lists achieve statistically equivalent solution quality, indicating that sophistication does not necessarily translate into superior performance. The use of Search Trajectory Networks (STNs) provided a deeper understanding of the inverse-move dynamics: stricter policies produced trajectories converging to common search paths, while more relaxed ones encouraged exploration.

In Chapter 6, we analyzed temperature control in SA, replacing static cooling schedules with Hyper-Heuristics (HHs) as adaptive mechanisms that dynamically select the temperature according to the search state. Across three (3) problem domains, this adaptive SA achieved results comparable to those obtained with finely tuned static schedules, while eliminating the need for extensive parameter calibration. This confirmed that heuristic-level adaptation can replicate the performance of exhaustive tuning, offering a pathway toward general, self-adjusting search algorithms.

Together, these studies contributed to a modular and interpretable view of metaheuristic design. On the methodological side, they demonstrated how decomposition into explicit components enables rigorous analysis; on the practical side, this modularity was embodied in the extension of the EasyLocal++ framework, which now supports component-level configuration and naturally integrates with the `HyFlex` architecture (Appendix B). The resulting ecosystem supports transparent experimentation and cross-domain solver development.

19.1.2 Oven Scheduling Problem

Part III applied advanced benchmarking methodologies to a real-world parallel batch scheduling problem, the Oven Scheduling Problem (OSP), thereby addressing the second goal of this thesis: *advanced benchmarking on real-world problems*. This part combined theoretical modelling, algorithmic design, and empirical analysis to establish a comprehensive understanding of the problem's structure and its algorithmic landscape. Table 19.2 reports an overview of the content and RQs.

In Chapter 9, we examined the OSP from a theoretical perspective, introducing several Lower Bounds (LBs) that enabled speed-ups on exact methods and the characterization of instance hardness. In Chapter 10, we developed two (2) metaheuristic algorithms tailored to the OSP, namely a multi-neighbourhood SA and a Large Neighborhood Search (LNS) algorithm. The proposed methods matched the best-known solutions for small instances and improved them for large-scale industrial cases. Beyond raw performance, the integration of advanced analysis tools, Instance Space Analysis (ISA) in Chapter 11

Table 19.2: Mapping of research questions (RQs) to chapters, employed methodologies, and summarized findings w.r.t. Part III (Goal — Advanced Benchmarking on Real-World Problems).

Research Question	Reference	Methodology	Answer
RQ1: Can exact methods be improved through theoretical insights?	Chapter 9	Development of theoretical LBs	LBs enable better solutions, faster proofs, and reduced runtime.
RQ2: Can exact methods' performance be enhanced by integrating heuristic approaches?	Chapter 10	Integration of exact models in LNS framework	Gains for medium instance; diminishing returns for large ones.
RQ3: How do classical metaheuristics perform on the OSP?	Chapter 10	Multi-neighbourhood SA in EasyLocal++	Comparable to exact solvers on small and superior on large instances.
RQ4: What features explain OSP instance difficulty?	Chapter 11	ISA and algorithm selection	Features linked to bounds and job–machine distribution predict difficulty.
RQ5: How can we characterize OSP search spaces?	Chapter 12	STNs	Search space is multimodal with scattered high-quality basins.

and STNs in Chapter 12, added an explanatory dimension: STNs revealed how search trajectories evolve in the search, while ISA linked instance features to algorithmic behavior.

These complementary perspectives allowed us to identify when and why certain metaheuristics perform best and to interpret their dynamics in relation to the problem structure. The combination of theoretical bounds, algorithmic advances, and empirical analysis thus positioned the OSP as a fully characterized benchmark that connects industrial relevance with methodological insights.

19.1.3 Home Healthcare Routing and Scheduling Problem

Part IV addressed the fragmentation of the literature by introducing a unified formulation for the Home Healthcare Routing and Scheduling Problem (HHCRSP), thereby tackling the third goal of this thesis: *unified formulations of problem variants*. This part integrated model design, instance representation, and solver adaptation into a single coherent framework that supports both research comparability and practical applicability. Table 19.3 reports an overview of the content and RQs.

In Chapter 14, we proposed a model that incorporates multiple real-world aspects within a single formulation. These features, previously considered only in isolation across separate studies, are here captured consistently through a modular constraint structure. The same chapter introduced a unified representation for instances and solutions, compatible with existing datasets yet extensible to new formulations. A complete toolbox was developed to support this ecosystem, including instance generation, format translation, and schema validation, thereby enabling transparent benchmarking and reproducible experimentation.

In Chapter 15, we generalized and extended two (2) state-of-the-art methods from the literature to the unified setting. Both were evaluated on translated benchmark instances and newly generated data covering an expanded feature space. The results provided empirical evidence of scalability and cost trade-offs across model variants, while the accompanying

Table 19.3: Mapping of research questions (RQs) to chapters, employed methodologies, and summarized findings w.r.t. Part IV (Goal — Unified Formulation of Problem Variants)

Research Question	Reference	Methodology	Answer
RQ1: Can a unified HHCRSP model subsume existing variants while ensuring compatibility and validation?	Chapter 14	Translation, validation, and instance generation	Unified model captures old and new variants; schema validates results.
RQ2: Do new instances expand coverage of the instance space?	Chapter 14	PCA of instance features	New instances improve instance space coverage.
RQ3: Can general-purpose solvers match specialized ones?	Chapter 15	Development of SA and MILP baselines	General-purpose solvers perform comparably or better in most cases.

open artifacts constitute the first comprehensive benchmark suite for this problem family.

These contributions transformed the HHCRSP from a collection of heterogeneous formulations into a unified, extensible, and openly available research domain. They demonstrate that methodological consistency through standardized models, datasets, and solvers can substantially enhance comparability.

19.1.4 Large Language Models

Part V explored how Large Language Models (LLMs) can interact with CO tasks, thereby addressing the fourth goal of this thesis: *critical exploration of LLMs in optimization*. Table 19.4 reports an overview of the content and related RQs.

Table 19.4: Mapping of research questions (RQs) to chapters, employed methodologies, and summarized findings w.r.t. Part V (Goal — Critical Exploration of LLMs in Optimization)

Research Question	Reference	Methodology	Answer
RQ1: What is the current landscape of LLMs in CO?	Chapter 17	Systematic literature review	LLMs used across pipeline tasks but mostly as black boxes; prompt and evaluation gaps persist.
RQ2: What do LLMs know about CO instances and algorithms?	Chapter 18	Direct querying and probing experiments	LLMs fail in feature extraction but succeed in algorithm selection; representation has minor effect.

Chapter 17 presented a systematic review conducted under Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guidelines. From more than 2,000 retrieved publications, 103 studies were retained and analyzed according to the stage of the optimization pipeline they addressed. The review revealed that most existing works treat LLMs as black-box tools, paying little attention to prompt style, cross-problem generalization, or the mechanisms underlying model behavior.

To investigate these aspects empirically, Chapter 18 examined LLMs through direct querying and probing experiments in two settings: feature extraction and algorithm selection. In the former, results showed high inconsistency even for simple, explicitly stated instance features, indicating limited reliability in factual recall and structured reasoning. In contrast,

the algorithm-selection experiments revealed that internal representations learned by the models could successfully predict the best-performing solver with remarkable accuracy. Together, these findings suggest that while current LLMs are unreliable as extractors, their latent knowledge captures meaningful regularities of optimization algorithms, opening promising avenues for hybrid, data-driven algorithm selection.

19.2 Contributions

Beyond the individual parts, this thesis makes a set of overarching contributions spanning methodological, empirical, and representational dimensions. Taken together, they outline a comprehensive view of how to design, analyze, and generalize metaheuristic and data-driven approaches for CO.

Methodological Contributions. The thesis advances the methodological foundations of empirical optimization by promoting a component-based and adaptive view of metaheuristics. The systematic dissection of TS and SA components clarified how design choices, such as tabu tenures and temperature control, affect search dynamics. These studies culminated in the development of general, problem-independent control mechanisms and their integration into reusable frameworks (EasyLocal++ and `HyFlex`), enabling transparent experimentation across domains.

Empirical Contributions. Through extensive benchmarking on both synthetic and real-world problems, the thesis generated new empirical insights into algorithm behavior. For the OSP, a combination of LBs, multi-neighbourhood search methods, and landscape analyses revealed the structural factors driving performance. For the HHCRSP, the comparative evaluation of generalized solvers provided quantitative evidence on scalability and trade-offs, grounding the unified model in practice.

Representational and Infrastructural Contributions. A major outcome of this research is the creation of standardized and open artefacts for reproducible optimization research, including the maintenance and extension of reusable frameworks and integration tools. The unified instance and solution schemas for the HHCRSP, the validation and translation utilities, and the curated datasets for the OSP collectively form a consistent benchmarking ecosystem. These artefacts, publicly released alongside the thesis (Appendix A), exemplify the shift from ad-hoc experimentation toward reusable, transparent infrastructures that support cumulative scientific progress.

19.3 Limitations

The research presented in this thesis is subject to several limitations that naturally delineate the boundaries of its validity and generality. These limitations are discussed below with the intent of clarifying the contexts in which the findings apply and outlining promising directions for future investigation. Limitations of the single experimental studies are reported within the chapters.

Scope and Generalizability Although the thesis spans multiple problem families and algorithmic frameworks, its empirical analyses primarily focus on discrete, single-objective optimization settings, and focus mainly on neighborhood search algorithms. The selected problems represent important but specific subclasses of assignment, scheduling, and routing problems. Extending the findings to other domains and paradigms, including multi-objective, would require additional validation to assess whether the observed patterns and component-level insights generalize.

Methodological and Computational Constraints The experimental studies involved extensive computational campaigns, yet practical limitations in runtime and available hardware constrained the range of tested configurations. In particular, parameter tuning and sensitivity analyses, while methodologically rigorous, were necessarily bounded by finite computational budgets. Similarly, the implemented frameworks (EasyLocal++ and connection with `HyFlex`) were optimized for flexibility rather than raw performance, which may limit direct comparison with highly specialized solvers. In addition, experiments with LLMs are limited by GPU memory.

Representativeness Despite efforts to promote open science, data availability in several domains remains limited due to privacy and institutional constraints. While this thesis introduced new instances, instance generators, and standardized formats, the diversity and realism of public datasets still fall short of reflecting the full range of possible operational conditions found in practice. As a consequence, the generality of empirical conclusions must be interpreted with caution.

Exploratory and Emergent Aspects The investigation of LLMs in optimization remains exploratory by design. The reproducibility and stability of results remain open challenges, given the rapid evolution of the underlying models and their nature. The findings in Part V therefore represent a snapshot of an emerging field rather than definitive evidence of capabilities. Further methodological standardization and open access to model interfaces will be crucial for confirming and extending these insights.

Taken together, these limitations do not undermine the validity of the results but rather highlight the complexity of conducting comprehensive, reproducible research in modern optimization and in the field of LLMs. They also point to fertile ground for future extensions, some of which are outlined in the following section.

19.4 Future Work

The research presented in this thesis opens several avenues for future investigation, both in the refinement of existing methodologies and in the exploration of new interdisciplinary directions. Building on the limitations discussed above, these perspectives can be grouped into four (4) main themes.

Advancing Adaptive and Learning-driven Metaheuristics Future studies could extend the adaptive mechanisms explored in Part II by integrating online learning in form og HHs in other metaheuristics. Such approaches would enable search algorithms to autonomously infer effective parameter settings or operator choices based on

real-time feedback from the search trajectory. In addition, combining landscape analysis tools such as STNs with online adaptation could yield algorithms capable of dynamically recognising and responding to the structure of the problem at hand. For instance as prohibition mechanisms on areas of the search space frequently visited by the algorithm.

Extending Unified Problem Formulations The unified modelling principles introduced for the HHCRSP can be extended to other domains that suffer from similar fragmentation, such as workforce rostering, emergency response logistics, or integrated production–distribution planning. A promising direction also lies in formalising cross-domain interoperability between problem families, enabling a shared representation space for instance generation, transfer learning, and meta-benchmarking.

Integrating AI and data-driven reasoning into optimization The investigation of LLMs in Part V opens the door to a deeper synergy between these models and CO. However, prior to an extreme pollution of the research environment, there is the need for rigorous evaluation protocols, transparent prompting standards, and open benchmark corpora to ensure that progress in this emerging area remains interpretable and reproducible.

Fostering Open, Reproducible, and Collaborative Infrastructures A key challenge for the optimization community is to transform isolated research artifacts into interoperable ecosystems, as advocated by the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action. Future work will involve the long-term maintenance and evolution of the frameworks developed in this thesis (EasyLocal++, unified HHCRSP schemas, and associated tools), as well as their integration with existing initiatives such the ROAR-NET Cost Action. Establishing shared repositories for algorithms, datasets, and analyses would ensure that reproducibility and transparency become permanent features of empirical research in the future.

Appendix A

Artifacts

All artifacts associated with this thesis are publicly available, including the corresponding experimental data. In a few cases, the links to the GitHub repositories may still be private while the related papers are under review. As of 2025–10–31, the private repositories concern the work on the Home Healthcare Routing and Scheduling Problem and the experiments involving Large Language Models.

Part II — Metaheuristic Components

- Component-based EasyLocal++: easylocal-v4
- Component-based Tabu Search: tabu-search
- Dynamic Temperature Simulated Annealing and `HyFlex` / EasyLocal++ interface: hhsa

Part III — Oven Scheduling Problem

- Theoretical Lower Bounds (LBs) implementation: oven-scheduling CLI
- Theoretical LBs data: lower-bounds-data
- Large Neighborhood Search implementation: osp-lns
- Simulated Annealing implementation: osp-ls
- Instance Space Analysis data and scripts: isa-osp
- Landscape and Search Trajectory Analysis tools and results: stn-osp
- Collection of instances:[1] osp-instances

[1]Note that the original Zenodo repository is by M. Lackner et al. [275], but a second version includes the author of this thesis.

Part IV — Home Healtchcare Routing and Scheduling Problem

- Data: data-and-managerial-insights, mirrored in Zenodo
- Complete toolbox: toolbox, mirrored in Zenodo
- ROAR-NET Problem Statements: problem-statements
- ROAR-NET API didactic examples: roar-net-api

Part V — Large Language Models

- Behavior and representation in large language models: Zenodo

Appendix B

EasyLocal++

Researchers face the imperative of evaluating various algorithms to identify the most suitable technique for a given problem. To navigate the panoply of possible algorithms, it would be helpful to turn to frameworks and libraries, which enable, in one single project, to implement and assess multiple metaheuristics [178]. Since the nineties, a sprout of tools has been proposed [383]. Among the earliest in this sense, our research group has developed EasyLocal++, a C++ white-box framework designed to support the development and evaluation of local search algorithms. We have been using and updating EasyLocal++ since its creation in 1999. Problems we have tackled with this tool belong to different domains, including timetabling [45, 76], rostering [80], scheduling [85], logistics [86], facility location problems [87], bioinformatics [143], and finance [142].

EasyLocal++ has also been used in solution methods that obtained state-of-the-art results in the benchmark datasets of many problems [44, 66, 80, 81, 517] and its usage extends beyond research, encompassing real-world and industrial applications. For instance, it has been employed in emergency facility location within the Italian Health Ministry project EasyNet[1] [122], in industrial scheduling within the context of steel production in collaboration with Danieli Automation [29], and in nurse rostering [79] in collaboration with Windex.[2] In addition, EasyLocal++ is used by the software house EasyStaff,[3] which was initially a spin-off of the University of Udine.

Most of the code presented in this thesis has been developed using EasyLocal++ and the author of this thesis is one of the current maintainers and has contributed both to its ongoing development and to one of its spin-offs, namely `JuLes`. Thus, the aim of this appendix is to describe EasyLocal++, its evolution through time and its main characteristics. In general,the idea of providing a 25-year perspective on software development was motivated by three key observations. First, much of the decision-making and design rationale for the original framework had only been transmitted orally since its first publications in the early 2000s; Second, the framework has seen substantial enhancements over time, particularly in response to evolving research needs; Third, there is renewed and growing interest within the community in problem modeling and framework design, especially in light of the Randomized Optimization Algorithms Research Network (ROAR-NET) Cost Action,

[1] https://easy-net.info/. Accessed 2025–10–10.

[2] https://www.windex.it. Accessed 2025–10–10.

[3] https:/www.easystaff.it. Accessed 2025–10–10.

which emphasizes the importance of interfaces and software libraries for randomized optimization algorithms.

EasyLocal++ is available on `GitHub` at the public repository https://github.com/iolab-uniud/easylocal. The stable version is tagged with v.3.3, whereas the newest is in the development branch v.4. The code is distributed under MIT license.

The content of this appendix comes from the following published work:

- Ceschia, S., **Da Ros, F.**, Di Gaspero, L., & Schaerf, A. (2024). EasyLocal++ a 25-year Perspective on Local Search Frameworks: The Evolution of a Tool for the Design of Local Search Algorithm. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (pp. 1658–1667). New York, NY: ACM.

This appendix is structured as follows. Section B.1 provides an overview on similar metaheuristic frameworks. Section B.2 presents the evolution of EasyLocal++ since 1999. Section B.3 describes the overall architecture of the project. Section B.4 details the most relevant features. Section B.5 reports of EasyLocal++ can be connected directly to `HyFlex`. Section B.6 outlines the teaching purposes of the tool. Finally, Section B.7 draws some conclusions and outlines future directions.

B.1 Related Work

In many optimization paradigms like Integer Linear Programming (ILP) and Constraint Programming (CP), numerous widely accepted tools exist, including `CP Optimizer`, `Gurobi`, `Gecode`, `OR-Tools`, and `MiniZinc`, among others. Such universally accepted tools have not been available for local search algorithms, or more in general metaheuristics and randomized optimization algorithms, where researchers still rely on customized coding solutions [450]. In fact, these implementations are often tailored to the particular problem being addressed, hindering the re-usability of the code and the replicability of the experiments. Despite this common practice, as early as three decades ago, there were already software packages available for metaheuristic implementations, such as those proposed by Andreatta, Carvalho, and Ribeiro [23] and Vaessens, Aarts, and Lenstra [473]. The first introduced a conceptual framework for local search based on design patterns, whereas the latter suggested a template to organize local search algorithms.

In Table B.1, we consider and summarize the main characteristics of a selection of metaheuristic software. The tools are primarily implemented in languages such as `C++` and `java`, although some attempts also exist in `Scala`, `C#`, and `Python`. Among the various packages, notable distinctions are made between libraries (e.g., GAlib [482], GAUL [2]) and frameworks (e.g., HotFrame [177] and ParadisEO [161]), with the former emphasizing code reuse (where user-defined code calls methods in the library) and the latter focusing on code-architecture reuse (where the framework code calls user-defined code). A few attempts are also in the form of Application Programming Interface (API), as in the case of the one sponsored by ROAR-NET,[4] and Domain-Specific Language (DSL).[5]

Certain tools are tailored to address a specific group of metaheuristics; for instance, Emili [379] is exclusively dedicated to local search methods, while PyMOO [52] focuses on

[4] https://github.com/roar-net/roar-net-api-py. Accessed 2025–10–10.

[5] https://www.hexaly.com/. Accessed 2025–10–10.

Genetic Algorithms (GAs). In contrast, other packages (e.g., ParadisEO) support several optimization paradigms.

Another relevant distinction of the tools concerns the support of different multi-criteria cost function evaluations. Some of them are capable of supporting Multi-Objective Optimization (MOO) (in the Pareto sense) instead of just dealing with single objective optimization. Furthermore, some tools, like Emili, provide support in the automatic design of algorithms [183] rather than treating metaheuristics as monoliths. Eventually, while most of the packages have been updated in the last few years, a few ones have not been maintained for quite some time (e.g., HotFrame). Note that, in this case, we refer to the most recent date among the last commit in the repository and the last publication exclusively describing the tool. For further disquisitions, we forward the interested reader to some literature reviews on the topic [383, 404, 432].

Table B.1: Comparison of Metaheuristic (MH) software, distinguishing by programming language, software type, implemented algorithms, multi-objective optimization (MOO) capabilities, automated design (AD) features, and last update.

Tool	Language	Type	MH	MOO	AD	Last update
EasyLocal++	C++	Framework	LS	✓	✓	2025
ECJ [425]	java	Framework	LS, EA	✓		2022
Emili [379]	C++	Framework	LS		✓	2022
ROAR-NET[a]	Python	API	LS, EA			2025
GALib[b]	C++	Library	GA			2024
GAUL [2]	C	Library	LS, GA			2010
Hexaly[c]	DSL	Modeling language	LS			2025
HeuristicLab[d]	C#	Framework	LS, EA, GA	✓		2024
Hotframe [177]	C++	Framework	LS			2002
jMetal [165]	java	Framework	EA	✓	✓	2024
OscaR [391]	Scala	Framework	CP-LS			2023
ParadisEO [161]	C++	Framework	LS, EA	✓	✓	2022
PyMOO [52]	Python	Framework	EA	✓		2024

[a] https://github.com/roar-net/roar-net-api-py. Accessed 2025–10–10.
[b] https://github.com/s-martin/galib. Accessed 2025–10–10.
[c] https://www.hexaly.com/. Accessed 2025–10–10.
[d] https://github.com/heal-research/HeuristicLab. Accessed 2025–10–10.

B.2 History

In this section, we provide an overview of the development of EasyLocal++, highlighting key events (Figure B.1).

Schaerf, Cadoli, and Lenzerini [423] started developing a general framework for local search which resulted in Local++. This initial concept, built upon a hierarchy of abstract template classes, was further refined, expanded, and developed by Di Gaspero and Schaerf [146, 147, 150], leading to the creation of EasyLocal++. Although resembling the present version to some extent, its neighborhood functionalities were initially limited, primarily focusing on basic neighborhood handling capabilities. In its second version, EasyLocal++ began managing multi-modal neighborhoods, incorporating up to three neighborhoods via

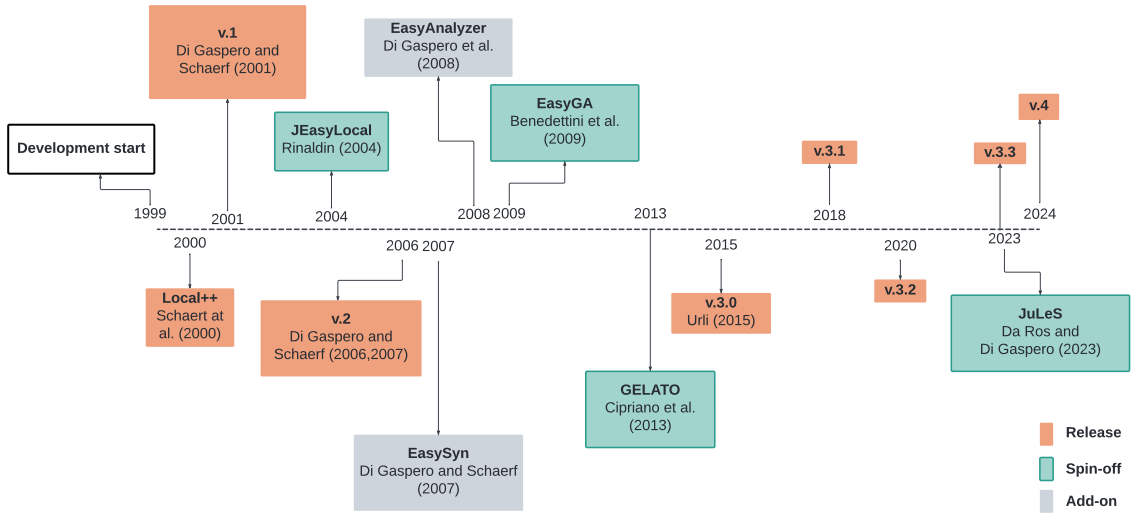


Figure B.1: The evolution of EasyLocal++ since its first development in 1999.

chains of `if` control structures. The adoption of a structured multi-neighborhood approach within EasyLocal++ was facilitated by its application to specific problem domains, such as the traveling tournament problem [145] and a plethora of timetabling problems [149]. This structured approach was later revised by Di Gaspero and Urli (see for further details Urli [471]) through template metaprogramming using recursion, ultimately giving rise to EasyLocal++ v.3. In this version, several improvements were made and the code was generally refactored (e.g., management of several data structures with shared pointers instead of stack-allocated objects).

Between 2018 and 2022, significant revisions and simplifications were made to EasyLocal++. These modifications included the creation of a header-only version of the code (leading to EasyLocal++ v.3.1), a major overhaul of the control structure for the Simulated Annealing (SA) algorithm (resulting in EasyLocal++ v.3.2), and refactoring in the handling of outputs together with the integration of learning mechanisms in the selection of neighborhoods in SA [82] (culminating in EasyLocal++ v.3.3, that is the current stable version).

Recently, additional key features have been added to the framework, including the integration of multi-objective algorithms [121] and automated algorithm design (resulting in an initial implementation of EasyLocal++ v.4, i.e., the development version).

EasyLocal++ provides not only an easy way to encode local search solution methods but also capabilities for easy testing of code and analysis of neighborhoods. In this sense, several add-ons have been developed to address ancillary activities over the years. For instance, EasySyn [148], a software tool for the automatic synthesis of the source code for a set of local search algorithms, and EasyAnalyzer [144]. Unfortunately, these initial efforts were tightly coupled with the framework code, and therefore, their development was left behind in the framework's evolution. Nevertheless, some of those concepts (e.g., testers, see Section B.4.4) were integrated directly into the core of EasyLocal++.

Along the evolution of EasyLocal++, several spin-off software have been developed, including porting of the main local search modules in `java`, i.e., JEasyLocal [412], and `julia`, i.e., `JuLes` [120]. Additionally, we have pursued experiments with other metaheuristics,

like GAs with EasyGA [47], and hybridization with CP in Gelato [108].

Despite being completely decoupled from specific problem domains (Section B.3.1), the development of EasyLocal++, the creation of software spin-off, and the addition of new features have been driven by the requirements of implementing specific applications. In this regard, its development can be viewed as bottom-up rather than top-down: as challenges arose during the application of EasyLocal++ to various optimization problems, we have continuously updated the framework by addressing missing functionalities and incorporating new elements and components, among other improvements. These efforts have made EasyLocal++ more robust and versatile.

At present, EasyLocal++ is undergoing significant refactoring aimed at simplifying its codebase using modern C++ features (Section B.4.1). The ongoing development has brought EasyLocal++ to version 4, reflecting its continual evolution and adaptation to meet new requirements and needs. Nevertheless, the seminal design principles are kept untouched w.r.t. its first version.

B.3 Architecture

The internal logic and structure of EasyLocal++ are fully exposed to and modifiable by the users. This transparency allows the user to understand, modify, and extend the functionalities of the framework with detailed insights into its workings.

We now outline the design principles of EasyLocal++ (Section B.3.1). We then describe the algorithms that are available off-the-shelf (Section B.3.2) and discuss the concepts related to multi-neighborhoods (Section B.3.3).

B.3.1 Design Principles

EasyLocal++ primary design goals are, on the one hand, the easy prototyping of solution methods for the problem at hand (i.e., applying an off-the-shelf local search to the problem); on the other hand, the possibility to extend the framework without modifying the existing code (i.e., developing new metaheuristics or new specific components).

The codebase is structured into distinct macro-modules, such as solvers and runners, each of which is further divided into modules (or classes). Each class is responsible for handling a specific aspect of local search (Figure B.2). This allows modularity, maintainability, scalability, and readability.

Modules within EasyLocal++ can be categorized based on their relevance to the specific optimization problem or their general applicability across different problems. The interface between these problem-specific and problem-independent modules relies on the *Inversion of Control* principle (also known as *Hollywood Principle*, recall Booch [59]): the control strategy of the metaheuristic is dealt with by the framework itself (see problem-independent part of Figure B.2), while the user implements specific code related to the problem (see problem-specific part of Figure B.2). Eventually, the framework will call the user's code when necessary. This allows a loose coupling and a clear chain of responsibilities among the different classes and opens the door to the definition of abstract metaheuristic.

EasyLocal++ classes can also be organized considering the distinction provided by Pree [395]; therefore, accounting for *hot* and *frozen spots*. Hot spots represent areas where users

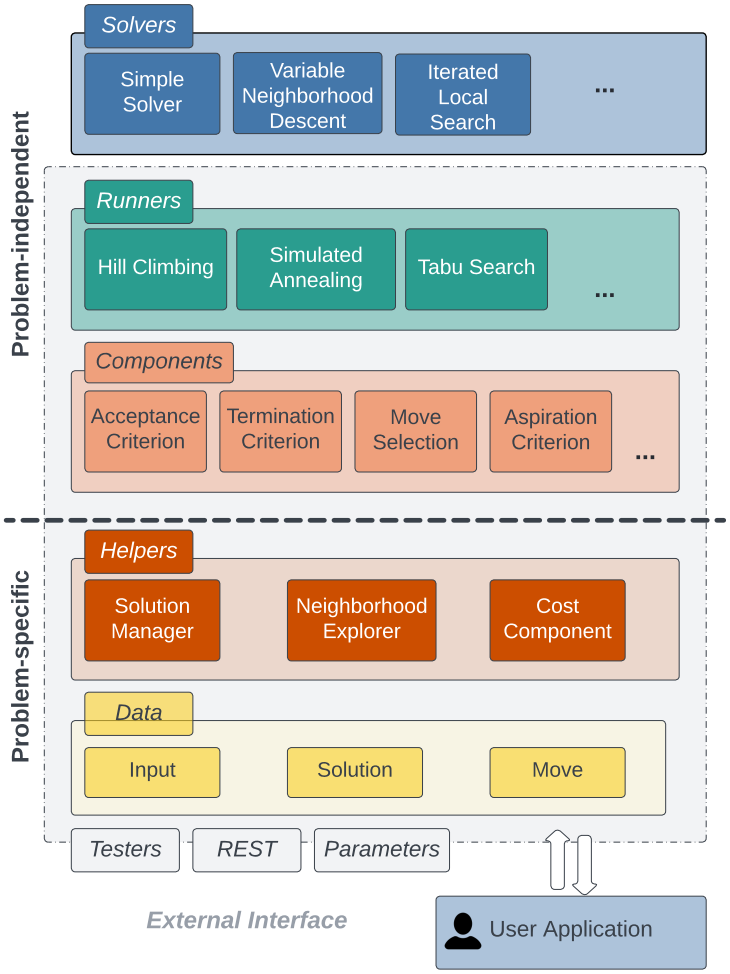


Figure B.2: EasyLocal++ system architecture. A distinction between problem-specific and problem-agnostic parts of the framework is made.

are actively required to develop and customize functionalities based on problem-specific details (i.e., data and helpers in Figure B.2). The frozen spots, conversely, are entirely implemented at the framework level and deal with problem-independent strategies (i.e., solvers, runners, and components in Figure B.2). While the user cannot modify frozen spots, they can be easily extended (i.e., they are closed to modifications but open to extension). For instance, the user can add a new component (e.g., a new termination criterion, a new acceptance criterion, etc.), develop a brand new local search method to be added to the runners or a different solving strategy (included in solvers).

We first present EasyLocal++ from the perspective of a user who aims to develop a local search solution method for a specific problem at hand and then from the perspective of a user who wants to extend the framework.

Solving a Problem

We now present the system architecture from the perspective of a EasyLocal++ user who aims to develop a local search solution method for a specific problem at hand. For representative purposes, we consider a well-known benchmarking problem in combinatorial

optimization, the Permutation Flowshop Scheduling Problem (PFSP) (Section 2.2).

The user has to start defining the *Data* of the problem at hand. The `Input` class takes care of the problem instance. In the case of the PFSP, reported in Listing B.1, it will store the number of jobs, the number of machines, etc.

Listing B.1: Implementation of the Input class for the PFSP.

```
class PFSP_Input {
public:
   // <...>
   size_t jobs, machines;
};
```

The solutions and their manipulations are explicitly represented through the concepts of `Solution` and `Move`. Conversely, other frameworks rely on an indirect representation of solutions and use standardized moves, which involve an encoding/decoding phase later (e.g., Biased Random-Key Genetic Algorithm (BRKGA) [462] relies on a $[0, 1]$-valued chromosome that needs to be decoded into an actual solution to the problem at hand).

For instance, a user tackling the PFSP with EasyLocal++ is required to implement a `Solution` class that deals with a vector to store the processing order of the jobs on the machines (Listing B.2), and a `Move` that describe the swap of two jobs in such a schedule (Listing B.3).

Listing B.2: Implementation of the Solution class for the PFSP.

```
class PFSP_Solution {
public:
   vector<size_t> schedule;
};
```

Listing B.3: Implementation of the Move class for the PFSP, specifically the code regards the swap of two jobs in the schedule.

```
class PFSP_SwapMove {
public:
  int pos_i, pos_j;
};
```

Additionally, the user must define some classes specific to local search that must be embedded into the helpers macro-module. The generation of an initial solution should be implemented into a `SolutionManager` class, while the `NeighborhoodExplorer` class is in charge of the neighborhood exploration of a specific `Move`.

For instance, Listing B.4 reports the co-routine method in the `NeighborhoodExplorer` related to the `PFSP_SwapMove` in charge of the complete enumeration of the neighborhood.

Listing B.4: Implementation of the neighborhood generator for swap move of the PFSP.

```
Generator<PFSP_SwapMove> PSFP_SwapNeighborhoodExplorer::Neighborhood(shared_ptr
    <const PFSP_Solution> sol) const {
  for (size_t pos1 = 0; pos1 < st->Jobs() - 1; ++pos1)
    for (size_t pos2 = pos1 + 1; pos2 < st->Jobs(); ++pos2)
```

```
        co_yield {pos1, pos2};
}
```

The composition of multiple moves (e.g., a swap move used together with an insert move) is synthesized from the framework itself (Section B.3.3).

We assume a minimization problem, thus, as a consequence, the objective function consists of `CostComponent`s that take care of the computation of the penalties for the problem. How the costs are evaluated depends on the objective function strategy (see Section B.4.6); currently, the costs can be aggregated into a scalar value or compared with a lexicographic ordering or in the Pareto sense.

Since the methods provided by all these classes are crucial in terms of performance, in EasyLocal++ v.4, they are implemented through static polymorphism (see Section B.4.1); that is, these elements are further supplied to problem-independent classes through template instantiation based on C++ 17 *concepts*. Indeed, when the user defines all the previously described elements, the implementation of an actual local search solution method simply consists of an instantiation of the suitable runner which takes care of the local search technique (e.g., Tabu Search (TS), SA, Hill Climbing (HC)) and, optionally, of the solver which combines single runners in a more sophisticated solution strategy. Each runners implements a slight variation, based on the characteristics of the local search algorithm itself, of the abstract local search algorithm (depicted in Algorithm 14). The behavior of a given runner can be further specified through a palette of components. For instance, a variety of stopping criteria could instantiate the `Terminate` function, based on the user's needs. From the command line, the user can specify which version of a particular component to use, as demonstrated in the example shown in Listing B.5. This capability facilitates the process of automatic algorithm design.

Listing B.5: An example of a configuration command line for a EasyLocal++-implemented Tabu Search (TS) to the PFSP.

```
./pfsp --seed 45 --instance DD_Ta035.txt --termination
    IdleIterationsTermination --aspiration-criteria AspirationByObjective --
    stop-exploration StopExplorationBestImprovement --generator
    FullNeighborhoodGenerator --timeout 1 --max-idle-iteration 100 --tabu-list
    RangeTabuList --min-iteration-tl 2 --max-iteration-tl 9
```

Extending the Framework

We now present the perspective of a user who aims to extend the framework. Such a user can tackle the frozen spots, thus they can add a new component that can be used by an existing runner, a new runner (therefore, a new local search algorithm), or a new solver (hence, provide a way of combining different local search algorithms).

The abstract code shown in Algorithm 14 depicts the typical procedure for local search. The abstract code relies on the specification of three primary generic data types (`Input`, `Solution`, and `CostFunction`). According to the local search scheme, firstly an `InitialSolution` is computed; afterward, the algorithm enters a loop in which a move is first selected (`SelectMove`), its contribution to the cost is computed (`Compute`Δ`Cost`), and if it is suitable (`AcceptableMove`) it is performed. The procedure iterates until a termination criterion is satisfied (`Terminate`). This overall procedure is a skeleton that can be adapted to several

different metaheuristic. Indeed, different local search metaheuristics, such as HC, SA, or TS, provide distinct implementations of those functions according to their specific exploration strategy, also adding new potential ones (e.g., in TS a function for dealing with the aspiration criteria is needed, while this is not necessary for HC and SA).

For instance, let us consider the case of adding to a runner for the HC algorithm. The C++ code is reported in Listing B.6 and replicates almost *verbatim* the pseudo-code in Algorithm 14. In such a case, the new runner can be used coupled with the plethora of already encoded components or may want to add a new component to be called in an existing runner. As an example, Listing B.7 shows the implementation of a termination criterion based on the number of idle iterations. Notice that this class will be subject to static polymorphism; therefore, in the local search code, it will be checked for the proper *concept* compliance.

Algorithm 14: Abstract EasyLocal++ local search procedure.

Hotspots: A specification of the `Input` I, the `Solution` S, the definition of a `Move` M, and a `CostFunction` F

function `LocalSearch`$\langle I, S, M, F \rangle$(inst:$I$)**:**

- s_0:=`InitialSolution`$\langle I, S \rangle$(inst)
- c_0:=F(inst, s_0)
- (s^*, c^*):= (s_0, c_0)
- i:=0
- **while** ¬`Terminate`$\langle S \rangle$(s_i, i) **do**
  - m:=`SelectMove`$\langle S, M \rangle$(inst, s_i)
  - ΔF:=`ComputeΔCost`$\langle S, M \rangle$(inst, s_i, m, F)
  - **if** `AcceptableMove`$\langle S, M \rangle$(m, s_i, ΔF) **then**
    - (s_{i+1}, c_{i+1}):=$(s_i \oplus m, c_i + \Delta F)$
    - **if** $c_{i+1} < c^*$ **then**
      - (s^*, c^*):=(s_{i+1}, c_{i+1})
  - i:=$i + 1$
- **return** s^*, c^*

Listing B.6: Implementation of a generic Hill Climbing (HC) scheme.

```
void HillClimbing::Go(shared_ptr<const Input> in) {
  // current_solution_v is a lazy cost structure bridging the solution and its
      cost components
  current_solution_v = sm->SolutionValuePtr(sm->InitialSolution(in));

  while (!termination.terminate(this)) {
      // analogously current_move_v stores the solution, the move, and the $\
          Delta$costs
      current_move_v = nhe->MoveValuePtr(select_move.select(this));
    if (accept_move.accept(this)) {
      // apply the move and store the new solution
      *current_solution_v = *current_move_v;
      idle_it = 0;
    } else
      idle_it++;
```

```
    it++;
  }
  // copy to the final solution
  final_solution_v = make_shared<SolutionValue>(*current_solution_v);
}
```

Listing B.7: Implementation of a termination criterion based on idle iterations.

```
template <RunnerIdleIterT Runner>
class IdleIterationsTermination : public Parametrized {
public:
  // <...>
  bool terminate(Runner* r) {
    return r->idle_it > max_idle_it;
  }
protected:
  size_t max_idle_it;
};
```

B.3.2 Off-shelf Local Search Algorithms

While extending the framework with the implementation of new metaheuristics is relatively straightforward, EasyLocal++ is already equipped with a portfolio of local search, related variants, and components. They range from simple forms of HC strategies, like steepest descent, first descent, and random descent, to more complex metaheuristics. For example, it includes different versions of Late Acceptance Hill Climbing (LAHC) [68], Iterated Local Search [317], and TS [200]. As an example of components, one can equip a TS choosing among different tabu list implementations (e.g., variable length tabu list, frequency-based tabu status).

The metaheuristic supported with the largest number of different versions is SA, given that it has been employed in most of our research work. It has provided state-of-the-art results for many optimization problems. In detail, besides the classic version of SA [256], we introduced the *cut-off* mechanism that speeds up the early stages of the search and a reheating mechanism that restarts the annealing procedure. In addition, we propose several termination criteria in the form of components, which are based on the number of iterations, the final temperature, and the timeout.

All the available metaheuristics are entirely implemented at the framework level. To be used, they require only the definition of a single object of the suitable type and suitable template instantiations.

B.3.3 Multi-neighborhood

One of the most remarkable features of EasyLocal++ is the possibility of synthesizing composite neighborhoods starting from the implementation of basic ones. This feature allows the development of multi-neighborhood search methods, which have proven very successful in various applications.

In detail, given two or more `NeighborhoodExplorer` classes, EasyLocal++ generates automatically, at compile time, the neighborhood obtained by their composition. The simplest,

though the most effective, form of composition is the union of the neighborhoods, although in EasyLocal++ v.3.3 also, other forms, such as cartesian product and move chains, are possible. For example, when a union of different neighborhoods is defined, the best move is selected by enumerating all moves of all basic neighborhoods. In the case of a local search approach based on random moves, the current move is obtained by selecting, in turn, the basic neighborhood and the specific move inside that neighborhood. In this case, the selection of the basic neighborhood is not necessarily made with a uniform probability. On the contrary, it is often more effective to assign different probabilities and bias the search toward specific directions, but without giving away the possibility of making different move types occasionally. This mechanism is advantageous when some neighborhoods are computationally more expensive than others or when some neighborhoods are more suitable for diversifying the search. In contrast, others improve the value of the cost function. As for any other algorithm parameter, these probabilities can be exposed as parameters and thus tuned.

Finally, EasyLocal++ includes the option of adjusting online the rates of the selection of the different neighborhoods so that they do not need to be tuned in advance and may adapt to the different stages of the search process. Specifically, the probabilities of each neighborhood are updated using the recency-weighted average bandit algorithm, with a reward function that involves both the relative improvement in the objective function and the computational cost of neighborhoods. Recently, this feature has been exploited by Ceschia et al. [82] to obtain state-of-the-art results on two different timetabling problems.

B.4 Features

This section provides an overview of the features and characteristics of EasyLocal++, highlighting their evolution over the years. First, in Section B.4.1, we describe the language evolution until C++ 23. Then, in Sections B.4.2 to B.4.5, we report how solvers can be used within EasyLocal++. Finally, in Section B.4.6, the support EasyLocal++ offers w.r.t. multi-criteria cost function evaluation.

B.4.1 Evolution toward C++23 Compatibility

EasyLocal++ has been developed in C++, following the language evolution over the years, and its last updates (e.g., usage of co-routines in the neighborhood generation) use C++ 23 characteristics.

C++ offers several advantages for metaheuristics. As an object-oriented language, C++ naturally promotes modularity, reusability, and maintainability, all essential characteristics for metaheuristic implementations [450]. Additionally, it features powerful template metaprogramming capabilities, enabling compile-time code generation and optimization, which helps implement generic algorithms and data structures with high performance and flexibility (e.g., in multi-neighborhood automatic synthesis). Moreover, as a compiled language, C++ is known for its speed, which allows performing intensive computations efficiently. This is advantageous for local search algorithms, which often require numerous iterations and exploration of possibly large neighborhoods. Lastly, C++ benefits from quite a large community of users; consequently, developers can rely on a wealth of online resources,

potentially providing valuable support for their projects.

As mentioned, EasyLocal++ has evolved alongside the advancements in the C++ language. As for its latest version (i.e., EasyLocal++ v.4), the implementation now utilizes static polymorphism instead of classical virtual function polymorphism for hot spots, which is now fully supported in newer C++ versions. This shift was motivated by the availability of C++ *concepts*, which allows for checking complex constraints in template instantiation.

The codebase has also been re-engineered to embrace functional-style programming, improving code organization and readability. Notably, neighborhood exploration now employs C++ 23 *coroutines*, enabling a streamlined approach to move generation (i.e., all the code for move generation is now contained in a single function, whereas before, a complex iterator structure was necessary). Furthermore, EasyLocal++ v.4 implements lazy evaluation for cost function computation, ensuring that cost components will be computed only when needed and cost values are cached to prevent unnecessary recomputation. For instance, in the context of MOO within the Pareto front, if no other solution dominates a specific cost component of a given solution, there is no immediate need to compute the values of the other components for that solution.

B.4.2 Benchmarking

Using a framework like EasyLocal++ standardizes the underlying code and data structures used to describe problems, enabling the comparison of the performance of various metaheuristics on a uniform basis. This standardization mitigates potential confounding factors from different implementations, thus ensuring a more accurate and fair performance comparison, as advocated by Swan et al. [450].

Specifically, EasyLocal++ promotes the comparability in two ways. On the one hand, the user can experiment with different algorithms tailored to the problem at hand without re-implementing problem-specific modules. This flexibility allows for efficient exploration of algorithmic variations while maintaining consistency in problem representation. On the other hand, it allows testing the same algorithm across diverse problem sets, ensuring that the underlying local search procedure remains consistent throughout. This capability enhances confidence in the reliability and generalizability of algorithmic results across different problem domains.

B.4.3 Automated Algorithm Design

One source of inspiration for EasyLocal++ is the Programming by Optimization (PbO) manifesto [222]. Such a paradigm advocates for a shift of perspective in software development, where the design of components is approached through optimization. This involves a systematic exploration of design alternatives toward the selection of an optimal configuration of the different components through the use of automated tools (e.g., `Irace` [316]). PbO is organized into five levels (numbered from 0 to 4). The first two levels deal with existing software and expose just parameters and constants (a.k.a. magic numbers). The upper levels (from 2 to 4) advocate a tighter integration of PbO into software design.

Until EasyLocal++ v.3.3, the only design choices exposed were algorithmic parameters (e.g., in TS the length of the tabu list and the idle iterations in the termination criteria) in compliance with levels 0 and 1 of PbO. Starting with EasyLocal++ v.4, following Franzin

and Stützle [183], we do not consider local search algorithms as monoliths anymore. Instead, the users can choose among different options for their design choices (i.e., components layer in Figure B.2), which are exposed for the automated design tool. These design choices are resolved at compile time, meaning no computational overhead is added at running time. Therefore, a panoply of algorithms with a meaningful combination of the design alternatives can be generated, and the relevant one can be directly selected through the Command Line Interface (CLI). For instance, when using HC, the user can choose which type of termination criterion to use (e.g., *number of idle iterations*, *maximum number of iterations*, *timeout*).

B.4.4 Testers

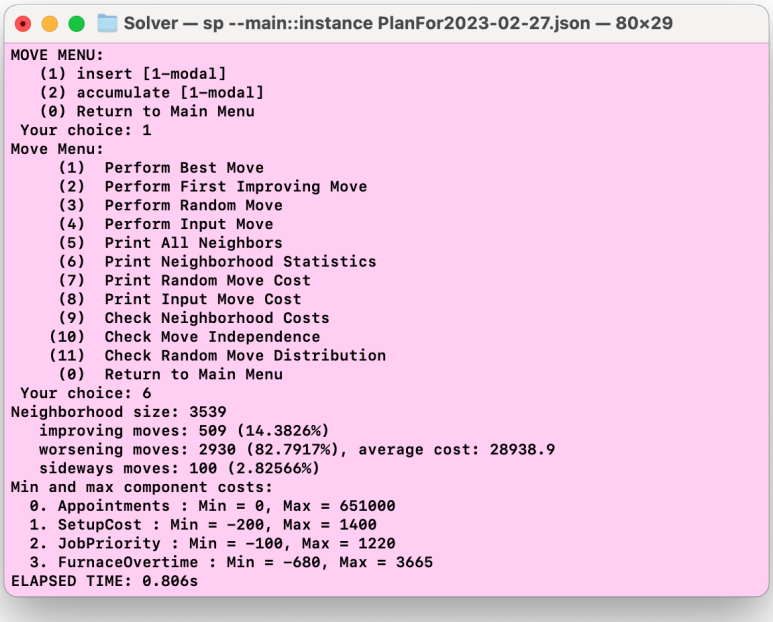


Figure B.3: A screenshot of the text-based user interface provided by testers.

In addition to the CLI, the testers enable the automatic creation of a *text-based* user interface designed to facilitate multi-level user interaction with the software. This interface increases the software usability by supporting the debugging of problem-specific code, such as verifying the consistency of incremental evaluations within cost components. Moreover, it allows for preliminary analyses of helpers, including neighborhood statistics (Figure B.3). Also, the interface enables direct experimentation with various solution methods in a sequential manner.

B.4.5 Optimization as a Service

An alternative mode of interacting with solution methods, which arose while collaborating with industrial partners, involves using a REST API. In particular, EasyLocal++ solvers

and helpers are encapsulated within an automatically generated REST HTTP interface, which enables their deployment as network-accessible services. Interactions with the API endpoints occur through the exchange of JSON-formatted files.

For instance, Listing B.8 displays the main REST endpoint that provides information about the current state of a running system. Specifically, it shows the availability of a total of three solution methods (i.e., HC, TS, and SA), two different types of neighborhoods (i.e., accumulate and insert), and the execution status of a recently executed optimization task (i.e., the task, with the given identifier, regards the execution of a HC and is finished).

Listing B.8: An example of main REST endpoint.

```
{
   "runners": ["/runner/HC", "/runner/TS", "/runner/SA"],
   "neighborhoods": ["/move/accumulate", "/move/insert"],
   "tasks": [ {
      "finished": true,
      "run_id": 55286,
      "runner": "HC"
      } ]
}
```

Listing B.9 shows the outcome of the optimization task, which can be inspected through an additional REST request.

Listing B.9: An example of task endpoint.

```
{
   "cost": {
       "components": {
           "Appointments": 0,
           "FurnaceOvertime": 0,
           "JobPriority": 28,
           "SetupCost": 7
            },
        "objective": 35
     },
    "finished": true,
    "run_id": 55286,
    "runner": "HC"
}
```

This approach shifts the focus from distributing the code to providing a quickly deployable optimization service and opens the possibility of building web applications for the interaction with the solution methods.

B.4.6 Multi-criteria Cost Function Evaluation

EasyLocal++ supports multi-criteria cost function evaluation in various forms, including aggregation (i.e., single-objective), hierarchical (i.e., lexicographic), and Pareto optimization. The framework accommodates single-cost components, which can be provided and integrated according to the desired optimization strategy.

Additionally, the implementation of lazy cost evaluation ensures that costs are computed only when necessary. The code in Listing B.10 shows an example of this behavior: the access operator to a cost component of a `SolutionValue` element computes its cost only when it is accessed and, eventually, caches its value.

Listing B.10: Lazy cost computation

```
T SolutionValue::operator[](size_t i) const {
  auto& val = const_cast<pair<bool, T>&>(this->at(i));
  if (!val.first) {
    val.second = cs->ComputeCost(sol, i);
    val.first = true;
  }
  return val.second;
}
```

B.5 Connection with Hyflex

EasyLocal++ components and algorithms can be used as Low Level Heuristics (LLHs) in `HyFlex`, a prominent framework for the development of Hyper-Heuristics (HHs).

The overall architecture is presented in Figure B.4. The integration of the `C++` code into the `HyFlex` framework is facilitated through a templated base class, `ELProblemDomainTemplate`, which builds upon EasyLocal++ components tailored to specific local search specifications. These specifications are identical to those used to instantiate a concrete algorithm in EasyLocal++, ensuring that the same computational logic applies across different algorithmic approaches. To expose these components to `HyFlex`, the class provides a concrete `HyFlex ProblemDomain` through its adaptation using the generic `ELProblemDomainAdapter java` class. This class acts as an adapter, enabling type conversions between `java`-based HHs and `C++`-implemented LLHs, which are instances of the `ELProblemDomainProvider` class.

On the `C++` side, the `ELProblemDomainTemplate` implements all the features required by the `HyFlex` framework to bridge the domain barrier of HHs (see also Figure 6.1 in Chapter 6). These functionalities include managing the inventory of problem instances and LLHs, initializing solutions, converting them to a textual representation, comparing solutions, and computing solution costs. Also, it supports the most crucial operation, i.e., the application of a specific LLH to a given solution. These capabilities are implemented using the local search metaheuristic concrete problem specifications for the specific optimization problem. For instance in Chapter 6, the LLHs are implemented as different Fixed Temperature Simulated Annealing (FTSA). However, the architecture is designed to support the easy inclusion of additional local search-based LLHs without requiring modifications.

On the `java` side, the integration relies on the `ELProblemDomainFactory`, a factory class responsible for loading the native `C++` classes for a specific problem domain via the `JNI` interface and creating instances of `ELProblemDomainAdapter` to interface with `HyFlex`.

The compiled `C++` native code is exposed to `java` as a `JNI` library. The JNI interface and the required `C++/java` wrappers are generated using `SWIG`, an interface compiler that connects `C++` programs with multiple languages to simplify development. This approach ensures efficient data exchange between the two languages.

The proposed architecture is guided by three primary objectives:

1. **Consistency:** It maintains uniform data structures and problem representations across algorithms. This design choice ensures fairness in performance evaluation by eliminating discrepancies in implementation details that could bias comparisons.

2. **Efficiency:** The use of natively compiled C++ code offers substantial speed advantages over JVM-based implementations, which is especially beneficial in local search contexts where performance bottlenecks can arise on large-scale instances.

3. **Modularity and extensibility:** The modular design allows easy integration of new problem domains. By leveraging template instantiation, the addition of new specifications requires only minor changes, primarily to instantiate the relevant C++/java bridging components, once the metaheuristic implementation is available.

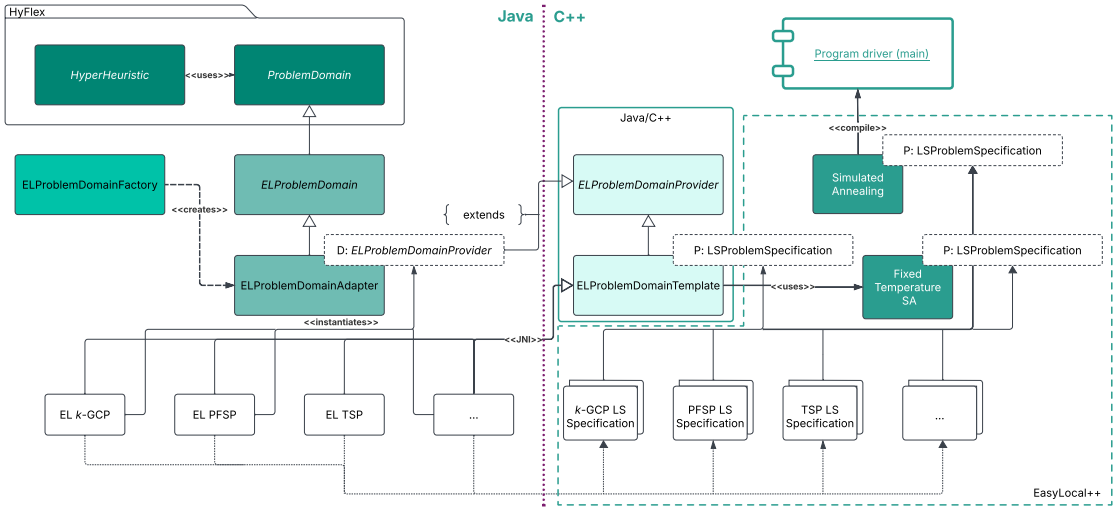


Figure B.4: Architecture of the HyFlex-EasyLocal++ bridge specified using UML notation.

B.6 EasyLocal++ for Teaching

Since 2010, EasyLocal++ has played a significant role in teaching optimization within the *Advanced Scheduling Systems* course for the Master's Degree Program of Industrial Engineering at the University of Udine.Over 13 years, around 200 students have received instruction on local search algorithms thanks to EasyLocal++. Students engage with the practical design and implementation of solution methods to real-world scheduling, timetabling, and routing problems. They are also encouraged to create their exam projects using EasyLocal++ to tackle specific optimization problems. The structured modularity and clear conceptual framework of EasyLocal++ have proven beneficial, particularly for students without a strong Computer Science background. These students have successfully implemented various techniques with reasonable programming efforts, allowing them to test and compare solutions across available instances.

B.7 Conclusion

We have described EasyLocal++, the white-box C++ framework for local searcg algorithms that our research group has developed since 1999.

Through its evolution to version 4, EasyLocal++ has consistently kept its foundational principles, including transparency to the user, modularity, maintainability, and readability, together with the separation of problem-specific and problem-independent classes. These principles have remained unchanged over its 25-year history.

The evolution, the addition, and the removal of features in EasyLocal++ have been primarily driven by its application to specific problems. Such an iterative development approach has ensured that the framework remains relevant, adaptable, and effective in addressing a wide range of optimization challenges across various application domains and w.r.t. different research trends. Indeed, over the years, it has proved effective in developing solution methods for real-world applications and academic benchmark problems belonging to a wide range of domains. In particular, it currently holds state-of-the-art results for examination timetabling, frequency assignment, home healthcare routing and scheduling, and medical student scheduling problems. In addition, thanks to its modular and clear system architecture, it has been successfully used for teaching purposes in the MSc of Industrial Engineering since 2010.

Looking to the future, several ways exist to enhance the capabilities of EasyLocal++. Firstly, the research presented by Ceschia et al. [82] can be expanded to incorporate other parameters of SA toward the concept of parameter-less algorithms. Secondly, performances could benefit from parallel executions. Additionally, there is potential to explore extensions towards other paradigms akin to local search, such as Large Neighborhood Search (LNS), which currently lack a unified software framework. Moreover, introducing hybrid approaches in the framework could offer promising avenues.

On a concluding note, while our project started 25 years ago, the interest in such software tools is far from outdated, as testified by ROAR-NET (with a specific reference to Working Group 1 — Problem modeling and user experience). Indeed, one of the network's goals is to propose a transparent white-box modeling framework (and the related software implementations) for a broad range of real-world applications, bridging the gap between modeling and solving phases. These are the same goals that initially motivated the development of EasyLocal++.

Appendix C

Chapter 5: Complete List of features

In this appendix, we provide additional details on the construction of the instance space introduced in Section 5.2.1.

A Principle Component Analysis (PCA) [197] is a dimensionality reduction approach that converts high-dimensional datasets into lower-dimensional representations while maximizing the retention of variance in the original data [95]. It identifies the orthogonal directions along which the data varies the most, allowing for a more interpretable visualization and analysis of datasets. To perform the analysis, we use the `scikit-learn` Python package.

To characterize the Permutation Flowshop Scheduling Problem (PFSP) instances, we extracted a set of direct features. Some of these features are expressed as a single value, while others require aggregation. In the latter case, we report the average, standard deviation, minimum ($\min$), maximum ($\max$), and range (i.e., $\max - \min$). The features are as follows:

- Number of jobs: Total number of jobs to be scheduled.
- Number of machines: Total number of machines available.
- Processing times: Processing times of the operations, reported with aggregations.
- Machine load: Workload assigned to each machine, reported with aggregations.
- Job load: Workload associated with each job, reported with aggregations.

Appendix D

Chapter 10: Additional Material

In this appendix, we present additional material on the algorithms presented in Chapter 10 to solve the Oven Scheduling Problem (OSP). Specifically, in Section D.1, we present a decision support system that was developed to bridge the gap between research and practice by developing user-friendly dashboards. These dashboards give decision-makers easy access to our optimization tools and their results in the form of Gantt Charts. Then, in Section D.2, we provide additional analysis on Large Neighborhood Search (LNS) and, in Section D.3, we use the proposed Simulated Annealing (SA) to solve a parallal batch scheduling problem related to the OSP.

D.1 Decision Support System

The Decision Support System, represented in Figure D.1, is built upon a microservices architecture, comprising three components: a dashboard, a message-passing component, and a distributed task queue. Users engage with the dashboard microservice, the sole exposed component, responsible for orchestrating the workflow and visually presenting results. The message-passing component connects the user's actions on the dashboard and the distributed task queue. The task queue manages possibly long-running tasks, i.e., the metaheuristics process.

The Decision Support System bridges decision-makers and the solution methods through an intuitive dashboard, functioning as a user-friendly human-computer interface. Its aim is to facilitate the execution of optimization tasks for decision-makers and provide convenient access to results. We display a few examples of the core functionalities of the interface. Figure D.2a reports an example of the LNS-related dashboard. The decision-maker is asked to select or upload an instance of the problem and to provide a timeout for the experiment. By pressing the Run LNS button, the execution starts. Eventually, the Gantt Chart of the batched solution and a set of indicators (i.e., the cost function components) are displayed. Moreover, Figure D.2b reports the analysis of the instance, displaying its main characteristics. The system is implemented in `Python` and the dashboard component leverages the `Plotly Dash` framework[1] for interaction and visualization.

[1] https://dash.plotly.com/. Accessed 2025–10–10

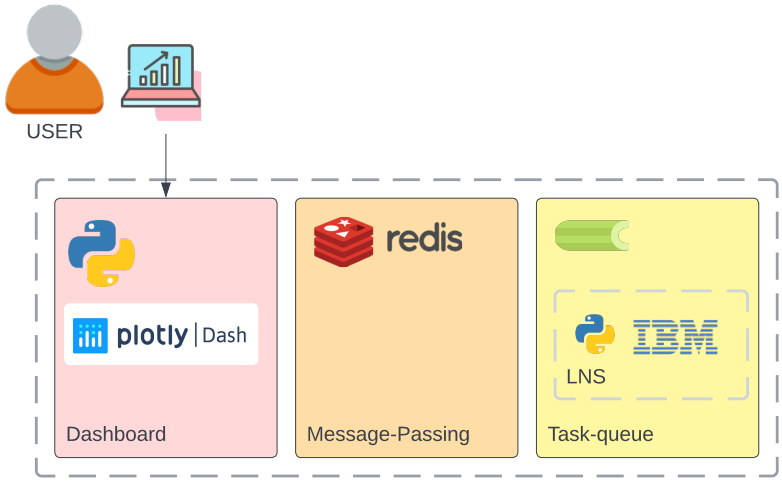
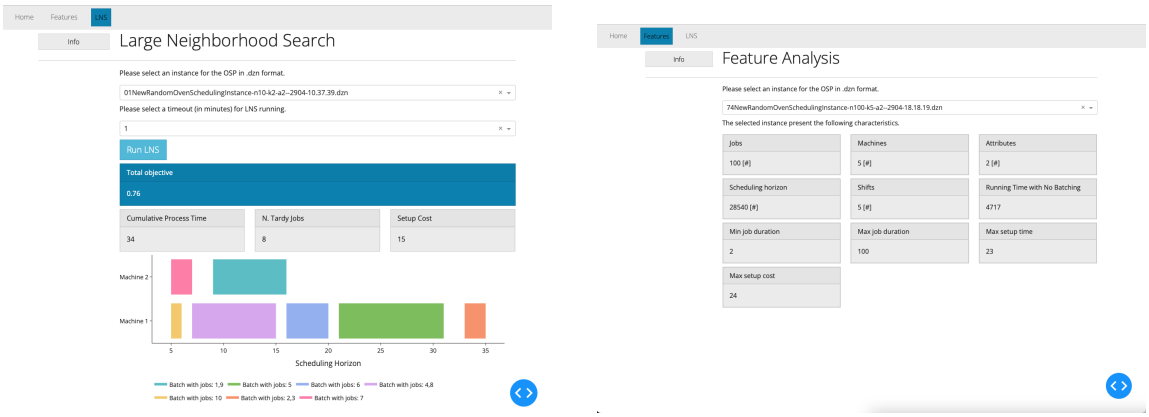


Figure D.1: Decision Support System architecture to interface with the solvers developed for the Oven Scheduling Problem (OSP).



(a) LNS dashboard. **(b)** Instance feature analysis

Figure D.2: Example of dashboard functionalities.

D.2 Additional Analysis for LNS

In this section, we provide additional analyses related to LNS. The objective is to compare LNS with exact methods under equivalent computational budgets. Specifically, when reporting the best result out of five runs of LNS, we allocate an equivalent five hours of runtime to the exact method. We acknowledge that this choice is debatable; however, it is adopted to ensure fairness and to preempt potential criticism regarding differences in computational effort. This analysis is conducted only for LNS and not for SA, since the latter already incorporates a stagnation-based stopping criterion, and additional experiments with longer runs did not produce different results.

We aim to answer the following questions:

- To what extent can LNS provide better results for the OSP w.r.t. exact methods (especially in large instances)?
- To what extent is the initial solution improved?

- To what extent can batching reduce energy consumption in electronic manufacturing?

The remainder of this section is structured as follows. Section D.2.1 details the experimental setup. Section D.2.2 reports the results.

D.2.1 Experimental Setup

We use instances with 50 and 100 jobs from the public benchmark [275], with varying number of machines (2 or 5) and attributes (2 or 5). We omit results for smaller instances (i.e., 10 and 25 jobs) since the exact methods solve them optimally, and our LNS algorithm manages to achieve the same solution quality.

The parameters and implementation details are the same detailed in Chapter 10.

All experiments are executed on a machine featuring 2x Intel Xeon Platinum 8368 2.4GHz 38C, 8×64GB RDIMM. Each algorithm is executed on a single thread.

D.2.2 Results

We analyze three distinct aspects: firstly, we compare the results of LNS with those achieved by the best-performing exact method from the literature [275]. Secondly, we evaluate the improvement of LNS compared to the initial solution. Lastly, we assess the environmental perspective by examining the reduction in cumulative processing time (p) between a solution with batching and one without.

Comparison with Exact Methods

To benchmark the performance of LNS, we compare it against Constraint Programming (CP) warm-started by the greedy heuristics as reported in the original paper by M. Lackner et al. [275] (we refer to this method indicating it with Optimization Programming Language (OPL)-ws). To account for the stochastic components of LNS, we execute the algorithms 5 times per instance with a timeout of 1 hour, eventually selecting the minimum cost. To ensure fairness in the comparison, we run the exact method with a cutoff of 5 hours, therefore granting the same amount of time to both solution approaches.

We evaluate the quality of a LNS solution using the relative gap of the objective w.r.t. the given baseline method (OPL-ws). A negative gap value implies that the result found by LNS is better than that found by the baseline method

Figure D.3 illustrates the number of instances where LNS outperforms OPL-ws, categorized by instance size (50 and 100 jobs) and classified into 5 improvement categories. In summary, LNS outperforms OPL-ws in 18 out of 40 instances while achieving equal performance in 19 cases (15 in the 50-job instances). In 3 cases, OPL-ws outperforms LNS. Additionally, OPL-ws successfully identifies 3 optimal solutions, all of which are also discovered by our LNS approach. The statistical distribution is reported in the boxplots of Figure D.4. Interestingly, for what concerns the 100-job instances, the gap ranges between -7.5% and and 1.16%, with an average of -1%

We also run the experiments considering OPL without the warm start; however, since the results do not differ substantially from OPL-ws, we do not report them here.

We assess the statistical significance of these results. Initially, we conduct a Friedman test (with $\alpha = 0.05$) to ascertain if algorithms exhibit divergent performances. Results

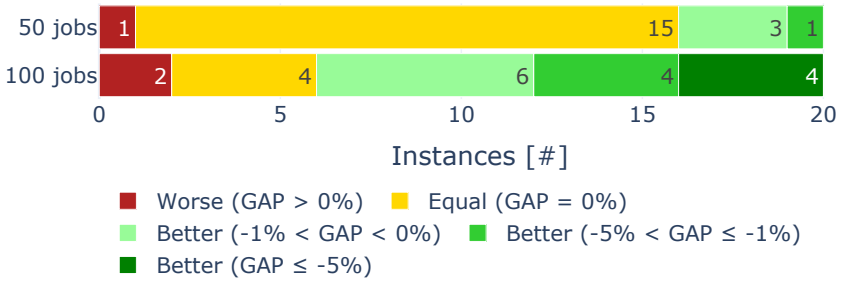


Figure D.3: Number of instances for which Large Neighborhood Search (LNS) performs better than OPL-ws, that is the Constraint Programming (CP) method, considering the instances by size. LNS is run 5 times per instance with a timeout of 1 hour, selecting the minimum cost among the runs. OPL-ws is executed with a timeout of 5 hours.

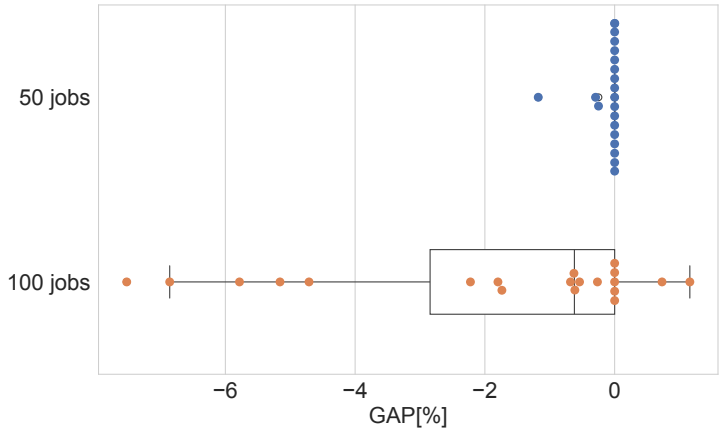


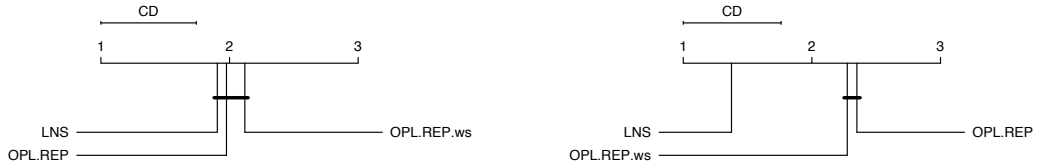
Figure D.4: Comparison of the performance gap between Large Neighborhood Search (LNS) and Constraint Programming (CP) with warm start (OPL-ws). LNS is run 5 times per instance with a timeout of 1 hour, selecting the minimum cost among the runs. OPL-ws is executed with a timeout of 5 hours.

indicate that for instances with 50 jobs, the performance of LNS is not significantly different from that of exact methods. Conversely, for instances with 100 jobs, the null hypothesis is rejected (p-value=0.001), indicating that the algorihtms perform differently.

Subsequently, we employ the Nemenyi post-hoc test and generate a Critical Difference (CD) plot. Figure D.5a presents the results for instances with 50 jobs, where differences are not statistically significant, although LNS tends to perform slightly better than exact methods. In contrast, for instances with 100 jobs, the CD plot reveals two distinct groups of algorithms: LNS occupies a rank close to 1, while exact methods form a separate group.

Improvement w.r.t. the Initial Solution

We examine the enhancement brought about by LNS in comparison to the initial solution. This analysis aims to showcase that while the initial solution may serve as a decent starting point, our methodology can significantly enhance these outcomes. We compute the relative gap considering the initial solution as the baseline and present the outcomes in Figure D.6. Notably, LNS can improve the objective value of the initial solution provided by the greedy



(a) Instances with 50 jobs. **(b)** Instance with 100 jobs.

Figure D.5: Critical Difference plots.

heuristic for all instances with 50 or 100 jobs. On average, the solution quality is improved by 20%; the improvement can be up to 50%.

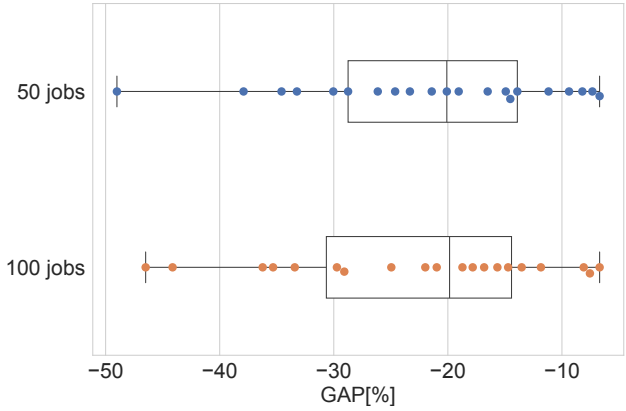


Figure D.6: Comparison of the performance gap between Large Neighborhood Search (LNS) and the initial solution. LNS is run 5 times per instance with a timeout of 1 hour, selecting the minimum cost among the runs.

Energy Saving Perspective

To highlight the potential of batching the jobs and optimizing their schedule, we compare the cumulative processing time (p) of the solutions found by LNS with solutions that do not present aggregation of jobs. When no batching is applied, p corresponds to the sum of the minimum processing time of each job (i.e., each job constitutes a batch on its own).

We use the gap taking as a baseline the case with no batching. Figure D.7 illustrates the outcomes: across all instances, batching is feasible and consistently leads to superior results, as evidenced by the reduced oven runtime. The reduction of oven runtime, and thus the reduction of energy consumption, ranges from a minimum of approximately 20% to a maximum of 83%, with an average of 55% and a standard deviation of 18%.

We repeat the experiment, considering the cumulative processing time of the solutions found by LNS alongside those found by the initial solution. Also in this case, we consider the gap. Figure D.8 illustrates that, also in this case, the energy reduction granted by the solutions found by LNS is more advantageous than that retrieved by the initial solution in terms of energy savings (ranges from a minimum of approximately 7% to a maximum of 49%, with an average of 22% and a standard deviation of 11%). However, such a result is expected since the objective function depends on components evaluated based on their

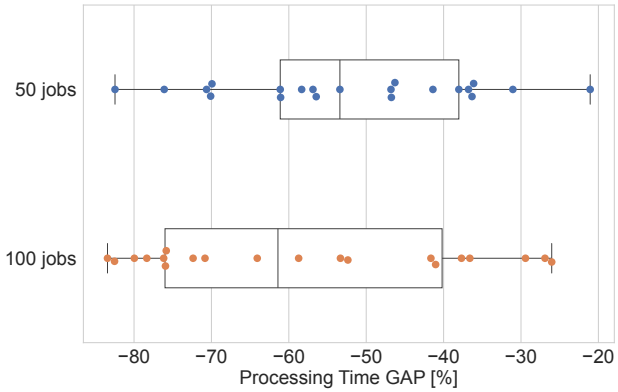


Figure D.7: Gap between the cumulative processing time of the solution retrieved by LNS (solution with batching) and the solution without batching. LNS is run 5 times per instance with a timeout of 1 hour, selecting the minimum cost among the runs.

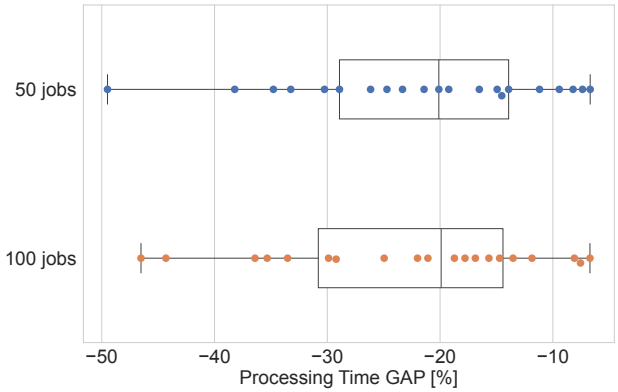


Figure D.8: Gap between the cumulative processing time of the solution retrieved by Large Neighborhood Search (LNS) and the solution provided by the initial solution. Both solution techniques involve batching. LNS is run 5 times per instance with a timeout of 1 hour, selecting the minimum cost among the runs.

lexicographic importance, and the processing time p is the most crucial component.

D.3 Robustness of SA for Parallel Batch Scheduling Problem

Given the good performance the proposed SA algorithm delivers for the OSP, this section examines whether it can deliver similarly good results for another related problem. The investigation considers the parallel batch scheduling problem tackled by Damodaran and Vélez-Gallego [125], since its nature is very similar to the OSP. Firstly, they both require aggregating jobs in batches and devising an optimal schedule for these batches. Secondly, both problems arise in electronic component manufacturing; in fact, Damodaran and Vélez-Gallego [125] stated that the problem was faced in a small electronics manufacturing facility that required assembling and testing printed circuit boards. Additionally, the parameters of the instance share some characteristics, the presence of different job sizes, release dates, etc. The formulations however distinguish from each other in several ways.

First, this problem accounts only for one objective (different from the ones tackled by the OSP), which is the makespan. Furthermore, the OSP and its instances offer many more real-world details, for example, the presence of setup times and costs, machine availabilities, machines with different sizes, etc.

Damodaran and Vélez-Gallego [125] solved the problem employing a SA algorithm employing two moves (swapping jobs between batches and moving one job to an existing batch, similarly as MoveJob). As mentioned in Section 3.4, the proposed SA distinguishes in several ways including the neighborhood composition, the cutoff mechanism, etc.

The remainder of this section is structured as follows. Section D.3.1 introduces the problem at hand. Section D.3.2 details the experimental setup we employed. Section D.3.3 reports the results. Finally, Section D.3.4 draws some conclusions.

D.3.1 Problem Description

An instance of the problem comprises a set $\mathcal{J}$ of n jobs that need to be processed by a set $\mathcal{M}$ of m identical machines arranged in parallel. Each job $j \in \mathcal{J}$ is characterized by a processing time p_j, a release date r_j, and a size s_j. All machines have the same size S and are considered to be always available. Jobs can be aggregated in batches, as long as the overall size of the batch does not exceed S. The processing time of the batch is equal to the maximum processing time of all its jobs and can start being processed only after all its jobs have been released. The objective is to find a schedule which minimizes the makespan.

D.3.2 Experimental Setup

We now present the experimental setup employed to compare the solution methods, including details on the instances, the parameter tuning procedure, and the technical details. Note that the implementation of Damodaran and Vélez-Gallego [125] is indicated as DA.

Instances

The benchmarks set comprises the original 600 instances from Damodaran and Vélez-Gallego [125]. They are characterized by the number of jobs (10, 20, 50, 100, or 200 jobs), number of machines (3 or 5), maximum job processing time (10 or 30), and maximum job size (5 or 20). For simplicity, they are categorized based on the number of jobs: small instances (10 or 20 jobs), medium instances (50 or 100 jobs), and large instances (200 jobs).

To make the instances compatible with our implementation of SA for the OSP, some assumptions regarding the missing values are made. For example, the maximum processing time of each job is equal to its minimum processing time, setup times and costs are always equal to zero, etc.

Parameter Tuning

The proposed SA is tuned using `Irace` v.3.5 [316]. The procedure is granted with a total of $30,000$ experiments and the SA equipped with a stopping criterion of $i_{\max} = 250,000$ iterations. The final temperature is fixed ($T_f = 0.001$) after a set of preliminary experimental trials. Table D.1 reports the tuning parameters and final values. It is important to note that

a re-tuning in comparison to the previous experiments is needed, as this problem has a different objective and uses instances that exhibit different characteristics.

Table D.1: Parameter tuning for the multi-neighborhood Simulated Annealing to solve the parallel batch scheduling problem formulated by Damodaran and Vélez-Gallego [125].

Name	Description	Range	Value
T_0	Start Temperature	50–200	69
α	Cooling rate	0.98–0.99	0.98
ρ	Neighborhood accepted ratio	0.10–0.40	0.15
σ_{S}	Bias of Swap neighborhood	0.00–1.00	0.22
σ_{NUB}	Bias of NewUnaryBatch neighborhood	0.00–1.00	0.13
σ_{NB}	Bias of NewBatch neighborhood	0.00–1.00	0.21
σ_{MJ}	Bias of MoveJob neighborhood	0.00–1.00	0.44

Technical Details

The experimental setup and code are the ones used for the previous experiments (Chapter 10), with the minor modification related to the insertion of a new objective function (makespan objective). Apart from this, the implementation is not further modified. This means that we check for more conditions than necessary (for example, we always check machine availabilities even though they are available throughout the entire horizon).

Since the work by Damodaran and Vélez-Gallego [125] dates back to the year 2012, the authors' original code is run on the same machine of the other experiments to ensure a fair comparison. DA is implemented in `MATLAB` and runs using version 24.2.0.2712019 (R2024b).

Regarding the stopping condition, the proposed SA considers the same number of iterations considered by the work of Damodaran and Vélez-Gallego [125], that is scaled w.r.t. number of jobs (i.e., $500 \cdot n$). The running time between SA and DA are comparable w.r.t. small (SA runs for an average time of 1.4 seconds and DA for 1.01 seconds) and medium-sized instances (SA runs for an average time of 4.32 seconds and DA for 4.59 seconds). On the contrary, for larger instances, our SA (average running time of 12.27 seconds) takes significantly less time than DA (average running time of 45.42 seconds). For this reason, SA is run on the larger instances for such an amount of time (i.e., $\mathcal{I} = 95,000$, with an average running time of 46.05 seconds). This version of SA is later indicated with SA-t. DA is not equipped with a stopping criterion of $\mathcal{I} = 1,000,000$, since preliminary experiments demonstrated that for even small instances the running times were too large w.r.t. the running times of SA (average running time of 10 minutes for DA and average running time of 4.2 seconds for SA).

D.3.3 Results

Each algorithm is run 10 times per instance; we then consider the mean value across the runs. The Friedman test confirms that the two algorithms exhibit significantly different performances and with SA delivering superior results, particularly for small and medium-sized instances. For what concerns large instances, the three versions (SA, SA-t, and DA) perform differently, with SA-t performing better than the other two (average ranking of 1.4 for SA-t, 2.1 for SA, and 2.5 for DA).

The gap is calculated considering as a baseline the solution found by the algorithm of Damodaran and Vélez-Gallego [125]. Table D.2 reports the comparison considering the number of instances. In the majority of the cases, SA produces superior (418 instances with SA and 433 instances if large instances use SA-t) or equal (136 instances with SA and 138 instances if large instances use SA-t) results than the competitor. However, for large instances, in some instances, the algorithm by Damodaran and Vélez-Gallego [125] provides better results. On average, the gap is -3.14% (± 3.77) for small instances, -4.62% (± 5.13) for medium instances, -3.56% (± 5.11) for large instances with SA, and -3.56% ($\pm 5.12\%$) for large instances with SA-t.

Table D.2: Comparison of our Simulated Annealing (SA) w.r.t. the algorithm by Damodaran and Vélez-Gallego [125] considering instance size.

Instance size	Total	Gap				
		$[-\infty;-1]$	$(-1;0)$	$=0$	$(0;1)$	$[1;+\infty]$
Small	240	132	29	79	0	0
Medium	240	142	42	44	5	7
Large	120	46	27	13	18	16
Large (SA-t)	120	61	27	15	10	7

In the original paper, Damodaran and Vélez-Gallego [125] also provided lower bounds to the instances. Considering the minimum across the runs, SA can achieve these bounds 369 times (377 considering SA-t for the larger instances), while DA does so 362 times.

D.3.4 Concluding Remarks

The comparison with the existing literature reaffirms the superiority of the proposed approach. Not only do the results outperform those of previous methods, but they also show the robustness of the algorithm. When tested on a related parallel batch scheduling problem, the SA method significantly improved solutions for many instances, outperforming, on the average, results reported by the original methodology [125].

Appendix E

Chapter 11: Complete List of features

In this appendix, we report the list of the 165 features that describe the instances of the Oven Scheduling Problem (OSP) and that have been used in Chapter 11 in the context of Instance Space Analysis (ISA) and algorithm selection. Please refer to the notation introduced in Section 8.2 for decoding the OSP parameters.

While some features are represented by a single value, many others require an aggregation over a set of values. In such cases, the following statistical figures are calculated: the minimum value ($\min$), the maximum value ($\max$), the range between maximum and minimum value ($range = \max - \min$), the mean value ($mean$), the first quartile ($q1$), the second quartile ($q2$), the third quartile ($q3$), and the standard deviation (std).

The remainder of this section is structured as follows. Section E.1 reports the list of direct features. Section E.2 details the features related to the Lower Bounds (LBs). Finally, Section E.3 considers graph-based features.

E.1 Direct Features

Direct features are features that can be directly extracted from the instances or require only minor calculations. These are:

- The number of jobs (n).
- The number of machines (k).
- The number of operations, calculated as $n \cdot k$, and its counterpart n/k.
- The skewness, calculated as $\max(n/k, k/m)$.
- The number of attributes (a).
- The horizon length (l).
- The number of shifts ($\mathcal{I}$).
- The shift durations, i.e., the duration of shift $i \in \mathcal{I}$ of machine $m \in \mathcal{M}$ is calculated as $\sigma^{\text{end}}_{m,i} - \sigma^{\text{start}}_{m,i}$. The set of statistical figures is calculated.

- The working time proportion, i.e., the ratio between the sum of the processing time of the jobs, i.e., $\sum_{j \in \mathcal{J}} \lambda_j^{\text{min}}$, and the total available time of all the machines, i.e., $\sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{I}} (\sigma_{m,i}^{\text{end}} - \sigma_{m,i}^{\text{start}})$.
- The maximal capacity of the machines (c_m). The set of statistical figures is calculated.
- The job sizes (s_j). The set of statistical figures is calculated.
- The number of jobs per attribute, i.e., given an attribute $v \in \mathcal{A}$ the following counting operation is performed $\sum_{j \in \mathcal{J}} [\alpha_j = v]$. The set of statistical figures is calculated.
- The number of jobs per machine considering the eligible machines of the jobs, i.e., given a machine $m \in \mathcal{M}$ the following counting operation is performed $\sum_{j \in \mathcal{J}} [m \in \mathcal{E}_j]$. The set of statistical figures is calculated.
- The number of eligible machines per job, i.e., given a job $j \in \mathcal{J}$, it is the cardinality of set $\mathcal{E}_j$. The set of statistical figures is calculated.
- The job release (τ_j^{start}) and due dates (τ_j^{end}). The set of statistical figures is calculated.
- The minimal (λ_j^{min}) and maximal processing times (λ_j^{max}) of the jobs. The set of statistical figures is calculated.
- The setup costs (SC) and times (ST). The set of statistical figures is calculated.
- The type of setup matrix, i.e., constant, arbitrary, realistic, or symmetric.
- The spread of earliest start times (ρ), as per genetator input.
- The time from the earliest start to the latest end of the job (ϕ). This value is an input value for the random generator.
- The combination of the weight and normalization constant for the three components as defined in Equation (8.1).

E.2 Lower and Upper Bounds as Features

The feature set includes the theoretical LBs for the OSP formulated in Chapter 9 and associated metrics:

- Lower bound on the number of batches.
- Lower bound on the number of tardy jobs (t).
- Lower bound on the total setup cost (sc).
- Lower bound on the processing time (p).
- Upper bound on the processing time, corresponding to the sum of the minimal processing time of the jobs (i.e., a solution with no batching).
- Upper bound on the reduction of processing time.
- Upper bound on the average number of jobs per batch.
- Time required to compute the lower bounds.

E.3 Graph Features

Graph features are based on an undirected graph representing job compatibility. In this graph, each node corresponds to a job, and arcs connect pairs of nodes (jobs) that can be processed together under the following conditions: *(i)* They share the same attribute. *(ii)* They can be processed on at least one common machine. *(iii)* Their processing times are compatible. *(iv)* Their release and due dates are compatible. Given that job tardiness is already considered in the objective function obj, we also introduce a relaxed version of the graph where condition *(iv)* is omitted.

For each graph G (the complete and relaxed version), we extract these features:

- The total number of edges of G, indicated as the set E. Note that, on the other hand, we do not consider the number of nodes, as this equals the number of jobs.
- The density of G calculated as $dens(G) = \frac{2|E|}{|V|(|V|-2)}$, where V is the set of nodes. Key statistical figures are calculated for $dens(G)$.
- The number of edges incident to each node v, also called degree of a node ($d(v)$). Key statistical figures are calculated for d.
- The number of isolated nodes, i.e., nodes with $d = 0$.
- The shortest path betweenness centrality (bc), that is the measure of how often each node v acts as a bridge along the shortest paths between two other nodes, normalized by the number of vertex pairs. It is calculated as $bc(v) = \sum_{s \neq t \in V \setminus \{v\}} \frac{\sigma(s,t|v)}{\sigma(s,t)}$, where $\sigma(s,t|v)$ is the number of shortest paths from s to t passing per v, and $\sigma(s,t)$ is the number of shortest paths from s to t. The normalization value is as follows ($\frac{(|V|-1)(|V|-2)}{2}$). Key statistical figures are calculated for bc.
- The size of the largest maximal clique in G.
- The number of connected components (subgraphs) in G.

Appendix F

Chapter 14: Complete List of features

In this appendix, we report the list of features that describe the instances of the Home Healthcare Routing and Scheduling Problem (HHCRSP) and that have been used in Chapter 14 to compare the newly generated instances with those existing in the literature. As in the case of the Oven Scheduling Problem (OSP, Appendix E), some features are naturally represented by a single value, while many others require an aggregation over a set of values. The aggregations are the same of the OSP.

The identified features are:

- Total number of patients.
- Number of patients requiring one service, two services, more than two services.
- Number of patients requiring simultaneous, sequential, and independent services.
- Number of optional patients.
- Number of patients expressing preferences toward caregivers.
- Number of patients expressing incompatibilities toward caregivers.
- Total number of required services.
- Number of types of services.
- Number of time windows per patients (aggregation required).
- Total number of time windows.
- Length of time windows (aggregation required).
- Sum of the length of all time windows.
- Length of service durations (aggregation required).
- Number of preferred caregivers per patient (aggregation required).
- Number of incompatible caregivers per patient (aggregation required).
- Occurrences of a given service (aggregation required).

- Occurrences of a preferred caregiver (aggregation required).
- Occurrences of an incompatible caregiver (aggregation required).
- Total number of caregivers
- Number of caregivers that need to have a lunch break.
- Number of services a caregiver can perform (aggregation required).
- Occurrences of a service in the skills of caregivers (aggregation required).
- Length of caregivers' shifts (aggregation required).
- Total length of caregivers' shifts.
- Total number of terminal points.
- Occurrences of a terminal point as arrival or departure point (aggregation required).
- Occurrences of a given terminal point as departure point (aggregation required).
- Occurrences of a given terminal point as arrival point (aggregation required).
- Start time, end time, duration (as end time $-$ start time), and minimal duration of the lunch break
- Number of compatible caregivers per service (aggregation required).
- Overlap between the caregivers shift and patients' time window considering the skills of the firsts and the services required by the latters (aggregation required).
- Total overlap between skilled caregivers shift and patients' time window.
- Number of overlap between skilled caregivers shift and patients' time window (aggregation required).
- Total number of overlap between skilled caregivers shift and patients' time window.
- Tardiness consideration. Useful time unit to start a service without being late (aggregation required).
- Distances metrics (graph) (aggregation required).
- Number of edges below the average distance.
- Number of edges above the average distance.
- Sum of the 25 shortest edges.
- Sum of the 25 longest edges.

Appendix G

Chapter 17: Additional Material

In this appendix, we complement the systematic literature review on Large Language Models (LLMs) for Combinatorial Optimization (CO) presented in Chapter 17. First, Section G.1 reports the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) checklists, then Section G.2 lists and describe the 103 studies included in the review. Sections G.3 to G.6 analyze the studies in tabular form.

G.1 PRISMA checklists

This appendix reproduces the two official PRISMA 2020 checklists [377, 378] that guided our systematic review of Large Language Model applications in Combinatorial Optimization:

1. The *PRISMA 2020 for Abstracts Checklist*, and
2. The full *PRISMA 2020 Checklist*.

They are included here verbatim w.r.t. the original publication related to the topic for transparency and ease of reference.

PRISMA 2020 for Abstracts Checklist

Section and Topic	Item #	Checklist item	Reported (Yes/No)
TITLE			
Title	1	Identify the report as a systematic review.	Yes
BACKGROUND			
Objectives	2	Provide an explicit statement of the main objective(s) or question(s) the review addresses.	Yes
METHODS			
Eligibility criteria	3	Specify the inclusion and exclusion criteria for the review.	Yes
Information sources	4	Specify the information sources (e.g. databases, registers) used to identify studies and the date when each was last searched.	Yes
Risk of bias	5	Specify the methods used to assess risk of bias in the included studies.	No
Synthesis of results	6	Specify the methods used to present and synthesise results.	Yes
RESULTS			
Included studies	7	Give the total number of included studies and participants and summarise relevant characteristics of studies.	Yes
Synthesis of results	8	Present results for main outcomes, preferably indicating the number of included studies and participants for each. If meta-analysis was done, report the summary estimate and confidence/credible interval. If comparing groups, indicate the direction of the effect (i.e. which group is favoured).	Yes
DISCUSSION			
Limitations of evidence	9	Provide a brief summary of the limitations of the evidence included in the review (e.g. study risk of bias, inconsistency and imprecision).	No
Interpretation	10	Provide a general interpretation of the results and important implications.	Yes
OTHER			
Funding	11	Specify the primary source of funding for the review.	Within paper body
Registration	12	Provide the register name and registration number.	Not relevant

From: Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ 2021;372:n71. doi: 10.1136/bmj.n71

For more information, visit: http://www.prisma-statement.org/

PRISMA 2020 Checklist

Section and Topic	Item #	Checklist item	Location where item is reported
TITLE			
Title	1	Identify the report as a systematic review.	Title
ABSTRACT			
Abstract	2	See the PRISMA 2020 for Abstracts checklist.	Abstract, see PRISMA for Abstracts checklist
INTRODUCTION			
Rationale	3	Describe the rationale for the review in the context of existing knowledge.	Section 1
Objectives	4	Provide an explicit statement of the objective(s) or question(s) the review addresses.	Section 2
METHODS			
Eligibility criteria	5	Specify the inclusion and exclusion criteria for the review and how studies were grouped for the syntheses.	Section 5.3.3
Information sources	6	Specify all databases, registers, websites, organisations, reference lists and other sources searched or consulted to identify studies. Specify the date when each source was last searched or consulted.	Section 5.3.2
Search strategy	7	Present the full search strategies for all databases, registers and websites, including any filters and limits used.	Section 5.3.2
Selection process	8	Specify the methods used to decide whether a study met the inclusion criteria of the review, including how many reviewers screened each record and each report retrieved, whether they worked independently, and if applicable, details of automation tools used in the process.	Section 5.3.3
Data collection process	9	Specify the methods used to collect data from reports, including how many reviewers collected data from each report, whether they worked independently, any processes for obtaining or confirming data from study investigators, and if applicable, details of automation tools used in the process.	Section 5.3.4
Data items	10a	List and define all outcomes for which data were sought. Specify whether all results that were compatible with each outcome domain in each study were sought (e.g. for all measures, time points, analyses), and if not, the methods used to decide which results to collect.	Section 6 and Appendix B
	10b	List and define all other variables for which data were sought (e.g. participant and intervention characteristics, funding sources). Describe any assumptions made about any missing or unclear information.	Section 6 and Appendix B
Study risk of bias assessment	11	Specify the methods used to assess risk of bias in the included studies, including details of the tool(s) used, how many reviewers assessed each study and whether they worked independently, and if applicable, details of automation tools used in the process.	Section 5.3
Effect measures	12	Specify for each outcome the effect measure(s) (e.g. risk ratio, mean difference) used in the synthesis or presentation of results.	Not relevant
Synthesis methods	13a	Describe the processes used to decide which studies were eligible for each synthesis (e.g. tabulating the study intervention characteristics and comparing against the planned groups for each synthesis (item #5)).	Section 5.3.3
	13b	Describe any methods required to prepare the data for presentation or synthesis, such as handling of missing summary statistics, or data conversions.	Not relevant
	13c	Describe any methods used to tabulate or visually display results of individual studies and syntheses.	Section 6
	13d	Describe any methods used to synthesize results and provide a rationale for the choice(s). If meta-analysis was performed, describe the model(s), method(s) to identify the presence and extent of statistical heterogeneity, and software package(s) used.	Section 6
	13e	Describe any methods used to explore possible causes of heterogeneity among study results (e.g. subgroup analysis, meta-regression).	Not relevant
	13f	Describe any sensitivity analyses conducted to assess robustness of the synthesized results.	Not relevant
Reporting bias assessment	14	Describe any methods used to assess risk of bias due to missing results in a synthesis (arising from reporting biases).	Section 8
Certainty assessment	15	Describe any methods used to assess certainty (or confidence) in the body of evidence for an outcome.	Not relevant
RESULTS			
Study selection	16a	Describe the results of the search and selection process, from the number of records identified in the search to the number of studies included in the review, ideally using a flow diagram.	Figure 2

PRISMA 2020 Checklist

Section and Topic	Item #	Checklist item	Location where item is reported
	16b	Cite studies that might appear to meet the inclusion criteria, but which were excluded, and explain why they were excluded.	Section 5.3.4
Study characteristics	17	Cite each included study and present its characteristics.	Appendix B
Risk of bias in studies	18	Present assessments of risk of bias for each included study.	Not relevant
Results of individual studies	19	For all outcomes, present, for each study: (a) summary statistics for each group (where appropriate) and (b) an effect estimate and its precision (e.g. confidence/credible interval), ideally using structured tables or plots.	Not relevant
Results of syntheses	20a	For each synthesis, briefly summarise the characteristics and risk of bias among contributing studies.	Section 6
	20b	Present results of all statistical syntheses conducted. If meta-analysis was done, present for each the summary estimate and its precision (e.g. confidence/credible interval) and measures of statistical heterogeneity. If comparing groups, describe the direction of the effect.	Not relevant
	20c	Present results of all investigations of possible causes of heterogeneity among study results.	Section 6
	20d	Present results of all sensitivity analyses conducted to assess the robustness of the synthesized results.	Not relevant
Reporting biases	21	Present assessments of risk of bias due to missing results (arising from reporting biases) for each synthesis assessed.	Section 8
Certainty of evidence	22	Present assessments of certainty (or confidence) in the body of evidence for each outcome assessed.	Not relevant
DISCUSSION			
Discussion	23a	Provide a general interpretation of the results in the context of other evidence.	Section 6
	23b	Discuss any limitations of the evidence included in the review.	Section 8
	23c	Discuss any limitations of the review processes used.	Section 8
	23d	Discuss implications of the results for practice, policy, and future research.	Section 7 and Section 9
OTHER INFORMATION			
Registration and protocol	24a	Provide registration information for the review, including register name and registration number, or state that the review was not registered.	Not relevant
	24b	Indicate where the review protocol can be accessed, or state that a protocol was not prepared.	Not relevant
	24c	Describe and explain any amendments to information provided at registration or in the protocol.	Not relevant
Support	25	Describe sources of financial or non-financial support for the review, and the role of the funders or sponsors in the review.	End of the paper, before references
Competing interests	26	Declare any competing interests of review authors.	End of the paper, before references
Availability of data, code and other materials	27	Report which of the following are publicly available and where they can be found: template data collection forms; data extracted from included studies; data used for all analyses; analytic code; any other materials used in the review.	The paper is self contained and all the material is included either in the paper or in the appendix.

From: Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, et al. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. BMJ 2021;372:n71. doi: 10.1136/bmj.n71
For more information, visit: http://www.prisma-statement.org/

G.2 Description of Included Studies

This appendix provides the full list of the 103 studies found in the literature using the methodology described in Section 17.1 and analyzed in Section 17.3. For each study, we report its type (INCL4-EXCL4) and provide a brief description of the application of LLMs in the field of CO. When applicable, we also mention the specific Combinatorial Optimization Problem (COP) involved.

- Abdullin et al. [1] published a research study introducing a goal-oriented conversational agent to assist users in constructing accurate Linear Programming (LP) models. The LLM is used in the dialogue generation phase to automate the interaction between a conversational agent and a simulated user, effectively extracting necessary information to formulate LP models.

- AhmadiTeshnizi, Gao, and Udell [11] published a research study investigating the usage of LLM to formulate and solve LP and Mixed Integer Linear Programming (MILP) problems from Natural Language (NL) descriptions. The LLM is employed to develop an agent that recognizes entities and translates NL descriptions into code. This code is subsequently used by a solver to find solutions and potentially debug them. The work has been extended by AhmadiTeshnizi et al. [10].

- Ahmed and Choudhury [12] published a research study to investigate the usage of LLMs for recognizing entities and translating NL descriptions in optimization problem formulations. A LLM is used for this task in both zero-shot and one-shot settings.

- Alipour-Vaezi and Tsui [17] proposed a research study on using LLMs to extract domain knowledge in the context of the motion picture industry for portfolio optimization.

- Almonacid [20] published a research study investigating LLM capabilities of generating optimization code. The LLM is used to generate the model and its code using `MiniZinc`, eventually debugging it.

- Amarasinghe et al. [21] published a research study introducing a framework based on Copilot, designed to automate the generation of code from NL descriptions. The LLM is used to automate the generation of `Python` code employing the `CPMpy` library starting from business descriptions.

- Bohnet et al. [58] published a research study on using LLMs in generating solutions for planning problems.

- Borazjanizadeh et al. [60] published a research study on solving search problems using LLMs. The framework involves generating solution code and producing the solution in a specific format. The results were evaluated in terms of feasibility, correctness, and optimality. The study also introduces a new benchmark called SearchBench.

- Cai et al. [72] published a position paper on the possible advantages LLMs can bring to evolutionary computation, including evolutionary-based optimization.

- Chacón Sartori, Blum, and Ochoa [91] published a research study integrating LLMs into a web-based tool for visualizing the behavior of optimization algorithms. The LLM generates prompts for the tool, enabling users to comprehend how multiple algorithms behave when applied to specific instances of a COP.

- Z. Chen et al. [99] published a research study related to FunSearch [415]; specifically, it introduces the concept of uncertainty to maintain diversity in the population of codes while ensuring a balance between exploration and intensification during the search. The results are compared to FunSearch and Evolution of Heuristic [302].

- Chin, Winkenbach, and Srivastava [103] published a research study addressing the resolution of a particular CO, related to the vehicle routing domain. The LLM is used to train a model in a supervised manner on computationally inexpensive, sub-optimal solutions obtained algorithmically.

- Da et al. [119] published a research study on integrating LLMs into traffic and routing contexts. Specifically, the framework enables user–LLM conversations, generates solutions based on domain knowledge, and allows for solution visualization and validation.

- De La Rosa et al. [130] published a research study on using LLMs for travel planning. The LLM retrieves information about a given city and generates a route to visit specified landmarks.

- De Zarzà et al. [132] published a research study addressing the resolution of problems in a specific CO domain (financial and budget optimization). The LLM is used to provide domain-specific advice to the user.

- Dhanaraj et al. [140] published a research study on integrating human preferences into task scheduling for human-robot teams using Constraint Programming (CP) and LLMs.

- Doan [154] published a research study on the topic of recognizing optimization entities. The LLM is used to recognize such entities from problem descriptions expressed in NL.

- Elhenawy et al. [169] proposed a research study on solving the Traveling Salesperson Problem (TSP) using LLMs. Differently from other studies, this one uses the visual capabilities of LLMs.

- Fan et al. [171] published a literature review exploring the integration of Artificial Intelligence (AI) within the Operations Research (OR) process, focusing on enhancing various stages such as parameter generation and model formulation.

- Freuder [185] published a position paper discussing the role of LLMs in constraint satisfaction problems, where traditionally a crucial component is the dialogue between an optimization expert and a domain expert. The LLM is used to replace the optimization expert.

- Gangwar and Kani [188] published a research study describing a method to translate NL descriptions into LP problem formulations.
- Ghiani, Solazzo, and Elia [198] proposed a research study on using LLMs to integrate stakeholders' preferences in decision-making processes. The methodology is applied to a last-mile delivery problem, and the LLM is used in two ways: to act as a construction heuristic and to set the weights of the optimizer (i.e., parameter tuning).
- P.-F. Guo et al. [207] published a research study on using LLMs to generate solutions. Specifically, the LLM is asked to mimic the behavior of well-known algorithms, such as Hill Climbing (HC).
- Hao, Zhang, and Fan [211] published a research study on using LLMs in planning. Specifically, LLMs are used to recognize entities, formulate the model, generate code, and catch errors.
- Hao Chen and Li [212] published a research study on the usage of LLM in detecting model infeasibility. The LLM is used to provide NL descriptions of the optimization model itself, identify potential sources of infeasibility, and offer suggestions to make the model feasible.
- He et al. [215] published a research study describing a method to generate LP problem formulations from NL descriptions. The LLM automates the transformation of text-based LP problem descriptions into structured formats, including entity recognition tasks.
- Y. Hu et al. [226] published a research study on finding solutions for graph-related problems. Additionally, an explainability layer is included to outline the reasoning process.
- B. Huang et al. [227] published a research study on using LLMs to generate solutions for COPs. The experiments are conducted on the TSP.
- C. Huang et al. [228] published a research study proposing a tool called OR-Instruct, used to create synthetic data tailored to optimization modeling. They test the approach by developing the IndustryOR dataset.
- S. Huang et al. [230] published a literature review exploring the integration of LLM and optimization, considering both the perspective of LLM for optimization and optimization for LLM. The LLM is used as a black-box optimization search model or to generate optimization algorithms.
- Y. Huang et al. [232] published a research study proposing an optimization framework based on LLMs in the context of Capacited Vehicle Routing Problem (CVRP). The LLM is used to generate solutions using textual and visual prompts.
- Huang, Shi, and Sukhatme [233] published a research study addressing a specific class of COPs, the Vehicle Routing Problems (VRPs). The LLM is used to generate `Python` code from NL descriptions.

- Jang [238] published a research study introducing a method to generate LP problem formulations. The LLM is used to translate NL description into LP formulation with entity recognition.

- C. Jiang et al. [241] published a research study on using LLMs in an end-to-end approach: the LLM generates solution code directly from NL problem descriptions.

- Jiang, Xie, and Luo [242] published a research study on using LLMs as optimizers, i.e., generating solutions in the context of planning.

- Jin et al. [244] proposed a research study on using LLMs for problem formulation and code generation in the context of energy management.

- Jobson and Li [245] published a research study on using LLMs to create feasible schedules for conferences. The LLM either groups presentation titles or generates solutions.

- D. Ju et al. [247] published a research study on using LLMs for travel planning. LLMs recognize entities, create a corresponding MILP model, and solve the code.

- Khan and Hamad [253] published a research study investigating LLMs's capabilities in solving COPs. Specifically, LLMs either generate code or produce solutions. The study considers the TSP, the assignment problem, the transportation problem, and the shortest path problem.

- Kikuta et al. [254] published a research study introducing a framework designed to explain the decision-making process for a certain class of problems. The LLM is used to explain the influence of each route edge and integrate counterfactual explanations. The COP addressed is the VRP.

- Lai et al. [277] reviewed the literature on LLMs regarding their mathematical capabilities and briefly discussed their applications in CO, along with math word problems, geometry problems, and theorem proving.

- Lawless et al. [282] published a research study on using LLMs to decide which cutting-plane strategy to use, thus assisting the MILP solver in choosing the appropriate parameters.

- Lawless et al. [283] published a research study introducing a hybrid framework that integrates LLMs with CP to enable interactive decision support. The LLM is used in the preference elicitation phase to translate user input into structured constraint functions that the CP model can understand. The COP addressed is the Meeting Scheduling Problem, a specific Constraint Satisfaction Problem (CSP).

- Li et al. [287] published a research study investigating the usage of LLM for reasoning about supply chain optimization. The LLM translates human queries into code, which is then utilized by an optimization solver. The final answer is returned via the LLM.

- Li et al. [288] published a research study on using LLMs to generate solutions for a travel planning problem. The solutions are evaluated and validated with user feedback.
- Li, Zhang, and Mak-Hau [290] published a research study describing a method for synthesizing MILP models directly from NL descriptions. The LLM is used in the initial modeling phase to identify and classify decision variables and constraints.
- S. Li et al. [292] published a research study on generating MILP codes (and instances) with LLMs to train learning-based methods. The pipeline is called MILP-Evolve.
- Li et al. [295] published a research study on reasoning with graphs and networks. LLMs are used for knowledge extraction, entity recognition, and code generation in `Python`.
- F. Liu et al. [301] published a research study on automating the design of search operators within multi-objective evolutionary procedures. The LLM is used to generate new individuals for each subproblem within the decomposition approach.
- F. Liu et al. [302] published a research study on the usage of LLM in automatic heuristic design. The LLM is used to generate and evolve heuristic algorithm components, tested on TSPs, Permutation Flowshop Scheduling Problem (PFSP), and online Bin Packing Problem (BPP).
- F. Liu et al. [303] published a research study on automatically generating optimization algorithms through an evolutionary framework. The LLM creates the initial solutions—each individual represents an algorithm—and applies mutation and crossover. The COP addressed is the TSP.
- F. Liu et al. [304] published a literature review on using LLMs for algorithm design, including optimization algorithms.
- F. Liu et al. [305] published a research study on algorithm design (heuristic code generation in `Python`), comparing results with other approaches such as FunSearch [415].
- S. Liu et al. [309] published a research study investigating LLMs as evolutionary combinatorial optimizers. A LLM selects parent solutions from a given population, performing crossover and mutation to generate offspring. The COP addressed is the TSP.
- Y. Liu et al. [311] published a research study investigating the integration of LLM in a delivery route optimization problem (a variation of the TSP). The LLM is used to gather knowledge regarding delivery patterns.
- Long et al. [314] published a literature review on Network Operations and Performance Optimization, discussing potential roles for LLMs.
- Mao et al. [329] published a research study investigating the usage of LLM to enhance evolutionary procedures for identifying critical nodes in a graph. The LLM crosses and mutates given functions to generate new code snippets.

- Martinek, Łukasik, and Gandomi [335] proposed a research study in which LLMs are used to set Metaheuristics (MHs) parameters, tested on the TSP and the Graph Coloring Problem.
- Michailidis, Tsouros, and Guns [343] proposed a research study examining LLMs from entity recognition to solution generation. In the context of entity recognition, the authors used Ner4Opt [123].
- Mo et al. [349] published a research study on generating solutions using LLMs in the context of planning.
- Mostajabdaveh et al. [357] published a research study analyzing the capabilities of LLMs in CO compared to human experts. A benchmark dataset called ORQUA is introduced, evaluating LLMs by asking them to answer questions based on NL problem descriptions.
- Mostajabdaveh et al. [358] published a research study on using LLMs in an end-to-end optimization process, where the LLM generates solution code from NL problem descriptions.
- Nana Teukam et al. [360] published a research study introducing a framework for optimizing enzymes. The LLM learns relationships between amino acid residues linked to structure and function, which are then used as input for an evolutionary search procedure aimed at improved catalytic performance.
- Ning et al. [366] published a research study on recognizing optimization entities and providing LP problem formulations. The LLM is used to identify these entities from NL problem descriptions and to generate the LP model.
- Obata et al. [367] proposed a research study on integrating LP and Dependency Graph approaches with LLMs for robot planning tasks.
- Pallagani et al. [380] proposed a review on the applications of LLMs in planning, including language translation, plan generation, model construction, multi-agent planning, interactive planning, heuristic optimization, tool integration, and brain-inspired planning.
- Ramamonjison et al. [402] published a research study to recognize entities and generate LP formulations from NL descriptions of LP problems.
- Ramamonjison et al. [403] published a technical report on the Natural Language for Optimization Modeling (NL4Opt) competition, describing tasks, datasets, metrics for evaluation, and competition statistics.
- Régin, De Maria, and Bonlarron [407] proposed a research study introducing GenCP, a framework combining CP with LLMs to handle structural constraints, including the semantic meaning in text generation.
- Reinhart and Statt [409] proposed a research study on using LLMs to generate solutions for macromolecule design. The LLM was also used to explain or justify the proposed solution.

- Romanko, Narayan, and Kwon [414] published a research study evaluating LLM stock-picking capability. The LLM is used to generate financial assets (solutions) for which a Mean-Variance Cardinality-Constrained Portfolio Optimization Model is computed.

- Romera-Paredes et al. [415] published a research study proposing a new evolutionary procedure. The LLM is integrated to find new solutions and heuristics for COPs by evolving the code of an initial program skeleton at each step. The COPs addressed are the Cap Set problem and online bin packing.

- Saka et al. [421] published a literature review investigating the usage of Generative Pre-trained Transformer (GPT) models in the Architecture, Engineering, and Construction Industry (AEC) industry. The review identifies opportunities, evaluates limitations, and validates a LLM use case for solution generation in AEC.

- Chacón Sartori et al. [88] published a research study proposing a tool called OptiPattern. The LLM extracts knowledge from instances and integrates it into the MH (Genetic Algorithm (GA)) process. The methodology is applied to a network problem.

- Srivastava and Pallagani [441] published a position paper on the application of LLMs to planning and scheduling.

- Sui et al. [447] published a literature review addressing deep learning methods for the TSP, which also mentions the role of LLMs.

- Sun et al. [448] presented a research study on leveraging LLMs to solve SAT problems through a framework called AutoSAT. The framework automatically selects and optimizes heuristics within a predefined search space, building on conflict-driven clause learning solvers.

- Tran and Hy [464] published a research study on using LLMs in protein design. The LLM generates feasible solutions from incomplete ones, acting similarly to a repair operator in Large Neighborhood Search (LNS)

- Tsouros et al. [467] published a research study introducing a framework for translating NL descriptions into CP. The LLM is used to produce executable code that can be run and debugged.

- Tupayachi et al. [469] published a research study on applying LLMs to domain knowledge and entity recognition. Specifically, the LLM builds knowledge graphs on the problem at hand (e.g., transportation networks).

- Ustyugov [472] published a research study on using LLMs for parameter tuning. Their methodology leverages BERT-based embeddings to predict solver parameters for MILP problems, aiming to automate and improve efficiency in `Gurobi`.

- Stein, Vermetten, and Bäck [444] published a research study on integrating LLM-generated code with parameter tuning. The code is generated via LLaMEA [442], and

the tuning is performed using `SMAC` [298]. While LLaMEA was originally designed for black-box optimization, this study also addresses CO (TSP and online bin packing).

- Voboril, Ramaswamy, and Szeider [479] proposed a research study introducing StreamLLM. The LLM enriches existing CP code in `Python` with additional constraints, aiming to reduce the search space.
- Wang, Chen, and Zheng [486] published a research study on the topic of recognizing optimization entities. The LLM identifies such entities from NL problem descriptions.
- Wang et al. [491] published a research study on using LLMs as optimizers, i.e., generating solutions for drone placement.
- Wasserkrug et al. [493] published a position paper advocating for introducing LLMs into CO to democratize optimization practices, helping non-experts across different sectors.
- Wu et al. [502] published a literature review on vehicle routing, noting LLMs among other deep learning methods for TSP.
- Wu et al. [500] published a literature review on the intersection between LLM and evolutionary computation, offering insights on possible research directions.
- Wu et al. [501] published a research study on using LLMs for algorithm selection. More specifically, the LLM extracts features related to the underlying optimization algorithms.
- Z. Xiao et al. [505] published a research study introducing a multi-agent framework that automates the translation of problem descriptions into mathematical formulations and executable code. The LLM formulates models from NL descriptions and generates runnable code.
- C. Yang et al. [508] published a research study on LLMs as optimizers. The LLM generates candidate solutions based on the problem description and previously evaluated solutions in the meta-prompt. The COP addressed is the TSP.
- Z. Yang et al. [509] proposed a research study on using LLMs for entity recognition and `Python` code generation. The final LLM output also includes the solution, though not directly computed by the LLM. Two datasets, Optibench and ReSocratic-29k, are introduced.
- Yao et al. [510] published a research study on generating code that balances two objectives: efficiency of the code and the quality of solutions generated by it, tested on TSP and online bin packing.
- Yatong, Yuchen, and Yuqi [511] published a research study on generating heuristic code for task scheduling in edge servers.
- Ye et al. [512] published a research study on integrating LLM into hyper-heuristics generation. The LLM is used to generate heuristic algorithms in `Python`, tested on

well-known COPs like TSP, CVRP, Orienteering Problem (OP), Decap Placement Problem (DPP), Multiple Knapsack Problem (MKP), and BPP.

- You et al. [513] published a research study on a domain-specific COP (robot task sequencing). The LLM is used as a black-box optimizer.
- Yu and Liu [514] published a research study on the usage of LLMs for robust network design. The LLM generates heuristic code.
- Yu and Liu [515] published a position paper on the evolution of optimization toward automation, mentioning the integration of LLMs.
- J. Zhang et al. [519] published a research study proposing OptLLM, a system that accepts user queries in natural language, converts them into mathematical formulations and programming code, and calls solvers for decision-making. OptLLM supports multi-round dialogues to iteratively refine modeling and solving.
- R. Zhang et al. [520] published a research study on automated heuristic design (code generation). Results are compared with ReEvo [512] and FunSearch [415].
- Zhao et al. [524] published a literature review on optimization-based task and motion planning within a domain-specific CO. The review discusses how LLMs might generate domain knowledge and goal descriptions for planning methods.

G.3 Classification of Studies by Optimization Process Step

This appendix provides a breakdown of the identified studies with respect to the optimization process (Table G.1). Columns display the general tasks (e.g., problem modeling), further divided into activities (e.g., domain knowledge). Furthermore, we report the optimization paradigm (e.g., CP) and technical details on the implementation (i.e., programming languages and commercial solvers). Each row groups together studies that share the same characteristics. The checkmark symbol (✓) identifies whether a study performs/is related to the column item.

Table G.1: Classification of the studies by Combinatorial Optimization task within the optimization process.

Studies	Prob. Mod.			Sol. Meth.				Ben.		Valid.		Models/Algorithms Types													Prog. Lang.	Lib./ Solv.
	Domain Know.	Entity Rec.	Model Creation	Code Gen.	Solution Gen.	Param. Tuning	Alg. Selection	Explain.	Visual Analysis	Sol. Valid.	Model Val.	CP	LP	ILP	MILP	Heuristic	HH	EA	GA	QAP	MOO	Non Linear	MH (general)	SAT		
[132, 311, 524]	✓																									
[88]	✓																		✓							
[130]	✓				✓																					
[245]	✓				✓									✓												
[17, 469]	✓	✓																								
[119]	✓	✓			✓				✓	✓																
[140]	✓	✓	✓									✓													CP-SAT	
[295]	✓	✓		✓																					Python	
[154, 486]		✓											✓													
[233]		✓		✓						✓															Python	
[290]		✓	✓												✓											
[12, 215, 238, 366, 402]		✓	✓										✓													
[367]		✓	✓		✓								✓													
[467]		✓	✓	✓								✓													Python	CMPy [205]
[247]		✓	✓	✓											✓											
[358]		✓	✓	✓							✓		✓	✓						✓						Zimpl
[211]		✓	✓	✓							✓				✓									✓	Python	MST, Gurobi
[10, 11, 241, 509]		✓	✓	✓									✓		✓										Python	
[519]		✓	✓	✓						✓	✓		✓		✓								✓		MAPL code	MindOpt
[343]		✓	✓	✓	✓							✓													Python	CMPy [205]

Continued on next page

Table G.1: Classification of the studies by Combinatorial Optimization task within the optimization process (continued).

Studies	Prob. Mod.			Sol. Meth.				Ben.		Valid.		Models/Algorithms Types													Prog. Lang.	Lib./ Solv.
	Domain Know.	Entity Rec.	Model Creation	Code Gen.	Solution Gen.	Param. Tuning	Alg. Selection	Explain.	Visual Analysis	Sol. Valid.	Model Val.	CP	LP	ILP	MILP	Heuristic	HH	EA	GA	QAP	MOO	Non Linear	MH (general)	SAT		
[407]			✓									✓														
[228]			✓	✓								✓	✓		✓						✓	✓			Python	coptpy
[244]			✓	✓											✓										Python	CVXPY [152]
[253]				✓	✓																				Python	
[198]					✓	✓		✓													✓					
[226, 409]					✓			✓																		
[169]					✓				✓	✓	✓															
[472]						✓									✓											
[288]					✓					✓																
[58, 103, 207, 227, 232, 242, 349, 421, 464, 491, 508, 513]					✓																					
[335]						✓													✓				✓		Python	Mealpy
[282]						✓									✓											
[99, 305, 510, 511, 514, 520]				✓												✓									Python	
[448]				✓												✓									C++	
[292]				✓											✓										Python	
[444]				✓														✓							Python	
[60]				✓	✓																				Python	
[501]							✓																			

Continued on next page

Table G.1: Classification of the studies by Combinatorial Optimization task within the optimization process (continued).

Studies	Prob. Mod.			Sol. Meth.				Ben.		Valid.		Models/Algorithms Types													Prog. Lang.	Lib./ Solv.
	Domain Know.	Entity Rec.	Model Creation	Code Gen.	Solution Gen.	Param. Tuning	Alg. Selection	Explain.	Visual Analysis	Sol. Valid.	Model Val.	CP	LP	ILP	MILP	Heuristic	HH	EA	GA	QAP	MOO	Non Linear	MH (general)	SAT		
[1, 188]			✓										✓													
[505]			✓	✓							✓		✓		✓										Python	Gurobi [208] (gurobipy)
[283]			✓	✓									✓												Python	
[303, 415]				✓												✓									Python	
[512]				✓													✓								Python	
[287]				✓											✓										Python	Gurobi [208] (gurobipy)
[329]				✓														✓							Python	
[20]				✓											✓										MiniZinc	
[302]				✓													✓								Python	
[21]				✓								✓													Python	CMPy [205]
[479]				✓								✓													Python	MiniZinc
[414]					✓										✓										Python	CVXPY [152]
[309]					✓												✓									
[301]					✓													✓			✓				Python	PyMoo [52]
[360]					✓														✓						Python	GT4SD [326]
[254]								✓																		
[91]									✓																R/Web Int.	
[212]											✓				✓										Python	Gurobi [208] (Pyomo [214])

G.4 Classification of Studies by Large Language Model Architecture

This appendix provides the analysis of the retrieved studies in terms of LLMs (Table G.2). We categorize each LLM by its architecture, the tasks it was employed for, its release date, evaluation metrics, and corresponding studies. The table also highlights the access type (`F/P`, indicating free or paid) and source availability (`O/C`, indicating open or closed). Overlapping naming conventions often create confusion, as the same term can refer to both a general architecture and specific models. Additionally, it is sometimes unclear whether researchers fine-tuned a model directly or used a version adapted for conversational purposes. When researchers did not specify the exact model in their studies, we refer to the base architecture and denote it with the keyword `Family`, as is commonly done with `GPT-4`.

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability.

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
T5 [401]	T5-Base	10/2019	Problem Modeling, Solution Method	F	O	/	[103, 215]
	CodeT5-finetuned_CodeRL [284]	07/2022	Problem Modeling, Solution Method	F	O	Training Loss, Success Rate, Correctness	[21]
BART [286]	BART-Base	10/2019	Problem Modeling	F	O	Accuracy	[188, 402]
	BART-Large	10/2019	Problem Modeling	F	O	Accuracy	[188, 238]
UnixCoder [206]	UnixCoder	03/2022	Solution Method	F	O	PAR10	[501]
GPT-3.5 [62]	Text-Davinci-Edit-001	07/2022	Solution Method	P	C	/	[20]
	Text-Davinci-003	11/2022	Problem Modeling, Solution Method	P	C	Accuracy	[20, 283, 287]
	GPT-3.5 (Family)	11/2022	Problem Modeling, Solution Method, Benchmarking, Validation	P	C	# API Call Rate, API Mismatching Rate, Error Raise Rate, Throughput, Average Travel Time, GAP	[11, 60, 119, 519, 520]
	ChatGPT 3.5	11/2022	Problem Modeling	F	C	Accuracy	[290]
	ChatGPT 3.5-Turbo	11/2022	Problem Modeling, Solution Method	P	C	/	[301, 303, 508]
	GPT-3.5-turbo	11/2022	Problem Modeling, Solution Method, Validation	P	C	Accuracy, Hypervolume, Inverted Generational Distance, Compile Error Rate, Runtime Error Rate	[302, 305, 505, 509–512]
	GPT-3.5-turbo-0613	06/2023	Problem Modeling, Solution Method	P	C	F1, ANC, Rank, Uncertainty, Policy Metric, Goal Metric	[12, 207, 309, 329]

Continued on next page

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability (continued).

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
	GPT-3.5-Turbo-1106	11/2023	Solution Method	P	C	Correctness, Output Consistency	[227]
GPT-4 [374]	GPT-4 (Family)	03/2023	Problem Modeling, Solution Method, Benchmarking, Validation	P	C	Accuracy, Feasibility, Optimality, Efficiency, ROUGE [296], BERTScore [521], Completeness, Homogeneity, No API Call Rate, API Mismatching Rate, Error Raise Rate, Throughput, Average Travel Time, Code Compilation Success, Debugging Success Rate, and Code Generation Efficiency	[1, 11, 60, 119, 130, 212, 233, 244, 245, 283, 287, 469, 508, 509, 519, 520]
	ChatGPT 4	03/2023	Problem Modeling, Solution Method, Benchmarking	P	C	Macro-F1 Score, Time	[17, 254, 414, 513]
	GPT-4-0613	06/2023	Problem Modeling, Solution Method	P	C	F1, Correctness, Output Consistency	[12, 132, 227]
	GPT-4-Turbo	11/2023	Problem Modeling, Solution Method, Benchmarking	P	C	Score	[91, 253, 303, 394, 491, 512, 514]
	GPT-4-Vision-Preview	12/2023	Solution Method	P	C	/	[232]
	GPT-4o	05/2024	Problem Modeling, Solution Method, Validation	P	C	Execution Rate, Solving Accuracy, Average Solving Times	[10, 88, 211, 241, 253, 292, 349, 479]
	ChatGPT-4o	05/2024	Problem Modeling, Solution Method, Benchmarking, Validation	P	C	Accuracy, Consistency, Stability, Solution Consistency, Constraints Robustness, Runtime Efficiency	[169, 198]
	GPT-4o-2024-05-13	05/2024	Solution Method	P	C	/	[444]
	ChatGPT-4-Turbo	07/2024	Problem Modeling, Solution Method	P	C	/	[226]

Continued on next page

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability (continued).

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
	GPT-4o-mini	07/2024	Solution Method	P	C	/	[226, 295, 305, 491]
	ChatGPT-4o-mini	07/2024	Problem Modeling, Solution Method	P	C	/	[226]
	GPT-4o-2024-08-06	08/2024	Solution Method	P	C	Solved Instances, Penalized Average Runtime	[448]
PaLM 2 [202]	Bard	05/2023	Problem Modeling	F	C	Accuracy	[290]
	Codey	05/2023	Solution Method	P	C	/	[415]
	PaLM 2-L	05/2023	Solution Method	P	C	Accuracy	[508]
	PaLM 2-L-IT	05/2023	Solution Method	P	C	Accuracy	[508]
	Text-Bison	05/2023	Solution Method	P	C	Accuracy	[508]
LLaMa 2 [193]	LLaMa 2 (Family)	07/2023	Solution Method	F	O	Accuracy	[103]
	LLaMa 2-7b	07/2023	Problem Modeling, Solution Method, Benchmarking, Validation	F	O	F1, Throughput, Average Travel Time, No API Call Rate, API Mismatching Rate, Error Raise Rate	[12, 119]
	LLaMa 2-13b	07/2023	Problem Modeling, Solution Method, Benchmarking, Validation	F	O	Throughput, Average Travel Time, No API Call Rate, API Mismatching Rate, Error Raise Rate	[119]
	LLaMa 2-13b-Chat	07/2023	Solution Method	F	O	Correctness, Output Consistency	[448]
	CodeLlama [460]	08/2023	Solution Method	F	O	/	[60, 302, 520]

Continued on next page

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability (continued).

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
	CodeLlama-Instruct [460]	08/2023	Problem Modeling, Solution Method, Validation	F	O	Accuracy, Feasibility, Efficiency	[60, 358]
	Tulu-v2-dpo-7b [192]	11/2023	Benchmarking	F	O	Score	[91]
Mistral [239]	Mistral-7B	09/2023	Problem Modeling, Solution Method	F	O	Accuracy, Pass@K	[228]
	Zephyr-7B-beta [468]	10/2023	Problem Modeling, Solution Method, Validation	F	O	/	[358]
	Le Chat	07/2024	Problem Modeling	P	C	/	[335]
Qwen [398]	Qwen1.5-14B	10/2023	Problem Modeling, Solution Method	F	O	Execution Rate, Solving Accuracy, Average Solving Times	[241]
	Qwen-Turbo	08/2024	Solution Method	F	O	/	[305]
	Qwen (LoRA Fine-Tuned)	08/2024	Problem Modeling, Solution Method	P	C	/	[519]
Gemini [458]	Gemini 1.0 Pro	12/2023	Problem Modeling, Solution Method, Validation	P	C	Feasibility, Optimality, Efficiency, F1	[233, 302]
	Gemini 1.5 Pro	02/2024	Problem Modeling	P	C	/	[58]
	Gemini 1.5 Flash	02/2024	Problem Modeling	P	C	/	[58]
	Gemini 2.0 Flash	08/2024	Problem Modeling	P	C	/	[335]
	Gemini 2.0 Pro	08/2024	Solution Method	P	C	/	[227]
Mixtral [240]	Mixtral-8×7b-instruct-v0.1	01/2024	Benchmarking	P	C	Score	[91]

Continued on next page

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability (continued).

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
	Mixtral-8×22B	07/2024	Problem Modeling, Solution Method	P	C	/	[88]
DeepSeek [134]	DeepSeek-LLM-7B-Base	01/2024	Solution Method	P	C	/	[302]
	DeepSeek-Math-7B [429]	06/2024	Problem Modeling, Solution Method	P	C	Accuracy, Pass@K	[228]
	DeepSeek-Coder-33B	07/2024	Solution Method	P	C	/	[99, 520]
	DeepSeek-V2 [135]	08/2024	Problem Modeling, Solution Method	P	C	/	[509]
	DeepSeek-Coder-V2 [136]	08/2024	Solution Method	P	C	/	[511]
Claude 3.5 [24]	Claude 3.5 Sonnet	06/2024	Problem Modeling, Solution Method, Benchmarking, Validation	P	C	Accuracy, Solution Quality, # Good Solutions, Convergence Rate, Instruction Adherence, Consistency, Stability, Prompt Sensitivity, Stochastic Variability, Constraints Robustness, Runtime Efficiency	[211, 409, 479]
	Claude 3.5 Opus	06/2024	Problem Modeling, Solution Method	P	C	/	[88, 520]
	Claude 3.5 Haiku	06/2024	Solution Method	P	C	/	[305]
	Claude-3.5-Sonnet-20241022	10/2024	Solution Method	P	C	/	[242]
Cohere [13]	Command-R+	06/2024	Problem Modeling, Solution Method	P	C	/	[88]

Continued on next page

Table G.2: Classification of studies by the role of 70 LLMs in combinatorial optimization tasks. The F/P column indicates free or paid access type, while O/C denotes open or closed source availability (continued).

Architecture	LLM	Date	Task	F/P	O/C	Metrics	Studies
LLaMa 3 [312]	LLaMa 3-70B	07/2024	Problem Modeling, Solution Method	F	O	/	[10, 247, 509, 512]
	LLaMa 3-8B	08/2024	Problem Modeling, Solution Method	F	O	Accuracy, Compilation Error Rate, Runtime Error Rate	[228]
	LLaMa 3-70B-Instruct	08/2024	Solution Method	F	O	Accuracy, Pass@K	[511]
	LLaMa 3.1-8B	08/2024	Solution Method	F	O	/	[305]
Qwen2 [399]	Qwen2.5-7B	07/2024	Problem Modeling, Solution Method	F	O	Accuracy, Pass@K	[228]
StarCoder [291]	StarCoder2 [318]	07/2024	Solution Method	P	C	/	[520]
GLM [164]	GLM-3-Turbo	07/2024	Solution Method	P	C	/	[305, 511]
OpenCoder [229]	OpenCoder-8B-Instruct	07/2024	Solution Method	P	C	/	[99]
Yi [461]	Yi-34B-Chat	07/2024	Solution Method	P	C	/	[305]
Gemma [459]	Gemma 2 27B	07/2024	Problem Modeling	P	O	/	[58]
InternLM2 [73]	InternLM2-20B-Chat	08/2024	Solution Method	P	C	Correctness, Output Consistency	[227]

G.5 Classification of Studies by Benchmark Dataset

This appendix provides the tabular analysis related to the classification of studies by dataset (Table G.3). Please refer to Section 17.3 for further context.

Table G.3: Classification of studies by benchmark dataset.

Name	Source	Studies	#
LPWP or NL4Opt	Ramamonjison et al. [402]	[1, 11, 12, 154, 188, 215, 228, 238, 241, 290, 343, 366, 402, 486, 505, 519]	16
ComplexOR	Z. Xiao et al. [505]	[11, 241, 505]	3
NLP4LP	AhmadiTeshnizi et al. [10] and AhmadiTeshnizi, Gao, and Udell [11]	[10, 11, 241]	3
IndustryOR	C. Huang et al. [228]	[228, 241]	2
Mamo	X. Huang et al. [231]	[228, 241]	2
GraphInstruct	Luo et al. [321]	[226, 295]	2
AI-copilot-data	Amarasinghe et al. [21]	[21]	1
Almonacid	Almonacid [20]	[20]	1
Safeguard, Code Generation	Lawless et al. [283]	[283]	1
Huang et al.	Huang, Shi, and Sukhatme [233]	[233]	1
OptiChat	Hao Chen and Li [212]	[212]	1
Michailidis et al.	Michailidis, Tsouros, and Guns [343]	[343]	1
Optibench, ReScratic-29k	Z. Yang et al. [509]	[509]	1
Mostajabdaveh et al.	Mostajabdaveh et al. [358]	[358]	1
Zhang et al.	J. Zhang et al. [519]	[519]	1
SearchBench	Borazjanizadeh et al. [60]	[60]	1
Ju et al.	D. Ju et al. [247]	[247]	1
Talk Like A Graph	Fatemi, Halcrow, and Perozzi [172]	[295]	1
LLM4DyG	Z. Zhang et al. [523]	[295]	1
GraphViz	N. Chen et al. [97]	[295]	1
NLGraph	H. Wang et al. [484]	[295]	1
GNN-AutoGL	Li et al. [295]	[295]	1
ORQUA	Mostajabdaveh et al. [357]	[357]	1

G.6 Classification of Studies by Application Domain

This appendix provides the tabular analysis related to the classification of studies by application domain (Table G.4). Please refer to Section 17.3 for further context.

Table G.4: Classification of studies by application domain.

Domain	COP/Detail	Studies	#
Routing	Traveling Salesperson	[99, 169, 227, 253, 301–303, 305, 311, 335, 444, 447, 508, 510, 512, 520]	26
	Vehicle Routing	[103, 119, 198, 232, 233, 253, 254, 305, 502, 512]	
	Orienteering	[512]	
	Travel	[130, 247, 288]	
Scheduling and Planning	Permutation Flowshop Scheduling	[21, 302]	14
	Meeting and Conferece Scheduling	[245, 283]	
	Server Scheduling	[511]	
	Planning	[58, 140, 211, 242, 349, 367, 380, 491, 513]	
Network and Graphs	Network Design	[226, 295, 314, 469, 514]	10
	Critical Node Identification	[329]	
	Coloring	[335]	
	Social Network	[88]	
	Shortest Path, Assignement	[253]	
	Path Finding	[60]	
Packing	Bin Packing	[99, 302, 305, 415, 444, 510, 512, 520]	8
	Multiple Knapsack	[512]	
Combinatorics	Cap Set	[415]	4
	Admissible Set	[520]	
	Puzzles	[60, 343]	
	Subset Sum, Sorting, Under-determined Systems	[60]	
Engineering	Construction	[421]	3
	Circuit Design	[512]	
	Energy Management	[244]	
Finance	Portfolio Optimization	[17, 132, 414]	3
Bioinformatics	Enzyme Design	[360, 409, 464]	3
Supply Chain	Warehousing	[287]	1
Strings	Text Generation	[407]	1

Appendix H

Chapter 18: Additional Material

This appendix provides additional material w.r.t. Chapter 18. Specifically, Section H.1 reports complete results of the Mean Absolute Error (MAE) for the case of features probing and Section H.2 reports the complete statistical tests conducted on algorithm selection probing.

H.1 Feature Probing Results

Tables H.1 to H.4 report the MAE values obtained for each problem, representation, and feature complexity level, comparing the performance of *direct querying* and *probing*.

Table H.1: Direct Querying and Probing MAE per Representation. Data represent the mean MAE over different pooling strategies and feature complexities for probing methods for BPP.

			Complexity		
Representation	Regressor	Pooling	○	◓	●
`standard`	LinearRegression	mean	116.40	62.44	61.56
		max	424.40	232.91	232.53
		last	512.41	271.10	272.45
	MLP	mean	39.67	11.08	17.00
		max	83.04	38.24	25.01
		last	84.06	34.21	35.20
	LightGBM	mean	140.37	75.37	73.52
		max	131.02	74.24	71.97
		last	183.57	87.63	85.08
	Direct Querying		471.22	197.49	102.77
`code-like`	LinearRegression	mean	118.11	60.22	59.64
		max	476.37	260.48	260.41
		last	305.86	172.94	172.64
	MLP	mean	80.45	22.55	21.12
		max	407.87	45.10	31.61
		last	189.94	42.54	33.15
	LightGBM	mean	154.69	70.27	70.91
		max	108.89	63.43	60.90
		last	109.45	71.51	68.41
	Direct Querying		34.82	70.85	133.32
`natural language`	LinearRegression	mean	127.82	70.72	70.13
		max	507.07	286.23	287.06
		last	493.88	270.37	270.17
	MLP	mean	123.56	29.23	15.45
		max	457.50	146.76	86.51
		last	113.29	37.11	27.05
	LightGBM	mean	146.52	75.71	73.81
		max	66.67	47.76	46.52
		last	165.65	96.26	93.12
	Direct Querying		2.98	27.33	158.88

Table H.2: Direct Querying and Probing MAE per Representation. Data represent the mean MAE over different pooling strategies and feature complexities for probing methods for GCP.

			Complexity		
Representation	Regressor	Pooling	○	◓	●
`standard`	LinearRegression	mean	7.89	—	0.80
		max	20.17	—	1.34
		last	20.55	—	1.41
	MLP	mean	13.65	—	1.03
		max	21.05	—	1.54
		last	18.53	—	1.18
	LightGBM	mean	13.30	—	0.69
		max	17.00	—	0.73
		last	36.44	—	1.39
	Direct Querying		910.47	—	22.89
`code-like`	LinearRegression	mean	6.97	—	0.76
		max	24.17	—	1.30
		last	10.39	—	1.20
	MLP	mean	12.77	—	0.86
		max	25.51	—	1.57
		last	12.95	—	1.03
	LightGBM	mean	11.52	—	0.61
		max	18.30	—	0.77
		last	16.85	—	1.08
	Direct Querying		12.56	—	81.31
`natural language`	LinearRegression	mean	5.42	—	0.73
		max	21.70	—	1.21
		last	12.64	—	1.30
	MLP	mean	13.19	—	0.93
		max	27.63	—	1.55
		last	14.82	—	1.10
	LightGBM	mean	12.11	—	0.66
		max	17.12	—	0.75
		last	20.44	—	1.07
	Direct Querying		0.00	—	179.94

Table H.3: Direct Querying and Probing MAE per Representation. Data represent the mean MAE over different pooling strategies and feature complexities for probing methods for JSP.

			Complexity		
Representation	Regressor	Pooling	○	◓	●
`standard`	LinearRegression	mean	2.20	68.99	1.41
		max	4.64	235.21	13.17
		last	3.96	126.26	5.39
	MLP	mean	1.32	27.92	1.16
		max	1.81	41.59	2.22
		last	1.42	31.93	4.53
	LightGBM	mean	2.13	37.43	1.31
		max	1.04	40.09	1.96
		last	3.14	66.71	4.10
	Direct Querying		20.53	643.63	18.08
`code-like`	LinearRegression	mean	1.37	35.68	4.25
		max	4.49	162.00	21.00
		last	3.41	22.19	4.97
	MLP	mean	1.13	19.12	2.22
		max	1.87	58.12	4.11
		last	2.21	17.64	3.33
	LightGBM	mean	1.70	26.34	1.42
		max	0.56	21.07	2.33
		last	5.40	22.08	7.64
	Direct Querying		0.00	446.61	9.59
`natural language`	LinearRegression	mean	1.07	36.24	3.12
		max	3.54	136.68	18.35
		last	2.12	50.16	5.28
	MLP	mean	0.72	24.39	1.82
		max	1.23	35.93	4.42
		last	0.89	22.28	2.80
	LightGBM	mean	1.44	27.42	1.68
		max	0.36	24.08	2.30
		last	2.18	36.20	5.52
	Direct Querying		0.42	541.31	5.97

Table H.4: Direct Querying and Probing MAE per Representation. Data represent the mean MAE over different pooling strategies and feature complexities for probing methods for KP.

			Complexity		
Representation	Regressor	Pooling	○	◓	●
`standard`	LinearRegression	mean	131,555.59	86.97	6.55
		max	201,199.19	159.87	24.13
		last	372,968.87	234.50	34.14
	MLP	mean	142,425.52	51.74	7.44
		max	148,486.27	69.17	14.43
		last	141,177.30	79.54	16.26
	LightGBM	mean	89,572.79	56.54	8.98
		max	74,578.54	62.44	11.82
		last	122,960.37	76.68	15.49
	Direct Querying		261,452.48	24,747.88	341.84
`code-like`	LinearRegression	mean	158,325.85	108.73	10.47
		max	150,884.66	171.30	33.16
		last	144,934.26	125.84	17.98
	MLP	mean	143,555.46	66.49	10.16
		max	152,150.76	74.78	15.80
		last	147,047.79	72.36	13.65
	LightGBM	mean	101,037.26	62.95	9.67
		max	48,281.40	61.16	13.57
		last	116,287.68	75.78	15.56
	Direct Querying		9,331.44	583,906.78	381.53
`natural language`	LinearRegression	mean	101,070.45	74.95	6.25
		max	170,965.33	171.01	35.50
		last	123,168.13	145.55	25.38
	MLP	mean	148,950.19	69.63	9.25
		max	153,472.86	81.83	20.94
		last	140,138.80	76.12	17.53
	LightGBM	mean	90,946.86	64.42	9.79
		max	62,731.61	65.14	14.98
		last	81,936.02	79.98	20.10
	Direct Querying		3,968.89	6,984.98	259.40

H.2 Complete Set of Statistical Analysis Tables

This section reports the full results of the mixed linear models used to assess the effects of classifiers, feature representations, and pooling strategies on accuracy. All models include random intercepts for group-level variability, with fixed-effect estimates reported alongside standard errors, z-statistics, p-values, and 95% confidence intervals.

- Table H.5 summarizes classifier effects.
- Table H.6 compares ISA-based and handcrafted features.
- Table H.7 presents the full factorial model including representation and pooling factors.
- Table H.8 reports the comparison between ISA and LLM-based standard representations using the LightGBM classifier.

Table H.5: Mixed linear model for classifier effects (`accuracy ~ C(classifier, Treatment(reference='MostFrequentClassifier'))`). The analysis is based on $n = 880$ observations and $g = 20$ groups.

Parameter	Coef.	Std.Err.	z	p-value	[0.025	0.975]
Intercept (`MostFrequentClassifier`)	0.699	0.025	28.033	< 0.001	0.650	0.748
Classifier: `LightGBM` (vs `MostFrequentClassifier`)	0.192	0.006	34.402	< 0.001	0.181	0.203
Classifier: `LogisticRegression` (vs `MostFrequentClassifier`)	0.149	0.006	26.705	< 0.001	0.138	0.160
Classifier: `MLPClassifier` (vs `MostFrequentClassifier`)	-0.011	0.006	-2.011	0.044	-0.022	-0.000
Group variance	0.012	(0.067)				

Table H.6: Results of the mixed linear model and post-hoc statistical tests comparing ISA-based and handcrafted features, specified as `accuracy ~ C(classifier, ref='LogisticRegression') + C(representation, ref='hc')`. Reported effects include the conditional effects of *representation* and *pooling*, and their interaction. The analysis is based on $n = 80$ observations and $g = 20$ groups.

Parameter	Coef.	Std.Err.	z	p-value	[0.025	0.975]
Intercept (`LogisticRegression`, Handcrafted)	0.786	0.025	30.921	< 0.001	0.737	0.836
Classifier: `LightGBM` (vs `LogisticRegression`)	0.091	0.004	22.221	< 0.001	0.083	0.099
Representation: ISA (vs Handcrafted)	0.031	0.004	7.543	< 0.001	0.023	0.039
Group variance	0.013	(0.255)				

Comparison	Test	Statistic (df)	p-value / CI	Interpretation
ISA vs Handcrafted	LRT	$\chi^2(1) = 40.017$	$p = 2.52 \times 10^{-10}$	**Significant**
ISA vs Handcrafted	Tukey HSD	$M_{\text{diff}} = 0.0309$	$p_{\text{adj}} = 0.2704$, $\text{CI}_{95\%} = [-0.0245, 0.0863]$	Not significant
ISA vs Handcrafted ($\varepsilon = 0.01$)	Wilcoxon TOST	$M_{\text{diff}} = 0.0295$	$p_{\text{low}} = 1.82 \times 10^{-12}$, $p_{\text{high}} \approx 1$, $\text{CI}_{95\%} = [-0.0031, 0.0662]$	Not equivalent

Table H.7: Results of the full factorial mixed linear model and corresponding likelihood–ratio (LRT) comparisons, specified as `accuracy ~ C(classifier, ref='LogisticRegression') + C(representation, ref='standard') * C(pooling, ref='mean')`. Reported effects include the conditional effects of *representation* and *pooling*, and their interaction. The analysis is based on $n = 360$ observations and $g = 20$ groups.

Parameter	**Coef.**	**Std.Err.**	**z**	**p-value**	**[0.025**	**0.975]**
Intercept (`LogisticRegression`, `standard`, Mean)	0.858	0.027	32.114	< 0.001	0.806	0.911
Classifier: `LightGBM` (vs `LogisticRegression`)	0.032	0.002	17.763	< 0.001	0.029	0.036
Representation: `code-like` (vs `standard`)	-0.006	0.004	-1.532	0.126	-0.013	0.002
Representation: `natural language` (vs `standard`)	0.001	0.004	0.242	0.808	-0.007	0.008
Pooling: Last (vs Mean)	-0.007	0.004	-1.854	0.064	-0.015	0.000
Pooling: Max (vs Mean)	0.011	0.004	2.795	0.005	0.003	0.018
Representation × Pooling: `code-like` × Last	0.005	0.005	1.002	0.316	-0.005	0.016
Representation × Pooling: `natural language` × Last	0.008	0.005	1.433	0.152	-0.003	0.018
Representation × Pooling: `code-like` × Max	-0.007	0.005	-1.198	0.231	-0.017	0.004
Representation × Pooling: `natural language` × Max	-0.004	0.005	-0.815	0.415	-0.015	0.006
Group variance	0.014	(0.268)				

Effect / Comparison	**Test**	**Statistic (df)**	**p-value**	**Significance**
Representation (conditional)	LRT	$\chi^2(6) = 21.315$	0.00161	**Significant**
Pooling (conditional)	LRT	$\chi^2(6) = 26.609$	0.000171	**Significant**
Representation × Pooling	LRT	$\chi^2(4) = 6.772$	0.1485	Not significant

Table H.8: Results of the mixed linear model for the comparison between Instance Space Analysis (ISA) and LLM-`standard` using LightGBM, specified as `accuracy` $\sim$ `C(representation, ref='standard')`. The analysis is based on $n = 80$ observations and $g = 20$ groups.

Parameter	**Coef.**	**Std.Err.**	**z**	**p-value**	**[0.025**	**0.975]**
Intercept (LLM)	0.891	0.054	16.440	< 0.001	0.785	0.997
Representation: ISA (vs LLM)	0.008	0.002	3.313	0.001	0.003	0.013
Group variance	0.012	(0.907)				

Comparison	**Test**	**Statistic**	**p-value / CI**	**Interpretation**
ISA vs LLM ($\varepsilon = 0.01$)	Wilcoxon TOST	$M_{\text{diff}} = 0.0054$	$p_{\text{low}} = 9.54 \times 10^{-17}$, $p_{\text{high}} = 0.021$, $\text{CI}_{95\%} = [-0.0039, 0.0244]$	**Equivalent**

References

[1] Yelaman Abdullin, Diego Molla, Bahadorreza Ofoghi, John Yearwood, and Qingyang Li. "Synthetic Dialogue Dataset Generation using LLM Agents". In: *Proceedings of the Third Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*. Singapore: ACL, 2023, pp. 181–191. URL: https://aclanthology.org/2023.gem-1.16 (cit. on pp. 235, 240, 243, 325, 337, 340, 345).

[2] Stewart Adcock. "Genetic Algorithms Utility Library (GAUL)". [Accessed: 2024-04-05]. 2005 (cit. on pp. 286, 287).

[3] Steven Adriaensen, Tim Brys, and Ann Nowé. "Fair-share ILS: a simple state-of-the-art iterated local search hyperheuristic". In: *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*. Ed. by Dirk V. Arnold. ACM, 2014, pp. 1303–1310. DOI: 10.1145/2576768.2598285 (cit. on p. 29).

[4] Steven Adriaensen and Ann Nowé. "Case study: An analysis of accidental complexity in a state-of-the-art hyper-heuristic for HyFlex". In: *IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016*. IEEE, 2016, pp. 1485–1492. DOI: 10.1109/CEC.2016.7743965 (cit. on pp. 29, 79).

[5] Stephen A. Adubi, Olufunke O. Oladipupo, and Oludayo O. Olugbara. "Configuring the Perturbation Operations of an Iterated Local Search Algorithm for Cross-domain Search: A Probabilistic Learning Approach". In: *IEEE Congress on Evolutionary Computation, CEC 2021, Kraków, Poland, June 28 - July 1, 2021*. IEEE, 2021, pp. 1372–1379. DOI: 10.1109/CEC45853.2021.9504841 (cit. on p. 29).

[6] Stephen A. Adubi, Olufunke O. Oladipupo, and Oludayo O. Olugbara. "Evolutionary Algorithm-Based Iterated Local Search Hyper-Heuristic for Combinatorial Optimization Problems". In: *Algorithms* 15.11 (2022), p. 405. DOI: 10.3390/A15110405. URL: https://doi.org/10.3390/a15110405 (cit. on p. 29).

[7] Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. "Optimization-Based Adaptive Large Neighborhood Search for the Production Routing Problem". In: *Transp. Sci.* 48.1 (2014), pp. 20–45. DOI: 10.1287/TRSC.1120.0443. URL: https://doi.org/10.1287/trsc.1120.0443 (cit. on p. 27).

[8] Ana Raquel Pena de Aguiar, Tânia Rodrigues Pereira Ramos, and Maria Isabel Gomes. "Home care routing and scheduling problem with teams' synchronization". In: *Socio-Economic Planning Sciences* 86 (2023), p. 101503. DOI: 10.1016/j.seps.2022.101503 (cit. on pp. 48, 49).

[9] Eduardo López Aguilar, Yannick Kergosien, Vincent Boyer, and Jean-Charles Billaut. "An adaptive large neighborhood search for an appointment scheduling problem in health care". In: *18ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*. 2017 (cit. on p. 27).

[10] Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, and Madeleine Udell. *OptiMUS-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale*. arXiv. 2024. DOI: 10.48550/arXiv.2407.19633 (cit. on pp. 235, 239, 240, 243, 325, 335, 340, 344, 345).

[11] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. "OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML '24. Vienna, Austria: JMLR.org, 2024, pp. 1234–1245. DOI: 10.5555/3692070.3692094 (cit. on pp. 235, 240, 241, 243, 325, 335, 339, 340, 345).

[12] Tasnim Ahmed and Salimur Choudhury. "LM4OPT: Unveiling the potential of Large Language Models in formulating mathematical optimization problems". In: *INFOR: Information Systems and Operational Research* 62.4 (2024), pp. 559–572. DOI: 10.1080/03155986.2024.2388452 (cit. on pp. 235, 239, 240, 243, 245, 325, 335, 339–341, 345).

[13] Cohere AI. *Command R+: A Large Language Model for Enterprise AI*. 2024. URL: https://docs.cohere.com/v2/docs/command-r-plus (cit. on pp. 238, 343).

[14] Syrine Roufaida Ait Haddadene, Nacima Labadie, and Caroline Prodhon. "A GRASPxILS for the vehicle routing problem with time windows, synchronization and precedence constraints". In: *Expert Systems with Applications* 66 (2016), pp. 274–294. DOI: 10.1016/j.eswa.2016.09.002 (cit. on pp. 47, 49, 197).

[15] Mehmet Anil Akbay and Christian Blum. "Two Examples for the Usefulness of STNWeb for Analyzing Optimization Algorithm Behavior". In: *Metaheuristics - 15th International Conference, MIC 2024, Lorient, France, June 4-7, 2024, Proceedings, Part II*. Ed. by Marc Sevaux, Alexandru-Liviu Olteanu, Eduardo G. Pardo, Angelo Sifaleras, and Salma Makboul. Vol. 14754. Lecture Notes in Computer Science. Springer, 2024, pp. 341–346. DOI: 10.1007/978-3-031-62922-8_25 (cit. on p. 40).

[16] Deniz Aksen, Onur Kaya, F. Sibel Salman, and Özge Tüncel. "An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem". In: *Eur. J. Oper. Res.* 239.2 (2014), pp. 413–426. DOI: 10.1016/J.EJOR.2014.05.043 (cit. on p. 27).

[17] Mohammad Alipour-Vaezi and Kwok-Leung Tsui. "Data-driven portfolio management for motion pictures industry: A new data-driven optimization methodology using a large language model as the expert". In: *Computers & Industrial Engineering* 197 (2024), p. 110574. DOI: 10.1016/j.cie.2024.110574 (cit. on pp. 235, 240, 325, 335, 340, 346).

[18] Mohamad Alissa, Kevin Sim, and Emma Hart. "Automated Algorithm Selection: from Feature-Based to Feature-Free Approaches". In: *J. Heuristics* 29.1 (2023), pp. 1–38. DOI: 10.1007/S10732-022-09505-4 (cit. on p. 42).

[19] Matteo Alleman, Jonathan Mamou, Miguel A. Del Rio, Hanlin Tang, Yoon Kim, and SueYeon Chung. "Syntactic Perturbations Reveal Representational Correlates of Hierarchical Phrase Structure in Pretrained Language Models". In: *Proceedings of the 6th Workshop on Representation Learning for NLP, RepL4NLP@ACL-IJCNLP 2021, Online, August 6, 2021*. Ed. by Anna Rogers, Iacer Calixto, Ivan Vulic, Naomi Saphra, Nora Kassner, Oana-Maria Camburu, Trapit Bansal, and Vered Shwartz. Association for Computational Linguistics, 2021, pp. 263–276. DOI: 10.18653/V1/2021.REPL4NLP-1.27 (cit. on p. 32).

[20] Boris Almonacid. *Towards an Automatic Optimisation Model Generator Assisted with Generative Pre-trained Transformer*. arXiv. 2023. DOI: 10.48550/arXiv.2305.05811 (cit. on pp. 235, 241, 244, 325, 337, 339, 345).

[21] Pivithuru Thejan Amarasinghe, Su Nguyen, Yuan Sun, and Damminda Alahakoon. *AI-Copilot for Business Optimisation: A Framework and A Case Study in Production Scheduling*. arXiv. 2023. DOI: 10.48550/arXiv.2309.13218 (cit. on pp. 233, 235, 240, 241, 243, 246, 325, 337, 339, 345, 346).

[22] Sotiris Anagnostidis and Jannis Bulian. *How Susceptible are LLMs to Influence in Prompts?* arXiv. 2024. DOI: 10.48550/arXiv.2408.11865 (cit. on p. 239).

[23] Alexandre A. Andreatta, Sergio E. R. de Carvalho, and Celso C. Ribeiro. "An Object-Oriented Framework for Local Search Heuristics". In: *TOOLS 1998: 26th International Conference on Technology of Object-Oriented Languages and Systems, 3-7 August 1998, Santa Barbara, CA, USA*. IEEE Computer Society, 1998, pp. 33–45. DOI: 10.1109/TOOLS.1998.711001 (cit. on p. 286).

[24] Anthropic. *Claude 3.5 Sonnet Model Card Addendum*. 2024. URL: https://www.anthropic.com/news/claude-3-5-sonnet (cit. on pp. 238, 343).

[25] Anthropic Team. *Introducing the next generation of Claude*. Accessed: 2024-06-18. 2024. URL: https://www.anthropic.com/news/claude-3-family (cit. on p. 31).

[26] Claus Aranha, Christian Leonardo Camacho-Villalón, Felipe Campelo, Marco Dorigo, Rubén Ruiz, Marc Sevaux, Kenneth Sörensen, and Thomas Stützle. "Metaphor-based metaheuristics, a call for action: the elephant in the room". In: *Swarm Intell.* 16.1 (2022), pp. 1–6. DOI: 10.1007/S11721-021-00202-9 (cit. on p. 4).

[27] Claudia Archetti, Leandro Coelho, Maria Grazia Speranza, and Pieter Vansteenwegen. "Beyond fifty years of vehicle routing: Insights into the history and the future". In: *European Journal of Operational Research* (2025). DOI: 10.1016/j.ejor.2025.06.014 (cit. on p. 4).

[28] Roberto Aringhieri, Davide Duma, and Giuseppe Squillace. "Online algorithms with foresight for radiotherapy patient scheduling". In: *Comput. Oper. Res.* 182 (2025), p. 107132. DOI: 10.1016/J.COR.2025.107132 (cit. on p. 3).

[29] Davide Armellini, Paolo Borzone, Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. "Modeling and solving the steelmaking and casting scheduling problem". In: *Int. Trans. Oper. Res.* 27.1 (2020), pp. 57–90. DOI: 10.1111/ITOR.12595. URL: https://doi.org/10.1111/itor.12595 (cit. on p. 285).

[30] Bruno de Athayde Prata, Levi Ribeiro de Abreu, and Victor Fernandez-Viagas. "A systematic review of permutation flow shop scheduling with due-date-related objectives". In: *Comput. Oper. Res.* 177 (2025), p. 106989. DOI: 10.1016/J.COR.2025.106989 (cit. on p. 15).

[31] Meral Azizoglu and Scott Webster. "Scheduling a batch processing machine with incompatible job families". In: *Comp. Ind. Eng.* 39.3 (2001), pp. 325–335. DOI: 10.1016/S0360-8352(01)00009-2 (cit. on p. 43).

[32] Kehinde O. Babaagba, Ritwik Murali, and Sarah L. Thomson. "Exploring the use of fitness landscape analysis for understanding malware evolution". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. Ed. by Xiaodong Li and Julia Handl. ACM, 2024, pp. 77–78. DOI: 10.1145/3638530.3664094 (cit. on p. 40).

[33] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv. 2016. DOI: 10.48550/arXiv.1409.0473 (cit. on p. 29).

[34] Simone Balloccu, Patrícia Schmidtová, Mateusz Lango, and Ondrej Dusek. "Leak, Cheat, Repeat: Data Contamination and Evaluation Malpractices in Closed-Source LLMs". In: *Proceedings of the 18th Conference of the European Chapter of the ACL (Volume 1: Long Papers)*. St. Julian's, Malta: ACL, 2024, pp. 67–93. URL: https://aclanthology.org/2024.eacl-long.5 (cit. on p. 239).

[35] Philippe Baptiste, Antoine Jouglet, and David Savourey. "Lower bounds for parallel machine scheduling problems". In: *International Journal of Operational Research* 3.6 (2008), pp. 643–664. DOI: 10.1504/IJOR.2008.019731 (cit. on pp. 44, 45, 123).

[36] Thomas Bartz-Beielstein, Marco Chiarandini, Luis Paquete, and Mike Preuss, eds. *Experimental methods for the analysis of optimization algorithms*. en. 2010th ed. Berlin, Germany: Springer, 2010 (cit. on p. 77).

[37] Thomas Bartz-Beielstein, Carola Doerr, Daan van den Berg, Jakob Bossek, Sowmya Chandrasekaran, Tome Eftimov, Andreas Fischbach, Pascal Kerschke, William La Cava, Manuel Lopez-Ibanez, Katherine M. Malan, Jason H. Moore, Boris Naujoks, Patryk Orzechowski, Vanessa Volz, Markus Wagner, and Thomas Weise. *Benchmarking in Optimization: Best Practice and Open Issues*. arXiv. 2020. URL: https://arxiv.org/abs/2007.03488 (cit. on p. 40).

[38] Michele Battistutta, Andrea Schaerf, and Tommaso Urli. "Feature-based tuning of single-stage simulated annealing for examination timetabling". In: *Ann. Oper. Res.* 252.2 (2017), pp. 239–254. DOI: 10.1007/S10479-015-2061-8 (cit. on p. 41).

[39] Roberto Battiti and Giampietro Tecchiolli. "The Reactive Tabu Search". In: *INFORMS J. Comput.* 6.2 (1994), pp. 126–140. DOI: 10.1287/IJOC.6.2.126 (cit. on p. 62).

[40] Mohammed Bazirha, Rachid Benmansour, and Abdeslam Kadrani. "An efficient two-phase heuristic for the home care routing and scheduling problem". In: *Comput. Ind. Eng.* 181 (2023), p. 109329. DOI: 10.1016/J.CIE.2023.109329 (cit. on pp. 48, 49, 185, 202, 203, 210–214).

[41] Mohammed Bazirha, Abdeslam Kadrani, and Rachid Benmansour. "Stochastic home health care routing and scheduling problem with multiple synchronized services". In: *Ann. Oper. Res.* 320.2 (2023), pp. 573–601. DOI: 10.1007/S10479-021-04222-W (cit. on pp. 48, 49, 185, 202, 203, 210–213).

[42] Sachidanand V. Begur, David M. Miller, and Jerry R. Weaver. "An integrated spatial DSS for scheduling and routing home-health-care nurses". In: *Interfaces* 27 (1997). DOI: 10.1287/inte.27.4.35 (cit. on p. 46).

[43] Yonatan Belinkov. "Probing Classifiers: Promises, Shortcomings, and Advances". In: *Comput. Linguistics* 48.1 (2022), pp. 207–219. DOI: 10.1162/COLI_A_00422 (cit. on p. 32).

[44] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. "Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling". In: *Comput. Oper. Res.* 132 (2021), p. 105300. DOI: 10.1016/J.COR.2021.105300 (cit. on pp. 3, 285).

[45] Ruggero Bellio, Sara Ceschia, Luca Di Gaspero, Andrea Schaerf, and Tommaso Urli. "Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem". In: *Comput. Oper. Res.* 65 (2016), pp. 83–92. DOI: 10.1016/J.COR.2015.07.002 (cit. on pp. 41, 77, 285).

[46] Meriem Beloucif and Chris Biemann. "Probing Pre-trained Language Models for Semantic Attributes and their Values". In: *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*. Ed. by Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih. Association for Computational Linguistics, 2021, pp. 2554–2559. DOI: 10.18653/V1/2021.FINDINGS-EMNLP.218 (cit. on p. 32).

[47] Stefano Benedettini, Andrea Roli, and Luca Di Gaspero. "EasyGenetic: A Template Metaprogramming Framework for Genetic Master-Slave Algorithms". In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, Second International Workshop, SLS 2009, Brussels, Belgium, September 3-4, 2009. Proceedings*. Ed. by Thomas Stützle, Mauro Birattari, and Holger H. Hoos. Vol. 5752. Lecture Notes in Computer Science. Springer, 2009, pp. 135–139. DOI: 10.1007/978-3-642-03751-1_14 (cit. on p. 289).

[48] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: A methodological tour d'horizon". In: *Eur. J. Oper. Res.* 290.2 (2021), pp. 405–421. DOI: 10.1016/J.EJOR.2020.07.063. URL: https://doi.org/10.1016/j.ejor.2020.07.063 (cit. on p. 11).

[49] Stefan Bertels and Torsten Fahle. "A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem". In: *Comput. Oper. Res.* 33.10 (2006), pp. 2866–2890. DOI: 10.1016/J.COR.2005.01.015 (cit. on p. 46).

[50] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. "Graph of Thoughts: Solving Elaborate Problems with Large Language Models". In: *Proceedings of the AAAI Conference on Artificial*

Intelligence 38.16 (2024), pp. 17682–17690. DOI: 10.1609/aaai.v38i16.29720 (cit. on p. 30).

[51] Roberto Bifulco, Federico Errica, Davide Sanvito, and Giuseppe Siracusano. *What Did I Do Wrong? Quantifying LLMs' Sensitivity and Consistency to Prompt Engineering*. arXiv. 2024. DOI: 10.48550/arXiv.2406.12334. URL: https://arxiv.org/abs/2406.12334 (cit. on p. 239).

[52] Julian Blank and Kalyanmoy Deb. "Pymoo: Multi-Objective Optimization in Python". In: *IEEE Access* 8 (2020), pp. 89497–89509. DOI: 10.1109/ACCESS.2020.2990567 (cit. on pp. 286, 287, 337).

[53] Ivo Blöchliger and Nicolas Zufferey. "A graph coloring heuristic using partial solutions and a reactive tabu scheme". In: *Comput. Oper. Res.* 35.3 (2008), pp. 960–975. DOI: 10.1016/J.COR.2006.05.014 (cit. on p. 63).

[54] Christian Blum, Tome Eftimov, and Peter Korosec. *Preface: Special Issue on "Understanding of Evolutionary Optimization Behavior", Part 2*. 2021. DOI: 10.1007/S11047-021-09858-Y (cit. on p. 4).

[55] Christian Blum and Gabriela Ochoa. "A comparative analysis of two matheuristics by means of merged local optima networks". In: *Eur. J. Oper. Res.* 290.1 (2021), pp. 36–56. DOI: 10.1016/J.EJOR.2020.08.008 (cit. on p. 40).

[56] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM Comput. Surv.* 35.3 (2003), pp. 268–308. DOI: 10.1145/937503.937505 (cit. on pp. 3, 237).

[57] Pim van den Bogaerdt and Mathijs de Weerdt. "Multi-machine scheduling lower bounds using decision diagrams". In: *Oper. Res. Lett.* 46.6 (2018), pp. 616–621. DOI: 10.1016/J.ORL.2018.11.003 (cit. on pp. 44, 45).

[58] Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. *Exploring and Benchmarking the Planning Capabilities of Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2406.13094 (cit. on pp. 235, 241, 246, 325, 336, 342, 344, 346).

[59] Grady Booch. *Object-Oriented Analysis and Design with Applications (2nd Ed.)* USA: Benjamin-Cummings Publishing Co., Inc., 1993 (cit. on pp. 66, 289).

[60] Nasim Borazjanizadeh, Roei Herzig, Trevor Darrell, Rogerio Feris, and Leonid Karlinsky. *Navigating the Labyrinth: Evaluating and Enhancing LLMs' Ability to Reason About Search Problems*. arXiv. 2024. DOI: 10.48550/arXiv.2406.12172 (cit. on pp. 236, 244, 246, 325, 336, 339–342, 345, 346).

[61] David Bredström and Mikael Rönnqvist. "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints". In: *Eur. J. Oper. Res.* 191.1 (2008), pp. 19–31. DOI: 10.1016/J.EJOR.2007.07.033 (cit. on p. 47).

[62] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. Virtual Conference: Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf (cit. on pp. 30, 237, 339).

[63] Maurizio Bruglieri, Simona Mancini, Roberto Peruzzini, and Ornella Pisacane. "The Multi-period Multi-trip Container Drayage Problem with Release and Due Dates". In: *Comput. Oper. Res.* 125 (2021), p. 105102. DOI: 10.1016/J.COR.2020.105102 (cit. on p. 3).

[64] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. *Sparks of Artificial General Intelligence: Early experiments with GPT-4*. arXiv. 2023. DOI: 10.48550/arXiv.2303.12712 (cit. on p. 30).

[65] David Van Bulck, Dries R. Goossens, Jan-Patrick Clarner, Angelos Dimitsas, George H. G. Fonseca, Carlos Lamas-Fernandez, Martin Mariusz Lester, Jaap Pedersen, Antony E. Phillips, and Roberto Maria Rosati. "Which algorithm to select in sports timetabling?" In: *Eur. J. Oper. Res.* 318.2 (2024), pp. 575–591. DOI: 10.1016/J.EJOR.2024.06.005 (cit. on p. 59).

[66] David Van Bulck, Dries R. Goossens, and Andrea Schaerf. "Multi-neighbourhood simulated annealing for the ITC-2007 capacitated examination timetabling problem". In: *J. Sched.* 28.2 (2025), pp. 217–232. DOI: 10.1007/S10951-023-00799-1 (cit. on pp. 17, 285).

[67] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. "A classification of hyper-heuristic approaches: Revisited". In: *Handbook of Metaheuristics*. Springer, 2019, pp. 453–477 (cit. on p. 28).

[68] Edmund K. Burke and Yuri Bykov. "The late acceptance Hill-Climbing heuristic". In: *Eur. J. Oper. Res.* 258.1 (2017), pp. 70–78. DOI: 10.1016/J.EJOR.2016.07.012 (cit. on p. 294).

[69] Edmund K. Burke, Michel Gendreau, Matthew R. Hyde, Graham Kendall, Barry McCollum, Gabriela Ochoa, Andrew J. Parkes, and Sanja Petrovic. "The Cross-Domain Heuristic Search Challenge - An International Research Competition". In: *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*. Ed. by Carlos A. Coello Coello. Vol. 6683. Lecture Notes in Computer Science. Springer, 2011, pp. 631–634. DOI: 10.1007/978-3-642-25566-3_49 (cit. on p. 28).

[70] Edmund K. Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. "Hyper-Heuristics: An Emerging Direction in Modern Search Technology". In: *Handbook of Metaheuristics*. Ed. by Fred W. Glover and Gary A.

Kochenberger. Vol. 57. International Series in Operations Research & Management Science. Kluwer / Springer, pp. 457–474. DOI: 10.1007/0-306-48056-5_16 (cit. on p. 27).

[71] Edmund K. Burke, Barry McCollum, Amnon Meisels, Sanja Petrovic, and Rong Qu. "A graph-based hyper-heuristic for educational timetabling problems". In: *Eur. J. Oper. Res.* 176.1 (2007), pp. 177–192. DOI: 10.1016/J.EJOR.2005.08.012 (cit. on p. 27).

[72] Jinyu Cai, Jinglue Xu, Jialong Li, Takuto Yamauchi, Hitoshi Iba, and Kenji Tei. "Exploring the Improvement of Evolutionary Computation via Large Language Models". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: ACM, 2024, pp. 83–84. ISBN: 9798400704956. DOI: 10.1145/3638530.3664086 (cit. on pp. 51, 246, 325).

[73] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. *InternLM2 Technical Report*. arXiv. 2024. DOI: 10.48550/arXiv.2403.17297 (cit. on p. 344).

[74] Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. "An analysis of why cuckoo search does not bring any novel ideas to optimization". In: *Comput. Oper. Res.* 142 (2022), p. 105747. DOI: 10.1016/J.COR.2022.105747 (cit. on p. 4).

[75] Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by *bestial* metaphors". In: *Int. Trans. Oper. Res.* 30.6 (2023), pp. 2945–2971. DOI: 10.1111/ITOR.13176 (cit. on p. 4).

[76] Mats Carlsson, Sara Ceschia, Luca Di Gaspero, Rasmus Ø. Mikkelsen, Andrea Schaerf, and Thomas Jacob Riis Stidsen. "Exact and metaheuristic methods for a real-world examination timetabling problem". In: *J. Sched.* 26.4 (2023), pp. 353–367. DOI: 10.1007/S10951-023-00778-6 (cit. on p. 285).

[77] Cenk Çelik and Inci Sariçiçek. "Tabu Search for Parallel Machine Scheduling with Job Splitting". In: *Sixth International Conference on Information Technology: New Generations, ITNG 2009, Las Vegas, Nevada, USA, 27-29 April 2009*. Ed. by Shahram Latifi. IEEE Computer Society, 2009, pp. 183–188. DOI: 10.1109/ITNG.2009.271 (cit. on p. 43).

[78] Gjorgjina Cenikj, Gasper Petelin, and Tome Eftimov. "TransOptAS: Transformer-Based Algorithm Selection for Single-Objective Optimization". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. Ed. by Xiaodong Li and Julia Handl. ACM, 2024, pp. 403–406. DOI: 10.1145/3638530.3654191 (cit. on p. 42).

[79] Sara Ceschia, Luca Di Gaspero, Vincenzo Mazzaracchio, Giuseppe Policante, and Andrea Schaerf. "Solving a real-world nurse rostering problem by Simulated Annealing". In: *Operations Research for Health Care* 36 (2023), p. 100379. ISSN: 2211-6923. DOI: 10.1016/j.orhc.2023.100379 (cit. on pp. 179, 285).

[80] Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. "Multi-neighborhood simulated annealing for the home healthcare routing and scheduling problem". In: *Int. Trans. Oper. Res.* (2024). DOI: 10.1111/itor.13585 (cit. on pp. 47, 180, 201, 202, 204, 205, 207, 208, 210, 285).

[81] Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. "Multi-Neighborhood simulated annealing for the minimum interference frequency assignment problem". In: *EURO J. Comput. Optim.* 10 (2022), p. 100024. DOI: 10.1016/J.EJCO.2021.100024 (cit. on p. 285).

[82] Sara Ceschia, Luca Di Gaspero, Roberto Maria Rosati, and Andrea Schaerf. "Reinforcement Learning for Multi-Neighborhood Local Search in Combinatorial Optimization". In: *Machine Learning, Optimization, and Data Science - 9th International Conference, LOD 2023, Grasmere, UK, September 22-26, 2023, Revised Selected Papers, Part II*. Ed. by Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Gabriele La Malfa, Panos M. Pardalos, and Renato Umeton. Vol. 14506. Lecture Notes in Computer Science. Springer, 2023, pp. 206–221. DOI: 10.1007/978-3-031-53966-4_16 (cit. on pp. 42, 288, 295, 301).

[83] Sara Ceschia, Luca Di Gaspero, and Andrea Schaerf. "Educational timetabling: Problems, benchmarks, and state-of-the-art results". In: *Eur. J. Oper. Res.* 308.1 (2023), pp. 1–18. DOI: 10.1016/J.EJOR.2022.07.011 (cit. on pp. 14, 45, 59).

[84] Sara Ceschia, Margaretha Gansterer, Simona Mancini, and Antonella Meneghetti. "Solving the Online On-Demand Warehousing Problem". In: *Comput. Oper. Res.* 170 (2024), p. 106760. DOI: 10.1016/J.COR.2024.106760 (cit. on p. 3).

[85] Sara Ceschia, Kevin Roitero, Gianluca Demartini, Stefano Mizzaro, Luca Di Gaspero, and Andrea Schaerf. "Task design in complex crowdsourcing experiments: Item assignment optimization". In: *Comput. Oper. Res.* 148 (2022), p. 105995. DOI: 10.1016/J.COR.2022.105995 (cit. on p. 285).

[86] Sara Ceschia and Andrea Schaerf. "Local search for a multi-drop multi-container loading problem". In: *J. Heuristics* 19.2 (2013), pp. 275–294. DOI: 10.1007/S10732-011-9162-6 (cit. on p. 285).

[87] Sara Ceschia and Andrea Schaerf. "Multi-neighborhood simulated annealing for the capacitated facility location problem with customer incompatibilities". In: *Comput. Ind. Eng.* 188 (2024), p. 109858. DOI: 10.1016/J.CIE.2023.109858 (cit. on p. 285).

[88] Camilo Chacón Sartori, Christian Blum, Filippo Bistaffa, and Guillem Rodríguez Corominas. "Metaheuristics and Large Language Models Join Forces: Toward an Integrated Optimization Approach". In: *IEEE Access* 13 (2025), pp. 2058–2079. DOI: 10.1109/ACCESS.2024.3524176 (cit. on pp. 234, 240, 246, 331, 335, 340, 343, 346).

[89] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. "An Extension of STNWeb Functionality: On the Use of Hierarchical Agglomerative Clustering as an Advanced Search Space Partitioning Strategy". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. Ed. by Xiaodong Li and Julia Handl. ACM, 2024. DOI: 10.1145/3638529.3654084 (cit. on p. 170).

[90] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. "Large Language Models for the Automated Analysis of Optimization Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 160–168. DOI: 10.1145/3638529.3654086 (cit. on p. 37).

[91] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. "Large Language Models for the Automated Analysis of Optimization Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '24. New York, NY, USA: ACM, 2024, pp. 160–168. DOI: 10.1145/3638529.3654086 (cit. on pp. 237, 242, 326, 337, 340, 342).

[92] Camilo Chacón Sartori, Christian Blum, and Gabriela Ochoa. "Search Trajectory Networks Meet the Web: A Web Application for the Visual Comparison of Optimization Algorithms". In: *Proceedings of the 12th International Conference on Software and Computer Applications, ICSCA 2023, Kuantan, Malaysia, February 23-25, 2023*. ACM, 2023, pp. 89–96. DOI: 10.1145/3587828.3587843 (cit. on p. 37).

[93] Jian Chang, Lifang Wang, Jin-Kao Hao, and Yang Wang. "Parallel iterative solution-based tabu search for the obnoxious p-median problem". In: *Comput. Oper. Res.* 127 (2021), p. 105155. DOI: 10.1016/J.COR.2020.105155 (cit. on p. 63).

[94] Ping-Yu Chang, Purushothaman Damodaran, and Sharif Melouk. "Minimizing makespan on parallel batch processing machines". In: *Int. J. Prod. Res.* 42.19 (2004), pp. 4211–4220. DOI: 10.1080/00207540410001711863 (cit. on p. 43).

[95] Yuan-chin Ivan Chang. *A Survey: Potential Dimensionality Reduction Methods*. arXiv. 2025. arXiv: 2502.11036 [stat.OT]. URL: https://arxiv.org/abs/2502.11036 (cit. on pp. 205, 303).

[96] David Chanin, Anthony Hunter, and Oana-Maria Camburu. "Identifying Linear Relational Concepts in Large Language Models". In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*. Ed. by Kevin Duh, Helena Gómez-Adorno, and Steven Bethard. Association for Computational Linguistics, 2024, pp. 1524–1535. DOI: 10.18653/V1/2024.NAACL-LONG.85 (cit. on p. 32).

[97] Nuo Chen, Yuhan Li, Jianheng Tang, and Jia Li. *GraphWiz: An Instruction-Following Language Model for Graph Problems*. arXiv. 2024. DOI: 10.48550/arXiv.2402.16029 (cit. on pp. 243, 345).

[98] Steve Chen, Apoorv Gautam, and Florian Weig. *Bringing energy efficiency in the Fab*. Technical report. 2013. URL: https://www.mckinsey.com/~/media/mckinsey/dotcom/client_service/operations/pdfs/bringing_fabenergyefficiency.ashx (cit. on p. 97).

[99] Zijie Chen, Zhanchao Zhou, Yu Lu, Renjun Xu, Lili Pan, and Zhenzhong Lan. *UBER: Uncertainty-Based Evolution with Large Language Models for Automatic Heuristic Design*. arXiv. 2024. DOI: 10.48550/arXiv.2412.20694 (cit. on pp. 235, 246, 326, 336, 343, 344, 346).

[100] Bayi Cheng, Qi Wang, Shanlin Yang, and Xiaoxuan Hu. "An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes". In: *Appl. Soft Comput.* 13.2 (2013), pp. 765–772. DOI: 10.1016/J.ASOC.2012.10.021 (cit. on p. 43).

[101] Eddie Cheng and Jennifer L. Rich. *A home health care routing and scheduling problem*. Tech. rep. CAAM TR98–04. Rice University, 1998 (cit. on p. 46).

[102] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, Ion Stoica, et al. *Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference*. arXiv. 2024. DOI: 10.48550/arXiv.2403.04132 (cit. on p. 242).

[103] Samuel J. K. Chin, Matthias Winkenbach, and Akash Srivastava. *Learning to Deliver: a Foundation Model for the Montreal Capacitated Vehicle Routing Problem*. arXiv. 2024. DOI: 10.48550/arXiv.2403.00026 (cit. on pp. 235, 240, 246, 326, 336, 339, 341, 346).

[104] Fuh-Der Chou. "Minimising the total weighted tardiness for non-identical parallel batch processing machines with job release times and non-identical job sizes". In: *Eur. J. Ind. Eng.* 7.5 (2013), pp. 529–557. DOI: 10.1504/EJIE.2013.057380 (cit. on p. 43).

[105] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. "PaLM: scaling language modeling with pathways". In: *Journal of Machine Learning Research* 24.1 (2024). ISSN: 1532-4435. URL: http://jmlr.org/papers/v24/22-1144.html (cit. on p. 30).

[106] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, and Siddhartha Brahma. "Scaling Instruction-Finetuned Language Models". In: *Journal of Machine Learning Research* 25.70 (2024), pp. 1–53. URL: http://jmlr.org/papers/v25/23-0870.html (cit. on p. 30).

[107] Marcel Chwiałkowski, Benjamin Doerr, and Martin S. Krejca. "Runtime Analysis of the Compact Genetic Algorithm on the LeadingOnes Benchmark". In: *IEEE Transactions on Evolutionary Computation* (2025). DOI: 10.1109/TEVC.2025.3549929 (cit. on p. 40).

[108] Raffaele Cipriano, Luca Di Gaspero, and Agostino Dovier. "A Multi-paradigm Tool for Large Neighborhood Search". In: *Hybrid Metaheuristics*. Ed. by El-Ghazali Talbi. Vol. 434. Studies in Computational Intelligence. Springer, 2013, pp. 389–414. DOI: 10.1007/978-3-642-30671-6_15 (cit. on p. 289).

[109] Mohamed Cissé, Semih Yalçındağ, Yannick Kergosien, Evren Şahin, Christophe Lenté, and Andrea Matta. "OR problems related to Home Health Care: A review of relevant routing and scheduling problems". In: *Operations Research for Health Care* 13-14 (2017), pp. 1–22. DOI: 10.1016/j.orhc.2017.06.001 (cit. on p. 46).

[110] Yoram Clapper, Joost Berkhout, René Bekker, and Dennis Moeke. "A model-based evolutionary algorithm for home health care scheduling". In: *Comput. Oper. Res.* 150 (2023), p. 106081. DOI: 10.1016/J.COR.2022.106081 (cit. on pp. 47, 49).

[111] Christopher W. Cleghorn and Gabriela Ochoa. "Understanding parameter spaces using local optima networks: a case study on particle swarm optimization". In: *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*. Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1657–1664. DOI: 10.1145/3449726.3463145 (cit. on p. 41).

[112] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. *Training Verifiers to Solve Math Word Problems*. arXiv. 2021. DOI: 10.48550/arXiv.2110.14168 (cit. on p. 245).

[113] Harry Cohn and Mark Fielding. "Simulated Annealing: Searching for an Optimal Temperature Schedule". In: *SIAM Journal on Optimization* 9.3 (1999), pp. 779–802. DOI: 10.1137/S1052623497329683 (cit. on pp. 26, 77).

[114] Chris Cooper, Andrew Booth, Nicky Britten, and Ruth Garside. "A comparison of results of empirical studies of supplementary search techniques and recommendations in review methodology handbooks: a methodological review". In: *Systematic Reviews* 6.1 (2017), p. 234. ISSN: 2046-4053. DOI: 10.1186/s13643-017-0625-1 (cit. on pp. 229, 232).

[115] Alvaro H. C. Correia, Daniel E. Worrall, and Roberto Bondesan. "Neural Simulated Annealing". In: *International Conference on Artificial Intelligence and Statistics, 25-27 April 2023, Palau de Congressos, Valencia, Spain*. Ed. by Francisco J. R. Ruiz, Jennifer G. Dy, and Jan-Willem van de Meent. Vol. 206. Proceedings of Machine Learning Research. PMLR, 2023, pp. 4946–4962. URL: https://proceedings.mlr.press/v206/correia23a.html (cit. on p. 42).

[116] Arnaud De Coster, Nysret Musliu, Andrea Schaerf, Johannes Schoisswohl, and Kate Smith-Miles. "Algorithm selection and instance space analysis for curriculum-based course timetabling". In: *J. Sched.* 25.1 (2022), pp. 35–58. DOI: 10.1007/S10951-021-00701-X (cit. on pp. 42, 155, 157).

[117] G. A. Croes. "A Method for Solving Traveling-Salesman Problems". In: *Operations Research* 6.6 (1958), pp. 791–812. DOI: 10.1287/opre.6.6.791 (cit. on p. 82).

[118] Oliverio Cruz-Mejía and Adam N. Letchford. "A survey on exact algorithms for the maximum flow and minimum-cost flow problems". In: *Networks* 82.2 (2023), pp. 167–176. DOI: 10.1002/net.22169 (cit. on p. 123).

[119] Longchao Da, Kuanru Liou, Tiejin Chen, Xuesong Zhou, Xiangyong Luo, Yezhou Yang, and Hua Wei. "Open-ti: open traffic intelligence with augmented language model". In: *International Journal of Machine Learning and Cybernetics* 15.10 (2024), pp. 4761–4786. ISSN: 1868-808X. DOI: 10.1007/s13042-024-02190-8 (cit. on pp. 234, 237, 240–242, 246, 326, 335, 339–341, 346).

[120] Francesca Da Ros and Luca Di Gaspero. "Exploring the Potential of JuLeS: A White Box Framework for Local Search Metaheuristics". In: *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*. Ed. by Sara Silva and Luís Paquete. ACM, 2023, pp. 191–194. DOI: 10.1145/3583133.3590676 (cit. on p. 288).

[121] Francesca Da Ros and Luca Di Gaspero. "Local Search Strategies for Multi-Objective Flowshop Scheduling: Introducing Pareto Late Acceptance Hill Climbing". In: *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*. Ed. by Sara Silva and Luís Paquete. ACM, 2023, pp. 61–62. DOI: 10.1145/3583133.3596428 (cit. on p. 288).

[122] Francesca Da Ros, Luca Di Gaspero, Kevin Roitero, David La Barbera, Stefano Mizzaro, Vincenzo Della Mea, Francesca Valent, and Laura Deroma. "Supporting Fair and Efficient Emergency Medical Services in a Large Heterogeneous Region". In: *J. Heal. Informatics Res.* 8.2 (2024), pp. 400–437. DOI: 10.1007/S41666-023-00154-1 (cit. on p. 285).

[123] Parag Pravin Dakle, Serdar Kadıoğlu, Karthik Uppuluri, Regina Politi, Preethi Raghavan, SaiKrishna Rallabandi, and Ravisutha Srinivasamurthy. "Ner4Opt: Named Entity Recognition for Optimization Modelling from Natural Language". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Cham: Springer Nature Switzerland, 2023, pp. 299–319. ISBN: 978-3-031-33271-5. DOI: 10.1007/978-3-031-33271-5_20 (cit. on p. 330).

[124] Purushothaman Damodaran, Krishnaswami Srihari, and Sarah S. Lam. "Scheduling a capacitated batch-processing machine to minimize makespan". In: *Robotics and Computer-Integrated Manufacturing* 23.2 (2007), pp. 208–216. DOI: 10.1016/j.rcim.2006.02.012 (cit. on p. 43).

[125] Purushothaman Damodaran and Mario C. Vélez-Gallego. "A simulated annealing algorithm to minimize makespan of parallel batch processing machines with unequal job ready times". In: *Expert Syst. Appl.* 39.1 (2012), pp. 1451–1458. DOI: 10.1016/J.ESWA.2011.08.029 (cit. on pp. 43–45, 114, 154, 161, 310–313).

[126] Ketil Danielsen and Lars Magnus Hvattum. "Solution-based versus attribute-based tabu search for binary integer programming". In: *International Transactions in Operational Research* 32.6 (2025), pp. 3780–3800. DOI: https://doi.org/10.1111/itor.70011 (cit. on pp. 24, 40, 59).

[127] George B. Dantzig and John H. Ramser. "The truck dispatching problem". In: *Management science* 6.1 (1959), pp. 80–91. URL: http://www.jstor.org/stable/2627477 (cit. on p. 122).

[128] Philipp Danzinger, Tobias Geibinger, Florian Mischek, and Nysret Musliu. "Solving the Test Laboratory Scheduling Problem with Variable Task Grouping". In: *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling, Nancy, France, October 26-30, 2020*. Ed. by J. Christopher Beck, Olivier Buffet, Jörg Hoffmann, Erez Karpas, and Shirin Sohrabi. AAAI Press, 2020, pp. 357–365. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/6681 (cit. on p. 3).

[129] Fabio Daolio, Sébastien Vérel, Gabriela Ochoa, and Marco Tomassini. "Local optima networks and the performance of iterated local search". In: *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11,*

2012. Ed. by Terence Soule and Jason H. Moore. ACM, 2012, pp. 369–376. DOI: 10.1145/2330163.2330217 (cit. on p. 40).

[130] Tomas De La Rosa, Sriram Gopalakrishnan, Alberto Pozanco, Zhen Zeng, and Daniel Borrajo. *TRIP-PAL: Travel Planning with Guarantees by Combining Large Language Models and Automated Planners*. arXiv. 2024. DOI: 10.48550/arXiv.2406.10196 (cit. on pp. 234, 241, 246, 326, 335, 340, 346).

[131] Stefano De Sabbata, Stefano Mizzaro, and Kevin Roitero. *Geospatial Mechanistic Interpretability of Large Language Models*. arXiv. 2025. DOI: 10.48550/ARXIV.2505.03368. URL: https://doi.org/10.48550/arXiv.2505.03368 (cit. on p. 223).

[132] I. De Zarzà, J. De Curtò, Gemma Roig, and Carlos T. Calafate. "Optimized Financial Planning: Integrating Individual and Cooperative Budgeting Models with LLM Recommendations". In: *AI* 5.1 (2024), pp. 91–114. ISSN: 2673-2688. DOI: 10.3390/ai5010006 (cit. on pp. 234, 240, 326, 335, 340, 346).

[133] Jérémy Decerle, Olivier Grunder, Amir Hajjam El Hassani, and Oussama Barakat. "A memetic algorithm for a home health care routing and scheduling problem". In: *Operations Research for Health Care* 16 (2018), pp. 59–71. DOI: 10.1016/j.orhc.2018.01.004 (cit. on pp. 47, 49).

[134] DeepSeek-AI, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, and Qiushi Du. *DeepSeek LLM: Scaling Open-Source Language Models with Longtermism*. arXiv. 2024. DOI: 10.48550/arXiv.2401.02954 (cit. on pp. 238, 343).

[135] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, and Damai Dai. *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*. arXiv. 2024. DOI: 10.48550/arXiv.2405.04434 (cit. on p. 343).

[136] DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu, Yukun Li, and Huazuo Gao. *DeepSeek-Coder-V2: Breaking the Barrier of Closed-Source Models in Code Intelligence*. arXiv. 2024. DOI: 10.48550/arXiv.2406.11931 (cit. on p. 343).

[137] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *The Journal of Machine learning research* 7 (2006), pp. 1–30 (cit. on p. 35).

[138] Renzhong Deng, Weijie Zheng, Mingfeng Li, Jie Liu, and Benjamin Doerr. "Runtime Analysis for State-of-the-Art Multi-objective Evolutionary Algorithms on the Subset Selection Problem". In: *Parallel Problem Solving from Nature - PPSN XVIII - 18th International Conference, PPSN 2024, Hagenberg, Austria, September 14-18, 2024, Proceedings, Part III*. Ed. by Michael Affenzeller, Stephan M. Winkler, Anna V. Kononova, Heike Trautmann, Tea Tusar, Penousal Machado, and Thomas Bäck. Vol. 15150. Lecture Notes in Computer Science. Springer, 2024, pp. 264–279. DOI: 10.1007/978-3-031-70071-2_17 (cit. on p. 40).

[139] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: ACL, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423 (cit. on pp. 30, 248).

[140] Neel Dhanaraj, Minseok Jeon, Jeon Ho Kang, Stefanos Nikolaidis, and Satyandra K. Gupta. "Preference Elicitation and Incorporation for Human-Robot Task Scheduling". In: *2024 IEEE 20th International Conference on Automation Science and Engineering (CASE)*. Bari, Italy: IEEE, 2024, pp. 3103–3110. DOI: 10.1109/CASE59546.2024.10711695 (cit. on pp. 235, 239, 246, 326, 335, 346).

[141] Luca Di Gaspero, Marco Chiarandini, and Andrea Schaerf. "A Study on the Short-Term Prohibition Mechanisms in Tabu Search". In: *ECAI 2006, 17th European Conference on Artificial Intelligence, August 29 - September 1, 2006, Riva del Garda, Italy, Including Prestigious Applications of Intelligent Systems (PAIS 2006), Proceedings*. Ed. by Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso. Vol. 141. Frontiers in Artificial Intelligence and Applications. IOS Press, 2006, pp. 83–87. URL: http://www.booksonline.iospress.nl/Content/View.aspx?piid=1651 (cit. on p. 40).

[142] Luca Di Gaspero, Giacomo di Tollo, Andrea Roli, and Andrea Schaerf. "Hybrid Local Search for Constrained Financial Portfolio Selection Problems". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 4th International Conference, CPAIOR 2007, Brussels, Belgium, May 23-26, 2007, Proceedings*. Ed. by Pascal Van Hentenryck and Laurence A. Wolsey. Vol. 4510. Lecture Notes in Computer Science. Springer, 2007, pp. 44–58. DOI: 10.1007/978-3-540-72397-4_4 (cit. on p. 285).

[143] Luca Di Gaspero and Andrea Roli. "Flexible Stochastic Local Search for Haplotype Inference". In: *Learning and Intelligent Optimization, Third International Conference, LION 3, Trento, Italy, January 14-18, 2009. Selected Papers*. Ed. by Thomas Stützle. Vol. 5851. Lecture Notes in Computer Science. Springer, 2009, pp. 74–88. DOI: 10.1007/978-3-642-11169-3_6 (cit. on p. 285).

[144] Luca Di Gaspero, Andrea Roli, and Andrea Schaerf. "EasyAnalyzer: An Object-Oriented Framework for the Experimental Analysis of Stochastic Local Search Algorithms". In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*. Ed. by Thomas Stützle, Mauro Birattari, and Holger H. Hoos. Vol. 4638. Lecture Notes in Computer Science. Springer, 2007, pp. 76–90. DOI: 10.1007/978-3-540-74446-7_6 (cit. on p. 288).

[145] Luca Di Gaspero and Andrea Schaerf. "A composite-neighborhood tabu search approach to the traveling tournament problem". In: *J. Heuristics* 13.2 (2007), pp. 189–207. DOI: 10.1007/S10732-006-9007-X (cit. on p. 288).

[146] Luca Di Gaspero and Andrea Schaerf. "EasyLocal++: An object-oriented framework for the design of Local Search Algorithms and Metaheuristics". In: *Pro-*

ceedings of the 4th Metaheuristics International Conference (MIC-2001). Vol. 2. Porto, Portugal, 2001, pp. 287–292 (cit. on p. 287).

[147] Luca Di Gaspero and Andrea Schaerf. "EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms". In: *Softw. Pract. Exp.* 33.8 (2003), pp. 733–765. DOI: 10.1002/SPE.524 (cit. on p. 287).

[148] Luca Di Gaspero and Andrea Schaerf. "EasySyn++: A Tool for Automatic Synthesis of Stochastic Local Search Algorithms". In: *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop, SLS 2007, Brussels, Belgium, September 6-8, 2007, Proceedings*. Ed. by Thomas Stützle, Mauro Birattari, and Holger H. Hoos. Vol. 4638. Lecture Notes in Computer Science. Springer, 2007, pp. 177–181. DOI: 10.1007/978-3-540-74446-7_13 (cit. on p. 288).

[149] Luca Di Gaspero and Andrea Schaerf. "Neighborhood Portfolio Approach for Local Search Applied to Timetabling Problems". In: *J. Math. Model. Algorithms* 5.1 (2006), pp. 65–89. DOI: 10.1007/S10852-005-9032-Z (cit. on p. 288).

[150] Luca Di Gaspero and Andrea Schaerf. "Writing local search algorithms using EASYLOCAL++". In: *Optimization Software Class Libraries*. Boston, MA: Springer, 2002, pp. 155–175 (cit. on p. 287).

[151] Luca Di Gaspero and Tommaso Urli. "A CP/LNS Approach for Multi-day Homecare Scheduling Problems". In: *Hybrid Metaheuristics - 9th International Workshop, HM 2014, Hamburg, Germany, June 11-13, 2014. Proceedings*. Ed. by Maria J. Blesa, Christian Blum, and Stefan Voß. Vol. 8457. Lecture Notes in Computer Science. Springer, 2014, pp. 1–15. DOI: 10.1007/978-3-319-07644-7_1 (cit. on pp. 46, 47).

[152] Steven Diamond and Stephen Boyd. "CVXPY: a python-embedded modeling language for convex optimization". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2909–2913. ISSN: 1532-4435. DOI: 10.5555/2946645.3007036 (cit. on pp. 336, 337).

[153] DIMACS. *The Second DIMACS International Algorithm Implementation Challenge: General Information*. Accessed on 26 Oct 2025. 1992. URL: http://archive.dimacs.rutgers.edu/pub/challenge/gen_info.tex (cit. on p. 256).

[154] Xuan-Dung Doan. *VTCC-NLP at NL4Opt competition subtask 1: An Ensemble Pretrained language models for Named Entity Recognition*. arXiv. 2022. DOI: 10.48550/arXiv.2212.07219 (cit. on pp. 235, 243, 326, 335, 345).

[155] Benjamin Doerr. "A Gentle Introduction to Theory (for Non-Theoreticians)". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. Ed. by Xiaodong Li and Julia Handl. ACM, 2024, pp. 800–829. DOI: 10.1145/3638530.3648402 (cit. on p. 77).

[156] Benjamin Doerr, Taha El Ghazi El Houssaini, Amirhossein Rajabi, and Carsten Witt. "How Well Does the Metropolis Algorithm Cope With Local Optima?" In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*. Ed. by Sara Silva and Luís Paquete. ACM, 2023, pp. 1000–1008. DOI: 10.1145/3583131.3590390 (cit. on p. 40).

[157] Benjamin Doerr, Martin S. Krejca, and Nguyen Vu. "Superior Genetic Algorithms for the Target Set Selection Problem Based on Power-Law Parameter Choices and Simple Greedy Heuristics". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. Ed. by Xiaodong Li and Julia Handl. ACM, 2024. DOI: 10.1145/3638529.3654140 (cit. on p. 77).

[158] Tansel Dökeroglu, Tayfun Kucukyilmaz, and El-Ghazali Talbi. "Hyper-heuristics: A survey and taxonomy". In: *Comput. Ind. Eng.* 187 (2024), p. 109815. DOI: 10.1016/J.CIE.2023.109815 (cit. on pp. 27, 77).

[159] Marco Dorigo, Mauro Birattari, and Thomas Stützle. "Ant colony optimization". In: *IEEE Comput. Intell. Mag.* 1.4 (2006), pp. 28–39. DOI: 10.1109/MCI.2006.329691. URL: https://doi.org/10.1109/MCI.2006.329691 (cit. on p. 13).

[160] John H. Drake, Ahmed Kheiri, Ender Özcan, and Edmund K. Burke. "Recent advances in selection hyper-heuristics". In: *Eur. J. Oper. Res.* 285.2 (2020), pp. 405–428. DOI: 10.1016/J.EJOR.2019.07.073 (cit. on pp. 27, 77).

[161] Johann Dréo, Arnaud Liefooghe, Sébastien Vérel, Marc Schoenauer, Juan Julián Merelo Guervós, Alexandre Quemy, Benjamin Bouvier, and Jan Gmys. "Paradiseo: from a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo". In: *GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021*. Ed. by Krzysztof Krawiec. ACM, 2021, pp. 1522–1530. DOI: 10.1145/3449726.3463276 (cit. on pp. 286, 287).

[162] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. "PaLM-E: an embodied multimodal language model". In: *Proceedings of the 40th International Conference on Machine Learning*. ICML'23. Honolulu, Hawaii, USA: JMLR.org, 2023. DOI: 10.5555/3618408.3618748 (cit. on p. 30).

[163] Aaron M. Drucker, Patrick Fleming, and An-Wen Chan. "Research Techniques Made Simple: Assessing Risk of Bias in Systematic Reviews". In: *Journal of Investigative Dermatology* 136.11 (2016), e109–e114. ISSN: 0022-202X. DOI: 10.1016/j.jid.2016.08.021 (cit. on p. 247).

[164] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. "GLM: General Language Model Pretraining with Autoregressive Blank Infilling". In: *Proceedings of the 60th Annual Meeting of the ACL (Volume 1: Long Papers)*. Dublin, Ireland: ACL, 2022, pp. 320–335. DOI: 10.18653/v1/2022.acl-long.26 (cit. on pp. 237, 344).

[165] Juan José Durillo and Antonio J. Nebro. "jMetal: A Java framework for multi-objective optimization". In: *Adv. Eng. Softw.* 42.10 (2011), pp. 760–771. DOI: 10.1016/J.ADVENGSOFT.2011.05.014 (cit. on p. 287).

[166] Cagla F. Dursunoglu, Okan Arslan, Sebnem Manolya Demir, Bahar Yetis Kara, and Gilbert Laporte. "A unifying framework for selective routing problems". In: *Eur. J. Oper. Res.* 320.1 (2025), pp. 1–19. DOI: 10.1016/J.EJOR.2024.02.037 (cit. on p. 45).

[167] Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Omer F. Rana, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, and Rajiv Ranjan. "Explainable AI (XAI): Core Ideas, Techniques, and Solutions". In: *ACM Comput. Surv.* 55.9 (2023), 194:1–194:33. DOI: 10.1145/3561048 (cit. on p. 158).

[168] Agoston E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. "Parameter control in evolutionary algorithms". In: *IEEE Trans. Evol. Comput.* 3.2 (1999), pp. 124–141. DOI: 10.1109/4235.771166 (cit. on p. 77).

[169] Mohammed Elhenawy, Ahmad Abutahoun, Taqwa I. Alhadidi, Ahmed Jaber, Huthaifa I. Ashqar, Shadi Jaradat, Ahmed Abdelhay, Sebastien Glaser, and Andry Rakotonirainy. "Visual Reasoning and Multi-Agent Approach in Multimodal Large Language Models (MLLMs): Solving TSP and mTSP Combinatorial Challenges". In: *Machine Learning and Knowledge Extraction* 6.3 (2024), pp. 1894–1920. DOI: 10.3390/make6030093 (cit. on pp. 235–237, 240, 242, 246, 249, 326, 336, 340, 346).

[170] Jalel Euchi, Malek Masmoudi, and Patrick Siarry. "Home health care routing and scheduling problems: a literature review". In: *4OR* 20.3 (2022), pp. 351–389. DOI: 10.1007/S10288-022-00516-2 (cit. on pp. 46, 179).

[171] Zhenan Fan, Bissan Ghaddar, Xinglu Wang, Linzi Xing, Yong Zhang, and Zirui Zhou. *Artificial Intelligence for Operations Research: Revolutionizing the Operations Research Process*. arXiv. 2024. DOI: 10.48550/arXiv.2401.03244 (cit. on pp. 5, 50, 223, 246, 326).

[172] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. *Talk like a Graph: Encoding Graphs for Large Language Models*. arXiv. 2023. DOI: 10.48550/arXiv.2310.04560 (cit. on pp. 243, 345).

[173] Chris Fawcett and Holger H. Hoos. "Analysing differences between algorithm configurations through ablation". In: *J. Heuristics* 22.4 (2016), pp. 431–458. DOI: 10.1007/S10732-014-9275-9 (cit. on pp. 40, 149).

[174] Sándor P. Fekete and Jörg Schepers. "New classes of fast lower bounds for bin packing problems". In: *Math. Program.* 91.1 (2001), pp. 11–31. DOI: 10.1007/S101070100243 (cit. on p. 45).

[175] Victor Fernandez-Viagas, Rubén Ruiz, and Jose M. Framiñan. "A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation". In: *Eur. J. Oper. Res.* 257.3 (2017), pp. 707–721. DOI: 10.1016/J.EJOR.2016.09.055 (cit. on p. 60).

[176] Christian Fikar and Patrick Hirsch. "Home health care routing and scheduling: A review". In: *Comput. Oper. Res.* 77 (2017), pp. 86–95. DOI: 10.1016/J.COR.2016.07.019 (cit. on pp. 46, 179).

[177] Andreas Fink and Stefan Voß. "HotFrame: A heuristic optimization framework". In: *Optimization Software Class Libraries*. Kluwer Academic Publishers, 2002, pp. 81–154 (cit. on pp. 286, 287).

[178] Andreas Fink, Stefan Voß, and David L. Woodruff. "Metaheuristic Class Libraries". In: *Handbook of Metaheuristics*. Ed. by Fred W. Glover and Gary A. Kochenberger. Vol. 57. International Series in Operations Research & Management Science. Kluwer / Springer, 2003, pp. 515–535. DOI: 10.1007/0-306-48056-5_18 (cit. on p. 285).

[179] Gerd Finke, Vincent Jost, Maurice Queyranne, and András Sebö. "Batch processing with interval graph compatibilities between tasks". In: *Discret. Appl. Math.* 156.5 (2008), pp. 556–568. DOI: 10.1016/J.DAM.2006.03.039 (cit. on pp. 117, 118, 127).

[180] John W. Fowler and Lars Mönch. "A survey of scheduling with parallel batch (p-batch) processing". In: *Eur. J. Oper. Res.* 298.1 (2022), pp. 1–24. DOI: 10.1016/J.EJOR.2021.06.012 (cit. on pp. 43, 97).

[181] John W. Fowler, Nipa Phojanamongkolkij, Jeffery K. Cochran, and Douglas C. Montgomery. "Optimal batching in a wafer fabrication facility using a multiproduct G/G/c model with batch processing". In: *Int. J. Prod. Res.* 40.2 (2002), pp. 275–292. DOI: 10.1080/00207540110081489 (cit. on p. 43).

[182] Alberto Franzin and Thomas Stützle. "A landscape-based analysis of fixed temperature and simulated annealing". In: *Eur. J. Oper. Res.* 304.2 (2023), pp. 395–410. DOI: 10.1016/J.EJOR.2022.04.014 (cit. on pp. 26, 41, 56, 59).

[183] Alberto Franzin and Thomas Stützle. "Revisiting simulated annealing: A component-based analysis". In: *Comput. Oper. Res.* 104 (2019), pp. 191–206. DOI: 10.1016/J.COR.2018.12.015 (cit. on pp. 4, 13, 25, 40, 59, 84, 149, 287, 296).

[184] Yolanda Freire, Andrea Santamaría Laorden, Jaime Orejas Pérez, Margarita Gómez Sánchez, Víctor Díaz-Flores García, and Ana Suárez. "ChatGPT performance in prosthodontics: Assessment of accuracy and repeatability in answer generation". In: *The Journal of Prosthetic Dentistry* 131.4 (2024), 659.e1–659.e6. ISSN: 0022-3913. DOI: 10.1016/j.prosdent.2024.01.018 (cit. on p. 239).

[185] Eugene C. Freuder. "Conversational Modeling for Constraint Satisfaction". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.20 (2024), pp. 22592–22597. DOI: 10.1609/aaai.v38i20.30268 (cit. on pp. 247, 326).

[186] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph Drawing by Force-directed Placement". In: *Softw. Pract. Exp.* 21.11 (1991), pp. 1129–1164. DOI: 10.1002/SPE.4380211102 (cit. on p. 169).

[187] Yaping Fu, Jie Dong, Kaizhou Gao, Ponnuthurai Nagaratnam Suganthan, Min Huang, and Lihua Zhu. "A review on metaheuristics for solving home health care routing and scheduling problems". In: *Appl. Soft Comput.* 182 (2025), p. 113560. DOI: 10.1016/J.ASOC.2025.113560 (cit. on p. 179).

[188] Neeraj Gangwar and Nickvash Kani. "Highlighting Named Entities in Input for Auto-formulation of Optimization Problems". In: *Intelligent Computer Mathematics*. Cham: Springer Nature Switzerland, 2023, pp. 130–141. ISBN: 978-3-031-42753-4. DOI: 10.1007/978-3-031-42753-4_9 (cit. on pp. 235, 240, 243, 327, 337, 339, 345).

[189] Michael R. Garey, David S. Johnson, and Ravi Sethi. "The Complexity of Flowshop and Jobshop Scheduling". In: *Math. Oper. Res.* 1.2 (1976), pp. 117–129. DOI: 10.1287/MOOR.1.2.117 (cit. on p. 11).

[190] Gecode Team. *Gecode: Generic Constraint Development Environment*. Accessed: 27-06-2024. 2006. URL: http://www.gecode.org (cit. on p. 13).

[191] Tobias Geibinger, Lucas Kletzander, and Nysret Musliu. "Instance Space Analysis for the Generalized Assignment Problem". In: *Metaheuristics - 14th International Conference, MIC 2022, Syracuse, Italy, July 11-14, 2022, Proceedings*. Ed. by Luca Di Gaspero, Paola Festa, Amir Nakib, and Mario Pavone. Vol. 13838. Lecture Notes in Computer Science. Springer, 2022, pp. 421–435. DOI: 10.1007/978-3-031-26504-4_30. URL: https://doi.org/10.1007/978-3-031-26504-4%5C_30 (cit. on p. 40).

[192] Meta GenAI. *Camels in a Changing Climate: Enhancing LM Adaptation with Tulu 2*. arXiv. 2023. DOI: 10.48550/arXiv.2311.10702 (cit. on p. 342).

[193] Meta GenAI. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. arXiv. 2023. DOI: 10.48550/arXiv.2307.09288 (cit. on pp. 30, 237, 238, 341).

[194] Michel Gendreau, Alain Hertz, and Gilbert Laporte. "A Tabu Search Heuristic for the Vehicle Routing Problem". In: *Management Science* 40.10 (1994), pp. 1276–1290. DOI: 10.1287/mnsc.40.10.1276 (cit. on p. 62).

[195] Michel Gendreau and Jean-Yves Potvin. "Tabu Search". In: ed. by Michel Gendreau and Jean-Yves Potvin. Boston, MA: Springer US, 2010, pp. 41–59 (cit. on p. 63).

[196] Nadine Genet, Wienke Boerma, Madelon Kroneman, Allen Hutchinson, and Richard B Saltman, eds. *Home care across Europe: current structure and future challenges*. Regional Office for Europe, Scherfigsvej 8, DK-2100 Copenhagen, Denmark: World Health Organization, 2012 (cit. on p. 179).

[197] Felipe L. Gewers, Gustavo R. Ferreira, Henrique Ferraz de Arruda, Filipi Nascimento Silva, Cesar H. Comin, Diego R. Amancio, and Luciano da Fontoura Costa. "Principal Component Analysis: A Natural Approach to Data Exploration". In: *ACM Comput. Surv.* 54.4 (2022), 70:1–70:34. DOI: 10.1145/3447755 (cit. on pp. 205, 303).

[198] Gianpaolo Ghiani, Gianluca Solazzo, and Gianluca Elia. "Integrating Large Language Models and Optimization in Semi-Structured Decision Making: Methodology and a Case Study". In: *Algorithms* 17.12 (2024). DOI: 10.3390/a17120582 (cit. on pp. 235, 241, 242, 246, 248, 327, 336, 340, 346).

[199] Fred W. Glover. "Future paths for integer programming and links to artificial intelligence". In: *Comput. Oper. Res.* 13.5 (1986), pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1 (cit. on p. 3).

[200] Fred W. Glover and Manuel Laguna. "Tabu Search". In: Kluwer, 1997. ISBN: 978-0-7923-9965-0. DOI: 10.1007/978-1-4615-6089-0 (cit. on pp. 13, 24, 59, 60, 63, 294).

[201] Say Leng Goh, Graham Kendall, and Nasser R. Sabar. "Simulated annealing with improved reheating and learning for the post enrolment course timetabling problem". In: *J. Oper. Res. Soc.* 70.6 (2019), pp. 873–888. DOI: 10.1080/01605682.2018.1468862 (cit. on p. 25).

[202] Google. *PaLM 2 Technical Report*. arXiv. 2023. DOI: 10.48550/arXiv.2305.10403 (cit. on pp. 30, 237, 341).

[203] Florian Grenouilleau, Antoine Legrain, Nadia Lahrichi, and Louis-Martin Rousseau. "A set partitioning heuristic for the home health care routing and scheduling problem". In: *Eur. J. Oper. Res.* 275.1 (2019), pp. 295–303. DOI: 10.1016/J.EJOR.2018.11.025 (cit. on p. 47).

[204] Luca Grieco, Martin Utley, and Sonya Crowe. "Operational research applied to decisions in home health care: A systematic literature review". In: *J. Oper. Res. Soc.* 72.9 (2021), pp. 1960–1991. DOI: 10.1080/01605682.2020.1750311 (cit. on p. 46).

[205] Tias Guns. "Increasing Modeling Language Convenience with a Universal N-Dimensional Array: CPpy as a Python-Embedded Example". In: *Proceedings of the 18th Workshop on Constraint Modelling and Reformulation at CP (ModRef 2019)*. Stamford, CT, USA: ACP, 2019. URL: https://modref.github.io/ModRef2019.html (cit. on pp. 335, 337).

[206] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. "UniXcoder: Unified Cross-Modal Pre-training for Code Representation". In: *Proceedings of the 60th Annual Meeting of the ACL (Volume 1: Long Papers)*. Dublin, Ireland: ACL, 2022, pp. 7212–7225. DOI: 10.18653/v1/2022.acl-long.499 (cit. on pp. 238, 339).

[207] Pei-Fu Guo, Ying-Hsuan Chen, Yun-Da Tsai, and Shou-De Lin. *Towards Optimizing with Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2310.05204 (cit. on pp. 235, 327, 336, 339).

[208] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. Technical Documentation. 2023. URL: https://www.gurobi.com (cit. on pp. 13, 143, 337).

[209] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the Python in Science Conferences* (2008), pp. 11–15. DOI: 10.25080/TCWV9851 (cit. on p. 158).

[210] John Michael Hammersley and David Christopher Handscomb. *Monte Carlo methods*. London: Chapman and Hall, 1964 (cit. on p. 204).

[211] Yilun Hao, Yang Zhang, and Chuchu Fan. *Planning Anything with Rigor: General-Purpose Zero-Shot Planning with LLM-based Formalized Programming*. arXiv. 2024. DOI: 10.48550/arXiv.2410.12112 (cit. on pp. 235, 236, 240–242, 246, 327, 335, 340, 343, 346).

[212] Gonzalo E. Constante-Flores Hao Chen and Can Li. "Diagnosing infeasible optimization problems using large language models". In: *INFOR: Information Systems and Operational Research* 62.4 (2024), pp. 573–587. DOI: 10.1080/03155986.2024.2385189 (cit. on pp. 236, 242, 244, 327, 337, 340, 345).

[213] Charles R. Harris, K. Jarrod Millman, Stéfan van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. "Array programming with NumPy". In: *Nat.* 585 (2020), pp. 357–362. DOI: 10.1038/S41586-020-2649-2 (cit. on p. 158).

[214] William E. Hart, Jean-Paul Watson, and David L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260. ISSN: 1867-2957. DOI: 10.1007/s12532-011-0026-8 (cit. on pp. 244, 337).

[215] JiangLong He, Mamatha N, Shiv Vignesh, Deepak Kumar, and Akshay Uppal. *Linear programming word problems formulation using EnsembleCRF NER labeler and text generator with data augmentations*. arXiv. 2022. DOI: 10.48550/arXiv.2212.14657 (cit. on pp. 223, 235, 239, 240, 243, 327, 335, 339, 345).

[216] Michael Heinrich, Luisa Hofmann, Hansjörg Baurecht, Peter M. Kreuzer, Helge Knüttel, Michael F. Leitzmann, and Corinna Seliger. "Suicide risk and mortality among patients with cancer". In: *Nature Medicine* 28.4 (2022), pp. 852–859. ISSN: 1546-170X. DOI: 10.1038/s41591-022-01745-y (cit. on p. 228).

[217] Jack Heller. "Some Numerical Experiments for an M × J Flow Shop and its Decision-Theoretical Aspects". In: *Operations Research* 8.2 (1960), pp. 178–184. DOI: 10.1287/opre.8.2.178 (cit. on p. 65).

[218] Pascal Van Hentenryck. "Constraint and Integer Programming in OPL". In: *INFORMS J. Comput.* 14.4 (2002), pp. 345–372. DOI: 10.1287/IJOC.14.4.345.2826 (cit. on p. 44).

[219] Alain Hertz and Marino Widmer. "Guidelines for the use of meta-heuristics in combinatorial optimization". In: *Eur. J. Oper. Res.* 151.2 (2003), pp. 247–252. DOI: 10.1016/S0377-2217(02)00823-8 (cit. on p. 40).

[220] Gerhard Hiermann, Matthias Prandtstetter, Andrea Rendl, Jakob Puchinger, and Günther R. Raidl. "Metaheuristics for solving a multimodal home-healthcare scheduling problem". In: *Central Eur. J. Oper. Res.* 23.1 (2015), pp. 89–113. DOI: 10.1007/S10100-013-0305-8 (cit. on pp. 46, 49, 202, 203, 205, 210).

[221] John H. Holland. "Genetic algorithms". In: *Scholarpedia* 7.12 (2012), p. 1482. DOI: 10.4249/SCHOLARPEDIA.1482 (cit. on p. 13).

[222] Holger H. Hoos. "Programming by optimization". In: *Commun. ACM* 55.2 (2012), pp. 70–80. DOI: 10.1145/2076450.2076469 (cit. on pp. 13, 41, 77, 296).

[223] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. "Learning to Solve Arithmetic Word Problems with Verb Categorization". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: ACL, 2014, pp. 523–533. DOI: 10.3115/v1/D14-1058 (cit. on p. 245).

[224] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv. 2021. DOI: 10.48550/arXiv.2106.09685 (cit. on p. 241).

[225] Kanxin Hu, Yuxin Che, Tsan Sheng Ng, and Jie Deng. "Unrelated parallel batch processing machine scheduling with time requirements and two-dimensional packing constraints". In: *Comput. Oper. Res.* 162 (2024), p. 106474. DOI: 10.1016/J.COR.2023.106474 (cit. on p. 43).

[226] Yuwei Hu, Runlin Lei, Xinyi Huang, Zhewei Wei, and Yongchao Liu. *Scalable and Accurate Graph Reasoning with LLM-based Multi-Agents*. arXiv. 2024. DOI: 10.48550/arXiv.2410.05130 (cit. on pp. 235, 237, 241–243, 246, 327, 336, 340, 341, 345, 346).

[227] Beichen Huang, Xingyu Wu, Yu Zhou, Jibin Wu, Liang Feng, Ran Cheng, and Kay Chen Tan. *Exploring the True Potential: Evaluating the Black-box Optimization Capability of Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2404.06290 (cit. on pp. 235, 246, 327, 336, 340, 342, 344, 346).

[228] Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. *ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling*. arXiv. 2024. DOI: 10.48550/arXiv.2405.17743 (cit. on pp. 235, 240, 241, 243, 249, 327, 336, 342–345).

[229] Hao Huang et al. *The Open Cookbook for Top-Tier Code Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2411.04905 (cit. on pp. 238, 344).

[230] Sen Huang, Kaixiang Yang, Sheng Qi, and Rui Wang. "When large language model meets optimization". In: *Swarm and Evolutionary Computation* 90 (2024), p. 101663. DOI: 10.1016/j.swevo.2024.101663 (cit. on pp. 5, 50, 51, 223, 246, 327).

[231] Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. *LLMs for Mathematical Modeling: Towards Bridging the Gap between Natural and Mathematical Languages*. arXiv. 2025. DOI: 10.48550/arXiv.2405.13144 (cit. on pp. 243, 345).

[232] Yuxiao Huang, Wenjie Zhang, Liang Feng, Xingyu Wu, and Kay Chen Tan. *How Multimodal Integration Boost the Performance of LLM for Optimization: Case Study on Capacitated Vehicle Routing Problems*. arXiv. 2024. DOI: 10.48550/arXiv.2403.01757 (cit. on pp. 235, 240, 246, 327, 336, 340, 346).

[233] Zhehui Huang, Guangyao Shi, and Gaurav S. Sukhatme. *From Words to Routes: Applying Large Language Models to Vehicle Routing*. arXiv. 2024. DOI: 10.48550/arXiv.2403.10795 (cit. on pp. 235, 236, 240–243, 246, 327, 335, 340, 342, 345, 346).

[234] Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. "Sparse Autoencoders Find Highly Interpretable Features in Language Models". In: *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=F76bwRSLeK (cit. on p. 32).

[235] Lars Magnus Hvattum. "On the Value of Aspiration Criteria in Tabu Search". In: *Int. J. Appl. Metaheuristic Comput.* 7.4 (2016), pp. 39–49. DOI: 10.4018/IJAMC.2016100103 (cit. on pp. 40, 59).

[236] IBM. *IBM CPLEX Optimizer*. 2025. URL: https://www.ibm.com/de-de/analytics/cplex-optimizer (cit. on pp. 195, 212).

[237] *IBM ILOG CPLEX Optimization Studio, Getting Started with Scheduling in CPLEX Studio*. Accessed: 27-06-2024. IBM. 2017. URL: https://www.ibm.com/docs/en/icos/20.1.0?topic=kit-getting-started-scheduling-in-cplex-studio (cit. on p. 13).

[238] Sanghwan Jang. *Tag Embedding and Well-defined Intermediate Representation improve Auto-Formulation of Problem Description*. arXiv. 2022. DOI: 10.48550/ARXIV.2212.03575 (cit. on pp. 223, 235, 240, 243, 328, 335, 339, 345).

[239] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. *Mistral 7B*. arXiv. 2023. DOI: 10.48550/arXiv.2310.06825 (cit. on pp. 31, 237, 342).

[240] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. *Mixtral of Experts*. arXiv. 2024. DOI: 10.48550/arXiv.2401.04088 (cit. on pp. 31, 238, 342).

[241] Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. "LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch". In: *Proceedings of the Thirteenth International Conference on Learning Representations*. Singapore: Conference Organizers, 2025, pp. 3160–3172. URL: https://openreview.net/forum?id=9OMvtboTJg (cit. on pp. 235, 240, 243, 328, 335, 340, 342, 345).

[242] Shuo Jiang, Min Xie, and Jianxi Luo. *Large Language Models for Combinatorial Optimization of Design Structure Matrix*. arXiv. 2024. DOI: 10.48550/arXiv.2411.12571 (cit. on pp. 235, 246, 328, 336, 343, 346).

[243] Feng Jin, Shiji Song, and Cheng Wu. "A simulated annealing algorithm for single machine scheduling problems with family setups". In: *Comput. Oper. Res.* 36.7 (2009), pp. 2133–2138. DOI: 10.1016/J.COR.2008.08.001 (cit. on p. 24).

[244] Ming Jin, Bilgehan Sel, Fnu Hardeep, and Wotao Yin. "Democratizing Energy Management with LLM-Assisted Optimization Autoformalism". In: *2024 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. Oslo, Norway: IEEE Communications Society, 2024, pp. 258–263. DOI: 10.1109/SmartGridComm60555.2024.10738100 (cit. on pp. 235, 239, 241, 328, 336, 340, 346).

[245] Deddy Jobson and Yilin Li. "Investigating the Potential of Using Large Language Models for Scheduling". In: *Proceedings of the 1st ACM International Conference on AI-Powered Software*. AIware 2024. Porto de Galinhas, Brazil: ACM, 2024, pp. 170–171. ISBN: 9798400706851. DOI: 10.1145/3664646.3665084 (cit. on pp. 234, 241, 246, 328, 335, 340, 346).

[246] David S. Johnson. "A theoretician's guide to the experimental analysis of algorithms". In: *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, Proceedings of a DIMACS Workshop, USA, 1999*. Ed. by Michael H. Goldwasser, David S. Johnson, and Catherine C. McGeoch. Vol. 59. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1999, pp. 215–250. DOI: 10.1090/DIMACS/059/11 (cit. on pp. 4, 77).

[247] Da Ju, Song Jiang, Andrew Cohen, Aaron Foss, Sasha Mitts, Arman Zharmagambetov, Brandon Amos, Xian Li, Justine T Kao, Maryam Fazel-Zarandi, and Yuandong Tian. "To the Globe (TTG): Towards Language-Driven Guaranteed Travel Planning". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Miami, Florida, USA: ACL, 2024, pp. 240–249. DOI: 10.18653/v1/2024.emnlp-demo.25 (cit. on pp. 235, 240, 241, 244, 246, 328, 335, 344–346).

[248] Tianjie Ju, Weiwei Sun, Wei Du, Xinwei Yuan, Zhaochun Ren, and Gongshen Liu. "How Large Language Models Encode Context Knowledge? A Layer-Wise Probing Study". In: *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation, LREC/COLING 2024, 20-25 May, 2024, Torino, Italy*. Ed. by Nicoletta Calzolari, Min-Yen Kan, Véronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue. ELRA and ICCL, 2024, pp. 8235–8246. URL: https://aclanthology.org/2024.lrec-main.722 (cit. on p. 31).

[249] Tomihisa Kamada and Satoru Kawai. "An Algorithm for Drawing General Undirected Graphs". In: *Inf. Process. Lett.* 31.1 (1989), pp. 7–15. DOI: 10.1016/0020-0190(89)90102-6 (cit. on pp. 74, 169).

[250] Richard M. Karp. "Reducibility among Combinatorial Problems". In: ed. by Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger. Boston, MA: Springer US, 1972, pp. 85–103 (cit. on p. 127).

[251] Safia Kedad-Sidhoum, Yasmín Á. Ríos-Solís, and Francis Sourd. "Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates". In: *Eur. J. Oper. Res.* 189.3 (2008), pp. 1305–1316. DOI: 10.1016/J.EJOR.2006.05.052 (cit. on pp. 44, 45).

[252] Hanan Khalil, Mary Ameen, Charles Davies, and Chaojie Liu. "Implementing value-based healthcare: a scoping review of key elements, outcomes, and challenges for sustainable healthcare systems." eng. In: *Front Public Health* 13 (2025), p. 1514098. ISSN: 2296-2565 (Electronic); 2296-2565 (Linking). DOI: 10.3389/fpubh.2025.1514098 (cit. on p. 215).

[253] Muhammad Asif Khan and Layth Hamad. *On the Capability of LLMs in Combinatorial Optimization*. TechRxiv. 2024. DOI: 10.36227/techrxiv.173092026.60478567/v1 (cit. on pp. 236, 240, 246, 328, 336, 340, 346).

[254] Daisuke Kikuta, Hiroki Ikeuchi, Kengo Tajiri, and Yuusuke Nakano. "RouteExplainer: An Explanation Framework for Vehicle Routing Problem". In: *Advances in Knowledge Discovery and Data Mining*. Singapore: Springer Nature Singapore, 2024, pp. 30–42. ISBN: 978-981-97-2259-4. DOI: 10.1007/978-981-97-2259-4_3 (cit. on pp. 237, 242, 246, 328, 337, 340, 346).

[255] Najoung Kim, Roma Patel, Adam Poliak, Alex Wang, Patrick Xia, R. Thomas McCoy, Ian Tenney, Alexis Ross, Tal Linzen, Benjamin Van Durme, Samuel R. Bowman, and Ellie Pavlick. *Probing What Different NLP Tasks Teach Machines about Function Word Comprehension*. arXiv. 2019. URL: http://arxiv.org/abs/1904.11544 (cit. on p. 32).

[256] Scott Kirkpatrick, Daniel Gelatt, and Mario Vecchi. "Optimization by simulated annealing". In: *Science* 220 (1983), pp. 671–680. DOI: 10.1126/science.220.4598.671 (cit. on pp. 13, 24, 25, 77, 294).

[257] Lucas Kletzander, Tommaso Mannelli Mazzoli, and Nysret Musliu. "Metaheuristic algorithms for the bus driver scheduling problem with complex break constraints". In: *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022*. Ed. by Jonathan E. Fieldsend and Markus Wagner. ACM, 2022, pp. 232–240. DOI: 10.1145/3512290.3528876 (cit. on pp. 3, 40).

[258] Lucas Kletzander and Nysret Musliu. "Hyper-heuristics for personnel scheduling domains". In: *Artif. Intell.* 334 (2024), p. 104172. DOI: 10.1016/J.ARTINT.2024.104172 (cit. on pp. 42, 81, 82).

[259] Lucas Kletzander and Nysret Musliu. "Large-State Reinforcement Learning for Hyper-Heuristics". In: *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*. Ed. by Brian Williams, Yiling Chen, and Jennifer Neville. AAAI Press, 2023, pp. 12444–12452. DOI: 10.1609/AAAI.V37I10.26466 (cit. on pp. 29, 80).

[260] Lucas Kletzander and Nysret Musliu. "Solving the general employee scheduling problem". In: *Comput. Oper. Res.* 113 (2020). DOI: 10.1016/J.COR.2019.104794 (cit. on p. 14).

[261] Krzysztof Kochanek, Henryk Skarzynski, and Wiktor W Jedrzejczak. "Accuracy and Repeatability of ChatGPT Based on a Set of Multiple-Choice Questions on Objective Tests of Hearing". In: *Cureus* 16.5 (2024). DOI: 10.7759/cureus.59857 (cit. on p. 239).

[262] Shie-Gheun Koh, Pyung-Hoi Koo, Dong-Chun Kim, and Won-Suk Hur. "Scheduling a single batch processing machine with arbitrary job sizes and incompatible job families". In: *Int. J. Prod. Econ.* 98.1 (2005), pp. 81–96. DOI: 10.1016/j.ijpe.2004.10.001 (cit. on pp. 45, 114).

[263] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. *Large Language Models are Zero-Shot Reasoners*. arXiv. 2023. DOI: 10.48550/arXiv.2205.11916 (cit. on p. 30).

[264] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. "Parsing Algebraic Word Problems into Equations". In: *Transactions of the ACL* 3 (2015), pp. 585–597. DOI: 10.1162/tacl_a_00153 (cit. on p. 245).

[265] Anna Konovalenko and Lars Magnus Hvattum. "Exploring Tabu Tenure Policies with Machine Learning". In: *Electronics* 14.13 (2025). DOI: 10.3390/electronics14132642 (cit. on p. 40).

[266] Ghazale Kordi, Ali Divsalar, and Saeed Emami. "Multi-objective home health care routing: a variable neighborhood search method". In: *Optim. Lett.* 17.9 (2023), pp. 2257–2298. DOI: 10.1007/S11590-023-01993-Y (cit. on pp. 48, 49).

[267] Sebastian Kosch and J. Christopher Beck. "A New MIP Model for Parallel-Batch Scheduling with Non-identical Job Sizes". In: *Integration of AI and OR Techniques in Constraint Programming - 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings*. Ed. by Helmut Simonis. Vol. 8451. Lecture Notes in Computer Science. Springer, 2014, pp. 55–70. DOI: 10.1007/978-3-319-07046-9_5 (cit. on p. 43).

[268] Fajri Koto, Jey Han Lau, and Timothy Baldwin. "Discourse Probing of Pretrained Language Models". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*. Ed. by Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou. Association for Computational Linguistics, 2021, pp. 3849–3864. DOI: 10.18653/V1/2021.NAACL-MAIN.301 (cit. on p. 32).

[269] Hanane Krim, Nicolas Zufferey, Jean-Yves Potvin, Rachid Benmansour, and David Duvivier. "Tabu search for a parallel-machine scheduling problem with periodic maintenance, job rejection and weighted sum of completion times". In: *J. Sched.* 25.1 (2022), pp. 89–105. DOI: 10.1007/S10951-021-00711-9 (cit. on pp. 43, 139).

[270] Wieslaw Kubiak. "On a conjecture for the university timetabling problem". In: *Discret. Appl. Math.* 299 (2021), pp. 26–49. DOI: 10.1016/J.DAM.2021.04.010 (cit. on p. 11).

[271] Alberto Francisco Kummer. *A study on the home care routing and scheduling problem*. Ph.D. Thesis. Universidade Federal do Rio Grande do Sul. 2021 (cit. on pp. 47, 202, 205, 210).

[272] Philippe Laban, Alexander R Fabbri, Caiming Xiong, and Chien-Sheng Wu. *Summary of a Haystack: A Challenge to Long-Context LLMs and RAG Systems*. arXiv. 2024. DOI: 10.48550/arXiv.2407.01370 (cit. on p. 31).

[273] Philippe Laborie, Jerome Rogerie, Paul Shaw, and Petr Vilím. "IBM ILOG CP optimizer for scheduling - 20+ years of scheduling with constraints at IBM/ILOG". In: *Constraints An Int. J.* 23.2 (2018), pp. 210–250. DOI: 10.1007/S10601-018-9281-X (cit. on p. 143).

[274] Marie-Louise Lackner, Christoph Mrkvicka, Nysret Musliu, Daniel Walkiewicz, and Felix Winter. *Benchmark instances and models for the Oven Scheduling Problem [Dataset]*. Zenodo. 2022. DOI: 10.5281/zenodo.7456937. (cit. on pp. 44, 129).

[275] Marie-Louise Lackner, Christoph Mrkvicka, Nysret Musliu, Daniel Walkiewicz, and Felix Winter. "Exact methods for the Oven Scheduling Problem". In: *Constraints An Int. J.* 28.2 (2023), pp. 320–361. DOI: 10.1007/S10601-023-09347-2 (cit. on pp. 43, 97, 98, 104, 128, 136–138, 142, 143, 147, 154, 160, 168, 173, 283, 307).

[276] Manuel Laguna, Rafael Martí, Anna Martinez-Gavara, Sergio Perez-Peló, and Mauricio G. C. Resende. *20 years of Greedy Randomized Adaptive Search Procedures with Path Relinking*. arXiv. 2023. DOI: 10.48550/arXiv.2312.12663 (cit. on p. 13).

[277] Han Lai, Bo Wang, Jiaqi Liu, Feijuan He, Chenxi Zhang, Haohan Liu, and Haoran Chen. *Solving Mathematical Problems Using Large Language Models: A Survey*. SSRN. 2024. DOI: 10.2139/ssrn.5002356 (cit. on pp. 50, 246, 328).

[278] Chaymaa Lamini, Said Benhlima, and Ali Elbekri. "Genetic Algorithm Based Approach for Autonomous Mobile Robot Path Planning". In: *Procedia Computer Science* 127 (2018), pp. 180–189. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.01.113 (cit. on p. 3).

[279] Sophie Lasfargeas, Caroline Gagné, and Aymen Sioud. "Solving the Home Health Care Problem with Temporal Precedence and Synchronization". In: *Bioinspired Heuristics for Optimization*. Cham: Springer International Publishing, 2019, pp. 251–267. ISBN: 978-3-319-95104-1. DOI: 10.1007/978-3-319-95104-1_16 (cit. on pp. 47, 180).

[280] Yuri Cossich Lavinas, Claus Aranha, and Gabriela Ochoa. "Search Trajectories Networks of Multiobjective Evolutionary Algorithms". In: *Applications of Evolutionary Computation - 25th European Conference, EvoApplications 2022, Held as Part of EvoStar 2022, Madrid, Spain, April 20-22, 2022, Proceedings*. Ed. by Juan Luis Jiménez Laredo, José Ignacio Hidalgo, and Kehinde O. Babaagba. Vol. 13224. Lecture Notes in Computer Science. Springer, 2022, pp. 223–238. DOI: 10.1007/978-3-031-02462-7_15 (cit. on p. 37).

[281] Yuri Cossich Lavinas, Marcelo Ladeira, Gabriela Ochoa, and Claus Aranha. "Multiobjective Evolutionary Component Effect on Algorithm Behaviour". In: *ACM Trans. Evol. Learn. Optim.* 4.2 (2024), p. 8. DOI: 10.1145/3612933 (cit. on p. 40).

[282] Connor Lawless, Yingxi Li, Anders Wikum, Madeleine Udell, and Ellen Vitercik. *LLMs for Cold-Start Cutting Plane Separator Configuration*. arXiv. 2024. DOI: 10.48550/arXiv.2412.12038 (cit. on pp. 235, 246, 248, 328, 336).

[283] Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. ""I Want It That Way": Enabling Interactive Decision Support Using Large Language Models and Constraint Programming". In: *ACM Transactions on Interactive Intelligent Systems* 14.3 (2024), pp. 8432–8448. DOI: 10.1145/3685053 (cit. on pp. 223, 235, 240, 244, 246, 328, 337, 339, 340, 345, 346).

[284] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C.H. Hoi. *CodeRL: mastering code generation through pretrained models and deep reinforcement learning*. New Orleans, LA, USA, 2022. DOI: 10.5555/3600270.3601819 (cit. on pp. 241, 339).

[285] Nuno Leite, Fernando Melício, and Agostinho C. Rosa. "A fast simulated annealing algorithm for the examination timetabling problem". In: *Expert Syst. Appl.* 122 (2019), pp. 137–151. DOI: 10.1016/J.ESWA.2018.12.048 (cit. on p. 24).

[286] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. arXiv. 2019. URL: https://arxiv.org/abs/1910.13461 (cit. on pp. 238, 339).

[287] Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. *Large Language Models for Supply Chain Optimization*. arXiv. 2023. DOI: 10.48550/arXiv.2307.03875 (cit. on pp. 235, 241, 328, 337, 339, 340, 346).

[288] Bohang Li, Kai Zhang, Yiping Sun, and Jianke Zou. "Research on Travel Route Planning Optimization based on Large Language Model". In: *2024 6th International Conference on Data-driven Optimization of Complex Systems (DOCS)*. Hangzhou, China: IEEE, 2024, pp. 352–357. DOI: 10.1109/DOCS63458.2024.10704489 (cit. on pp. 235, 236, 246, 329, 336, 346).

[289] Daoyang Li, Haiyan Zhao, Qingcheng Zeng, and Mengnan Du. *Exploring Multilingual Probing in Large Language Models: A Cross-Language Analysis*. arXiv. 2025. URL: https://arxiv.org/abs/2409.14459 (cit. on p. 31).

[290] Qingyang Li, Lele Zhang, and Vicky Mak-Hau. *Synthesizing mixed-integer linear programming models from natural language descriptions*. arXiv. 2023. DOI: 10.48550/arXiv.2311.15271 (cit. on pp. 234, 235, 240, 243, 329, 335, 339, 341, 345).

[291] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. *StarCoder: May the Source Be with You!* arXiv. 2023. DOI: 10.48550/arXiv.2305.06161 (cit. on p. 344).

[292] Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. *Towards Foundation Models for Mixed Integer Linear Programming*. arXiv. 2024. DOI: 10.48550/arXiv.2410.08288 (cit. on pp. 235, 241, 329, 336, 340).

[293] Xiaolin Li, Huaping Chen, Bing Du, and Qi Tan. "Heuristics to schedule uniform parallel batch processing machines with dynamic job arrivals". In: *Int. J. Comput. Integr. Manuf.* 26.5 (2013), pp. 474–486. DOI: 10.1080/0951192X.2012.731612 (cit. on pp. 45, 114).

[294] XiaoLin Li, YuPeng Li, and YanLi Huang. "Heuristics and lower bound for minimizing maximum lateness on a batch processing machine with incompatible job families". In: *Comput. Oper. Res.* 106 (2019), pp. 91–101. DOI: 10.1016/J.COR.2019.02.012 (cit. on p. 45).

[295] Xin Li, Qizhi Chu, Yubin Chen, Yang Liu, Yaoqi Liu, Zekai Yu, Weize Chen, Chen Qian, Chuan Shi, and Cheng Yang. *Facilitating Large Language Model-based Graph Analysis via Multi-Agent Collaboration*. arXiv. 2024. DOI: 10.48550/arXiv.2410.18032 (cit. on pp. 234, 235, 240, 241, 243, 246, 329, 335, 341, 345, 346).

[296] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Barcelona, Spain: ACL, 2004, pp. 74–81. URL: https://aclanthology.org/W04-1013 (cit. on p. 340).

[297] Hanli Lin, Liqun Zhang, Ruizhi Zheng, and Yishan Zheng. "The prevalence, metabolic risk and effects of lifestyle intervention for metabolically healthy obesity: a systematic review and meta-analysis: A PRISMA-compliant article". In: *Medicine* 96.47 (2017). DOI: 10.1097/MD.0000000000008838 (cit. on p. 228).

[298] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research* 23.54 (2022), pp. 1–9. URL: http://jmlr.org/papers/v23/21-0888.html (cit. on pp. 41, 235, 332).

[299] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. "Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems". In: *Proceedings of the 55th Annual Meeting of the ACL (Volume 1: Long Papers)*. Vancouver, Canada: ACL, 2017, pp. 158–167. DOI: 10.18653/v1/P17-1015 (cit. on p. 245).

[300] Chang Liu, Kate Smith-Miles, Tony Wauters, and Alysson M. Costa. "Instance space analysis for 2D bin packing mathematical models". In: *Eur. J. Oper. Res.* 315.2 (2024), pp. 484–498. DOI: 10.1016/J.EJOR.2023.12.008 (cit. on p. 42).

[301] Fei Liu, Xi Lin, Zhenkun Wang, Shunyu Yao, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. *Large Language Model for Multi-objective Evolutionary Optimization*. arXiv. 2024. DOI: 10.48550/arXiv.2310.12541 (cit. on pp. 235, 236, 241, 246, 329, 337, 339, 346).

[302] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. *Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model*. arXiv. 2024. DOI: 10.48550/arXiv.2401.02051 (cit. on pp. 235, 241, 246, 326, 329, 337, 339, 341–343, 346).

[303] Fei Liu, Xialiang Tong, Mingxuan Yuan, and Qingfu Zhang. *Algorithm Evolution Using Large Language Model*. arXiv. 2023. DOI: 10.48550/arXiv.2311.15249 (cit. on pp. 235, 241, 246, 329, 337, 339, 340, 346).

[304] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Zhe Zhao, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. *A Systematic Survey on Large Language Models for Algorithm Design*. arXiv. 2024. DOI: 10.48550/arXiv.2410.14716 (cit. on pp. 50, 246, 329).

[305] Fei Liu, Rui Zhang, Zhuoliang Xie, Rui Sun, Kai Li, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. *LLM4AD: A Platform for Algorithm Design with Large Language Model*. arXiv. 2024. DOI: 10.48550/arXiv.2412.17287 (cit. on pp. 235, 241, 246, 329, 336, 339, 341–344, 346).

[306] Kelvin Liu. *Using Instance Space Analysis to Study the Bin Packing Problem*. Master Thesis. 2020. URL: https://matilda.unimelb.edu.au/matilda/matildadata/bin_packing/bin_packing.pdf (cit. on p. 256).

[307] Ran Liu, Biao Yuan, and Zhibin Jiang. "A branch-and-price algorithm for the home-caregiver scheduling and routing problem with stochastic travel and service times". In: *FFlex. Serv. Manuf.* 31 (2019), pp. 989–1011. DOI: 10.1007/s10696-018-9328-8 (cit. on pp. 48, 49, 202).

[308] Ran Liu, Biao Yuan, and Zhibin Jiang. "Mathematical model and exact algorithm for the home care worker scheduling and routing problem with lunch break requirements". In: *Int. J. Prod. Res.* 55.2 (2017), pp. 558–575. DOI: 10.1080/00207543.2016.1213917 (cit. on pp. 48, 49).

[309] Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. *Large Language Models as Evolutionary Optimizers*. arXiv. 2024. DOI: 10.48550/arXiv.2310.19046 (cit. on pp. 235, 241, 329, 337, 339).

[310] Wenheng Liu, Mahjoub Dridi, Hongying Fei, and Amir Hajjam El Hassani. "Hybrid metaheuristics for solving a home health care routing and scheduling problem with time windows, synchronized visits and lunch breaks". In: *Expert Syst. Appl.* 183 (2021), p. 115307. DOI: 10.1016/J.ESWA.2021.115307 (cit. on pp. 48, 49).

[311] Yang Liu, Fanyou Wu, Zhiyuan Liu, Kai Wang, Feiyue Wang, and Xiaobo Qu. "Can language models be used for real-world urban-delivery route optimization?" In: *The Innovation* 4.6 (2023), p. 100520. ISSN: 2666-6758. DOI: 10.1016/j.xinn.2023.100520 (cit. on pp. 234, 239, 246, 329, 335, 346).

[312] AI at Meta Llama Team. *The Llama 3 Herd of Models*. arXiv. 2024. DOI: 10.48550/arXiv.2407.21783 (cit. on pp. 30, 237, 238, 344).

[313] Mariana A. Londe, Luciana S. Pessoa, Carlos Eduardo de Andrade, and Mauricio G. C. Resende. "Biased random-key genetic algorithms: A review". In: *Eur. J. Oper. Res.* 321.1 (2025), pp. 1–22. DOI: 10.1016/J.EJOR.2024.03.030. URL: https://doi.org/10.1016/j.ejor.2024.03.030 (cit. on p. 13).

[314] Sifan Long, Jingjing Tan, Bomin Mao, Fengxiao Tang, Yangfan Li, Ming Zhao, and Nei Kato. "A Survey on Intelligent Network Operations and Performance Optimization Based on Large Language Models". In: *IEEE Communications Surveys & Tutorials* (2025), pp. 1–1. DOI: 10.1109/COMST.2025.3526606 (cit. on pp. 246, 329, 346).

[315] Manuel López-Ibáñez, Jürgen Branke, and Luís Paquete. "Reproducibility in Evolutionary Computation". In: *ACM Trans. Evol. Learn. Optim.* 1.4 (2021), 14:1–14:21. DOI: 10.1145/3466624 (cit. on p. 59).

[316] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58. DOI: 10.1016/j.orp.2016.09.002 (cit. on pp. 14, 41, 67, 82, 141, 210, 296, 311).

[317] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. "Iterated local search". In: *Handbook of metaheuristics*. Springer, 2003, pp. 320–353 (cit. on p. 294).

[318] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, and et al. *StarCoder 2 and The Stack v2: The Next Generation*. arXiv. 2024. DOI: 10.48550/arXiv.2402.19173 (cit. on p. 344).

[319] Scott M. Lundberg, Gabriel G. Erion, Hugh Chen, Alex J. DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. "From local explanations to global understanding with explainable AI for trees". In: *Nat. Mach. Intell.* 2.1 (2020), pp. 56–67. DOI: 10.1038/S42256-019-0138-9 (cit. on p. 159).

[320] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. 2017, pp. 4765–4774. URL: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html (cit. on p. 158).

[321] Zihan Luo, Xiran Song, Hong Huang, Jianxun Lian, Chenhao Zhang, Jinqi Jiang, and Xing Xie. *GraphInstruct: Empowering Large Language Models with Graph Understanding and Reasoning Capability*. arXiv. 2024. URL: https://arxiv.org/abs/2403.04483 (cit. on pp. 243, 345).

[322] Dennis Luxen and Christian Vetter. "Real-time routing with OpenStreetMap data". In: *19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACM-GIS 2011, November 1-4, 2011, Chicago, IL, USA, Proceedings*. Ed. by Isabel F. Cruz, Divyakant Agrawal, Christian S. Jensen, Eyal Ofek, and Egemen Tanin. ACM, 2011, pp. 513–516. DOI: 10.1145/2093973.2094062 (cit. on p. 204).

[323] Xiaomeng Ma, Yaping Fu, Kaizhou Gao, Ali Sadollah, and Kai Wang. "Integration routing and scheduling for multiple home health care centers using a multi-objective cooperation evolutionary algorithm with stochastic simulation". In: *Swarm Evol. Comput.* 75 (2022), p. 101175. DOI: 10.1016/J.SWEVO.2022.101175 (cit. on pp. 48, 49).

[324] Paula Maddigan and Teo Susnjak. "Chat2VIS: Generating Data Visualizations via Natural Language Using ChatGPT, Codex and GPT-3 Large Language Models". In: *IEEE Access* 11 (2023), pp. 45181–45193. DOI: 10.1109/ACCESS.2023.3274199 (cit. on p. 243).

[325] Maryam Karimi Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. "Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art". In: *Eur. J. Oper. Res.* 296.2 (2022), pp. 393–422. DOI: 10.1016/J.EJOR.2021.04.032 (cit. on pp. 11, 42).

[326] Matteo Manica, Jannis Born, Joris Cadow, Dimitrios Christofidellis, Ashish Dave, Dean Clarke, Yves Gaetan Nana Teukam, Giorgio Giannone, Samuel C. Hoffman, Matthew Buchan, Vijil Chenthamarakshan, Timothy Donovan, Hsiang Han Hsu, Federico Zipoli, Oliver Schilter, Akihiro Kishimoto, Lisa Hamada, Inkit Padhi, Karl Wehden, Lauren McHugh, Alexy Khrabrov, Payel Das, Seiji Takeda, and John R. Smith. "Accelerating material design with the generative toolkit for scientific discovery". In: *npj Computational Materials* 9.1 (2023), p. 69. ISSN: 2057-3960. DOI: 10.1038/s41524-023-01028-1 (cit. on p. 337).

[327] Dorota Slawa Mankowska, Frank Meisel, and Christian Bierwirth. "The home health care routing and scheduling problem with interdependent services". In: *Health Care Management Science* 17.1 (2014), pp. 15–30. DOI: 10.1007/s10729-013-9243-1 (cit. on pp. 47–49, 180, 182, 185, 197, 202, 207, 210, 218).

[328] James Manyika and Sissie Hsiao. *An overview of Bard: an early experiment with generative AI*. AI. Google Static Documents. Accessed: 2024-06-27. 2023. URL: https://ai.google/static/documents/google-about-bard.pdf (cit. on p. 31).

[329] Jinzhu Mao, Dongyun Zou, Li Sheng, Siyi Liu, Chen Gao, Yue Wang, and Yong Li. *Identify Critical Nodes in Complex Network with Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2403.03962 (cit. on pp. 235, 241, 246, 329, 337, 339, 346).

[330] Samuel Marks and Max Tegmark. *The Geometry of Truth: Emergent Linear Structure in Large Language Model Representations of True/False Datasets*. arXiv. 2024. URL: https://arxiv.org/abs/2310.06824 (cit. on p. 31).

[331] Marie-Eléonore Marmion, Aymeric Blot, Laetitia Jourdan, and Clarisse Dhaenens. "Neutrality in the Graph Coloring Problem". In: *Learning and Intelligent Optimization*. Ed. by Giuseppe Nicosia and Panos Pardalos. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 125–130. DOI: 10.1007/978-3-642-44973-4_14 (cit. on p. 87).

[332] Silvano Martello and Paolo Toth. "Lower bounds and reduction procedures for the bin packing problem". In: *Discret. Appl. Math.* 28.1 (1990), pp. 59–70. DOI: 10.1016/0166-218X(90)90094-S (cit. on pp. 45, 114, 115).

[333] Rafael Martí, Marc Sevaux, and Kenneth Sörensen. "Fifty years of metaheuristics". In: *Eur. J. Oper. Res.* 321.2 (2025), pp. 345–362. DOI: 10.1016/J.EJOR.2024.04.004 (cit. on p. 13).

[334] Raúl Martín-Santamaría, Manuel López-Ibáñez, Thomas Stützle, and J. Manuel Colmenar. "On the automatic generation of metaheuristic algorithms for combinatorial optimization problems". In: *Eur. J. Oper. Res.* 318.3 (2024), pp. 740–751. DOI: 10.1016/J.EJOR.2024.06.001 (cit. on p. 41).

[335] Alicja Martinek, Szymon Łukasik, and Amir H. Gandomi. "Large Language Models as Tuning Agents of Metaheuristics". In: *32nd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2024)*. Bruges, Belgium: i6doc.com, 2024, pp. 631–636. DOI: 10.14428/ESANN/2024.ES2024-209 (cit. on pp. 235, 242, 246, 248, 330, 336, 342, 346).

[336] Maria di Mascolo, Cléa Martinez, and Marie-Laure Espinouse. "Routing and scheduling in Home Health Care: A literature survey and bibliometric analysis". In: *Comput. Ind. Eng.* 158 (2021), p. 107255. DOI: 10.1016/J.CIE.2021.107255 (cit. on p. 46).

[337] Malek Masmoudi, Bassem Jarboui, and Rahma Borchani. "Efficient metaheuristics for the home (health)-care routing and schedulingproblem with time windows and synchronized visits". In: *Optim. Lett.* 17.9 (2023), pp. 2135–2167. DOI: 10.1007/S11590-023-02006-8 (cit. on p. 47).

[338] Andrea Matarazzo and Riccardo Torlone. *A Survey on Large Language Models with some Insights on their Capabilities and Limitations*. arXiv. 2025. URL: https://doi.org/10.48550/arXiv.2501.04040 (cit. on p. 251).

[339] Chandra Sen Mazumdar, Muthu Mathirajan, Ravindran Gopinath, and Appa Iyer Sivakumar. "Tabu Search methods for scheduling a burn-in oven with non-identical job sizes and secondary resource constraints". In: *International Journal of Operational Research* 3 (2008), pp. 119–139. DOI: 10.1504/IJOR.2008.016157 (cit. on p. 43).

[340] Tommaso Mannelli Mazzoli, Lucas Kletzander, Pascal Van Hentenryck, and Nysret Musliu. "Investigating Large Neighbourhood Search for Bus Driver Scheduling". In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*. Ed. by Sara Bernardini and Christian Muise. AAAI Press, 2024, pp. 360–368. DOI: 10.1609/ICAPS.V34I1.31495 (cit. on p. 40).

[341] Jordan Meadows and Andre Freitas. *A Survey in Mathematical Language Processing*. arXiv. 2024. DOI: 10.48550/arXiv.2205.15231 (cit. on pp. 5, 223).

[342] Sharif Melouk, Purushothaman Damodaran, and Ping-Yu Chang. "Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing". In: *Int. J. Prod. Econ.* 87.2 (2004), pp. 141–147. DOI: 10.1016/S0925-5273(03)00092-6 (cit. on p. 43).

[343] Kostis Michailidis, Dimos Tsouros, and Tias Guns. "Constraint Modelling with LLMs Using In-Context Learning". In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Vol. 307. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 20:1–20:27. DOI: 10.4230/LIPIcs.CP.2024.20 (cit. on pp. 235, 239, 243, 244, 330, 335, 345, 346).

[344] Minizinc. *The MiniZinc Examples Archive - v1.0.0*. Accessed on 26 Oct 2025. 2017. URL: https://github.com/MiniZinc/minizinc-examples/tree/master (cit. on p. 256).

[345] Florian Mischek and Nysret Musliu. "Leveraging problem-independent hyper-heuristics for real-world test laboratory scheduling". In: *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*. Ed. by Sara Silva and Luís Paquete. ACM, 2023, pp. 321–329. DOI: 10.1145/3583131.3590354 (cit. on pp. 27, 29, 42).

[346] Florian Mischek and Nysret Musliu. "Reinforcement Learning for Cross-Domain Hyper-Heuristics". In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*. Ed. by Luc De Raedt. ijcai.org, 2022, pp. 4793–4799. DOI: 10.24963/IJCAI.2022/664 (cit. on p. 80).

[347] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe. "An Intelligent Hyper-Heuristic Framework for CHeSC 2011". In: *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*. Ed. by Youssef Hamadi and Marc Schoenauer. Vol. 7219. Lecture Notes in Computer Science. Springer, 2012, pp. 461–466. DOI: 10.1007/978-3-642-34413-8_45 (cit. on pp. 28, 79).

[348] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. "Convergence and finite-time behavior of simulated annealing". In: *1985 24th IEEE Conference on Decision and Control*. 1985, pp. 761–767. DOI: 10.1109/CDC.1985.268600 (cit. on pp. 26, 77).

[349] Kangtong Mo, Wenyan Liu, Fangzhou Shen, Xuanzhen Xu, Letian Xu, Xiran Su, and Ye Zhang. "Precision Kinematic Path Optimization for High-DoF Robotic Manipulators Utilizing Advanced Natural Language Processing Models". In: *2024 5th International Conference on Electronic Communication and Artificial Intelligence (ICECAI)*. Shenzhen, China: IEEE, 2024, pp. 649–654. DOI: 10.1109/ICECAI62591.2024.10675146 (cit. on pp. 235, 240, 241, 246, 330, 336, 340, 346).

[350] Khaled Mohammed, Margaret B Nolan, Tamim Rajjo, Nilay D Shah, Larry J Prokop, Prathibha Varkey, and Mohammad H Murad. "Creating a patient-centered health care delivery system: a systematic review of health care quality from the patient perspective". In: *American journal of medical quality* 31.1 (2016), pp. 12–21. DOI: 10.1177/1062860614545124 (cit. on p. 217).

[351] David Moher, Deborah J Cook, Susan Eastwood, Ingram Olkinm, Drummond Rennie, and Donna F Stroup. "Improving The Quality of Reports of Meta-analyses of Randomised Controlled Trials: The QUOROM Statement". In: *The Lancet* 354.9193 (1999), pp. 1896–1900. ISSN: 0140-6736. DOI: 10.1016/S0140-6736(99)04149-5 (cit. on p. 228).

[352] David Moher, Alessandro Liberati, Jennifer Tetzlaff, Douglas G. Altman, and The PRISMA Group. "Preferred Reporting Items for Systematic Reviews and Meta-Analyses: The PRISMA Statement". In: *PLOS Medicine* 6.7 (2009), pp. 1–6. DOI: 10.1371/journal.pmed.1000097 (cit. on p. 228).

[353] Roberto Montemanni. "Solving a Home Healthcare Routing and Scheduling Problem with Real-World Features". In: *Proceedings of the 9th International Conference on Machine Learning and Soft Computing*. Springer, to appear, 2025. DOI: 10.1007/978-981-96-6403-0_10 (cit. on p. 191).

[354] Roberto Montemanni, Sara Ceschia, and Andrea Schaerf. "A compact model for the home healthcare routing and scheduling problem". In: *EURO J. Comput. Optim.* 13 (2025), p. 100101. DOI: 10.1016/J.EJCO.2024.100101 (cit. on pp. 47, 180, 191, 207, 208, 210).

[355] Roberto Montemanni and Derek H. Smith. *On Solving the Maximum Flow Problem with Conflict Constraints*. arXiv. 2025. DOI: 10.48550/arXiv.2503.19685 (cit. on p. 3).

[356] Maximilian Moser, Nysret Musliu, Andrea Schaerf, and Felix Winter. "Exact and metaheuristic approaches for unrelated parallel machine scheduling". In: *J. Sched.* 25.5 (2022), pp. 507–534. DOI: 10.1007/S10951-021-00714-6. URL: https://doi.org/10.1007/s10951-021-00714-6 (cit. on p. 14).

[357] Mahdi Mostajabdaveh, Timothy T. Yu, Samarendra Chandan Bindu Dash, Rindranirina Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. *Evaluating LLM Reasoning in the Operations Research Domain with ORQA*. arXiv. 2024. DOI: 10.48550/arXiv.2412.17874 (cit. on pp. 244, 330, 345).

[358] Mahdi Mostajabdaveh, Timothy T. Yu, Rindranirina Ramamonjison, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. "Optimization modeling and verification from problem specifications using a multi-agent multi-stage LLM framework". In: *INFOR: Information Systems and Operational Research* 62.4 (2024), pp. 599–617. DOI: 10.1080/03155986.2024.2381306 (cit. on pp. 235, 236, 240–242, 244, 330, 335, 342, 345).

[359] Mario Andrés Muñoz, neelofarhassan, and Jeffrey Christiansen. *InstanceSpace: February 2023*. Version v0.3.3. 2023. DOI: 10.5281/zenodo.7700434 (cit. on p. 158).

[360] Yves Gaetan Nana Teukam, Federico Zipoli, Teodoro Laino, Emanuele Criscuolo, Francesca Grisoni, and Matteo Manica. "Integrating Genetic Algorithms and Language Models for Enhanced Enzyme Design". In: *Briefings in Bioinformatics* 26.1 (2024), bbae675. DOI: 10.1093/bib/bbae675 (cit. on pp. 235, 239, 330, 337, 346).

[361] Lise-Marie Nassen, Heidi Vandebosch, Karolien Poels, and Kathrin Karsay. "Opt-out, abstain, unplug. A systematic review of the voluntary digital disconnection literature". In: *Telematics and Informatics* 81 (2023), p. 101980. ISSN: 0736-5853. DOI: https://doi.org/10.1016/j.tele.2023.101980 (cit. on p. 228).

[362] Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. "MiniZinc: Towards a Standard CP Modelling Language". In: *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*. Ed. by Christian Bessiere. Vol. 4741. Lecture Notes in Computer Science. Springer, 2007, pp. 529–543. DOI: 10.1007/978-3-540-74970-7_38 (cit. on pp. 13, 44).

[363] Alberto Francisco Kummer Neto, Olinto C. B. de Araújo, Luciana S. Buriol, and Mauricio G. C. Resende. "A biased random-key genetic algorithm for the home health care problem". In: *Int. Trans. Oper. Res.* 31.3 (2024), pp. 1859–1889. DOI: 10.1111/ITOR.13221 (cit. on pp. 47, 180).

[364] Alberto Francisco Kummer Neto, Luciana S. Buriol, and Olinto César Bassi de Araújo. "A biased random key genetic algorithm applied to the VRPTW with skill requirements and synchronization constraints". In: *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*. Ed. by Carlos Artemio Coello Coello. ACM, 2020, pp. 717–724. DOI: 10.1145/3377930.3390209 (cit. on pp. 47, 180).

[365] Chong Man Ngoo, Say Leng Goh, San-Nah Sze, Nasser R. Sabar, Salwani Abdullah, and Graham Kendall. "A Survey of the Nurse Rostering Solution Methodologies: The State-of-the-Art and Emerging Trends". In: *IEEE Access* 10 (2022), pp. 56504–56524. DOI: 10.1109/ACCESS.2022.3177280 (cit. on p. 59).

[366] Yuting Ning, Jiayu Liu, Longhu Qin, Tong Xiao, Shangzi Xue, Zhenya Huang, Qi Liu, Enhong Chen, and Jinze Wu. *A Novel Approach for Auto-Formulation of Optimization Problems*. arXiv. 2023. DOI: 10.48550/arXiv.2302.04643 (cit. on pp. 235, 243, 330, 335, 345).

[367] Kazuma Obata, Tatsuya Aoki, Takato Horii, Tadahiro Taniguchi, and Takayuki Nagai. "LiP-LLM: Integrating Linear Programming and Dependency Graph With Large Language Models for Multi-Robot Task Planning". In: *IEEE Robotics and Automation Letters* 10.2 (2025), pp. 1122–1129. DOI: 10.1109/LRA.2024.3518105 (cit. on pp. 235, 239, 246, 330, 335, 346).

[368] Object Management Group. *Business Process Model and Notation (BPMN), Version 2.0*. Retrieved from the Object Management Group website. Object Management Group, 2011. URL: https://www.omg.org/spec/BPMN/2.0/PDF (cit. on p. 229).

[369] Gabriela Ochoa, Matthew R. Hyde, Timothy Curtois, José Antonio Vázquez Rodríguez, James D. Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J. Parkes, Sanja Petrovic, and Edmund K. Burke. "HyFlex: A Benchmark Framework for Cross-Domain Heuristic Search". In: *Evolutionary Computation in Combinatorial Optimization - 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*. Ed. by Jin-Kao Hao and Martin Middendorf.

Vol. 7245. Lecture Notes in Computer Science. Springer, 2012, pp. 136–147. DOI: 10.1007/978-3-642-29124-1_12 (cit. on p. 28).

[370] Gabriela Ochoa, Katherine M. Malan, and Christian Blum. "Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics". In: *Appl. Soft Comput.* 109.C (2021). ISSN: 1568-4946. DOI: 10.1016/j.asoc.2021.107492 (cit. on pp. 4, 37, 40, 60, 72, 167, 170, 171, 237).

[371] Gabriela Ochoa, Sébastien Verel, Fabio Daolio, and Marco Tomassini. "Local Optima Networks: A New Model of Combinatorial Fitness Landscapes". In: ed. by Hendrik Richter and Andries Engelbrecht. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 233–262. DOI: 10.1007/978-3-642-41888-4_9 (cit. on p. 40).

[372] Nastaran Oladzad-Abbasabady, Reza Tavakkoli-Moghaddam, Mehrdad Mohammadi, and Behdin Vahedi Nouri. "A bi-objective home care routing and scheduling problem considering patient preference and soft temporal dependency constraints". In: *Eng. Appl. Artif. Intell.* 119 (2023), p. 105829. DOI: 10.1016/J.ENGAPPAI.2023.105829 (cit. on pp. 47, 49).

[373] Büsra Olgun, Çagri Koç, and Fulya Altiparmak. "A hyper heuristic for the green vehicle routing problem with simultaneous pickup and delivery". In: *Comput. Ind. Eng.* 153 (2021), p. 107010. DOI: 10.1016/J.CIE.2020.107010 (cit. on p. 27).

[374] OpenAI. *GPT-4 Technical Report*. arXiv. 2024. DOI: 10.48550/arXiv.2303.08774 (cit. on pp. 30, 237, 238, 340).

[375] James B. Orlin. "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Math. Program.* 77 (1997), pp. 109–129. DOI: 10.1007/BF02614365 (cit. on p. 127).

[376] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. "Training language models to follow instructions with human feedback". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 27730–27744. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf (cit. on p. 30).

[377] Matthew J Page, David Moher, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and Joanne E McKenzie. "PRISMA 2020 Explanation and Elaboration: Updated Guidance And Exemplars For Reporting Systematic Reviews". In: *BMJ* 372 (2021). DOI: 10.1136/bmj.n160 (cit. on pp. 228, 321).

[378] Matthew J. Page, Joanne E. McKenzie, Patrick M. Bossuyt, Isabelle Boutron, Tammy C. Hoffmann, Cynthia D. Mulrow, Larissa Shamseer, Jennifer M. Tetzlaff, Elie A. Akl, Sue E. Brennan, Roger Chou, Julie Glanville, Jeremy M. Grimshaw, Asbjørn Hróbjartsson, Manoj M. Lalu, Tianjing Li, Elizabeth W. Loder, Evan Mayo-Wilson, Steve McDonald, Luke A. McGuinness, Lesley A. Stewart, James Thomas, Andrea C. Tricco, Vivian A. Welch, Penny Whiting, and David Moher. "The PRISMA 2020 Statement: An Updated Guideline For Reporting Systematic Reviews". In: *BMJ* 372 (2021). DOI: 10.1136/bmj.n71 (cit. on pp. 228, 249, 321).

[379] Federico Pagnozzi and Thomas Stützle. "Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems". In: *Eur. J. Oper. Res.* 276.2 (2019), pp. 409–421. DOI: 10.1016/J.EJOR.2019.01.018 (cit. on pp. 3, 139, 286, 287).

[380] Vishal Pallagani, Bharath Chandra Muppasani, Kaushik Roy, Francesco Fabiano, Andrea Loreggia, Keerthiram Murugesan, Biplav Srivastava, Francesca Rossi, Lior Horesh, and Amit Sheth. "On the prospects of incorporating large language models (LLMs) in automated planning and scheduling (APS)". In: *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling*. ICAPS '24. Alberta, Canada: AAAI Press, 2025. ISBN: 1-57735-889-9. DOI: 10.1609/icaps.v34i1.31503 (cit. on pp. 246, 330, 346).

[381] Quan-Ke Pan and Rubén Ruiz. "Local search methods for the flowshop scheduling problem with flowtime minimization". In: *Eur. J. Oper. Res.* 222.1 (2012), pp. 31–43. DOI: 10.1016/J.EJOR.2012.04.034 (cit. on p. 3).

[382] Rui Pan, Shuo Xing, Shizhe Diao, Wenhe Sun, Xiang Liu, KaShun Shum, Jipeng Zhang, Renjie Pi, and Tong Zhang. "Plum: Prompt Learning using Metaheuristics". In: *Findings of the ACL: ACL 2024*. Bangkok, Thailand: ACL, 2024, pp. 2177–2197. DOI: 10.18653/v1/2024.findings-acl.129 (cit. on p. 248).

[383] José Antonio Parejo, Antonio Ruiz Cortés, Sebastián Lozano, and Pablo Fernandez. "Metaheuristic optimization frameworks: a survey and benchmarking". In: *Soft Comput.* 16.3 (2012), pp. 527–561. DOI: 10.1007/S00500-011-0754-8 (cit. on pp. 13, 285, 287).

[384] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12 (2011), pp. 2825–2830. DOI: 10.5555/1953048.2078195 (cit. on p. 159).

[385] Paola Pellegrini, Franco Mascia, Thomas Stützle, and Mauro Birattari. "On the sensitivity of reactive tabu search to its meta-parameters". In: *Soft Comput.* 18.11 (2014), pp. 2177–2190. DOI: 10.1007/S00500-013-1192-6 (cit. on p. 40).

[386] Alessio Pellegrino, Özgür Akgün, Nguyen Dang, Zeynep Kiziltan, and Ian Miguel. "Transformer-Based Feature Learning for Algorithm Selection in Combinatorial Optimisation". In: *31st International Conference on Principles and Practice of Constraint Programming, CP 2025, August 10-15, 2025, Glasgow, Scotland*. Ed. by

Maria Garcia de la Banda. Vol. 340. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 31:1–31:22. DOI: 10.4230/LIPICS.CP.2025.31 (cit. on p. 42).

[387] Laurent Perron and Frédéric Didier. *Google OR-Tools — CP-SAT*. Version 9.12. 2025 (cit. on pp. 195, 211).

[388] Laurent Perron and Vincent Furnon. *OR-Tools*. Version v9.10. Google, May 7, 2024. URL: https://developers.google.com/optimization/ (cit. on p. 13).

[389] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the ACL: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: ACL, 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202 (cit. on p. 30).

[390] Felipe Pezoa, Juan L. Reutter, Fernando Suárez, Martín Ugarte, and Domagoj Vrgoc. "Foundations of JSON Schema". In: *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*. Ed. by Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao. ACM, 2016, pp. 263–273. DOI: 10.1145/2872427.2883029 (cit. on p. 198).

[391] Pierre Schaus, Renaud De Landtsheer. *OscaR User-guide*. 2016 (cit. on p. 287).

[392] Michal Pluhacek, Anezka Kazikova, Tomas Kadavy, Adam Viktorin, and Roman Senkerik. "Leveraging Large Language Models for the Generation of Novel Metaheuristic Optimization Algorithms". In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. GECCO '23 Companion. Lisbon, Portugal: ACM, 2023, pp. 1812–1820. ISBN: 9798400701207. DOI: 10.1145/3583133.3596401 (cit. on pp. 233, 248).

[393] Michal Pluhacek, Anezka Kazikova, Adam Viktorin, Tomas Kadavy, and Roman Senkerik. "Investigating the Potential of AI-Driven Innovations for Enhancing Differential Evolution in Optimization Tasks". In: *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. Honolulu, Oahu, HI, USA: IEEE, 2023, pp. 1070–1075. DOI: 10.1109/SMC53992.2023.10394233 (cit. on p. 248).

[394] Michal Pluhacek, Jozef Kovac, Adam Viktorin, Peter Janku, Tomas Kadavy, and Roman Senkerik. "Using LLM for Automatic Evolvement of Metaheuristics from Swarm Algorithm SOMA". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO '24 Companion. Melbourne, VIC, Australia: ACM, 2024, pp. 2018–2022. ISBN: 9798400704956. DOI: 10.1145/3638530.3664181 (cit. on p. 340).

[395] Wolfgang Pree. "Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design". In: *Object-Oriented Programming, Proceedings of the 8th European Conference, ECOOP '94, Bologna, Italy, July 4-8, 1994*. Ed. by Mario Tokoro and Remo Pareschi. Vol. 821. Lecture Notes in Computer Science. Springer, 1994, pp. 150–162. DOI: 10.1007/BFB0052181 (cit. on p. 289).

[396] Yasha Pushak and Holger H. Hoos. "Algorithm Configuration Landscapes: - More Benign Than Expected?" In: *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*. Ed. by Anne Auger, Carlos M. Fonseca, Nuno Lourenço, Penousal Machado, Luís Paquete, and L. Darrell Whitley. Vol. 11102. Lecture Notes in Computer Science. Springer, 2018, pp. 271–283. DOI: 10.1007/978-3-319-99259-4_22 (cit. on p. 41).

[397] Michell F. Queiroz, Flavien Lucas, and Kenneth Sörensen. "Instance generation tool for on-demand transportation problems". In: *Eur. J. Oper. Res.* 317.3 (2024), pp. 696–717. DOI: 10.1016/J.EJOR.2024.03.006 (cit. on pp. 46, 197).

[398] Alibaba Group Qwen Team. *Qwen Technical Report*. arXiv. 2023. DOI: 10.48550/arXiv.2309.16609 (cit. on pp. 237, 342).

[399] Alibaba Group Qwen Team. *Qwen2 Technical Report*. arXiv. 2024. DOI: 10.48550/arXiv.2407.10671 (cit. on pp. 238, 344).

[400] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv. 2024. DOI: 10.48550/arXiv.2305.18290) (cit. on p. 242).

[401] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu, et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv. 2023. DOI: 10.48550/arXiv.1910.10683 (cit. on pp. 238, 339).

[402] Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-dehkordi, Zirui Zhou, and Yong Zhang. "Augmenting Operations Research with Auto-Formulation of Optimization Models From Problem Descriptions". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Abu Dhabi, UAE: ACL, 2022, pp. 29–62. DOI: 10.18653/v1/2022.emnlp-industry.4 (cit. on pp. 235, 240, 243, 247, 330, 335, 339, 345).

[403] Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. *NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions*. arXiv. 2023. DOI: 10.48550/arXiv.2303.08233 (cit. on pp. 50, 236, 330).

[404] Aurora Ramírez, Rafael Barbudo, and José Raúl Romero. "An experimental comparison of metaheuristic frameworks for multi-objective optimization". In: *Expert Syst. J. Knowl. Eng.* 40.4 (2023). DOI: 10.1111/EXSY.12672 (cit. on p. 287).

[405] Matias Sevel Rasmussen, Tor Justesen, Anders Dohn, and Jesper Larsen. "The Home Care Crew Scheduling Problem: Preference-based visit clustering and temporal dependencies". In: *Eur. J. Oper. Res.* 219.3 (2012), pp. 598–610. DOI: 10.1016/J.EJOR.2011.10.048 (cit. on p. 46).

[406] Colin R. Reeves. "A genetic algorithm for flowshop sequencing". In: *Comput. Oper. Res.* 22.1 (1995), pp. 5–13. DOI: 10.1016/0305-0548(93)E0014-K (cit. on p. 65).

[407] Florian Régin, Elisabetta De Maria, and Alexandre Bonlarron. "Combining Constraint Programming Reasoning with Large Language Model Predictions". In: *30th International Conference on Principles and Practice of Constraint Programming (CP 2024)*. Vol. 307. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 25:1–25:18. DOI: 10.4230/LIPIcs.CP.2024.25 (cit. on pp. 235, 239, 330, 336, 346).

[408] Gerhard Reinelt. "TSPLIB — A Traveling Salesman Problem Library". In: *INFORMS J. Comput.* 3.4 (1991), pp. 376–384. DOI: 10.1287/IJOC.3.4.376 (cit. on p. 84).

[409] Wesley F. Reinhart and Antonia Statt. "Large language models design sequence-defined macromolecules via evolutionary optimization". In: *NPJ Computational Materials* 10.1 (2024), p. 262. DOI: 10.1038/s41524-024-01449-6 (cit. on pp. 235, 241, 242, 330, 336, 343, 346).

[410] Andrea Rendl, Matthias Prandtstetter, Gerhard Hiermann, Jakob Puchinger, and Günther R. Raidl. "Hybrid Heuristics for Multimodal Homecare Scheduling". In: *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012. Proceedings*. Ed. by Nicolas Beldiceanu, Narendra Jussien, and Eric Pinson. Vol. 7298. Lecture Notes in Computer Science. Springer, 2012, pp. 339–355. DOI: 10.1007/978-3-642-29828-8_22 (cit. on p. 46).

[411] John R. Rice. "The Algorithm Selection Problem". In: *Adv. Comput.* 15 (1976), pp. 65–118. DOI: 10.1016/S0065-2458(08)60520-3 (cit. on pp. 35, 36, 42).

[412] Paolo Rinaldin. *JEasyLocal: Refactoring, riprogettazione e sviluppo di un framework orientato agli oggetti per la ricerca locale*. Master Thesis. Università degli Studi di Udine. 2003 (cit. on p. 288).

[413] Erick Rodríguez-Esparza, Antonio D. Masegosa, Diego Oliva, and Enrique Onieva. "A new Hyper-heuristic based on Adaptive Simulated Annealing and Reinforcement Learning for the Capacitated Electric Vehicle Routing Problem". In: *Expert Syst. Appl.* 252 (2024), p. 124197. DOI: 10.1016/J.ESWA.2024.124197 (cit. on p. 42).

[414] Oleksandr Romanko, Akhilesh Narayan, and Roy H. Kwon. "ChatGPT-Based Investment Portfolio Selection". In: *SN Operations Research Forum* 4.4 (2023), pp. 1–27. DOI: 10.1007/s43069-023-00277-6 (cit. on pp. 235, 237, 240, 331, 337, 340, 346).

[415] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. "Mathematical discoveries from program search with large language models". In: *Nature* 625.7995 (2024), pp. 468–475. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06924-6 (cit. on pp. 235, 241, 246, 326, 329, 331, 333, 337, 341, 346).

[416] Roberto Maria Rosati, Matteo Petris, Luca Di Gaspero, and Andrea Schaerf. "Multi-neighborhood simulated annealing for the sports timetabling competition ITC2021". In: *J. Sched.* 25.3 (2022), pp. 301–319. DOI: 10.1007/S10951-022-00740-Y (cit. on p. 17).

[417] Roberto Maria Rosati and Andrea Schaerf. "Multi-Neighborhood Simulated Annealing for the Capacitated Dispersion Problem". In: *Expert Syst. Appl.* 255 (2024), p. 124484. DOI: 10.1016/J.ESWA.2024.124484 (cit. on p. 17).

[418] Jamie Ross, Fiona Stevenson, Rosa Lau, and Elizabeth Murray. "Factors that influence the implementation of e-health: a systematic review of systematic reviews (an update)". In: *Implementation Science* 11.1 (2016), p. 146. ISSN: 1748-5908. DOI: 10.1186/s13012-016-0510-7 (cit. on p. 228).

[419] Francesca Rossi, Peter van Beek, and Toby Walsh, eds. *Handbook of Constraint Programming*. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. ISBN: 978-0-444-52726-4. URL: https://www.sciencedirect.com/science/bookseries/15746526/2 (cit. on p. 13).

[420] Tiasa Singha Roy, Aditeya Baral, Ayush Rajesh Jhaveri, and Yusuf Baig. *Can LLMs understand Math? - Exploring the Pitfalls in Mathematical Reasoning*. arXiv. 2025. URL: https://doi.org/10.48550/arXiv.2505.15623 (cit. on p. 262).

[421] Abdullahi Saka, Ridwan Taiwo, Nurudeen Saka, Babatunde Abiodun Salami, Saheed Ajayi, Kabiru Akande, and Hadi Kazemi. "GPT models in construction industry: Opportunities, limitations, and a use case validation". In: *Developments in the Built Environment* 17 (2024), p. 100300. ISSN: 2666-1659. DOI: https://doi.org/10.1016/j.dibe.2023.100300 (cit. on pp. 235, 246, 331, 336, 346).

[422] Alberto Santini, Stefan Ropke, and Lars Magnus Hvattum. "A comparison of acceptance criteria for the adaptive large neighbourhood search metaheuristic". In: *J. Heuristics* 24.5 (2018), pp. 783–815. DOI: 10.1007/S10732-018-9377-X (cit. on p. 40).

[423] Andrea Schaerf, Marco Cadoli, and Maurizio Lenzerini. "LOCAL++: A C++ framework for local search algorithms". In: *Softw. Pract. Exp.* 30.3 (2000), pp. 233–257. DOI: 10.1002/(SICI)1097-024X(200003)30:3\<233::AID-SPE297\>3.0.CO;2-K (cit. on p. 287).

[424] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation". In: *IEEE Transactions on Software Engineering* 50.1 (2024), pp. 85–105. DOI: 10.1109/TSE.2023.3334955 (cit. on p. 236).

[425] Eric O. Scott and Sean Luke. "ECJ at 20: toward a general metaheuristics toolkit". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. Ed. by Manuel López-Ibáñez, Anne Auger, and Thomas Stützle. ACM, 2019, pp. 1391–1398. DOI: 10.1145/3319619.3326865 (cit. on p. 287).

[426] Amin Shahmardan and Mohsen S. Sajadieh. "Truck scheduling in a multi-door cross-docking center with partial unloading - Reinforcement learning-based simulated annealing approaches". In: *Comput. Ind. Eng.* 139 (2020), p. 106134. DOI: 10.1016/J.CIE.2019.106134 (cit. on p. 42).

[427] Sina Shahnejat-Bushehri, Reza Tavakkoli-Moghaddam, Mehdi Boronoos, and Ahmad Ghasemkhani. "A robust home health care routing-scheduling problem with temporal dependencies under uncertainty". In: *Expert Syst. Appl.* 182 (2021), p. 115209. DOI: 10.1016/J.ESWA.2021.115209 (cit. on pp. 48, 49).

[428] Claude E. Shannon. "A mathematical theory of communication". In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. DOI: 10.1002/J.1538-7305.1948.TB01338.X (cit. on p. 170).

[429] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2402.03300 (cit. on p. 343).

[430] Hadar Shavit and Holger H. Hoos. "Revisiting SATZilla Features in 2024". In: *27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India*. Ed. by Supratik Chakraborty and Jie-Hong Roland Jiang. Vol. 305. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024, 27:1–27:26. DOI: 10.4230/LIPICS.SAT.2024.27 (cit. on p. 42).

[431] Paul Shaw. "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems". In: *Principles and Practice of Constraint Programming - CP98, 4th International Conference, Pisa, Italy, October 26-30, 1998, Proceedings*. Ed. by Michael J. Maher and Jean-Francois Puget. Vol. 1520. Lecture Notes in Computer Science. Springer, 1998, pp. 417–431. DOI: 10.1007/3-540-49481-2_30 (cit. on pp. 13, 27).

[432] Maria Amélia Lopes Silva, Sérgio Ricardo de Souza, Marcone Jamilson Freitas Souza, and Moacir Felizardo de França Filho. "Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis". In: *Appl. Soft Comput.* 71 (2018), pp. 433–459. DOI: 10.1016/J.ASOC.2018.06.050 (cit. on pp. 13, 287).

[433] Aditi Singla, Aditya Singh, and Kanishk Kukreja. *A bi-objective ϵ-constrained framework for quality-cost optimization in language model ensembles*. arXiv. 2023. DOI: 10.48550/arXiv.2312.16119 (cit. on p. 248).

[434] Kate Smith-Miles and Simon Bowly. "Generating new test instances by evolving in instance space". In: *Comput. Oper. Res.* 63 (2015), pp. 102–113. DOI: 10.1016/J.COR.2015.04.022 (cit. on pp. 254–256).

[435] Kate Smith-Miles, Jeffrey Christiansen, and Mario Andrés Muñoz. "Revisiting *where are the hard knapsack problems?* via Instance Space Analysis". In: *Comput. Oper. Res.* 128 (2021), p. 105184. DOI: 10.1016/J.COR.2020.105184 (cit. on p. 256).

[436] Kate Smith-Miles and Mario Andrés Muñoz. "Instance Space Analysis for Algorithm Testing: Methodology and Software Tools". In: *ACM Comput. Surv.* 55.12 (2023), 255:1–255:31. DOI: 10.1145/3572895 (cit. on pp. 4, 35, 36, 40, 42, 100, 153, 154, 156, 201).

[437] Michael Soprano, Kevin Roitero, David La Barbera, Davide Ceolin, Damiano Spina, Gianluca Demartini, and Stefano Mizzaro. "Cognitive Biases in Fact-Checking and Their Countermeasures: A Review". In: *Information Processing & Management* 61.3 (2024), p. 103672. ISSN: 0306-4573. DOI: 10.1016/j.ipm.2024.103672 (cit. on p. 228).

[438] Kenneth Sörensen. "Metaheuristics - the metaphor exposed". In: *Int. Trans. Oper. Res.* 22.1 (2015), pp. 3–18. DOI: 10.1111/ITOR.12001 (cit. on p. 4).

[439] Kenneth Sörensen, Marc Sevaux, and Fred W. Glover. "A History of Metaheuristics". In: *Handbook of Heuristics*. Ed. by Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende. Springer, 2018, pp. 791–808. DOI: 10.1007/978-3-319-07124-4_4 (cit. on p. 3).

[440] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. *Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models*. arXiv. 2022. DOI: 10.48550/arXiv.2206.04615 (cit. on p. 245).

[441] Biplav Srivastava and Vishal Pallagani. *The Case for Developing a Foundation Model for Planning-like Tasks from Scratch*. 2024. DOI: 10.48550/arXiv.2404.04540 (cit. on pp. 247, 331).

[442] Niki van Stein and Thomas Bäck. "LLaMEA: A Large Language Model Evolutionary Algorithm for Automatically Generating Metaheuristics". In: *IEEE Transactions on Evolutionary Computation* (2024), pp. 1–1. DOI: 10.1109/TEVC.2024.3497793 (cit. on p. 331).

[443] Niki van Stein, Anna V. Kononova, Lars Kotthoff, and Thomas Bäck. "Code Evolution Graphs: Understanding Large Language Model Driven Design of Algorithms". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. New York, NY, USA: Association for Computing Machinery, 2025, pp. 943–951. DOI: 10.1145/3712256.3726328 (cit. on p. 249).

[444] Niki van Stein, Diederick Vermetten, and Thomas Bäck. *In-the-loop Hyper-Parameter Optimization for LLM-Based Automated Design of Heuristics*. arXiv. 2024. DOI: 10.48550/arXiv.2410.16309 (cit. on pp. 235, 246, 331, 336, 340, 346).

[445] Niki van Stein, Haoran Yin, Anna V. Kononova, Thomas Bäck, and Gabriela Ochoa. *Behaviour Space Analysis of LLM-driven Meta-heuristic Discovery*. arXiv. 2025. DOI: 10.48550/arXiv.2507.03605. URL: https://arxiv.org/abs/2507.03605 (cit. on p. 249).

[446] Simon Strassl and Nysret Musliu. "Instance space analysis and algorithm selection for the job shop scheduling problem". In: *Comput. Oper. Res.* 141 (2022), p. 105661. DOI: 10.1016/J.COR.2021.105661 (cit. on pp. 40, 65, 155, 157, 256).

[447] Jingyan Sui, Shizhe Ding, Xulin Huang, Yue Yu, Ruizhi Liu, Boyang Xia, Zhenxin Ding, Liming Xu, Haicang Zhang, Chungong Yu, and Dongbo Bu. "A survey on deep learning-based algorithms for the traveling salesman problem". In: *Frontiers of Computer Science* 19.6 (2024), p. 196322. DOI: 10.1007/s11704-024-40490-y (cit. on pp. 246, 331, 346).

[448] Yiwen Sun, Furong Ye, Xianyin Zhang, Shiyu Huang, Bingzhen Zhang, Ke Wei, and Shaowei Cai. *AutoSAT: Automatically Optimize SAT Solvers via Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2402.10705 (cit. on pp. 235, 236, 241, 331, 336, 341).

[449] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. arXiv. 2022. DOI: 10.48550/arXiv.2210.09261 (cit. on pp. 245, 246).

[450] Jerry Swan, Steven Adriænsen, Alexander E. I. Brownlee, Kevin Hammond, Colin G. Johnson, Ahmed Kheiri, Faustyna Krawiec, Juan Julián Merelo, Leandro L. Minku, Ender Özcan, Gisele L. Pappa, Pablo García-Sánchez, Kenneth Sörensen, Stefan Voß, Markus Wagner, and David R. White. "Metaheuristics "In the Large"". In: *Eur. J. Oper. Res.* 297.2 (2022), pp. 393–406. DOI: 10.1016/J.EJOR.2021.05.042 (cit. on pp. 6, 13, 40, 249, 286, 295, 296).

[451] Jerry Swan, Steven Adriaensen, Mohamed Bishr, Edmund K Burke, John A Clark, Patrick De Causmaecker, Juanjo Durillo, Kevin Hammond, Emma Hart, Colin G Johnson, et al. "A Research Agenda for Metaheuristic Standardization". In: *Proceedings of the XI Metaheuristics International Conference (MIC)*. Agadir, Morocco, 2015, pp. 1–3 (cit. on pp. 6, 13, 249).

[452] E. Taillard. "Benchmarks for basic scheduling problems". In: *European Journal of Operational Research* 64.2 (1993), pp. 278–285. DOI: 10.1016/0377-2217(93)90182-M (cit. on pp. 65, 84).

[453] Éric D. Taillard. "A Short List of Combinatorial Optimization Problems". In: *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem*. Cham: Springer International Publishing, 2023, pp. 31–61 (cit. on p. 14).

[454] Éric D. Taillard. "Robust taboo search for the quadratic assignment problem". In: *Parallel Comput.* 17.4-5 (1991), pp. 443–455. DOI: 10.1016/S0167-8191(05)80147-4 (cit. on p. 62).

[455] Tanya Y. Tang and J. Christopher Beck. "CP and Hybrid Models for Two-Stage Batching and Scheduling". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings*. Ed. by Emmanuel Hebrard and Nysret Musliu. Vol. 12296. Lecture Notes in Computer Science. Springer, 2020, pp. 431–446. DOI: 10.1007/978-3-030-58942-4_28 (cit. on p. 43).

[456] Yixuan Tang and Yi Yang. *Pooling And Attention: What Are Effective Designs For LLM-Based Embedding Models?* 2024. arXiv: 2409.02727 [cs.CL]. URL: https://arxiv.org/abs/2409.02727 (cit. on p. 258).

[457] Gemini Team. *Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context*. arXiv. 2024. DOI: 10.48550/arXiv.2403.05530 (cit. on p. 31).

[458] Gemini Team. *Gemini: A Family of Highly Capable Multimodal Models*. arXiv. 2024. DOI: 10.48550/arXiv.2312.11805 (cit. on p. 342).

[459] Gemma Team. *Gemma 2: Improving Open Language Models at a Practical Size*. arXiv. 2024. DOI: 10.48550/arXiv.2408.00118 (cit. on p. 344).

[460] Meta AI Team. *Code Llama: Open Foundation Models for Code*. arXiv. 2024. DOI: 10.48550/arXiv.2308.12950 (cit. on pp. 341, 342).

[461] Yi Team. *Yi: Open Foundation Models by 01.AI*. arXiv. 2024. DOI: 10.48550/arXiv.2403.04652 (cit. on pp. 238, 344).

[462] Rodrigo F. Toso and Mauricio G. C. Resende. "A C++application programming interface for biased random-key genetic algorithms". In: *Optim. Methods Softw.* 30.1 (2015), pp. 81–93. DOI: 10.1080/10556788.2014.890197 (cit. on p. 291).

[463] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. *LLaMA: Open and Efficient Foundation Language Models*. arXiv. 2023. DOI: 10.48550/arXiv.2302.13971 (cit. on p. 30).

[464] Thanh V. T. Tran and Truong Son Hy. "Protein Design by Directed Evolution Guided by Large Language Models". In: *IEEE Transactions on Evolutionary Computation* (2024), pp. 1–1. DOI: 10.1109/TEVC.2024.3439690 (cit. on pp. 235, 331, 336, 346).

[465] German Treimun-Costa, Elizabeth Montero, Gabriela Ochoa, and Nicolás Rojas-Morales. "Modelling parameter configuration spaces with local optima networks". In: *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*. Ed. by Carlos Artemio Coello Coello. ACM, 2020, pp. 751–759. DOI: 10.1145/3377930.3390199 (cit. on p. 41).

[466] Renan Spencer Trindade, Olinto C. B. de Araujo, and Marcia Fampa. "Arc-Flow Approach for Parallel Batch Processing Machine Scheduling with Non-identical Job Sizes". In: *Combinatorial Optimization - 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May 4-6, 2020, Revised Selected Papers*. Ed. by Mourad Baiou, Bernard Gendron, Oktay Gunluk, and Ali Ridha Mahjoub. Vol. 12176. Lecture Notes in Computer Science. Springer, 2020, pp. 179–190. DOI: 10.1007/978-3-030-53262-8_15 (cit. on p. 43).

[467] Dimosthenis C. Tsouros, Hélène Verhaeghe, Serdar Kadioglu, and Tias Guns. *Holy Grail 2.0: From Natural Language to Constraint Models*. arXiv. 2023. DOI: 10.48550/ARXIV.2308.01589 (cit. on pp. 11, 223, 234, 235, 247, 331, 335).

[468] Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. *Zephyr: Direct Distillation of LM Alignment*. arXiv. 2023. DOI: 10.48550/arXiv.2310.16944 (cit. on p. 342).

[469] Jose Tupayachi, Haowen Xu, Olufemi A. Omitaomu, Mustafa Can Camur, Aliza Sharmin, and Xueping Li. "Towards Next-Generation Urban Decision Support Systems through AI-Powered Construction of Scientific Ontology Using Large Language Models—A Case in Optimizing Intermodal Freight Transportation". In: *Smart Cities* 7.5 (2024), pp. 2392–2421. DOI: 10.3390/smartcities7050094 (cit. on pp. 235, 240, 246, 331, 335, 340, 346).

[470] Renata Turkes, Kenneth Sörensen, and Lars Magnus Hvattum. "Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search". In: *Eur. J. Oper. Res.* 292.2 (2021), pp. 423–442. DOI: 10.1016/J.EJOR.2020.10.045 (cit. on pp. 4, 27, 40, 59).

[471] Tommaso Urli. *Hybrid meta-heuristics for combinatorial optimization*. Ph.D. Thesis. Università degli Studi di Udine. 2014 (cit. on p. 288).

[472] Vyacheslav Ustyugov. "On Different Methods For Automated MILP Solver Configuration". In: *2024 20th International Asian School-Seminar on Optimization Problems of Complex Systems (OPCS)*. Issyk-Kul Lake, Kyrgyzstan: IEEE, 2024, pp. 24–27. DOI: 10.1109/OPCS63516.2024.10720437 (cit. on pp. 235, 239, 247, 331, 336).

[473] Rob J. M. Vaessens, Emile H. L. Aarts, and Jan Karel Lenstra. "A local search template". In: *Comput. Oper. Res.* 25.11 (1998), pp. 969–979. DOI: 10.1016/S0305-0548(97)00093-2 (cit. on p. 286).

[474] Mauricio Varas, Felipe Baesler, Franco Basso, Juan Pablo Contreras, Raúl Pezoa, María Francisca Rojas-Goldsack, and Ricardo Ronco. "A home hospitalization assignment and routing problem with multiple time windows, mandatory returns and perishable biological samples: A Chilean case study". In: *Comput. Ind. Eng.* 189 (2024), p. 109951. DOI: 10.1016/J.CIE.2024.109951 (cit. on pp. 48, 49, 202, 219).

[475] Johannes Vass, Marie-Louise Lackner, Christoph Mrkvicka, Nysret Musliu, and Felix Winter. "Exact and meta-heuristic approaches for the production leveling problem". In: *J. Sched.* 25.3 (2022), pp. 339–370. DOI: 10.1007/S10951-022-00721-1 (cit. on p. 59).

[476] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All You Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Long Beach, CA, USA: Curran Associates, Inc., 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 29, 232).

[477] Kristian Verduin, Thomas Weise, and Daan van den Berg. "Why is the traveling tournament problem not solved with genetic algorithms?" In: *Evo* 2023 – Late-Breaking Abstracts Volume*. Brno, Czech Republic: Species, 2023, pp. 13–18. URL: https://arxiv.org/pdf/2403.13950 (cit. on p. 235).

[478] Bruno Vieira, Derya Demirtas, Jeroen B. van de Kamer, Erwin W. Hans, Willem Jongste, and Wim van Harten. "Radiotherapy treatment scheduling: Implementing operations research into clinical practice". In: *PLOS ONE* 16 (2021), pp. 1–13. DOI: 10.1371/journal.pone.0247428 (cit. on p. 179).

[479] Florentina Voboril, Vaidyanathan Peruvemba Ramaswamy, and Stefan Szeider. *Generating Streamlining Constraints with Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2408.10268 (cit. on pp. 235, 332, 337, 340, 343).

[480] Stefan Voss, Silvano Martello, Ibrahim Osman, and Catherine Roucairol. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Springer New York, NY, 1999. DOI: 10.1007/978-1-4615-5775-3 (cit. on p. 3).

[481] Ivan Vulic, Edoardo Maria Ponti, Robert Litschko, Goran Glavas, and Anna Korhonen. "Probing Pretrained Language Models for Lexical Semantics". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu. Association for Computational Linguistics, 2020, pp. 7222–7240. DOI: 10.18653/V1/2020.EMNLP-MAIN.586 (cit. on p. 32).

[482] Matthew B. Wall. *A Genetic Algorithm for Resource-Constrained Scheduling*. Ph.D. Thesis. MIT Mechanical Engineering Department. 1996 (cit. on p. 286).

[483] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. "Do NLP Models Know Numbers? Probing Numeracy in Embeddings". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Association for Computational Linguistics, 2019, pp. 5306–5314. DOI: 10.18653/V1/D19-1534 (cit. on p. 32).

[484] Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. *Can Language Models Solve Graph Problems in Natural Language?* arXiv. 2024. DOI: 10.48550/arXiv.2305.10037 (cit. on pp. 243, 345).

[485] Hui-Mei Wang and Fuh-Der Chou. "Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics". In: *Expert Syst. Appl.* 37.2 (2010), pp. 1510–1521. DOI: 10.1016/J.ESWA.2009.06.070 (cit. on p. 43).

[486] Kangxu Wang, Ze Chen, and Jiewen Zheng. *OPD@NL4Opt: An ensemble approach for the NER task of the optimization problem*. arXiv. 2023. DOI: 10.48550/arXiv.2301.02459 (cit. on pp. 235, 243, 332, 335, 345).

[487] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. "A Survey on Large Language Model Based Autonomous Agents". In: *Frontiers of Computer Science* 18.6 (2024). ISSN: 2095-2236. DOI: 10.1007/s11704-024-40231-1 (cit. on p. 249).

[488] Qi Wang, Nan Huang, Zhuo Chen, Xiaowen Chen, Hanying Cai, and Yunpeng Wu. "Environmental data and facts in the semiconductor manufacturing industry: An unexpected high water and energy consumption situation". In: *Water Cycle* 4 (2023), pp. 47–54. DOI: https://doi.org/10.1016/j.watcyc.2023.01.004 (cit. on p. 97).

[489] Yang Wang and Kai Li. *Large Language Models and Operations Research: A Structured Survey*. arXiv. 2025. DOI: 10.48550/arXiv.2509.18180 (cit. on p. 50).

[490] Yuan Wang, Lokesh Kumar Sambasivan, Mingang Fu, and Prakhar Mehrotra. *Pivoting Retail Supply Chain with Deep Generative Techniques: Taxonomy, Survey and Insights*. arXiv. 2024. DOI: 10.48550/arXiv.2403.00861 (cit. on p. 233).

[491] Yuhui Wang, Junaid Farooq, Hakim Ghazzai, and Gianluca Setti. *Multi-UAV Placement for Integrated Access and Backhauling Using LLM-Driven Optimization*. TechRxive. 2024. DOI: 10.36227/techrxiv.172833400.07230719/v1 (cit. on pp. 235, 246, 332, 336, 340, 341, 346).

[492] Michael L. Waskom. "seaborn: statistical data visualization". In: *J. Open Source Softw.* 6.60 (2021), p. 3021. DOI: 10.21105/JOSS.03021 (cit. on p. 159).

[493] Segev Wasserkrug, Leonard Boussioux, Dick den Hertog, Farzaneh Mirzazadeh, Ilker Birbil, Jannis Kurtz, and Donato Maragno. *From Large Language Models and Optimization to Decision Optimization CoPilot: A Research Manifesto*. arXiv. 2024. DOI: 10.48550/arXiv.2402.16269 (cit. on pp. 234, 247, 332).

[494] Jean-Paul Watson, J. Christopher Beck, Adele E. Howe, and L. Darrell Whitley. "Problem difficulty for tabu search in job-shop scheduling". In: *Artif. Intell.* 143.2 (2003), pp. 189–217. DOI: 10.1016/S0004-3702(02)00363-6 (cit. on p. 60).

[495] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. *Emergent Abilities of Large Language Models*. arXiv. 2022. DOI: 10.48550/arXiv.2206.07682 (cit. on p. 30).

[496] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Richter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 35. Virtual Conference: Curran Associates, Inc., 2022, pp. 24824–24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf (cit. on p. 30).

[497] Felix Winter, Nysret Musliu, Emir Demirovic, and Christoph Mrkvicka. "Solution Approaches for an Automotive Paint Shop Scheduling Problem". In: *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019*. Ed. by J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava. AAAI Press, 2019, pp. 573–581. URL: https://ojs.aaai.org/index.php/ICAPS/article/view/3524 (cit. on p. 14).

[498] David H. Wolpert and William G. Macready. "No free lunch theorems for optimization". In: *IEEE Trans. on Evo. Comp.* 1.1 (1997), pp. 67–82. DOI: 10.1109/4235.585893 (cit. on pp. 14, 42).

[499] Chao Wu, Yiling Ge, Xinyan Zhang, Yanling Du, Shizhe He, Zhaohua Ji, and Hongjuan Lang. "The combined effects of Lamaze breathing training and nursing intervention on the delivery in primipara: A PRISMA systematic review meta-analysis". In: *Medicine* 100.4 (2021). DOI: 10.1097/MD.0000000000023920 (cit. on p. 228).

[500] Xingyu Wu, Sheng-hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. *Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap*. arXiv. 2024. DOI: 10.48550/arXiv.2401.10034 (cit. on pp. 5, 50, 223, 246, 332).

[501] Xingyu Wu, Yan Zhong, Jibin Wu, Bingbing Jiang, and Kay Chen Tan. "Large Language Model-Enhanced Algorithm Selection: Towards Comprehensive Algorithm Representation". In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. IJCAI '24. Jeju Island, South Korea: International Joint Conferences on Artificial Intelligence, 2024. DOI: 10.24963/ijcai.2024/579 (cit. on pp. 235, 242, 248, 332, 336, 339).

[502] Xuan Wu, Di Wang, Lijie Wen, Yubin Xiao, Chunguo Wu, Yuesong Wu, Chaoyu Yu, Douglas L. Maskell, and You Zhou. *Neural Combinatorial Optimization Algorithms for Solving Vehicle Routing Problems: A Comprehensive Survey with Perspectives*. arXiv. 2024. DOI: 10.48550/arXiv.2406.00415 (cit. on pp. 246, 332, 346).

[503] Ting Xiang, Yanfeng Li, and Wai Yuen Szeto. "The daily routing and scheduling problem of home health care: based on costs and participants' preference satisfaction". In: *Int. Trans. Oper. Res.* 30.1 (2023), pp. 39–69. DOI: 10.1111/ITOR.13043 (cit. on pp. 47, 49).

[504] Xin Xiao, Bin Ji, Samson S. Yu, and Guohua Wu. "A tabu-based adaptive large neighborhood search for scheduling unrelated parallel batch processing machines with non-identical job sizes and dynamic job arrivals". In: *Flex. Serv. Manuf.* 36.2 (2024), pp. 409–452. DOI: 10.1007/s10696-023-09488-9 (cit. on p. 43).

[505] Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. "Chain-of-Experts: When LLMs Meet Complex Operations Research Problems". In: *The Twelfth International Conference on Learning Representations (ICLR 2024)*. Virtual Conference: OpenReview, 2024. URL: https://openreview.net/forum?id=HobyL1B9CZ (cit. on pp. 235, 236, 239, 241–243, 332, 337, 339, 345).

[506] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *J. Artif. Intell. Res.* 32 (2008), pp. 565–606. DOI: 10.1613/JAIR.2490 (cit. on p. 42).

[507] Niteesh Yadav and Ajinkya N. Tanksale. "An integrated routing and scheduling problem for home healthcare delivery with limited person-to-person contact". In: *Eur. J. Oper. Res.* 303.3 (2022), pp. 1100–1125. DOI: 10.1016/J.EJOR.2022.03.022 (cit. on pp. 48, 49).

[508] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. *Large Language Models as Optimizers*. arXiv. 2024. DOI: 10.48550/arXiv.2309.03409 (cit. on pp. 235, 240, 245, 246, 332, 336, 339–341, 346).

[509] Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. *OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling*. 2024. DOI: 10.48550/arXiv.2407.09887 (cit. on pp. 235, 239, 244, 332, 335, 339, 340, 343–345).

[510] Shunyu Yao, Fei Liu, Xi Lin, Zhichao Lu, Zhenkun Wang, and Qingfu Zhang. *Multi-objective Evolution of Heuristic Using Large Language Model*. arXiv. 2024. DOI: 10.48550/arXiv.2409.16867 (cit. on pp. 235, 241, 246, 332, 336, 339, 346).

[511] Wang Yatong, Pei Yuchen, and Zhao Yuqi. *TS-EoH: An Edge Server Task Scheduling Algorithm Based on Evolution of Heuristic*. arXiv. 2024. DOI: 10.48550/arXiv.2409.09063 (cit. on pp. 235, 241, 246, 332, 336, 339, 343, 344, 346).

[512] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. "ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution". In: *Proceedings of the Thirty-Eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*. Vancouver, Canada: Neural Information Processing Systems Foundation, 2024, pp. 1–32. URL: https://openreview.net/forum?id=483IPG0HWL (cit. on pp. 235, 241, 246, 332, 333, 337, 339, 340, 344, 346).

[513] Hengxu You, Yang Ye, Tianyu Zhou, Qi Zhu, and Jing Du. "Robot-Enabled Construction Assembly with Automated Sequence Planning Based on ChatGPT: RoboGPT". In: *Buildings* 13.7 (2023). ISSN: 2075-5309. DOI: 10.3390/buildings13071772 (cit. on pp. 235, 241, 246, 333, 336, 340, 346).

[514] He Yu and Jing Liu. *AutoRNet: Automatically Optimizing Heuristics for Robust Network Design via Large Language Models*. arXiv. 2024. DOI: 10.48550/arXiv.2410.17656 (cit. on pp. 235, 241, 246, 333, 336, 340, 346).

[515] He Yu and Jing Liu. *Deep Insights into Automated Optimization with Large Language Models and Evolutionary Algorithms*. arXiv. 2024. DOI: 10.48550/arXiv.2410.20848 (cit. on pp. 51, 247, 333).

[516] Vincent F. Yu, Hadi Susanto, Panca Jodiawan, Tsai-Wei Ho, Shih-Wei Lin, and Yu-Tsung Huang. "A Simulated Annealing Algorithm for the Vehicle Routing Problem With Parcel Lockers". In: *IEEE Access* 10 (2022), pp. 20764–20782. DOI: 10.1109/ACCESS.2022.3152062 (cit. on p. 24).

[517] Eugenia Zanazzo, Sara Ceschia, Agostino Dovier, and Andrea Schaerf. "Solving the medical student scheduling problem using simulated annealing". In: *J. Sched.* 28.2 (2025), pp. 233–246. DOI: 10.1007/S10951-024-00806-Z (cit. on p. 285).

[518] Eugenia Zanazzo, Sara Ceschia, and Andrea Schaerf. "Solving the Integrated Patient-to-Room and Nurse-to-Patient Assignment by Simulated Annealing". In: *Metaheuristics - 15th International Conference, MIC 2024, Lorient, France, June 4-7, 2024, Proceedings, Part I*. Ed. by Marc Sevaux, Alexandru-Liviu Olteanu, Eduardo G. Pardo, Angelo Sifaleras, and Salma Makboul. Vol. 14753. Lecture Notes in Computer Science. Springer, 2024, pp. 158–163. DOI: 10.1007/978-3-031-62912-9_15 (cit. on p. 14).

[519] Jihai Zhang, Wei Wang, Siyan Guo, Li Wang, Fangquan Lin, Cheng Yang, and Wotao Yin. "Solving General Natural-Language-Description Optimization Problems with Large Language Models". In: *Proceedings of the 2024 Conference of the North American Chapter of the ACL: Human Language Technologies (Volume 6: Industry Track)*. Mexico City, Mexico: ACL, 2024, pp. 483–490. DOI: 10.18653/v1/2024.naacl-industry.42 (cit. on pp. 235, 236, 240–244, 333, 335, 339, 340, 342, 345).

[520] Rui Zhang, Fei Liu, Xi Lin, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. "Understanding the Importance of Evolutionary Search in Automated Heuristic Design with Large Language Models". In: *Parallel Problem Solving from Nature – PPSN XVIII*. Cham: Springer Nature Switzerland, 2024, pp. 185–202. DOI: 10.1007/978-3-031-70068-2_12 (cit. on pp. 235, 241, 246, 333, 336, 339–341, 343, 344, 346).

[521] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. *BERTScore: Evaluating Text Generation with BERT*. arXiv. 2020. DOI: 10.48550/arXiv.1904.09675 (cit. on p. 340).

[522] Ting Zhang, Yang Liu, Xintong Yang, Jingjing Chen, and Jiaming Huang. "Home health care routing and scheduling in densely populated communities considering complex human behaviours". In: *Comput. Ind. Eng.* 182 (2023), p. 109332. DOI: 10.1016/J.CIE.2023.109332 (cit. on pp. 48, 49).

[523] Zeyang Zhang, Xin Wang, Ziwei Zhang, Haoyang Li, Yijian Qin, and Wenwu Zhu. *LLM4DyG: Can Large Language Models Solve Spatial-Temporal Problems on Dynamic Graphs?* arXiv. 2024. DOI: 10.48550/arXiv.2310.17110 (cit. on pp. 243, 345).

[524] Zhigen Zhao, Shuo Cheng, Yan Ding, Ziyi Zhou, Shiqi Zhang, Danfei Xu, and Ye Zhao. "A Survey of Optimization-Based Task and Motion Planning: From Classical to Learning Approaches". In: *IEEE/ASME Transactions on Mechatronics* 29.5 (2024), pp. 1–27. DOI: 10.1109/TMECH.2024.3452509 (cit. on pp. 234, 246, 333, 335).

[525] ZiYan Zhao, Shixin Liu, MengChu Zhou, Xiwang Guo, and Liang Qi. "Decomposition Method for New Single-Machine Scheduling Problems From Steel Production Systems". In: *IEEE Trans Autom. Sci. Eng.* 17.3 (2020), pp. 1376–1387. DOI: 10.1109/TASE.2019.2953669 (cit. on p. 43).

[526] Nicolas Zufferey. "Metaheuristic: Some principles for an efficient design". In: *Computer Technology and Applications* 3 (2012), pp. 446–462 (cit. on p. 40).

Author's Declarations

Authorship Statement

The candidate declares that the research presented in this thesis is the result of original work carried out during the doctoral program. Portions of the work have been published or submitted for publication in peer-reviewed venues. Where applicable, each chapter explicitly indicates the corresponding publication and thus acknowledges the contributions of co-authors. The thesis as a whole represents a unified and extended synthesis of these contributions, integrated, and developed by the author to provide a coherent account of the research conducted.

Positionality Statement

The author of this thesis acknowledges that her personal and academic background, research experiences, and position within the scientific community inevitably shape her perspectives and interpretations. Her work is situated within the field of metaheuristics and combinatorial optimization, and her exposure to specific models and methodologies may influence her approach to problem formulation, algorithm selection, and evaluation.

Several of the studies presented in this thesis have been conducted in collaboration with researchers from diverse academic disciplines, primarily within computer science. These interdisciplinary exchanges have broadened the author's understanding and enriched her methodological approach.

While the author strives for rigor and impartiality, she recognizes that complete objectivity is inherently challenging. Her interpretations are inevitably influenced by her academic trajectory, theoretical perspectives, and methodological choices. By acknowledging these influences, the author aims to maintain transparency in her analyses and encourage critical engagement with her findings.

Usage of Artificial Intelligence

Regarding the use of Artificial Intelligence in this thesis, Chapter 18 integrates Large Language Models (LLMs) as part of the research methodology itself, where such tools are explicitly reported and analyzed as a subject of scientific inquiry.

Beyond this, LLMs were occasionally employed to assist in improving the readability and grammar of the written text. All intellectual content, analyses, and arguments remain the sole responsibility of the author.

Funding and Computational Resources

Part of this thesis derives from published works and research projects supported by various national and international funding bodies. In what follows, the author lists only those institutions, projects, and funding bodies that directly supported her work, omitting those that supported colleagues involved in related research activities.

- The majority of computational experiments were conducted using the LABIC research infrastructure, funded by the Regione Autonoma Friuli Venezia Giulia, the Italian Ministry of Foreign Affairs, and the Italian Ministry of University and Research.
- Several contributions were developed within *Strategic Plan of the University of Udine — Interdepartmental Project on Artificial Intelligence (2020–2025)* and within the *Strategic Plan of the University of Udine (2022–2025)*.
- The first research stay at Technische Universität Wien, and the related work on the Oven Scheduling Problem (OSP), was supported by a three-month allowance from the Society for the Promotion of Evolutionary Computation in Europe and its Surroundings (SPECIES).
- The work reported in Part IV was conducted within the project *Modeling and Solving Real-world Home Healthcare Routing and Scheduling Problems* (NextGenerationEU).
- Some of the work reported in Chapter 2 and Part IV was developed in the context of the First ROAR-NET Training School, which was held in Luxembourg in June 2025.