Algorithmica
https://doi.org/10.1007/s00453-022-00989-x

Solving String Problems on Graphs Using the Labeled Direct Product

Nicola Rizzo[1] · Alexandru I. Tomescu[1] · Alberto Policriti[2]

Received: 6 September 2021 / Accepted: 1 June 2022

Abstract
Suffix trees are an important data structure at the core of optimal solutions to many fundamental string problems, such as *exact pattern matching*, *longest common substring*, *matching statistics*, and *longest repeated substring*. Recent lines of research focused on extending some of these problems to vertex-labeled graphs, either by using efficient ad-hoc approaches which do not generalize to all input graphs, or by indexing difficult graphs and having worst-case exponential complexities. In the absence of an ubiquitous and polynomial tool like the suffix tree for labeled graphs, we introduce the labeled direct product of two graphs as a general tool for obtaining optimal algorithms in the worst case: we obtain conceptually simpler algorithms for the quadratic problems of string matching (SMLG) and longest common substring (LCSP) in labeled graphs. Our algorithms run in time linear in the size of the labeled product graph, which may be smaller than quadratic for some inputs, and their run-time is predictable, because the size of the labeled direct product graph can be precomputed efficiently. We also solve LCSP on graphs containing cycles, which was left as an open problem by Shimohira et al. in 2011. To show the power of the labeled product graph, we also apply it to solve the matching statistics (MSP) and the longest repeated string (LRSP) problems in labeled graphs. Moreover, we show that our (worst-case quadratic) algorithms are also optimal, conditioned on the Orthogonal Vectors Hypothesis. Finally, we complete the complexity picture around LRSP by studying it on undirected graphs.

Keywords Longest repeated substring · Longest common substring · String algorithm · Graph algorithm · Motif discovery · Fine-grained complexity

✉ Nicola Rizzo
nicola.rizzo@helsinki.fi

✉ Alexandru I. Tomescu
alexandru.tomescu@helsinki.fi

✉ Alberto Policriti
alberto.policriti@uniud.it

1 Department of Computer Science, University of Helsinki, Helsinki, Finland

2 Department of Mathematics, Computer Science and Physics, University of Udine, Udine, Italy

1 Introduction

Motivated by various application domains appearing during the last decades, a significant branch of string algorithm research has focused on extending string problems from texts to more complex objects, such as labeled rooted trees (e.g. modeling XML documents [1]) and labeled graphs (e.g. modeling pan-genome graphs [2, 3]). For example, the *string matching in labeled graphs* (SMLG) problem asks to find an *occurrence* of a given string S inside a labeled graph G, that is, a walk of G whose concatenation of vertex labels (*spelling*) is S. On rooted trees, SMLG can be solved in linear time [4], but on general graphs it admits both quadratic-time conditional lower bounds [5–8] and optimal algorithms of matching time complexity [9–11]. Specific graph classes like *de Bruijn graphs* and *Wheeler graphs* admit linear-time algorithms for SMLG [12, 13], and Burrows-Wheeler Transform (BWT) approaches have been recently generalized to generic graphs [14, 15].

Despite this active interest in the SMLG problem, the graph extensions of three other fundamental string problems have received none or little attention so far: *longest common substring*, *matching statistics*, *longest repeated substring* (Fig. 1). On strings, the former two problems can also be seen as relaxations of the exact string matching problem (for e.g. handling approximate matching) [16, 17], and all problems can be seen as basic instances of *pattern/motif discovery in strings* [18]. In this paper we consider their natural generalizations to Σ-*labeled graphs*, namely to tuples $G = (V, E, L)$, with V and E the sets of vertices and edges, respectively, and $L\colon V \to \Sigma$ assigning to each vertex a *label* from Σ (the original string problems can be obtained by taking all graphs to be labeled paths).

Problem 1 (*String matching problem* (SMLG)) Given a Σ-labeled graph G and a string S, find an occurrence of S in G.

Problem 2 (*Longest common string problem* (LCSP)) Given G_1, G_2 Σ-labeled graphs, find a longest string S occurring in G_1 and in G_2.

Problem 3 (*Matching statistics problem* (MSP)) Given G_1, G_2 Σ-labeled graphs, compute for every vertex v of G_1 the length $\mathrm{MS}(v)$ of a longest string S such that S occurs in G_1 starting at v and S occurs anywhere in G_2.

Problem 4 (*Longest repeated string problem* (LRSP)) Given a Σ-labeled graph G, find a longest string S having at least two distinct occurrences in G.

When defined on strings, all problems can be solved in linear time and space as basic textbook applications of the suffix tree [16, 19], or of the suffix array and the longest common prefix (LCP) array [20, 21], under the standard assumption to be working with an *integer alphabet*, i.e. containing integers from a range that is linear-sized with respect to the input. On labeled graphs, LCSP has been considered by Shimohira et al. [22]. They solved it in time $O(|E_1| \cdot |E_2|)$, where E_1 and E_2 are the edge sets of G_1 and G_2, respectively, and only if one of the graphs is acyclic, and they left the general case of two cyclic graphs as an open problem [22]. Moreover, there are no known analogous algorithms for MSP and LRSP, and no characterization of the

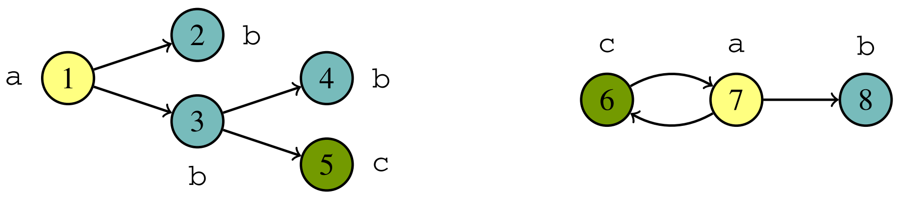

Fig. 1 An example of an {a, b, c}-labeled graph made up of two components: the longest common string between the two components is ab, and the longest repeated string of the whole graph is ab as well, since all longer strings spelled by some walk have exactly one occurrence. Taking the right component as G_1 and the left component as G_2, we have that $\mathrm{MS}(6) = 1$, $\mathrm{MS}(7) = 2$, and $\mathrm{MS}(8) = 1$

possible solutions for LRSP. Regarding hardness, note that the $O(|E_1| \cdot |E_2|)$-time algorithm of [22] for LCSP is optimal under the same conditional lower bounds as for SMLG [5–8], since the decision version of SMLG (i.e. whether there is an occurrence of the string) is linear-time reducible to LCSP. In fact, the same holds also for MSP. Nevertheless, there exists no analogous lower bound for LRSP. Note that the four problems defined by specifying occurrences as *directed paths*, i.e. visiting each vertex at most once, are NP-complete (Observation 1). In this paper we mostly concentrate on walk occurrences (Fig. 1).

Problem 5 (*Strict string problems on graphs*) We define strict-SMLG, strict-LCSP, strict-MSP, and strict-LRSP as the modifications of SMLG, LCSP, MSP, and LRSP, respectively, such that any occurrence of a string S in a graph is based on paths, that is, the walks spelling any string S cannot visit any vertex more than once.

SMLG, LCSP and LRSP have connections also to Automata Theory, since the spellings of all walks of a finite labeled graph form a regular language. Indeed, one can transform a labeled graph into an NFA by making every vertex a final state, adding a new initial state connected to all vertices, and moving the label of each vertex on each incoming edge: a string occurs in a labeled graph if it is accepted by its corresponding NFA; strings common to two labeled graphs correspond to strings accepted by the intersection of their corresponding NFAs; repeated strings of a labeled graph correspond to *ambiguous words* of the resulting NFA, namely words having at least two accepting computations. *Ambiguity* of automata (or its lack thereof) has been studied in the context of Descriptional Complexity Theory [23–26], not to be confused with Descriptive Complexity Theory. For example, the *degree of ambiguity* of an NFA is the maximum number of accepting computations of any word by the automaton. While there are works about studying upper bounds of such metric [27, 28], to the best of our knowledge there is no research on the *longest* ambiguous words of an NFA.

As a first result on labeled graphs, we observe that on labeled directed trees (i.e. rooted trees with all edges oriented away from the root) LRSP and LCSP can also be solved in linear time and space as an easy application of the tree counterparts of the suffix tree and the suffix array: the *suffix tree of a tree* [29] and the *XBW transform of a tree* (XBWT) [1]. The former, introduced by Kosaraju in 1989, generalizes the suffix tree to represent all suffixes of the strings spelled by the upwards paths of a given tree and admits linear-time construction algorithms [30, 31]. The latter, introduced by Ferragina et al. in 2005, is an invertible transform, also computable in linear time, encoding a tree as an ordered list of elements each corresponding to a vertex: the order of these elements first follows the lexicographical ordering of the unique path from the parent of the corresponding vertex to the root of the tree, then the pre-order visit of the tree. LRSP of a tree can be solved directly by either structure in the same way as LRSP of a string, while LCSP of two trees can be solved with a simple adaptation of either structure.

In this paper we introduce the *labeled direct product* of two labeled graphs G_1 and G_2, denoted $G_1 \otimes G_2$, inspired from e.g. the Cartesian product construction for the intersection language accepted by two finite state automata (Sect. 2). While not a completely novel idea, this product cleanly encodes each and every pair of walks of the input graphs spelling the same string and it appears as the right conceptual tool to *optimally* solve string problems on graphs. Our results are as follows.

1.1 Conceptually Simpler and More Efficient Algorithms

The current state-of-the-art algorithm for SMLG was introduced in 1997 by Amir et al. [9] in the context of hypertexts (i.e. directed graphs such that each vertex is labeled with a string). Given a Σ-labeled graph $G = (V, E, L)$ and a string $S = S[1] \cdots S[m] \in \Sigma^*$, the algorithm works by constructing a directed acyclic graph (DAG) G' having vertex v^i for each vertex $v \in V$ and for each position $i \in \{1, \dots, m\}$, such that there is an edge between two vertices v^i, w^{i+1} if $L(v) = S[i]$ and $(v, w) \in E$: SMLG is then solved by finding and reporting a path of length $|S| - 1$ in G'. Instead, by treating the pattern S as a labeled path G_S, we can solve SMLG by simply finding a path of length $|S| - 1$ in $G \otimes G_S$ (Sect. 2). Since G_S is a path, then $G \otimes G_S$ is a DAG, and thus such a path can be found in time linear in the size of $G \otimes G_S$. Our labeled product is a subgraph of the DAG G' by Amir et al.: G' considers mismatching vertices v^i such that $L(v) \neq S[i]$ and it avoids computing the edges *from* mismatching vertices but not the (remaining) edges *to* mismatching vertices. Thus, their algorithm always takes time $\Omega(|V| \cdot |S|)$, even when $G \otimes G_S$ has smaller size, and takes time $\Theta(|E| \cdot |S|)$ for some families of inputs where $G \otimes G_S$ has smaller size.

Moreover, LCSP on DAGs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$ is equivalent to finding a path of maximum length of the DAG $G_1 \otimes G_2$, which is also solvable in time and space linear in the size of $G_1 \otimes G_2$ (Sect. 2). Thus, our LCSP algorithm using $G_1 \otimes G_2$ is not only a conceptually simpler version of the $O(|E_1| \cdot |E_2|)$-time and $\Theta(|V_1| \cdot |V_2|)$-space dynamic programming algorithm of Shimohira et al. [22] for LCSP, but can also be faster and use less space, if $G_1 \otimes G_2$ has size $O(|V_1| \cdot |V_2|)$ or the alphabet Σ has constant size (Remark 3). Otherwise, our algorithm implies a greater space usage, since it stores $G_1 \otimes G_2$: choosing not to store the edges of $G_1 \otimes G_2$ and instead computing them when needed results in a time and space complexity closer to that of the existing algorithm for LCSP (Remark 5).

1.2 Simple Solution to an Open Problem

In addition to providing simple algorithms on DAGs, the labeled product graph also allows for conceptually simple and efficient solutions on arbitrary graphs. For example,

Table 1 Summary, for some variants of LCSP on two graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, of the time complexities. The linear-time algorithms assume an integer alphabet and the quadratic-time algorithms are optimal under OVH (Theorem 5)

$G_1 \backslash G_2$	path	tree	DAG	graph
path	$O(\|V_1\| + \|V_2\|)$ w/ suffix tree [16,32]	$O(\|V_1\| + \|V_2\|)$ w/ suffix tree [4], or w/ XBWT [1]	$O(\|E_1\| \cdot \|E_2\|)$ w/ dynamic programming algorithm [22], or w/ labeled direct product graph, Sect. 2	
tree	–	$O(\|V_1\| + \|V_2\|)$ w/ suffix tree of a tree [30,31], or w/ XBWT [1]		
DAG	–	–		
graph	–	–	–	$O(\|E_1\| \cdot \|E_2\|)$ w/ labeled direct product graph Sect. 3

LCSP on two graphs containing cycles was left open by Shimohira et al. [22], and in Sect. 4 we show that it is solvable by just checking whether $G_1 \otimes G_2$ has a cycle, and if not, still finding a path of maximum length in $G_1 \otimes G_2$ (see Table 1 for a summary of the complexity results for LCSP).

1.3 Solutions to New Problems

The labeled direct product also allows for solutions to related problems. For MSP on DAGs we analogously find paths of maximum length from some vertices of $G_1 \otimes G_2$ (Theorem 1). We generalize this algorithm on arbitrary graphs by computing the strongly connected components (SCCs) of $G_1 \otimes G_2$ and by checking a condition analogous to that of LCSP for every vertex v of G_1 (Theorem 2). These algorithms use time and space linear in the size of $G_1 \otimes G_2$.

LRSP on a DAG G is equivalent to finding paths of maximum length passing through specific vertices of $G \otimes G$ (Theorem 1). On arbitrary graphs, we use further interesting connections between purely graph-theoretic concepts of the labeled product graph (SCCs) and string-theoretic ones (non-deterministic vertices). The difference of LRSP with respect to LCSP and MSP is that the problem may admit repeated strings of infinite length or repeated strings of unbounded lengths—these two scenarios may not coincide. Even if the difference between these two concepts may seem artificial, their study is necessary for the natural characterization of LRSP solutions. Indeed, in Sect. 3 we show that such cases can be efficiently identified (infinite repeated strings can be identified in $G \otimes G$ by checking reachability from a certain set of vertices to a non-trivial SCC, while repeated strings of unbounded length can be identified by checking reachability from a non-trivial SCC to some non-deterministic vertex). If none of these cases happen, we show that the problem is solvable with the DAG algorithm for LRSP on an acyclic subgraph of $G \otimes G$ (Theorem 3). The entire procedures take time and space linear in the size of $G \otimes G$. In addition, we can also output a *linear-size* representation of a longest repeated string, infinite or not.

1.4 Optimality under Conditional Lower Bounds

In Sect. 4 we show that the above algorithms of worst-case quadratic-time complexity are also conditionally optimal. First, we note how the quadratic lower bounds of [6, 8] imply the same quadratic lower bounds for LCSP and MSP. Second, in Theorem 4 of Sect. 4, we show that on DAGs that are *deterministic* (i.e. the labels of the out-neighbors of every vertex are all distinct) the SMLG problem has a linear-time reduction to LRSP, which thus implies the same lower bounds for LRSP as in [6, 8] (holding also for deterministic DAGs). To the best of our knowledge such a reduction does not exist when the problems are defined on strings. Third, in Theorem 5 we show that, under the Orthogonal Vectors Hypothesis (OVH) [33], there can be no truly sub-quadratic algorithm solving LRSP, even when the graph is a DAG, with vertex labels from a binary alphabet, maximum in-degree and out-degree of any vertex at most 2, and is deterministic. Our reduction for LRSP is simpler than that of [6], but with an interesting difficulty arising from the fact that we must encode the orthogonal vectors input in the *same* graph, and must ensure that the occurrences of the longest repeated string are distinct.

Moreover, in the same way as the labeled direct product graph is a general tool for obtaining algorithms, the construction behind our reduction could also be a general approach to obtain conditional lower bounds for string problems on graphs. For example, our OVH construction (simpler than [6]) also provides a conditional lower bound for LCSP, and in Corollary 3 we show our reduction also proves the same conditional lower bound for a variant of MSP.

1.5 The Full Complexity Picture of LRSP

Finally, since on directed graphs LRSP turned out the most complex problem to solve, in Sect. 5 we complete its complexity picture by studying it also on undirected graphs, by similarly considering undirected paths, trees and graphs, and the path and walk variants of the problem (see Table 2). While these results are simpler than for directed graphs, they exhibit some interesting complexity dichotomies on analogous classes of graphs. For example, for walk occurrences the problem is *linear-time* solvable on general undirected graphs, as opposed to having a conditional quadratic-time lower bound on general *directed* graphs). Note that the SMLG problem has the same complexity on both directed and undirected graphs [6], making this dichotomy for LRSP more interesting. Moreover, when defined on paths, we obtain only a quadratic-time algorithm on undirected trees, even though LRSP is linear on *directed* trees. As such, we put forward as an interesting open problem either improving this complexity, or proving a lower bound.

Table 2 Summary, for all the variants of LRSP on a graph (V, E, L), of the time complexities. The linear-time algorithms assume an integer alphabet and the quadratic-time algorithms for directed graphs are optimal under OVH (Theorem 5). We leave as an open problem improving our solution to strict-LRSP on undirected trees, or proving it is conditionally tight

Graph Class	Graph Type			
	directed		undirected	
paths	$O(\lvert V\rvert)$ w/ suffix tree [16,32]		$O(\lvert V\rvert)$ w/ repeated strings of length 2 check Sect. 5	$O(\lvert V\rvert)$ w/ suffix tree Sect. 5
trees	$O(\lvert V\rvert)$ w/ suffix tree of a tree [30,31], or w/ XBW transform of a tree [1]			$O(\lvert V\rvert^2)$ w/ reduction to LRSP on directed trees Sect. 5
DAGs	$O(\lvert E\rvert^2)$ w/ labeled direct self-product graph Sect. 2.2		–	–
graphs	$O(\lvert E\rvert^2)$ w/ labeled direct self-product graph Sect. 3	NP-complete Observation 1	$O(\lvert E\rvert)$ w/ repeated strings of length 2 check Sect. 5	NP-complete Sect. 5
	LRSP	strict-LRSP	LRSP	strict-LRSP
	Occurrence definition			

1.6 Related Work on SMLG

Due to the success of string data structures based on the Burrows-Wheeler Transform (BWT) [34], research efforts have been spent for extending the BWT to index and compress labeled graphs—or specific classes of graphs like de Bruijn graphs [12]—while supporting navigation and matching queries [12–15, 35, 36]. Most notably:

- *Wheeler graphs* [36] and *languages* [35] are a specific class of labeled graph admitting a total (lexicographic) order of its nodes that enables string matching in time linear in the pattern's size; however, recognizing if a given labeled graph is Wheeler is (in general) NP-complete [37];
- recently Cotumaccio and Prezza in [14] generalized this approach to index NFAs—and, thus, labeled graphs—by showing that a *co-lexicographic partial order* of the states always exists, and it allows the matching of a string S in time $O(|S| \cdot p^2 \cdot \log(p \cdot |\Sigma|))$, where p is the partial order's *width*, that is, the size of its largest antichain ($p = 1$ for Wheeler graphs); p explains well the compressibility and non-determinism of the NFA, since the authors proved that the classic powerset NFA-to-DFA determinization algorithm generates a deterministic automaton with at most $2^p(n - p + 1) - 1$ states;
- also recently, Nellore et al. proposed the *nength* of a labeled graph [15] as an invertible transform based on the same NFA-to-DFA determinization algorithm; the result allows for the matching of a pattern in linear time, even though the size of the nength can be exponential in the number of nodes of the graph.

The above results do not violate the quadratic lower bounds of Equi et al. [6, 7] and they clearly should be preferred to the labeled direct product in the specific cases where they guarantee total subquadratic string matching. However, if the graph and the patterns are known, the sizes of the corresponding labeled direct products can be computed in linear time (Remark 5): the space and run time of our solution for SMLG is predictable, so the labeled direct product can be the initial step of a general framework that assesses the need—or the maximum time allowed—for more sophisticated methods.

1.7 Notation and Preliminaries

Given a non-empty and finite alphabet Σ, we denote with Σ^* and Σ^ω the set of all finite and infinite strings over Σ, respectively. For convenience, we also define $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$, with ε the empty string. We say that Σ is an *integer alphabet* if it contains integers from a range that is linear-sized with respect to the input of the problem at hand, allowing linear-time lexicographical sorting. Given the Σ-labeled graph $G = (V, E, L)$, a *walk* in G is any finite or infinite sequence of vertices $p = (p_0, p_1, p_2, \dots)$, such that there is an edge from any p_i to its successor in p. If all vertices of p are pairwise distinct, then p is called a *path*. The *length* of a finite walk p is its number of edges. Just as strings can be concatenated to form longer strings, walks can be concatenated to form longer walks under the condition that the result is still a walk in G. Two walks $p = (p_0, p_1, p_2, \dots)$, $q = (q_0, q_1, q_2, \dots)$ in G are *distinct*, in symbols $p \neq q$, if they have different length or there is an index i such that $p_i \neq q_i$. A finite (resp. infinite) *string occurring in* G (or simply, a string of G) is any string $S \in \Sigma^*$ (resp. $S \in \Sigma^\omega$) such that there is a finite (resp. infinite) walk $p = (p_0, p_1, p_2, \dots)$ in G with $S = L(p) := L(p_0)L(p_1)L(p_2)\dots$. We say that p is an *occurrence* of S in G, that p *spells* S in G or that S has a *match* in G. A string S occurring in G is *repeated* if there are at least two distinct occurrences of S in G, in symbols $\exists p, q$ walks in G such that $p \neq q \wedge L(p) = L(q) = S$.

Throughout the paper, we will assume that every vertex has at least one in-neighbor or out-neighbor, so it holds that $|V| \leq 2|E|$, $|V| \in O(|E|)$ and we can simplify a complexity bound such as $O(|V| + |E|)$ into $O(|E|)$.

Remark 1 In solving LCSP, MSP, LRSP, and their strict counterparts, we can assume that for any input labeled graph $G = (V, E, L)$ it holds that $|V| \in O(|E|)$, because:

- the problems become trivial when considering only walks of length 0, in the sense that there is a common or repeated string of length one if and only if there are different vertices labeled with the same character in the respective graphs; if in G there are vertices without both incoming and outgoing edges, they can be treated separately since the strings they generate have all length 1, thus we will assume throughout the rest of the paper that every vertex $v \in V$ has at least one incoming or outgoing edge, meaning that $|E| \geq |V|/2$.
- the answer to LCSP and LRSP is the empty string ε if and only if the sets of labels used in G_1 and G_2 do not intersect, or if each vertex of G is labeled with a different character (implying $|\Sigma| \geq |V|$); this can be easily checked assuming we are working with an integer alphabet, if not it is still $O(|V| \log|V|)$, so we

will assume that there is a common or repeated string of length 1, unless stated otherwise.

2 The Labeled Direct Product

Recall that the direct product of two graphs is the graph whose vertex set is the Cartesian product of the vertex sets of the initial graphs where we have an edge between two vertices if there are corresponding edges in the initial graphs between vertices on the first component and between vertices on the second component. This product has been studied in the literature in both the undirected and directed setting, under the names *conjunction*, *tensor product*, *Kronecker product*, and others (see [38, p. 21] and [39]). We will use instead the *labeled* direct product of G_1 and G_2, obtained as the subgraph of the direct product of G_1 and G_2 induced by the vertices for which their two components have the same label. Although this notion is similar to the automaton recognizing the intersection of two automata (see [40]), the key difference is that the labeled direct product graph does not contain any pair of edges/transitions with mismatching labels.

2.1 Definition and Basic Properties

Consider the following definition, and see also Figs. 2 and 3.

Definition 1 (*Labeled direct product graph*) Given two Σ-labeled graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, we define *the labeled direct product graph* $G_1 \otimes G_2 = (V', E', L')$, where:

$$V' = \{(u, v) \in V_1 \times V_2 : L_1(u) = L_2(v)\},$$
$$E' = \Big\{\big((u, v), (u', v')\big) \in V' \times V' \ : \ (u, u') \in E_1 \wedge (v, v') \in E_2\Big\},$$

and L' is defined so that $L'(u, v) = L_1(u) = L_2(v)$ for each $(u, v) \in V'$.

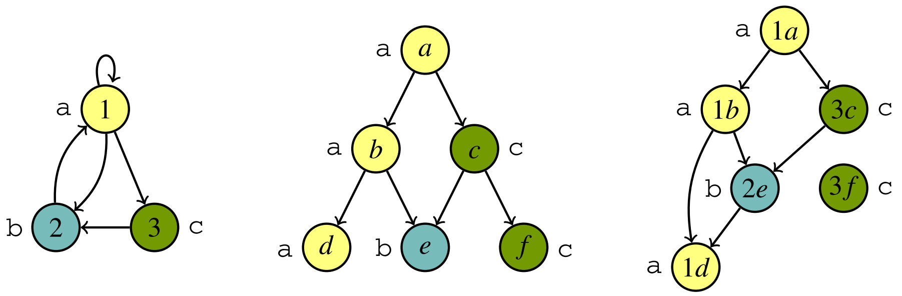

Fig. 2 An example of two {a, b, c}-labeled graphs G_1, G_2 and their labeled direct product graph $G_1 \otimes G_2$ on the right. Since G_2 is a DAG, $G_1 \otimes G_2$ is a DAG as well

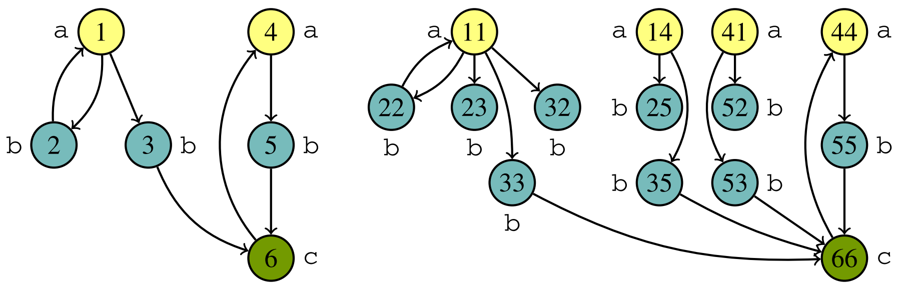

Fig. 3 A $\{a, b, c\}$-labeled directed graph G (left) and its labeled direct self-product $G \otimes G$ (right). Since G is cyclic, $G \otimes G$ is cyclic as well

Given a vertex $q = (u, v) \in V'$, let $\pi_1(q) := u$ and $\pi_2(q) := v$. Given a walk $q = \big((p_0, p'_0), (p_1, p'_1), \dots\big)$ in $G_1 \otimes G_2$, we denote with $\pi_1(q)$ and $\pi_2(q)$ the walks $(p_0, p_1, \dots)$ and $(p'_0, p'_1, \dots)$ in G_1 and G_2, respectively. We state the following basic fact about the correspondence between the pairs of walks in G_1 and G_2 and the walks in $G_1 \otimes G_2$. In particular, this implies that the projections of any cycle in $G_1 \otimes G_2$ are two cycles in G_1 and G_2 reading the same string and vice versa.

Remark 2 Given G_1, G_2 Σ-labeled graphs, for each pair $(p_0, p_1, \dots)$, $(p'_0, p'_1, \dots)$ of finite (resp. infinite) walks in G_1 and G_2, respectively, reading the same finite (resp. infinite) string $S \in \Sigma^*$ (resp. $S \in \Sigma^\omega$), $p \otimes p' := \big((p_0, p'_0), (p_1, p'_1), \dots\big)$ is a finite (resp. infinite) walk in $G_1 \otimes G_2$ reading S and vice versa.

Since all the algorithms we develop consist in analyzing the labeled direct product of the input graphs, we must take great care in the time and space spent on its construction. Moreover, in Remark 5 we show that its size can be computed efficiently, making the run time of our algorithms predictable.

Remark 3 The construction of $G_1 \otimes G_2 = (V', E', L')$ takes $O(|V_1| \cdot |V_2| + |E_1| \cdot |E_2|)$ time and space, because each pair of vertices and each pair of edges need to be considered at most once. Assuming Σ to be an integer or a constant-size alphabet we can do better than the naive construction algorithm with respect to time or space:

- if Σ is an integer alphabet, by first sorting lexicographically the lists of edges of G_1 and G_2, the product $G_1 \otimes G_2$ can then be built in linear-time with respect to its size, by simply pairing all edges of G_1 and G_2 with matching labels;
- if Σ has constant size, there is no need to store the edges of $G_1 \otimes G_2$, since for all $a \in \Sigma$ we can report in time linear in the solution all a-labeled out-neighbors of any vertex (u, v) by pairing all a-labeled out-neighbors of u and v in G_1 and G_2, respectively;
- if Σ is an integer alphabet, preprocessing G_1, G_2 in order to report the (number of) out-neighbors of any vertex (u, v) of $G_1 \otimes G_2$ is equivalent to the SetIntersection problem, for which Goldstein et al. proved conditional lower bounds on the trade-off between the space and time used in its solution [41]; if we choose not to store

at all the edges of $G_1 \otimes G_2$, the algorithms exploiting $G_1 \otimes G_2$ will then take $\Theta(|V'|)$ space and $O(|V'| + |E_1| \cdot |E_2|)$ time.[1]

Remark 4 Since vertices and edges in $G_1 \otimes G_2$ correspond to vertices and edges in G_1 and G_2 *with matching labels*, the size of $G_1 \otimes G_2$ could be much less than $|V_1| \cdot |V_2| + |E_1| \cdot |E_2|$ in practice, or for some families of labeled graphs. In particular, if each $a \in \Sigma$ is the label of at most $O(1)$ pairs of vertices in $V_1 \times V_2$ then $G_1 \otimes G_2$ has size $O(|V_1| + |V_2| + |E_1| + |E_2|)$: this is not in contradiction with the conditional lower bounds of Sect. 4 because the graph obtained in the reduction of Theorem 5 uses only two labels, $\Theta(|V|)$ times each.

Remark 5 If Σ is an integer alphabet, the size of $G_1 \otimes G_2$ can be computed in time linear in the size of the input graphs G_1 and G_2. Indeed, let V_1^a, V_2^a be the sets of a-labeled vertices of G_1, G_2, respectively, and let $E_i^{a,b}$ be the set edges of G_i connecting an a-labeled vertex to a b-labeled vertex, with $i = 1, 2$. Then it is easy to see that

$$|V'| = \sum_{a \in \Sigma} |V_1^a| \cdot |V_2^a| \quad \text{and} \quad |E'| = \sum_{a,b \in \Sigma} |E_1^{a,b}| \cdot |E_2^{a,b}|$$

and that $|V'| + |E'|$ can be easily computed after sorting the vertex and edge sets of G_1 and G_2. Note that if Σ has constant size, the size of $G_1 \otimes G_2$ can be found in constant time after the independent sorting of G_1 and G_2.

2.2 Optimal Algorithms for DAGs

We first consider the case when the direct product graph is a DAG. Note that $G \otimes G$ is DAG if and only if G itself is, $G_1 \otimes G_2$ is a DAG if at least one between G_1 and G_2 is a DAG, but $G_1 \otimes G_2$ might be a DAG even if both G_1 and G_2 contain cycles.

Thanks to Remark 2, SMLG, LCSP, MSP and LRSP can be solved by finding paths of maximum length in the corresponding direct product graph: an occurrence of pattern S in graph G corresponds to a path of length $|S| - 1$ in $G \otimes G_S$, where G_S is a labeled path of $|S|$ vertices spelling S; a longest common string of G_1 and G_2 is spelled by a path of maximum length in $G_1 \otimes G_2$; the matching statistics $\mathrm{MS}(v)$ of G_1 and G_2, with $v \in V_1$, is equal to one plus the length of a path of maximum length in $G_1 \otimes G_2$ starting from any vertex in $\{v\} \times V_2$; a longest repeated string of a DAG G is spelled by a path of maximum length of $G \otimes G$ visiting at least one vertex (u, v) such that $u \neq v$.

Indeed, for every vertex (u, v) of the product graph (V', E', L') we can compute by dynamic programming the length $\ell^+(u, v)$ of the longest path starting at (u, v):

- SMLG is solved by finding a path of length $|S|$ starting from a vertex (u, v) such that $\ell^+(u, v) = |S|$;

[1] We speculate that a careful implementation of our algorithms using bitvectors might take $\Theta(|V'|)$ space and $O\big(|V'| + |E'| + |V'| \cdot |\Sigma| / \log(|\Sigma|)\big)$ time. The conditional lower bounds by Goldstein et al. ignore logarithmic factors, so this would not be a contradiction.

- LCSP is solved by finding a vertex (u, v) of $G_1 \otimes G_2$ such that $\ell^+(u, v)$ has maximum value and by retrieving the string corresponding to a path of length $\ell^+(u, v)$ starting at (u, v);
- MSP is solved by finding for each $v \in V_1$ the maximum value of $\ell^+(v, w) + 1$, with (v, w) a vertex of $G_1 \otimes G_2$, and this can be done by iterating once over all vertices in V'.

We can analogously compute for each $(u, v) \in V'$ the length $\ell^-(u, v)$ of the longest path in (V', E', L') ending at (u, v):

- LRSP is solved by iterating over all vertices (u, v) of $G \otimes G$ such that $u \neq v$, and obtaining the length of the longest repeated string in G whose occurrences pass through the distinct vertices u and v, as $\ell^+(u, v) + \ell^-(u, v) + 1$ (a longest repeated string of this length can then be retrieved).

Theorem 1 *Given $G_1 = (V_1, E_1, L_1)$, $G_2 = (V_2, E_2, L_2)$ Σ-labeled directed graphs, LCSP and MSP on G_1, G_2 are solvable in $O(|E_1| \cdot |E_2|)$ time and taking $O(|V_1| \cdot |V_2|)$ words in space. Analogously, LRSP on a Σ-labeled graph $G = (V, E, L)$ is solvable in $O(|E|^2)$ time and taking $O(|V|^2)$ words in space. For all three problems plus SMLG, if the product graph is given, then the solution takes linear time in the size of the product graph.*

3 Optimal Algorithms for General Graphs

Since strict-LCSP, strict-MSP, and strict-LRSP are NP-complete (see Problem 5 and Sect. 4), in this section we focus on LCSP, MSP, and LRSP. If we deal with graphs containing cycles, then the length of the walks and strings to consider is not bounded anymore so we modify the three problems to require the detection of the relative cases. In fact, we will show that in all cases we can also report a *linear-size* representation of the corresponding common or repeated strings. As we stated in the introduction, the three problems admit worst-case quadratic-time solutions based on the labeled direct product graph, that is, $G_1 \otimes G_2$ for LCSP and MSP and $G \otimes G$ for LRSP.

Definition 2 Given a labeled direct product graph $G_1 \otimes G_2$ or $G \otimes G$, we define:

- V'_{cyc} as the set of all vertices of the product graph involved in a cycle, namely those belonging to a strongly connected component (SCC) consisting of at least two vertices;

also, for $G \otimes G = (V', E', L')$ we define:

- V'_{diff} as the set of vertices $(u, v) \in V'$ with $u \neq v$;
- V'_{ndet} as the set of all vertices $(v, v) \in V'$ with v a *non-deterministic* vertex of G, that is, v has two out-neighbors labeled with the same character.

3.1 LCSP and MSP

Since the graphs can contain cycles, the common strings in LCSP and MSP can now have infinite length. The algorithm solving LCSP on any two Σ-labeled graphs G_1, G_2 consists of the following simple checks in $G_1 \otimes G_2$:

Infinite length common strings Check if $G_1 \otimes G_2$ contains a cycle; if so, return (i) the string spelled by any cycle and (ii) the symbol ω; otherwise
Finite length common strings Proceed as in the algorithm for the DAG case from Sect. 2.2 on $G_1 \otimes G_2$.

The correctness of this algorithm follows from the fact that there is a common string of infinite length if and only if there is a common string of infinite length of the form S^ω (see also Lemma 1 below).

MSP can be solved as well by studying the SCCs of $G_1 \otimes G_2 = (V', E', L')$:

Infinite length matching statistics For all $(u, v) \in V'_{\text{cyc}}$ set $\text{MS}(u) = \infty$.
Finite length matching statistics Proceed for the remaining vertices of V_1, i.e. the vertices $u \in V_1$ such that no $(u, v) \in V'$ is also in V'_{cyc}, as in the algorithm for the DAG case from Sect. 2.2. Note that in this second step we consider an acyclic subgraph of $G_1 \otimes G_2$.

Theorem 2 *The above algorithms correctly solve* LCSP *and* MSP. *Moreover, if $G_1 \otimes G_2$ is given, they can be implemented to run in time linear in the size of $G_1 \otimes G_2$. If not, they run in time $O(|E_1| \cdot |E_2|)$, where E_1 and E_2 are the edge sets of G_1 and G_2, respectively.*

3.2 LRSP

In LRSP on general graphs we have one of the following *three* cases, as seen in Fig. 4:

1. The graph has an infinite repeated string.
2. The graph does not have any infinite repeated string, but the length of the repeated strings is unbounded.
3. The length of the longest repeated string is bounded and there are repeated strings of a finite maximum length (as is the case for texts, trees and DAGs).

An undesirable feature of infinite strings is that they can be aperiodic. However, analogously to some results of Büchi automata theory stating that the "important" strings are *ultimately periodic* [42, p. 137], that is they are of the form RS^ω with $R \in \Sigma^*$ and $S \in \Sigma^+$, in Lemma 1 we show that the presence of infinite repeated

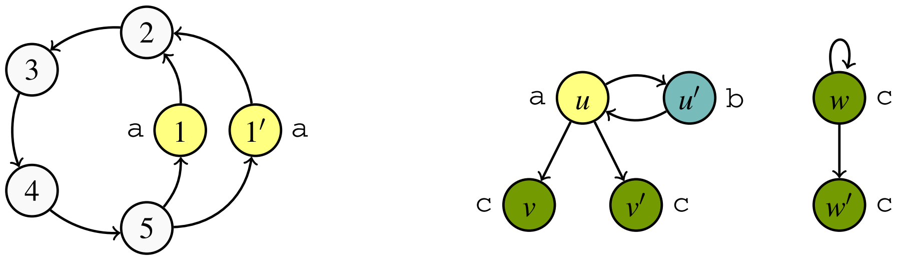

Fig. 4 An example (left) of a non-deterministic graph G with two distinct cycles $(1, 2, 3, 4, 5)$ and $(1', 2, 3, 4, 5)$; their infinite repetition generates the same infinite string; and a non-deterministic graph (right) with no infinite repeated strings but with finite repeated strings of unbounded length, precisely of the form $(\mathtt{ba})^k\mathtt{c}$, $(\mathtt{ab})^k\mathtt{ac}$ and $\mathtt{c}^k\mathtt{c}$, for every $k \geq 0$

strings can be detected by looking for ultimately periodic strings. Its easy proof, which we omit, finds a cycle in $G \otimes G$ used by two distinct occurrences of an infinite repeated string w (since G is finite): we can build RS^ω by identifying R as the prefix of w spelled by the path reaching the cycle and S as the string spelled by the cycle.

Lemma 1 *Given a Σ-labeled graph $G = (V, E, L)$, there is an infinite repeated string occurring in G if and only if there is a string $RS^\omega \in \Sigma^\omega$ in G spelled by two distinct walks rs^ω and $r's'^\omega$ in G, with $R = L(r) = L(r') \in \Sigma^*$ and $S = L(s) = L(s') \in \Sigma^+$.*

The two distinct walks spelling RS^ω provided by Lemma 1 imply the existence of a walk in $G \otimes G$ passing trough a vertex q in V'_{diff} and reaching a vertex q' in V'_{cyc}. Note that $q = q'$ can hold, in which case the infinite repeated string is of the form S^ω. Thus, we obtain:

Corollary 1 (Infinite repeated strings) *G has an infinite repeated string if and only if any $q \in V'_{diff}$ reaches any $q' \in V'_{cyc}$, and if R is the spelling of a path from q to q' and S is the spelling of a cycle starting and ending in q', then RS^ω is an infinite repeated string in G.*

If the graph has no infinite repeated string, the remaining difficulty is that of repeated strings of unbounded length. Formally, we say that G has *repeated strings of unbounded length* if for each $n \in \mathbb{N}$ there is a repeated string $S \in \Sigma^*$ occurring in G such that $|S| > n$. It is easy to see that in the graph of Fig. 4 (right) there are no infinite repeated strings and the unbounded repeated strings are of the form R^+S, with $R, S \in \Sigma^+$. Indeed, these unbounded strings are of this form and their occurrences have a common prefix after which they diverge. This divergence happens by visiting a non-deterministic vertex, as shown by the next two results.

Lemma 2 *Given a Σ-labeled graph $G = (V, E, L)$ without infinite repeated strings, G has repeated strings of unbounded length if and only if there are $R, S \in \Sigma^+$ such that $R^m S$ is repeated in G for each $m \geq 1$.*

Proof ($\Leftarrow$) This side is trivial. ($\Rightarrow$) Let $G \otimes G = (V', E', L')$. If G has repeated strings of unbounded length, then there must be some $k \in \mathbb{N}$ such that there is a repeated string $T \in \Sigma^*$ of length $k > |V'|$, with $p = (p_0, p_1, \ldots, p_{k-1})$ and $p' = (p'_0, p'_1, \ldots, p'_{k-1})$ two distinct occurrences of T in G. Then $q := (q_0, q_1, \ldots, q_{k-1}) := p \otimes p'$ is a walk in $G \otimes G$ visiting more than $|V'|$ vertices, so by the pigeonhole principle there must be a vertex visited more than once: let $j, j' \in \mathbb{N}$ be two indices such that $0 \leq j < j' \leq k-1$ and $q_j = q_{j'}$. Since p and p' are distinct walks in G, there must be also an index $i \in \mathbb{N}$ such that $0 \leq i \leq k-1$ and $p_i \neq p'_i$. Index i can be in three different positions relative to j and j':

1. if $i < j < j'$, then $(q_i, \ldots, q_{j-1})(q_j, \ldots, q_{j'-1})^\omega$ is an infinite and ultimately periodic walk in $G \otimes G$ and its projections are occurrences of an ultimately periodic, infinite and repeated string in G, since $\pi_1(q_i) = p_i \neq p'_i = \pi_2(q_i)$, contradicting our hypothesis;
2. if $j \leq i \leq j'$, then $(q_j, \ldots, q_{j'-1})^\omega$ is a periodic walk in $G \otimes G$ and its projections are occurrences of a periodic, infinite and repeated string in G, a contradiction;

3. if $j < j' < i$, then $(q_j, \ldots, q_{j'-1})(q_{j'}, \ldots, q_i)$ is a walk in $G \otimes G$ with a cyclic prefix that can be pumped, so $(q_j, \ldots, q_{j'-1})^m(q_{j'}, \ldots, q_i)$ is a walk in $G \otimes G$ for each $m \geq 1$; the projections in G of these strings are occurrences of repeated strings of the form $R^m S$, with $R = L'\big((q_j, \ldots, q_{j'-1})\big)$ and $S = L'\big((q_{j'}, \ldots, q_i)\big)$, since $\pi_1(q_i) = p_i \neq p'_i = \pi_2(q_i)$. □

Point 3. of the above proof shows that the index where the projections of the distinct walks differ must occur after every cycle of the walk considered in $G \otimes G$, proving that any of these walks has a proper prefix of vertices of the form (u, u) containing a cycle and this prefix ends in a vertex $(v, v) \in V'_{\text{ndet}}$. Note that $(u, u) = (v, v)$ can hold. We obtain the following result.

Corollary 2 (Unbounded repeated strings) *If G has no infinite repeated string, then G has repeated strings of unbounded length if and only if any $(u, u) \in V'_{cyc}$ reaches any $(v, v) \in V'_{ndet}$ with a path. In this case, if R is the spelling of a cycle starting and ending in (u, u), S is the spelling of a path starting from (u, u) and ending in (v, v), and c is the label of at least two out-neighbors of (v, v), then $R^m Sc$ is a repeated string for each $m \geq 1$.*

We solve LRSP on the general graph G by combining Corollaries 1 and 2 and Theorem 1:

Infinite length repeats Check if any $q \in V'_{\text{diff}}$ reaches any $q' \in V'_{\text{cyc}}$ even with an empty path. If so, return (i) the string spelled by the path from q to q', (ii) the string spelled by any cycle starting from q' and (iii) the symbol ω.
Unbounded length repeats Check if any $(u, u) \in V'_{\text{cyc}}$ reaches any $(v, v) \in V'_{\text{ndet}}$ even with an empty path. If so, return (i) the string spelled by any cycle starting (and ending) at (u, u), (ii) the symbol $+$ and (iii) the string spelled by the path from (u, u) to an out-neighbor of (v, v) with a sibling having the same label (since $(v, v) \in V'_{\text{ndet}}$).
Finite length repeats Remove from $G \otimes G$ all vertices in V'_{cyc} (obtaining a DAG), and proceed as in the algorithm for the DAG case from Sect. 2.2 on this graph.

Theorem 3 *The above algorithm correctly solves LRSP. Moreover, if $G \otimes G$ is given, it can be implemented to run in time linear in the size of $G \otimes G$. If not, it runs in time $O(|E|^2)$, where E is the edge set of G.*

Proof If G has infinite repeated strings, then Corollary 1 guarantees the correctness of the first check. Otherwise, Corollary 2 guarantees the correctness of the second check.

Suppose now that both of these checks return false. First, since the first check failed, $V'_{\text{diff}} \cap V'_{\text{cyc}} = \emptyset$, because any vertex in $V'_{\text{diff}} \cap V'_{\text{cyc}}$ reaches itself with an empty path. Second, from any $q \in V'_{\text{cyc}}$ no vertex $q' \in V'_{\text{diff}}$ is reachable (with a non-empty path, since $V'_{\text{diff}} \cap V'_{\text{cyc}} = \emptyset$). Indeed, suppose for a contradiction that $q' \in V'_{\text{diff}}$ is a vertex reached from q with a shortest (non-empty) path P, and let $q^* \in V'$ be the vertex on this path right before q'. Since P is shortest, then $q^* \notin V'_{\text{diff}}$, and thus $q^* \in V'_{\text{ndet}}$. However, this contradicts the assumption that the second check of the algorithm returned false.

Finally, since the two occurrences of a repeated string must pass through a vertex in V'_{diff}, and no vertex in V'_{diff} is reached, or reaches a vertex in V'_{cyc}, then we can remove all vertices in V'_{cyc} from $G \otimes G$, obtaining a DAG. In this DAG, as in Sect. 2.2, we look for the longest path passing through a vertex in V'_{diff}.

The SCCs of $G \otimes G$ and the sets V'_{cyc}, V'_{diff}, V'_{ndet} can be computed in linear time in the size $G \otimes G$. Reachability between two sets of vertices of $G \otimes G$ (and a corresponding path) can also be implemented in linear time in the size of $G \otimes G$. The algorithm for the final DAG case runs in linear time in the size of $G \otimes G$, by Theorem 1. □

4 Hardness

The NP-hardness of strict-SMLG (the SMLG problem defined on path occurrences, see Problem 5 and Sect. 1.7), implying the NP-hardness of strict-LCSP and of strict-MSP was already observed in previous works such as [6]. We similarly observe that the same holds also for strict-LRSP.

Observation 1 strict-LRSP is NP-hard, even if we restrict alphabet Σ to contain just a single character. This follows by reducing from the Hamiltonian Path problem on directed graphs. Given a graph G, create a graph G' made up of two copies of G and label all vertices with the same character. It easily holds that G has a Hamiltonian path if and only if the length of the longest repeated string in G' equals the number of vertices of G.

As noted in the introduction, the quadratic lower bounds of [6, 8] for SMLG imply the same quadratic lower bounds for LCSP and MSP, namely that the two problems cannot be solved in truly sub-quadratic time under the Orthogonal Vectors Hypothesis (that we discuss below in this section) and that the shaving from the quadratic-time complexity of arbitrarily high or high enough logarithmic factors would contradict other hardness conjectures. We now show a linear-time reduction from SMLG to LRSP on deterministic DAGs, which thus implies the same lower bounds for LRSP as in [6, 8] (since they hold also for deterministic DAGs).

Theorem 4 *Given a string $P \subseteq \Sigma^*$ and a Σ-labeled deterministic DAG $G = (V, E, L)$, there exists a Σ'-labeled DAG G', with $\Sigma' = \Sigma \cup V$, having $O(|V| + |E| + |P|)$ vertices and edges, computable in linear time in the size of P and G, and such that P has an occurrence in G if and only if the longest repeated string of G' has length $|V| + |P| + 1$.*

Proof Given a deterministic graph $G = (V, E, L)$ and a pattern $P = P[1] \cdots P[m]$ labeled on alphabet Σ, with $V = \{v_1, \ldots, v_n\}$ and $m, n > 0$, the reduction consists of transforming pattern P into a labeled graph G_P, that is a path of m vertices spelling P, and building a Σ'-labeled graph G' with $\Sigma' = \Sigma \cup V$, assuming $V \cap \Sigma = \emptyset$. Graph G', as seen in Fig. 5, contains G, G_P, and two copies H_1, H_2 of a simple gadget H appropriately connected to them. Gadget H is made of a path of n vertices spelling a^n, with $a \in \Sigma$ chosen arbitrarily, ending in a level of n vertices each labeled with a

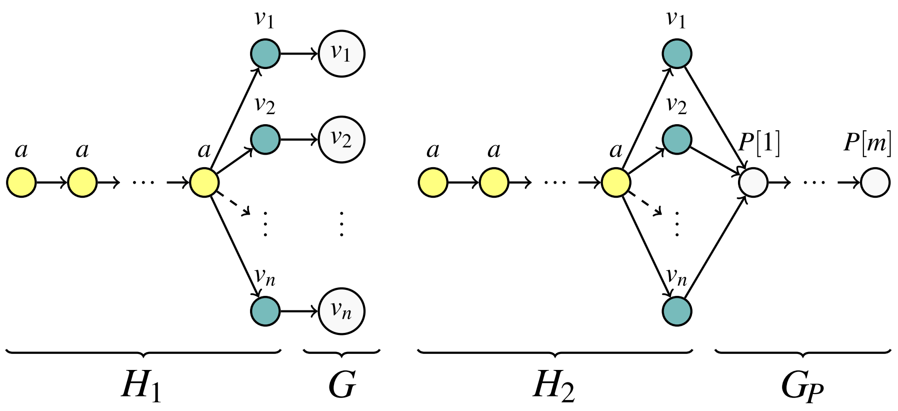

Fig. 5 Scheme for the reduction of SMLG to LRSP: H_1 and H_2 are two copies of the same gadget made of $n+1$ levels and the edges of G are not shown. Note that the reduction holds only if G is a deterministic DAG

different vertex of V. In H_1 each of these final vertices is connected to the respective vertex of G and in H_2 they are all connected to the source of G_P. The resulting graph G' is a deterministic DAG made of two connected components each having exactly one source, and it is easy to see that the longest repeated string of G' has length at most $|V|+|P|+1$. Each repeated string of this maximum length has one occurrence per component, starting at the respective source. If P has an occurrence $(u_1, \ldots, u_m)$ in G, then $a^n u_1 P$ is a repeated string in G' of maximum length. Conversely, every repeated string in G' of maximum length $|V|+|P|+1$ is of the form $a^n u_i P$, with $u_i \in V$, and its occurrence in the first component has as its suffix an occurrence of P in G.

Graph G' has $O(|V|+|E|+|P|)$ vertices and edges and its construction is straightforward, so the reduction takes linear time in the size of the starting SMLG instance. □

Two interesting aspects of Theorem 4 are as follows:

- the reduction does not hold if G contains cycles or if G is a DAG with some non-deterministic vertices, because there could be infinite repeated strings in G or the repeated strings of G could be extended by gadget H_1;
- to the best of our knowledge, such a reduction does not exist when the problems are defined on strings.

Nevertheless, this reduction creates an instance of LRSP with vertices of arbitrarily high in-degree and out-degree, and also increases the alphabet size by the number of edges of the graph. Therefore, in the rest of this section we give a direct reduction from the Orthogonal Vectors Problem (OVP) to LRSP, which will allow for both constant in- and out-degree, and binary alphabet.

In OVP we are given two sets of binary vectors $A, B \subseteq \{0,1\}^d$, with $|A| = |B| = n$ and $d = \omega(\log n)$, and we need to determine whether there exist $a \in A$, $b \in B$ so that $a \cdot b = 0$, where $a \cdot b = \sum_{i=1}^{d} a[i] \cdot b[i]$. The Orthogonal Vectors Hypothesis (OVH) states that no (randomized) algorithm can solve OVP on instances of size n in

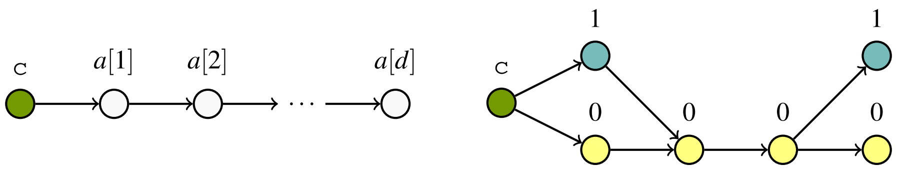

Fig. 6 Gadget G_a (left), with $a = (a[1], \dots, a[d]) \in A$, and $\overline{G}_b$ (right), with $b = (0, 1, 1, 0) \in B$. Note that since $b[2] = b[3] = 1$, the second and third levels of $\overline{G}_b$ only have one 0-labeled vertex

$O(n^{2-\varepsilon}\,\text{poly}(d))$ time for constant $\varepsilon > 0$ [33]. Given an instance A, B for OVP, we will construct a DAG G such that A and B contain a pair of orthogonal vectors if and only if the length of the longest repeated string in G is of a certain value, to be introduced at the end of the reduction.

To start with, we use the alphabet $\Sigma = \{0, 1, \mathtt{c}\}$, where $\mathtt{c}$ is used to simplify the proofs. At the end, we will observe that all $\mathtt{c}$-labeled vertices can be relabeled with 0. We start by building two types of gadgets:

- for each $a = (a[1], \dots, a[d]) \in A$, graph G_a is a path consisting of a starting $\mathtt{c}$-labeled vertex followed by d vertices, where the i-th vertex is labeled with $a[i]$ (Fig. 6, left);
- for each $b = (b[1], \dots, b[d]) \in B$, graph $\overline{G}_b$ is a DAG with $d + 1$ levels such that: (i) the zeroth level consists of a single $\mathtt{c}$-labeled source vertex; (ii) the i-th level has both a 0-labeled vertex and a 1-labeled vertex if $b[i] = 0$, otherwise (if $b[i] = 1$) it just has a 0-labeled vertex. All vertices in each level have edges going to all vertices in the next level, and there are no edges between vertices of the same level (Fig. 6, right). This is the same type of gadget used also in [6].

To build up the intuition, take $a \in A$ and $b \in B$ and observe, similarly as in [6], that the string spelled by G_a has an occurrence in $\overline{G}_b$ if and only if a and b are orthogonal. Thus, the graph made up of a copy of G_a and a copy of $\overline{G}_b$ has a longest repeated string of length $d + 1$ if and only if a and b are orthogonal. However, we cannot put together all gadgets G_a and $\overline{G}_b$ as separate components of the same graph, because such a simple construction cannot restrict the location of occurrences of the longest repeated string. Intuitively, we need the longest repeated string to have one occurrence in the part of the graph corresponding to the G_a gadgets, and one occurrence in the part of the graph corresponding to the $\overline{G}_b$ gadgets. We achieve this by (i) building a tree structure on top of the $\overline{G}_b$ gadgets that assigns to each gadget its own unique prefix; and by (ii) building a "universal" structure on top of gadgets G_a to make them reachable by reading any possible prefix added to gadgets $\overline{G}_b$. More specifically, we introduce the following two gadgets with $\lceil \log_2 n \rceil + 1 = k + 1$ levels:

- gadget T, seen in Fig. 7 (left), is a complete binary tree of height $k + 1$ and with $2^k \geq n$ leaves, in which the root is $\mathtt{c}$-labeled, all left children are 0-labeled and all right children are 1-labeled; trivially, any root-to-leaf path in such a tree has a different label;
- a "universal" DAG U with a $\mathtt{c}$-labeled source followed by k levels of vertices where each level has two vertices, labeled with 0 and 1, and each vertex in a level is connected to the vertices of the next level, as can be seen in Fig. 7 (right).

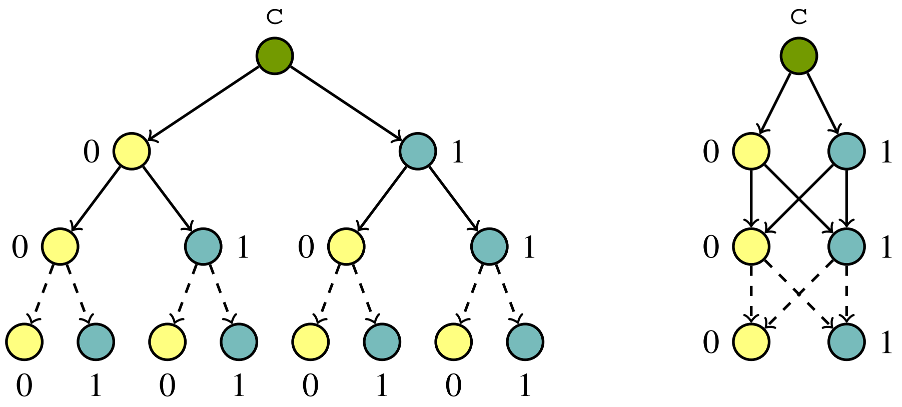

Fig. 7 Gadget T (left), a complete binary tree with $k+1$ levels, and gadget U (right), reading every possible string of length $k+1$ that can be read in T

Our gadgets can be arranged in a *non-deterministic* DAG as seen in Fig. 8 (left): the two sinks of gadget U are connected to each source of gadgets G_a, with $a \in A$, and each leaf of the tree gadget T is connected to the source of a different gadget $\overline{G}_b$, with $b \in B$; if n is not a power of two, some leaves of gadget T can be left without any out-neighbors. To have a *deterministic* DAG, we can further merge all gadgets G_a in a keyword tree (trie) $K_{G_{a_1},\dots,G_{a_n}}$ (see Fig. 8, right), so the set of strings spelled by the entire graph is unchanged (and the leaves of the keyword tree remain all distinct since all vectors in A are distinct). Note that the longest path in this graph has length $k+d+1$, with $k = \lceil \log_2 n \rceil$.

Lemma 3 *For an instance A and B for OVP, the deterministic DAG G built as in Fig. 8 has a repeated string of length $k+d+2$, with $k = \lceil \log_2 n \rceil$, if and only if there exist $a \in A$, $b \in B$ orthogonal.*

Proof By construction, the longest paths in G have length $k+d+1$ and thus they spell strings of length $k+d+2$. Moreover, all these longest strings are of the form $\mathtt{c}S_1\mathtt{c}S_2$, with $S_1 \in \{0,1\}^k$ and $S_2 \in \{0,1\}^d$.

($\Rightarrow$) If there is a repeated string $\mathtt{c}S_1\mathtt{c}S_2$ of length $k+d+2$ then there must be exactly two occurrences of it, one in G_A and one in G_B, since the graph is deterministic and has only two sources from which the longest strings can be read. This implies the existence of $a' \in A$, $b' \in B$ such that $L(a') = S_2$ and $a' \cdot b' = 0$, due to trie $K_{G_{a_1},\dots,G_{a_n}}$ and due to the properties of the longest strings of gadgets $G_{a'}$ and $\overline{G}_{b'}$.

($\Leftarrow$) Given $a' \in A$, $b' \in B$ such that $a' \cdot b' = 0$, let $\mathtt{c}S_1$, with $S_1 \in \{0,1\}^k$, be the string corresponding to the unique path from the $\mathtt{c}$-labeled source of G_B to gadget $\overline{G}_{b'}$. Then string $\mathtt{c}S_1\mathtt{c}S_2$, with $S_2 \in \{0,1\}^d$ the linearization of vector a', has two occurrences in G, one in G_A, passing through gadgets U and $K_{G_{a_1},\dots,G_{a_n}}$, and one in G_B, passing through T and $\overline{G}_{b'}$. □

To make the alphabet binary, it is easy to see that it suffices to relabel all $\mathtt{c}$-labeled vertices with 0. This proves that, under OVH, there can be no truly sub-quadratic time algorithm.

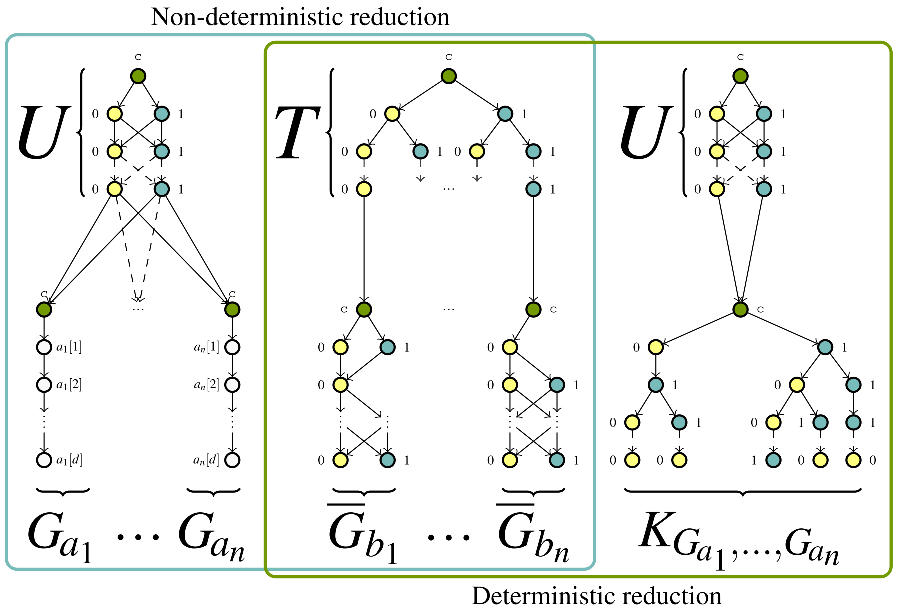

Fig. 8 First scheme (left) for the OV reduction, made of two subgraphs G_A (left) and G_B (right): G_A is non-deterministic; second scheme (right) for the OV reduction: $K_{G_{a_1},\dots,G_{a_n}}$ is the keyword tree (trie) of gadgets G_a, $a \in A$

Theorem 5 *If OVH holds, then for no $\varepsilon > 0$ there is a $O\left(|V|^{2-\varepsilon}\right)$-time or $O\left(|E|^{2-\varepsilon}\right)$-time algorithm for LRSP, even when restricted to deterministic DAGs, labeled with a binary alphabet, in which both the maximum in-degree and out-degree of any vertex are at most 2.*

Proof It is easy to check that the maximum in-degree, and out-degree of any vertex of the graph in the reduction of Fig. 8 is at most 2. Lemma 3 guarantees the correctness of the reduction, so it remains to analyze its complexity. The resulting graph G has $O(nd)$ vertices and $O(nd)$ edges and can be constructed in $O(nd)$ time, since the keyword tree can be constructed in time linear in the size of its inputs. Thus, if LRSP has an $O\left(|V|^{2-\varepsilon}\right)$-time or an $O\left(|E|^{2-\varepsilon}\right)$-time algorithm for some $\varepsilon > 0$, OVP has an $O\left((nd)^{2-\varepsilon}\right)$-time algorithm, contradicting OVH. □

Our OVH reduction for LRSP immediately proves an OVH reduction also for LCSP (by taking the two components of G as input graphs G_1 and G_2 for LCSP).

It also provides a quadratic lower bound for an apparently simpler version of MSP, which on two paths (i.e. strings) can be solved in a trivial manner. Let MSP* be defined as MSP, with the difference that we are given a *single* vertex v_1 of G_1, a *single* vertex v_2 of G_2, and we need to compute the length of the longest string having an occurrence in G_1 starting at v_1 and an occurrence in G_2 starting at v_2. To obtain the OVH reduction, it can be easily checked that it suffices to take as G_1 the subgraph of G built from B (Fig. 8, middle) with v_1 being its source vertex, and as G_2 the graph build from A (Fig. 8, right) with v_2 being its source vertex.

Corollary 3 *If OVH holds, then for no $\varepsilon > 0$ there is a $O\big((|V_1| \cdot |V_2|)^{1-\varepsilon}\big)$-time or $O\big((|E_1| \cdot |E_2|)^{1-\varepsilon}\big)$-time algorithm for MSP*, even when both input graphs are deterministic DAGs, labeled with a binary alphabet, in which the maximum in-degree and out-degree of any vertex is at most 2.*

Even though the above result about MSP* holds also for deterministic DAGs, its hardness stems from the fact that we do not know which path in G_1 to match with a path in G_2 in order to maximise their length. However, if G_2 is just a path, then the problem is solvable in linear time.

5 LRSP on Undirected Graphs

In this section we study LRSP and strict-LRSP on undirected graphs, namely when the occurrences of a repeated string are allowed to use an undirected edge in any of its two directions. Even if these results are easier than the results on directed graphs, they complete the complexity picture of LRSP. We will study the variants of LRSP and strict-LRSP and consider the same classes of undirected graphs: paths, trees[2] and general graphs.[3]

Theorem 6 *strict-LRSP on undirected graphs can be solved as follows:*

1. *On an undirected graph G that is a path, strict-LRSP can be solved in linear time.*
2. *On an undirected graph G that is a tree, strict-LRSP can be solved in quadratic time.*
3. *On general undirected graphs, strict-LRSP is NP-complete, since the same reduction as in Observation 1 works also for undirected graphs.*

Proof For 1., note that the occurrences of a repeated string are obtained by either moving only forward or only backward (because strict-LRSP is defined on path occurrences, and thus occurrences cannot repeat vertices). Thus, if T is the spelling of G from one end to another, we can reduce strict-LRSP on G to finding the longest repeated substring of the text $T\$T^{-1}$, where \$ is a new separator character, and T^{-1} is T reversed. Thus, we can solve strict-LRSP on G in linear time.

For 2., we show that strict-LRSP on an undirected tree with n vertices, defined with path occurrences, can be reduced to LRSP on a directed tree with $O(n^2)$ vertices (where there is no distinction between path and walk occurrences). Indeed, let $v_1, \ldots, v_n$ be the vertices of a Σ-labeled undirected tree T. For each $i \in \{1, \ldots, n\}$, construct the directed tree T_{v_i} by setting v_i as root and orienting all edges away from v_i. Also, let $(u_1, \ldots, u_n)$ be a directed path of n new vertices. Construct the directed tree T', rooted at u_1, by combining the path $(u_1, \ldots, u_n)$ with trees $T_{v_1}, \ldots, T_{v_n}$, adding the directed edges (u_n, v_i), for all $i \in \{1, \ldots, n\}$. The vertices of each T_{v_i} are labeled as in T, and $u_1, \ldots, u_n$ are labeled with a new character $\$ \notin \Sigma$. Clearly, the number of

[2] By *undirected tree* we mean an unrooted tree, where an occurrence can use an undirected edge in either direction.

[3] For simplicity, we assume that self-loops in undirected graphs are not present, even if they do not change the results.

vertices of T' is $n^2 + n$. We claim that T has a longest repeated string of length ℓ if and only if T' has a longest repeated string of length $n + \ell$, spelled by a path starting with $(u_1, \ldots, u_n)$. This is proved by combining the following remarks:

- All occurrences of any repeated string of T' longer than n must start in the same vertex u_i, with $i \in \{1, \ldots, n\}$, since $\$ \notin \Sigma$. Moreover, if a repeated string of *maximum* length in T' has length greater than n then its occurrences start at u_1.
- Consider two distinct occurrences of a longest repeated string in T' of length $n+\ell$, with $\ell \geq 1$, both starting from u_1. By construction of T', their suffixes of length ℓ correspond to two distinct occurrences of a string of length ℓ in T. Vice versa, given two distinct occurrences of a repeated string w in T, there are two corresponding distinct occurrences of $\$^n w$ in T' of length $n + \ell$, both starting from u_1.

Thus, the longest repeated strings in T correspond to the longest repeated strings in T' and vice versa (if there are no repeated strings in T then the repeated strings of T' have length lesser than n), so we can apply the linear-time solutions based on the suffix tree of a tree or the XBWT to obtain a globally quadratic-time algorithm.

For 3., observe that the same reduction as in Observation 1 works also for undirected graphs (since the Hamiltonian path problem is NP-hard also on undirected graphs).

□

For LRSP, which is defined on walk occurrences, observe first that we can replace each undirected edge with a pair of edges oriented in opposite directions. Thus, we can solve the problem in quadratic time, using the algorithm from Sect. 3 for directed graphs. However, we show that LRSP can be solved in *linear* time on general undirected graphs, using the following lemma, greatly simplifying the problem.

Lemma 4 *Given a Σ-labeled undirected graph $G = (V, E, L)$, G has a repeated string of length at least 2 if and only if G has an infinite repeated string.*

Proof Let $p = (p_0, p_1, \ldots, p_{k-1})$ and $p' = (p'_0, p'_1, \ldots, p'_{k-1})$ be distinct occurrences of a string, such that $p_i \neq p'_i$ for some $0 \leq i \leq k-1$. We can build an infinite repeated string with just two pair of adjacent vertices visited by p and p':

- if $i = 0$ then $(p_0, p_1)^\omega$, $(p'_0, p'_1)^\omega$ are distinct occurrences of infinite string $(L(p_0)L(p_1))^\omega$;
- if $i > 0$ then $(p_{i-1}, p_i)^\omega$, $(p'_{i-1}, p'_i)^\omega$ are distinct occurrences of infinite string $(L(p_{i-1})L(p_i))^\omega$. □

Lemma 4 implies we can just check if there is a repeated string of length 2 (in which case there is an infinite repeated string) and, if not, of length 1. These two checks can be done in linear time, provided that the vertex set and the edge set of G are already ordered lexicographically: there is a repeated string of length 2 if and only if there are two distinct edges with matching labels, and there is a repeated string of length 1 if and only if there are two distinct vertices with the same label.

6 Conclusions and Future Work

In this paper we introduced the labeled direct product graph as a straightforward algorithmic tool, since it naturally encodes all pairs of walks in the original graphs having matching labels. Through simple applications, we developed optimal and predictable algorithms for existing problems on labeled graphs—string matching in labeled graphs (SMLG) and longest common substring (LCSP)—and for extensions of string problems that we introduced—matching statistics (MSP) and longest repeated string (LRSP). For SMLG and LCSP this resulted in simpler and in some cases more efficient algorithms than the existing quadratic-time solutions, since the product graph excludes all pairs of mismatching vertices and edges. In the SMLG and graph-indexing setting, the labeled product graph can serve as a valuable comparison to the more sophisticated tools offering the string matching of arbitrary queries in sub-quadratic time.

Regarding complexity, we extended the existing conditional quadratic-time lower bounds for SMLG of [6, 8] to LRSP with a linear-time reduction, if the input graph of SMLG is a deterministic DAG. Since the SMLG lower bounds trivially hold for LCSP and MSP, this means that our algorithms (and the existing one for LCSP) are conditionally optimal. Moreover, we designed a single, more efficient reduction from the Orthogonal Vectors Problem (OVP) to LCSP, MSP and LRSP proving that the three problems cannot be solved in truly sub-quadratic time under the Orthogonal Vectors Hypothesis (OVH), even if the graphs in input are acyclic, deterministic (i.e. every vertex has at most one a-labeled out-neighbor, for every $a \in \Sigma$), labeled from a binary alphabet, and such that the maximum in-degree and out-degree of any vertex are at most 2. An interesting aspect of these results is that there is no known reduction of SMLG to LRSP when the problems are defined on strings and that the OVP reduction holds also for the modification of MSP trying to match the walks starting from just two vertices, even when the graphs have the same restrictions as before.

Our algorithms are based on linear-time analyses of the labeled direct product graph corresponding to each problem, so we spent some effort in studying its construction. Indeed, if the sets of vertices and edges of the graphs are sorted following the lexicographical order, then the construction of the product graph takes time and space linear in its size, thus under the standard assumption to work with an integer alphabet our algorithms globally reach this time and space complexity. This also means that the size of the labeled direct product graph is a finer complexity upper bound for SMLG, LCSP, MSP and LRSP. Plus, the size of the product graph can be precomputed in time linear in the size of the input graphs, making it possible to report the run time of our algorithms before their computation. If the alphabet has constant size, there is no need to store the edges of the product graph, whereas if the alphabet is an integer one then the choice not to store the edges of the graph is a version of the SetIntersection problem, leading to a space and time trade-off.

Finally, we presented a complete complexity picture of LCSP and LRSP on different classes of directed graphs and we did the same for LRSP and strict-LRSP on undirected graphs. The only open case is strict-LRSP in undirected trees, for which we obtained only a quadratic-time algorithm in Sect. 5, with no matching lower bound. Since the number of different paths of an undirected tree is only quadratic, we believe this problem cannot encode an OVP instance. Thus, we pose the open problem of finding a linear-time algorithm for this variant.

Recall that in the introduction we encoded a labeled graph as an NFA and we argued that SMLG, LCSP and LRSP (where we focus on finite strings) are special cases of similarly defined problems for finite-state automata (over finite words). The quadratic-time conditional lower bounds automatically carry over to these problems, and as the classical quadratic-size construction of an NFA recognizing each and every word accepted by two input NFAs solves LCSP, we deem that there is a quadratic-size NFA encoding all ambiguous words of any input NFA thus solving LRSP, and we leave this for future work.

We also leave as future work to find more problems on labeled graphs solved by the labeled direct product graph, or that can be tackled with the same general strategy of precomputing a data structure to globally obtain time savings during the actual computation.

Acknowledgements We are very grateful to Roberto Grossi, for initial discussions on the longest repeated string problem that spurred this line of research, and to Veli Mäkinen and Massimo Equi for their helpful comments and many discussions on the results of this paper.

Funding Open Access funding provided by University of Helsinki including Helsinki University Central Hospital. This work was partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 851093, SAFEBIO) and by the Academy of Finland (grants No. 322595, 328877).

Availability of data and material Not applicable.

Code availability Not applicable.

Declarations

Conflicts of interest/Competing interests The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Structuring labeled trees for optimal succinctness, and beyond. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings, pp. 184–196. IEEE Computer Society, (2005). https://doi.org/10.1109/SFCS.2005.69
2. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., Paten, B., Durbin, R.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nat. Biotechnol. **36**, 875 (2018). https://doi.org/10.1038/nbt.422710.1038/nbt.4227
3. Schneeberger, K., Hagmann, J., Ossowski, S., Warthmann, N., Gesing, S., Kohlbacher, O., Weigel, D.: Simultaneous alignment of short reads against multiple genomes. Genome Biol. **10**, 98 (2009)
4. Akutsu, T.: A linear time pattern matching algorithm between a string and a tree. In: 4th Symposium on Combinatorial Pattern Matching, Padova, Italy, pp. 1–10 (1993)

5. Backurs, A., Indyk, P.: Which regular expression patterns are hard to match? In: IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA, pp. 457–466 (2016)
6. Equi, M., Grossi, R., Mäkinen, V., Tomescu, A.I.: On the complexity of string matching for graphs. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece. LIPIcs, vol. 132, pp. 55–15515. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.55
7. Equi, M., Mäkinen, V., Tomescu, A.I.: Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In: Bureš, T., Dondi, R., Gamper, J., Guerrini, G., Jurdziński, T., Pahl, C., Sikora, F., Wong, P.W.H. (eds.) SOFSEM 2021: Theory and Practice of Computer Science, pp. 608–622. Springer, Cham (2021)
8. Gibney, D., Hoppenworth, G., Thankachan, S.V.: Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In: Le, H.V., King, V. (eds.) 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021, pp. 232–242. SIAM, (2021). https://doi.org/10.1137/1.9781611976496.26
9. Amir, A., Lewenstein, M., Lewenstein, N.: Pattern matching in hypertext. J. Algorithms **35**(1), 82–99 (2000)
10. Rautiainen, M., Marschall, T.: Aligning sequences to general graphs in $O(V + mE)$ time. bioRxiv, 216–127 (2017)
11. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence to graph alignment. In: Cowen, L.J. (ed.) Research in Computational Molecular Biology, pp. 85–100. Springer, Cham (2019)
12. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de bruijn graphs. In: Raphael, B.J., Tang, J. (eds.) Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7534, pp. 225–235. Springer, (2012). https://doi.org/10.1007/978-3-642-33122-0_18
13. Mäkinen, V., Välimäki, N., Sirén, J.: Indexing graphs for path queries with applications in genome research. IEEE ACM Trans. Comput. Biol. Bioinform. **11**(2), 375–388 (2014). https://doi.org/10.1109/TCBB.2013.2297101
14. Cotumaccio, N., Prezza, N.: On indexing and compressing finite automata. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021, pp. 2585–2599. SIAM, (2021). https://doi.org/10.1137/1.9781611976465.153
15. Nellore, A., Nguyen, A., Thompson, R.F.: An invertible transform for efficient string matching in labeled digraphs. In: Gawrychowski, P., Starikovskaya, T. (eds.) 32nd Annual Symposium on Combinatorial Pattern Matching, CPM 2021, July 5-7, 2021, Wrocław, Poland. LIPIcs, vol. 191, pp. 20–12014. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2021). https://doi.org/10.4230/LIPIcs.CPM.2021.20
16. Gusfield, D.: Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology. Cambridge University Press (1997). https://doi.org/10.1017/cbo9780511574931
17. Mäkinen, V., Belazzougui, D., Cunial, F., Tomescu, A.I.: Genome-Scale Algorithm Design: Biological Sequence Analysis in the Era of High-Throughput Sequencing. Cambridge University Press, (2015). https://doi.org/10.1017/CBO9781139940023
18. Parida, L.: Pattern Discovery in Bioinformatics: Theory & Algorithms, 1st edn. Chapman & Hall/CRC (2007)
19. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific (2002). https://doi.org/10.1142/4838
20. Ohlebusch, E.: Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction. Oldenbusch Verlag, (2013). http://www.oldenbusch-verlag.de/
21. Puglisi, S.J., Smyth, W.F., Turpin, A.: A taxonomy of suffix array construction algorithms. ACM Comput. Surv. **39**(2), 4 (2007). https://doi.org/10.1145/1242471.1242472
22. Shimohira, K., Inenaga, S., Bannai, H., Takeda, M.: Computing longest common substring/subsequence of non-linear texts. In: Holub, J., Zdárek, J. (eds.) Proceedings of the Prague Stringology Conference 2011, Prague, Czech Republic, August 29-31, 2011, pp. 197–208. Prague Stringology Club, Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, (2011). http://www.stringology.org/event/2011/p17.html
23. Han, Y., Salomaa, A., Salomaa, K.: Ambiguity, nondeterminism and state complexity of finite automata. Acta Cybern. **23**(1), 141–157 (2017). https://doi.org/10.14232/actacyb.23.1.2017.9

24. Colcombet, T.: Unambiguity in automata theory. In: Shallit, J.O., Okhotin, A. (eds.) Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9118, pp. 3–18. Springer, (2015). https://doi.org/10.1007/978-3-319-19225-3_1
25. Goldstine, J., Kappes, M., Kintala, C.M.R., Leung, H., Malcher, A., Wotschke, D.: Descriptional complexity of machines with limited resources. J. Univers. Comput. Sci. **8**(2), 193–234 (2002). https://doi.org/10.3217/jucs-008-02-0193
26. Book, R.V., Even, S., Greibach, S.A., Ott, G.: Ambiguity in graphs and expressions. IEEE Trans. Computers **20**(2), 149–153 (1971). https://doi.org/10.1109/T-C.1971.223204
27. Weber, A., Seidl, H.: On the degree of ambiguity of finite automata. Theor. Comput. Sci. **88**(2), 325–349 (1991). https://doi.org/10.1016/0304-3975(91)90381-B
28. Allauzen, C., Mohri, M., Rastogi, A.: General algorithms for testing the ambiguity of finite automata and the double-tape ambiguity of finite-state transducers. Int. J. Found. Comput. Sci. **22**(4), 883–904 (2011). https://doi.org/10.1142/S0129054111008477
29. Kosaraju, S.R.: Efficient tree pattern matching (preliminary version). In: 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989, pp. 178–183. IEEE Computer Society, (1989). https://doi.org/10.1109/SFCS.1989.63475
30. Breslauer, D.: The suffix tree of a tree and minimizing sequential transducers. Theor. Comput. Sci. **191**(1–2), 131–144 (1998). https://doi.org/10.1016/S0304-3975(96)00319-2
31. Shibuya, T.: Constructing the suffix tree of a tree with a large alphabet. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **86–A**(5), 1061–1066 (2003)
32. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997, pp. 137–143. IEEE Computer Society, (1997). https://doi.org/10.1109/SFCS.1997.646102
33. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoret. Comput. Sci. **348**(2), 357–365 (2005). https://doi.org/10.1016/j.tcs.2005.09.023
34. Burrows, M., Wheeler, D.: A block-sorting lossless data compression algorithm. In: Digital SRC Research Report (1994). Citeseer
35. Alanko, J., D'Agostino, G., Policriti, A., Prezza, N.: Wheeler languages. Inf. Comput. **281**, 104820 (2021). https://doi.org/10.1016/j.ic.2021.104820
36. Gagie, T., Manzini, G., Sirén, J.: Wheeler graphs: A framework for bwt-based data structures. Theor. Comput. Sci. **698**, 67–78 (2017). https://doi.org/10.1016/j.tcs.2017.06.016
37. Gibney, D., Thankachan, S.V.: On the hardness and inapproximability of recognizing wheeler graphs. In: Bender, M.A., Svensson, O., Herman, G. (eds.) 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany. LIPIcs, vol. 144, pp. 51–15116. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, (2019). https://doi.org/10.4230/LIPIcs.ESA.2019.51
38. Harary, F.: Graph Theory. Addison-Wesley (1991)
39. Harary, F., Wilcox, G.W.: Boolean operations on graphs. Mathematica Scandinavica, **20**(1), 41–51 (1967). https://www.jstor.org/stable/pdf/24490249.pdf
40. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). https://doi.org/10.1147/rd.32.0114
41. Goldstein, I., Kopelowitz, T., Lewenstein, M., Porat, E.: Conditional lower bounds for space/time trade-offs. In: Ellen, F., Kolokolova, A., Sack, J. (eds.) Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10389, pp. 421–436. Springer, (2017). https://doi.org/10.1007/978-3-319-62127-2_36
42. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics, pp. 133–191. Elsevier and MIT Press, (1990). https://doi.org/10.1016/b978-0-444-88074-1.50009-3