On the Complexity of String Matching for Graphs

Equi, Massimo

Schloss Dagstuhl - Leibniz-Zentrum für Informatik
2019

Equi , M , Grossi , R , Mäkinen , V & Tomescu , A 2019 , On the Complexity of String Matching for Graphs . in C Baier , I Chatzigiannakis , P Flocchini & S Leonardi (eds) , 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019) . Leibniz International Proceedings in Informatics (LIPIcs) , vol. 132 , Schloss Dagstuhl - Leibniz-Zentrum für Informatik , Dagstuhl, Germany , pp. 55:1--55:15 , International Colloquium on Automata, Languages and Programming , Patras , Greece , 08/07/2019 . https://doi.org/10.4230/LIPIc

http://hdl.handle.net/10138/310140
https://doi.org/10.4230/LIPIcs.ICALP.2019.55

On the Complexity of String Matching for Graphs

Massimo Equi
Department of Computer Science, University of Helsinki, Finland
massimo.equi@helsinki.fi

Roberto Grossi
Dipartimento di Informatica, Università di Pisa, Italy
grossi@di.unipi.it

Veli Mäkinen
Department of Computer Science, University of Helsinki, Finland
veli.makinen@helsinki.fi

Alexandru I. Tomescu
Department of Computer Science, University of Helsinki, Finland
alexandru.tomescu@helsinki.fi

Abstract

Exact string matching in labeled graphs is the problem of searching paths of a graph $G = (V, E)$ such that the concatenation of their node labels is equal to the given pattern string $P[1..m]$. This basic problem can be found at the heart of more complex operations on variation graphs in computational biology, of query operations in graph databases, and of analysis operations in heterogeneous networks.

We prove a conditional lower bound stating that, for any constant $\epsilon > 0$, an $O(|E|^{1-\epsilon}\, m)$-time, or an $O(|E|\, m^{1-\epsilon})$-time algorithm for exact string matching in graphs, with node labels and patterns drawn from a binary alphabet, cannot be achieved unless the Strong Exponential Time Hypothesis (SETH) is false. This holds even if restricted to undirected graphs with maximum node degree two, i.e. to *zig-zag matching in bidirectional strings*, or to *deterministic* directed acyclic graphs whose nodes have maximum sum of indegree and outdegree three. These restricted cases make the lower bound stricter than what can be directly derived from related bounds on regular expression matching (Backurs and Indyk, FOCS'16). In fact, our bounds are tight in the sense that lowering the degree or the alphabet size yields linear-time solvable problems.

An interesting corollary is that exact and approximate matching are equally hard (quadratic time) in graphs under SETH. In comparison, the same problems restricted to strings have linear-time vs quadratic-time solutions, respectively (approximate pattern matching having also a matching SETH lower bound (Backurs and Indyk, STOC'15)).

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases exact pattern matching, graph query, graph search, labeled graphs, string matching, string search, strong exponential time hypothesis, heterogeneous networks, variation graphs

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.55

Category Track A: Algorithms, Complexity and Games

Related Version This paper is based on our two technical reports [14, 15] available at `https://arxiv.org/abs/1901.05264` and `https://arxiv.org/abs/1902.03560`.

Funding This work has been partially supported by Academy of Finland (grant 309048).

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 55; pp. 55:1–55:15

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

1 Introduction

String matching is the classical problem of finding the occurrences of a pattern string as a substring of a text string [22]. As most of today's data is linked, it is natural to investigate string matching in labeled graphs. Indeed, large-scale labeled graphs are becoming ubiquitous in several areas, such as graph databases [6, 16, 30, 27], graph mining [19, 12], and computational biology [11]. Applications require sophisticated operations on these graphs, and often rely on primitives that locate paths whose nodes have labels or types matching a pattern given at query time. The most basic pattern is a string and, as we will see, this already poses a challenge when performing string matching in graphs.

Problem Definition

Given an alphabet Σ of symbols, consider a labeled graph $G = (V, E, L)$, where (V, E) represents a directed or undirected graph and $L : V \to \Sigma$ is a function that defines which symbol from Σ is assigned to each node as label.[1] A node labeled with $\sigma \in \Sigma$ is called a *σ-node*, and an edge whose endpoints are labeled σ_1 and σ_2, respectively, is called a *$\sigma_1\sigma_2$-edge*. If G is a directed graph, we say that G is *deterministic* if, for any two out-neighbors of the same node, their labels are different. In the following, we introduce the acronym *3-DDAG* to indicate a deterministic directed acyclic graph (DAG) such that its nodes are labeled with a binary alphabet and the sum of indegree and outdegree of each node is at most 3.

Given a pattern string $P[1..m]$ over Σ, we say that P has a *match* in G if there is a path $u_1, \ldots, u_k$ such that $P = L(u_1) \cdots L(u_k)$ (we also say that P *occurs* in G, and that $u_1, \ldots, u_k$ is an *occurrence* of P).

▶ **Problem 1** (String Matching in Labeled Graphs (SMLG)).
INPUT*: A labeled graph $G = (V, E, L)$ and a pattern string P, both over an alphabet Σ.*
OUTPUT*:* True *if and only if there is at least one occurrence of P in G.*

Results

We give conditional bounds for the SMLG problem using the Orthogonal Vectors (OV) hypothesis [34]. The latter states that for any constant $\epsilon > 0$, no algorithm can solve in $O(n^{2-\epsilon}\text{poly}(d))$ time the OV problem: given two sets $X, Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$ and $d = \omega(\log n)$, decide whether there exist $x \in X$ and $y \in Y$ such that x and y are orthogonal, namely, $x \cdot y = 0$. We observe that it is common practice to use the Strong Exponential Time Hypothesis (SETH) [20] but, since SETH implies the OV hypotesis [34], it suffices to use the OV hypothesis in the bounds, as they hold also for SETH.

First, we consider the SMLG problem on directed graphs. Their weakest form is a 3-DDAG, for which we prove in Section 2 that subquadratic time for exact string matching cannot be achieved unless the OV hypothesis is false.

▶ **Theorem 1.** *For any constant $\epsilon > 0$, the String Matching in Labeled Graphs (SMLG) problem for a binary alphabet and a labeled* deterministic *directed acyclic graph (DAG) cannot be solved in either $O(|E|^{1-\epsilon}\, m)$ or $O(|E|\, m^{1-\epsilon})$ time unless the OV hypothesis fails. This holds even if it is restricted to graphs in which the sum of outdegree and indegree of any node is at most three (i.e, 3-DDAGs).*

[1] Note that we can also define the node labels as nonempty strings, but it suffices to use single symbols to show that string matching in graphs is challenging.

Table 1 Legend: V = set of nodes, E = set of edges, occ = number of matches for the pattern in the graph, m = pattern length, N = total length of text in all nodes, *(1)* errors only in the pattern, *(2)* errors in the graph, *(3)* matches span only one edge. The two rows highlighted in gray report the best known bounds for exact and approximate string matching, respectively.

State of the art for SMLG				
Year	**Authors**	**Graph**	**Exact/ Approximate**	**Time**
1992	Manber, Wu [24]	DAG	approximate$^{(1)}$	$O(m\|E\| + occ \lg\lg m)$
1993	Akutsu [2]	tree	exact	$O(N)$
1995	Park, Kim [26]	DAG	exact$^{(3)}$	$O(N + m\|E\|)$
1997	Amir et al. [5]	general	exact	$O(N + m\|E\|)$
1997	Amir et al. [5]	general	approximate$^{(2)}$	NP-Hard
1997	Amir et al. [5]	general	approximate$^{(1)}$	$O(Nm \lg N + m\|E\|)$
1998	Navarro [25]	general	approximate$^{(1)}$	$O(Nm + m\|E\|)$
2017	Rautiainen, Marschall [29]	general	approximate$^{(1)}$	$O(N + m\|E\|)$
2019	Jain et al. [21]	general binary alphabet	approximate$^{(2)}$	NP-Hard

Next, we consider the SMLG problem on undirected graphs and introduce the *zig-zag* pattern matching problem in strings, which models searching a string P along a path of an undirected graph. While an exact occurrence of P in a text string is found by scanning the text forward for increasing positions in P, a *zig-zag* occurrence of P can be found by partially scanning forward and backward adjacent text positions, as many times as needed (e.g. for an edge $\{u, v\}$ with $L(u) = \mathtt{a}$ and $L(v) = \mathtt{b}$, all patterns of the form $\mathtt{a}, \mathtt{ab}, \mathtt{aba}, \mathtt{abab}, \ldots$ occur starting from u). We prove in Section 3 the following result.

▶ **Theorem 2.** *The conditional lower bound stated in Theorem 1 holds even if it is restricted to undirected graphs whose nodes have degree at most 2, where the pattern and the node labels are drawn from a binary alphabet.*

Our results can cover arbitrary graphs in this way. Interpreting the graphs from Theorem 2 as directed, we observe that they have nodes with both indegree and outdegree 2. Looking at Theorem 1, we observe that it involves directed graphs with both nodes of indegree at most 1 and outdegree 2, and nodes with outdegree at most 1 and indegree 2. Thus, the only uncovered case is that of directed graphs with only nodes of indegree at most 1, or directed graphs with only nodes of outdegree at most 1. For such graphs, we observe that their edges can be decomposed into forests of directed trees (arborescences), whose roots may be connected in a directed cycle (at most one cycle per forest). In the extended version of this work we will show that the Knuth-Morris-Pratt algorithm [22] can be easily extended to solve exact string matching for these special directed graphs in linear time, thus completing the full picture.

History and Implications

The idea of extending the problem of string matching to graphs, as given in SMLG, is not new. If the nodes $u_1, \ldots, u_k$ are required to be distinct (i.e., to be a *simple* path), this problem is NP-hard as it solves the well-known Hamiltonian Path problem, so this

requirement is removed for this reason. The SMLG problem was studied over 25 years ago as a search problem for hypertext by Manber and Wu [24]. The history of key contributions is given in Table 1, where a common feature of the reported bounds is the appearance of the quadratic term $m\,|E|$ (except for some special cases). The quadratic cost of the approximate matching in graphs is asymptotically optimal under the Strong Exponential Time Hypothesis (SETH) [20] as (i) it solves the approximate string matching as a special case, since a graph consisting of just one directed path of $|E|+1$ nodes and $|E|$ edges is a text string of length $n = |E|+1$, and (ii) it has been recently proved that the edit distance of two strings of length n cannot be computed in $O(n^{2-\epsilon})$ time, for any constant $\epsilon > 0$, unless SETH is false [7]. This conditional lower bound explains why the $O(m|E|)$ barrier has been difficult to cross in the approximate case. Specifically, Amir et al. [4, 5], gave a quadratic-time solution for exact string matching in $O(N + m \cdot |E|)$ time, where $N = \sum_{u \in V} |L(u)|$. Rautiainen and Marschall [29] and Jain et al. [21] recently gave the best bound for errors in pattern only, $O(N + m \cdot |E|)$ time, same as the exact string matching. The two best results for exact and approximate pattern matching, both taking quadratic time in the worst case, are highlighted in Table 1. As allowing errors in the graph makes the problem NP-hard [5], we consider here errors in the pattern only.

In this scenario and the application domains mentioned at the beginning, our results have a number of implications discussed below.

- While we can explain the complexity of *approximate* string matching in graphs, not much is known on the complexity of *exact* string matching in graphs. The classical exact string matching can be solved in linear time [22], so one could expect the corresponding problem on graphs to be easier than approximate string matching. A lower bound (i.e., NP-hard, as mentioned above) exists only in the case when the pattern is restricted to match only simple paths in the graph. Extensions of this type of matching for special graph classes have been studied in [23]. Here we study the general case, where paths can pass through nodes multiple times. Somewhat surprisingly Theorems 1 and 2 imply that *exact and approximate pattern matching are equally hard in graphs*, even if they are 3-DDAGs.
- Our results imply that the algorithm for directed graphs by Amir et al. [4, 5] is essentially the best we can hope for asymptotic bounds unless the OV hyptothesis is false. This also applies to the case of undirected graphs by the simple transformation so that each edge $\{u, v\}$ is transformed into a pair of arcs (u, v) and (v, u). Note that we need also Theorem 2 to explicitly state that this is the best possible also for undirected graphs of maximum degree 2. To complete the picture, we show how to get linear time for the above special case of directed graphs where each node has indegree at most 1, or directed graphs whose nodes have outdegree at most 1.
- Our results also explain why it has been difficult to find indexing schemes for fast exact string matching in graphs, with other than best-case or average-case guarantees [31, 17], except for limited search scenarios [32]. They complement recent findings about *Wheeler graphs* [17, 18, 3]. Wheeler graphs are a class of graphs admitting an index structure that can be constructed in linear time and that supports linear-time exact pattern matching. Gibney and Thankachan [18] claim that it is NP-complete to recognize whether a (non-deterministic) DAG is a Wheeler graph. Alanko et al. [3] claim a linear-time algorithm for recognizing whether a deterministic DAG is a Wheeler graph. Theorem 1 shows that converting an arbitrary deterministic DAG into an equivalent Wheeler graph should take at least quadratic time unless the OV hypothesis is false. In particular, the 3-DDAG obtained in the reduction from OV in the proof of Theorem 1 is not a Wheeler graph.

- We observe that, for any given pattern P, the 3-DDAG obtained by our reduction admits at most one occurrence per node, as it is deterministic and acyclic. When both P and a node u are given, it takes linear time to search P starting from u. Interestingly, if P alone is given, we observe that our results imply that an algorithm reporting whether there exists a node u from which an occurrence of P starts cannot take subquadratic time unless the OV hypothesis is false. Indeed, this hypothetical algorithm would be able to solve the SMLG problem also on that 3-DDAG.
- In the extended version of this work we will describe a simple transformation so that we can see our 3-DDAG and the pattern P as two DFAs, so that our SMLG problem reduces to the emptiness intersection for the string sets recognized by these two DFAs. In this way we give a quadratic conditional lower bound for the latter problem using OV. This adds to the results known in the literature for DFAs (tree automata) under SETH [33] and 3SUM [13]. It is worth noting that our SMLG problem on tree automata takes instead linear time, as will be discussed in the extended version of this work, and this seems to suggest that SMLG could be easier than emptiness intersection for two DFAs under SETH, even though both problems have conditional quadratic lower bounds.

Our reductions share some similarities with those for string problems [7, 10, 1, 8, 9]. The closest connection is with a conditional hardness of several forms of regular expression matching [8]. Especially, one could start with a *non-deterministic finite automaton (NFA)* derived from the regular expression matching of type $|\cdot|$, and add universal "jolly" gadgets (see our reduction) to come up with an OV lower bound for exact pattern matching in directed non-deterministic graphs. (For the interested reader, this is what we have done in an early version of this work [14].) However, to cover the deterministic and bounded degree cases, we build our reduction using a different strategy. This strategy yields a graph of small degree and enables local merging of non-deterministic subgraphs into deterministic counterparts. This locality feature of our reduction is crucial, since converting an NFA into a *deterministic finite automaton (DFA)* can take exponential time [28]. Finally, while this reduction works also for undirected graphs of small degree, it does not cover undirected graphs of degree two. For this case (zig-zag matching in a bidirectional string), we need a more intricate reduction as the underlying graph has less structure.

2 Deterministic Directed Acyclic Graphs

In this section we reduce the OV problem to the SMLG problem for the restricted case of 3-DDAGs. Since any SMLG algorithm for arbitrary directed graphs can be applied also to 3-DDAGs in the same complexity, we will show that a subquadratic-time SMLG algorithm would make the OV hypothesis false. In this scenario, 3-DDAGs are the most restricted case, as otherwise the SMLG problem can be solved in linear time.

Given an OV instance with sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ of d-dimensional binary vectors, we show how to build a pattern P and a 3-DDAG G such that P will have a match in G if and only if there exists a vector in X orthogonal to one in Y. We first describe how to build P and how to obtain a directed graph whose nodes are labeled with a constant-sized alphabet. Then we discuss how to turn such graph into the 3-DDAG G.

2.1 Pattern

Pattern P is over the alphabet $\Sigma = \{\mathtt{b}, \mathtt{e}, \mathtt{0}, \mathtt{1}\}$, has length $|P| = O(nd)$, and can be built in $O(nd)$ time from the first set of vectors $X = \{x_1, \ldots, x_n\}$. Namely, we define $P = \mathtt{bb}P_{x_1}\mathtt{e}\,\mathtt{b}P_{x_2}\mathtt{e}\ldots\mathtt{b}P_{x_n}\mathtt{ee}$ where P_{x_i} is a string of length d that is associated with $x_i \in X$,

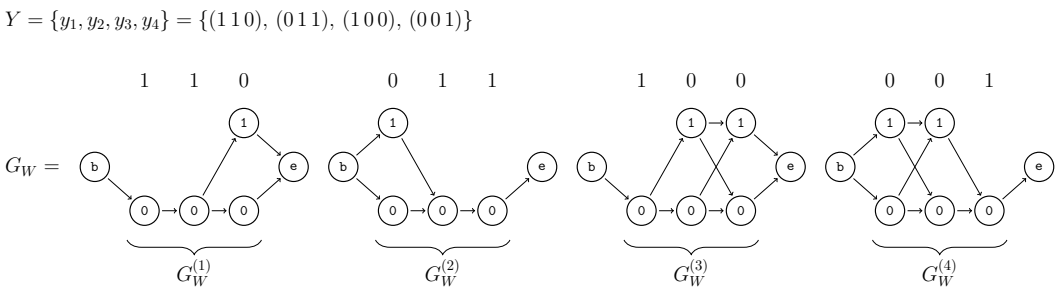

Figure 1 Gadget G_W.

for $1 \leq i \leq n$. The h-th symbol of P_{x_i} is either `0` or `1`, for each $h \in \{1, \ldots, d\}$, such that $P_{x_i}[h] = \texttt{1}$ if and only if $x_i[h] = 1$.[2] We thus view the vectors in X as subpatterns P_{x_i}s which are concatenated by placing separator characters `eb`. Note that P starts with `bb` and ends with `ee`: such strings are found nowhere else in P, marking thus its beginning and its end.

2.2 Directed Graph

The gadget implementing the main logic of the reduction is a directed graph $G_W = (V_W, E_W, L_W)$, illustrated in Figure 1. Starting from the second set of vectors Y, set V_W can be seen as n disjoint *groups* of nodes $V_W^{(1)}, V_W^{(2)}, \ldots, V_W^{(n)}$ (plus some extra nodes), where the nodes in $V_W^{(j)}$ are uniquely associated with vector $y_j \in Y$, for $1 \leq j \leq n$. The corresponding induced subgraph $G_W^{(j)} = (V_W^{(j)}, E_W^{(j)})$ will contain an occurrence of a subpattern P_{x_i} if and only if $x_i \cdot y_j = 0$. We give more details below.

The nodes in $V_W^{(j)}$ are defined as follows. For $1 \leq h \leq d$, we consider entry $y_j[h]$ of vector $y_j \in Y$. If $y_j[h] = 1$, we place just a `0`-node w_{jh}^0 to indicate that we only accept $P_{x_i}[h] = \texttt{0}$ for this h coordinate. Instead, if $y_j[h] = 0$, we place both a `0`-node w_{jh}^0 and a `1`-node w_{jh}^1 to indicate that the value of $P_{x_i}[h]$ does not matter. The nodes in $V_W^{(j)}$ are preceded by a special begin `b`-node $b_W^{(j)}$ and succeeded by a special end `e`-node $e_W^{(j)}$. The overall nodes are thus $V_W = \bigcup_{1 \leq j \leq n} (V_W^{(j)} \cup \{b_W^{(j)}, e_W^{(j)}\})$, and it holds that $|V_W| = O(nd)$.

As for the edges in $E_W^{(j)}$, they properly connect the nodes inside each group $V_W^{(j)}$. Specifically, node $b_W^{(j)}$ is connected to w_{j1}^0 and, if it exists, to w_{j1}^1. Also, we place edges connecting both nodes w_{jd}^0 and w_{jd}^1 (if this exists) to node $e_W^{(j)}$. Moreover, there is an edge for every pair of nodes that are consecutive in terms of h coordinate, for $1 \leq h < d$ (e.g., w_{jh}^1 is connected to $w_{j\,h+1}^0$). The overall edges are thus $E_W = \bigcup_{1 \leq j \leq n} E_W^{(j)}$, where $|E_W| = O(nd)$.

In this way we define the directed graph $G_W = (V_W, E_W, L_W)$, which can be built in $O(nd)$ time from set Y and consists of n connected components $G_W^{(j)}$, one for each vector $y_j \in Y$.

We observe that pattern occurrences in G_W have some useful combinatorial properties. The lemma below is an immediate observation, which follows from the fact that each $G_W^{(j)}$ is acyclic and not connected to any other $G_W^{(j')}$.

▶ **Lemma 3.** *If subpattern* $\texttt{b}P_{x_i}\texttt{e}$ *has a match in* G_W *then the nodes matching* P_{x_i} *share the same* j *coordinate and have distinct and consecutive* h *coordinates.*

Lemma 4 (whose proof will be provided in the extended version of this work) relates the occurrence of a subpattern to the OV problem.

[2] Note that `1` is a symbol of Σ while 1 is the truth value in x_i.

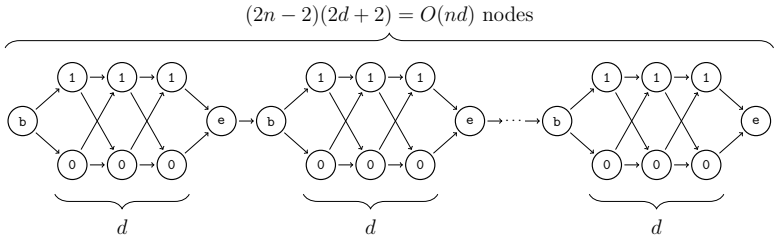

Figure 2 Gadget G_U.

▶ **Lemma 4.** *Subpattern* $\texttt{b}P_{x_i}\texttt{e}$ *has a match in* G_W *if and only if there exist* $y_j \in Y$ *such that* $x_i \cdot y_j = 0$.

In the following we will also use gadget $G_U = (V_U, E_U, L_U)$, the degenerate case of G_W with $2n-2$ (instead of just n) connected components $G_U^{(j)}$ where, for all $1 \leq j \leq 2n-2$ and $1 \leq h \leq d$, we place both a `0`-node and a `1`-node: we call these two nodes u_{jh}^0 and u_{jh}^1, respectively, to distinguish them from those in G_W. Moreover, every `e`-node of this gadget is connected with the next `b`-node, in terms of j coordinate (see Figure 2). As it can be seen, any subpattern P_{x_i} occurs in G_U, so it can be used as a "jolly" gadget.

2.3 Non-deterministic Graph

A possible approach is based on suitably combining one instance of gadget G_W and two instances of gadgets G_U, named G_{U1} and G_{U2}. The idea is that, when $x_i \cdot y_j = 0$, we want P to occur in G, so that the three conditions below hold.

- Instance G_{U1}: P_{x_1} occurs in $G_{U1}^{(n-1+j-(i-1))}$, …, $P_{x_{i-1}}$ occurs in $G_{U1}^{(n-1+j-1)}$.
- Instance G_W: P_{x_i} occurs in $G_W^{(j)}$.
- Instance G_{U2}: $P_{x_{i+1}}$ occurs in $G_{U2}^{(j)}$, …, P_{x_n} occurs in $G_{U2}^{(j+n-i-1)}$.

On the other hand, when $x_i \cdot y_j \neq 0$, we do not want P_{x_i} to occur in $G_W^{(j)}$. We can suitably link the instances G_W, G_{U1} and G_{U2} so that we get the above conditions: we connect the `e`-nodes in G_{U1} to `b`-nodes in G_W, the `e`-nodes in G_W to `b`-nodes in G_{U2} and we place additional starting `b`-nodes and additional ending `e`-nodes, to properly match the `bb` and `ee` prefix and suffix of P, respectively. However, even if G_W, G_{U1} and G_{U2} are deterministic, their resulting composition is not so, because of the out-neighbours of the `e`-nodes.[3] We show below how to obtain a deterministic graph by suitably merging G_W with portions of G_U.

2.4 Deterministic Graph

In order to obtain a *deterministic* DAG, we need to suitably combine one instance of gadget G_W with the two instances G_{U1} and G_{U2} (recall that both G_{U1} and G_{U2} have instances of gadget $G_U^{(j)}$, for all $1 \leq j \leq 2n-2$). While G_{U2} will be used as is, G_{U1} needs to be partially merged with G_W to obtain determinism. We start building our final graph G from G_W by adding parts of G_{U1} when needed, obtaining a deterministic graph called G_{U1W}, as shown in Figure 3. Consider subgraph $G_W^{(j)}$ and assume that the first position in which the `1`-node is lacking is h. We place a partial version of subgraph $G_{U1}^{(j')}, j' := n-1+j$, by adding to the graph the nodes and edges of $G_{U1}^{(j')}$ that are located between position $h+1$ and node

[3] An `e`-node can have two `b`-nodes as out-neighbors when linking G_{U1} to G_W, see [15].

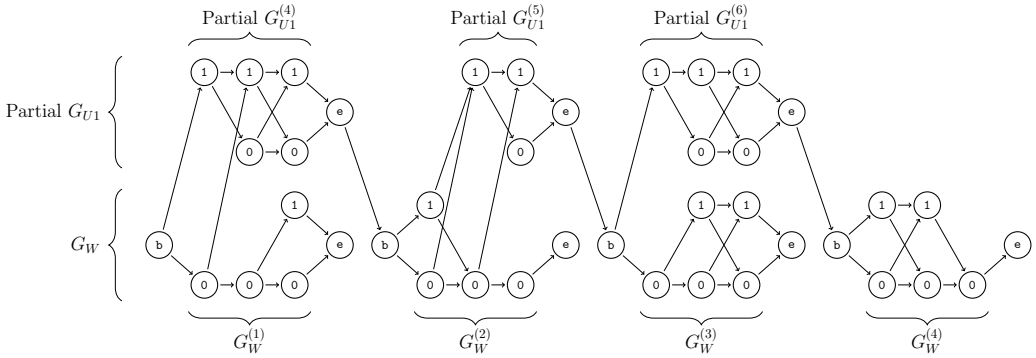

Figure 3 Graph G_{U1W} after merging G_{U1} (from Figure 2) with G_W (from Figure 1).

$e_{U1}^{(j')}$ (included). If $h = d$ we place only node $e_{U1}^{(j')}$. We also place 1-node u_{jh}^1 and we connect the 0-node and the 1-node (if any) of $G_W^{(j)}$ in position $h-1$ to it (if $h > 1$), or we connect $b_W^{(j)}$ to it (if $h = 1$). Moreover, we connect node u_{jh}^1 to the first 0- and 1-node of partial $G_{U1}^{(j')}$. If $h = d$ we connect u_{jh}^1 to $e_{U1}^{(j')}$. Then we scan $G_W^{(j)}$ from left to right looking for those positions h', $h \leq h' < d$, such that there is no 1-node in position $h'+1$. We connect the 0-node and the 1-node (if any) of $G_W^{(j)}$ in position h' to the 1-node of $G_{U1}^{(j')}$ in position $h'+1$. Finally, we place edge $(e_{U1}^{(j')}, b_W^{(j+1)})$. To complete the merging task, we apply the above modification to all $G_W^{(j)}$, for $1 \leq j \leq n-1$, and thus obtain gadget G_{U1W}.

At this point, we place gadget G_{U2} and we connect G_{U1W} to it by placing edges $(e_W^{(j)}, b_{U2}^{(j)})$, for all $1 \leq j \leq n$. Also, for every b-node of G_{U1W} we place an additional b-node as in-neighbor. We do the same for every e-node of G_{U2}, placing an e-node as out-neighbor. Adding subgraphs $G_{U1}^{(1)}, \ldots, G_{U1}^{(n-1)}$ with one additional b-node as in-neighbor of their b-nodes, and connecting the e-node of $G_{U1}^{(n-1)}$ to the b-node of $G_W^{(1)}$, completes the transformation into the wanted deterministic directed acyclic graph, which we call G. Figure 4 gives an overall picture of G.

It is easy to verify that every b- and e-node in G can have no more than two out-neighbours and, in such case, they have different labels. This shows that graph G is deterministic.

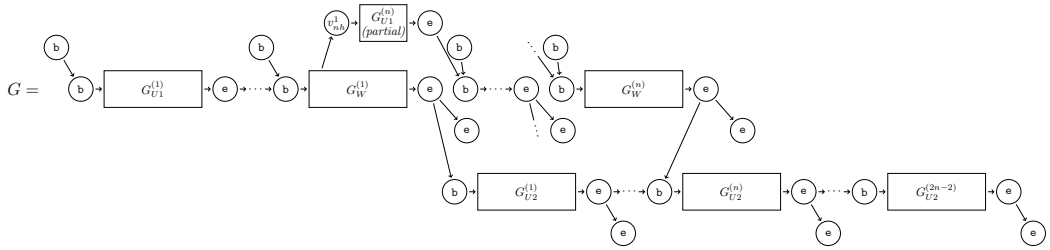

Figure 4 Final deterministic DAG G. In such graph there are $n-1$ instances of $G_{U1}^{(j)}$, $n-1$ partial instances of $G_{U1}^{(j)}$, n instances of $G_W^{(j)}$ and $2n-2$ instances of $G_{U2}^{(j)}$.

The deterministic DAG G has a crucial property which, combined with Lemma 3 and Lemma 4, is key to ensure the correctness of our reduction.

▶ **Lemma 5.** *Pattern P has a match in G if and only if a subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ of P has a match in the underlying subgraph G_W of G_{U1W}.*

Proof. For the ($\Rightarrow$) implication, because of the directed eb-edges, each distinct subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ matches a path from either a distinct portion of G_{U1W} (or from the $G_{U1}^{(j)}$ subgraphs, $1 \leq j \leq n-1$, before it) or G_{U2}. Moreover, each occurrence of P must begin with bb and end with ee. String bb can be matched only in G_{U1W} (or in the $G_{U1}^{(j)}$ subgraphs before it), hence the match must start here. On the other hand, string ee is found either in G_{U1W} or in G_{U2}. Observe that, by construction, once a match for pattern P is started in G_{U1W} (or in the $G_{U1}^{(j)}$ subgraphs before it), the only way to successfully conclude it is either by matching ee within G_{U1W}, or by matching also a portion of G_{U2} and then ee. Because of the structure of the graph, in both cases a subpattern $\mathtt{b}P_{x_i}\mathtt{e}$ of P must match one of the subgraphs $G_W^{(j)}$ that are present in G_{U1W}.

The ($\Leftarrow$) implication is trivial. In fact, if $\mathtt{b}P_{x_i}\mathtt{e}$ has a match in one subgraph $G_W^{(j)}$, then by construction we can match $\mathtt{b}P_{x_1}\mathtt{e}\ldots\mathtt{b}P_{x_{i-1}}\mathtt{e}$ possibly in the $G_{U1}^{(j)}$ subgraphs before G_{U1W}, then possibly in the partial $G_{U1}^{(j)}$ subgraphs of G_{U1W}. We can then match $\mathtt{b}P_{x_{i+1}}\mathtt{e}\ldots\mathtt{b}P_{x_n}\mathtt{e}$ in G_{U2}, and thus have a full match for P in G. ◀

We are now ready to prove our main result.

Proof of Theorem 1. First, we prove that the reduction is correct. Then we analyze its cost and show how a subquadratic-time algorithm for SMLG would contradict the OV hypothesis. Then, we explain how the graph can be modified to become a 3-DDAG using a binary alphabet.

Correctness. We need to ensure that pattern P has a match in G if and only if there exist vectors $x_i \in X$ and $y_j \in Y$ which are orthogonal. This follows from Lemma 5, which guarantees that P has a match in G if and only if a subpattern P_{x_i} has a match in G_W, and the fact that, by Lemma 4, this holds if and only if $x_i \cdot y_j = 0$.

Cost. As observed during the construction in Section 2.1 and Section 2.2, both pattern P and graph G have size $O(nd)$. Indeed, for each one of the n vectors $x_i \in X$ we place in P characters b and e plus d characters that can be either 0 or 1. In graph G, the size of each subgraph is proportional to the dimension d of the vectors and we place $O(n)$ of them.

Using the OV hypothesis. The last step is to show that any $O(|E|^{1-\epsilon}\, m)$-time or $O(|E|\, m^{1-\epsilon})$-time algorithm A for SMLG contradicts the OV hypothesis. Given two sets of vectors X and Y, we can perform our reduction obtaining pattern P and graph G in $O(nd)$ time, by observing that $|E| = O(nd)$ and $m = O(nd)$. No matter whether A has $O(|E|^{1-\epsilon}m)$ or $O(|E|\, m^{1-\epsilon})$ time complexity, we will end up with an algorithm deciding if there exists a pair of orthogonal vectors between X and Y in $O(nd \cdot (nd)^{1-\epsilon}) = O(n^{2-\epsilon}\mathrm{poly}(d))$ time, which contradicts the OV hypothesis.

Maximum sum of indegree and outdegree 3. Observe that every node in G can have at most 2 in-neighbours and 2 out-neighbours. An emblematic case is that of four nodes, say v, w, v', and w', with edges (v,w), (v,w'),(v',w), and (v',w'). To reduce to 1 the outdegree of v and v', and the indegree of w and w', the idea is to add two dummy nodes $\bar{v}$ and $\bar{w}$ connected by an edge $(\bar{v},\bar{w})$, and then replace the four edges above with $(v,\bar{v})$, $(v',\bar{v})$, $(\bar{w},w)$, and $(\bar{w},w')$. The dummy nodes can be labeled e.g. with 0 and then one can do a symmetric modification in the pattern. One needs to apply such transformations between any two consecutive columns of G.

Alphabet size. Our alphabet is of size 4. One can reduce the alphabet size to binary using the encoding $\alpha(\mathtt{0}) = \mathtt{0000}$, $\alpha(\mathtt{1}) = \mathtt{1111}$, $\alpha(\mathtt{b}) = \mathtt{10}$, and $\alpha(\mathtt{e}) = \mathtt{01}$ for both the pattern and the graph. (That is, we replace each σ-node with a path of as many nodes as characters in $\alpha(\sigma)$.) Observe that, after small adjustments that do not weaken our results, there is a bijection from matches before and after applying the encoding. ◀

As discussed in the Introduction, G has both nodes of indegree at most 1 (and outdegree more than 1), and nodes of outdegree at most 1 (and indegree more than 1). In the extended version of this work we will give a linear-time algorithm for directed graphs with nodes of only one such type.

3 Undirected Graphs: Zig-zag Matching

The lower bound given for the SMLG problem can cover the special case of an undirected graph with maximum degree 2. To this end, we need to modify the reduction defining a new alphabet, pattern and graph. The original alphabet $\Sigma = \{\mathtt{b}, \mathtt{e}, \mathtt{0}, \mathtt{1}\}$ is replaced with $\Sigma' = \{\mathtt{b}, \mathtt{e}, \mathtt{A}, \mathtt{B}, \mathtt{s}, \mathtt{t}\}$. Characters `1` and `0` are encoded in the following manner:

$\mathtt{1} = \mathtt{ABA}$ and $\mathtt{0} = \mathtt{ABABABA}$.

When such encoding is applied, character `s` will be used as a separator marking the beginning and the end of the old characters. As an example, the subpattern

$P_{x_i} = \mathtt{1\ 0\ 1}$ will be encoded as $P'_{x_i} = \mathtt{s\ ABA\ s\ ABABABA\ s\ ABA\ s}$.

A new pattern P' is built applying this encoding to each one of the subpatterns P_{x_i}, thus obtaining new subpatterns P'_{x_i}. We then concatenate all the subpatterns P'_{x_i} by placing the new character `t` to separate them, instead of `eb`. Finally, we place characters `bt` at the beginning of the new pattern, and `te` at the end. Here follows an example:

$P = \mathtt{bb\ 100\ e\ b\ 101\ ee}$

$$P' = \mathtt{b\ \ t\ s\ \ \underset{1}{ABA}\ \ s\ \ \underset{0}{ABABABA}\ \ s\ \ \underset{0}{ABABABA}\ \ s\ \ t\ s\ \ \underset{1}{ABA}\ \ s\ \ \underset{0}{ABABABA}\ \ s\ \ \underset{1}{ABA}\ \ s\ \ t\ e}$$

Note that for each subpattern we are introducing a constant number of new characters, hence the size of the entire pattern P' still is $O(nd)$.

An analogous encoding will be applied to the graph. The strategy is to encode G_W in an undirected path by concatenating subpaths representing each $G_W^{(j)}$, one after another.

The positions h in which both a `0`- and a `1`-node are present in $G_W^{(j)}$ are replaced by a path that can be matched both by $\mathtt{0} = \mathtt{ABABABA}$ and $\mathtt{1} = \mathtt{ABA}$. Positions h with only a `0`-node and no `1`-node are encoded instead with a path that can be matched only by $\mathtt{0} = \mathtt{ABABABA}$ (see Figure 5). We use `s`-nodes to separate these paths. We denote by $LG_W^{(j)}$ (*Linear* $G_W^{(j)}$) this linearized version of $G_W^{(j)}$. Moreover, given subgraph $G_W^{(j)}$, two new `t`-nodes will mark the beginning and the ending of its encoding. Figure 6 illustrates this transformation for $G_W^{(j)}$.

In a similar manner, G_U is also encoded as a path. We do not need to encode all its $2n-2$ subgraphs: since the matching path can go through nodes more than once, we only need to encode one of these subgraphs, in the same manner as done for $G_W^{(j)}$. Let LG_U be the linearized version of only one of the "jolly" gadgets that were composing the original G_U.

Then, for each $1 \leq j \leq n$, we build structure $LG^{(j)}$ by placing `t`-nodes, LG_U instances, $LG_W^{(j)}$, a `b`-node on the left and an `e`-node on the right, as in Figure 7. In such structure the `b`-node and the `e`-node delimit the beginning and the end of a viable match for a pattern. The `t`-nodes are separating the LG_U structures from $LG_W^{(j)}$ and, in general, they are marking the beginning and the end of a match for a subpattern P'_{x_i}. The idea behind $LG^{(j)}$ is that a match of P can traverse LG_U from beginning to end, backwards and forwards as many

(a) (b)

Figure 5 New substructures. (a) The old substructure is replaced by an undirected path that can match either `sABAs` (which represents `1`) by going forward only, or `sABABABAs` (which represents `0`), by going forward, backward, and forward again. (b) The an undirected path replacing a `0`-node can match only the string `sABABABAs`.

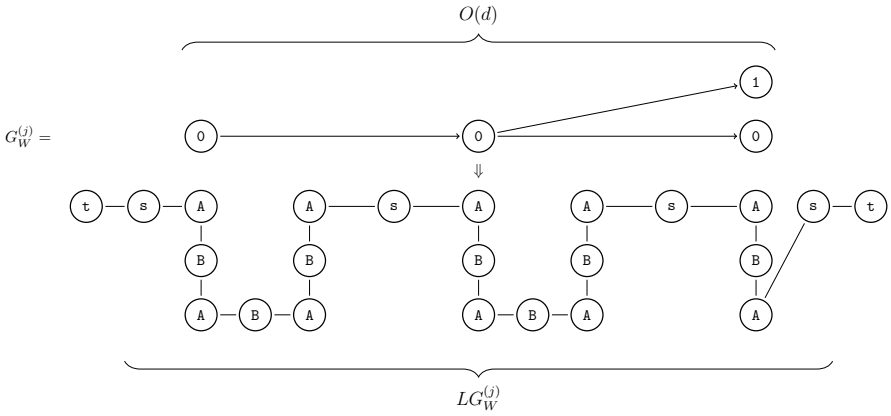

Figure 6 A subgraph $G_W^{(j)}$ is converted into a linear structure $LG_W^{(j)}$ using `s` as separator.

times as needed, before starting a match of some subpattern P'_{x_i} inside $LG_W^{(j)}$. Notice also that this allows only subpatterns on even positions i to match inside $LG_W^{(j)}$. We will address this minor issue at the end (see page 12).

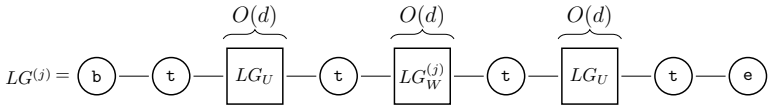

Figure 7 The $LG_W^{(j)}$ structure surrounded by two instances of LG_U. The `t`-nodes establish the beginning and the end of a match for a subpattern $\texttt{t}P'_{x_i}\texttt{t}$ while the `b`- and `e`-nodes are the starting and ending point for a match of the whole pattern P'.

In order to construct the final graph LG we concatenate all $LG^{(1)}$, $LG^{(2)}$, ..., $LG^{(n)}$ into a single undirected path. Figure 8 gives a picture of the end result.

No issues arise regarding the size of the graph, since we are replacing every `0`-node, or every pair of a `0`-node and a `1`-node, with a constant number of new nodes. By construction, the two gadgets LG_U and LG_W both have size $O(d)$, since for each one of the d entries of a vector we place one of the two possible encodings. In LG there are n instances of $LG_W^{(j)}$, each one surrounded by two LG_U instances. Hence the total size of the graph remains $O(nd)$.

In order to prove the correctness of the reduction, we will show some properties on LG by introducing the following lemmas, whose formal proofs will be available in the extended version of this work. We use $t_l LG_W^{(j)} t_r$ to refer to $LG_W^{(j)}$ extended with the `t`-nodes on its left and on its right. When referring to the k-th `s`-character in P'_{x_i} we mean the k-th `s`-character found scanning P'_{x_i} from left to right; in the same manner we refer to the k-th `s`-node in $LG_W^{(j)}$.

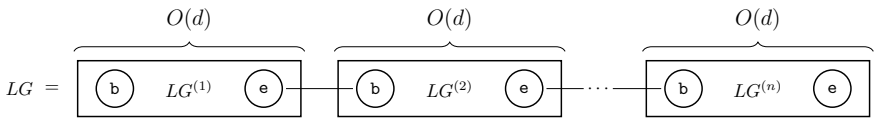

Figure 8 The final graph LG.

▶ **Lemma 6.** *If subpattern* $\mathtt{t}P'_{x_i}\mathtt{t}$ *has a match in* $t_l LG_W^{(j)} t_r$ *starting at* t_l *and ending at* t_r*, then the* k*-th* `s`*-character in* P'_{x_i} *matches the* k*-th* `s`*-node in* $LG_W^{(j)}$*, for all* $1 \leq k \leq d+1$*.*

▶ **Lemma 7.** *Subpattern* $\mathtt{t}P'_{x_i}\mathtt{t}$ *has a match in* $t_l LG_W^{(j)} t_r$ *starting at* t_l *and ending at* t_r *if and only if there exist* $y_j \in Y$ *such that* $x_i \cdot y_j = 0$*.*

The main difference with the original proof resides in assuming that a match for P'_{x_i} starts at t_l and ends at t_r. This feature is crucial for the correctness of the reduction and can be safely exploited since, as shown in the following, the `b`- and `e`-nodes guarantee that in case of a match for P' we will cross the $LG_W^{(j)}$ gadget from left to right at least once.

▶ **Lemma 8.** *Pattern* P' *has a match in* LG *if and only if there exist* i *and* j *such that* i *is even and subpattern* $\mathtt{t}P'_{x_i}\mathtt{t}$ *has a match in* $t_l LG_W^{(j)} t_r$ *starting at* t_l *and ending at* t_r*.*

Proof. For the ($\Rightarrow$) implication, first observe that the `b`- and `e`-nodes in LG are forcing a direction to follow. Let $LG_{Ul}^{(j)}$ and $LG_{Ur}^{(j)}$ be the LG_U gadgets to the left and to the right of $LG_W^{(j)}$, respectively. Since pattern P' starts with a `b` and ends with an `e`, a match can only start at the `b`-node on the left of $LG_{Ul}^{(j)}$ and end at the `e`-node on the right of $LG_{Ur}^{(j)}$, for some j. Hence $LG_W^{(j)}$ needs to be crossed by a match from left to right at least once. Thus, there must exist a subpattern $\mathtt{t}P'_{x_i}\mathtt{t}$ that has a match starting at t_l and ending at t_r. For such a pattern Lemma 7 applies. Moreover, because of our construction, only a subpattern on even position can achieve such a match.

The ($\Leftarrow$) implication is immediate since given a subpattern $\mathtt{t}P'_{x_i}\mathtt{t}$ which has a match in $t_l LG_U^{(j)} t_r$ one can match $\mathtt{bt}P'_{x_1}\mathtt{t}\ldots\mathtt{t}P'_{x_{i-1}}\mathtt{t}$ in $LG_{Ul}^{(j)}$ and $\mathtt{t}P'_{x_{i+1}}\mathtt{t}\ldots\mathtt{t}P'_{x_n}\mathtt{te}$ in $LG_{Ur}^{(j)}$ and have a full match for P' in LG. ◀

Since Lemma 8 gives us a property which holds only if a subpattern is in even position, we need to tweak pattern P' to make the reduction work. Indeed, we define two patterns. The first pattern $P'^{(1)}$ is P' itself; the second pattern $P'^{(2)}$ is obtained by swapping the subpatterns P'_{x_i} on odd position with the next subpatterns $P'_{x_{i+1}}$ on even position, for every $i = 1, 3, \ldots$. For example, if n is even, we will have:

$$P'^{(1)} = \mathtt{bt}\ P'_{x_1}\ \mathtt{t}\ P'_{x_2}\ \mathtt{t}\ P'_{x_3}\ \mathtt{t}\ P'_{x_4}\ \mathtt{t}\ \ldots\ \mathtt{t}\ P'_{x_{n-1}}\ \mathtt{t}\ P'_{x_n}\ \mathtt{te} = P'$$
$$P'^{(2)} = \mathtt{bt}\ P'_{x_2}\ \mathtt{t}\ P'_{x_1}\ \mathtt{t}\ P'_{x_4}\ \mathtt{t}\ P'_{x_3}\ \mathtt{t}\ \ldots\ \mathtt{t}\ P'_{x_n}\ \mathtt{t}\ P'_{x_{n-1}}\ \mathtt{te}$$

While $P'^{(1)}$ checks the even positions of P', $P'^{(2)}$ checks the odd ones. If n is even then the last subpattern would not have the chance to be matched against any $G_W^{(j)}$. In such case we can simply add a dummy subpattern $\bar{P} = \mathtt{s\ ABA\ s\ ABA\ s}\ldots\mathtt{s\ ABA\ s}$ (with d repetitions of `ABA`) at the end of P as it were its last subpattern, so that the number of subpatterns becomes odd. Indeed, observe that $\bar{P}$ corresponds to vector $\bar{x} = (11\ldots1)$, which has null product only with vector $\bar{y} = (00\ldots0)$. Hence if $\bar{y} \notin Y$ then $\bar{P}$ does not have a match in any $LG^{(j)}$, while if $\bar{y} \in Y$ every subpattern P'_{x_i} has a match in the $LG^{(j)}$ built on top of $\bar{y}$. This means that $\bar{P}$ does not disrupt our reduction.

Now we are ready to present the end result.

▶ **Lemma 9.** *Either $P'^{(1)}$ or $P'^{(2)}$ has a match in LG if and only if there exist vectors $x_i \in X$ and $y_j \in Y$ which are orthogonal.*

Proof. For ($\Rightarrow$) we assume that either $P'^{(1)}$ or $P'^{(2)}$ have a match in LG. By Lemma 8 this means that there exists a subpattern $P'^{(q)}_{x_i}$, $q \in \{1, 2\}$ which has a match in $LG^{(j)}_W$, for some j. Lemma 7 then ensures that $x_i \cdot y_j = 0$, thus x_i and y_j are orthogonal. For the other implication ($\Leftarrow$) we assume that there exists two orthogonal vectors $x_i \in X$ and $y_j \in Y$. Thanks to Lemma 7 we find a subpattern P'_{x_i} matching $LG^{(j)}_W$. By construction, P'_{x_i} has to be in even position either in $P'^{(1)}$ or in $P'^{(2)}$. By Lemma 8 this means that either $P'^{(1)}$ or $P'^{(2)}$ has a match in LG. ◀

Theorem 2 follows directly from the correctness of these constructions, except for the alphabet size reduction to binary, which will be covered in the extended version of this work.

References

1. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.
2. Tatsuya Akutsu. A linear time pattern matching algorithm between a string and a tree. In *4th Symposium on Combinatorial Pattern Matching, Padova, Italy*, pages 1–10, 1993.
3. Jarno Alanko, Alberto Policriti, and Nicola Prezza. On Prefix-Sorting Finite Automata. *arXiv e-prints*, page arXiv:1902.01088, February 2019. arXiv:1902.01088.
4. Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. In *WADS'97, Halifax, LNCS 1272*, pages 160–173, 1997.
5. Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern Matching in Hypertext. *J. Algorithms*, 35(1):82–99, 2000.
6. Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1):1:1–1:39, February 2008. doi:10.1145/1322432.1322433.
7. Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM. doi:10.1145/2746539.2746612.
8. Arturs Backurs and Piotr Indyk. 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*, pages 457–466, 2016.
9. Arturs Backurs and Christos Tzamos. Improving Viterbi is Hard: Better Runtimes Imply Faster Clique Algorithms. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70, pages 311–321. PMLR, 2017.
10. Karl Bringmann and Marvin Kunnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, FOCS '15, pages 79–97, Washington, DC, USA, 2015. IEEE Computer Society. doi:10.1109/FOCS.2015.15.
11. The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018. doi:10.1093/bib/bbw089.
12. Alessio Conte, Gaspare Ferraro, Roberto Grossi, Andrea Marino, Kunihiko Sadakane, and Takeaki Uno. Node Similarity with q -Grams for Real-World Labeled Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1282–1291, 2018. doi:10.1145/3219819.3220085.

13 Mateus de Oliveira Oliveira and Michael Wehar. Intersection Non-emptiness and Hardness Within Polynomial Time. In *DLT*, volume 11088 of *Lecture Notes in Computer Science*, pages 282–290. Springer, 2018.

14 Massimo Equi, Roberto Grossi, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Binary Strings and Bounded Degree. *arXiv e-prints*, page arXiv:1901.05264, January 2019. arXiv:1901.05264.

15 Massimo Equi, Roberto Grossi, Alexandru I. Tomescu, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Determinism and Zig-Zag Matching. *arXiv e-prints*, page arXiv:1902.03560, February 2019. arXiv:1902.03560.

16 Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1433–1445, 2018. doi:10.1145/3183713.3190657.

17 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017. doi:10.1016/j.tcs.2017.06.016.

18 Daniel Gibney and Sharma V. Thankachan. On the Hardness and Inapproximability of Recognizing Wheeler Graphs. *arXiv e-prints*, page arXiv:1902.01960, February 2019. arXiv:1902.01960.

19 Shohei Hido and Hisashi Kashima. A Linear-Time Graph Kernel. In Wei Wang 0010, Hillol Kargupta, Sanjay Ranka, Philip S. Yu, and Xindong Wu, editors, *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, pages 179–188. IEEE Computer Society, 2009.

20 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

21 Chirag Jain, Haowen Zhang, Yu Gao, and Srinivas Aluru. On the Complexity of Sequence to Graph Alignment. *bioRxiv*, 2019. To appear in RECOMB 2019. doi:10.1101/522912.

22 Donald E. Knuth, James H. Morris, and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal on Computing*, 6(2):323–350, March 1977.

23 Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de Bruijn graphs. *BMC Bioinformatics*, 17:237, 2016. doi:10.1186/s12859-016-1103-9.

24 U. Manber and S. Wu. Approximate string matching with arbitrary costs for text and hypertext. In *IAPR Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland*, pages 22–33, 1992.

25 Gonzalo Navarro. Improved approximate pattern matching on hypertext. *Theoretical Computer Science*, 237(1-2):455–463, 2000.

26 K. Park and D. Kim. String matching in hypertext. In *6th Symposium on Combinatorial Pattern Matching, Espoo, Finland*, page 318, 1995.

27 Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF. World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, January 2008.

28 M. O. Rabin and D. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, April 1959. doi:10.1147/rd.32.0114.

29 Mikko Rautiainen and Tobias Marschall. Aligning sequences to general graphs in $O(V + mE)$ time. *bioRxiv*, pages 216–127, 2017.

30 Marko A. Rodriguez. The Gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages, Pittsburgh, PA, USA, October 25-30, 2015*, pages 1–10, 2015. doi:10.1145/2815072.2815073.

31 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing Graphs for Path Queries with Applications in Genome Research. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 11(2):375–388, March 2014. doi:10.1109/TCBB.2013.2297101.

32 Chris Thachuk. Indexing hypertext. *Journal of Discrete Algorithms*, 18:113–122, 2013. Selected papers from the 18th International Symposium on String Processing and Information Retrieval (SPIRE 2011). doi:10.1016/j.jda.2012.10.001.

33 Michael Wehar. Hardness Results for Intersection Non-Emptiness. In *ICALP (2)*, volume 8573 of *Lecture Notes in Computer Science*, pages 354–362. Springer, 2014.

34 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.