On the Complexity of Exact Pattern Matching in Graphs: Determinism and Zig-Zag Matching

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 pattern matching in labeled graphs is the problem of searching paths of a graph \( G = (V, E) \) that spell the same string as the given pattern \( P[1..m] \). This basic problem can be found at the heart of more complex operations on variation graphs in computational biology, query operations in graph databases, and analysis of heterogeneous networks, where the nodes of some paths must match a sequence of labels or types. In our recent work we described a conditional lower bound stating that the exact pattern matching problem in labeled graphs cannot be solved in less than quadratic time, namely, \( O(|E|^{1-\epsilon} m) \) time or \( O(|E|^\epsilon m^{1-\epsilon}) \) time for any constant \( \epsilon > 0 \), unless the Strong Exponential Time Hypothesis (SETH) is false. The result holds even if node labels and pattern \( P \) are drawn from a binary alphabet, and \( G \) is restricted to undirected graphs of maximum degree three or directed acyclic graphs of maximum sum of indegree and outdegree three. It was left open what happens on undirected graphs of maximum degree two, i.e., when the pattern can have a zig-zag match in a (cyclic) bidirectional string. Also, the reduction created a non-deterministic directed acyclic graph, and it was left open if determinism would make the problem easier. In this work, we show through the Orthogonal Vectors hypothesis (OV) that the same conditional lower bound holds even for these restricted cases.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Problems, reductions and completeness; Theory of computation → Pattern matching

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

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

1 Introduction

Large-scale labeled graphs are becoming ubiquitous in several areas, such as computational biology, graph databases, and graph mining. 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. We refer the reader to the introduction in Equi et al. [8] for a thorough review of these applications and references therein.

The Pattern Matching in Labeled Graphs (PMLG) problem is as follows. Given a labeled graph \( G = (V, E, L) \) and a pattern string \( P \) over the alphabet \( \Sigma \), where \( L : \rightarrow \Sigma^+ \) represents the labeling of the nodes with strings, we say that \( P \) occurs in \( G \) if there is a sequence of adjacent (and not necessarily distinct) nodes \( u_1, \ldots, u_k \) (where \( (u_i, u_{i+1}) \in E \) for \( 1 \leq i < k \))...
such that $P$ occurs as a substring in the string obtained by their label concatenation $L(u_1) \cdots L(u_k)$. Given a pattern $P$ and a labeled graph $G$ over $\Sigma$, the exact pattern matching problem on graphs requires to establish if $P$ occurs in $G$. Notice that allowing matching nodes not to be distinct is an essential property to make our problem of interest with respect to conditional hardness: With the requirement of distinctness, one can easily derive NP-hardness from the Hamiltonian path problem with a pattern consisting of $a^{|V|}$ and each node labeled with $a$. Generalization of such reductions are considered in [12].

Although there is a growing need to perform pattern matching on graphs, the idea of extending the problem of string searching in sequences to pattern matching in graphs was studied over 25 years ago as a search problem in hypertext [13]. Interestingly, the best bounds achieved for both exact pattern matching [3] and approximate pattern matching [15] in graphs are the same; both solutions take $O(N + m|E|)$ time, where $N = \sum_{u \in V} |L(u)|$ is the total length of text strings in all nodes, $m = |P|$ is the pattern length, and $|E|$ is the number of edges in $G$.

The quadratic cost of the approximate matching in graphs by Rautiainen and Marschall [15] is asymptotically optimal under the Strong Exponential Time Hypothesis (SETH) as (i) they solve the approximate string matching as a special case, since a graph consisting of just one 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 the Strong Exponential Time Hypothesis (SETH) is false [4]. Hence this conditional lower bound explains why the $O(m|E|)$ barrier has been difficult to cross.

In our recent work [5], we showed that exact and approximate pattern matching are equally hard on graphs under SETH. Namely, we showed the conditional lower bound that an $O(|E|^{1-\epsilon} m)$-time or an $O(|E| m^{1-\epsilon})$-time algorithm for exact pattern matching on graphs cannot be achieved unless SETH is false. This result explains why it has been difficult to find indexing schemes for graphs with other than best case or average case guarantees for fast exact pattern matching [16, 9].

Our reduction [5] covered a wide range of graphs and patterns: the reduction graph has maximum degree three, string and node labels are drawn from binary alphabet, and the graph is non-deterministic when interpreted as acyclic directed graph (DAG). While some special cases are already covered in literature (see conclusions in our earlier work [5]), there were two important cases left to complete the picture: 1) zig-zag matching in a (cyclic) bidirectional string (undirected graph of maximum degree two), and 2) matching on deterministic directed acyclic graphs. In this work, we give two new reductions that show that the same conditional lower bound holds even in these restricted cases.

Like our previous reduction [5], our new reductions share some similarities with those for string problems [1, 7, 1, 5, 6]; now we also inherit the use of the Orthogonal Vectors hypothesis (OV) to establish the connection with SETH, while earlier [5] we gave a direct reduction from SETH. The closest connection is with a conditional hardness on several forms of regular expression matching [5]. Our reduction has additional constructions to allow a pattern to match inside any type of graphs, while a non-deterministic finite automaton (NFA) built on top of those specific regular expressions would accept only certain sub-patterns. For zig-zag matching the reduction is more intricate as the underlying graph has less structure. To cover the deterministic case, the new reduction has a special design to enable local merging of graph substructures that avoids exponential growth. This part of the reduction is especially interesting, as converting an NFA into a deterministic finite automaton (DFA) can take exponential time [14], but this gap appears to have no significance with regards to hardness.
of exact pattern matching. This result complements recent findings about Wheeler graphs \cite{9, 10, 2}. Wheeler graphs are a class of graphs that admit an index structure supporting linear time exact pattern matching. Gibney and Thakachan \cite{10} show that it is NP-complete to recognize whether a (non-deterministic) DAG is a Wheeler graph. Alanko, Policriti, and Prezza \cite{2} give a linear time algorithm for recognizing whether a deterministic DAG is a Wheeler graph. Our result shows that converting an arbitrary deterministic DAG into an equivalent Wheeler graph should take quadratic time unless SETH fails.

2 Definitions

In this work we present two reductions from the Orthogonal Vectors (OV) problem to the two special cases, deterministic and zig-zag, of the pattern matching in labeled graphs (PMLG) problem. These reductions will eventually let us conclude that it is not possible to solve PMLG even in these special cases either in $O(|E|^{1-\epsilon}m)$ or $O(|E|m^{1-\epsilon})$ without contradicting the OV Hypothesis, complementing the earlier results \cite{5}. Contradiction with OV Hypotheses implies contradiction with SETH \cite{17}.

Before giving the formal definitions of the PMLG problem, we recall the OV Hypothesis. In the OV problem we are given two sets $X,Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$ and $d = \omega(\log n)$, and we are asked to determine whether or not there exist $x \in X$ and $y \in Y$ such that $x \cdot y = 0$, where $x \cdot y = \sum_{i=0}^{d} x[i] \cdot y[i]$. The OV Hypothesis \cite{17} states that for any constant $\epsilon > 0$, no algorithm can solve OV in $O(n^{2-\epsilon}\text{poly}(d))$ time. We can consider now the PMLG problem.

\textbf{Definition 1 (Labeled graph).} Given an alphabet $\Sigma$, a labeled graph $G$ is a triplet $(V,E,L)$ where $(V,E)$ is a directed or undirected graph and $L: V \mapsto \Sigma^+$ is a function that defines which string (i.e., label) over $\Sigma$ is assigned to each node.

If $G$ is a directed graph, then we say that $G$ is deterministic if for any node, it holds that the first symbol in the labels of any two out-neighbors is always different.

A node labeled with a character $\sigma \in \Sigma$ is called a $\sigma$-node. And edge whose endpoints have labels $\sigma_1$ and $\sigma_2$, respectively, is called a $\sigma_1\sigma_2$-edge.

\textbf{Definition 2 (Match).} Let $u_1, \ldots, u_j$ be a path in a labeled graph $G$ (paths are allowed to repeat nodes) and let $P$ be a pattern. Also, let $L(u)[l:]$ and $L(u)[:l]$ denote the suffix of $L(u)$ starting at position $l$, and the prefix of $L(u)$ ending at position $l'$, respectively. We say that $u_1, \ldots, u_j$ is a match for $P$ in $G$ with offset $l$ if the concatenation of the strings $L(u_1)[l:] \cdot L(u_2) \cdot \ldots \cdot L(u_{j-1}) \cdot L(u_j)[:l']$ equals $P$, for some $l'$.

\textbf{Problem 1 (Pattern Matching in Labeled Graphs (PMLG)).}

INPUT: A labeled graph $G = (V,E,L)$ and a pattern $P$, both over an alphabet $\Sigma$.

OUTPUT: All the matches for $P$ in $G$.

Before presenting our main result, we give a new proof regarding the general PMLG problem.

\textbf{Theorem 3.} For any constant $\epsilon > 0$, the Pattern Matching in Labeled Graphs (PMLG) problem for a binary alphabet cannot be solved in either $O(|E|^{1-\epsilon}m)$ or $O(|E|m^{1-\epsilon})$ time unless the OV Hypothesis fails.

An analogous statement was proven in our earlier work \cite{5} with respect to SETH. Although that reduction \cite{5} can easily be modified to use OV instead, converting it to cover the deterministic case appears not to be easy. Therefore we give a new reduction that can be strengthened to yield the following result.
Theorem 4. The conditional lower bound stated in Theorem 3 holds even if it is restricted to labeled deterministic directed acyclic graphs (DAGs) with maximum sum of outdegree and indegree three.

The new reduction does not cover the other special case — zig-zag matching in bidirectional string. For this we give another reduction and prove the following result.

Theorem 5. The conditional lower bound stated in Theorem 3 holds even if it is restricted to undirected graphs of maximum degree two, where the pattern and the node labels are drawn from an alphabet of size at least 6.

3 Undirected Graphs

Given an instance of OV with sets X and Y, we will construct a pattern P and a graph G such that P will have a match in G if and only if there exist a pair of orthogonal vectors between X and Y. We first describe how to build the pattern P and then how to obtain the graph G.

3.1 Pattern

Pattern P is defined over the alphabet \( \Sigma = \{b, e, 0, 1\} \), using the first set of vectors \( X = \{x_1, \ldots, x_n\} \), as the concatenation \( P = bbP_{x_1}e bP_{x_2}e \ldots bP_{x_n}ee \), where for all \( h \in \{1, \ldots, d\} \), the \( h \)-th symbol of string \( P_{x_i} \) is either 0 or 1, such that \( P_{x_i}[h] = 1 \) if and only of \( x_i[h] = 1 \). We thus view the vectors in X as sub-patterns which are concatenated placing separator characters. Note that the pattern starts with two consecutive characters b and ends with two consecutive characters e. Such strings are found nowhere else in the pattern, thus they mark its beginning and its end.

3.2 Graph

The gadget implementing the main logic of the reduction is an undirected graph \( GW = (V_W, E_W, L_W) \), illustrated in Figure 1 and defined as follows, starting from the second set of vectors \( Y = \{y_1, \ldots, y_n\} \). Set \( V_W \) can be seen as \( n \) conceptual groups of nodes

\[
Y = \{y_1, y_2, y_3, y_4\} = \{(110), (011), (100), (001)\}
\]

![Figure 1 Gadget \( GW \).](image-url)

\( V_W^{(1)}, V_W^{(2)}, \ldots, V_W^{(n)} \), one for each vector in \( Y \), organized in \( n \) sub-structures \( G_W^{(1)} = (V_W^{(1)}, E_W^{(1)}) \), \( \ldots \), \( G_W^{(n)} = (V_W^{(n)}, E_W^{(n)}) \). See Figure 1 for an example.
The idea is to make $G^{(j)}_W$ accept some sub-pattern $P_{x_j}$ if and only if $x_i \cdot y_j = 0$. When building $V^{(j)}_W$ we scan every entry $y_j[h]$ of vector $y_j \in Y$, for $1 \leq h \leq d$. If $y_j[h] = 1$ then vector $x_i$ must be accepted if and only if $P_{x_i}[h] = 0$, hence we place only a 0-node $v^0_{jh}$ (i.e., labeled with 0). Instead, if $y_j[h] = 0$, the value of vector $x_i$ in position $h$ does not matter and we place both a 0-node $v^0_{jh}$ and a 1-node $v^1_{jh}$ (i.e., labeled with 0 and 1, respectively).

For each $y_j \in Y$, set $V_W$ also contains some special nodes: a begin b-node $b^{(j)}_W$ and an end e-node $e^{(j)}_W$ (i.e., labeled with b and e, respectively). Formally, we have:

$$V^{(j)}_W = \{v^0_{jh} \mid y_j \in Y, 1 \leq h \leq d\} \cup \{v^1_{jh} \mid y_j[h] = 0, y_j \in Y, 1 \leq h \leq d\},$$

$$V_W = \bigcup_{j=1}^{n} \left( V^{(j)}_W \cup \{b^{(j)}_W, e^{(j)}_W\} \right).$$

The edges in set $E_W$ properly connect both the nodes inside each group $V^{(j)}_W$ and the groups themselves with each other. In group $V^{(j)}_W$, node $b^{(j)}_W$ is connected with $v^0_{j1}$, and, if present, with $v^1_{j1}$. Also, we place edges connecting nodes $v^0_{jd}$ and $v^1_{jd}$ (if present) with node $e^{(j)}_W$. Moreover, there is an edge for every pair of nodes that share the same $j$ and are consecutive in terms of $h$ coordinate (e.g., $v^0_{jh}, v^0_{j+1h}$), for $1 \leq j \leq n$ and $1 \leq h \leq d$. When considering a single $j$, the edges described so far form set $E^{(j)}_W$, namely the edges connecting the nodes in group $V^{(j)}_W$. In this way we obtain sub-structure $G^{(j)}_W = (V^{(j)}_W, E^{(j)}_W)$. We combine such sub-structures together by connecting each $G^{(j)}_W$ to its predecessor via edge $(e^{(j-1)}_W, b^{(j)}_W)$, for all $2 \leq j \leq n$. Thus, the final set of edges is:

$$E^{(j)}_W = \{v^0_{jh}, v^0_{j+1h} \mid v^0_{jh}, v^0_{j+1h} \in V_W\} \cup \{v^0_{jh}, v^1_{j+1h} \mid v^0_{jh}, v^1_{j+1h} \in V_W\} \cup \{v^1_{jh}, v^0_{j+1h} \mid v^1_{jh}, v^0_{j+1h} \in V_W\} \cup \{v^1_{jh}, v^1_{j+1h} \mid v^1_{jh}, v^1_{j+1h} \in V_W\},$$

$$E_W = \bigcup_{j=1}^{n} E^{(j)}_W \cup \{b^{(j)}_W, v^0_{j1} \mid b^{(j)}_W, v^0_{j1} \in V_W\} \cup \{b^{(j)}_W, v^1_{j1} \mid b^{(j)}_W, v^1_{j1} \in V_W\} \cup \{v^0_{jd}, e^{(j)}_W \mid v^0_{jd}, e^{(j)}_W \in V_W\} \cup \{v^1_{jd}, e^{(j)}_W \mid v^1_{jd}, e^{(j)}_W \in V_W\} \cup \{e^{(j)}_W, b^{(j+1)}_W \mid e^{(j)}_W, b^{(j+1)}_W \in V_W\}.$$
On the Complexity of Exact Pattern Matching in Graphs

has \(d + 2\) characters and the shortest path from \(b^{(j)}_W\) to \(e^{(j)}_W\) is \(d + 2\) nodes long (\(b^{(j)}_W\) and \(e^{(j)}_W\) included). Thus, \(bP_x, e\) has not enough characters to reach any \(e^{(j')}_{W}\), for \(j' > j\). Moreover, if one node in the path from \(b^{(j)}_W\) to \(e^{(j)}_W\) is matched twice, sub-pattern \(bP_x, e\) has not enough characters to reach \(e^{(j)}_W\). Finally, if one of the matching nodes were not consecutive in terms of \(h\) coordinate, by construction we know that we would not be following the shortest path to \(e^{(j)}_W\) hence it would not be possible to complete the match. Therefore, every matching node lies on a simple path from \(b^{(j)}_W\) to \(e^{(j)}_W\), and has distinct and consecutive \(h\) coordinates. ▶

Lemma 7. Sub-pattern \(bP_x, e\) has a match in \(G_W\) if and only if there exist \(y_j \in Y\) such that \(x_i \cdot y_j = 0\).

Proof. Recall that, by construction, \(v^0_{j\theta} \in V^{(j)}_i W\) and \(v^1_{j\theta} \in V^{(j)}_i W\) hold for those \(h\) such that \(y_j[h] = 0\), while \(v^0_{j\theta} \in V^{(j)}_i W\) and \(v^1_{j\theta} \in V^{(j)}_i W\) hold in case \(y_j[h] = 1\). We handle the two implications of the statement individually.

(⇒) By Lemma 3 we can focus on the \(d\) distinct and consecutive nodes of \(G^{(j)}_W\) that match \(P_x, e\). In particular we know that each character \(P_x, e\) is matched by either \(v^0_{j\theta}\) or \(v^1_{j\theta}\). Consider vectors \(x_i \in X\) and \(y_j \in Y\). If \(P_x, e\) has a match in \(G^{(j)}_W\) it means that node \(v^1_{j\theta}\) exists and hence \(y_j[h] = 0\), implying \(x_i[h] \cdot y_j[h] = 0\). If \(P_x, e\) has \(0\), by construction we know that \(x_i[h] = 0\) and, no matter the type of match in \(G^{(j)}_W\), it clearly holds that \(x_i[h] \cdot y_j[h] = 0\).

At this point, we can conclude that \(x_i[h] \cdot y_j[h] = 0\) for every \(1 \leq h \leq d\), thus \(x_i \cdot y_j = 0\).

(⇐) Consider vectors \(x_i \in X\) and \(y_j \in Y\) that are such that \(x_i \cdot y_j = 0\). For \(h = 1, 2, \ldots\), if \(y_j[h] = 0\) then \(v^0_{j\theta}, v^1_{j\theta} \in V^{(j)}_i W\) and \(P_x, e\) can match either \(v^0_{j\theta}\) or \(v^1_{j\theta}\) in \(G^{(j)}_W\). If \(y_j[h] = 1\) it must be \(x_i[h] = 0\) since \(x_i \cdot y_j = 0\), thus \(P_x, e\) can match node \(v^0_{j\theta}\), which is always present in \(G^{(j)}_W\). Finally, characters \(b\) and \(e\) can match nodes \(b^{(j)}_W\) and \(e^{(j)}_W\), respectively. All characters of \(bP_x, e\) have now a matching node and the definition of the edges in \(E_W\) allows to visit all such nodes via a matching path starting at \(b^{(j)}_W\) and ending at \(e^{(j)}_W\). ▶

While the previous gadget is useful to check whether a vector \(x_i \in X\) is orthogonal to the ones in \(Y\) using a given sub-pattern \(P_x, e\) in \(b[0, 1]^d\), we need another “jolly” gadget that matches all sub-patterns in \(b[0, 1]^d\) (this is useful when \(x_i\) is not orthogonal to any \(y_j \in Y\).

We will concatenate \(2(n - 1) = 2n - 2\) instances of this “jolly” gadget, thus obtaining the graph \(G_U = (V_U, E_U, L_U)\) illustrated in Figure 3. The reason for using this specific number will be clear when we will describe how to combine this gadget with \(G_W\).

The \(j\)-th copy of the “jolly” gadget \(G^{(j)}_{U}\) has a \(b\)-node \(b^{(j)}_{U}\) followed by 0- and 1-nodes \(v^0_{j\theta}, v^1_{j\theta}\) and then \(e\)-node \(e^{(j)}_{U}\), with \(1 \leq j \leq n - 1\) and \(1 \leq h \leq d\). We place the edges \((b^{(j)}_{U}, v^0_{j1}), (b_{j}, v^0_{j1}), (v^1_{jd}, e^{(j)}_{U}), (v^1_{jd}, e^{(j)}_{U})\) for connecting the beginning and ending nodes of each gadget with its inner part. We connect nodes \(v^0_{j\theta}\) and \(v^1_{j\theta}\) with the edges \((v^0_{j\theta}, v^0_{j+h\theta}), (v^0_{j\theta}, v^0_{j+h\theta}), (v^1_{j\theta}, v^1_{j+h\theta}), (v^1_{j\theta}, v^1_{j+h\theta})\). In the inner part, we add all edges \((e^{(j)}_{U}, b^{(j)}_{U+1}),\) for all \(j = 1, \ldots, n - 1\).

3.3 Putting All Together

To finish our reduction we use the gadgets described above to build an actual graph \(G\) implementing the needed logic. Consider one instance of gadget \(G_W\) and two instances of gadgets \(G_U\), named \(G_{U1}\) and \(G_{U2}\). The three gadgets are arranged on three different lines starting with \(G_{U1}\) at the top, then \(G_W\) in the middle and finally \(G_{U2}\) on the bottom line. Sub-structure \(G^{(n-1+j-1)}_{U1}\) is placed in such a way to precede \(G^{(j)}_{W}\), which in turn
will precede $G_{U_2}^{(j)}$, for all $1 \leq j \leq n$. We now connect the three gadgets adding edges $(e_{U_1}^{(n-1)}, b_{W_1}^{(1)}), \ldots, (e_{U_1}^{(2n-2)}, b_{W_1}^{(n)})$ and $(e_{W_1}^{(1)}, b_{U_2}^{(1)}), \ldots, (e_{W_1}^{(n)}, b_{U_2}^{(n)})$. See Figure 3.

The idea is to force the pattern to cross $G_W$ during a match so that it will detect a vector $y_j \in Y$ (represented by $G_{W}^{(j)}$) orthogonal to some $x_i \in X$ (encoded as $P_{x_i}$), if such a vector exists. As observed earlier, pattern $P$ starts with sequence $bb$ and ends with $ee$. Moreover, such sequences do not appear anywhere else in the pattern. Thus, we place a new $b$-node for every $b_{W_1}^{(j)}$ and we connect these two with an edge as in Figure 3, for all $1 \leq j \leq n$. We do the same for $b_{U_1}^{(j)}$, for all $1 \leq j' \leq 2n - 2$. In the same manner we add new $e$-nodes for every $e_{W_1}^{(j)}$ and $e_{U_2}^{(j'')}$, for all $1 \leq j \leq n$, $1 \leq j'' \leq 2n - 2$. In this way pattern $P$ can start matching in the graph only in $G_{U_1}$ or $G_W$ and end only in $G_W$ or $G_{U_2}$. Finally, $2(n - 1) = 2n - 2$ is the minimum number of instances of sub-gadgets of $G_{U_1}$ and $G_{U_2}$ required by pattern $P$ to be able to match in $G_W$ any of its sub-patterns (i.e., any sub-pattern from $P_{x_1}$ to $P_{x_n}$).

$Y = \{y_1, y_2, y_3, y_4\} = \{(110), (011), (100), (001)\}$

We now prove that the reduction is correct, first focusing on the sub-patterns of $P$. 

![Figure 2 Gadget $G_U$.](image)

![Figure 3 Final graph $G$.](image)
Lemma 8. Pattern $P$ has a match in $G$ if and only if a sub-pattern $bP_x, e$ of $P$ has a match in $G_W$.

Proof. For the $(\Rightarrow)$ implication, the $eb$-edges can only be traversed once in this direction, as $P$ contains the sequence $eb$ but does not contain $be$. For this reason each distinct sub-pattern $bP_x, e$ matches a path from either a distinct portion of $G_{U\ell}$ ($\ell = 1, 2$) or $G_W$. Moreover, each occurrence of $P$ must begin with $bb$ and end with $ee$. String $bb$ can be matched only in $G_{U1}$ or $G_W$ while $ee$ is present only in $G_W$ or $G_{U2}$. Hence, in order to have a full match for pattern $P$ there must exist a sub-pattern $bP_x, e$ having a match in $G_W$.

The $(\Leftarrow)$ implication is trivial. In fact, if $bP_x, e$ has a match in $G_W$ then we can match $bP_x, e \ldots bP_{x-1}, e$ in $G_{U1}$ and $bP_{x+1}, e \ldots bP_x, e$ in $G_{U2}$ by construction, and have a full match for $P$ in $G$.

We are finally ready to prove Theorem 8

Proof of Theorem 8 First, we prove that the reduction is correct, then we analyze its cost and show how a sub-quadratic time algorithm for PMLG would contradict the OV Hypothesis.

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 8 which guarantees that $P$ has a match in $G$ if and only if a sub-pattern $P_x$ has a match in $G_W$, and the fact that, by Lemma 7 this holds if and only if $x_i \cdot y_j = 0$.

Alphabet size. Our alphabet is of size 4. Analogously to the earlier reduction 8 one can use encoding $\alpha(0) = 0000$, $\alpha(1) = 1111$, $\alpha(b) = 10$, and $\alpha(e) = 01$ and observe that (after small adjustments considered below) there is a bijection from matches before and after applying the encoding. Nodes with labels of length $2$ and $4$ can be replaced by chains of nodes labeled by single characters each. In order to make such encoding work, we need to make the pattern start with characters $e b b$ and end with characters $e e b$ to exploit the properties of the sequence $e b$. Moreover, we have to place and connect a new $e$-node to each $b$-node used to mark the beginning of a viable match and, in the same manner, we need to add a new $b$-node after every $e$-node used to mark the end of a match. (See our earlier reduction 8 for a full proof of an analogous adjustment.)

Cost. We analyze the cost of the reduction proving that, given sets of vectors $X$ and $Y$ with $n$ vectors each, the corresponding pattern $P$ and graph $G$ can be built in $O(nd)$ time and space. We observe that each $bP_x, e$ in $P$ has $d$ symbols that can be either 0 or 1 plus symbols $b$ and $e$. Since $P$ has $n$ sub-patterns $bP_x, e$ ad an additional $b$ at the beginning as well as one more $e$ at the end, summing everything up we get a length of $m = n(d + 2) = 2d = O(nd)$ symbols. As for $G_U$, it has $2n - 2$ sub-gadgets each one having $d$ nodes labeled with 0, $d$ nodes labeled with 1, and nodes $b^{(j)}_U$ and $e^{(j)}_U$. Hence there are $(2n - 2)(2d + 2) = O(nd)$ total nodes. Each node has a constant number of incident edges (at most 4) thus their size is $O(nd)$ as well. The same reasoning applies for $G_W$ as it has $O(nd)$ nodes labeled with 0 and at most $O(nd)$ nodes labeled with 1, plus those with $b$ and $e$. Also in this case, each node has a constant number of incident edges. For connecting $G_W$ to the two instances of $G_U$ we are adding one edge for every node $b^{(j)}_W$ and $e^{(j)}_W$ in $G_W$, which sum up to $O(nd)$ edges.

We are adding one additional $b$ node and edge for each sub-gadget $G^{(j)}_{U1}$ and $G^{(j)}_{W1}$ and one additional $e$ node and edge for each sub-gadget $G^{(j)}_{W2}$, which again sum up to $O(nd)$. Since the pattern and the graph have size $O(dn)$, we conclude that the cost of our reduction is indeed $O(nd)$ time and space. The alphabet reduction to size 2 considered above adds a constant factor.

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 PMLG 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 exist a
pair of orthogonal vectors between $X$ and $Y$ in $O(nd \cdot n^{1-\epsilon}d) = O(n^{2-\epsilon}\text{poly}(d))$ time, which
contradicts the OV Hypothesis.

4 Deterministic DAGs

In this section we show that the undirected graph $G$ obtained from the reduction described in
Section 3 can be transformed into a deterministic DAG. We first observe that it is immediate
to modify the proof of Theorem 3 so that it holds for a DAG. Instead, non-trivial changes
have to be applied in order to obtain a deterministic DAG, as stated in Theorem 4.

4.1 DAGs

Consider the definitions of edges $E_W$ and $E_U$ in the proof of Theorem 3. We immediately
obtain a directed acyclic graph just by considering such edges to be directed. From now
onward, when referring to $G$ we will intend its directed version. Indeed, each match of the
pattern must begin with a $bb$-edge, end with an $ee$-edge, and lay along a path between them.
So the edges can be oriented by construction from left (first $bb$-edge) to right (last $ee$-edge)
and from top ($G_{U1}$) to bottom ($G_{U2}$).

4.2 Determinization

The directed acyclic graph $G$ obtained in the previous paragraph must be further modified
to complete the proof of Theorem 3. Looking at $G$ it is clear that the non-deterministic
out-neighbours are the $e$-nodes that are connected with $b$-nodes in forming the $eb$-edges of
the graph. For instance, node $e_W^{(j)}$ has an edge to both $b_W^{(j+1)}$ and $b_U^{(j)}$. This issue can be
handled by merging $G_W$ and $G_{U1}$ into a new gadget $G_{WU}$. See Figure 4 for an example.

We start from $G_W$ and add parts of $G_{U1}$ when needed. Consider only sub-structure $G_{W}^{(j)}$
and assume that the first position in which the $1$-node is lacking is $h$. We place node $v_{jh}^{1}$
and the edges $(v_{jh}^{1},v_{jh+1},v_{jh}^{1})$ and $(v_{jh+1},v_{jh+1},v_{jh}^{1})$, or edge $(b_{W}^{(j)},v_{jh}^{1})$ in case $h = 1$. Moreover, we
place a partial version of sub-structure $G_{U1}^{(j)}$, $j' = n - 1 + j$ by adding to the graph the nodes and
edges of $G_{U1}^{(j')}$ that are located between position $h + 1$ and node $v_{U1}^{(n-1+j)}$ (included).
Now we connect the new node $v_{jh}^{1}$ to the first $0$-node and $1$-node of such substructure. If
there are other positions $h' > h$ lacking the $1$-node, then $v_{jh'}^{1}$ exists in the partial $G_{U1}^{(j')}$
and we connect nodes $v_{j,h'-1}^{0}$ and $v_{j,h'-1}^{1}$ to it. In case the missing $v_{j,h'}^{1}$ is the last one
$(h' = d)$, we connect $v_{j,d-1}^{0}$ and $v_{j,d-1}^{1}$ to the $e$-node of the partial $G_{U1}^{(j)}$. Finally, we place
edge $(v_{U1}^{(n-1+j)},b_{W}^{(j+1)})$. To conclude the merging, we apply such modification to $G_{W}^{(j)}$ and we
remove the edge $(e_{W}^{(j)},b_{W}^{(j+1)})$ for $1 \leq j \leq n - 1$.

At this point, we place gadget $G_{U2}$ and we connect $G_{WU}$ to it as in $G$. Adding substructures
$G_{U1}^{(1)}, \ldots, G_{U1}^{(n-1)}$ with one additional $b$-node connected to each of them, and connecting $G_{U1}^{(n-1)}$
to $G_{W}^{(j)}$ as in $G$, completes the transformation into the new directed acyclic graph, which we
call $G'$. Figure 5 gives a big picture perspective of such new graph.

Now every node $e_{W}^{(j)}$ and $e_{U2}^{(j)}$ of graph $G'$ has either only one outgoing edge, or an edge
to a $b$-node and one to an $e$-node, which means that we removed its only non-deterministic
feature.
4.3 Correctness and Complexity

We are now left to prove that the reduction is still correct also for the deterministic DAG $G'$. To this end, we have to adapt the statement of Lemmas 6, 7 and 8 so that they hold for $G'$. The proofs of such lemmas will be almost immediate since they share the most part of the reasoning with the original reduction.

Lemma 9. If sub-pattern $bP_xe$ has a match in $G_{WU}$ then all the nodes matching $P_x$ share the same $j$ coordinate and have distinct and consecutive $h$ coordinates (i.e., $P_x$ has a match in $G_W^{(j)}$).

Proof. During the merging process, we added edges for connecting $G_W$ the partial $G_{U1}$ structure and removed the ones between $G_W^{(j)}$ and $G_W^{(j+1)}$, for $1 \leq j \leq n - 1$. Nodes $b^{(j)}_W$ and $e^{(j)}_W$ were left connected with $G_W^{(j)}$. Thus, a match can now span one of the new edges and move into the partial $G_{U1}$. In doing so, such match maintains the same $j$ coordinate and the shortest path between node $b^{(j)}_W$ and either node $e^{(j)}_W$ or $e^{(j)}_{U1}$ still is $d + 2$ nodes long (b-node and e-node included). Hence, this lemma follows from Lemma 6 and the fact that sub-structures $G_W^{(j)}$ in $G'$ are the same as in $G$ but for the fact that the edges are directed.
Lemma 10. Sub-pattern \( bP_{x,e} \) has a match in the underlying sub-structure \( G_W \) of \( G_{WU} \) if and only if there is \( y_j \in Y \) such that \( x_i \cdot y_j = 0 \).

Proof. The proof follows from Lemmas 7 and 9, and from the fact that sub-structures \( G^{(i)}_W \) are entirely present also in graph \( G' \).

Lemma 11. Pattern \( P \) has a match in \( G' \) if and only if a sub-pattern \( bP_{x,e} \) of \( P \) has a match in the underlying sub-structure \( G_W \) of \( G_{WU} \).

Proof. For the \((\Rightarrow)\) implication, notice that the \( eb \)-edges are still present also in graph \( G' \). Hence, as in Lemma 8, the \( eb \)-edges can only be traversed once in this direction and each distinct sub-pattern \( bP_{x,e} \) matches a path from either a distinct portion of \( G_{U\ell} \) \((\ell = 1, 2)\) or \( G_{WU} \). Moreover, each occurrence of \( P \) must begin with \( bb \) and end with \( ee \). String \( bb \) can be matched only in \( G_{WU} \), hence the match must start in this gadget. String \( ee \) is found either in \( G_{U2} \) or in \( G_{WU} \). Observe that, by construction, once a match for pattern \( P \) is started in \( G_{WU} \) the only way in which it can be successfully concluded is either by matching \( ee \) within this gadget or by first matching a portion of \( G_{U2} \) and then \( ee \). Because of the structure of the graph, in both cases a sub-pattern \( bP_{x,e} \) of \( P \) must match one of the sub-structures \( G^{(i)}_W \) that are present in \( G_{WU} \).

The \((\Leftarrow)\) implication is trivial and can be proven as in Lemma 8.

Thanks to the above lemmas, the correctness of this reduction can be easily proven also for graph \( G' \).

Theorem 12. 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.

Proof. The proof follows from Lemmas 9, 10, and 11, just like in the proof of Theorem 3.

To conclude the proof of Theorem 12, we are now left with proving that the complexity of the new reduction remains the same, namely \( O(nd) \), and that our graph can be tweaked to achieve maximum sum of indegree and outdegree 3. The complexity can be checked immediately looking at the structure of \( G' \). First, notice that gadget \( G_W \) has size \( O(nd) \) and it is still present within the new gadget \( G_{WU} \), hence \( G' \) must have at least \( O(nd) \) nodes and edges. On the other hand, obtaining \( G' \) by merging together \( G_W \) and \( G_{U1} \) means that the size of \( G' \) cannot be greater than the size of \( G \), which is indeed \( O(nd) \). We conclude that the size of graph \( G' \) is \( O(nd) \), thus the reduction can be performed in \( O(nd) \) time and space. Degree 3 can be easily obtained applying the same modification used in our previous work [8], as the \( G^{(i)}_U \) and \( G^{(i)}_W \) substructures are basically the same and pairs of dummy nodes can be placed to reduce the degree of the 0 and 1-nodes.

5 Zig-zag Matching

The lower bound given for the PMLG problem can be extended to the special case of an undirected graph with maximum degree 2, that is, an undirected path. To this end, we need to modify the reduction defining a new alphabet, pattern and graph.

The original alphabet \( \Sigma = \{b, e, 0, 1\} \) is replaced with \( \Sigma' = \{b, e, A, B, a, t\} \). Characters 1 and 0 are encoded in the following manner:

\[ 1 = ABA \]
\[ 0 = ABABABA \]
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 sub-pattern
\[
P_{x_i} = 1\ 0\ 1
\]
will be encoded as
\[
P'_{x_i} = s\ ABA\ s\ ABABABA\ s\ ABA\ s\ .
\]

### 5.1 Pattern

A new pattern \(P'\) is built applying the aforementioned encoding to each one of the sub-patterns \(P_{x_i}\), thus obtaining a new sub-pattern \(P'_{x_i}\). We then concatenate all the sub-patterns \(P'_{x_i}\) 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:
\[
P = \text{bb 100 e b 101 ee}
\]
\[
P' = \begin{array}{c}
1 & 0 & 0 \\
b & t & s & ABA & s & ABABABA & s & ABA & s & \\
t & s & ABA & s & ABABABA & s & ABA & s & t & e
\end{array}
\]

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

### 5.2 Graph

The same encoding used for the pattern will be applied to the graph. The strategy is to encode \(G_W\) in a linear structure concatenating chains of nodes representing each sub-structure \(G^{(j)}_W\) one after the other.

The positions \(h\) in which both a 0- and a 1-node are present are replaced by a path that can be matched both by \(0 = ABABABA\) and \(1 = ABA\). Positions \(h\) with only a 0-node and no 1-node are encoded instead with a path that can be matched only by \(0 = ABABABA\) (see Figure 6). We use \(s\)-nodes to separate these paths. We denote by \(LG^{(j)}_W\) (Linear \(G^{(j)}_W\)) this linearized version of \(G^{(j)}_W\). Moreover, given sub-structure \(G^{(j)}_W\), two new \(t\)-nodes will mark the beginning and the ending of its encoding. Figure 7 depicts how to perform such transformation for \(G^{(j)}_W\).

In a similar manner, \(G_U\) is also encoded as a path. We do not need to encode all its \(2n-2\) substructures: since the matching path is now allowed to pass through nodes more than once, we only need to encode one of them, in the same manner as done for \(G^{(j)}_W\). Let \(LG_U\) (Linear \(G_U\)) be the linearized version of a “jolly” gadget that was 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, \(LG^{(j)}_W\), a \(b\)-node on the left and an \(e\)-node on the right as in Figure 8. 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 \(LG^{(j)}_W\) and, in general, they are marking the beginning and the end of a match for a sub-pattern \(P'_{x_i}\). In order to construct the final graph \(LG\) we concatenate all the gadgets \(LG^{(1)}, LG^{(2)}, \ldots, LG^{(n)}\) into one chain of nodes. Figure 9 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,
Figure 6 New substructures. (a) The old substructure is replaced by a chain of nodes that can match either sABAs (which represents 1) or sABABABAs (which represents 0). (b) The chain of nodes that is replacing a 0-node can match only the string sABABABAs.

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 $LG^{(j)}_W$, each one surrounded by two $LG_U$ instances. Hence the total size of the graph remains $O(nd)$.

5.3 Putting All Together

In order to achieve the desired reduction we will prove some properties on $LG$.

We introduce some new notation to simplify the exposition of the upcoming lemmas. We use $t_i LG^{(j)}_W t_r$ to refer to $LG^{(j)}_W$ extended with the t-nodes on its left and on its right. When referring to the $k$-th s-character in $P'_x$, we mean the $k$-th s-character found scanning $P'_x$ from left to right; in the same manner we refer to the $k$-th s-node in $LG^{(j)}_W$. Moreover, $s_k^{(P'_x)}$ denotes the $k$-th s-character in $P'_x$, and $s_k^{(LG^{(j)}_W)}$ denotes the $k$-th s-node in $LG^{(j)}_W$.

Lemma 13. If sub-pattern $tP'_x t$ has a match in $t_i LG^{(j)}_W t_r$ starting at $t_l$ and ending at $t_r$, then the $k$-th s-character in $P'_x$, matches the $k$-th s-node in $LG^{(j)}_W$, for all $1 \leq k \leq d + 1$.

Proof. First we prove that all the s-nodes in $t_i LG^{(j)}_W t_r$ are matched exactly once by $tP'_x t$. By construction, sub-pattern $P'_x$ has $d + 1$ s-characters, and $LG^{(j)}_W$ has $d + 1$ s-nodes. Since we are working on a chain of nodes and the match is starting at $t_l$ and ending at $t_r$, all the nodes between $t_l$ and $t_r$ have to be matched at least once by $P'_x$. Assume by contradiction that one such s-node is matched more than once. Sub-pattern $P'_x$ is left with strictly less than $d$ s-characters available for matching the other $d$ s-nodes and we reach a contradiction. Now we can prove the statement of the lemma by induction on $k$, i.e the index of the s-characters and s-nodes.
0

\[ G^{(j)}_W = \]

\[ \rightarrow \quad LG^{(j)}_W \]

**Figure 7** A row of \( G_F \) is converted in a linear structure using characters \( s \) and \( t \) as markers.

**Base Case** \( k = 1 \). The match starts at \( t_l \) hence the only node that \( s_1^{(P'_x)} \) can match is the first \( s \)-node to the right on \( t_l \), i.e., \( s_1^{(LG^{(j)}_W)} \).

**Inductive Case** \( k > 1 \). The inductive hypothesis tells us that all the nodes up to \( s_k^{(LG^{(j)}_W)} \) have been matched by consecutive \( s \)-characters of \( P'_x \) up to \( s_k^{(P'_x)} \). We have to prove the statement for \( k + 1 \). Starting from node \( s_k^{(LG^{(j)}_W)} \) the next \( s \)-nodes that can be matched by \( s_{k+1}^{(P'_x)} \) are \( s_{k-1}^{(LG^{(j)}_W)} \) and \( s_{k+1}^{(LG^{(j)}_W)} \). Character \( s_{k+1}^{(P'_x)} \) cannot match node \( s_{k-1}^{(LG^{(j)}_W)} \) since it has already been matched by \( s_{k-1}^{(P'_x)} \) and, as argued earlier, every \( s \)-node can be matched only once. Thus \( s_{k+1}^{(P'_x)} \) has to match \( s_{k+1}^{(LG^{(j)}_W)} \).

**Lemma 14.** Sub-pattern \( tP'_x t \) has a match in \( t_t LG^{(j)}_W t_r \) starting at \( t_l \) and ending at \( t_r \) if and only if there exist \( y_j \in Y \) such that \( x_i \cdot y_j = 0 \).

**Proof.** This property has already been proved for gadget \( G_W \) in Lemma 7, thus what we are left to prove is that \( LG^{(j)}_W \) behaves the same as the sub-gadget \( G^{(j)}_W \). First recall that in the
construction of \( LG^{(j)}_W \) we placed an encoded \( 1 \) if in \( G^{(j)}_W \) we had both a \( 0 \)-node and a \( 1 \)-node in the same position, while we placed an encoded \( 0 \) if we had only a \( 0 \)-node. Lemma 13 guarantees that the encoding in \( P \) of a single character of \( P \) is aligned with the encoding in \( LG^{(j)}_W \) of a single node of \( G_W \), preventing (the encoding of) a character of \( P \) from matching (the encoding of) multiple nodes of \( G_W \) and vice versa. By construction, \( 1 = ABA \) can match the encoding of a 1-node while it fails to match the encoding of the \( 0 \)-nodes, since their encoding involves too many characters. On the other hand, \( 0 = ABABABA \) can match an encoded \( 0 \)-node with a natural alignment, but it can also match the encoding of a 1-node by scanning it forwards, backwards and forwards again. Therefore the logic behind \( LG^{(j)}_W \) safely implements the one of \( G^{(j)}_W \), and from this point onward, one can follow the same reasoning as in Lemma 7 to complete the proof.

\[\text{Figure 8} \quad \text{The} \; LG^{(j)}_W \text{ structure surrounded by two instances of } LG_U. \text{ The } t \text{-nodes establish the beginning and the end of a match for a sub-pattern } tP'_x \; t \text{ while nodes } b \; \text{and } e \text{ are the starting and ending point for a match of the whole pattern } P'.\]

\[\text{Figure 9} \quad \text{The final graph } LG.\]

The main difference with the original proof resides in assuming that a match for \( P'_x \) 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 \( LG^{(j)}_W \) gadget from left to right at least once.

\[\text{Lemma 15.} \quad \text{Pattern } P' \text{ has a match in } LG \text{ if and only if there exist } i \text{ and } j \text{ such that } i \text{ is even and sub-pattern } tP'_x \; t \text{ has a match in } t_lLG^{(j)}_W \; t_r \text{ starting at } t_l \text{ and ending at } t_r.\]

\[\text{Proof.} \quad \text{For the } (\Rightarrow) \text{ implication, first observe that the } b \text{- and } e \text{-nodes in } LG \text{ are forcing a direction to follow. Let } LG^{(j)}_{U_l} \text{ and } LG^{(j)}_{U_r} \text{ be the } LG_U \text{ gadgets to the left and to the right of } LG^{(j)}_W, \text{ respectively. Since pattern } P' \text{ starts with a } b \text{ and ends with an } e, \text{ a match can only start at the } b \text{-node on the left of } LG^{(j)}_{U_l} \text{ and end at the } e \text{-node on the right of } LG^{(j)}_{U_r}, \text{ for some } j. \text{ Hence } LG^{(j)}_W \text{ needs to be crossed by a match from left to right at least once. Thus, there must exist a sub-pattern } tP'_x \; t \text{ that has a match starting at } t_l \text{ and ending at } t_r. \text{ For such a pattern Lemma 13 applies. Moreover, because of our construction, only a sub-pattern on even position can achieve such a match.}\]
The ($\Rightarrow$) implication is immediate since given a sub-pattern $tP'_x,t$ which has a match in $t(\mathit{LG}^{(j)}_U)_t$, one can match $btP'_x,t \ldots tP'_{x_{i-1}},t$ in $\mathit{LG}^{(j)}_{Uj}$ and $tP'_{x_{i+1}},t \ldots tP'_x,te$ in $\mathit{LG}^{(j)}_{Ue}$ and have a full match for $P'$ in $\mathit{LG}$.

Since Lemma 15 gives us a property which holds only if a sub-pattern 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 sub-patterns $P'_{x_i}$ on odd position with the next sub-patterns $P'_{x_{i+1}}$ on even position, for every $i = 1,3,\ldots$. For example, if $n$ is even, we will have:

$$
P'^{(1)} = btP'_{x_1}tP'_{x_2}tP'_{x_3}tP'_{x_4}\ldots tP'_{x_{n-1}}tP'_xt_e = P'
$$

$$
P'^{(2)} = btP'_{x_2}tP'_{x_1}tP'_{x_4}tP'_{x_3}\ldots tP'_{x_{n-1}}tP'_{x_n}t_e
$$

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

Now we are ready to present the end result.

**Theorem 16.** Either $P'^{(1)}$ or $P'^{(2)}$ has a match in $\mathit{LG}$ if and only if there exist vectors $x_i \in \mathit{X}$ and $y_j \in \mathit{Y}$ which are orthogonal.

**Proof.** For ($\Rightarrow$) we assume that either $P'^{(1)}$ or $P'^{(2)}$ have a match in $\mathit{LG}$. By Lemma 15 this means that there exists a sub-pattern $P_{x_i}^{(q)}$, $q \in \{1,2\}$ which has a match in $\mathit{LG}^{(j)}_W$, for some $j$. Lemma 14 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 exist two orthogonal vectors $x_i \in \mathit{X}$ and $y_j \in \mathit{Y}$. Thanks to Lemma 14 we find a sub-pattern $P'_{x_i}$ matching $\mathit{LG}^{(j)}_W$. By construction, $P'_{x_i}$ has to be in even position either in $P'^{(1)}$ or in $P'^{(2)}$. By Lemma 15 this means that either $P'^{(1)}$ or $P'^{(2)}$ has a match in $\mathit{LG}$.

Theorem 5 on page 4 follows directly from the correctness of these constructions.

**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. Jarno Alanko, Alberto Policriti, and Nicola Prezza. On Prefix-Sorting Finite Automata. *arXiv e-prints*, page arXiv:1902.01088, February 2019. [arXiv:1902.01088](https://arxiv.org/abs/1902.01088)

3. Amihood Amir, Moshe Lewenstein, and Noa Lewenstein. Pattern matching in hypertext. *J. Algorithms*, 35(1):82–99, 2000.

4. 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. [URL: http://doi.acm.org/10.1145/2746539.2746612](http://doi.acm.org/10.1145/2746539.2746612) doi:10.1145/2746539.2746612
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.

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.

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. URL: http://dx.doi.org/10.1109/FOCS.2015.15, doi:10.1109/FOCS.2015.15.

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.

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

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.

Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. Journal of Computer and System Sciences, 62(2):367 – 375, 2001. URL: http://www.sciencedirect.com/science/article/pii/S0022000000917276 doi:https://doi.org/10.1006/jcss.2000.1727.

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

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.

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.

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

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. URL: http://dx.doi.org/10.1109/TCBB.2013.2297101 doi:10.1109/TCBB.2013.2297101.

Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. Theoretical Computer Science, 348(2):357 – 365, 2005. URL: http://www.sciencedirect.com/science/article/pii/S0304397505005438 doi:https://doi.org/10.1016/j.tcs.2005.09.023.