Algorithms for Molecular Biology

RESEARCH

Open Access

Pattern matching with Elastic-Degenerate strings and Elastic-Founder graphs

Rocco Ascone[1], Giulia Bernardini[1,6*], Alessio Conte[2], Massimo Equi[3], Esteban Gabory[4,5], Roberto Grossi[2] and Nadia Pisanti[2*]

Abstract

Elastic Degenerate (*ED*) strings and Elastic Founder (*EF*) graphs, here collectively named *variable* strings, are two representations of acyclic components of pangenomes which extend the well-known notion of indeterminate string. Recent studies have focused extensively on algorithmic tasks involving these structures and other forms of variable strings that they generalize. Among such tasks, the basic operation of matching a pattern into a text, a fundamental toolkit for pangenomic data analysis, deserves special attention. In this paper, (1) we establish a clear taxonomy across *ED* strings and *EF* graphs, categorizing types of variable strings from the simplest linear (*solid*) string to the most complex general cases; (2) we consider the problem MATCH(*X*,*Y*) of matching a solid or variable pattern of type *X* into a variable text of type *Y*, and investigate its time complexity when *X* and *Y* are chosen from types of variable strings in the taxonomy of (1). For all possible *X* and *Y*, we either provide a non-trivial, often sub-quadratic, upper bound for MATCH(*X*,*Y*), or we prove a quadratic conditional lower bound, taking as a reference the existing quadratic conditional lower bounds for MATCH(SOLID,*ED*) and MATCH(SOLID,*EF*). A preliminary version of this work appeared in [Ascone et al., WABI 2024].

Keywords Pangenomics, Pattern matching, Degenerate string, Founder graph, Fine-grained complexity, Genome variant

Introduction

Analysing an ever-increasing number of genome sequences and determining which genome to select as a reference are two major challenges in genomic data analysis. Recently, these challenges have converged into a powerful approach: using a *pangenome* – rather than a single genome – as a reference. According to [28], a pangenome is indeed "any collection of genomic sequences to be analyzed jointly or to be used as a reference". The new -omics science *pangenomics* thus imposed a paradigm shift: in several analysis tasks, and in particular for species like humans that enjoy a widespread availability of sequencing data as well as a growing awareness of genomic variants, the simple linear genomic sequence is being replaced by more complex graph-like structures [42, 69]. Unlike a linear reference, a pangenome reference allows a simultaneous and compact representation of variations and commonalities among the underlying sequences. The most general pangenome representations are edge- and/or node-labeled directed graphs [9], such as the *variation graphs* [22, 36, 48] (with their haplotype aware version [85]) and *sequence graphs* [78]. Simpler *acyclic* alternatives to these representations are *Elastic-Degenerate strings* [53, 58] (*ED* strings) and the slightly more haplotype-aware *Elastic*

*Correspondence:
Giulia Bernardini
giulia.bernardini@units.it
Nadia Pisanti
nadia.pisanti@unipi.it
[1] University of Trieste, Trieste, Italy
[2] University of Pisa, Pisa, Italy
[3] University of Helsinki, Helsinki, Finland
[4] CWI Amsterdam, Amsterdam, Netherlands
[5] University of Palermo, Palermo, Italy
[6] University of Milan, Milan, Italy

Founder graphs [79] (*EF* graphs), as well as their non-elastic versions *Generalised Degenerate strings* [3, 4, 67] (*GD* strings) and *Founder graphs* [66] (*F* graphs).

Figure 1 shows the taxonomy of variable strings, from the simplest solid string (top) to the most general variable strings (bottom-left and bottom-right). 1-*D* strings are also known in the Bioinformatics literature as *indeterminate* strings. They have been extensively employed and investigated [1, 2, 5, 6, 20, 30, 32–35, 49, 57, 59, 60, 65, 73, 74, 82, 83, 86] as they naturally represent the IUPAC encoding [62] of DNA and RNA nucleotides subsets and can be used to highlight SNPs (Single Nucleotide Polymorphisms, i.e., substitutions of single letters in the DNA sequence). A wide literature exists also for strings with wildcards [12–14, 24, 44, 54, 55] that can be seen as a special case of 1-*D* strings. The class k-*D* generalises 1-*D* in that all variants have length $k \geq 1$, while the class *GD* only requires the variants at any locus to be of the same length, without requiring them to have the same length all over. *GD* strings thus represent gapless multiple sequence alignments of fixed width, e.g. high-scoring local alignments of multiple sequences. Finally, *ED* strings allow variants of any size at any locus, including the empty string, which allows explicitly representing short INDELs (insertions and deletions of letters in the genomic sequence). *ED* strings correspond to the VCF file format [31] for genomic variants. The same taxonomy holds for founder graphs with the addition of edges that connect variants of adjacent segments according to haplotype information and the further difference that *EF* graphs cannot contain the empty string. The term *variable string* collectively refers to *ED* strings, *EF* graphs, and any of their above-mentioned special cases.

In contrast with more general representations like variation and string graphs, *ED* strings support both theoretically [7, 16, 17] and practically [21, 23, 53, 76, 76, 77] fast *on-line* exact and approximate pattern matching [15, 18, 19] when the pattern is a solid (i.e. non-variable) string. Nevertheless, conditional lower bounds for this problem [11, 17] and the problem of indexing *ED* strings [50] rule out the existence of even faster methods. Efficient algorithms also exist to decide whether the intersection of the sets of sequences represented by two *ED* strings is non-empty [45–47] (*intersection query*). *ED* strings cannot be efficiently indexed, but filtering methods [70–72] could be applied to their k-mers, that can be efficiently indexed [64]. Furthermore, *GD* strings support fast dynamic-programming-based alignment [52, 67, 68] for approximate matching with a solid string and linear-time answer to intersection queries [3, 4].

For *EF* graphs, efficient off-line pattern matching algorithms are known under specific conditions which can be ensured with a linear-time construction from a gapless multiple sequence alignment (for *F* graphs), or a near-linear time construction from a general multiple sequence alignment (for *EF* graphs) [40, 41, 66, 80, 81, 87]. Here, "near-linear time" means that the time complexity is linear when parametrised in the maximum number of variations appearing in a single locus of the *EF* graph.

While the main advantage of variable strings is their supporting provably efficient and accurate methods, their limitation is in their expressivity: both *ED* strings and *EF* graphs are acyclic structures that can only express mismatches and INDELs, and cannot adequately represent structural genomic variation such as repetitions, translocations, or inversions. A main challenge in computational pangenomics is to balance the pangenome representations' accuracy

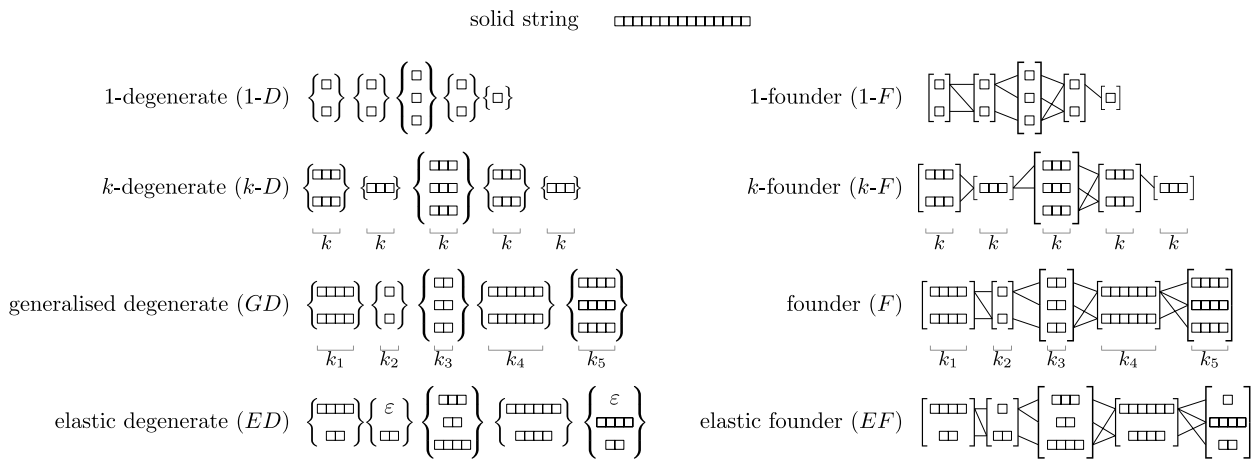

Fig. 1 The landscape of variable strings considered in this work. The leftmost column depicts degenerate strings in increasing order of generality; the rightmost column shows an analogous progression for founder graphs. Squares represent characters, ε denotes the empty string

and expressivity and the efficiency of the methods for their construction and analysis: variable strings are a trade-off between expressive power and efficiency.

This paper aims to provide a clear taxonomy and a time-complexity landscape of matching problems in variable strings. More formally, we analyse the complexity of the problem, denoted MATCH(X, Y), of matching a solid or variable pattern of type X in a variable text of type Y: the types of variable strings we consider for the pattern and text are in Figure 1. For instance, MATCH(1-D,k-F) is the problem of finding occurrences of a 1-D pattern within a k-F graph.

Our results

For the problem of matching a solid pattern of size M into an ED or EF text of size N, it is known from the literature that an algorithm with complexity $\mathcal{O}(M^{1-\varepsilon}N)$ or $\mathcal{O}(MN^{1-\epsilon})$ with $\epsilon > 0$ would contradict the Strong Exponential Time Hypothesis (SETH) [11, 16, 17, 40, 51], and the contradiction holds even if such complexity is achieved at query time after a polynomial-time indexing step [37, 38, 50]. Quadratic-time algorithms are known [17, 23]. Moreover, strongly sub-quadratic algorithms are known for MATCH(SOLID, 1-D) [26] and for MATCH(1-D, 1-D) [60], in the latter case restricted to constant-size alphabets.

As a consequence, our reference bound is quadratic: given two types X and Y of strings (variable or solid), either this is proved as a lower bound for MATCH(X, Y), or a better algorithm – an upper bound – should be exhibited. We aim to paint a complete picture of the complexity of this task for all types X of patterns and all types Y of variable texts, investigating both lower and upper bounds. All our results are anticipated in Section Definitions and summary of the results and summarised in Tables 1 and 2, where columns correspond to the pattern and rows correspond to the text. Note that, due to space constraints, in the tables we write $\Omega((MN)^{1-\epsilon})$, for every $\epsilon > 0$, to denote the quadratic lower bound, but in fact, all our lower bound results prove both bounds $\Omega(M^{1-\epsilon}N)$ and $\Omega(MN^{1-\epsilon})$ (for every $\epsilon > 0$).

This work is an extended version of the conference paper [8].

Table 1 Complexity charts for problem MATCH(SOLID, Y), where Y ranges over columns

Pattern \ Text	1-D	k-D	GD	ED
SOLID	[26] $\mathcal{O}(n + n\log^2 m)$	**Thm. 1** $\mathcal{O}(N + N\log^2 m)$	**Thm. 3** $\mathcal{O}(mn + N)$	**Thm. 11** $\Omega((mN)^{1-\epsilon})$

(a) Complexity of matching a solid pattern in several types of degenerate strings, in increasing order of generality.

Pattern \ Text	1-F	k-F	F	EF
SOLID	**Thm. 2** $\mathcal{O}(\sqrt{m}(\lvert E\rvert + N + n\log^2 m))$		**Thm. 4** $\mathcal{O}(mn + N + \lvert E\rvert)$	[40] $\Omega((m\lvert E\rvert)^{1-\epsilon})$

(b) Complexity of matching a solid pattern in several types of founder graphs, in increasing order of generality.

Symbols M and N denote the total size of the pattern and the text, respectively, while m and n denote the respective *number of segments* (see Section Definitions and summary of the results). For graphs, $|E|$ denotes the number of edges. Green cells are for truly subquadratic $\mathcal{O}(NM^{1-\epsilon})$ (for some $\epsilon > 0$) upper bounds, yellow cells indicate that the upper bounds are subquadratic whenever $N/n = \omega(1)$, and red cells are for a quadratic lower bound $\Omega((MN)^{1-\epsilon})$ (for every $\epsilon > 0$) conditioned on SETH, even when the alphabet is of constant size and the length of the variants in each segment is $\mathcal{O}(1)$ (our bounds are even tighter, check the referred results for details). Note that the reduction for EF in [40] implies also the $\Omega((mN)^{1-\epsilon})$ bound declared in the table. A truly subquadratic upper bound for MATCH(SOLID,1-D) was known before this work. A quadratic conditional lower bound for MATCH(SOLID,ED) was also known, as it is directly implied by [10, Theorem 10], but this result only holds when the maximum variant length in the ED text of length n is $\omega(\log n)$. In Theorem 11 we prove that the lower bound applies also to cases where the maximum variant length is $\mathcal{O}(1)$. All the other results are novel

Definitions and summary of the results

We start with basic definitions and notation on solid strings following [29]. Given an ordered alphabet Σ whose elements are called *letters*, we denote by Σ^n the set of *strings* $T = T[1]\dots T[n]$ (also called *solid* strings) of length $|T| = n$ over Σ, and by Σ^* the set of all possible strings of any length, including the *empty string* ε. By T_1T_2 or $T_1 \cdot T_2$ we denote the *concatenation* of two strings T_1 and T_2, i.e. $T_1T_2 = T_1[1]\dots T_1[|T_1|]T_2[1]\dots T_2[|T_2|]$. For any string T and integer $k > 0$, T^k denotes the concatenation of k copies of T. For any $1 \le i \le j \le n$, $T[i\,..\,j]$ is the *fragment* of T starting at position i and ending at position j. The fragment $T[i\,..\,j]$ is an *occurrence* of the underlying *substring* $P = T[i]\dots T[j]$; we say that P occurs at *position* i in T, and that P is a *factor* of T. A *prefix* of T is a fragment of the form $T[1\,..\,j]$ and a *suffix* of T is a fragment of the form $T[i\,..\,n]$. Given a string T over Σ, we call *suffix tree* of T the compacted trie of all suffixes of $T \cdot \$$, where $\$ \notin \Sigma$. The suffix tree of a string $T[1\,..\,n]$ occupies $\mathcal{O}(n)$ space and it can be constructed in $\mathcal{O}(n)$ time for integer alphabets of size $|\Sigma| = \mathcal{O}(n^{\mathcal{O}(1)})$ [43, 88]. Unless specified otherwise, in this paper we assume that $\Sigma = [0, \sigma]$ with $\sigma = n^{\mathcal{O}(1)}$; note that, without this assumption, the time complexity of some of our algorithms is increased by logarithmic factors.

An *elastic-degenerate* (*ED*) *string* T over an alphabet Σ is a sequence $T = T[1]\cdots T[n]$ of n finite sets, called *segments*, where each $T[i]$ is a subset of Σ^*; we call $|T| = n$ the *length* of T. The *size* $\|T\| = N$ of T is the total number of characters in T, i.e. $N = N_\varepsilon + \sum_{i=1}^{n}\sum_{S\in T[i]}|S|$, where N_ε is the total number of empty strings in the segments of T; the *cardinality* B of T is the total number of strings in all segments, i.e. $B = \sum_{i=1}^{n}|T[i]|$. We call the set $\mathcal{V}$ of distinct strings that appear in at least one segment of T its *vocabulary*. We also denote by N_i and B_i the size and the cardinality of segment $T[i]$, respectively; finally, for any $1 \le i \le j \le n$, $T[i\,..\,j]$ denotes the fragment $T[i]\cdots T[j]$ between segments i and j of T.

Table 2 Complexity charts for problem

MATCH(X, Y), where X ranges over rows and Y over columns of each table.

Pattern \ Text	1-D	k-D	GD,ED
1-D	[60] $\mathcal{O}(n + n \log m)$	**Thm. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 2** $\Omega((MN)^{1-\epsilon})$
k-D	**Thm. 6** $\Omega((MN)^{1-\epsilon})$	**Thm. 7** $\Omega((MN)^{1-\epsilon})$	**Cor. 3** $\Omega((MN)^{1-\epsilon})$
GD,ED	**Cor. 1** $\Omega((MN)^{1-\epsilon})$	**Cor. 3** $\Omega((MN)^{1-\epsilon})$	**Cor. 3** $\Omega((MN)^{1-\epsilon})$

(a) Degenerate pattern, degenerate text

Pattern \ Text	1-F	k-F	F,EF
1-D	**Thm. 8** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$
k-D	**Cor. 4** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$
GD,ED	**Cor. 4** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$	**Cor. 4** $\Omega((MN)^{1-\epsilon})$

(b) Degenerate pattern, founder text

Pattern \ Text	1-D	k-D	GD,ED
1-F	**Thm. 9** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$
k-F	**Cor. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$
F,EF	**Cor. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$	**Cor. 5** $\Omega((MN)^{1-\epsilon})$

(c) Founder pattern, degenerate text

Pattern \ Text	1-F	k-F	F,EF
1-F	**Thm. 10** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$
k-F	**Cor. 6** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$
F,EF	**Cor. 6** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$	**Cor. 6** $\Omega((MN)^{1-\epsilon})$

(d) Founder pattern, founder text

The yellow cell in the upper-left corner indicates that the upper bound holds only for constant-size alphabets, as it has an exponential dependency on the alphabet size. Red cells denote a quadratic lower bound $\Omega((MN)^{1-\epsilon})$ (for every $\epsilon > 0$) conditioned on SETH, which holds even when the alphabet is of constant size and the length of the variants in each segment is $\mathcal{O}(1)$ (our bounds are even tighter, check the referred results for details). Orange cells denote that the quadratic lower bound only holds for variable strings in which the length of the variants in the segments is $\omega(\log n)$: it is an open problem whether these lower bounds also hold when the length of the variants is $\mathcal{O}(\log n)$. Letters in the time complexities are as in Table 1. All the results in this table are novel, except for the subquadratic upper bound for MATCH(1-D,1-D) that was known before this work

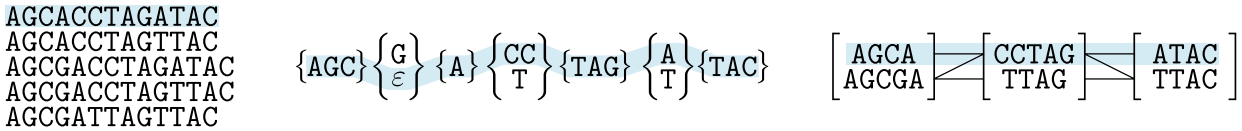

Fig. 2 An *ED* string (center) and an *EF* graph (right) built from the same set of strings (left). Note that the strings on the left are a subset of the languages of the *ED* string and the *EF* graph

If for every i the strings in $T[i]$ have all the same length k_i (called the *width* of $T[i]$), we say that T is a *generalised degenerate* (*GD*) string. If in addition all segments $T[i]$ have the same width k, T is a *k-degenerate* string (k-D, in short). In the special case $k = 1$, T is known in the literature as a *degenerate* or *indeterminate* string. Finally, if for every i it holds that $B_i = 1$, then we have a solid string. A schematic representation of these classes is in Figure 1.

An *elastic founder* (*EF*) *graph* is a pair $G = (T, E)$, where T is an *ED* string s.t. the empty string ε is not an element of any of its segments,[1] and $E = \bigcup_{i=1}^{n-1} E_i$, where E_i is the set of edges from $T[i]$ to $T[i+1]$, which can be identified with a subset of the Cartesian product $[1, B_i] \times [1, B_{i+1}]$. G is a *founder graph*, F graph in short (resp. a *k-founder* graph, k-F in short) if T is a GD (resp a k-D) string. $G[i..j] = (T[i..j], \bigcup_{\ell=i}^{j-1} E_\ell)$ is the fragment of G between $T[i]$ and $T[j]$. The size of G is $N + |E|$, i.e. the sum of the size of the underlying *ED* string T and the total number of edges of G. Figure 2 shows an *ED* string of length $n = 7$ and size $N = 17$, and an *EF* graph with $n = 3$, $N = 26$, $|E| = 6$, and hence total size 32.

The language $\mathcal{L}(T)$ of an *ED* string T consists of all the strings that can be obtained by concatenating one string per segment, maintaining their order: $\mathcal{L}(T) = \{S_1 \cdots S_n : S_i \in T[i] \ \forall\ i \in [1, n]\}$. The language of an *EF* graph G is defined analogously, but only strings that are connected by an edge can be concatenated: $\mathcal{L}(G) = \{S_1 \cdots S_n : S_i \in T[i] \ \forall\ i \in [1, n]$ and $(S_i, S_{i+1}) \in E_i \ \forall\ i \in [1, n-1]\}$. We remark that if T is an *ED* string (resp. *EF* graph), then in general $\mathcal{L}(T)$ may contain strings of different sizes, whereas if T is any among 1-D, 1-F, k-D, k-F, GD, F, then all strings in $\mathcal{L}(T)$ must have the same size.

For example, the language of the *EF* graph of Figure 2 consists of the 5 strings shown on the left; the language of the *ED* string includes in addition the strings {AGCATTAGATAC, AGCATTAGTAC, AGCGATTAGATAC} , thus consists of 8 strings in total.

Remark 1 A common logical pitfall is to believe that it is possible to transform all non-elastic variable strings

[1] We keep this distinction from *ED* strings to match the existing literature.

(*GD* strings, *F* graphs, *k-D* strings and *k-F* graphs) to equivalent 1-*F* graphs by splitting each segment into multiple segments (one containing the first letter of each string in the original segment, one containing the second letter, and so on) and adding appropriate edges to preserve letter sequences. However, the language of the 1-*F* graph obtained with this procedure is generally larger than that of the original variable string. Figure 3 exemplifies this issue when trying to transform a *k-D* string into a 1-*F* graph.

Recall that by *variable strings* we collectively refer to *ED* strings, *EF* graphs, and any of the special cases introduced above. In general, we say that there is an occurrence of a variable pattern P in a variable text T ending at $T[j]$ if there exists $1 \leq i \leq j$ such that a string in the language of P occurs as a substring of a string $t = t_i \cdot t_{i+1} \cdots t_j \in \mathcal{L}(T[i\,..\,j])$, with $t_k \in T[k]\ \forall\ k \in [i, j]$, ending within the last occurrence of t_j in t. In Figure 2, the solid pattern CTAGA occurs in *ED* ending at position 6, and in *EF* ending at position 3. Figure 4 visually represents two occurrences of an *ED* pattern in an *EF* text. We provide more technical definitions of variable pattern matching, specialised for the different cases studied in this paper, in sections Matching a solid pattern in a *GD* or *F* text and Matching a solid pattern in a *k-D* or *k-F* text.

PATTERN MATCHING ON VARIABLE TEXT: MATCH(X, Y)

Input: A solid or variable pattern P of type X and a variable text T of type Y

Output: All positions j such that a string in $\mathcal{L}(P)$ occurs in T ending at $T[j]$

Strings X and Y can be any of the types of variable strings mentioned above. Following the existing literature [53], the output of MATCH only specifies the ending segments of the occurrences, giving no information about the specific positions within the segments. In particular, if multiple occurrences of the pattern end within the same text segment, MATCH reports the segment only once. This convention is justified by the fact that the size of the language of a variable string, and hence the number of occurrences of a pattern, can be exponential in the input size. We say that MATCH has *constant alphabet* if the alphabet Σ of both the pattern and the text is of constant size.

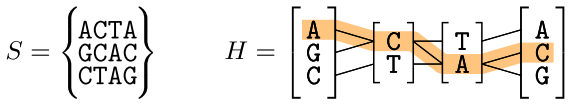

Fig. 3 A *k-D* string S and the 1-*F* graph H obtained applying the transformation of Remark 1. $\mathcal{L}(S)$ consists of the three length-4 strings in its only segment; $\mathcal{L}(H)$ consists of the 8 length-4 strings *ACTA*, *ACAC*, *ACAG*, *GCTA*, *GCAC*, *GCAG*, *CTAC*, *CTAG*, thus it contains 5 spurious strings with respect to $\mathcal{L}(S)$, including the highlighted one

In the rest of this section, we summarize the results that we will prove in Sections Matching a solid pattern in a *GD* or *F* text and Matching a solid pattern in a *k-D* or *k-F* text, consisting of upper and lower bounds for the problem MATCH(X, Y) for several choices of pattern type X and text type Y. We recall that we denote by m and M, respectively, the length and the size of the pattern; by n and N, respectively, the length and the size of the text; and by E the edges of an (elastic) founder graph text.

Upper bounds

In Section Matching a solid pattern in a *k-D* or *k-F* text, we give two truly sub-quadratic algorithms for MATCH(X, Y) when the pattern is solid and the text is either a *k-D* string or a *k-F* graph.

Theorem 1 *MATCH(SOLID,k-D) can be solved in* $\mathcal{O}(N + kn\log^2(\frac{m}{k})) = \mathcal{O}(N + N\log^2 m)$ *time.*

Theorem 2 *MATCH(SOLID,k-F) can be solved in* $\mathcal{O}(\sqrt{m}(|E| + N + n\log^2 m))$ *time.*

In Section Matching a solid pattern in a *GD* or *F* text, we consider the cases where the pattern is solid and the text is either a *GD* string or an *F* graph. In these cases, a mn term appears in the time complexity of the algorithms we propose. Consequently, these algorithms are sub-quadratic only when $N/n = \omega(1)$, and it is an open problem whether strongly sub-quadratic algorithms exist when $N/n = \mathcal{O}(1)$.

Theorem 3 *MATCH(SOLID,GD) can be solved in* $\mathcal{O}(mn + N)$ *time.*

Theorem 4 *MATCH(SOLID,F) can be solved in* $\mathcal{O}(mn + N + |E|)$ *time.*

Lower bounds

When both the pattern and the text are variable strings, we show conditional lower bounds for MATCH(X, Y). The quadratic conditional lower bound MATCH(1-*D*, 1-*D*) given in [60], which we discuss in more detail in Section Matching a variable pattern in a *k-D* or *k-F* text, holds only for non-constant alphabets, while a

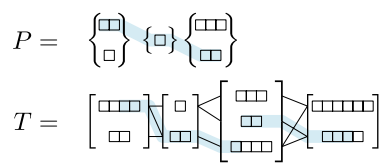

Fig. 4 Representation of two possible occurrences of an *ED* string P into an *EF* graph T

truly subquadratic algorithm exists for constant-size alphabets. All the other lower bounds we present hold even for constant-size alphabets. However, the reductions underlying Theorems 5 and 6 and Corollaries 1 and 2 only work when the length k of the strings in every segment is $\omega(\log n)$, where n is the length of the text, thus they do not rule out the existence of subquadratic-time algorithm when $k = \mathcal{O}(\log n)$. This is in contrast with the rest of the reductions, which all apply even when $k = \mathcal{O}(1)$ and thus are, in this sense, *stronger*. In particular, Theorem 7 proves that MATCH(k-*D*,k-*D*) requires at least quadratic time even when $k = 2$. The conditional lower bounds we prove rely on a famous conjecture: the *Orthogonal Vectors Hypothesis* (OVH), which is, in turn, implied by the Strong Exponential Time Hypothesis (SETH) [61, 89]. See Conjecture 1 in Section Matching a variable pattern in a *k-D* or *k-F* text.

Theorem 5 *No algorithm can solve* MATCH(1-*D*,k-*D*) *on constant alphabet for every k in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 6 *No algorithm can solve* MATCH(k-*D*,1-*D*) *on constant alphabet for every k in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 7 *No algorithm can solve* MATCH(2-*D*,2-*D*) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 8 *No algorithm can solve* MATCH(1-*D*,1-*F*) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 9 *No algorithm can solve* MATCH(1-*F*,1-*D*) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 10 *No algorithm can solve* MATCH(1-*F*,1-*F*) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Theorem 11 *No algorithm can solve* MATCH(SOLID,*ED*) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, for any ED string such that the difference in the lengths of the strings in any segment is at most* 1, *and the length of all such strings is at most* 4, *unless* OVH *is false.*

Since a pattern of type 1-*D* is a special case of k-*D*, *GD* and *ED* (and k-*D* is a special case of *GD* and *ED*), and since 1-*F* is a special case of k-*F*, *F*, and *EF*, the lower bounds above propagate along the taxonomies for *P* and/or *T*. Therefore, we have the following corollaries. Corollaries 1 and 2 directly follow from Theorems 5 and 6, thus only hold for the general cases where the maximum length of the strings in the segments is $\omega(\log n)$; in contrast, Corollary 3 follows from Theorem 7, thus it applies even to the restricted case where the maximum length of any string in any segment is 2.

Corollary 1 *No algorithm can solve* MATCH(1-*D*,*Y*) *for $Y = k$-*D*, *GD*, *ED on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Corollary 2 *No algorithm can solve* MATCH(*X*,1-*D*) *for $X = k$-*D*, *GD*, *ED on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Corollary 3 *No algorithm can solve* MATCH(*X*,*Y*) *for $X =$ k-*D*, *GD*, *ED and $Y =$ k-*D*, *GD*, *ED on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Corollaries 4, 5 and 6 below follow from Theorems 8, 9 and 10, respectively.

Corollary 4 *No algorithm can solve* MATCH(*X*,*Y*) *for $X =$ 1-*D*, k-*D*, *GD*, *ED and $Y =$ 1-*F*, k-*F*, *F*, *EF on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Corollary 5 *No algorithm can solve* MATCH(*X*,*Y*) *for $X =$ 1-*F*, k-*F*, *F*, *EF and $Y =$ 1-*D*, k-*D*, *GD*, *ED on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Corollary 6 *No algorithm can solve* MATCH(*X*,*Y*) *for $X =$ 1-*F*, k-*F*, *F*, *EF and $Y =$ 1-*F*, k-*F*, *F*, *EF on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Matching a solid pattern in a *GD* or *F* text

We start by formally defining an occurrence of a solid pattern in a *GD* string or an *F* graph. Let $T^{\ell,r}[i\,..\,j]$ be obtained from $T[i\,..\,j]$ by removing a prefix of length ℓ from all the strings of $T[i]$ and a suffix of length r from all the strings of $T[j]$, for some $\ell < k_i$ and $r < k_j$ (an example is in Figure 5).

Definition 1 Let T be a *GD* string or an *F* graph. A solid pattern P has an occurrence ending at position j in T if $\exists i \in [1, j]$ s.t. there exist $\ell \in [0, k_i - 1]$ and $r \in [0, k_j - 1]$

such that $P \in \mathcal{L}(T^{\ell,r}[i\,..\,j])$. We call ℓ and $k_j - r$ the starting and ending *offset* of the occurrence, respectively.

In Theorem 3, we show that the $\mathcal{O}(m^2n + N)$-time algorithm proposed in [53] for matching a solid pattern of length m in an *ED* text of length n and size N requires only $\mathcal{O}(mn + N)$ time when applied to *GD* texts.

Theorem 3 *MATCH*(SOLID,*GD*) *can be solved in* $\mathcal{O}(mn + N)$ *time.*

Proof Consider the pattern-matching algorithm of [53] for *ED* texts. The algorithm first constructs the suffix tree $\mathcal{T}_P$ of the pattern P in a preprocessing step; then it reads the text T segment by segment, from left to right. For any segment $T[i]$ of width k_i, the algorithm solves the following 4 sub-problems: (i) If $k_i \geq m$, does P occur in a string from $T[i]$? (*easy case*) (ii) Compute every suffix of a string $t \in T[i]$ that matches a prefix of P (*suffix case*); (iii) Compute every prefix of a string $t \in T[i]$ that matches a suffix of P (*prefix case*); (iv) If $k_i < m$, find the strings $t \in T[i]$ that are factors of P (*anchor case*). All occurrences of P are then computed from the solutions to these four problems. The easy and the suffix/prefix case can be solved both for *ED* and *GD* texts in $\mathcal{O}(N + m)$ time by using any standard pattern-matching algorithm (e.g. KMP [63]) and $\mathcal{T}_P$, respectively (see [53] for details).

We now focus on the anchor case for a segment $T[i]$ and show that, in the case of *GD* texts, it can be solved in time $\mathcal{O}(N_i + m)$ by solving an instance of the *Active Prefixes* (AP) problem [16]. AP applied to segment $T[i]$ takes as input a bit-vector U of length m s.t. $U[j] = 1$ iff the prefix $P[1\,..\,j]$ matches a suffix of a string from $\mathcal{L}(T[1\,..\,i-1])$; it outputs a bit-vector U of size m such that $U[j + k_i] = 1$ iff $U[j] = 1$ and $P[j + 1\,..\,j + k_i] \in T[i]$ (in other words, some string from $T[i]$ occurs in P starting at position $j + 1$: see Example 1). To solve AP it thus suffices to compute all the occurrences of all the strings of $T[i]$ in P and check whether they extend some active prefixes stored in U, namely, if they start at some position $j + 1$ with $U[j] = 1$. The strings in $T[i]$ are all of the same length k_i, thus no two of them can occur at the same position in P. We have the following crucial observation.

Observation 1 The cumulative number of occurrences in P of all the strings from a *GD* segment $T[i]$ is bounded by m.

Observation 1 implies that all such occurrences can be computed and stored in an auxiliary bit-vector OCC_i of size m in $\mathcal{O}(N_i + m)$ time, using $\mathcal{T}_P$. U can then be obtained by left-shifting OCC_i by one position, taking its bit-wise AND with U, and shifting the result vector by k_i positions to the right. Solving the anchor case for every segment thus takes $\mathcal{O}(N + mn)$ time. Note that the difference in the time complexities of *AP* for *GD* texts and *ED* texts is because, in the latter case, Observation 1 does not hold, since the number of occurrences of the strings from $T[i]$ in P can only be bounded by m^2. □

Example 1 Consider T and P as in Figure 5. The input of AP applied to $T[2]$ is $U = [1, 1, 0, 0, 0]$; the output is $V = [0, 1, 1, 0, 0]$. This is because the prefix of length 1 of P, which is a suffix of the first string of $T[1]$, and the prefix of length 2 of P, which is a suffix of the second string of $T[1]$, can be extended by string b of $T[2]$.

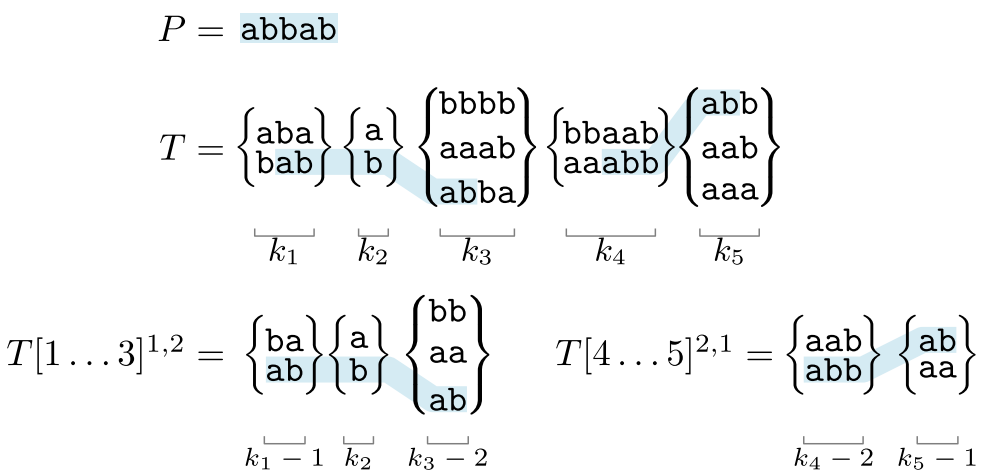

Fig. 5 Two occurrences of a solid pattern in a *GD* string. The starting offset of the leftmost occurrence is 1, and its ending offset is $k_3 - 2 = 2$; the starting and ending offsets of the second occurrence are 2 and $k_5 - 1 = 2$, respectively

We next adapt the algorithm of Theorem 3 to solve pattern matching on F graphs.

Theorem 4 *MATCH*(SOLID,F) *can be solved in* $\mathcal{O}(mn + N + |E|)$ *time.*

Proof Let $G = (T, E)$ be a founder graph of length n, size N and cardinality B, and let P be a solid pattern of length m. We denote by $T[i][j]$ the j-th string of $T[i]$, $j \in [1, B_i]$ (recall that B_i is the cardinality of segment i, and their sum over all i gives the total number B of strings in the segments of T), assuming any fixed order; an edge $(j, j') \in E_{i-1}$ thus connects string $T[i-1][j]$ to $T[i][j']$. We process separately the occurrences of P that span only one segment (easy case), only two segments, or more. The easy case is trivially solved in $\mathcal{O}(m + N)$ time using any linear-time pattern-matching algorithm. Let us now focus on the other two cases.

Analogously to the suffix and prefix subproblems listed in the proof of Theorem 3 we precompute, for each string in each segment, the length of all suffixes that are equal to some prefix of P, and of all prefixes that are equal to some suffix of P. While in the case of *GD* text, it suffices to store the lengths of all such overlaps cumulatively for each segment, in the case of founder graphs, due to the presence of edges, we need to retain this information separately for each string in each segment. We thus compute two binary arrays $b_{i,j}$ and $e_{i,j}$ for each string $T[i][j]$, each of the same length k_i, s.t. $b_{i,j}[\ell] = 1$ if and only if the suffix of length ℓ of $T[i][j]$ is equal to a prefix of P, and $e_{i,j}[\ell] = 1$ if and only if the prefix of length ℓ of $T[i][j]$ is equal to a suffix of P. All such arrays can be constructed in $\mathcal{O}(m + N)$ total time using e.g. the suffix trees of P and of its reversal, and occupy total space $\mathcal{O}(N)$. See Figure 6.

Occurrences spanning only two segments. We consider all pairs of consecutive segments $T[i], T[i+1]$ such that $k_i + k_{i+1} \geq m$ (the only candidates for occurrences of this kind). For each edge $(j, j') \in E_i$, let p_j be the length of the longest suffix of $T[i][j]$ that overlaps a prefix of P, i.e. the largest index of $b_{i,j}$ set to 1, and let $s_{j'}$ be the length of the longest prefix of $T[i+1][j']$ that overlaps a suffix of P, i.e. the largest index of $e_{i+1,j'}$ set to 1. The following observation characterises the occurrences of P spanning $T[i], T[i+1]$ (see also Example 2).

Observation 2 P has an occurrence spanning $T[i], T[i+1]$ iff there exists an edge $(j, j') \in E_i$ such that P occurs in the concatenation of the suffix of length p_j of $T[i][j]$ and the prefix of length $s_{j'}$ of $T[i+1][j']$, which are equal, respectively, to $P[1..p_j]$ and $P[m - s_{j'} + 1..m]$.

The main tool we use to spot these occurrences is a data structure to efficiently answer the following queries: given any pair of positions $1 \leq s \leq p \leq m$, return the set *occ* of occurrences of P in the concatenation $P[1..p]P[s..m]$ of its prefix of length p and suffix of length $m - s + 1$. A classical data structure to answer these queries is the *border tree* [56]; recently, Pissis [75] introduced a data structure alternative to border trees that can be constructed in $\mathcal{O}(m/\log_\sigma m)$ time and answers queries in $\mathcal{O}(1 + |occ|)$ time. After constructing the data structure of [75] for P, we thus process each pair $T[i], T[i+1]$ such that $k_i + k_{i+1} \geq m$, and apply the following algorithm, whose correctness follows from Observation 2: for each $(j, j') \in E_i$, query the border data structure with indices $p_j, m - s_{j'} + 1$. If the returned set *occ* is nonempty, break and return an occurrence of P in G ending at position $i + 1$. Each such query takes $\mathcal{O}(1 + |occ|)$ time [75], and $|occ| \leq m$. Since we stop asking queries as soon as we find a nonempty set of occurrences, the total time for $T[i], T[i+1]$ is $\mathcal{O}(|E_i| + m)$, implying time $\mathcal{O}(|E| + mn)$ to process the whole G.

Occurrences spanning at least three segments. This case is analogous to the anchor case of Theorem 3, and can only happen when the second of the (at least three) segments has a width smaller than m. We process the segments of G from left to right and maintain an array V of size m that keeps track of the prefixes of P that match up to a certain segment (*partial occurrences*): however, now V is an array of integers, rather than a simple bit-vector, and it only keeps track of partial occurrences that span at least one full segment (thus excluding prefixes of P that match a proper suffix of some string from some segment:

$P =$ abbabbabbabb

$G =$ [aaaaabbabb, abbabbabba, aaaabbabba] [bbabba, abbabb] [bbab, abba]

$k_1 = 10$ $k_2 = 6$ $k_3 = 4$

$b_{1,1} = [0, 0, 1, 0, 0, 1, 0, 0, 0, 0]$ $e_{2,1} = [1, 1, 0, 1, 1, 0]$
$b_{1,2} = [1, 0, 0, 1, 0, 0, 1, 0, 0, 1]$ $e_{2,2} = [0, 0, 1, 0, 0, 1]$
$b_{1,3} = [1, 0, 0, 1, 0, 0, 1, 0, 0, 0]$ $e_{3,1} = [1, 1, 0, 0]$
$b_{2,1} = [1, 0, 0, 1, 0, 0]$ $e_{3,2} = [0, 0, 1, 0]$
$b_{2,2} = [0, 0, 1, 0, 0, 1]$

Fig. 6 Arrays $b_{i,j}$ and $e_{i,j}$ (used in the proof of Theorem 4) for an F graph of length 3 and a solid pattern of length 12. Highlighted in blue is a suffix/prefix overlap of length 3 and the corresponding entry of $b_{1,1}$; in green, a prefix/suffix overlap and the corresponding entry of $e_{3,1}$. Arrays $e_{1,\cdot}$ and $b_{3,\cdot}$ are not shown as they may not start or end any occurrence of P

these shorter partial occurrences will be treated differently, as we explain in the following).

Let V_{i-1} denote the state of V after processing a segment $T[i-1]$ ($V_{i-1} = 0^m$ if $k_{i-1} > m$ or $i = 1$). When processing segment $T[i]$ (assuming $k_i < m$), our task is to compute the next state V_i s.t. $V_i[\ell] = j$ iff $P[1..\ell]$ is a suffix of some string from $\mathcal{L}(G[1..i])$ that ends with the whole string $T[i][j]$, and $V_i[\ell] = 0$ otherwise. Note that the values of V_i are uniquely defined: see Observation 3. In particular, this implies that the first $k_i - 1$ positions of V_i, corresponding to partial occurrences that do not contain an entire string from $T[i]$, are always 0. Further, note that positions between k_i and $\min\{k_i + k_{i-1} - 1, m\}$ of V_i correspond to partial occurrences of P that contain an entire string from $T[i]$ but not from $T[i-1]$. We compute $V_i[k_i..k_i + k_{i-1} - 1]$ and $V_i[k_i + k_{i-1}..m]$ using two different procedures (the second sub-array is empty in the case $k_i + k_{i-1} > m$).

To process $T[i]$, we first compute an array OCC_i of size m such that $OCC_i[\ell] = j$ iff $T[i][j]$ occurs in P starting at position ℓ; by Observation 1, OCC_i is well defined and can be computed in $\mathcal{O}(N_i + m)$ time using the suffix tree $\mathcal{T}_P$. Note that OCC_i contains all potential extensions of partial occurrences of P *with whole strings* from $T[i]$. We use the following crucial observation, which is a direct consequence of Observation 1.

Observation 3 Any partial occurrence of P ending at $T[i-1]$ can be extended by at most one occurrence of one string from $T[i]$.

In particular, the partial occurrence represented by $V_{i-1}[\ell] = j$ can only be extended by the unique string from $T[i]$ occurring in P at position $\ell + 1$, if any. This implies that it suffices to check the following necessary and sufficient conditions to compute $V_i[k_i + k_{i-1}..m]$: for each position $\ell \in [k_i + k_{i-1}, m]$, we set $V_i[\ell] = j'$ iff $OCC_i[\ell - k_i + 1] = j'$, $V_{i-1}[\ell - k_i] = j$ and $(j, j') \in E_{i-1}$. To verify these conditions, collect all triplets (j, j', ℓ) such that $V_{i-1}[\ell - k_i] = j$ and $OCC_i[\ell - k_i + 1] = j'$, for all $\ell \in [k_i + k_{i-1}, m]$, and sort them lexicographically together with all the pairs from E_i, using radix sort. Then, scan the resulting sorted list and set $V_i[\ell] = j'$ iff a triplet (j, j', ℓ) is immediately preceded by the pair (j, j'). Since there is at most one triplet for each value of ℓ, this requires $\mathcal{O}(m + |E_i|)$ total time per segment and thus $\mathcal{O}(mn + |E|)$ time over the whole G.

We set $V_i[k_i] = j$ if and only if $T[i][j]$ occurs in P starting at position 1. Let us now focus on computing $V_i[k_i + 1..k_i + k_{i-1} - 1]$. This portion of V_i corresponds to partial occurrences of P matching a proper suffix of some string from $T[i-1]$ and extended with an occurrence of some string from $T[i]$ stored in $OCC_i[2..k_{i-1}]$. Note that we cannot use the same technique as for $V_i[k_i + k_{i-1}..m]$, because two strings from $T[i-1]$ can have equal suffixes. Instead, we scan $OCC_i[2..k_{i-1}]$: if $OCC_i[\ell] = j' \neq 0$, we check, for all $(j, j') \in E_{i-1}$, whether $b_{i-1,j}[\ell - 1] = 1$, and set $V_i[\ell + k_i - 1] = j'$ if this is the case. In other words, we check, for each occurrence of $T[i][j']$ starting at $P[\ell]$, whether the suffix of length $\ell - 1$ of some of the strings from $T[i-1]$ connected to $T[i][j']$ is equal to $P[1..\ell - 1]$. See also Example 3. This procedure has a total cost $\mathcal{O}(N_{i-1} + m)$ because each position of each $b_{i-1,j}$ is accessed at most once; and each time we read an edge, we access exactly one position of one array b, thus we read at most N_{i-1} edges. This implies a total time $\mathcal{O}(N + mn)$ over all segments.

Finally, to check whether some partial occurrence represented by $V_{i-1}[\ell] = j$ for $\ell \in [m - k_i + 2..m]$ can be extended to a full occurrence of P with a proper prefix of some string from $T[i]$, it suffices to check, for each edge $(j, j') \in E_{i-1}$, whether $e_{i,j'}[m - \ell + 1] = 1$. This requires $\mathcal{O}(N + mn)$ total time because each position of each array e is accessed at most once. We obtain a total time complexity of $\mathcal{O}(N + mn + |E|)$. □

Example 2 Consider the instance of Figure 6. To find occurrences of P spanning the first two segments only, we consider one edge (j, j') at a time, compute the lengths p_j and $s_{j'}$ from $b_{1,j}$, and $e_{2,j'}$ and check whether P occurs in the concatenation of $P[1..p_j]$ and $P[m - s_{j'} + 1..m]$, as described in the proof of Theorem 4. Consider the edge (1, 1), connecting $T[1][1]$ and $T[2][1]$. We have $p_1 = 6$ (as this is the largest index of $b_{1,1}$ set to 1, i.e. the length of the longest prefix of P which is also a suffix of $T[1][1]$) and $s_1 = 5$ (the largest index of $e_{2,1}$ set to 1). P does not occur in the concatenation of $P[1..6] =$ `abbabb` and $P[8..12] =$ `bbabb`, thus no occurrence of P uses edge (1, 1).

Now consider edge (2, 1). The longest suffix of $T[1][2]$ that is also a prefix of P is of length $p_2 = 10$; and the longest suffix of P that is also a prefix of $T[2][1]$ has length $s_1 = 5$. P occurs in the concatenation of $P[1..10]$ and $P[8..12]$, i.e. `abbabbabba`$\cdot$ `bbabb`, thus an occurrence of P ending at segment $T[2]$ is reported. The algorithm will not examine the rest of the edges; therefore, although the pattern also occurs in the concatenation $P[1..7] \cdot P[8..12] =$ `abbabba`$\cdot$ `bbabb`, corresponding to edge (3, 1), this extra occurrence will not count towards the complexity of the algorithm.

Example 3 Consider again the instance in Figure 6, and focus on the occurrences of P spanning the three segments, which are computed as described in the proof of

Theorem 4. We have $V_1 = [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]$ as the whole string $T[1][2]$ matches the prefix of length 10 of P. We know that the first $k_2 - 1 = 5$ positions of V_2 are 0, as they correspond to (possible) partial occurrences of P that do not contain a whole string from $T[2]$. We have $OCC_2 = [2, 1, 0, 2, 1, 0, 2, 0, 0, 0, 0, 0]$ because $T[2][1]$ occurs in P starting at positions 2 and 5 and $T[2][2]$ occurs in P starting at positions 1, 4 and 7. Since $k_1 + k_2 = 15 > m = 12$, no occurrence of P can contain an occurrence of a whole string from $T[1]$ *and* from $T[2]$. Since $OCC_2[1] = 2$, we set $V_2[6] = 2$. To compute $V_2[7\,..\,12]$, we scan $OCC_2[2\,..\,10]$ from left to right. Reading $OCC_2[2] = 1$, we need to check the edges incoming to the first string of $T[2]$ – in this case, edges (1, 1), (2, 1) and (3, 1) – and access the first position of the corresponding array b. For edge (1, 1), we have $b_{1,1}[1] = 0$, thus we do not update any value of V_2; for edge (2, 1), we have $b_{1,2}[1] = 1$, thus we set $V_2[7] = 1$ to indicate that there is a partial match of $P[1\,..\,7]$ ending with an occurrence of the whole string $T[2][1]$. The same information is also captured by the edge (3, 1) because $b_{1,3}[1] = 1$. $OCC_2[3] = 0$ does not trigger any update of V_2; reading $OCC_2[4] = 2$, we check the only edge incoming to $T[2][2]$, which is (2, 2): since $b_{1,2}[3] = 0$, no value of V_2 is updated. Reading $OCC_2[5] = 1$, we check again edges (1, 1), (2, 1) and (3, 1), find out that $b_{1,2}[4] = 1$, and thus set $V_2[10] = 1$. The rest of the positions of OCC_2 do not cause any further update of V_2, which at the end of this procedure is $V_2 = [0, 0, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0]$.

Matching a solid pattern in a *k-D* or *k-F* text

In this section, we investigate the complexity of the pattern-matching problem in cases where the pattern is solid and the text is either a k-D string or a k-F graph.

The problem of finding all the occurrences of a solid pattern of length m in a 1-D string of length $n > m$ and size N can be seen as an instance of the *Subset Matching* problem, defined by Cole and Hariharan [25]. This problem asks, given a 1-D text and a 1-D pattern, to find all the locations in the text where each segment of the pattern is a subset of the corresponding segment of the text; an $\mathcal{O}(n \log^2 m)$-time deterministic algorithm exists [26][2]. In this section, we leverage this result to prove sub-quadratic upper bounds for MATCH(SOLID, 1-F), MATCH(SOLID,k-D) and MATCH(SOLID,k-F) by reducing each of these problems to several instances of MATCH(SOLID,1-D).

Theorem 1 *MATCH(SOLID,k-D) can be solved in $\mathcal{O}(N + kn \log^2(\frac{m}{k})) = \mathcal{O}(N + N \log^2 m)$ time.*

Proof Let P be a solid pattern of length m and T a k-D string of length n, cardinality B and total size N. In a preprocessing step, we compute all suffix/prefix and prefix/suffix overlaps of P and each string in the vocabulary of T. More precisely, for each segment $T[i]$ we compute two binary arrays b_i and e_i of length k such that $b_i[j] = 1$ if and only if a suffix of length j of one of the strings in $T[i]$ is equal to a prefix of P, and $e_i[j] = 1$ if and only if a a prefix of length j of one of the strings in $T[i]$ is equal to a suffix of P. The arrays b_i and e_i occupy $\mathcal{O}(kn) = \mathcal{O}(N)$ words of space and can be computed in $\mathcal{O}(N)$ time by e.g. building the generalised suffix tree of all the strings in all segments of T.

Consider the case $m \geq 2k - 1$, which implies that each occurrence of P contains at least 1 full string from some $T[i]$. Let P_k denote the set of length-k substrings of P and let $h(s)$ denote the lexicographic[3] rank of $s \in P_k$, to be used as a unique ID for s. The values $h(s)$ can be computed and stored in $\mathcal{O}(m)$ time and space by constructing the suffix tree of P and annotating the nodes at string depth k (possibly making them explicit) with their rank, obtained with a lexicographic traversal of the tree.

Using these values, we construct k instances of MATCH(SOLID,1-D), one per each possible starting offset of occurrences of P in T. The pattern $P^{(\ell)}$ of the ℓ-th instance, $\ell \in [0, k-1]$, is obtained from P by splitting it into consecutive non-overlapping fragments of length k: the first fragment starts at position $\ell + 1$ and the possible remaining suffix of length $m - \ell - k\lfloor\frac{m-\ell}{k}\rfloor$ is discarded. Each of such fragments is then replaced by its ID. The length of $P^{(\ell)}$ is thus $\lfloor\frac{m-\ell}{k}\rfloor$, and the cumulative space occupied by all such instances for all $\ell \in [0, k-1]$ is $\mathcal{O}(m)$. The text $T^{(\ell)}$ of the ℓ-th instance is obtained from T by replacing each string $s \in T[i]$ by its ID, for all $i \in [1, n]$, as follows. If $s \in P_k$ and $h(s)$ occurs in $P^{(\ell)}$, then s is replaced by $h(s)$; otherwise, (thus if $h(s)$ does not occur in $P^{(\ell)}$ or s does not occur in P), s is discarded.[4] Each $T^{(\ell)}$ has length n and total size $\mathcal{O}(\min\{B, n\frac{m}{k}\})$, therefore they collectively occupy $\mathcal{O}(kB) = \mathcal{O}(N)$ space. See Figure 7 for an example of this construction.

[2] The upper bound explicitly proved by the authors in the cited paper is $\mathcal{O}(N \log^2 N)$; however, some observations made by the same authors in [25, Section 4] that lead to better bounds apply, and indeed the authors state that Subset Matching can be solved deterministically in $\mathcal{O}(n \log^2 m)$ time both in the abstract of [26] and in their later work [27].

[3] Note that any other total order would also work for this purpose.

[4] Note that, applying this procedure, some of the segments of $T^{(\ell)}$ might be empty. To avoid so, one can use a special symbol \$ different from all the IDs $h(s)$ and place it in the otherwise empty segments of $T^{(\ell)}$.

The k patterns can be constructed all at the same time by traversing the suffix tree of P in $\mathcal{O}(m)$ total time. The k texts can be constructed simultaneously by using the suffix tree of P, further annotated as follows. At each node v at string depth k (which are all and only the nodes annotated with some ID $h(s)$), we store the list of values ℓ such that $\ell = i \mod k$ for some leaf i descending from v (that corresponds to the suffix starting at position i). Thus ℓ is in the list of v if and only if the length-k substring corresponding to v is one of the fragments P is split into to build $P^{(\ell)}$. Since these nodes partition the leaves, and the number of descending leaves bounds the size of each list, such lists collectively occupy $\mathcal{O}(m)$ space and can be computed in $\mathcal{O}(m)$ time with a single tree traversal. Armed with this annotated suffix tree, to construct the k texts we simply search each string of T in the tree and write the value $h(s)$ (if any) in the corresponding segment of all texts $T^{(\ell)}$ such that ℓ is in the list of the reached node. This procedure thus requires $\mathcal{O}(N)$ total time.

We now show that each occurrence of P in T that fully contains at least a string from a segment of T (which is always the case when $m \geq 2k-1$) corresponds to an occurrence of some $P^{(\ell)}$ in $T^{(\ell)}$ for some $\ell \in [0, k-1]$. We treat the other occurrences separately, as we will

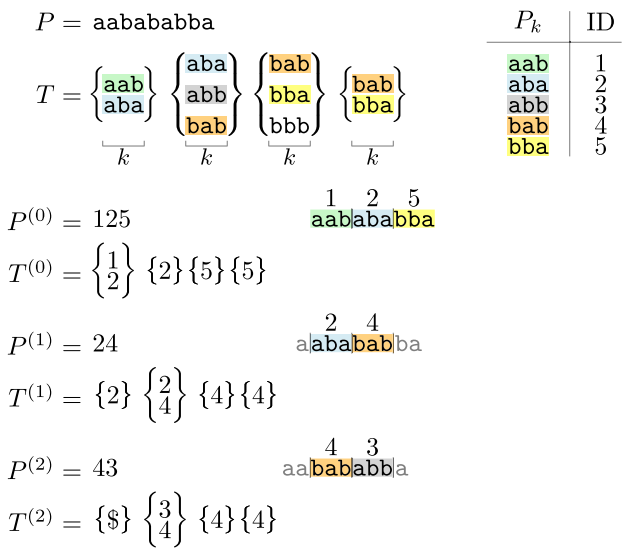

Fig. 7 Illustration of the reduction described in the proof of Theorem 1. Note that P occurs twice in T, ending at $T[3]$ with starting offset 0, and ending at $T[4]$ with starting offset 2. The first occurrence corresponds to an occurrence of $P^{(0)}$ in $T^{(0)}$ ending at $T^{(0)}[3]$. The second can be retrieved by the occurrence of $P^{(1)}$ in $T^{(1)}$ ending at $T^{(1)}[3]$ using the pre-computed arrays $b_1 = [1, 0, 1]$ and $e_4 = [0, 1, 0]$: indeed, $b_1[1] = 1$ and $e_4[2] = 1$, indicating that the suffix of length 1 of some string from $T[1]$ is equal to a prefix of P and a prefix of length 2 of some string from $T[4]$ is equal to a suffix of P, respectively, which complete the occurrence

detail at the end of the proof. The key observation is that if an occurrence of P in T starts at offset $q = k - \ell + 1$ in $T[i]$ and ends at offset r in $T[j]$, then the following hold: (i) a string from each of the segments $T[i+1], \ldots, T[j-1]$ occurs consecutively in P starting from position $\ell + 1$; (ii) a prefix of length ℓ of P matches a suffix of some string in $T[i]$; and (iii) a suffix of length r matches a prefix of some string in $T[j]$. Observation (i) implies, by construction, an occurrence of $P^{(\ell)}$ in $T^{(\ell)}$ starting at position $i+1$; moreover, Observations (ii) and (iii) imply that $b_i[\ell] = e_j[r] = 1$. This gives us the following algorithm. For each possible offset $\ell = 0, 1, \ldots, k-1$:

1. Find the occurrences of $P^{(\ell)}$ in $T^{(\ell)}$
2. For each occurrence $T^{(\ell)}[i..j]$ (note that $j = i - 1 + \lfloor \frac{m-\ell}{k} \rfloor$), check if it corresponds to an occurrence of P in T by checking whether $b_{i-1}[\ell] = e_{j+1}[r] = 1$ and report an occurrence $T[i-1..j+1]$ if this condition holds.

Note that when $\ell = 0$ we only need to check whether $e_{j+1}[r] = 1$ and the corresponding occurrence is $T[i..j+1]$; and symmetrically, if $r = 0$, we only check if $b_{i-1}[\ell] = 1$, the occurrence being $T[i-1..j]$. For a fixed ℓ, Step (1) can be done in $\mathcal{O}(n \log^2(\frac{m}{k}))$ time using the algorithm from [26] for subset matching; and Step (2) requires $\mathcal{O}(1)$ time per occurrence. Since each $P^{(\ell)}$ can occur in at most n positions, the total time for step (2) over all $\ell = 0, 1, \ldots, k-1$ is $\mathcal{O}(kn) = \mathcal{O}(N)$; and the total time for Step (1) is $k \cdot \mathcal{O}(n \log^2(\frac{m}{k}))$.

Finally, we can find all the occurrences of P in T that do not fully contain a string from some segment (which can only happen if $m < 2k-1$) in $\mathcal{O}(N)$ total time. These occurrences either (i) span exactly two consecutive segments of T, or (ii) are entirely contained in some string of some segment. To find all occurrences of type (ii) in $\mathcal{O}(N)$ time it suffices to run e.g. KMP [63]. To find the occurrences of type (i), we scan each array b_i: for each $j \in [1, k]$ s.t. $b_i[j] = 1$, we check whether $e_{i+1}[m-j] = 1$ and report an occurrence ending in segment $i+1$ if this is the case. This requires $\mathcal{O}(kn)$ total time for all $i \in [1, n-1]$. □

Let us now consider the case where the text is a k-F graph. Let P be a solid string of length m over an alphabet Σ, let ℓ be an integer which divides m, and let us write $P = P_1..P_d$ where $|P_i| = \ell$ for $i = 1..d$. We define the ℓ *th spread* of P as the string $\mathbf{spr}^{\ell}(P) = \prod_{i=1}^{d-1} P_i \cdot \$ \cdot P_{i+1}$, where $\$ \notin \Sigma$. In other words, each fragment of length ℓ P_i is repeated twice, separated by a gadget letter \$, except for the first and last one.

Example 4 Let $P = \texttt{GTTCGTTATATG}$. One has $\mathbf{spr}^3(P) = \texttt{GTT} \cdot \$ \cdot \texttt{CGT CGT} \cdot \$ \cdot \texttt{TAT TAT} \cdot \$ \cdot \texttt{ATG}$. Let $P' = \texttt{CGTATTATT}$. One has $\mathbf{spr}^3(P') = \texttt{CGT} \cdot \$ \cdot \texttt{ATT ATT} \cdot \$ \cdot \texttt{ATT}$.

Given a *k-F* graph $G = (T, \bigcup_{i=1}^{n} E_i)$ of length n, we define its *disentanglement* as the $(2k+1)$-*D* string $\mathcal{D}(G) = D_1 \dots D_{n-1}$ where for every $i = 1 \dots n-1$, the set D_i contains every string $t \cdot \$ \cdot t'$ such that $(t, t') \in E_i$. An example is shown in Figure 8.

We prove the following lemma:

Lemma 1 *Given a k-F graph $G = (T, E)$ and a solid string P of length m such that k divides m, the string $\mathbf{spr}^k(P)$ occurs in $\mathcal{D}(G)$ if and only if there exists i, j with $P \in \mathcal{L}(G[i \dots j])$, namely P occurs in G and can be constructed as a concatenation of entire successive strings in $T[i] \dots T[j]$ that are connected by edges.*

Proof If $P \in \mathcal{L}(G[i \dots j])$ and $P = P_1 \dots P_d$ is a factorisation of P in substrings of length k, then for every $\ell = 1 \dots d-1$ one has $P_\ell \in T[i+\ell-1]$, $P_{\ell+1} \in T[i+\ell]$ and $(P_\ell, P_{\ell+1}) \in E_{i+\ell-1}$, which means that the set $\mathcal{D}(G)[i+\ell-1]$ contains the string $P_\ell \cdot \$ \cdot P_{\ell+1}$, and the concatenation of those strings for each ℓ is equal to $\mathbf{spr}^k(P)$. Conversely, observe that for every $\ell = 1 \dots d-1$, one has $\mathbf{spr}^k(P)[(\ell-1)(2k+1)+k+1] = \$$. If $\mathbf{spr}^k(P)$ occurs in $\mathcal{D}(G)$, since by construction the letter \$ occurs only at position $k+1$ of each set in $\mathcal{D}(G)$, the occurrence has to start at the first position of a set, and that means that $\mathbf{spr}^k(P) \in \mathcal{L}(\mathcal{D}(G)[i \dots j])$ for some $1 \le i \le j \le n$ (namely, it occurs with starting and ending offset 0). This implies that for such i, j and for each $1 \le \ell \le d-1$ one has $\mathbf{spr}^k(P)[(2k+1)(\ell-1) \dots (2k+1)\ell] \in \mathcal{D}(G)[i+\ell-1]$. But $\mathbf{spr}^k(P)[(2k+1)(\ell-1) \dots (2k+1)\ell] = P_\ell \$ P_{\ell+1}$, which means that $P_\ell \in T[i+\ell-1]$, $P_{\ell+1} \in T[i+\ell]$, and $(P_\ell, P_{\ell+1}) \in E_{i+\ell-1}$. Since this holds for every $1 \le \ell \le d-1$, we deduce that $P \in \mathcal{L}(G[i \dots j])$. □

Example 5 Let $P = \texttt{GTTCGTTATATG}$, and $P' = \texttt{CGTATTATT}$ as in Example 4, and let $G = (\tilde{T}, E)$ be the *k-F* graph from Figure 8. The pattern P occurs once in G, ending in the fourth segment (underlined occurrence in the figure). This corresponds to a unique occurence of $\mathbf{spr}^3(P) = \texttt{GTT} \cdot \$ \cdot \texttt{CGTCGT} \cdot \$ \cdot \texttt{TATTAT} \cdot \$ \cdot \texttt{ATG}$ in $\mathcal{D}(G)$, ending in the third segment. The string P' occurs twice in G (in red and in blue on the figure). However, the string $\mathbf{spr}^3(P') = \texttt{CGT} \cdot \$ \cdot \texttt{ATTATT} \cdot \$ \cdot \texttt{ATT}$ occurs only once in $\mathcal{D}(G)$ (in red). This is because the blue occurrence has nonzero starting and ending offsets.

As illustrated in the above example, Lemma 1 allows us, given a *k-F* G and a pattern P whose length is a multiple of k, to use a $(2k+1)$-*D* string to detect the occurrences

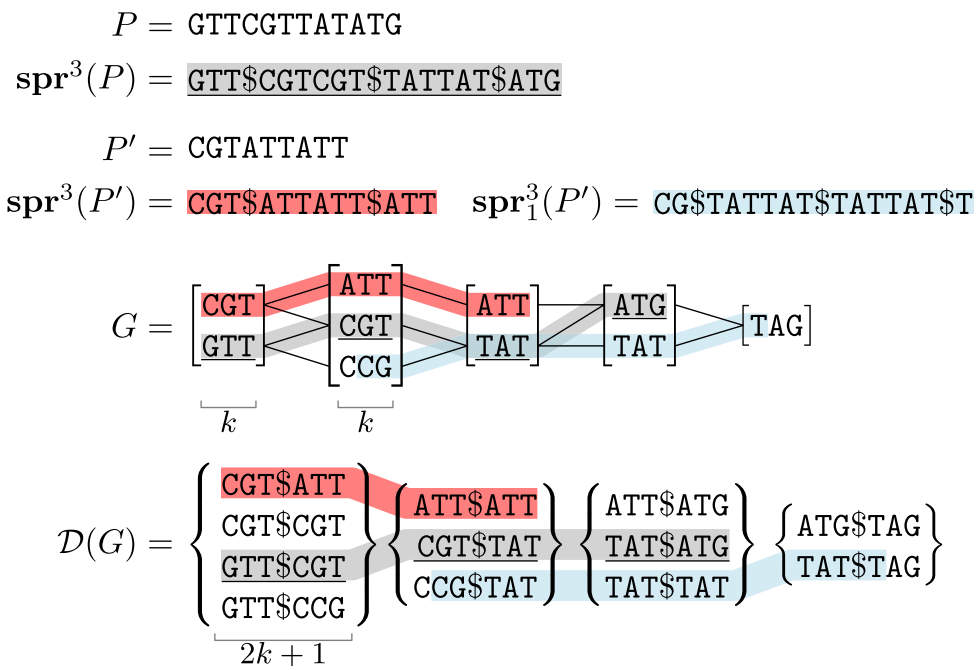

Fig. 8 A *k-F* graph G (top) and its disentanglement $\mathcal{D}(G)$ (bottom). Pattern $P = \texttt{GTTCGTTATATG}$ occurs once in G (underlined), and pattern $\mathbf{spr}^3(P)$ occurs once in $\mathcal{D}(G)$ (also underlined). Patterns $P' = \texttt{CGTATTATT}$ occur twice in G (in red and blue respectively), and each occurrence can be detected in $\mathcal{D}(G)$ as in Theorem 2 from the strings highlighted in red and blue

of P starting at the first position of a segment of G. We now extend this result to arbitrary pattern length and starting position, in order to prove Theorem 2.

Theorem 2 *MATCH(SOLID,k-F) can be solved in $\mathcal{O}(\sqrt{m}(|E| + N + n\log^2 m))$ time.*

Proof Let $G = (T, E)$ be a k-F graph of length n and $P = P[1..m]$ be a solid pattern. Let us first assume that $k < \sqrt{m}$. We construct the disentanglement of G, which is a $(2k+1)$-D string $\mathcal{D}(G)$. Let us assume that P occurs in G starting at position i and ending at position j (we have $i < j$ because $k < \sqrt{m}$ and thus every occurrence spans more than one segment). This means, by definition, that there exist strings s, t and $P_1, .., P_{j-i+1}$, with $0 \le |s|, |t| < k$, $0 < |P_1|, |P_{j-i+1}| \le k$ and $|P_2| = \ldots = |P_{j-i}| = k$, such that $s \cdot P_1 \in T[i]$, $P_{j-i+1} \cdot t \in T[j]$ and $P_{\ell-i+1} \in T[\ell]$ for every $i < \ell < j$, and such that $P_1 \cdots P_{j-i+1} = P$. The lengths of s and t are the starting and ending offsets of the occurrence, respectively, which uniquely determine the string $\mathbf{spr}^k_{|s|}(P) := P_1 \cdot \$ \cdot P_2 \cdot \mathbf{spr}^k(P_2 .. P_{j-i}) \cdot P_{j-i} \cdot \$ \cdot P_{j-i+1}$. Let $\hat{P} := s \cdot P \cdot t$. We have that $\hat{P} \in \mathcal{L}(G[i..j])$, which, by Lemma 1, means precisely that $\mathbf{spr}^k(\hat{P})$ occurs in $\mathcal{D}(G)$. We can rewrite $\mathbf{spr}^k(\hat{P}) = s \cdot \mathbf{spr}^k_{|s|}(P) \cdot t$, and we deduce that P occurs in G with starting offset $|s|$ in some segment if and only if $\mathbf{spr}^k_{|s|}(P)$ occurs in $\mathcal{D}(G)$ (observing the occurrences of \$ in the constructed strings, every occurrence of $\mathbf{spr}^k_{|s|}(P)$ is always part of an occurrence of $s \cdot \mathbf{spr}^k_{|s|}(P) \cdot t$, for some s, t of the right lengths). To find every occurrence of P in G, we can then search for the k patterns $\mathbf{spr}^k_0(P), .., \mathbf{spr}^k_{k-1}(P)$ in $\mathcal{D}(G)$.

Notice that $\mathcal{D}(G)$ has size $\mathcal{O}(k|E|)$. To search for $\mathbf{spr}^k_0(P), .., \mathbf{spr}^k_{k-1}(P)$ we can apply a modified version of the algorithm from Theorem 1. The difference is that, while in Theorem 1 we need k searches for a single pattern (one for each possible starting offset in the k-D string), here we are already given k distinct patterns, each encoding a different possible starting offset in G, and we must search for occurrences of each of such patterns in $\mathcal{D}(G)$ that start at the beginning of a segment and end at the end of another one. Hence, we only need to compute the IDs for the length-$(2k+1)$ substrings of $\mathbf{spr}^k_0(P), .., \mathbf{spr}^k_{k-1}(P)$ that contain a \$ at their $k+1$th position (as they are the only substrings that can match a full string from a segment of $\mathcal{D}(G)$). This can be done simultaneously for all $\mathbf{spr}^k_0(P), .., \mathbf{spr}^k_{k-1}(P)$ in $\mathcal{O}(m)$ total time by using the suffix tree of P. Indeed, each substring for which we need to compute an ID consists of a substring of P of length $2k$, with an extra central dollar. Similarly, we can compute the IDs of all the strings from $\mathcal{D}(G)$ (and discard those that do not occur in any of the patterns) in $\mathcal{O}(k|E|)$ total time.

In addition to the substring IDs, we construct arrays e_i and b_i in $\mathcal{O}(k|E|)$ time (corresponding to the total size of $\mathcal{D}(G)$), as in Theorem 1. The search takes $\mathcal{O}(n\log^2(\frac{m}{k}))$ time for each pattern; since each of the patterns is searched only once, we obtain a total time of $\mathcal{O}(k|E| + kn\log^2(\frac{m}{k})) = \mathcal{O}(\sqrt{m}(|E| + n\log^2 m))$ for the cases $k < \sqrt{m}$.

Now consider the case $k \ge \sqrt{m}$. Observe that, in this case, we have $n \le \frac{N}{\sqrt{m}}$, because it always holds that $n \le \frac{N}{k}$. By simply applying the algorithm of Theorem 4 in this special case we obtain a time complexity in $\mathcal{O}(N\sqrt{m} + |E|\log m)) = \mathcal{O}(\sqrt{m}(N + |E|))$. Combining the two cases, the total time becomes $\mathcal{O}(\sqrt{m}(|E| + N + n\log^2 m))$. □

Example 6 Consider again $P' =$ `CGTATTATT` as in Example 4 and the two occurrences in the F graph shown in Figure 8. We can now detect its blue occurrence in G by searching for $\mathbf{spr}^3_1(P') =$ `CG` $\cdot \$ \cdot$ `TATTAT` $\cdot \$ \cdot$ `TATTAT` $\cdot \$ \cdot$ `T`, that occurs in $\mathcal{D}(G)$.

Matching a variable pattern in a *k-D* or *k-F* text

In this section, we study the complexity of finding all the occurrences of non-solid patterns in a k-D or k-F text. We begin by formally extending Definition 1 to the more general case where both P and T are either GD strings or F graphs.

Definition 2 A pattern P of type GD or F has an occurrence ending at position j in a text of type GD or F if $\exists i \in [1, j]$ such that $\mathcal{L}(P) \cap \mathcal{L}(T^{\ell,r}[i..j]) \neq \emptyset$ for some $\ell \in [0, k_i - 1]$ and $r \in [0, k_j - 1]$.

The case where both P and T are 1-D is well-studied in the literature. The definition of *indeterminate* string given in [60] coincides with our definition of 1-D string; in the same paper, the authors proposed an $\mathcal{O}(n\log m)$ -time algorithm for MATCH(1-D,1-D) in the case where the alphabet is of constant size [60, Lemma 17]. A similar algorithm for constant-size alphabets was also proposed in [84]. These results were complemented in [60, Theorem 22] with a quadratic conditional lower bound for cases where the alphabet is not of constant size.

This lower bound clearly applies also to MATCH(1-D,k-D) and MATCH(k-D,1-D) when the alphabet size is not constant. In Section Matching a 1-D pattern in a k-D

text, we prove that, in these cases, a quadratic conditional lower bound holds even when the alphabet has only three letters. In Section Matching a 1-*F* pattern in a 1-*D* text, we consider problems MATCH(1-*D*,1-*F*), MATCH(1-*F*,1-*D*), MATCH(1-*F*,1-*F*). For all these cases, we prove a quadratic conditional lower bound for constant-size alphabets. This implies that when the pattern is not solid and at least one between the pattern and the text is a graph, the best we can hope to achieve is a quadratic-time algorithm.

The conditional lower bounds we prove rely on a famous conjecture: the *Orthogonal Vectors Hypothesis*, which is, in turn, implied by SETH [61, 89].

Definition 3 (*Orthogonal Vectors* (OV)) Given two sets $X, Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$ and $d = \omega(\log n)$, determine whether there exist $x \in X$ and $y \in Y$ such that x and y are orthogonal, namely, $x \cdot y = \sum_{i=1}^{d} x[i] \cdot y[i] = 0$.

In the rest of the paper, we will use the instance $X = \{x_1 = 010, x_2 = 100, x_3 = 011\}$, $Y = \{y_1 = 001, y_2 = 010, y_3 = 110\}$ of OV as our running example. Note that $x_2 = 100$, $y_2 = 010$ is a valid solution since $x_2 \cdot y_2 = 0$.

Conjecture 1 (Orthogonal Vectors Hypothesis (OVH)) *No (deterministic or randomized) algorithm can solve* OV *on vector sets* $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$, *in time* $\mathcal{O}(n^{2-\epsilon}\text{poly}(d))$ *for any* $\epsilon > 0$.

Here we summarize the general idea used in the proofs of this section. We start with an instance of OV: $X, Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$. Then we construct in $\mathcal{O}(nd)$ time a pattern P and a text T such that there is a match of P in T if and only if there exists a pair of orthogonal vectors between X and Y. We will ensure that the size of both P and T is $\mathcal{O}(nd)$. In this way, a subquadratic algorithm for matching P in T would imply an $\mathcal{O}((nd)(nd)^{1-\epsilon}) = \mathcal{O}(n^{2-\epsilon}\text{poly}(d))$ time algorithm for OV, which would contradict OVH. The reader familiar with the work of Backurs and Indyk [11] might wonder if some of the reductions they proposed – in particular, those proving SETH-hardness for pattern matching or membership testing with regular expressions of the type $\cdot|\cdot$ – may be directly applied or easily modified to prove lower bounds for the problems in this section. We remark that this is not the case, as the reductions in [11] rely heavily on the strings in each segment having different lengths, while the segments of *k*-*D* strings and *k*-*F* graphs contain strings of equal length.

Matching a 1-*D* pattern in a *k*-*D* text

We start by introducing a gadget that will be used in several reductions. Given a vector $y \in \{0,1\}^d$, let $Q(y)$ be a 1-*D* string with d segments $Q(y)[h]$, $1 \le h \le d$, defined as

$$Q(y)[h] = \begin{cases} 0 \\ 1 \end{cases} \quad \text{if } y[h] = 0;$$
$$Q(y)[h] = \{0\} \quad \text{if } y[h] = 1.$$

A similar encoding of the vectors is often used in the reductions from OV: the key property, which is clear by construction, is that a string x occurs in $Q(y)$ only if it encodes a vector orthogonal to y, as stated in the following lemma.

Lemma 2 *Let* $x, y \in \{0,1\}^d$, *then the string* $x[1]x[2]\cdots x[d]$ *occurs in* $Q(y)$ *if and only if* $x \cdot y = 0$.

Theorem 5 *No algorithm can solve* MATCH(1-*D*,*k*-*D*) *on constant alphabet for every* k *in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless* OVH *is false.*

Proof Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be an instance of OV. We define a 1-*D* pattern P and a *k*-*D* text T such that P occurs in T if and only if $\exists x \in X$ and $y \in Y$ with $x \cdot y = 0$. We start by constructing pattern gadgets $Q(y_i)$, for each $y_i \in Y$. We then concatenate such gadgets into a single 1-*D* pattern using an extra character \$ that will force synchronisation with T:

$$P = \{\$\}Q(y_1)\{\$\}\cdots\{\$\}Q(y_n).$$

We remark that the size of P is $M = \mathcal{O}(nd)$. To build the text T, we list all the vectors from X in one segment W, surrounded by $n-1$ segments of the form $Z = \{\$0^d\}$ on both sides:

$$T = \underbrace{\{\$\overbrace{0\cdots0}^{d \text{ times}}\}\cdots\{\$0\cdots0\}}_{n-1 \text{ times}} \underbrace{\left\{\begin{matrix}\$x_1[1]\cdots x_1[d] \\ \vdots \\ \$x_n[1]\cdots x_n[d]\end{matrix}\right\}}_{W} \underbrace{\overbrace{\{\$0\cdots0\}}^{Z}\cdots\{\$0\cdots0\}}_{n-1 \text{ times}}$$

Clearly, T is a $(d+1)$-D string of size $N = \mathcal{O}(nd)$. The idea is that Z can match any gadget $Q(y_i)$, while W allows matches only from gadgets encoding vectors that are orthogonal to a vector in X (Lemma 2). Since P has n gadgets of length d, any occurrence of P in T must span a string in W (see Figure 9). Summing up, if P has a match in T starting at $T[i]$, then it must start at the first position of $T[i]$ because the \$ symbol matches nowhere else. Then i must be less or equal than n, and the intersection of $\mathcal{L}(Q(y_{n-i+1}))$ and $\mathcal{L}(W)$ must be non empty, implying that vector y_{n-i+1} is orthogonal to some vector of X. Therefore, deciding if there exists a pair of orthogonal vectors from X and Y can be reduced in $\mathcal{O}(nd)$ time to an instance of matching a 1-D pattern P of size $\mathcal{O}(nd)$ in a $(d+1)$-D text T of size $\mathcal{O}(nd)$. If we could find a match for P in T in $\mathcal{O}(N^{1-\epsilon}M)$ or $\mathcal{O}(NM^{1-\epsilon})$ time, then we could solve OV in $\mathcal{O}((nd)(nd)^{1-\epsilon}) = \mathcal{O}(n^{2-\epsilon}\,\mathrm{poly}(d))$ time, which would contradict OVH. □

Theorem 6 *No algorithm can solve* MATCH(k-D,1-D) *on constant alphabet for every k in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless* OVH *is false.*

Proof The reduction is entirely analogous to that of Theorem 5, except now we define a 1-D text $T = \{\$\}Q(y_1)\{\$\}\cdots\{\$\}Q(y_n)$ and $(d+1)$-D pattern $P = W$ of length 1 containing all the vectors from X. We can conclude as before. □

Remark that in both Theorems 5 and 6 we have $k = \omega(\log n)$, where n is both the number of vectors of OV and the length of the text; it remains open whether there exist subquadratic algorithms when $k = \mathcal{O}(\log n)$.

Matching a 1-*F* pattern in a 1-*D* text

In the following proofs, we will use a three-level strategy first introduced in [39, 41].

We start by defining two 1-D strings, P and T (one for the pattern and one for the text), underlying all the 1-F graphs in the reductions of this section. Let $X, Y \subseteq \{0,1\}^d, |X| = |Y| = n$ be any instance of OV. T and P are over the constant-size alphabet $\Sigma = \{0, 1, u, b, \$\}$. The role of the letters u and b is to identify parts of the segments that we will call their *u*pper and *b*ottom level. We build P as the concatenation of n 1-D string gadgets, one for each vector of X: given $x_i \in X$, we define

$$p(x_i) = \begin{Bmatrix} u \\ x_i[1] \\ b \end{Bmatrix} \cdots \begin{Bmatrix} u \\ x_i[d] \\ b \end{Bmatrix}.$$

We add the symbol \$ at the beginning and end of the pattern to force the occurrences to start/end in some specific parts of the text. The complete 1-D pattern then is

$$P = \{\$\}p(x_1)p(x_2)\cdots p(x_n)\{\$\}, \tag{1}$$

thus $|P| = nd + 2 = \mathcal{O}(nd)$ and $\|P\| = 3nd + 2 = \mathcal{O}(nd)$ (see Figure 10 for an example).

T consists of three different 1-D strings: T_{left}, $T^{\perp}$ and T_{right}. The 1-D string T_{left} consists of a single segment $\{\$\}$ followed by $n-1$ gadgets $U_{left} = \{u\}^{d-1}\begin{Bmatrix}\$\\ u\end{Bmatrix}$. In a symmetric way, T_{right} consists of $n-1$ gadgets $U_{right} = \begin{Bmatrix}b\\ \$\end{Bmatrix}\{b\}^{d-1}$ followed by $\{\$\}$. The 1-$D$ string $T^{\perp}$ is built using a slight variation of the gadgets $Q(y_i)$ (see proof of Theorem 5) extended with a three-level structure. Given $y_j \in Y$, we define $T_j^{\perp}[1] = \{u\} \cup Q(y_j)[1] \cup \{b, \$\}$,

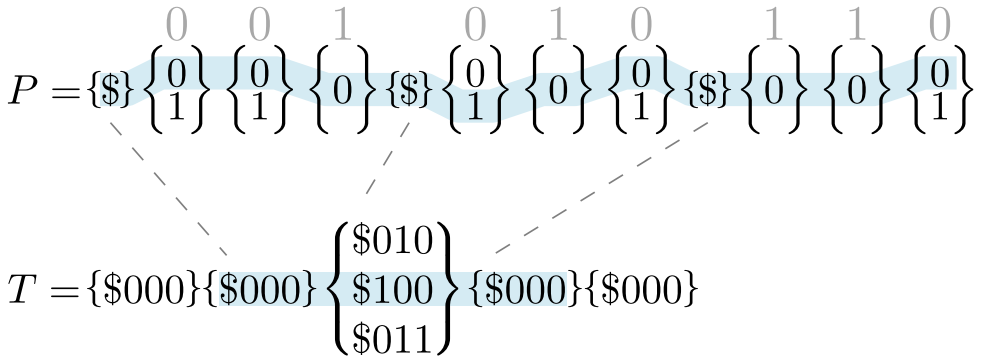

Fig. 9 Example of the pattern and text constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ in the proof of Theorem 5. The highlighted paths represent an occurrence of P in T, which identifies two orthogonal vectors: $x_2 = 100 \in X$ and $y_2 = 010 \in Y$. The dashed lines are drawn to emphasise the synchronisation forced by the symbol \$

$T_j^{\perp}[h] = \{u\} \cup Q(y_j)[h] \cup \{b\}$ for $2 \le h \le d-1$ and $T_j^{\perp}[d] = \{u, \$\} \cup Q(y_j)[d] \cup \{b\}$. We define $T^{\perp}$ as the concatenation of $T_1^{\perp} \cdots T_n^{\perp}$. Finally, the full 1-*D* text T is the concatenation of T_{left}, $T^{\perp}$ and T_{right} (see Figure 11 for an example):

$$T = \{\$\} U_{left}^{n-1} T_1^{\perp} \cdots T_n^{\perp} U_{right}^{n-1} \{\$\}. \tag{2}$$

We remark that the size of T is $\mathcal{O}(nd)$, since it is built from three 1-*D* strings each of size $\mathcal{O}(nd)$.

Theorem 8 *No algorithm can solve* MATCH(1-*D*,1-*F*) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless* OVH *is false.*

Proof Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the pattern P as in Equation (1). To build the 1-*F* text G_T, we consider T as in Equation (2) and we first separately construct three different 1-*F* graphs: $G_{left} = (T_{left}, E_{left})$, $G^{\perp} = (T^{\perp}, E^{\perp})$ and $G_{right} = (T_{right}, E_{right})$. The set of edges E_{left} of G_{left} contain every possible edge between consecutive segments except for that of type $(u, \$)$, which has no edge incoming to \$. Symmetrically, the set E_{right} of G_{right} contains every possible edge between consecutive segments except for the one of the form $(\$, b)$, which has no edge outgoing from \$.

To define $E^{\perp}$, we first define the edges for each $T_j^{\perp}$. For these gadgets, we allow all the edges of the form (u, u), (b, b) and (x, y) for any $x, y \in \{0, 1\}$. In this way, any matching string of $\mathcal{L}(P)$ passing through $T_j^{\perp}$ must follow the upper level identified by letters u, the bottom level identified by letters b or the orthogonal level corresponding to $Q(y_j)$ (recall that any string $x \in \{0,1\}^d$ matches $Q(y_j)$ if and only if $x \cdot y_j = 0$). To complete $E^{\perp}$, for all consecutive segments $T_i^{\perp}[d]$ and $T_{i+1}^{\perp}[1]$ we build two sets of edges: the first connects the upper level of $T_i^{\perp}[d]$ (i.e. u and \$) to the upper and orthogonal level of $T_{i+1}^{\perp}[1]$; the second connects the orthogonal and bottom level

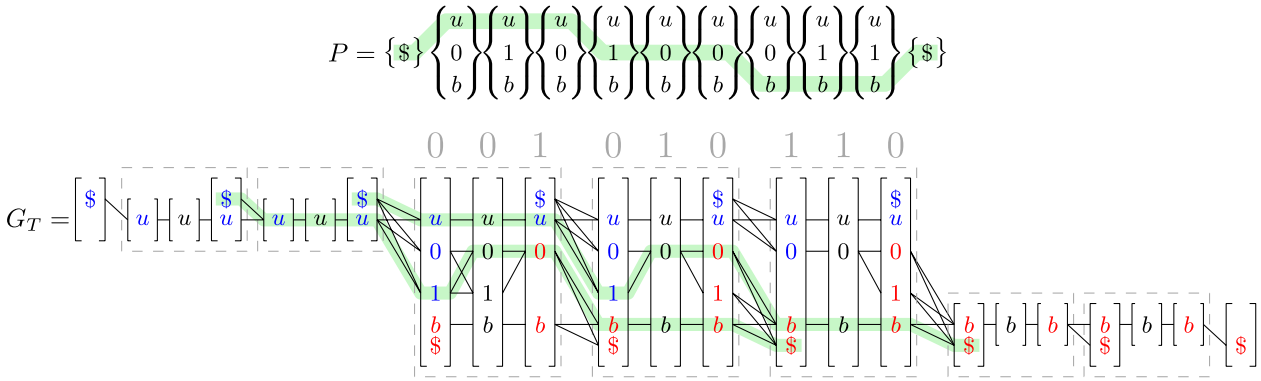

Fig. 10 Example of a 1-*D* pattern P and 1-*F* text G_T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem 8. Highlighted paths show some occurrences of P in T, given by the string $\$uuu100bbb\$ \in \mathcal{L}(P)$ which belongs also to $\mathcal{L}(T[4 \dots 14])$ and $\mathcal{L}(T[7 \dots 17])$. This corresponds to the fact that $x_2 = 100$ is orthogonal to $y_1 = 100$ and $y_2 = 010$

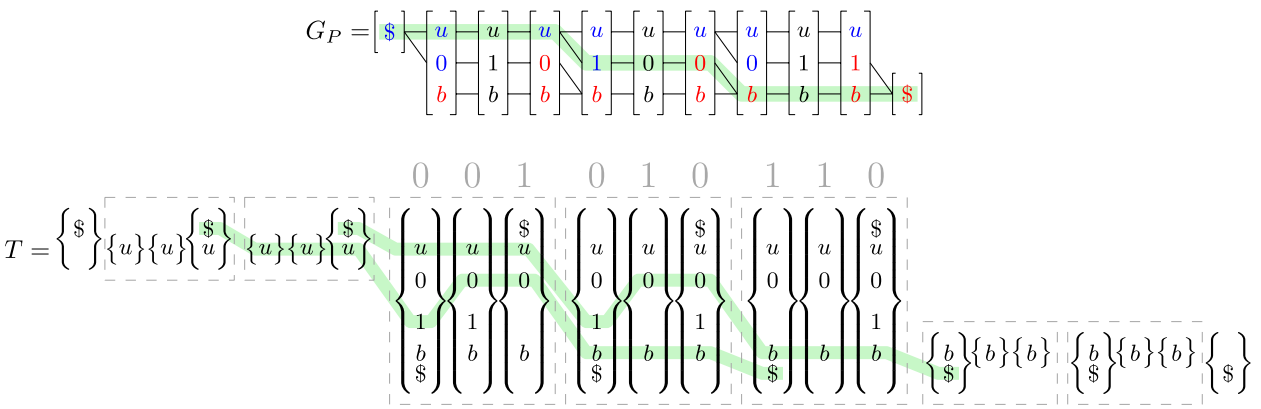

Fig. 11 Example of a 1-*F* pattern G_P and a 1-*D* text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem 9, with occurrences of G_P in T highlighted

of $T_i^{\perp}[d]$ with the bottom level of $T_{i+1}^{\perp}[1]$ (i.e. b and \$). Finally, $G_T = (T, E_T)$ is the union of G_{left}, $G^{\perp}$ and G_{right} where we add to E_T all the edges from the last segment of T_{left} to the upper level and orthogonal level of $T_1^{\perp}[1]$ and all the edges from the orthogonal and bottom level of $T_n^{\perp}[d]$ to the first segment of T_{right}. We remark that $|E_T| = \mathcal{O}(nd)$, thus since $\|T\| = \mathcal{O}(nd)$, the size of G_T is $\mathcal{O}(nd)$.

From the construction of the edges, we have that the only allowed edges from \$ are $(\$, u)$, $(\$, 0)$ and $(\$, 1)$ and the only allowed edges to \$ are $(0, \$)$, $(1, \$)$ and $(b, \$)$. Hence, the pattern must start either on the upper level or in G_{left} and finish either on the bottom level or in G_{right}. Since the bottom level (thus also G_{right}) can be reached only after reading the orthogonal level, we deduce by Lemma 2 that the pattern has a match in G_T if and only if x_i matches with $Q(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 10 for an example. □

Theorem 9 *No algorithm can solve* MATCH(1-*F*,1-*D*) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless* OVH *is false.*

Proof Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the text T as in Equation (2). To build the pattern $G_P = (P, E_P)$, we consider P as in Equation (1) and we construct the set of edges E_P with the same criteria used for G_T in the proof of Theorem 8. At the beginning of the pattern, we add the edges $(\$, u)$, $(\$, x_1[1])$, and at the end we add the edges $(x_n[d], \$)$ and $(b, \$)$. For each $x_i \in X$, we add to $p(x_i)$ all the edges of the form (u, u), (b, b) and (x, y) for any $x, y \in \{0, 1\}$. Furthermore, from $p(x_i)$ to $p(x_{i+1})$ we add the edges: (u, u), $(u, x_{i+1}[1])$, $(x_i[d], b)$, (b, b). Clearly, $|E_P| \leq 4nd = \mathcal{O}(nd)$, thus the size of G_P is $\|P\| + |E_P| = \mathcal{O}(nd)$. Furthermore, we can deduce that the language of G_P is $\mathcal{L}(G_P) = \{\$u^{(i-1)d} \cdot x_i \cdot b^{(n-i)d}\$: i = 1, \ldots, n\}$. Remark that each x_i is over the alphabet $\{0, 1\}$. Thus, because of the position of the symbol \$, Lemma 2 applies, that is, there is an occurrence of P in T if and only if x_i matches with $Q(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 11 for an example. □

Finally, the following result follows directly from a reduction that uses G_T as in the proof of Theorem 8 and G_P as in the proof of Theorem 9.

Theorem 10 *No algorithm can solve* MATCH(1-*F*,1-*F*) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless* OVH *is false.*

Matching a 2-*D* pattern in a 2-*D* text

This section considers the problem MATCH(2-*D*, 2-*D*). Note that Theorem 5 and Theorem 6 directly imply a quadratic conditional lower bound for the general MATCH(k-*D*, k-*D*) problem. However, in the reductions underlying the two theorems, the width k of the k-*D* string is $\mathcal{O}(d)$, i.e. linear in the length of the vectors of OV, which in turn is $\omega(\log n)$, where n is both the length of the text and the number of vectors of OV. Therefore, in principle, those results do not rule out the possibility that MATCH(k-*D*, k-*D*) can be solved faster than quadratically when e.g. $k = \mathcal{O}(1)$. In Theorem 7, we prove this is not the case, as a quadratic conditional lower bound holds already when $k = 2$ and the alphabet is of constant size.

The reduction is similar in spirit to those of Theorems 5 and 6 and relies on the following gadgets. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be any instance of OV. We construct a pattern P and a text T over the alphabet $\Sigma = \{0, 1, u, b, \$\}$. To build P, we first define n 2-*D* strings gadgets, one for each vector $x_i \in X$:

$$p^{(2D)}(x_i) = \left\{ \begin{matrix} uu \\ x_i[1]x_i[1] \\ bb \end{matrix} \right\} \cdots \left\{ \begin{matrix} uu \\ x_i[d]x_i[d] \\ bb \end{matrix} \right\}.$$

P is obtained by concatenating these gadgets, and adding between $p^{(2D)}(x_i)$ and $p^{(2D)}(x_{i+1})$ the segment:

$$\left\{ \begin{matrix} uu \\ ux_{i+1}[1] \\ x_i[d]b \\ bb \end{matrix} \right\}.$$

Furthermore, at the beginning of the pattern, we append the segment $\left\{ \begin{matrix} \$u \\ \$x_1[1] \end{matrix} \right\}$, and at the end we append the segment $\left\{ \begin{matrix} x_n[d]\$ \\ b\$ \end{matrix} \right\}$. The complete 2-*D* pattern then is

$$P = \left\{ \begin{matrix} \$u \\ \$x_1[1] \end{matrix} \right\} p^{(2D)}(x_1) \left\{ \begin{matrix} uu \\ ux_2[1] \\ x_1[d]b \\ bb \end{matrix} \right\} p^{(2D)}(x_2) \cdots \left\{ \begin{matrix} uu \\ ux_n[1] \\ x_{n-1}[d]b \\ bb \end{matrix} \right\} p^{(2D)}(x_n) \left\{ \begin{matrix} x_n[d]\$ \\ b\$ \end{matrix} \right\}, \tag{3}$$

thus $|P| = \mathcal{O}(nd)$ and $\|P\| = \mathcal{O}(nd)$ (see Figure 12 for an example). Note that this pattern encodes exactly the edges in the construction of G_P (see proof of Theorem 9).

To build the text T, we introduce a variation of the gadget $Q(y)$ used in Sections Matching a 1-*D* pattern in a *k*-*D* text and Matching a 1-*F* pattern in a 1-*D* text. Given a vector $y \in \{0,1\}^d$, let $R(y)$ be a 2-*D* string with $d+1$ segments, defined as follows (see Figure 12 for an example):

- for $1 < h < d+1$:

$$R(y)[h] = \begin{Bmatrix} 00 \\ 01 \\ 10 \\ 11 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 0;$$

$$R(y)[h] = \begin{Bmatrix} 00 \\ 10 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 1;$$

$$R(y)[h] = \begin{Bmatrix} 00 \\ 01 \end{Bmatrix} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 0;$$

$$R(y)[h] = \{00\} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 1;$$

- for $h = 1$:

$$R(y)[1] = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix} \quad \text{if } y[1] = 0;$$

$$R(y)[1] = \{00\} \quad \text{if } y[1] = 1;$$

- for $h = d+1$:

$$R(y)[d+1] = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix} \quad \text{if } y[d] = 0;$$

$$R(y)[d+1] = \{00\} \quad \text{if } y[d] = 1.$$

The following result is analogous to Lemma 2.

Lemma 3 *Let $x, y \in \{0,1\}^d$, then the string $\rho(x) = x[1]^3 x[2]^2 \cdots x[d-1]^2 x[d]^3$ occurs in $R(y)$ if and only if $x \cdot y = 0$.*

Proof Notice that the length of string $\rho(x)$ and the length of the strings in $\mathcal{L}(R(y))$ are both $2d+2$. Thus, any occurrence of $\rho(x)$ must span $R(y)$ entirely from left to right. Moreover, by simple case analysis, $x[1]^3$ matches in $R(y)[1]R(y)[2]$ if and only if $x[1] \cdot y[1] = 0$. The same holds for $x[d]^3$, which matches in $R(y)[d]R(y)[d+1]$ if and only if $x[d] \cdot y[d] = 0$. For a generic substring $x[h]^2$, one character must match in $R(y)[h]$ and the other in $R(y)[h+1]$. Every possible configuration of $R(y)[h]$ always has a 0 both in the first and in the second column, hence $x[h]^2 = 00$ can always match. For $x[h]^2 = 11$, the match is not possible if $R(y)[h]$ does not have any 1 in the second column and $R(y)[h+1]$ does not have any 1 in the first column, but this can happen only if $y[h] = 1$. Summing up, the substring $x[h]^2$ has an occurrence in $R(y)[h]R(y)[h+1]$ if and only if $x[h] \cdot y[h] = 0$. □

We are now able to build the text T. For each $y_j \in Y$, we define $T_j^{\perp}[1] = \{uu\} \cup R(y_j)[1] \cup \{bb, \$\$\}$, $T_j^{\perp}[h] = \{uu\} \cup R(y_j)[h] \cup \{bb\}$ for $2 \le h \le d$ and $T_j^{\perp}[d+1] = \{uu, \$\$\} \cup R(y_j)[d+1] \cup \{bb\}$. The full 2-*D* text T is (see Figure 12 for an example):

$$T = \{\$\$\} U_{left}^{n-1} T_1^{\perp} \cdots T_n^{\perp} U_{right}^{n-1} \{\$\$\} \tag{4}$$

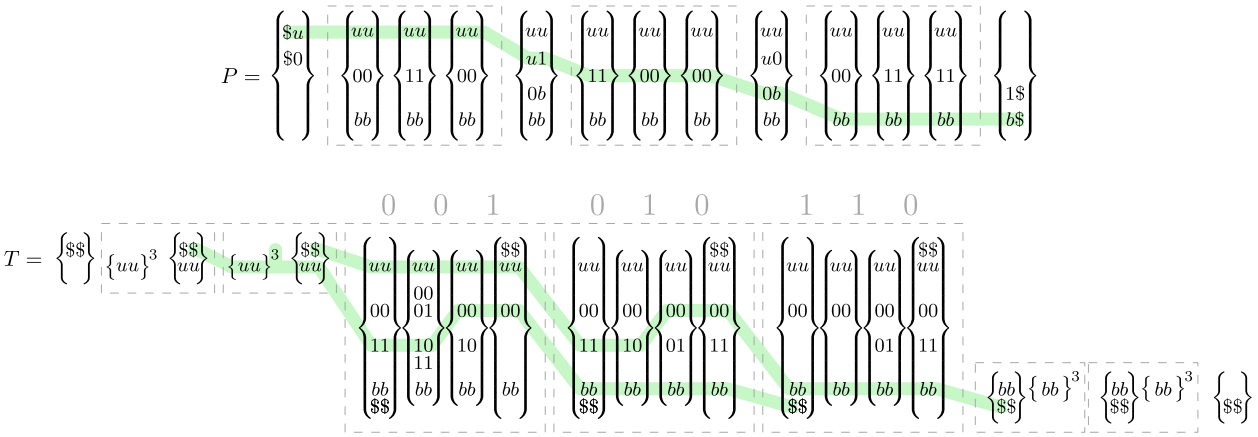

Fig. 12 Example of a 2-*D* pattern P and 2-*D* text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem 7. Highlighted paths show some occurrences of P in T, given by the string $\$u^8 1^3 0^2 0^3 b^8 \$ \in \mathcal{L}(P)$ which corresponds to the fact that $x_2 = 100$ is orthogonal to $y_1 = 100$ and $y_2 = 010$

where

$$U_{left} = \{uu\}^d \begin{Bmatrix} \$\$ \\ uu \end{Bmatrix} \quad \text{and} \quad U_{right} = \begin{Bmatrix} bb \\ \$\$ \end{Bmatrix} \{bb\}^d.$$

Since $T_i^{\perp}$, U_{left}, U_{right} have all size $\mathcal{O}(d)$, then size of T is $\mathcal{O}(nd)$.

Theorem 7 *No algorithm can solve* MATCH(2-*D*,2-*D*) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless* OVH *is false.*

Proof Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the pattern P as in Equation (3) and the text T as in Equation (4). Since any string in $\mathcal{L}(P)$ must start and end with the symbol \$, any occurrence of P in T must begin in the middle of a segment containing the string \$\$. Moreover, because of its length, any occurrence of P must either begin in the U_{left} part or end in the U_{right} part. In the first case, this implies that the occurrence must finish in the middle of a segment $T_j^{\perp}[1]$, and in the second case that it must start in the middle of a segment $T_i^{\perp}[d+1]$.

From the fact that the pattern always occurs with offset 1 w.r.t. the text, we can deduce that the only possible strings in $\mathcal{L}(P)$ that could match P are of the form:

$$\{\$u^{(i-1)(2d+2)} \cdot \rho(x_i) \cdot b^{(n-i)(2d+2)}\$: i = 1, \ldots, n\} \subsetneq \mathcal{L}(P).$$

This is true since at any matched character we are forced, by either the pattern or the text, to follow a match with the same rules explained in Theorem 8 and Theorem 9. Remark that $\rho(x_i)$ is over the alphabet $\{0,1\}$. Thus, by Lemma 3, we conclude that there is an occurrence of P in T if and only if $\rho(x_i)$ matches $R(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 12 for an example of this reduction. □

Matching a solid pattern in an *ED* text

In this section, we prove that a quadratic conditional lower bound for MATCH(SOLID, *ED*) holds even for a severely restricted type of *ED* strings, in which the length of the strings in each segment can differ by at most 1. Note that a quadratic conditional lower bound for MATCH(SOLID, *ED*) for general *ED* strings already exists, as it is directly implied by [10, Theorem 10]. The latter result is a reduction from OV to the problem of determining if a string P can be derived from a regular expression of the form $\cdot|\cdot$ (composition of concatenation, *or*, concatenation), and it implies a lower bound for MATCH(SOLID, *ED*) by further applying the following simple reduction. Any regular expression of the form $\cdot|\cdot$ naturally corresponds to an *ED* string: the first concatenation generates strings of characters, which are the terms of an *or* operator and can be seen as strings belonging to the same segment of an *ED* string. The second concatenation operator is emulated by concatenating the segments, thus forming an *ED* string. We show this construction in the following example.

$$[a|bb][aa|abb|abab][bb|abab] \rightarrow \begin{Bmatrix} a \\ bb \end{Bmatrix} \begin{Bmatrix} aa \\ abb \\ abab \end{Bmatrix} \begin{Bmatrix} bb \\ abab \end{Bmatrix}$$

Let T' be the *ED* string obtained by appending a segment with a single new character \$ at the beginning and the end of the *ED* string we generated from the regular expression, and let $P' = \$P\$$. Through this reduction, any algorithm solving MATCH(SOLID,*ED*) in subquadratic time could be used to determine if P can be derived from T, contradicting the lower bound of [10, Theorem 10].

The result we present in this section is *stronger* than the lower bound implied by [10, Theorem 10] in the sense that the *ED* strings constructed from instances of OV through our reduction are way less general than those implied by the former result. We start by recalling a definition originally given in [58].

Definition 4 (*Global elasticity*) Given a degenerate string $T = T[1] \cdots T[n]$ let $|T[i]|_{\max}$ (resp. $|T[i]|_{\min}$) be the length of the longest (resp. shortest) string in $T[i]$. The *global elasticity* of T is

$$\Delta(T) = \sum_{i=1}^{n} (|T[i]|_{\max} - |T[i]|_{\min}).$$

The quantity Δ measures the difference between the length of the longest and the shortest string in the language $\mathcal{L}(T)$ and thus it gives a global property of the *ED* string. To obtain a measure of how much the lengths of the strings in the segments of *ED* string vary locally, we introduce the following definition.

Definition 5 (*Local elasticity*) Given a degenerate string $T = T[1] \cdots T[n]$, we define the *local elasticity* of T as

$$\Delta_{loc}(T) = \max_{i=1,\dots,n} (|T[i]|_{\max} - |T[i]|_{\min}).$$

By definitions 4 and 5, it holds that $0 \leq \Delta_{loc}(T) \leq \Delta(T)$ and that $\Delta_{loc}(T) = 0$ if and only if $\Delta(T) = 0$, that is, if and only if T is a *GD* string. Intuitively, $\Delta_{loc}(T)$ measures "how far" is T from being a *GD*.

In the proof of [10, Theorem 10], the elastic degenerate string to which an instance of OV is reduced has local elasticity $\mathcal{O}(d)$ and global elasticity $\mathcal{O}(nd)$, where d is the length of the vectors of OV. In Theorem 11, we prove that the quadratic conditional lower bound holds even if we restrict to *ED* strings with $\Delta_{loc} = 1$ (thus as close to *GD* strings as possible), on constant-size alphabet, and with global elasticity $\mathcal{O}(n)$.

Let us describe the gadgets we will use in our reduction. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be any instance of OV. We will construct a solid pattern P and an *ED* text T over the constant-size alphabet $\Sigma = \{0, 1, \sharp, \flat, \$\}$. To build P, we define n gadgets, one for each vector of $x_i \in X$:

$$\alpha(x_i) = x_i[1]^3 \sharp x_i[2]^3 \sharp \cdots \sharp x_i[d]^3.$$

Then, we concatenate the gadgets in the following way:

$$P = \$\alpha(x_1)\flat^3\alpha(x_2)\flat^3 \cdots \flat^3\alpha(x_n)\$. \tag{5}$$

It can be easily verified that $\|P\| = |P| = \mathcal{O}(nd)$.

To define the *ED* text T, we first define the following sets of strings, which will be used as a tool to construct the segments of T:

$$U_{up} = \begin{Bmatrix} \$\$0 \\ \$\$1 \\ \flat\flat0 \\ \flat\flat1 \end{Bmatrix} \begin{Bmatrix} 00\sharp0 \\ 00\sharp1 \\ 11\sharp0 \\ 11\sharp1 \end{Bmatrix}^{d-1} \begin{Bmatrix} 00\flat \\ 11\flat \end{Bmatrix}$$

$$U_{mid} = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat00 \\ \flat\flat11 \end{Bmatrix} \begin{Bmatrix} 0\sharp00 \\ 0\sharp11 \\ 1\sharp00 \\ 1\sharp11 \end{Bmatrix}^{d-1} \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix}$$

$$U_{down} = \begin{Bmatrix} \flat000 \\ \flat111 \end{Bmatrix} \begin{Bmatrix} \sharp000 \\ \sharp111 \end{Bmatrix}^{d-1} \begin{Bmatrix} \flat\flat \\ \$\$ \end{Bmatrix}$$

We then define the *universal gadget* U such that $U[h] = U_{up}[h] \cup U_{mid}[h] \cup U_{down}[h]$ for all $1 \leq h \leq d+1$ (see Figure 13 for an example); we remark that $\Delta_{loc}(U) = 1$ and $\Delta(U) = 2$. Note that U will not appear as-is in T: it will rather be used to define the gadgets encoding the vectors from set Y. The following lemma will be useful to prove the correctness of our reduction in Theorem 11. We say that a string t occurs in the *up* level of U (resp. *mid*, *down* level) if t occurs in U_{up} (resp. U_{mid}, U_{down}).

Lemma 4 *For every $x \in \{0,1\}^d$ there exist exactly three disjoint occurrences of $\flat\alpha(x)\flat$ in U (one for each level).*

Proof Comparing the length of $\flat\alpha(x)\flat$ to the lengths of the strings in $\mathcal{L}(U)$, we deduce that any occurrence must start at $U[1]$ and end at $U[d+1]$. In $U[1]$, we can either match $\flat x[1]$, $\flat x[1]^2$ or $\flat x[1]^3$. Since $U[2]$ is a 4-*D* string, we can extend the previous partial occurrences in at most one way (see Observation 3), that is: $x[1]^2\sharp x[2]$ extends $\flat x[1]$ in the *up* level, $x[1]\sharp x[2]^2$ extends $\flat x[1]^2$ in the *mid*

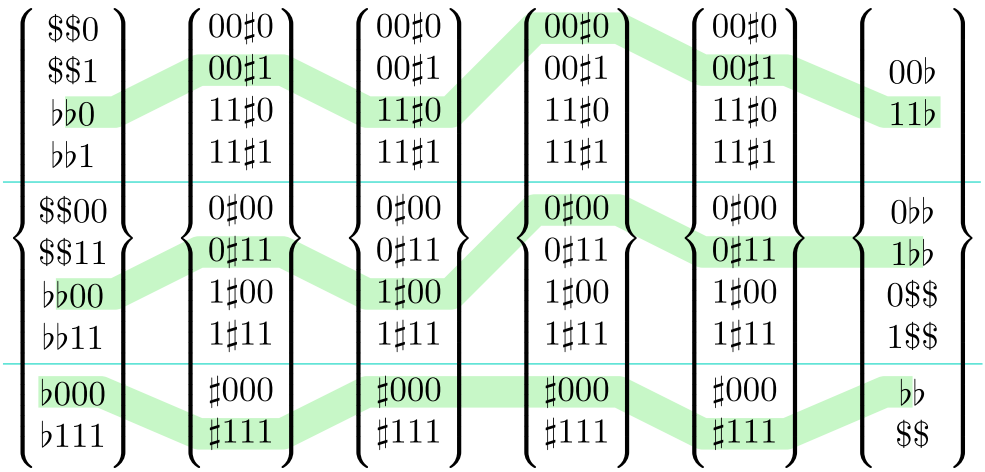

Fig. 13 Occurrences of $\flat\alpha(x)\flat$ in U where $x = 01001$ (thus $d = 5$). The horizontal lines divide the *up*, *mid* and *down* levels

level and $\sharp x[2]^3$ extends $\flat x[1]^3$ in the *down* level. Iterating the above reasoning, we have the thesis. See Figure 13 for an example. □

Lemma 4 holds also for strings $\$\alpha(x)\flat$ and $\flat\alpha(x)\$$.

We now construct a gadget $T_i^{\perp}$ for each vector $y_i \in Y$ by means of the universal gadget U: each of these gadgets will consist of $d+1$ segments, each containing a carefully selected subset of the strings from the segments of U. Gadget $T_i^{\perp}$, corresponding to vector $y_i \in Y$, is obtained from U using the same logic as gadgets $R(y_i)$ defined in Section Matching a 2-*D* pattern in a 2-*D* text: we remove some strings from the segments of U_{mid} in such a way that the remaining strings in the language of U_{mid} represent the vectors orthogonal to y_i. More precisely, given a vector $y \in Y$, let $U_{mid}^{\perp}(y)$ be a *GD* string with $d+1$ segments, defined as follows:

- for $1 < h < d+1$:

$$U_{mid}^{\perp}(y)[h] = \begin{Bmatrix} 0\sharp 00 \\ 0\sharp 11 \\ 1\sharp 00 \\ 1\sharp 11 \end{Bmatrix} \quad \text{if } y[h-1]=0 \text{ and } y[h]=0;$$

$$U_{mid}^{\perp}(y)[h] = \begin{Bmatrix} 0\sharp 00 \\ 1\sharp 00 \end{Bmatrix} \quad \text{if } y[h-1]=0 \text{ and } y[h]=1;$$

$$U_{mid}^{\perp}(y)[h] = \begin{Bmatrix} 0\sharp 00 \\ 0\sharp 11 \end{Bmatrix} \quad \text{if } y[h-1]=1 \text{ and } y[h]=0;$$

$$U_{mid}^{\perp}(y)[h] = \{0\sharp 00\} \quad \text{if } y[h-1]=1 \text{ and } y[h]=1;$$

- for $h = 1$:

$$U_{mid}^{\perp}(y)[1] = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat 00 \\ \flat\flat 11 \end{Bmatrix} \quad \text{if } y[1]=0;$$

$$U_{mid}^{\perp}(y)[1] = \begin{Bmatrix} \$\$00 \\ \flat\flat 00 \end{Bmatrix} \quad \text{if } y[1]=1;$$

- for $h = d+1$:

$$U_{mid}^{\perp}(y)[d+1] = \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix} \quad \text{if } y[d]=0;$$

$$U_{mid}^{\perp}(y)[d+1] = \begin{Bmatrix} 0\flat\flat \\ 0\$\$ \end{Bmatrix} \quad \text{if } y[d]=1.$$

We define $T_i^{\perp}[h] = U_{up}[h] \cup U_{mid}^{\perp}(y_i)[h] \cup U_{down}[h]$ for all $1 \le h \le d+1$ and for all $y_i \in Y$. The proof of the following lemma is entirely analogous to that of Lemma 3.

Lemma 5 *Let $x_i, y_j \in \{0,1\}^d$; then the strings $\flat\alpha(x_i)\flat$, $\$\alpha(x_i)\flat$, and $\flat\alpha(x_i)\$$ occurs in the mid level of T_j if and only if $x_i \cdot y_j = 0$.*

For example, let $y = 010$ and $x = 100$, we have that

$$U_{mid}^{\perp}(y) = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat 00 \\ \flat\flat 11 \end{Bmatrix} \begin{Bmatrix} 0\sharp 00 \\ 1\sharp 00 \end{Bmatrix} \begin{Bmatrix} 0\sharp 00 \\ 0\sharp 11 \end{Bmatrix} \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix}$$

and $\flat\alpha(x)\flat = \flat 111\sharp 000\sharp 000\flat$ occurs in $U_{mid}^{\perp}(y)$ since x and y are orthogonal (see Figure 14 for a visual representation of the match).

Finally, the complete text is defined as:

$$T = U_{left}^{n-1}\{\flat\}T_1^{\perp}T_2^{\perp}\cdots T_n^{\perp}\{\flat\}U_{right}^{n-1}, \tag{6}$$

where

$$U_{left} = \begin{Bmatrix} \$\$\$ \\ \flat\flat\flat \end{Bmatrix} \begin{Bmatrix} 000 \\ 111 \end{Bmatrix} \begin{Bmatrix} \sharp 000 \\ \sharp 111 \end{Bmatrix}^{d-1} \quad \text{and}$$

$$U_{right} = \begin{Bmatrix} 000 \\ 111 \end{Bmatrix} \begin{Bmatrix} \sharp 000 \\ \sharp 111 \end{Bmatrix}^{d-1} \begin{Bmatrix} \flat\flat\flat \\ \$\$\$ \end{Bmatrix}.$$

Since $T_i^{\perp}$, U_{left}, U_{right} have all size $\mathcal{O}(d)$, the size of T is $\mathcal{O}(nd)$. Furthermore, since $\Delta_{loc}(U_{left}) = \Delta_{loc}(U_{right}) = 0$ and $\Delta_{loc}(T_i^{\perp}) = 1$ for all $i = 1, \ldots, n$, then $\Delta_{loc}(T) = 1$ and $\Delta(T) = \mathcal{O}(n)$. Armed with this reduction, we are ready to prove Theorem 11.

Theorem 11 *No algorithm can solve* MATCH(SOLID,ED) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, for any ED string such that the difference in the lengths of the strings in any segment is at most* 1*, and the length of all such strings is at most* 4*, unless* OVH *is false.*

Proof The correctness of our reduction relies on Lemmas 4 and 5 and the following facts. (i) No occurrence of P in T can end within U_{left}^{n-1} or start within U_{right}^{n-1}; (ii) Any occurrence of P either starts within U_{left}^{n-1} or in the *first* segment of some gadget $T_i^{\perp}$, and ends either in U_{right}^{n-1} or in the *last* segment of some gadget $T_j^{\perp}$; (iii) Consider a partial occurrence of P ending in the last segment of some gadget $T_i^{\perp}$, $i \le n-1$. If the occurrence ends matching some string in the *up* level of $T_i^{\perp}$, then it can only be extended by strings in the *up* or *mid* level of $T_{i+1}^{\perp}$; if it ends with a match of some string in the *mid* or *down* level of $T_i^{\perp}$, then it can only be extended by strings in the *down* level of $T_{i+1}^{\perp}$. Furthermore, any partial occurrence ending at the segment $\{\flat\}$ right after U_{left}^{n-1} can only be extended by strings in the *up* or *mid* level of $T_1^{\perp}$, and no

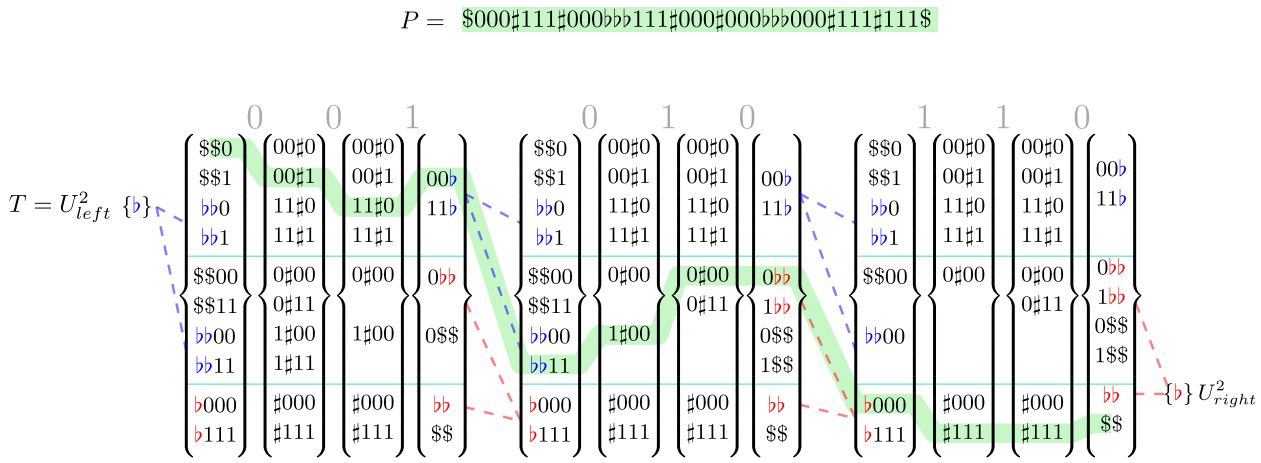

Fig. 14 Example of a solid pattern P and ED text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem 11: horizontal lines divide the *up*, *mid*, and *down* levels. Dashed lines highlight the allowed matching from $T_i^\perp$ to $T_{i+1}^\perp$, which are forced by the construction of the pattern P. The highlighted path shows an occurrence of P in T, which corresponds to the fact that $x_2 = 100$ is orthogonal to $y_2 = 010$

partial occurrence ending with a string in the *up* level of $T_n^\perp$ can be further extended (see Figure 14).

Fact (i) holds because, by construction, P is strictly longer than the strings in the languages $\mathcal{L}(U_{left}^{n-1})$ and $\mathcal{L}(U_{left}^{n-1})$. To prove Fact (ii), we notice that P always starts with \$ and the second letter is always 0 or 1. Since no string in the first segment of any gadget $T_i^\perp$ begins with 0 or 1, no partial occurrence of P starting within the last segment of a gadget $T_i^\perp$ can be extended to $T_{i+1}^\perp$ (nor to segment $\{\flat\}$). Symmetrically, the last two characters of P are always either 0 or 1 followed by \$, and since no string from the last segment of any gadget $T_i^\perp$ ends with 0 or 1, no occurrence can be completed with a \$ in the first segment of $T_{i+1}^\perp$.

To prove Fact (iii) notice that, in the pattern, $\flat\flat\flat$ occurs always between $\alpha(x_j)$ and $\alpha(x_{j+1})$. Furthermore, if $\alpha(x_j)\flat$ occurs in some segment $T_i^\perp$, then either (a) $x_j[d]\flat$ occurs in $T_i^\perp[d+1]$ (the last segment of $T_i^\perp$) and $\flat\flat x_{j+1}[1]$ occurs in $T_{i+1}^\perp[1]$ (the first segment of $T_{i+1}^\perp$) or (b) $x_j[d]\flat\flat$ occurs in $T_i^\perp[d+1]$ and $\flat x_{j+1}[1]$ occurs in $T_{i+1}^\perp[1]$. By construction (see Figure 14), Case (a) can only happen if $\alpha(x_j)\flat$ occurs in the *up* level and $\flat\flat\alpha(x_{j+1})$ occurs in the *up* or *mid* level. Similarly, Case (b) can only happen if $\alpha(x_j)\flat\flat$ occurs in the *mid* or *down* level and $\flat\alpha(x_{j+1})$ occurs in the *down* level.

By Fact (ii), and by the positions of the \$ and $\flat$ symbols in T and P, any complete occurrence of P must either start in the *up* or *mid* level of some $T_i^\perp$ or start in U_{left}^{n-1} and extend to the *up* or *mid* level of $T_1^\perp$. Similarly, it must either end in the *mid* or *down* level of some $T_j^\perp$ or in U_{right}^{n-1}, which can only be reached from the *mid* or *down* level of $T_n^\perp$. Combining this observation with Facts (i) and (iii) and Lemma 4, we obtain that any occurrence of P must entirely traverse the *mid* level of some $T_i^\perp$, thus a substring of P containing some $\alpha(x_j)$ occurs in the *mid* level of $T_i^\perp$. By Lemma 5, $\alpha(x_j)$ occurs in the *mid* level of $T_i^\perp$ if and only if y_i and x_j are orthogonal. □

We conjecture that the reduction underlying Theorem 11 can be tweaked to prove a quadratic conditional lower bound for MATCH(SOLID,*ED*) in the case of a binary alphabet and $\Delta_{loc} = \mathcal{O}(1)$. We leave this as an open problem.

Open Problems

This work classified the complexity of problem MATCH(X, Y) as either truly subquadratic or at least quadratic (conditioned on SETH) for almost all combinations of types of variable strings X and Y. The two main cases that remain open are when $X =$ SOLID and $Y = GD$, and when $X =$ SOLID and $Y = F$. For these problems, corresponding to the yellow cells in Table 1, our algorithms, although subquadratic in most cases, are quadratic in the worst case, and no lower bound is known. Furthermore, it remains to determine the time complexity of matching a 1-D pattern in a k-D, GD or ED text when the maximum length of the strings in the segments of the text is at most logarithmic in the text length; and symmetrically, the time complexity of matching a k-D, GD or ED pattern in a 1-D text when the maximum length of the strings in the segments of

the pattern is at most logarithmic. Finally, it remains to show whether the quadratic conditional lower bound for matching a solid pattern in an *ED* text holds when the alphabet is binary and has constant local elasticity.

Acknowledgements
This work was partially supported by the PANGAIA (RG and NP) and ALPACA (EG and NP) projects that received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No. 872539 and 956229, respectively. GB is a member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM) and was partially supported by the MUR - FSE REACT EU - PON R&I 2014–2020 and the INdAM - GNCS Project CUP_E53C24001950001. ME was funded by the Helsinki Institute for Information Technology (HIIT). RG and NP were partially supported by NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem. NP was partially supported by MUR PRIN 2022 YRB97K PINC.

Author contributions
R.A., G.B., M.E., E.G., and R.G. designed the algorithms and the reductions; R.A., G.B., M.E., E.G. and N.P. wrote the main manuscript text; R.A. and A.C. prepared the figures; G.B. coordinated the work; all authors reviewed the manuscript.

Data availability
No datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Received: 1 November 2024 Accepted: 17 June 2025
Published online: 28 April 2026

References

1. Abrahamson KR. Generalized string matching. SIAM J Comput. 1987;16(6):1039–51.
2. Alanko JN, Biagi E, Puglisi SJ, Vuohtoniemi J. Subset wavelet trees. In: 21st International Symposium on Experimental Algorithms (SEA), volume 265 of LIPIcs, pages 4:1–4:14, 2023.
3. Alzamel M, Bernardini G, Grossi RP, Iliopoulos CS, Pisanti NP, Pissis SP, Rosone G. Degenerate string comparison and applications. In: 18th International Workshop on Algorithms in Bioinformatics (WABI), volume 113 of LIPIcs, pages 21:1–21:14, 2018.
4. Alzamel M, Ayad LAK, Bernardini G, Grossi R, Iliopoulos CS, Pisanti N, Pissis SP, Rosone G. Comparing degenerate strings. Fundam Informaticae. 2020;175(1–4):41–58.
5. Amir A, Itzhaki M. Reconstructing General Matching Graphs. In 35th Annual Symposium on Combinatorial Pattern Matching (CPM 2024), volume 296 of Leibniz International Proceedings in Informatics (LIPIcs), pages 2:1–2:15, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
6. Antoniou P, Crochemore M, Iliopoulos CS, Jayasekera I, Landau GM. Conservative string covering of indeterminate strings. In: Proceedings of the Prague Stringology Conference, pages 2008;108–115.
7. Aoyama K, Nakashima Y, Tomohiro I, Inenaga S, Bannai H, Takeda M. Faster online elastic degenerate string matching. In: 29th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 105 of LIPIcs, pages 9:1–9:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.
8. Ascone R, Bernardini G, Conte A, Equi M, Gabory E, Grossi R, Pisanti N. A unifying taxonomy of pattern matching in degenerate strings and founder graphs. In: 24th International Workshop on Algorithms in Bioinformatics, (WABI), volume 312 of LIPIcs, pages 14:1–14:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
9. Baaijens JA, Bonizzoni P, Boucher C, Vedova GD, Pirola Y, Rizzi R, Sirén J. Computational graph pangenomics: a tutorial on data structures and their applications. Nat Comput. 2022;21(1):81–108.
10. Backurs A, Indyk P. Which regular expression patterns are hard to match? CoRR, arXiv:1511.07070, 2015.
11. Backurs A, Indyk P. Which regular expression patterns are hard to match? In: 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS), 457–466, 2016.
12. Barton C. On the average-case complexity of pattern matching with wildcards. Theor Comput Sci. 2022;922:37–45.
13. Bathie G, Charalampopoulos P, Starikovskaya T. Pattern matching with mismatches and wildcards. In: Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, 32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom, volume 308 of LIPIcs, 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
14. Battaglia G, Cangelosi D, Grossi R, Pisanti N. Masking patterns in sequences: a new class of motif discovery with don't cares. Theor Comput Sci. 2009;410(43):4327–40.
15. Bernardini G, Gabory E, Pissis SP, Stougie L, Sweering M, Zuba W. Elastic-degenerate string matching with 1 error. In: 15th Latin American Symposium on Theoretical Informatics (LATIN), volume 13568 of Lecture Notes in Computer Science, 20–37. Springer, 2022.
16. Bernardini G, Gawrychowski P, Pisanti N, Pissis SP, Rosone G. Even faster elastic-degenerate string matching via fast matrix multiplication. In: 46th International Colloquium on Automata, Languages, and Programming (ICALP), volume 132 of LIPIcs, 2019;21:1–21:15.
17. Bernardini G, Gawrychowski P, Pisanti N, Pissis SP, Rosone G. Elastic-degenerate string matching via fast matrix multiplication. SIAM J Comput. 2022;51(3):549–76.
18. Bernardini G, Pisanti N, Pissis SP, Rosone G. Pattern matching on elastic-degenerate text with errors. In: 24th International Symposium on String Processing and Information Retrieval (SPIRE), volume 10508 of Lecture Notes in Computer Science, 74–90. Springer, 2017.
19. Bernardini G, Pisanti N, Pissis SP, Rosone G. Approximate pattern matching on elastic-degenerate text. Theor Comput Sci. 2020;812:109–22.
20. Bille P, Li Gørtz I, Stordalen T. Rank and select on degenerate strings. In: 2024 Data Compression Conference (DCC), 2024;283–292.
21. Büchler T, Olbrich J, Ohlebusch E. Efficient short read mapping to a pangenome that is represented by a graph of ED strings. Bioinformatics. 2023;39(5):btad320.
22. Cicherski A, Lisiecka A, Dojer N. Alfapang: Alignment free algorithm for pangenome graph construction. In: 24th International Workshop on Algorithms in Bioinformatics, WABI, volume 312 of LIPIcs, 2024;23:1–23:18.
23. Cislak A, Grabowski S, Holub J. Sopang: online text searching over a pangenome. Bioinform. 2018;34(24):4290–2.
24. Clifford P, Clifford R. Simple deterministic wildcard matching. Inf Process Lett. 2007;101(2):53–4.
25. Cole R, Hariharan R. Tree pattern matching and subset matching in randomized o(n $\log^3$m) time. In Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC), 66–75. ACM, 1997.
26. Cole R, Hariharan R. Verifying candidate matches in sparse and wildcard matching. In: Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC), 592–601. ACM, 2002.
27. Cole R, Hariharan R. Tree pattern matching to subset matching in linear time. SIAM J Comput. 2003;32(4):1056–66.
28. The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. Brief Bioinformatics 2018;19(1):118–135.
29. Crochemore M, Hancart C, Lecroq T. Algorithms on strings. Cambridge: Cambridge University Press; 2007.
30. Crochemore M, Iliopoulos CS, Kociumaka T, Radoszewski J, Rytter W, Walen T. Covering problems for partial words and for indeterminate strings. Theor Comput Sci. 2017;698:25–39.
31. Danecek P, Auton A, Abecasis GR, Albers CA, Banks E, DePristo MA, Handsaker RE, Lunter G, Marth GT, Sherry ST, McVean G, Durbin R. The variant call format and vcftools. Bioinform. 2011;27(15):2156–8.
32. Daykin JW, Groult R, Guesnet Y, Lecroq T, Lefebvre A, Léonard M, Mouchard L, Prieur É, Watson BW. Efficient pattern matching in degenerate strings with the burrows-wheeler transform. Inf Process Lett. 2019;147:82–7.

33. Daykin JW, Watson BW. Indeterminate string factorizations and degenerate text transformations. Math Comput Sci. 2017;11(2):209–18.
34. Doerr D, Stoye J, Böcker S, Jahn K. Identifying gene clusters by discovering common intervals in indeterminate strings. BMC Genomics. 2014;15(Suppl 6):S2.
35. Dorey-Robinson D, Maccari G, Hammond JA. Igmat: immunoglobulin sequence multi-species annotation tool for any species including those with incomplete antibody annotation or unusual characteristics. BMC Bioinform. 2023;24(1):491.
36. Eizenga JM, Novak AM, Kobayashi E, Villani F, Cisar C, Heumos S, Hickey G, Colonna V, Paten B, Garrison E. Efficient dynamic variation graphs. Bioinform. 2021;36(21):5139–44.
37. Equi M, Mäkinen V, Tomescu AI. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In: 47th International Conference on Current Trends in Theory and Practice of Computer Science, (SOFSEM), volume 12607 of Lecture Notes in Computer Science, pages 608–622. Springer, 2021.
38. Equi M, Mäkinen V, Tomescu AI. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. Theor Comput Sci. 2023;975: 114128.
39. Equi M, Mäkinen V, Tomescu AI, Grossi R. On the complexity of string matching for graphs. ACM Trans Algorithms. 2023;19(3):21:1-21:25.
40. Equi M, Norri T, Alanko J, Cazaux B, Tomescu AI, Mäkinen V. Algorithms and complexity on indexing elastic founder graphs. In: 32nd International Symposium on Algorithms and Computation (ISAAC), volume 212 of LIPIcs, 2021;20:1–20:18.
41. Equi M, Norri T, Alanko J, Cazaux B, Tomescu AI, Mäkinen V. Algorithms and complexity on indexing founder graphs. Algorithmica. 2023;85(6):1586–623.
42. Liao, et al. A draft human pangenome reference. Nature. 2023;617(7960):312–24.
43. Farach M. Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997, 137–143, 1997.
44. Federico M, Pisanti N. Suffix tree characterization of maximal motifs in biological sequences. Theor Comput Sci. 2009;410(43):4391–401.
45. Gabory E, Mwaniki MN, Pisanti N, Pissis SP, Radoszewski J, Sweering M, Zuba W. Elastic-degenerate string comparison. Inf Comput. 2025;304: 105296.
46. Gabory E, Mwaniki NM, Pisanti N, Pissis SP, Radoszewski J, Sweering M, Zuba W. Comparing elastic-degenerate strings: Algorithms, lower bounds, and applications. In: 34th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 259 of LIPIcs, 11:1–11:20, 2023.
47. Gabory E, Mwaniki NM, Pisanti N, Pissis SP, Radoszewski J, Sweering M, Zuba W. Pangenome comparison via ED strings. Front Bioinformatics. 2024;4:11:1-11:20.
48. Garrison E, Sirén J, Novak AM, Hickey G, Eizenga JM, Dawson ET, Jones W, Garg S, Markello C, Lin MF, Paten B, Durbin R. Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nat Biotechnol. 2018;36(9):875–9.
49. Gawrychowski P, Ghazawi S, Landau GM. On indeterminate strings matching. In: 31st Annual Symposium on Combinatorial Pattern Matching (CPM), volume 161 of LIPIcs, 2020;14:1–14:14.
50. Gibney D. An efficient elastic-degenerate text index? not likely. In: 27th International Symposium on String Processing and Information Retrieval, volume 12303 of Lecture Notes in Computer Science, 76–88. Springer, 2020.
51. Gibney D, Hoppenworth G, Thankachan SV. Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In: 4th SIAM Symposium on Simplicity in Algorithms (SOSA), 2021;232–242.
52. Koerkamp RG. A*PA2: Up to 19× faster exact global alignment. In 24th International Workshop on Algorithms in Bioinformatics (WABI), volume 312 of LIPIcs, 17:1–17:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
53. Grossi R, Iliopoulos CS, Liu C, Pisanti N, Pissis SP, Retha A, Rosone G, Vayani F, Versari L. On-line pattern matching on similar texts. In 28th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 78 of LIPIcs, 2017;9:1–9:14.
54. Grossi R, Pietracaprina A, Pisanti N, Pucci G, Upfal E, Vandin F. MADMX: A novel strategy for maximal dense motif extraction. In Steven Salzberg and Tandy J. Warnow, editors, Algorithms in Bioinformatics, 9th International Workshop, WABI 2009, Philadelphia, PA, USA, September 12-13, 2009. Proceedings, volume 5724 of Lecture Notes in Computer Science, 362–374. Springer, 2009.
55. Grossi R, Pietracaprina A, Pisanti N, Pucci G, Upfal E, Vandin F. MADMX: a strategy for maximal dense motif extraction. J Comput Biol. 2011;18(4):535–45.
56. Gu M, Farach M, Beigel R. An efficient algorithm for dynamic text indexing. In: Proceedings of the 5th annual ACM-SIAM symposium on Discrete algorithms (SODA), 1994;697–704.
57. Holub J, Smyth WF, Wang S. Fast pattern-matching on indeterminate strings. J Discrete Algorithms. 2008;6(1):37–50.
58. Iliopoulos CS, Kundu R, Pissis SP. Efficient pattern matching in elastic-degenerate strings. Inf Comput. 2021;279: 104616.
59. Iliopoulos CS, Mouchard L, Rahman MS. A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. Math Comput Sci. 2008;1(4):557–69.
60. Iliopoulos CS, Radoszewski J. Truly subquadratic-time extension queries and periodicity detection in strings with uncertainties. In: 27th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 54 of LIPIcs, 2016;8:1–8:12.
61. Impagliazzo R, Paturi R. On the complexity of k-sat. J Comput Syst Sci. 2001;62(2):367–75.
62. IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. Biochemistry, 9(20):4022–4027, 1970.
63. Knuth DE, Morris Jr JH, Pratt VR. Fast pattern matching in strings. SIAM J Comput. 1977;6(2):323–50.
64. Lemane T, Chikhi R, Peterlongo P. KMDIFF: large-scale and user-friendly differential k-mer analyses. Bioinform. 2022;38(24):5443–5.
65. Louza FA, Mhaskar N, Smyth WF. A new approach to regular & indeterminate strings. Theor Comput Sci. 2021;854:105–15.
66. Mäkinen V, Cazaux B, Equi M, Norri T, Tomescu AI. Linear time construction of indexable founder block graphs. In:20th International Workshop on Algorithms in Bioinformatics (WABI), volume 172 of LIPIcs, 2020;7:1–7:18.
67. Mwaniki NM, Garrison E, Pisanti N. Fast exact string to D-texts alignments. In 16th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC), 70–79. SCITEPRESS, 2023.
68. Mwaniki NM, Pisanti N. Optimal sequence alignment to ED-strings. In: 18th International Symposium Bioinformatics Research and Applications (ISBRA), volume 13760 of Lecture Notes in Computer Science, pages 204–216. Springer, 2022.
69. Paten B, Novak AM, Eizenga JM, Garrison E. Genome graphs and the evolution of genome inference. Genome Res. 2017;27(5):665–76.
70. Peterlongo P, Pisanti N, Boyer F, do Lago AP, Sagot M-F. Lossless filter for multiple repetitions with hamming distance. J Discrete Algorithms. 2008;6(3):497–509.
71. Peterlongo P, Pisanti N, Boyer F, Sagot M-F. Lossless filter for finding long multiple approximate repetitions using a new data structure, the bi-factor array. In Mariano P. Consens and Gonzalo Navarro, editors, String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005, Proceedings, volume 3772 of Lecture Notes in Computer Science, 179–190. Springer, 2005.
72. Peterlongo P, Sacomoto GAT, do Lago AP, Pisanti N, Sagot M-F. Lossless filter for multiple repeats with bounded edit distance. Algorithms Mol Biol. 2009;4:1–20.
73. Pisanti N, Soldano H, Carpentier M. Incremental inference of relational motifs with a degenerate alphabet. In 16th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 3537 of Lecture Notes in Computer Science, 229–240. Springer, 2005.
74. Pisanti N, Soldano H, Carpentier M, Pothier J. A relational extension of the notion of motifs: application to the common 3d protein substructures searching problem. J Computut Biol. 2009;16(12):1635–60.
75. Pissis SP. Optimal prefix-suffix queries with applications. In: 2025 Symposium on Simplicity in Algorithms (SOSA), 166–171. SIAM, 2025.
76. Pissis SP, Retha A. Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line. In: 17th International Symposium on Experimental Algorithms (SEA), volume 103 of LIPIcs, 2018;16:1–16:14.

77. Procházka P, Cvacho O, Krcál L, Holub J. Backward pattern matching on elastic-degenerate strings. SN Comput Sci. 2023;4(5):442.
78. Rakocevic G, Semenyuk V, Lee W-P, Spencer J, Browning J, Johnson IJ, Arsenijevic V, Nadj J, Ghose K, Suciu MC, Ji S-G, Demir G, Li L, Toptaş BÇ, Dolgoborodov A, Pollex B, Spulber I, Glotova I, Kómár P, Stachyra AL, Li Y, Popovic M, Källberg M, Jain A, Kural D. Fast and accurate genomic analyses using genome graphs. Nat Genet. 2019;51:354–62.
79. Rizzo N, Equi M, Norri T, Mäkinen V. Elastic founder graphs improved and enhanced. Theor Comput Sci. 2024;982: 114269.
80. Rizzo N, Mäkinen V. Indexable elastic founder graphs of minimum height. In: 33rd Annual Symposium on Combinatorial Pattern Matching (CPM), volume 223 of LIPIcs, 2022;19:1–19:19.
81. Rizzo N, Mäkinen V. Linear time construction of indexable elastic founder graphs. In 33rd International Workshop on Combinatorial Algorithms (IWOCA), volume 13270 of Lecture Notes in Computer Science, 480–493. Springer, 2022.
82. Sagot M-F, Viari A, Soldano H. Multiple sequence comparison: A peptide matching approach. In: 6th Annual Symposium on Combinatorial Pattern Matching (CPM), volume 937 of Lecture Notes in Computer Science, pages 366–385. Springer, 1995.
83. Sagot M-F, Viari A, Soldano H. Multiple sequence comparison–a peptide matching approach. Theor Comput Sci. 1997;180(1–2):115–37.
84. Shiftan A, Porat E. Set intersection and sequence matching with mismatch counting. Theor Comput Sci. 2016;638:3–10.
85. Sirén J, Garrison E, Novak AM, Paten B, Durbin R. Haplotype-aware graph indexes. In: 18th International Workshop on Algorithms in Bioinformatics (WABI), volume 113 of LIPIcs, pages 2018;4:1–4:13.
86. Soldano H, Viari A, Champesme M. Searching for flexible repeated patterns using a non-transitive similarity relation. Pattern Recognit Lett. 1995;16(3):233–46.
87. Thachuk C. Indexing hypertext. J Discrete Algorithms. 2013;18:113–22.
88. Weiner P. Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973, 1–11, 1973.
89. Williams R. A new algorithm for optimal 2-constraint satisfaction and its implications. Theor Comput Sci. 2005;348(2–3):357–65.

Publisher's Note