



HAL
open science

A Unifying Taxonomy of Pattern Matching in Degenerate Strings and Founder Graphs

Rocco Ascone, Giulia Bernardini, Alessio Conte, Massimo Equi, Esteban Gabory, Roberto Grossi, Nadia Pisanti

► **To cite this version:**

Rocco Ascone, Giulia Bernardini, Alessio Conte, Massimo Equi, Esteban Gabory, et al.. A Unifying Taxonomy of Pattern Matching in Degenerate Strings and Founder Graphs. WABI 2024 - Workshop on Algorithms in BioInformatics, Sep 2024, London, United Kingdom. 10.4230/LIPIcs.WABI.2024.14 . hal-04747535

HAL Id: hal-04747535

<https://inria.hal.science/hal-04747535v1>

Submitted on 22 Oct 2024



HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.



L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License



A Unifying Taxonomy of Pattern Matching in Degenerate Strings and Founder Graphs



Rocco Ascone  
University of Trieste, Italy



Giulia Bernardini  
University of Trieste, Italy

Alessio Conte  
University of Pisa, Italy

Massimo Equi  
University of Helsinki, Finland

Esteban Gabory  
CWI, Amsterdam, The Netherlands

Roberto Grossi  
University of Pisa, Italy

Nadia Pisanti  
University of Pisa, Italy

Abstract

Elastic Degenerate (ED) strings and Elastic Founder (EF) graphs are two versions of acyclic components of pangenomes. Both ED strings and EF graphs (which we collectively name *variable strings*) extend the well-known notion of indeterminate string. Recent work has extensively investigated algorithmic tasks over these structures, and over several other variable strings notions that they generalise. Among such tasks, the basic operation of matching a pattern into a text, which can serve as a toolkit for many pangenomic data analyses using these data structures, deserves special attention. In this paper we: (1) highlight a clear taxonomy within both ED strings and EF graphs ranging through variable strings of all types, from the linear string up to the most general one; (2) investigate the problem $PVART(X, Y)$ of matching a solid or variable pattern of type X into a variable text of type Y ; (3) using as a reference the quadratic conditional lower bounds that are known for $PVART(SOLID, ED)$ and $PVART(SOLID, EF)$, for all possible types of variable strings X and Y we either prove the quadratic conditional lower bound for $PVART(X, Y)$, or provide non-trivial, often sub-quadratic, upper bounds, also exploiting the above-mentioned taxonomy.

2012 ACM Subject Classification Theory of computation \rightarrow Pattern matching; Theory of computation \rightarrow Problems, reductions and completeness; Applied computing \rightarrow Molecular sequence analysis; Applied computing \rightarrow Computational genomics

Keywords and phrases Pangenomics, pattern matching, degenerate string, founder graph, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.WABI.2024.14

Funding 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. RG and NP were partially supported by NextGeneration EU programme PNRR ECS00000017 Tuscany Health Ecosystem.

Giulia Bernardini: GB is a member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM) and was supported by the MUR - FSE REACT EU - PON R&I 2014-2020.

Massimo Equi: ME was funded by the Helsinki Institute for Information Technology (HIIT).

Nadia Pisanti: NP was partially supported by MUR PRIN 2022 YRB97K PINC.



© Rocco Ascone, Giulia Bernardini, Alessio Conte, Massimo Equi, Esteban Gabory, Roberto Grossi, and Nadia Pisanti;

licensed under Creative Commons License CC-BY 4.0

24th International Workshop on Algorithms in Bioinformatics (WABI 2024).

Editors: Solon P. Pissis and Wing-Kin Sung; Article No. 14; pp. 14:1–14:21

Leibniz International Proceedings in Informatics



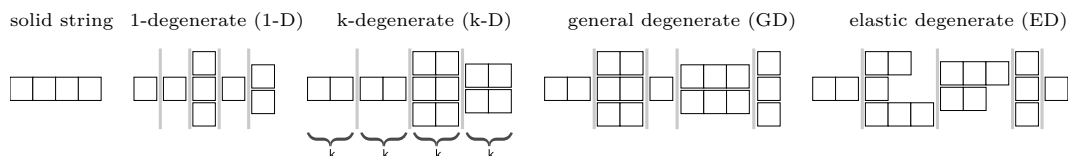
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Genomic data analysis has been facing important challenges that include analyzing an ever-increasing number of genome sequences and choosing which genome should be used as a *reference*. In recent years, these two challenges were merged into the powerful opportunity of using a *pangenome* – rather than a single genome – as a reference. According to [21], a *pangenome* is indeed “any collection of genomic sequences to be analyzed jointly or to be used as a reference”. As a consequence, the new -omics science *pangenomics* 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 [52, 34]. As opposed to a *linear* reference, a pangenome reference allows a simultaneous representation, in a compact manner, of variations and commonalities among the underlying sequences. The most general pangenome representations are edge- and/or node-labeled directed graphs [8], such as the *variation graphs* [27, 28] (with their haplotype aware version of [64]) and *sequence graphs* [57]. Simpler *acyclic* alternatives to these representations are *Elastic-Degenerate strings* [39, 42] (ED strings) and *Elastic Founder graphs* [58] (EF graphs), which are slightly more haplotype-aware, as well as their non-elastic versions *Degenerate strings* [3, 4, 50] (D strings) and *Founder graphs* [49] (F graphs).

Figure 1 shows the taxonomy of (elastic/generalised) degenerate strings, from the simple solid pattern (left) to the most general *ED* string (right). The simplest case of degeneracy, the 1-*D* strings, are well known as *indeterminate* strings in the bioinformatics literature. They have been extensively employed and investigated [1, 61, 65, 62, 53, 43, 25, 22, 24, 36, 26, 2, 15, 44, 6, 48, 41, 54, 5] as they naturally represent the IUPAC encoding [46] of DNA and RNA nucleotides subsets and can be used to highlight SNPs. 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. Finally, *ED* strings allow variants of any size even at the same locus, including the option of the empty string that allows explicitly representing short INDELS. *ED* strings correspond to the VCF file format [23] 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: from left to right, we would have 1-*F*, *k-F*, *F*, *EF*. We use the term *variable string/text* to mean either *ED* string or *EF* graph or any of their above-mentioned special cases.

In contrast with more general representations, *ED* strings support both theoretically [7, 11, 12] and practically [39, 55, 17, 16, 55, 56] fast *on-line* exact and approximate pattern matching [10, 13, 14] when the pattern is a solid (i.e. non-variable) string. Moreover, it can be efficiently decided whether two *ED* strings share a string [35] (*intersection query*). Finally, *GD* strings support fast dynamic-programming-based alignment [50, 51] for approximate matching with a solid string, and linear-time answer to the intersection query [3, 4].



■ **Figure 1** Degenerate strings in increasing order of generality. Squares represent characters, and segments are separated by grey bars. An analogous parallel progression can be traced for founder graphs by adding edges between consecutive segments.

On the other hand, ED strings cannot be indexed efficiently [37]. 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 MSA (for F graphs), or a near-linear time construction from a general MSA (for EF graphs) [32, 33, 66, 59, 60, 49]. Here, “near-linear time” means that the time complexity is linear when parametrized in the maximum number of variations appearing in a single locus of the EF graph.

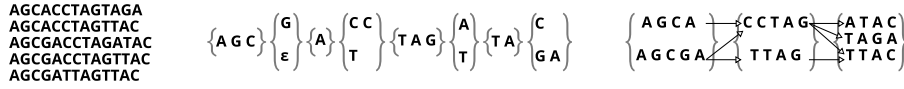
While supporting provably efficient and accurate methods, both ED strings and EF graphs are acyclic structures that impose a global alignment allowing only matches, mismatches, and short INDELS as variants: they cannot adequately represent structural variation such as repetitions, translocations, or inversions. Computational pangenomics must balance efficiency, accuracy, and the complexity of representing variable strings as graphs [21]. This paper aims to provide a clear taxonomy of matching problems involving variable strings.

Pattern matching is a basic toolkit for many (pan)genomic data analysis tasks. In this work, we analyse the complexity of the problem, denoted $\text{PVART}(X, Y)$, of matching a pattern of type X in a text of type Y , where both X and Y can be any of the types of variable strings described above: for instance, $\text{PVART}(1-D, k-F)$ is the problem of finding occurrences of a $1-D$ pattern within a $k-F$ graph.

For the problem of matching a solid (non-variable) pattern of size M into an ED or EF text of size N , it is known that an algorithm with complexity $O(M^{1-\epsilon}N)$ or $O(MN^{1-\epsilon})$ with $\epsilon > 0$ would contradict **SETH** [9, 11, 38, 32, 12], and the contradiction holds even if such complexity is achieved at query time after a polynomial-time indexing step [37, 29, 30], while quadratic-time algorithms are known. Moreover, strongly sub-quadratic algorithms are known for $\text{PVART}(\text{SOLID}, 1-D)$ [19] and for $\text{PVART}(1-D, 1-D)$ [44], in the latter case restricted to constant-size alphabets.

■ **Table 1** Complexity chart for problem $\text{PVART}(X, Y)$, where X ranges over rows and Y over columns. Green cells are for truly subquadratic $O(NM^{1-\epsilon})$ (for some $\epsilon > 0$) upper bounds, yellow cells are for subquadratic upper bounds under special conditions, and red cells are for a quadratic lower bound $\Omega((MN)^{1-\epsilon})$ (for every $\epsilon > 0$) conditioned on **SETH**, even with a constant-size alphabet (our bounds are even tighter, check the referred results for details). Capital 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 2). Note that the reduction for EF [32] implies also a $\Omega((mN)^{1-\epsilon})$ bound. Symbol \dagger indicates that the bound holds only for constant-size alphabets, as it has an exponential dependency on the alphabet size; symbol \ddagger indicates that the bound is subquadratic whenever $N/n = \omega(1)$. Only the subquadratic upper bounds for $\text{PVART}(\text{SOLID}, 1-D)$, $\text{PVART}(1-D, 1-D)$ and the quadratic lower bounds for $\text{PVART}(\text{SOLID}, ED)$, $\text{PVART}(\text{SOLID}, EF)$ were known before this work; the other results are proven in this paper.

Text Pattern	1-D	1-F	k-D	k-F	GD	F	ED	EF
SOLID	[19] $O(n \log^2 m)$	Thm. 2 $O(\sqrt{m}(E + N \log^2 m))$	Thm. 1 $O(N + N \cdot \log^2 m)$	Thm. 2 $O(\sqrt{m}(E + N \log^2 m))$	Thm. 3\ddagger $O(nm + N)$	Thm. 4\ddagger $O(nm + N + E \log m)$	[9] $\Omega((mN)^{1-\epsilon})$	[32] $\Omega((m E)^{1-\epsilon})$
1-D	[44] $O(n \log m)^\dagger$	Thm. 7 $\Omega((MN)^{1-\epsilon})$	Thm. 5 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$
1-F	Thm. 8 $\Omega((MN)^{1-\epsilon})$	Thm. 9 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$
k-D	Thm. 6 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$
k-F, F, EF	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$	Cor. 13 $\Omega((MN)^{1-\epsilon})$	Cor. 14 $\Omega((MN)^{1-\epsilon})$
GD, ED	Cor. 11 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$	Cor. 10 $\Omega((MN)^{1-\epsilon})$	Cor. 12 $\Omega((MN)^{1-\epsilon})$



■ **Figure 2** An ED string (center) and an EF graph (right) built from the same set of strings (left).

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 $\text{PVART}(X, Y)$, or a better algorithm – an upper bound – should be exhibited. To the purpose of exhaustively performing this task for all types X of patterns, and all types Y of variable texts, our contribution is to complete Table 1, where columns correspond to the pattern and rows correspond to the text.

All our results are anticipated in Section 2 and summarized in Table 1. Note that, due to space constraints, in Table 1 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 $\epsilon > 0$).

2 Definitions and summary of the results

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. $N = N_\epsilon + \sum_{i=1}^n \sum_{S \in T[i]} |S|$, 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 $1 \leq i \leq j \leq n$, $T[i..j]$ denotes the fragment $T[i] \cdots T[j]$ between segments i and j of T .

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 $k = 1$, then 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, that is a standard plain text.

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, 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 $n = 8$ and size $N = 19$, and an EF graph with $n = 3$, $N = 30$, $|E| = 7$, and hence size 37.

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: $\mathcal{L}(G) = \{S_1 \cdots S_n : S_i \in T[i] \forall i \in [1, n] \text{ and } (S_i, S_{i+1}) \in E_i \forall i \in [1, n-1]\}$. 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 includes the set of the 5 strings shown on the left plus $\{\text{AGCACCTAGATAC}, \text{AGCGACCTAGATAC}\}$, while the language of the ED string includes even more strings, for a total of 16. Informally, we say that there is an occurrence of a (possibly 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 Fig. 2, the solid pattern CTAGA occurs in ED ending at position 6, and in EF ending at position 3. We provide formal definitions of variable pattern matching for the cases studied in this paper in sections 3 and 4.

PATTERN MATCHING ON VARIABLE TEXT: PVART(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 $T[j]$

Note that X and Y can be any of the string types we defined above. Note that, following the existing literature [39], the output of PVART only specifies the ending segments of the occurrences, and gives no information about the specific position within the segment. We say that PVART has *constant alphabet* if the alphabet Σ of the strings in both the pattern and the text is of constant size. Here we anticipate the results that we will prove in Sections 3 and 4, consisting of upper and lower bounds for the problem PVART(X, Y) for several choices of pattern type X and text type Y where: m and M are, respectively, the length and the size of the pattern; n , N are, respectively, the length and the size of the text; and finally, E refers to the edges of the text when this is a founder graph.

Upper bounds

We give two sub-quadratic upper bounds for PVART(X, Y) when $X=\text{SOLID}$ (solid pattern) and Y is either a k - D string or a k - F graph.

► **Theorem 1.** PVART(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.** PVART(SOLID, k - F) can be solved in $\mathcal{O}(\sqrt{m}(|E| + N \log^2 m))$ time.

When the segments of the text do not have a fixed number of characters, a quadratic term appears in our time complexities, that remain sub-quadratic only when $N/n = \omega(1)$.

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

► **Theorem 4.** PVART(SOLID, F) can be solved in $\mathcal{O}(nm + N + |E| \log m)$ time.

We remark that the bounds of Theorem 3 and 4 are only subquadratic when $N/n = \omega(1)$; whether there exist subquadratic algorithms for the case $N/n = \mathcal{O}(1)$ is an open problem, as no conditional lower bound for PVART(SOLID, GD) or PVART(SOLID, F) is known yet.

Lower bounds

When both the pattern and the text are variable strings, we show conditional lower bounds for PVART(X, Y), which all hold even for constant alphabets (except for PVART(1- D , 1- D)). In Section 4.2 we discuss more in detail the case PVART(1- D , 1- D), for which a quadratic conditional lower bound exists in the literature for non-constant alphabets.

► **Theorem 5.** No algorithm can solve PVART(1- D , k - 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 6.** No algorithm can solve PVART(k - D , 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 7.** *No algorithm can solve $\text{PVART}(1-D,1-F)$ 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 8.** *No algorithm can solve $\text{PVART}(1-F,1-D)$ 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 9.** *No algorithm can solve $\text{PVART}(1-F,1-F)$ 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.*

Since a pattern of type $1-D$ is a special case of $k-D$, GD or ED (and $k-D$ is a special case of GD or 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. The first one directly follows from Theorem 5.

► **Corollary 10.** *No algorithm can solve $\text{PVART}(X,Y)$ for $X = 1-D, k-D, GD, ED$ 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.*

► **Corollary 11.** *No algorithm can solve $\text{PVART}(X,1-D)$ 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 11 above follows from Theorem 6, while Corollary 12, 13 and 14 below are from Theorems 7, 8 and 9, respectively.

► **Corollary 12.** *No algorithm can solve $\text{PVART}(X,Y)$ 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 13.** *No algorithm can solve $\text{PVART}(X,Y)$ 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 14.** *No algorithm can solve $\text{PVART}(X,Y)$ 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.*

3 Matching a solid pattern in GD and F

Let us start by defining an occurrence of a solid pattern into a GD string or an F graph.

► **Definition 15.** *A solid pattern P of length m has an occurrence ending at position j in a GD or F text T of length n 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 $P \in \mathcal{L}(T^{\ell,r}[i..j])$, where $T^{\ell,r}[i..j]$ is obtained from the fragment $T[i..j]$ 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 $T[j]$. We call ℓ and $k_j - r$ the starting and ending offset of the occurrence, respectively.*

In Theorem 3, we show that the on-line $\mathcal{O}(m^2n + N)$ -time algorithm proposed in [39] for pattern matching in ED texts requires only $\mathcal{O}(mn + N)$ time when applied to GD texts.

► **Theorem 3.** *$\text{PVART}(\text{SOLID},GD)$ can be solved in $\mathcal{O}(nm + N)$ time.*

Proof. Consider the pattern-matching algorithm of [39] for ED texts. The algorithm consists of reading the text T segment by segment, from left to right, after constructing the suffix tree \mathcal{T}_P of the pattern P in a preprocessing step. Given a segment $T[i]$ of width k_i , the 4 following sub-problems are solved: (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) 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: it is straightforward to solve the easy and the suffix/prefix case in $\mathcal{O}(N + m)$ time using any standard pattern-matching algorithm (e.g. KMP [47]) and using \mathcal{T}_P , respectively (see [39] 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 [11]. AP applied to segment $T[i]$ asks, given 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 $\mathcal{L}(T[1..i-1])$, to produce a second bit-vector V of size m such that $V[j + k_i] = 1$ 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 $j + 1$).

To solve AP it thus suffices to compute all the occurrences in P of all the strings from $T[i]$ and check whether they extend some active prefixes stored in U . Since the strings in $T[i]$ are all of the same length k_i , thus no two distinct such strings can occur at the same position in P , we have the following crucial observation.

► **Observation 16.** *The cumulative number of occurrences in P of all the strings from a GD segment $T[i]$ is bounded by m .*

Observation 16 implies that all such occurrences can be computed and stored in an auxiliary bit-vector OCC of size m in $\mathcal{O}(N_i + m)$ time using \mathcal{T}_P . V can then be obtained by left-shifting OCC 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 $\mathcal{O}(N + nm)$ time. Note that the difference in the time complexities of AP for GD texts and ED texts is because, in the latter case, Observation 16 does not hold, and the number of occurrences of the strings from $T[i]$ in P can only be bounded by m^2 . ◀

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

► **Theorem 4.** *PVART(SOLID, F) can be solved in $\mathcal{O}(nm + N + |E| \log m)$ 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 $T[i]$, $j \in [1, B_i]$, 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 all proper suffix/prefix and prefix/suffix overlaps between P and each string from each segment, i.e., for each string, we compute the length of all suffixes that are equal to prefixes of P , and of all prefixes that are equal to suffixes of P . While in the case of GD text, it suffices to store the length 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 $T[i][j]$ has a

suffix/prefix overlap of length ℓ with P , and $e_{i,j}[\ell] = 1$ if and only if $T[i][j]$ has a prefix/suffix overlap of length ℓ with 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 $\mathcal{O}(N)$.

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 $T[i], T[i + 1]$.

► **Observation 17.** *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 the *border tree* [40] of P , a data structure that, given any pair of positions $1 \leq s \leq p \leq m$, returns 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 $m - s + 1$. The border tree can be constructed in $\mathcal{O}(m)$ time and answers queries in $\mathcal{O}(\log m + |\text{occ}|)$ time [40]. After constructing the border tree of 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 17: for each $(j, j') \in E_i$, query the border tree of P 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 $\mathcal{O}(\log m + |\text{occ}|)$ time [40], and $|\text{occ}| \leq m$. 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 $\mathcal{O}(|E_i| \log m + m)$, implying time $\mathcal{O}(|E| \log m + mn)$ 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: 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 $k_{i-1} > m$). 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 18. 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* of size m such that $\text{OCC}[\ell] = j$ iff $T[i][j]$ occurs in P starting at position ℓ ; by Observation 16, *OCC* is well defined and can be computed in $\mathcal{O}(N_i + m)$ time using the suffix tree \mathcal{T}_P . Note that *OCC* 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 16.

► **Observation 18.** *Any partial occurrence of P ending at $T[i-1]$ can be extended by at most 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 $V_i[k_i + k_{i-1} \dots m]$: for each position $\ell \in [k_i + k_{i-1}, m]$, we set $V_i[\ell] = j'$ iff $OCC[\ell - k_i + 1] = j'$, $V_{i-1}[\ell - k_i] = j$ 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 $OCC[\ell - k_i + 1] = j'$, 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 $\mathcal{O}(nm + |E|)$ time over the whole G .

Let us now focus on computing $V_i[k_i \dots k_i + k_{i-1} - 1]$. 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 $OCC[2 \dots k_{i-1}]$. Note that we cannot use the same technique as for $V_i[k_i + k_{i-1} \dots m]$, because two strings from $T[i-1]$ can have equal suffixes. Instead, we scan $OCC[2 \dots k_{i-1}]$: if $OCC[\ell] = j' \neq 0$, we check whether $b_{i-1,j}[\ell - 1] = 1$ for all $(j, j') \in E_{i-1}$, 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 $P[1 \dots \ell - 1]$. 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 $\mathcal{O}(N)$ over all segments.

Finally, to check whether some partial occurrence represented by $V_{i-1}[\ell] = j$ for $\ell \in [m - k_i + 2 \dots m]$ 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 $\mathcal{O}(N + mn + |E| \log m)$. ◀

4 Pattern matching in k - D and k - F texts

In this section, we investigate the complexity of the pattern-matching problem in cases where the text is either a k - D string or a k - F graph. In Section 4.1 we consider the cases where the pattern is solid; in Section 4.2, the pattern is a k - D string or a k - F graph as well.

4.1 Solid pattern

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 [18], for which an $\mathcal{O}(n \log^2 m)$ -time deterministic algorithm exists [19]¹. In this section, we leverage this result to prove sub-quadratic upper bounds for $\text{PVART}(\text{SOLID}, 1\text{-}F)$, $\text{PVART}(\text{SOLID}, k\text{-}D)$ and $\text{PVART}(\text{SOLID}, k\text{-}F)$ by reducing each of these problems to several instances of $\text{PVART}(\text{SOLID}, 1\text{-}D)$.

¹ 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 [18, 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 [19] and in their later work [20].

► **Theorem 1.** $\text{PVART}(\text{SOLID}, k\text{-}D)$ 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 there is a suffix/prefix overlap of length j between one of the strings in $T[i]$ and P , and $e_i[j] = 1$ if and only if there is a prefix/suffix overlap of length j between a string in $T[i]$ and 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 generalized suffix tree of all the strings in all segments of T .

Consider the case $m > k$. Let P_k denote the set of length- k substrings of P and let $h(s)$ denote the lexicographic 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. We then obtain from T a new 1- D text T' of the same length n by replacing each string $s \in T[i]$ such that $s \in P_k$ with $h(s)$, $\forall i \in [1, n]$, and discarding those that do not occur in P . T' has total size at most $\min\{B, nm\}$ and can be constructed in $\mathcal{O}(N)$ total time by searching each string of T in the suffix tree of P .

We then construct k instances of $\text{PVART}(\text{SOLID}, 1\text{-}D)$ using k patterns $P^{(0)}, \dots, P^{(k-1)}$ s.t. $P^{(\ell)}$ is obtained from P by replacing each non-overlapping fragment of length k by its ID, starting from position $\ell + 1$ and ignoring the possible remaining suffix of length $m - \ell - k \lfloor \frac{m-\ell}{k} \rfloor$. The length of $P^{(\ell)}$ is thus $\lfloor \frac{m-\ell}{k} \rfloor$. The input of the ℓ -th instance consists of $P^{(\ell)}$ and a text $T'^{(\ell)}$, obtained from T' by removing the IDs that do not appear in $P^{(\ell)}$. The k patterns can be constructed by sliding a window of length k over P and searching each of the corresponding fragments in the suffix tree of P to retrieve their ID in $\mathcal{O}(m)$ total time using the suffix links. $T'^{(\ell)}$ has size at most $\min\{B, n \frac{m}{k}\}$ and can be obtained in $\mathcal{O}(\min\{B, n \frac{m}{k}\})$ time from T' , thus constructing all of them 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 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 a string from each of the segments $T[i + 1], \dots, T[j - 1]$ occurs consecutively in P starting from position $\ell + 1$; a prefix of length ℓ of P must match a suffix of some string in $T[i]$; and a suffix of length r must match a prefix of some string in $T[j]$. This implies, by construction, an occurrence of $P^{(\ell)}$ in $T'^{(\ell)}$ starting at position $i + 1$; moreover, it must be $b_i[\ell] = e_j[r] = 1$. This gives us the following algorithm. For each possible offset $\ell = 0, 1, \dots, k - 1$:

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 [19] 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 $\ell = 0, 1, \dots, k - 1$ 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 [47]. 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 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 on an alphabet Σ of length m , let ℓ be an integer which divides m , and let us write $P = P_1 \dots P_d$ where $|P_i| = \ell$ for $i = 1 \dots d$. 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.

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 $(t, t') \in E_i$. We prove the following lemma:

▶ **Lemma 19.** *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 $\ell = 1 \dots d$, one has $\mathbf{spr}^k(P)[k + 1] = \$$. 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 since k divides m that means that $\mathbf{spr}^k(P) \in \mathcal{L}(\mathcal{D}(G)[i \dots j])$ for some $1 \leq i \leq j \leq n$. This implies that for such i, j and for each $\ell \leq d - 1$ one has $\mathbf{spr}^k(P)[(2k + 1)(\ell - 1) \dots (2k + 1)\ell] \in \mathcal{D}(G)[i + \ell - 1]$. But we notice that $\mathbf{spr}^k(P)[(2k + 1)(\ell - 1) \dots (2k + 1)\ell] = P_\ell \$ P_{\ell+1}$, 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 $1 \leq \ell \leq d - 1$, we deduce that $P \in \mathcal{L}(G[i \dots j])$. ◀

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

Proof (Sketch). Let $G = (T, E)$ be a k - F graph of length n and $P = P[1 \dots m]$ 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 $i < j$ since $k < \sqrt{m}$). This means, by definition, that there exist two (possibly empty) strings s, t and P_1, \dots, P_{j-i+1} 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 $P_1 \dots P_{j-i+1} = P$. We notice that one necessarily has $|s| + m + |t| = 0 \pmod k$, hence the string $P_1 \cdot \$ \cdot P_2 \cdot \mathbf{spr}^k(P_2 \dots P_{j-i}) \cdot P_{j-i} \cdot \$ \cdot P_{j-i+1}$ is uniquely determined by $|s|$, so we call it $\mathbf{spr}_{|s|}^k(P)$. Writing $\hat{P} = s \cdot P \cdot t$, we have that $\hat{P} \in \mathcal{L}(G[i \dots j])$, which, by Lemma 19, means precisely that $\mathbf{spr}^k(\hat{P})$ occurs in $\mathcal{D}(G)$. We can rewrite $\mathbf{spr}^k(\hat{P}) = s \cdot \mathbf{spr}_{|s|}^k(P) \cdot t$, and we deduce that P occurs in G with starting offset $|s|$ in some segment if and only if $\mathbf{spr}_{|s|}^k(P)$ occurs in $\mathcal{D}(G)$ (observing the occurrences of $\$$ in the constructed strings, every occurrence of $\mathbf{spr}_{|s|}^k(P)$ is always part of an occurrence of $s \cdot \mathbf{spr}_{|s|}^k(P) \cdot t$, for some pair s, t having suitable lengths). To find every occurrence of P in G , we can then search for the k patterns $\mathbf{spr}_0^k(P), \dots, \mathbf{spr}_{k-1}^k(P)$ in $\mathcal{D}(G)$.

Notice that $\mathcal{D}(G)$ has size $\mathcal{O}(k|E|)$. To search for $\mathbf{spr}_0^k(P), \dots, \mathbf{spr}_{k-1}^k(P)$ we can apply the algorithm of Theorem 1, however, while therein the algorithm reduces each instance $\text{PVART}(\text{SOLID}, k-D)$ to k instances of $\text{PVART}(\text{SOLID}, 1-D)$, one for each possible starting offset of the occurrences of P , here the presence of $\$$ already determines a unique offset for each $\mathbf{spr}_i^k(P)$, thus the time needed for each pattern is only $\mathcal{O}(n \log^2 m)$, implying a time complexity of $\mathcal{O}(k|E| + kn \log^2 m)$ for all patterns. Finally, to check whether each of the found occurrences of each pattern is actually an occurrence of the original P in G , we can proceed as described at the end of the proof of Theorem 1, at an extra cost of $\mathcal{O}(k|E|) = \mathcal{O}(\sqrt{m}|E|)$. We thus obtain a time complexity of $\mathcal{O}(\sqrt{m}(|E| + N \log^2 m))$ for the cases $k < \sqrt{m}$.

Now consider the case $k \geq \sqrt{m}$. Observe that, in this case, we have $n \leq \frac{N}{\sqrt{m}}$, because it always holds that $n \leq \frac{N}{k}$. By simply applying the algorithm of Theorem 4 in this special case we obtain a time complexity in $\mathcal{O}(N\sqrt{m} + |E| \log m)$.

Combining the two cases, the total time becomes $\mathcal{O}(\sqrt{m}(|E| + N \log^2 m))$. ◀

4.2 Variable pattern

In this section, we study the complexity of finding all the occurrences of non-solid patterns into a $k-D$ or $k-F$ text. We begin by formally by extending Definition 15 to the more general case where both P and T are either GD strings or F graphs.

► **Definition 20.** *A GD string or an F graph pattern P of length m has an occurrence ending at position j in a GD strings or an F graph text of length n if $\exists i \in [1, j]$ such that there exist $\ell \in [0, k_i - 1]$ and $r \in [0, k_j - 1]$ such that $\mathcal{L}(P) \cap \mathcal{L}(T^{\ell, r}[i..j]) \neq \emptyset$.*

Pattern 1- D and Text 1- D

The case where both P and T are 1- D is well-studied in the literature. In [44], the definition of *indeterminate* string coincides with our definition of 1- D string and an $\mathcal{O}(n \log m)$ -time algorithm for $\text{PVART}(1-D, 1-D)$ for the case of constant-size alphabets is given [44, Lemma 17]. A similar algorithm for constant-size alphabets has also been proposed in [63]. These results are complemented in [44, Theorem 22] with a quadratic conditional lower bound for the cases in which the alphabet size is not bounded by a constant.

This lower bound clearly applies also to both $\text{PVART}(1-D, k-D)$, $\text{PVART}(k-D, 1-D)$ when the alphabet size is not constant. In subsection 4.2.1, we prove that, in these cases, a quadratic lower bound holds even when the alphabet has only three letters.

In subsection 4.2.2, we consider the pattern-matching problem where at least one between pattern and text is in 1- F and the other is in 1- D . In every possible case, i.e. $\text{PVART}(1-D, 1-F)$, $\text{PVART}(1-F, 1-D)$, $\text{PVART}(1-F, 1-F)$, we prove a quadratic conditional lower bound for constant-size alphabets. This implies that whenever we are considering the $\text{PVART}(X, Y)$ problem with variable patterns, if at least one between pattern and text is a founder, the best that we can hope to achieve is a quadratic algorithm.

All the conditional lower bounds rely on a famous conjecture: the *Orthogonal Vectors Hypothesis*, which is implied by SETH [45, 67].

► **Definition 21 (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, $x \cdot y = \sum_{i=1}^d x[i] \cdot y[i] = 0$.*

Throughout 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 $x_2 \cdot y_2 = 0$.

► **Definition 22** (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 $\mathcal{O}(n^{2-\epsilon} \text{poly}(d))$.*

Here we summarize the general idea used in the following proofs. We will 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 contradicts OVH.

4.2.1 Pattern 1- D , text k - D

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 given by d segments $Q(y)[h]$, $1 \leq h \leq d$, defined as

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

The key property, which is clear by construction, is that a string x matches in $Q(y)$ only if it encodes a vector orthogonal to y .

► **Lemma 23.** *Let $x, y \in \{0, 1\}^d$, then the string $x[1]x[2] \cdots x[d]$ matches in $Q(y)$ if and only if $x \cdot y = 0$.*

► **Theorem 5.** *No algorithm can solve PVART(1- D, k - 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.*

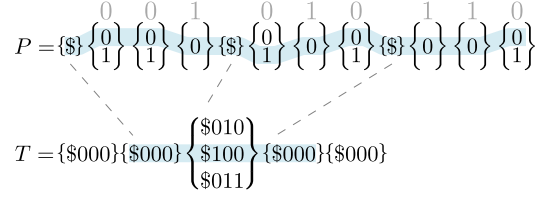
Proof. Let $X, Y \subseteq \{0, 1\}^d$, $|X| = |Y| = n$ be an instance of OV. We define a 1- D pattern P and a k - D text T such that P occurs in T if and only if $\exists x \in X, y \in Y, x \cdot y = 0$. We start by constructing pattern gadgets $Q(y_i)$, for each $y_i \in Y$. We then concatenate such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

$$T = \underbrace{\{\$0 \cdots 0\} \cdots \{\$0 \cdots 0\}}_{n-1 \text{ times}} \underbrace{\left\{ \begin{array}{c} \$x_1[1] \cdots x_1[d] \\ \vdots \\ \$x_n[1] \cdots x_n[d] \end{array} \right\}}_W \underbrace{\{\$0 \cdots 0\} \cdots \{\$0 \cdots 0\}}_{n-1 \text{ times}}^Z$$

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 23). Since P has n gadgets of length d , any match of P in T must span a string in W (see Figure 3). 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 between 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 k - 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 $\mathcal{O}((nd)(nd)^{1-\epsilon}) = \mathcal{O}(n^{2-\epsilon} \text{poly}(d))$ time, which contradicts OVH. ◀



■ **Figure 3** 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 $\$$.

► **Theorem 6.** *No algorithm can solve $\text{PVART}(k-D, 1-D)$ 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. 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 $k = \omega(\log n)$; it remains open whether there exist subquadratic algorithms when $k = O(\log n)$.

4.2.2 Pattern $1-F$, text $1-D$

In the following proofs we will use a three-level strategy first introduced in [33, 31].

We start by defining the two underlying $1-D$ strings P and T (one for the pattern and one for the text) common to 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 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 *upper* and *bottom* level. We build P as the concatenations of n $1-D$ string gadgets, one for each vector of X : given $x_i \in X$, we define

$$p(x_i) = \left\{ \begin{array}{c} u \\ x_i[1] \\ b \end{array} \right\} \cdots \left\{ \begin{array}{c} u \\ x_i[d] \\ b \end{array} \right\}.$$

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 4 for an example).

To build text T , we consider 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 $U_{\text{left}} = \{u\}^{d-1} \left\{ \begin{array}{c} \$ \\ u \end{array} \right\}$.

In a symmetric way, T_{right} consists of $n - 1$ gadgets $U_{\text{right}} = \left\{ \begin{array}{c} b \\ \$ \end{array} \right\} \{b\}^{d-1}$ followed by $\{\$\}$.

The $1-D$ string T^\perp is built using a slight variation of the gadgets $Q(y_j)$ (see proof of Theorem 5) extended with a three-level structure. Given $y_j \in Y$, we define $T_j^\perp[1] = \{u\} \cup Q(y_j)[1] \cup \{b, \$\}$, $T_j^\perp[h] = \{u\} \cup Q(y_j)[h] \cup \{b\}$ for $2 \leq h \leq d - 1$ 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 5 for an example):

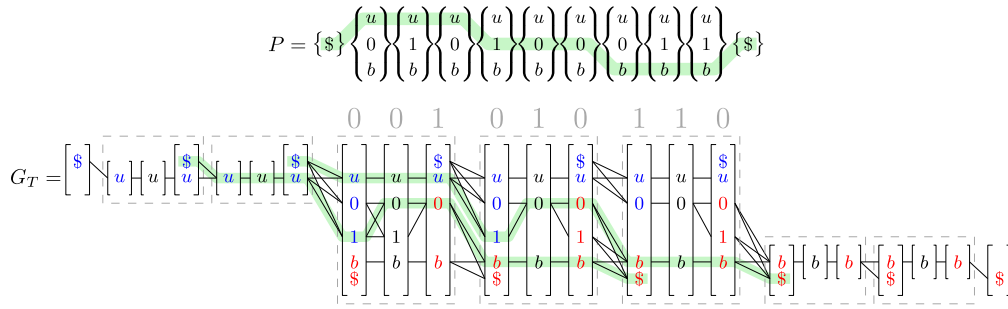


Figure 4 Example of a 1-D pattern P and 1-F text G_T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem 7. Highlighted paths show some occurrences of P in T , given by the string $\$uu100bbb\$ \in \mathcal{L}(P)$ which belongs also to $\mathcal{L}(T[4 \dots 14])$ and $\mathcal{L}(T[7 \dots 17])$. This corresponds to the fact that $x_2 = 100$ is orthogonal to $y_1 = 100$ and $y_2 = 010$.

$$T = \{\$\} U_{left}^{n-1} T_1^\perp \dots T_n^\perp U_{right}^{n-1} \{\$\} \quad (2)$$

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

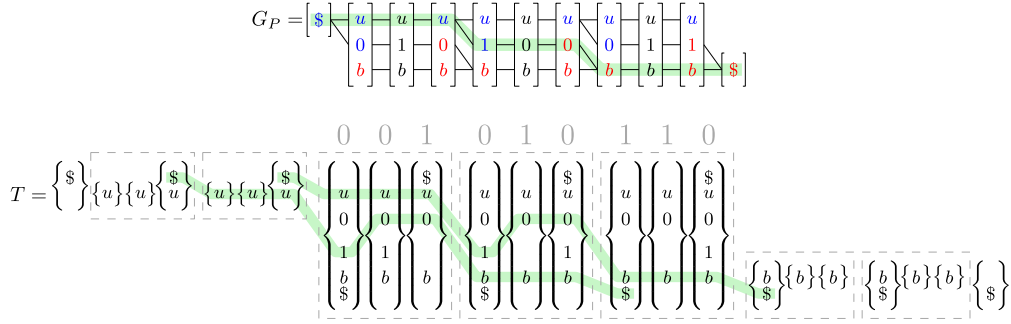
► Theorem 7. *No algorithm can solve $\text{PVART}(1\text{-D}, 1\text{-F})$ on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, unless OVH is false.*

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 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 incoming edge 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 outgoing edge 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 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 23 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 4 for an example. ◀

► Theorem 8. *No algorithm can solve $\text{PVART}(1\text{-F}, 1\text{-D})$ 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.*



■ **Figure 5** 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 8, with occurrences of G_P in T highlighted.

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 7. 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 $x, y \in \{0, 1\}$. Instead, 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 $\mathcal{L}(G_P) = \{\$u^{(i-1)d} \cdot x_i \cdot b^{(n-i)d}\$: i = 1, \dots, n\}$. Remark that each x_i is over the alphabet $\{0, 1\}$. Thus, because of the position of the symbol $\$$, Lemma 23 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 5 for an example of this reduction. ◀

Finally, the following result directly follows from a reduction that uses G_T as in the proof of Theorem 7 and G_P as in the proof of Theorem 8.

► **Theorem 9.** No algorithm can solve $\text{PVART}(1-F, 1-F)$ 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.

5 Open Problems

After this work, for almost all combinations of variable strings X and Y , problem $\text{PVART}(X, Y)$ received either a truly subquadratic upper bound or a quadratic lower bound conditioned on SETH . Notably, two cases are left open: when $X = \text{SOLID}$ and $Y = \text{GD}$, and when $X = \text{SOLID}$ and $Y = F$. In these cases, our algorithms, although subquadratic in some cases, are quadratic in the worst case and no lower bound is known. These problems correspond to the yellow cells in the first row of Table 1. Notice that the other yellow cell, that is $X = 1-D$ and $Y = 1-D$, is a completely solved case for which all bounds are known.

References

- 1 Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.
- 2 Jarno N. Alanko, Elena Biagi, Simon J. Puglisi, and Jaakko Vuhtoniemi. Subset wavelet trees. In *21st International Symposium on Experimental Algorithms (SEA)*, volume 265 of *LIPICs*, pages 4:1–4:14, 2023. doi:10.4230/LIPICs.SEA.2023.4.

- 3 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Degenerate string comparison and applications. In *18th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 113 of *LIPICs*, pages 21:1–21:14, 2018. doi:10.4230/LIPICs.WABI.2018.21.
- 4 Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Comparing degenerate strings. *Fundam. Informaticae*, 175(1-4):41–58, 2020. doi:10.3233/FI-2020-1947.
- 5 Amihud Amir and Michael Itzhaki. 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. doi:10.4230/LIPICs.CPM.2024.2.
- 6 Pavlos Antoniou, Maxime Crochemore, Costas S. Iliopoulos, Inuka Jayasekera, and Gad M. Landau. Conservative string covering of indeterminate strings. In *Proceedings of the Prague Stringology Conference*, pages 108–115, 2008. URL: <http://www.stringology.org/event/2008/p10.html>.
- 7 Kotaro Aoyama, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. 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. doi:10.4230/LIPICs.CPM.2018.9.
- 8 Jasmijn A. Baaijens, Paola Bonizzoni, Christina Boucher, Gianluca Della Vedova, Yuri Pirola, Raffaella Rizzi, and Jouni Sirén. Computational graph pangenomics: a tutorial on data structures and their applications. *Nat. Comput.*, 21(1):81–108, 2022. doi:10.1007/s11047-022-09882-6.
- 9 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.
- 10 Giulia Bernardini, Estéban Gabory, Solon P. Pissis, Leen Stougie, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string matching with 1 error. In *15th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 13568 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2022. doi:10.1007/978-3-031-20624-5_2.
- 11 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPICs*, pages 21:1–21:15, 2019. doi:10.4230/LIPICs.ICALP.2019.21.
- 12 Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput.*, 51(3):549–576, 2022. doi:10.1137/20M1368033.
- 13 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. 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*, pages 74–90. Springer, 2017. doi:10.1007/978-3-319-67428-5_7.
- 14 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.*, 812:109–122, 2020. doi:10.1016/J.TCS.2019.08.012.
- 15 Philip Bille, Inge Li Gørtz, and Tord Stordalen. Rank and select on degenerate strings. In *2024 Data Compression Conference (DCC)*, pages 283–292, 2024. doi:10.1109/DCC58796.2024.00036.
- 16 Thomas Büchler, Jannik Olbrich, and Enno Ohlebusch. Efficient short read mapping to a pangenome that is represented by a graph of ED strings. *Bioinformatics*, 39(5):btad320, 2023. doi:10.1093/bioinformatics/btad320.
- 17 Aleksander Cislak, Szymon Grabowski, and Jan Holub. Sopang: online text searching over a pan-genome. *Bioinform.*, 34(24):4290–4292, 2018. doi:10.1093/BIOINFORMATICS/BTY506.

- 18 Richard Cole and Ramesh Hariharan. 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)*, pages 66–75. ACM, 1997. doi:10.1145/258533.258553.
- 19 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.
- 20 Richard Cole and Ramesh Hariharan. Tree pattern matching to subset matching in linear time. *SIAM J. Comput.*, 32(4):1056–1066, 2003. doi:10.1137/S0097539700382704.
- 21 The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings Bioinformatics*, 19(1):118–135, 2018.
- 22 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Covering problems for partial words and for indeterminate strings. *Theor. Comput. Sci.*, 698:25–39, 2017. doi:10.1016/J.TCS.2017.05.026.
- 23 Petr Danecek, Adam Auton, Gonçalo R. Abecasis, Cornelis A. Albers, Eric Banks, Mark A. DePristo, Robert E. Handsaker, Gerton Lunter, Gabor T. Marth, Stephen T. Sherry, Gilean McVean, and Richard Durbin. The variant call format and vcftools. *Bioinform.*, 27(15):2156–2158, 2011. doi:10.1093/BIOINFORMATICS/BTR330.
- 24 Jacqueline W. Daykin, Richard Groult, Yannick Guesnet, Thierry Lecroq, Arnaud Lefebvre, Martine Léonard, Laurent Mouchard, Élise Prieur, and Bruce W. Watson. Efficient pattern matching in degenerate strings with the burrows-wheeler transform. *Inf. Process. Lett.*, 147:82–87, 2019. doi:10.1016/J.IPL.2019.03.003.
- 25 Jacqueline W. Daykin and Bruce W. Watson. Indeterminate string factorizations and degenerate text transformations. *Math. Comput. Sci.*, 11(2):209–218, 2017. doi:10.1007/S11786-016-0285-X.
- 26 Daniel Dorey-Robinson, Giuseppe Maccari, and John A. Hammond. Igmatt: immunoglobulin sequence multi-species annotation tool for any species including those with incomplete antibody annotation or unusual characteristics. *BMC Bioinform.*, 24(1):491, 2023. doi:10.1186/S12859-023-05624-2.
- 27 E.Garrison, J.Sirén, A.M.Novak, G.Hickey, J.M.Eizenga, E.T.Dawson, W.Jones, S.Garg, C.Markello, M.F.Lin MF, B.Paten B, and R.Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, 2018. doi:10.1038/nbt.4227.
- 28 Jordan M. Eizenga, Adam M. Novak, Emily Kobayashi, Flavia Villani, Cecilia Cisar, Simon Heumos, Glenn Hickey, Vincenza Colonna, Benedict Paten, and Erik Garrison. Efficient dynamic variation graphs. *Bioinform.*, 36(21):5139–5144, 2021. doi:10.1093/bioinformatics/btaa640.
- 29 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. 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. doi:10.1007/978-3-030-67731-2_44.
- 30 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. *Theor. Comput. Sci.*, 975:114128, 2023. doi:10.1016/J.TCS.2023.114128.
- 31 Massimo Equi, Veli Mäkinen, Alexandru I. Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Trans. Algorithms*, 19(3):21:1–21:25, 2023. doi:10.1145/3588334.
- 32 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing elastic founder graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPICs*, pages 20:1–20:18, 2021. doi:10.4230/LIPICs.ISAAC.2021.20.

- 33 Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing founder graphs. *Algorithmica*, 85(6):1586–1623, 2023. doi:10.1007/S00453-022-01007-W.
- 34 Liao et al. A draft human pangenome reference. *Nature*, 617(7960):312–324, 2023.
- 35 Estéban Gabory, Njagi Moses Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Comparing elastic-degenerate strings: Algorithms, lower bounds, and applications. In *34th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 259 of *LIPICs*, pages 11:1–11:20, 2023. doi:10.4230/LIPICs.CPM.2023.11.
- 36 Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. On indeterminate strings matching. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 161 of *LIPICs*, pages 14:1–14:14, 2020. doi:10.4230/LIPICs.CPM.2020.14.
- 37 Daniel Gibney. An efficient elastic-degenerate text index? not likely. In Christina Boucher and Sharma V. Thankachan, editors, *27th International Symposium on String Processing and Information Retrieval*, volume 12303 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2020. doi:10.1007/978-3-030-59212-7_6.
- 38 Daniel Gibney, Gary Hoppenworth, and Sharma V. Thankachan. Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In *4th SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 232–242, 2021. doi:10.1137/1.9781611976496.26.
- 39 Roberto Grossi, Costas S. Iliopoulos, Chang Liu, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, Giovanna Rosone, Fatima Vayani, and Luca Versari. On-line pattern matching on similar texts. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of *LIPICs*, pages 9:1–9:14, 2017. doi:10.4230/LIPICs.CPM.2017.9.
- 40 Ming Gu, Martin Farach, and Richard Beigel. An efficient algorithm for dynamic text indexing. In *Proceedings of the 5th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 697–704, 1994. URL: <https://dl.acm.org/doi/pdf/10.5555/314464.314675>.
- 41 Jan Holub, William F. Smyth, and Shu Wang. Fast pattern-matching on indeterminate strings. *J. Discrete Algorithms*, 6(1):37–50, 2008. doi:10.1016/J.JDA.2006.10.003.
- 42 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Information and Computation*, 279:104616, 2021. doi:10.1016/j.ic.2020.104616.
- 43 Costas S. Iliopoulos, Laurent Mouchard, and Mohammad Sohel Rahman. A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. *Math. Comput. Sci.*, 1(4):557–569, 2008. doi:10.1007/S11786-007-0029-Z.
- 44 Costas S. Iliopoulos and Jakub Radoszewski. 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*, pages 8:1–8:12, 2016. doi:10.4230/LIPICs.CPM.2016.8.
- 45 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.
- 46 IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970. doi:10.1016/0022-2836(71)90319-6.
- 47 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.
- 48 Felipe A. Louza, Neerja Mhaskar, and W. F. Smyth. A new approach to regular & indeterminate strings. *Theor. Comput. Sci.*, 854:105–115, 2021. doi:10.1016/J.TCS.2020.12.007.
- 49 Veli Mäkinen, Bastien Cazaux, Massimo Equi, Tuukka Norri, and Alexandru I. Tomescu. Linear time construction of indexable founder block graphs. In *20th International Conference on Algorithms in Bioinformatics (WABI)*, volume 172 of *LIPICs*, pages 7:1–7:18, 2020. doi:10.4230/LIPICs.WABI.2020.7.

- 50 Njagi Moses Mwaniki, Erik Garrison, and Nadia Pisanti. Fast exact string to D-texts alignments. In *16th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC)*, pages 70–79. SCITEPRESS, 2023. doi:10.5220/0011666900003414.
- 51 Njagi Moses Mwaniki and Nadia Pisanti. 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. doi:10.1007/978-3-031-23198-8_19.
- 52 Benedict Paten, Adam M. Novak, Jordan M. Eizenga, and Erik Garrison. Genome graphs and the evolution of genome inference. *Genome Res*, 27(5):665–676, 2017.
- 53 Nadia Pisanti, Henry Soldano, and Mathilde Carpentier. 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*, pages 229–240. Springer, 2005. doi:10.1007/11496656_20.
- 54 Nadia Pisanti, Henry Soldano, Mathilde Carpentier, and Joël Pothier. A relational extension of the notion of motifs: Application to the common 3d protein substructures searching problem. *Journal of Computational Biology*, 16(12):1635–1660, 2009. doi:10.1089/CMB.2008.0019.
- 55 Solon P. Pissis and Ahmad Retha. 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*, pages 16:1–16:14, 2018. doi:10.4230/LIPICs.SEA.2018.16.
- 56 Petr Procházka, Ondrej Cvacho, Lubos Krcál, and Jan Holub. Backward pattern matching on elastic-degenerate strings. *SN Comput. Sci.*, 4(5):442, 2023. doi:10.1007/S42979-023-01760-X.
- 57 Goran Rakocevic, Vladimir Semenyuk, Wan-Ping Lee, James Spencer, John Browning, Ivan J. Johnson, Vladan Arsenijevic, Jelena Nadj, Kaushik Ghose, Maria C. Suci, Sun-Gou Ji, Gülfem Demir, Lizao Li, Berke Ç. Toptaş, Alexey Dolgoborodov, Björn Pollex, Iosif Spulber, Irina Glotova, Péter Kómar, Andrew L. Stachyra, Yilong Li, Milos Popovic, Morten Källberg, Amit Jain, and Deniz Kural. Fast and accurate genomic analyses using genome graphs. *Nature Genetics*, 51:354–362, 2019.
- 58 Nicola Rizzo, Massimo Equi, Tuukka Norri, and Veli Mäkinen. Elastic founder graphs improved and enhanced. *Theor. Comput. Sci.*, 982:114269, 2024. doi:10.1016/J.TCS.2023.114269.
- 59 Nicola Rizzo and Veli Mäkinen. Indexable elastic founder graphs of minimum height. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 223 of *LIPICs*, pages 19:1–19:19, 2022. doi:10.4230/LIPICs.CPM.2022.19.
- 60 Nicola Rizzo and Veli Mäkinen. Linear time construction of indexable elastic founder graphs. In *33rd International Workshop on Combinatorial Algorithms (IWOCA)*, volume 13270 of *Lecture Notes in Computer Science*, pages 480–493. Springer, 2022. doi:10.1007/978-3-031-06678-8_35.
- 61 Marie-France Sagot, Alain Viari, and Henry Soldano. 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. doi:10.1007/3-540-60044-2_55.
- 62 Marie-France Sagot, Alain Viari, and Henry Soldano. Multiple sequence comparison - A peptide matching approach. *Theor. Comput. Sci.*, 180(1-2):115–137, 1997. doi:10.1016/S0304-3975(96)00137-5.
- 63 Ariel Shiftan and Ely Porat. Set intersection and sequence matching with mismatch counting. *Theor. Comput. Sci.*, 638:3–10, 2016. doi:10.1016/J.TCS.2016.01.003.
- 64 Jouni Sirén, Erik Garrison, Adam M. Novak, Benedict Paten, and Richard Durbin. Haplotype-aware graph indexes. In *18th International Conference on Algorithms in Bioinformatics (WABI)*, volume 113 of *LIPICs*, pages 4:1–4:13, 2018. doi:10.4230/LIPICs.WABI.2018.4.

- 65 Henry Soldano, Alain Viari, and Marc Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognit. Lett.*, 16(3):233–246, 1995. doi:10.1016/0167-8655(94)00095-K.
- 66 Chris Thachuk. Indexing hypertext. *J. Discrete Algorithms*, 18:113–122, 2013. doi:10.1016/J.JDA.2012.10.001.
- 67 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.