ELASTIC-DEGENERATE STRING MATCHING VIA FAST MATRIX MULTIPLICATION*

GIULIA BERNARDINI†, PAWEŁ GAWRYCHOWSKI‡, NADIA PISANTI§, SOLON P. PISSIS¶, AND GIOVANNA ROSONE‖

Abstract. An elastic-degenerate (ED) string is a sequence of n sets of strings of total length N, which was recently proposed to model a set of similar sequences. The ED string matching (EDSM) problem is to find all occurrences of a pattern of length m in an ED text. The EDSM problem has recently received some attention in the combinatorial pattern matching community, and an $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$-time algorithm is known [Aoyama et al., CPM 2018]. The standard assumption in the prior work on this question is that N is substantially larger than both n and m, and thus we would like to have a linear dependency on the former. Under this assumption, the natural open problem is whether we can decrease the 1.5 exponent in the time complexity, similarly as in the related (but, to the best of our knowledge, not equivalent) *word break* problem [Backurs and Indyk, FOCS 2016].

Our starting point is a conditional lower bound for the EDSM problem. We use the popular combinatorial Boolean Matrix Multiplication (BMM) conjecture stating that there is no truly subcubic *combinatorial* algorithm for BMM [Abboud and Williams, FOCS 2014]. By designing an appropriate reduction we show that a combinatorial algorithm solving the EDSM problem in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, refutes this conjecture. Our reduction should be understood as an indication that decreasing the exponent requires fast matrix multiplication.

String periodicity and fast Fourier transform are two standard tools in string algorithms. Our main technical contribution is that we successfully combine these tools with fast matrix multiplication to design a non-combinatorial $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$-time algorithm for EDSM, where ω denotes the matrix multiplication exponent and the $\tilde{\mathcal{O}}(\cdot)$ notation suppresses polylog factors. To the best of our knowledge, we are the first to combine these tools. In particular, using the fact that $\omega < 2.373$ [Alman and Williams, SODA 2021; Le Gall, ISSAC 2014; Williams, STOC 2012], we obtain an $\mathcal{O}(nm^{1.373} + N)$-time algorithm for EDSM. An important building block in our solution, that might find applications in other problems, is a method of selecting a small set of length-ℓ substrings of the pattern, called anchors, so that any occurrence of a string from an ED text set contains at least one but not too many (on average) such anchors inside.

Key words. string algorithms, pattern matching, elastic-degenerate string, matrix multiplication, fast Fourier transform

AMS subject classifications. 68W01, 68W32, 68Q25, 68Q17

*Submitted to the editors DATE. A preliminary version appeared in ICALP 2019 [14].

Funding: GB is supported by the Netherlands Organisation for Scientific Research (NWO) under project OCENW.GROOT.2019.015 "Optimization for and with Machine Learning (OPTIMAL)". PG is partially supported by the Bekker programme of the Polish National Agency for Academic Exchange (PPN/BEK/2020/1/00444). NP and GR are supported by the University of Pisa under the "PRA Progetti di Ricerca di Ateneo" (Institutional Research Grants) - Project no. PRA_20202021_26 "Metodi Informatici Integrati per la Biomedica" and partially supported by MIUR-SIR project CMACBioSeq "Combinatorial methods for analysis and compression of biological sequences" grant no. RBSI146R5L. NP, SPP and GR are partially supported by the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 956229. SPP is partially supported by the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement no. 872539.

†CWI, Amsterdam, The Netherlands (giulia.bernardini@cwi.nl).

‡University of Wrocław, Wrocław, Poland (gawry@cs.uni.wroc.pl).

§University of Pisa, Pisa, Italy (nadia.pisanti@unipi.it).

¶CWI, Amsterdam, The Netherlands and Vrije Universiteit, Amsterdam, The Netherlands (solon.pissis@cwi.nl).

‖University of Pisa, Pisa, Italy (giovanna.rosone@unipi.it).

1. Introduction. Boolean matrix multiplication (BMM) is one of the most fundamental computational problems. Apart from its theoretical interest, it has a wide range of applications [34, 36, 44, 55, 64]. BMM is also the core combinatorial part of integer matrix multiplication. In both problems, we are given two $\mathcal{N} \times \mathcal{N}$ matrices and we are to compute $\mathcal{N}^2$ values. Integer matrix multiplication can be performed in *truly subcubic* time, i.e., in $\mathcal{O}(\mathcal{N}^{3-\epsilon})$ operations over the field, for some $\epsilon > 0$. The fastest known algorithms for this problem run in $\mathcal{O}(\mathcal{N}^{2.373})$ time [4, 51, 66]. These algorithms are known as algebraic: they rely on the ring structure of matrices over the field.

There also exists a different family of algorithms for the BMM problem known as combinatorial. Their focus is on unveiling the combinatorial structure in the Boolean matrices to reduce redundant computations. A series of results [9, 11, 20] culminating in an $\hat{\mathcal{O}}(\mathcal{N}^3/\log^4 \mathcal{N})$-time algorithm [70, 71] (the $\hat{\mathcal{O}}(\cdot)$ notation suppresses polyloglog factors) has led to the popular combinatorial BMM conjecture stating that there is no combinatorial algorithm for BMM working in time $\mathcal{O}(\mathcal{N}^{3-\epsilon})$, for any $\epsilon > 0$ [2]. There has been ample work on applying this conjecture to obtain BMM hardness results: see, e.g., [2, 22, 40, 49, 50, 52, 60].

String matching is another fundamental problem, asking to find all fragments of a string text of length n that match a string pattern of length m. This problem has several linear-time solutions [28]. In many real-world applications, it is often the case that letters at some positions are either unknown or uncertain. A way of representing these positions is with a subset of the alphabet Σ. Such a representation is called *degenerate string*. A special case of a degenerate string is when at such unknown or uncertain positions the only subset of the alphabet allowed is the whole alphabet. These special degenerate strings are more commonly known as strings with wildcards. The first efficient algorithm for a text and a pattern, where both may contain wildcards, was published by Fischer and Paterson in 1974 [35]. It has undergone several improvements since then [25, 26, 43, 46]. The first efficient algorithm for a standard text and a degenerate pattern, which may contain any non-empty subset of the alphabet, was published by Abrahamson in 1987 [3], followed by several practically efficient algorithms [41, 56, 69].

Degenerate letters are used in the IUPAC notation [45] to represent a position in a DNA sequence that can have multiple possible alternatives. These are used to encode the consensus of a population of sequences [5, 6, 37, 57, 63] in a multiple sequence alignment (MSA). In the presence of insertions or deletions in the MSA, we may need to consider alternative representations. Consider the following MSA of three closely-related sequences (on the left):

```
GCAACGGGTA--TT
GCAACGGGTATATT
GCACCTGG----TT
```

$$\tilde{T} = \left\{ \texttt{GCA} \right\} \cdot \left\{ \begin{matrix} \texttt{A} \\ \texttt{C} \end{matrix} \right\} \cdot \left\{ \texttt{C} \right\} \cdot \left\{ \begin{matrix} \texttt{G} \\ \texttt{T} \end{matrix} \right\} \cdot \left\{ \texttt{GG} \right\} \cdot \left\{ \begin{matrix} \texttt{TA} \\ \texttt{TATA} \\ \varepsilon \end{matrix} \right\} \cdot \left\{ \texttt{TT} \right\}$$

These sequences can be compacted into a single sequence $\tilde{T}$ of sets of strings (on the right) containing some deterministic and some non-deterministic segments. A non-deterministic segment is a finite set of deterministic strings and may contain the empty string ε corresponding to a deletion. The total number of segments is the *length* of $\tilde{T}$ and the total number of letters is the *size* of $\tilde{T}$. We denote the length by $n = |\tilde{T}|$ and the size by $N = ||\tilde{T}||$.

This representation has been defined in [42] by Iliopoulos et al. as an *elastic-degenerate* (ED) string. Being a sequence of subsets of Σ^*, it can be seen as a generalization of a degenerate string. The natural problem that arises is finding all matches of a deterministic pattern P in an ED text $\tilde{T}$. This is the *elastic-degenerate string matching* (EDSM) problem. Since its introduction in 2017 [42], it has attracted some attention in the combinatorial pattern matching community [58], and a series of results have been published. The simple algorithm by Iliopoulos et al. [42] for EDSM was first improved by Grossi et al. in the same year, who showed that, for a pattern of length m, the EDSM problem can be solved *on-line* in $\mathcal{O}(nm^2 + N)$ time [39]; on-line means that it reads the text segment-by-segment and reports an occurrence as soon as this is detected. This result was improved by Aoyama et al. [8] who presented an $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$-time algorithm. An important feature of these bounds is their *linear dependency* on N. A different branch of on-line algorithms waiving the linear-dependency restriction exists [23, 24, 39, 59]. Moreover, the EDSM problem has been considered under Hamming and edit distance [16]. Recent results on founder block graphs [53] can also be casted on elastic-degenerate strings.

A question with a somewhat similar flavor is the *word break* problem. We are given a dictionary $\mathcal{D}$, $m = ||\mathcal{D}||$, and a string S, $n = |S|$, and the question is whether we can split S into fragments that appear in $\mathcal{D}$ (the same element of $\mathcal{D}$ can be used multiple times). Backurs and Indyk [10] designed an $\tilde{\mathcal{O}}(nm^{1/2-1/18} + m)$-time algorithm for this problem[1]. Bringmann et al. [18] improved this to $\tilde{\mathcal{O}}(nm^{1/3} + m)$ and showed that this is optimal for combinatorial algorithms by a reduction from k-Clique. Their algorithm uses fast Fourier transform (FFT), and so it is not clear whether it should be considered combinatorial. While this problem seems similar to EDSM, there does not seem to be a direct reduction and so their lower bound does not immediately apply.

Our Results. It is known that BMM and triangle detection (TD) in graphs either both have truly subcubic combinatorial algorithms or none of them do [68]. Recall also that the currently fastest algorithm with linear dependency on N for the EDSM problem runs in $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$ time [8]. In this paper we prove the following two theorems.

THEOREM 1.1. *If the EDSM problem can be solved in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, with a combinatorial algorithm, then there exists a truly subcubic combinatorial algorithm for TD.*

Arguably, the notion of combinatorial algorithms is not clearly defined, and Theorem 1.1 should be understood as an indication that in order to achieve a better complexity one should use fast matrix multiplication. Indeed, there are examples where a lower bound conditioned on BMM was helpful in constructing efficient algorithms using fast matrix multiplication [1, 17, 21, 30, 54, 67, 72]. We successfully design such a non-combinatorial algorithm by combining three ingredients: a string periodicity argument, FFT, and fast matrix multiplication. While periodicity is the usual tool in combinatorial pattern matching [29, 47, 48] and using FFT is also not unusual (for example, it often shows up in approximate string matching [3, 7, 25, 38]), to the best of our knowledge, we are the first to combine these with fast matrix multiplication. Specifically, we show the following result for the EDSM problem, where ω denotes the matrix multiplication exponent.

THEOREM 1.2. *The EDSM problem can be solved on-line in $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$ time.*

In order to obtain a faster algorithm for the EDSM problem, we focus on the *active prefixes* (AP) problem that lies at the heart of all current solutions [8, 39]. In

[1] The $\tilde{\mathcal{O}}(\cdot)$ notation suppresses polylog factors.

the AP problem, we are given a string P of length m and a set of arbitrary prefixes $P[1\,..\,i]$ of P, called *active prefixes*, stored in a bit vector U so that $U[i] = 1$ if $P[1\,..\,i]$ is active. We are further given a set $\mathcal{S}$ of strings of total length N and we are asked to compute a bit vector V which stores the new set of active prefixes of P. A new active prefix of P is a concatenation of $P[1\,..\,i]$ (such that $U[i] = 1$) and some element of $\mathcal{S}$.

Using the algorithmic framework introduced in [39], EDSM is addressed by solving an instance of the AP problem per each segment i of the ED text corresponding to set $\mathcal{S}$ of the AP problem. Hence, an $\mathcal{O}(f(m)+N_i)$ solution for the AP problem (with N_i being the size of a single segment of the ED text) implies an $\mathcal{O}(nf(m)+N)$ solution of EDSM, as $f(m)$ is repeated n times and $N = \sum_{i=1}^{n} N_i$. The algorithm of [8] solves the AP problem in $\mathcal{O}(m^{1.5}\sqrt{\log m} + N_i)$ time leading to $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$ time for the EDSM problem. Our algorithm partitions the strings of each segment i of the ED text into three types according to a periodicity criterion, and then solves a restricted instance of the AP problem for each of the types. In particular, we solve the AP problem in $\tilde{\mathcal{O}}(m^{\omega-1} + N_i)$ time leading to $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$ time for the EDSM problem. Given this connection between the two problems and, in particular, between their size parameter N, in the rest of the paper we will denote with N also the parameter N_i of the AP problem.

An important building block in our solution that might find applications in other problems is a method of selecting a small set of length-ℓ substrings of the pattern, called *anchors*, so that any relevant occurrence of a string from an ED text set contains at least one but not too many such anchors inside. This is obtained by rephrasing the question in a graph-theoretical language and then generalizing the well-known fact that an instance of the hitting set problem with m sets over $[n]$, each of size at least k, has a solution of size $\mathcal{O}(n/k \cdot \log m)$. While the idea of carefully selecting some substrings of the same length is not new (for example Kociumaka et al. [48] used it to design a data structure for pattern matching queries on a string), our setting is different and hence so is the method of selecting these substrings.

In addition to the conditional lower bound for the EDSM problem (Theorem 1.1), we also exhibit a reduction from BMM to AP that leads to the following conditional lower bound for AP.

THEOREM 1.3. *If the AP problem can be solved in $\mathcal{O}(m^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, with a combinatorial algorithm, then there exists a truly subcubic combinatorial algorithm for the BMM problem.*

We remark that Theorem 1.3 is also implied by Theorem 1.1, as described at the end of Section 4, but we believe that a direct reduction from BMM to AP serves as a good starting point for the more complicated reduction from BMM to EDSM.

Roadmap. Section 2 provides the necessary definitions and notation as well as the algorithmic toolbox used throughout the paper. In Section 3 we prove our lower bound result for the AP problem (Theorem 1.3). The lower bound result for the EDSM problem is proved in Section 4 (Theorem 1.1). In Section 5 we present our algorithm for EDSM (Theorem 1.2); this is the most technically involved part of the paper.

2. Preliminaries.

Let $T = T[1]T[2]\ldots T[n]$ be a string of length $|T| = n$ over a finite ordered alphabet Σ of size $|\Sigma| = \sigma$. For two positions i and j on T, we denote by $T[i\,..\,j] = T[i]\ldots T[j]$ the substring of T that starts at position i and ends at position j (it is of length 0 if $j < i$). By ε we denote the empty string of length 0. A prefix of T is a substring of the form $T[1\,..\,j]$, and a suffix of T is a substring of the form $T[i\,..\,n]$.

T^r denotes the reverse of T, that is, $T[n]T[n-1]\dots T[1]$. We say that a string X is a power of a string Y if there exists an integer $k > 1$, such that X is expressed as k consecutive concatenations of Y, denoted by $X = Y^k$. A period of a string X is any integer $p \in [1, |X|]$ such that $X[i] = X[i+p]$ for every $i = 1, 2, \dots, |X| - p$, and *the period*, denoted by $\mathrm{per}(X)$, is the smallest such p. We call a string X *strongly periodic* if $\mathrm{per}(X) \leq |X|/4$.

LEMMA 2.1 ([33]). *If p and q are both periods of the same string X, and additionally $p + q \leq |X| + 1$, then $\gcd(p, q)$ is also a period of X.*

A *trie* is a tree in which every edge is labeled with a single letter, and every two edges outgoing from the same node have different labels. The label of a node u in such a tree T, denoted by $\mathcal{L}(u)$, is defined as the concatenation of the labels of all the edges on the path from the root of T to u. By replacing each path p consisting of nodes with exactly one child by an edge labeled by the concatenation of the labels of the edges of p we obtain a *compact trie*. The nodes of the trie that are removed after this transformation are called *implicit*, while the remaining ones are referred to as *explicit*. The suffix tree of a string S is the compact trie representing all suffixes of $S\$$, $\$ \notin \Sigma$, where instead of explicitly storing the label $S[i \mathinner{.\,.} j]$ of an edge we represent it by the pair (i, j).

A *heavy path decomposition* of a tree T is obtained by selecting, for every non-leaf node $u \in T$, its child v such that the subtree rooted at v is the largest. This decomposes the nodes of T into node-disjoint paths, with each such path p (called a heavy path) starting at some node, called the *head* of p, and ending at a leaf. An important property of such a decomposition is that the number of distinct heavy paths above any leaf (that is, intersecting the path from a leaf to the root) is only logarithmic in the size of T [62].

Let $\tilde{\Sigma}$ denote the set of all finite non-empty subsets of Σ^*. Previous works (cf. [8, 15, 39, 42, 59]) define $\tilde{\Sigma}$ as the set of all finite non-empty subsets of Σ^* excluding $\{\varepsilon\}$ but we waive here the latter restriction as it has no algorithmic implications. An *elastic-degenerate string* $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$, or ED string, over alphabet Σ, is a string over $\tilde{\Sigma}$, i.e., an ED string is an element of $\tilde{\Sigma}^*$, and hence each $\tilde{T}[i]$ is a set of strings.

Let $\tilde{T}$ denote an ED string of length n, i.e. $|\tilde{T}| = n$. We assume that for any $1 \leq i \leq n$, the set $\tilde{T}[i] \in \tilde{\Sigma}$ is implemented as an array and can be accessed by an index, i.e., $\tilde{T}[i] = \{\tilde{T}[i][k] \mid k = 1, \dots, |\tilde{T}[i]|\}$. For any $\tilde{\sigma} \in \tilde{\Sigma}$, $||\tilde{\sigma}||$ denotes the total length of all strings in $\tilde{\sigma}$, and for any ED string $\tilde{T}$, $||\tilde{T}||$ denotes the total length of all strings in all $\tilde{T}[i]$s. We will denote $N_i = \sum_{k=1}^{|\tilde{T}[i]|} |\tilde{T}[i][k]|$ the total length of all strings in $\tilde{T}[i]$ and $N = \sum_{i=1}^{n} ||\tilde{T}[i]||$ the *size* of $\tilde{T}$. An ED string $\tilde{T}$ can be thought of as a compact representation of the set of strings $\mathcal{A}(\tilde{T})$ which is the Cartesian product of all $\tilde{T}[i]$s; that is, $\mathcal{A}(\tilde{T}) = \tilde{T}[1] \times \dots \times \tilde{T}[n]$ where $A \times B = \{xy \mid x \in A, y \in B\}$ for any sets of strings A and B.

For any ED string $\tilde{X}$ and a pattern P, we say that P *matches* $\tilde{X}$ if:

1. $|\tilde{X}| = 1$ and P is a substring of some string in $\tilde{X}[1]$, or,
2. $|\tilde{X}| > 1$ and $P = P_1 \dots P_{|\tilde{X}|}$, where P_1 is a suffix of some string in $\tilde{X}[1]$, $P_{|\tilde{X}|}$ is a prefix of some string in $\tilde{X}[|\tilde{X}|]$, and $P_i \in \tilde{X}[i]$, for all $1 < i < |\tilde{X}|$.

We say that an occurrence of a string P ends at position j of an ED string $\tilde{T}$ if there exists $i \leq j$ such that P matches $\tilde{T}[i] \dots \tilde{T}[j]$. We will refer to string P as the *pattern* and to ED string $\tilde{T}$ as the *text*. We define the main problem considered in this paper.

> Elastic-Degenerate String Matching (EDSM)
> **INPUT:** A string P of length m and an ED string $\tilde{T}$ of length n and size $N \geq m$.
> **OUTPUT:** All positions in $\tilde{T}$ where at least one occurrence of P ends.

Example 1. *$P =$ `GTAT` ends at positions 2, 6, and 7 of the following text $\tilde{T}$.*

$$\tilde{T} = \left\{ \texttt{ATGTA} \right\} \cdot \left\{ \begin{array}{c} \texttt{A} \\ \texttt{T} \end{array} \right\} \cdot \left\{ \texttt{C} \right\} \cdot \left\{ \begin{array}{c} \texttt{G} \\ \texttt{T} \end{array} \right\} \cdot \left\{ \texttt{CG} \right\} \cdot \left\{ \begin{array}{c} \texttt{TA} \\ \texttt{TATA} \\ \varepsilon \end{array} \right\} \cdot \left\{ \begin{array}{c} \texttt{TATGC} \\ \texttt{TTTTA} \end{array} \right\}$$

Whenever $|\tilde{T}| = 1$, the problem reduces to Case 1 only (searching for P in all strings of $\tilde{T}[1]$), which can be done in $\mathcal{O}(N)$ time using any linear-time pattern-matching algorithm. In the general case of $|\tilde{T}| > 1$, at a high-level, previous on-line solutions to EDSM consist of the following steps: (i) For each $\tilde{T}[i]$, for each $S \in \tilde{T}[i]$ that is long enough, search for occurrences of the whole of P in S (this corresponds to Case 1 of the definition of a match of P given above). Then (Case 2 of the definition of a match of P, in which an occurrence of P spans over several sets of strings), (ii) find the prefixes of P that match any suffix of some $S \in \tilde{T}[i]$, (iii) try to extend at $\tilde{T}[i]$ every partial occurrence of P, which has started earlier in $\tilde{T}$, by solving an instance of AP, and (iv) if a full occurrence of P also ends at $\tilde{T}[i]$, then output position i; otherwise store the prefixes of P extended at $\tilde{T}[i]$, which will be further extended at $\tilde{T}[i+1]$.

Aoyama et al. [8] obtained an on-line $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$-time algorithm by identifying Step (iii) as the bottleneck in this approach, observing that all other steps can be implemented in $\mathcal{O}(n + M)$ time, and designing an improved solution for Step (iii). We formally define the task that needs to be solved in Step (iii) as the Active Prefixes problem:

> Active Prefixes (AP)
> **INPUT:** A string P of length m, a bit vector U of size m, a set $\mathcal{S}$ of strings of total length N.
> **OUTPUT:** A bit vector V of size m with $V[j] = 1$ if and only if there exists $S \in \mathcal{S}$ and $i \in [1, m], U[i] = 1$, such that $P[1\,.\,.\,i] \cdot S = P[1\,.\,.\,i + |S|]$ and $j = i + |S|$.

In particular, given an ED text $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$, one should consider an instance of the AP problem per each $\tilde{T}[i]$. Hence, an $\mathcal{O}(f(m) + N_i)$ solution for AP (N_i being the size of $\tilde{T}[i]$) implies an $\mathcal{O}(n \cdot f(m) + N)$ solution for EDSM, as $f(m)$ is repeated n times and $N = \sum_{i=1}^{n} N_i$. We provide an example of the AP problem.

Example 2. *Let $P =$ `ababbababab` of length $m = 11$, $U = 01000100000$, and $\mathcal{S} = \{\varepsilon, \texttt{ab}, \texttt{abb}, \texttt{ba}, \texttt{baba}\}$. We have that $V = 01011101010$.*

For our lower bound results we rely on BMM and the following closely related problem.

> Boolean Matrix Multiplication (BMM)
> **INPUT:** Two $\mathcal{N} \times \mathcal{N}$ Boolean matrices A and B.
> **OUTPUT:** $\mathcal{N} \times \mathcal{N}$ Boolean matrix C, where $C[i,j] = \bigvee_k (A[i,k] \wedge B[k,j])$.

> Triangle Detection (TD)
> **INPUT:** Three $\mathcal{N} \times \mathcal{N}$ Boolean matrices A, B and C.
> **OUTPUT:** Are there i, j, k such that $A[i,j] = B[j,k] = C[k,i] = 1$?

An algorithm is called *truly subcubic* if it runs in $\mathcal{O}(\mathcal{N}^{3-\epsilon})$ time, for some $\epsilon > 0$. TD and BMM either both have truly subcubic combinatorial algorithms, or none of them do [68].

3. AP Conditional Lower Bound. As a warm-up, in order to investigate the hardness of the EDSM problem, we first show that an $\mathcal{O}(m^{1.5-\epsilon} + N)$-time solution to the active prefixes problem, that constitutes the core of the solutions proposed in [8, 39], would imply a truly subcubic combinatorial algorithm for Boolean matrix multiplication (BMM). We recall that in the AP problem, we are given a string P of length m and a set of prefixes $P[1\,..\,i]$ of P, called *active prefixes*, stored in a bit vector U ($U[i] = 1$ if and only if $P[1\,..\,i]$ is active). We are further given a set $\mathcal{S}$ of strings of total length N and we are asked to compute a bit vector V storing the new set of active prefixes of P: a prefix of P that extends $P[1\,..\,i]$ (such that $U[i] = 1$) with some element of $\mathcal{S}$. Of course, we can solve BMM by working over integers and using one of the fast matrix multiplication algorithms; plugging in the best known bounds results in an $\mathcal{O}(\mathcal{N}^{2.373})$-time algorithm [4]. However, such an algorithm is not *combinatorial*, i.e., it uses *algebraic* methods. In comparison, the best known combinatorial algorithm for BMM works in $\hat{\mathcal{O}}(\mathcal{N}^3/\log^4 \mathcal{N})$ time [71]. This leads to the following popular conjecture.

Conjecture 1 ([2]). *There is no combinatorial algorithm for the BMM problem working in time $\mathcal{O}(\mathcal{N}^{3-\epsilon})$, for any $\epsilon > 0$.*

Aoyama et al. [8] showed that the AP problem can be solved in $\mathcal{O}(m^{1.5}\sqrt{\log m}+N)$ time for constant-sized alphabets. Together with some standard string-processing techniques applied similarly as in [39], this is then used to solve the EDSM problem by creating an instance of the AP problem for every set $\tilde{T}[i]$ of $\tilde{T}$, i.e., with $\mathcal{S} = \tilde{T}[i]$.

We argue that, unless Conjecture 1 is false, the AP problem cannot be solved in time $\mathcal{O}(m^{1.5-\epsilon} + N)$, for any $\epsilon > 0$, with a combinatorial algorithm (note that the algorithm of Aoyama et al. [8] uses FFT, and so it is not completely clear whether it should be considered to be combinatorial). We show this by a reduction from combinatorial BMM. Assume that, for the AP problem, we seek combinatorial algorithms with the running time $\mathcal{O}(m^{1.5-\epsilon}+N)$, i.e., with linear dependency on the total length of the strings. We need to show that such an algorithm implies that the BMM problem can be solved in $\mathcal{O}(\mathcal{N}^{3-\epsilon'})$ time, for some $\epsilon' > 0$, with a combinatorial algorithm, thus implying that Conjecture 1 is false.

Theorem 1.3. *If the AP problem can be solved in $\mathcal{O}(m^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, with a combinatorial algorithm, then there exists a truly subcubic combinatorial algorithm for the BMM problem.*

Proof. Recall that in the BMM problem the matrices are denoted by A and B. In order to compute $C = A \times B$, we need to find, for every $i, j = 1, \ldots, \mathcal{N}$, an index k such that $A[i, k] = 1$ and $B[k, j] = 1$. To this purpose, we split matrix A into blocks of size $\mathcal{N} \cdot L$ and B into blocks of size $L \cdot L$. This corresponds to considering values of j and k in intervals of size L, and clearly there are $\mathcal{N}/L$ such intervals. Matrix B is thus split into $(\mathcal{N}/L)^2$ blocks, giving rise to an equal number of instances of the AP problem, each one corresponding to an interval of j and an interval of k. We will now describe the instance corresponding to the (K, J)-th block, where $1 \leq K, J \leq \mathcal{N}/L$.

We build the string P of the AP problem, for any block, as a concatenation of $\mathcal{N}$ gadgets corresponding to $i = 1, \ldots, \mathcal{N}$, and we construct the bit vector $U^{(K,J)}$ of the AP problem as a concatenation of $\mathcal{N}$ bit vectors, one per gadget. Each gadget consists of the same string $\mathtt{a}^L\mathtt{b}\mathtt{a}^L$; we set to 1 the k'-th bit of the i-th gadget bit vector if $A[i,(K-1)L+k']=1$. The solution of the AP problem $V^{(K,J)}$ will allow us to recover the solution of BMM, as we will ensure that the bit corresponding to the j'-th $\mathtt{a}$ in the second half of the gadget is set to 1 if and only if, for some $k' \in [L]$, $A[i,(K-1)L+k']=1$ and $B[(K-1)L+k',(J-1)L+j']=1$. In order to enforce this, we will include the following strings in set $\mathcal{S}^{(K,J)}$:

$$\mathtt{a}^{L-k'}\mathtt{b}\mathtt{a}^{j'},\ \text{for every } k',j' \in [L] \text{ such that } B[(K-1)L+k',(J-1)L+j']=1.$$

This guarantees that after solving the AP problem we have the required property, and thus, after solving all the instances, we have obtained matrix $C=A\times B$. Indeed, consider values j, i.e., the index that runs on the columns of C, in intervals of size L. By construction and by definition of BMM, the i-th line of the J-th column interval of C is obtained by taking the disjunction of the second half of the i-th interval of each (K,J)-th bit vector for every $K=1,2,\ldots,\mathcal{N}/L$.

We have a total of $(\mathcal{N}/L)^2$ instances. In each of them, the total length of all strings is $\mathcal{O}(L^3)$, and the length of the input string P is $(2L+1)\mathcal{N}=\mathcal{O}(L\cdot\mathcal{N})$. Using our assumed algorithm for each instance, we obtain the following total time:

$$\mathcal{O}((\mathcal{N}/L)^2\cdot(L^3+(\mathcal{N}\cdot L)^{1.5-\epsilon}))=\mathcal{O}(\mathcal{N}^2\cdot L+\mathcal{N}^{3.5-\epsilon}/L^{0.5+\epsilon}).$$

If we set $L=\mathcal{N}^{(1.5-\epsilon)/(1.5+\epsilon)}$, then the total time becomes:

$$\begin{aligned}
&\mathcal{O}(\mathcal{N}^{2+(1.5-\epsilon)/(1.5+\epsilon)}+\mathcal{N}^{3.5-\epsilon-(0.5+\epsilon)(1.5-\epsilon)/(1.5+\epsilon)})\\
&=\mathcal{O}(\mathcal{N}^{2+(1.5-\epsilon)/(1.5+\epsilon)}+\mathcal{N}^{2+(1.5-\epsilon)-(1.5-\epsilon)(0.5+\epsilon)/(1.5+\epsilon)})\\
&=\mathcal{O}(\mathcal{N}^{2+(1.5-\epsilon)/(1.5+\epsilon)}+\mathcal{N}^{2+(1.5-\epsilon)(1.5+\epsilon-0.5-\epsilon)/(1.5+\epsilon)})\\
&=\mathcal{O}(\mathcal{N}^{2+(1.5-\epsilon)/(1.5+\epsilon)}).
\end{aligned}$$

Hence we obtain a combinatorial BMM algorithm with complexity $\mathcal{O}(\mathcal{N}^{3-\epsilon'})$, where $\epsilon'=1-(1.5-\epsilon)/(1.5+\epsilon)>0$. ◻

EXAMPLE 3. *Consider the following instance of the BMM problem with* $\mathcal{N}=6$ *and* $L=3$.

$$\overset{A}{\left[\begin{array}{ccc|ccc}
0&1&0&0&1&0\\
1&0&1&0&0&0\\
0&0&0&0&0&1\\
\hline
1&0&0&0&1&0\\
0&0&0&1&0&0\\
0&1&0&0&0&0
\end{array}\right]}
\times
\overset{B}{\left[\begin{array}{ccc|ccc}
0&0&0&0&0&1\\
1&0&0&0&0&0\\
0&0&1&0&1&0\\
\hline
0&1&0&0&0&0\\
0&0&0&1&0&0\\
1&0&0&0&1&0
\end{array}\right]}
=
\overset{C}{\left[\begin{array}{ccc|ccc}
\mathbf{1}&\mathbf{0}&\mathbf{0}&1&0&0\\
0&0&1&0&1&1\\
1&0&0&0&1&0\\
\hline
0&0&0&1&0&1\\
0&1&0&0&0&0\\
1&0&0&0&0&0
\end{array}\right]}$$

From matrices A and B, we now show how the resulting matrix C can be found by building and solving 4 instances of the AP problem constructed as follows. The pattern is

$$P=\mathtt{aaabaaa}\cdot\mathtt{aaabaaa}\cdot\mathtt{aaabaaa}\cdot\mathtt{aaabaaa}\cdot\mathtt{aaabaaa}\cdot\mathtt{aaabaaa}$$

where the six gadgets are separated by a '$\cdot$' to be highlighted. For the AP instances, the vectors $U^{(K,J)}$ *shown below are the input bit vectors, and the sets* $S^{(K,J)}$ *are the input set of strings. For each instance, the bit vector* $V^{(K,J)}$ *shown below is the output of the AP problem.*

i	1	2	3	4	5	6
$U^{(1,1)}$:	[0100000	1010000	0000000	1000000	0000000	0100000]
$S^{(1,1)}$:	{aba,baaa}					
$V^{(1,1)}$:	[0000**100**	0000001	0000000	0000000	0000000	0000100]
$U^{(1,2)}$:	[0100000	1010000	0000000	1000000	0000000	0100000]
$S^{(1,2)}$:	{aabaaa,baa}					
$V^{(1,2)}$:	[0000000	0000011	0000000	0000001	0000000	0000000]
$U^{(2,1)}$:	[0100000	0000000	0010000	0100000	1000000	0000000]
$S^{(2,1)}$:	{aabaa,ba}					
$V^{(2,1)}$:	[0000**000**	0000000	0000100	0000000	0000010	0000000]
$U^{(2,2)}$:	[0100000	0000000	0010000	0100000	1000000	0000000]
$S^{(2,2)}$:	{aba,baa}					
$V^{(2,2)}$:	[0000100	0000000	0000010	0000100	0000000	0000000]

*As an example on how to obtain matrix C, consider the bold part of C above (*i.e.*, the first line of block $(1,1)$ of C). This is obtained by taking the disjunction of the bold parts of $V^{(1,1)}$ and $V^{(2,1)}$.*

4. EDSM Conditional Lower Bound. Since the lower bound for the AP problem does not imply *per se* a lower bound for the whole EDSM problem, in this section we show a conditional lower bound for the EDSM problem. Specifically, we perform a reduction from Triangle Detection to show that, if the EDSM problem could be solved in $\mathcal{O}(nm^{1.5-\epsilon}+N)$ time, this would imply the existence of a truly subcubic algorithm for TD. We show that TD can be reduced to the decision version of the EDSM problem: the goal is to detect whether there exists at least one occurrence of P in $\tilde{T}$. To this aim, given three matrices A, B, C, we first decompose matrix B into blocks of size $\mathcal{N}/s \times \mathcal{N}/s$, where s is a parameter to be determined later; the pattern P is obtained by concatenating a number (namely $z = \mathcal{N}s^2$) of constituent parts P_i of length $\mathcal{O}(\mathcal{N}/s)$, each one built with five letters from disjoint subalphabets. The ED text $\tilde{T}$ is composed of three parts: the central part consists of three degenerate segments, the first one encoding the 1s of matrix A, the second one those of matrix B and the third one those of matrix C. These segments are built in such a way that the concatenation of strings of subsequent segments is of the same form as the pattern's building blocks. This central part is then padded to the left and to the right with sets containing appropriately chosen concatenations of substrings P_i of P, so that an occurrence of the pattern in the text implies that one of its building blocks matches the central part of the text, thus corresponding to a triangle. Formally:

THEOREM 1.1. *If the EDSM problem can be solved in $\mathcal{O}(nm^{1.5-\epsilon}+N)$ time, for any $\epsilon > 0$, with a combinatorial algorithm, then there exists a truly subcubic* *combinatorial algorithm for TD.*

Proof. Consider an instance of TD, where we are given three $\mathcal{N} \times \mathcal{N}$ Boolean matrices A, B, C, and the question is to check if there exist i, j, k such that $A[i, j] = B[j, k] = C[k, i] = 1$. Let s be a parameter, to be determined later, that corresponds to decomposing B into blocks of size $(\mathcal{N}/s) \times (\mathcal{N}/s)$. We reduce to an instance of EDSM over an alphabet Σ of size $\mathcal{O}(\mathcal{N})$. Let us remark that, since we search for exact occurrences of the pattern, it would also be possible to assume that the instance of EDSM we reduce to is over a constant-sized (binary) alphabet. We could in fact replace each letter of the $\mathcal{O}(\mathcal{N})$-sized alphabet with its binary encoding, increasing the length of the involved strings by only a logarithmic factor.

Pattern P. We construct P by concatenating, in some fixed order, the following strings:

$$P(i, x, y) = v(i)xa^{\mathcal{N}/s}x\$ya^{\mathcal{N}/s}yv(i)$$

for every $i = 1, 2, \ldots, \mathcal{N}$ and $x, y = 1, 2, \ldots, s$, where $a \in \Sigma_1$, $\$ \in \Sigma_2$, $x \in \Sigma_3$, $y \in \Sigma_4$, $v(i) \in \Sigma_5$, and $\Sigma_1, \Sigma_2, \ldots, \Sigma_5$ are disjoint subsets of Σ.

ED text $\tilde{T}$. The text $\tilde{T}$ consists of three parts. Its middle part encodes all the entries equal to 1 in matrices A, B and C, and consists of three string sets $\mathcal{X} = \mathcal{X}_1 \cdot \mathcal{X}_2 \cdot \mathcal{X}_3$, where:

1. $\mathcal{X}_1$ contains all strings of the form $v(i)xa^j$, for some $i \in [\mathcal{N}]$, $x \in [s]$ and $j \in [\mathcal{N}/s]$ such that $A[i, (x-1) \cdot (\mathcal{N}/s) + j] = 1$;
2. $\mathcal{X}_2$ contains all strings of the form $a^{\mathcal{N}/s-j}\ x\$ya^{\mathcal{N}/s-k}$, for some $x, y \in [s]$ and $j, k \in [\mathcal{N}/s]$ such that $B[(x-1) \cdot (\mathcal{N}/s) + j, (y-1) \cdot (\mathcal{N}/s) + k] = 1$, i.e., if the corresponding entry of B is 1;
3. $\mathcal{X}_3$ contains all strings of the form $a^k yv(i)$, for some $i \in [\mathcal{N}]$, $y \in [s]$ and $k \in [\mathcal{N}/s]$ such that $C[(y-1) \cdot (\mathcal{N}/s) + k, i] = 1$.

It is easy to see that $|P(i, x, y)| = \mathcal{O}(\mathcal{N}/s)$. This implies the following:

1. The length of the pattern is $m = \mathcal{O}(\mathcal{N} \cdot s^2 \cdot \mathcal{N}/s) = \mathcal{O}(\mathcal{N}^2 \cdot s)$;
2. The total length of $\mathcal{X}$ is $||\mathcal{X}|| = \mathcal{O}(\mathcal{N} \cdot s \cdot \mathcal{N}/s \cdot \mathcal{N}/s + s^2 \cdot (\mathcal{N}/s)^2 \cdot \mathcal{N}/s + \mathcal{N} \cdot s \cdot \mathcal{N}/s \cdot \mathcal{N}/s) = \mathcal{O}(\mathcal{N}^3/s)$.

By the above construction, we obtain the following fact.

FACT 1. *$P(i, x, y)$ matches $\mathcal{X}$ if and only if, for some $j, k = 1, 2, \ldots, \mathcal{N}/s$, we have $A[i, (x-1) \cdot (\mathcal{N}/s) + j] = 1$, $B[(x-1) \cdot (\mathcal{N}/s) + j, (y-1) \cdot (\mathcal{N}/s) + k] = 1$ and $C[(y-1) \cdot (\mathcal{N}/s) + k, i] = 1$.*

Solving the TD problem thus reduces to taking the disjunction of all such conditions. Let us write down all strings $P(i, x, y)$ in some arbitrary but fixed order to obtain $P = P_1 P_2 \ldots P_z$ with $z = \mathcal{N}s^2$ being a power of 2, where every $P_t = P(i, x, y)$, for some i, x, y. We aim to construct a small number of sets of strings that, when considered as an ED text, match any prefix $P_1 P_2 \ldots P_t$ of the pattern, $1 \leq t \leq z-1$; a similar construction can be carried on to obtain sets of strings that match any suffix $P_k \ldots P_{z-1} P_z$, $2 \leq k \leq z$. These sets will then be added to the left and to the right of $\mathcal{X}$, respectively, to obtain the ED text $\tilde{T}$.

ED Prefix. We construct $\log z$ sets of strings as follows. The first one contains the empty string ε and $P_1 P_2 \ldots P_{z/2}$. The second one contains ε, $P_1 P_2 \ldots P_{z/4}$ and $P_{z/2+1} \ldots P_{z/2+z/4}$. The third one contains ε, $P_1 P_2 \ldots P_{z/8}$, $P_{z/4+1} \ldots P_{z/4+z/8}$, $P_{z/2+1} \ldots P_{z/2+z/8}$ and $P_{z/2+z/4+1} \ldots P_{z/2+z/4+z/8}$.

Formally, for every $i = 1, 2, \ldots, \log z$, the i-th of such sets is:

$$\tilde{T}_i^p = \varepsilon \cup \{P_{j\frac{z}{2^{i-1}}+1} \ldots P_{j\frac{z}{2^{i-1}}+\frac{z}{2^i}} \mid j = 0, 1, \ldots, 2^{i-1} - 1\}. \quad (393)$$

ED Suffix. We similarly construct $\log z$ sets to be appended to $\mathcal{X}$:

$$\tilde{T}_i^s = \varepsilon \cup \{P_{z-j\frac{z}{2^{i-1}}-\frac{z}{2^i}+1} \ldots P_{z-j\frac{z}{2^{i-1}}} \mid j = 0, 1, \ldots, 2^{i-1}-1\}.$$

The total length of all the ED prefix and ED suffix strings is $\mathcal{O}(\log z \cdot \mathcal{N}^2 \cdot s) = \mathcal{O}(\mathcal{N}^2 \cdot s \cdot \log \mathcal{N})$. The whole ED text $\tilde{T}$ is thus: $\tilde{T} = \tilde{T}_1^p \cdot \cdots \cdot \tilde{T}_{\log z}^p \cdot \mathcal{X} \cdot \tilde{T}_{\log z}^s \cdot \cdots \cdot \tilde{T}_1^s$. We next show how a solution of such instance of EDSM corresponds to the solution of TD.

LEMMA 4.1. *The pattern P occurs in the ED text $\tilde{T}$ if and only if there exist i, j, k such that $A[i, j] = B[j, k] = C[k, i] = 1$.*

Proof. By Fact 1, if such i, j, k exist then P_t matches $\mathcal{X}$, for some $t \in \{1, \ldots, z\}$. Then, by construction of the sets $\tilde{T}_i^p$ and $\tilde{T}_i^s$, the prefix $P_1 \ldots P_{t-1}$ matches the ED prefix (this can be proved by induction), and similarly the suffix $P_{t+1} \ldots P_z$ matches the ED suffix, so the whole P matches $\tilde{T}$, and so P occurs therein. In the other direction, assume that there is an occurrence of the pattern P in $\tilde{T}$. Because the letter \$ appears only in the center of every P_i and in the strings from $\mathcal{X}_2$, and it can be verified that in any string from $\tilde{T}_1^p \cdot \cdots \cdot \tilde{T}_{\log z}^p$ or $\tilde{T}_{\log z}^s \cdot \cdots \cdot \tilde{T}_1^s$ there are fewer than z such letters, it must be the case that for some P_t its \$ is aligned with a \$ from some $X_2 \in \mathcal{X}_2$. But then by the subalphabets being disjoint we must have $X_1 X_2 X_3 = P_t$ for some $X_1 \in \mathcal{X}_1$, $X_2 \in \mathcal{X}_2$, $X_3 \in \mathcal{X}_3$, and by Fact 1 there exists a triangle. ∎

Note that for the EDSM problem we have $m = \mathcal{N}^2 \cdot s$, $n = 1 + 2\log z$ and $N = ||\mathcal{X}|| + \mathcal{O}(\mathcal{N}^2 \cdot s \cdot \log \mathcal{N})$. Thus if we had a solution running in $\mathcal{O}(\log z \cdot m^{1.5-\epsilon} + ||\mathcal{X}|| + \mathcal{N}^2 \cdot s \cdot \log \mathcal{N}) = \mathcal{O}(\log \mathcal{N} \cdot (\mathcal{N}^2 \cdot s)^{1.5-\epsilon} + \mathcal{N}^3/s)$ time, for some $\epsilon > 0$, by choosing a sufficiently small $\alpha > 0$ and setting $s = \mathcal{N}^\alpha$ we would obtain, for some $\delta > 0$, an $\mathcal{O}(\mathcal{N}^{3-\delta})$-time algorithm for TD. This ends the proof of Theorem 1.1. ∎

In order to show that AP cannot be solved in time $\mathcal{O}(m^{1.5-\epsilon} + N)$ with a combinatorial algorithm unless there is a truly subcubic combinatorial algorithm for BMM (Theorem 1.3), in Section 3, we have exhibited a fully detailed reduction from BMM to the AP problem. However, now that we have proved a lower bound for EDSM, we remark that Theorem 1.1 also implies Theorem 1.3. Indeed, assuming that the AP problem can be solved in $\mathcal{O}(m^{1.5-\epsilon} + N)$ time, then by calling the AP problem n times (as described in Section 2 under the definition of the EDSM problem), we could solve the EDSM problem in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time. At that point, we could apply Theorem 1.1 and obtain a truly subcubic combinatorial algorithm for BMM.

5. An $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$-time Algorithm for EDSM. Our goal is to design a non-combinatorial $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$-time algorithm for EDSM, which in turn can be achieved with a non-combinatorial $\tilde{\mathcal{O}}(m^{\omega-1} + N)$-time algorithm for the AP problem, that is the bottleneck of EDSM (cf. [39]).

We reduce AP to a logarithmic number of restricted instances of the same problem, based on the length of the strings in $\mathcal{S}$. We start by giving a lemma that we will use to process naïvely the strings of length up to a constant c, to be determined later, in $\mathcal{O}(m \log m + N)$ time.

LEMMA 5.1. *For any integer t, all strings in $\mathcal{S}$ of length at most t can be processed in $\mathcal{O}(m \log m + mt + N)$ time.*

Proof. We first construct the suffix tree ST of P in $\mathcal{O}(m \log m)$ time [65]. Let us remark that we are spending $\mathcal{O}(m \log m)$ time and not just $\mathcal{O}(m)$ so as to avoid any assumptions on the size of the alphabet. For every explicit node $u \in ST$, we construct a perfect hash function mapping the first letter on every edge outgoing from u to the corresponding edge. This takes $\mathcal{O}(m \log m)$ total time [61] and allows us to navigate in ST in constant time per letter. For every $S \in \mathcal{S}$, find and mark its corresponding (implicit or explicit) node of ST. This takes $\mathcal{O}(N)$ time overall. For every possible length $t' \leq t$, scan P with a window of length t' while maintaining its corresponding (implicit or implicit) node of ST. To move the window to the right, we first follow the suffix link of the current node (if the node is implicit, we follow the suffix link of its nearest explicit ancestor, and then descend to find the node corresponding to the truncated window), and then follow the appropriate edge. This takes $\mathcal{O}(mt)$ total time by standard amortization based on counting the number of explicit ancestors of the current node. If the current window $P[i \mathinner{.\,.} (i+t'-1)]$ corresponds to a marked node of ST and additionally $U[i-1]=1$, we set $V[i+t'-1]=1$. ◻

We build the restricted instances of the AP problem by considering only strings in $\mathcal{S}_k \subseteq \mathcal{S}$ of length in $[(19/18)^k, (19/18)^{k+1})$ for each integer k ranging from $\left\lceil \frac{\log c}{\log(19/18)} \right\rceil$ to $\left\lfloor \frac{\log m}{\log(19/18)} \right\rfloor$. These sets form a partition of the set of all strings in $\mathcal{S}$ of lengths up to m; longer strings are not needed when solving the AP problem.

For each integer k from $\left\lceil \frac{\log c}{\log(19/18)} \right\rceil$ to $\left\lfloor \frac{\log m}{\log(19/18)} \right\rfloor$, let ℓ be an integer such that the length of every string in $\mathcal{S}_k$ belongs to $[9/8 \cdot \ell, 5/4 \cdot \ell)$. Note that such an integer always exists for an appropriate choice of the integer constant c. In fact, it must hold that

$$\frac{9}{8} \cdot \ell \;\leq\; \left(\frac{19}{18}\right)^k < \;\left(\frac{19}{18}\right)^{k+1} \;\leq\; \frac{5}{4} \cdot \ell \iff \frac{4}{5} \cdot \left(\frac{19}{18}\right)^{k+1} \;\leq\; \ell \;\leq\; \frac{8}{9} \cdot \left(\frac{19}{18}\right)^k.$$

To ensure that there exists an *integer* ℓ satisfying such conditions, we require that

$$\frac{4}{5} \cdot \left(\frac{19}{18}\right)^{k+1} + 1 \;\leq\; \frac{8}{9} \cdot \left(\frac{19}{18}\right)^k \iff \frac{45}{2} \;\leq\; \left(\frac{19}{18}\right)^k.$$

The last equation holds for $k \geq 58$, implying that we will process naïvely the strings of length up to $c = 23$, and each $\mathcal{S}_k$, for k ranging from 58 to $\left\lfloor \frac{\log m}{\log(19/18)} \right\rfloor$, will be processed separately as described in the next paragraph.

Remark 5.2. The length of every string in $\mathcal{S}$ belonging to $[9/8 \cdot \ell, 5/4 \cdot \ell)$ implies that every string in $\mathcal{S}$ contains at most $\ell/4$ length-ℓ substrings (and at least $1 + \ell/8$ of them).

Denoting by N_k the total size of strings in $\mathcal{S}_k$, we have that, if we solve every such instance of AP in $\mathcal{O}(N_k + f(m))$ time, then we can solve the original instance of AP in $\mathcal{O}(N + f(m) \log m)$ time by taking the disjunction of the results. Switching to $\tilde{\mathcal{O}}$ notation that disregards polylog factors, it thus suffices to solve each such instance of the AP problem in $\tilde{\mathcal{O}}(N + m^{\omega-1})$ time.

We further partition the strings in $\mathcal{S}_k$ into three types, compute the corresponding bit vector V for each type separately and, finally, take the disjunction of the resulting bit vectors V to obtain the answer for each restricted instance.

Partitioning $\mathcal{S}_k$. Keeping in mind that from now on (until Section 5.4) we address the AP problem assuming that $\mathcal{S}$ only contains strings of length in $[9/8 \cdot \ell, 5/4 \cdot \ell)$, and thus is in fact $\mathcal{S}_k$, to lighten the notation we now switch back to denote it simply with $\mathcal{S}$, and similarly write N to denote the total length of all strings given as the input to the AP problem. The three types of strings are as follows:

Type 1: Strings $S \in \mathcal{S}$ such that every length-ℓ substring of S is not strongly periodic.

Type 2: Strings $S \in \mathcal{S}$ containing at least one length-ℓ substring that is not strongly periodic and at least one length-ℓ substring that is strongly periodic.

Type 3: Strings $S \in \mathcal{S}$ such that every length-ℓ substring of S is strongly periodic (in Lemma 5.3 we show that in this case $\mathrm{per}(S) \leq \ell/4$).

These three types are evidently a partition of $\mathcal{S}$. We start with showing that, in fact, strings of type 3 are exactly strings with period at most $\ell/4$. It is straightforward to verify that strings with period at most $\ell/4$ are such that all their length-ℓ substrings have period at most $\ell/4$; the following lemma addresses the other (less obvious) direction.

LEMMA 5.3. *Let S be a string. If $\mathrm{per}(S[j\,..\,j+\ell-1]) \leq \ell/4$ for every j then $\mathrm{per}(S) \leq \ell/4$.*

Proof. We first show that, for any string W and letters a, b, if $\mathrm{per}(aW) \leq |aW|/4$ and $\mathrm{per}(Wb) \leq |Wb|/4$ then $\mathrm{per}(aW) = \mathrm{per}(Wb)$. This follows from Lemma 2.1: since $\mathrm{per}(aW)$ and $\mathrm{per}(Wb)$ are both periods of W and $(1+|W|)/4 \leq |W|/2$, then we have that $d = \gcd(\mathrm{per}(aW), \mathrm{per}(Wb))$ is a period of W. Assuming by contradiction that $\mathrm{per}(aW) \neq \mathrm{per}(Wb)$, then it must be that either $d < \mathrm{per}(aW)$ or $d < \mathrm{per}(Wb)$; by symmetry it is enough to consider the former possibility, and we claim that then d is a period of aW. Indeed, $a = W[\mathrm{per}(aW)]$ (observe that, since $\mathrm{per}(aW) \leq (1+|W|)/4 \leq |W|/2$, in particular $\mathrm{per}(aW) < |W|$) and $W[i] = W[i+d]$ for any $i = 1, 2, \ldots, |W|-d$, so by $\mathrm{per}(aW)$ being a multiple of d, we obtain that $a = W[\mathrm{per}(aW)] = W[d]$, which is a contradiction because, by definition of $\mathrm{per}(aW)$, we have that $d < \mathrm{per}(aW)$ cannot be a period of aW.

If $\mathrm{per}(S[j\,..\,j+\ell-1]) \leq \ell/4$ for every j then by the above reasoning the periods of all substrings $S[j\,..\,j+\ell-1]$ are all equal to the same $p \leq \ell/4$. But then $S[i] = S[i+p]$ for every i, so $\mathrm{per}(S) \leq \ell/4$. ∎

Before proceeding with the algorithm, we show that, for each string $S \in \mathcal{S}$, we can determine its type in $\mathcal{O}(|S|)$ time.

LEMMA 5.4. *Given a string S we can determine its type in $\mathcal{O}(|S|)$ time.*

Proof. It is well-known that $\mathrm{per}(T)$ can be computed in $\mathcal{O}(|T|)$ time for any string T (cf. [28]). We partition S into blocks $T_\alpha = S[\alpha\lfloor\ell/2\rfloor\,..\,(\alpha+1)\lfloor\ell/2\rfloor-1]$ of size $\lfloor\ell/2\rfloor$, and compute $\mathrm{per}(T_\alpha)$ for every α in $\mathcal{O}(|S|)$ total time. Observe that every substring $S[i\,..\,i+\ell-1]$ contains at least one whole block inside.

If $\mathrm{per}(T_\alpha) > \ell/4$ then the period of any substring $S[i\,..\,i+\ell-1]$ that contains T_α is also larger than $\ell/4$. Consequently, if $\mathrm{per}(T_\alpha) > \ell/4$ for every α, then we declare S to be of type 1.

Consider any α such that $p = \mathrm{per}(T_\alpha) \leq \ell/4$. If the period p' of a substring $S' = S[i\,..\,i+\ell-1]$ that contains T_α is at most $\ell/4$, then in fact it must be equal to p, because $p' \geq p$ and so, by Lemma 2.1 applied on T_α, p' must be a multiple of p and, by repeatedly applying $S'[j] = S'[j+p']$ and $T_\alpha[j] = T_\alpha[j+p]$ and using the fact that T_α occurs inside S', we conclude that in fact $S'[j] = S'[j+p]$ for any j, and thus $p' = p$. This allows us to check whether there exists a substring $S' = S[i\,..\,i+\ell-1]$ that contains T_α such that $\mathrm{per}(S') \leq \ell/4$ by computing, in $\mathcal{O}(\ell)$ time, how far the period p extends to the left and to the right of T_α in $T_{\alpha-1}T_\alpha T_{\alpha+1}$ (if either $T_{\alpha-1}$ or $T_{\alpha+1}$ do not exist, then we do not extend the period in the corresponding direction). There exists such a substring S' if and only if the length of the extended substring with period p is at least ℓ. Therefore, for every α we can check in $\mathcal{O}(\ell)$ time if there exists a length-ℓ substring S' containing T_α with $\text{per}(S') \le \ell/4$. By repeating this procedure for every α, we can distinguish between S of type 2 and S of type 3 in $\mathcal{O}(|S|)$ total time. ∎

Since we have shown how to efficiently partition the strings of S into the three types, in what follows we present our solution of the AP problem for each type of strings separately.

5.1. Type 1 Strings. In this section we show how to solve a restricted instance of the AP problem where every string $S \in \mathcal{S}$ is of type 1, that is, each of its length-ℓ substrings is not strongly periodic, and furthermore $|S| \in [9/8 \cdot \ell, 5/4 \cdot \ell)$ for some $\ell \le m$. Observe that all (and hence at most $\ell/4$ by Remark 5.2) length-ℓ substrings of any $S \in \mathcal{S}$ must be distinct, as otherwise we would be able to find two occurrences of a length-ℓ substring at distance at most $\ell/4$ in S, making the period of the substring at most $\ell/4$ and contradicting the assumption that S is of type 1.

We start with constructing the suffix tree ST of P (our pattern in the EDSM problem) and storing, for every node, the first letters on its outgoing edges in a static dictionary with constant access time. Then, for every $S \in \mathcal{S}$, we check in $\mathcal{O}(|S|)$ time using ST if it occurs in P and, if not, we disregard it from further consideration. Therefore, from now on we assume that all strings S, and thus all their length-ℓ substrings, occur in P. We will select a set of length-ℓ substrings of P, called the *anchors*, each represented by one of its occurrences in P, such that:

1. The total number of occurrences of all anchors in P is $\mathcal{O}(m/\ell \cdot \log m)$.
2. For every $S \in \mathcal{S}$, at least one of its length-ℓ substrings is an anchor.
3. The total number of occurrences of all anchors in strings $S \in \mathcal{S}$ is $\mathcal{O}(|\mathcal{S}| \cdot \log m)$.

We formalize this using the following auxiliary problem, which is a strengthening of a well-known *Hitting Set* problem, which given a collection of m sets over $[n]$, each of size at least k, asks to choose a subset of $[n]$ of size $\mathcal{O}(n/k \cdot \log m)$ that nontrivially intersects every set.

> Node Selection (NS)
> **INPUT:** A bipartite graph $G = (U, V, E)$ with $\deg(u) \in (d, 2d]$ for every $u \in U$ and weight $w(v)$ for every $v \in V$, where $W = \sum_{v \in V} w(v)$.
> **OUTPUT:** A set $V' \subseteq V$ of total weight $\mathcal{O}(W/d \cdot \log |U|)$ such that $N[u] \cap V' \neq \emptyset$ for every node $u \in U$, and $\sum_{u \in U} |N[u] \cap V'| = \mathcal{O}(|U| \log |U|)$.

We reduce the problem of finding anchors to an instance of the NS problem, by building a bipartite graph G in which the nodes in U correspond to strings $S \in \mathcal{S}$, the nodes in V correspond to distinct length-ℓ substrings of P, and there is an edge (u, v) if the length-ℓ string corresponding to v occurs in the string S corresponding to u. Using suffix links, we can find the node of the suffix tree corresponding to every length-ℓ substring of S in $\mathcal{O}(|S|)$ total time, so the whole construction takes $\mathcal{O}(m \log m + \sum_{S \in \mathcal{S}} |S|) = \mathcal{O}(m \log m + N)$ time. The size of G is $\mathcal{O}(m + N)$, and the degree of every node in U belongs to $(\ell/8, \ell/4]$. We set the weight of a node $v \in V$ to be its number of occurrences in P, and solve the obtained instance of the NS problem to obtain the set of anchors. We remark that, because each string $S \in \mathcal{S}$ can be assumed to be a substring of P and we do not need to keep duplicate strings in $\mathcal{S}$, we have $\log |U| = \Theta(\log m)$ and the three required properties indeed hold assuming that we have found a solution. However, it is not immediately clear that an instance of the NS problem always has a solution. We show that indeed it does, and that it can be found in linear time.

LEMMA 5.5. *A solution to an instance of the NS problem always exists and can be found in linear time in the size of G.*

Proof. We first show a solution that uses the probabilistic method and leads us to an efficient Las Vegas algorithm; we will then derandomize the solution using the method of conditional expectations.

We independently choose each node of V with probability p to obtain the set V' of selected nodes. The expected total weight of V' is $\sum_{v\in V} p\cdot w(v) = p\cdot W$, so by Markov's inequality it exceeds $4p\cdot W$ with probability at most $1/4$. For every node $u\in U$, the probability that $N[u]$ does not intersect V' is at most $(1-p)^d \le e^{-pd}$. Finally, $\mathbb{E}[\sum_{u\in U}|N[u]\cap V'|] \le |U|\cdot 2pd$, so by Markov's inequality $\sum_{u\in U}|N[u]\cap V'|$ exceeds $|U|\cdot 8pd$ with probability at most $1/4$. We set $p=\ln(4|U|)/d$ (observe that if $p>1$ then we can select all nodes in V). By union bound, the probability that V' is not a valid solution is at most $3/4$, so indeed a valid solution exists. Furthermore, this reasoning gives us an efficient Las Vegas algorithm that chooses V' randomly as described above and then verifies if it constitutes a valid solution. Each iteration takes linear time in the size of G, and the expected number of required iterations is constant.

To derandomize the above procedure we apply the method of conditional expectations. Let $X_1, X_2, \ldots$ be the binary random variables corresponding to the nodes of V. Recall that in the above proof we set $X_i = 1$ with probability p. Now we will choose the values of $X_1, X_2, \ldots$ one-by-one. Define a function $f(X_1, X_2, \ldots)$ that bounds the probability that $X_1, X_2, \ldots$ corresponds to a valid solution as follows:

$$f(X_1, X_2, \ldots) = \frac{\sum_v X_v\cdot w(v)}{4W/d\cdot \ln(4|U|)} + \sum_{u\in U}\prod_{v\in N[u]}(1-X_v) + \frac{\sum_{u\in U}\sum_{v\in N[u]} X_v}{8|U|\ln(4|U|)}.$$

As explained above, we have $\mathbb{E}[f(X_1, X_2, \ldots)] = 3/4$. Assume that we have already fixed the values $X_1 = x_1, \ldots, X_i = x_i$. Then there must be a choice of $X_{i+1} = x_{i+1}$ that does not increase the expected value of $f(X_1, X_2, \ldots)$ conditioned on the already chosen values. We want to compare the following two quantities:

$$\mathbb{E}[f(X_1, X_2, \ldots) \mid X_1 = x_1, \ldots, X_i = x_i, X_{i+1} = 0]$$
$$\mathbb{E}[f(X_1, X_2, \ldots) \mid X_1 = x_1, \ldots, X_i = x_i, X_{i+1} = 1]$$

and choose x_{i+1} corresponding to the smaller one. Canceling out the shared terms, we need to compare the expected values of:

$$0 \quad + \sum_{u\in N[i+1]}\prod_{v\in N[u]}(1-X_v) \; + \quad 0 \qquad \text{and}$$
$$\frac{w(i+1)}{4W/d\cdot\ln(4|U|)} \; + \quad 0 \quad + \frac{\deg(i+1)}{8|U|\ln(4|U|)}.$$

The second quantity can be computed in constant time. We claim that (ignoring the issue of numerical precision) the first quantity can be computed in time $\mathcal{O}(\deg(i+1))$ after a linear-time preprocessing as follows. In the preprocessing we compute and store $E[i] = (1-p)^i$, for every $i = 0, 1, \ldots, |V|$ in $\mathcal{O}(|V|)$ total time. Then, during the computation we maintain, for every $u\in U$, the number $c[u]$ of $v\in N[u]$ for which we still need to choose the value X_e, and a single bit $b[u]$ denoting whether for some

$v \in N[u] \cap \{1, \ldots, i\}$ we already have $x_v = 1$. This information can be updated in $\mathcal{O}(\deg(i+1))$ time after selecting x_{i+1}. Now to compute the first quantity, we iterate over $u \in N[i+1]$ and, if $b[u] = 0$ then we add $E[c[u]]$ to the result. Finally, we claim that it is enough to implement all calculations with precision of $\Theta(\log |V|)$ bits. This is because such precision allows us to calculate both quantities with relative accuracy $1/(8|V|)$, so the expected value of $f(X_1, X_2, \ldots)$ might increase by a factor of $(1+1/(4|V|))$ in every step, which is at most $(1+1/(4|V|))^{|V|} \le e^{1/4}$ overall. This still guarantees that the final value is at most $3/4 \cdot e^{1/4} < 1$, so we obtain a valid solution. ∎

In the rest of this section we explain how to compute the bit vector V from the bit vector U, and thus solve the AP problem, after having obtained a set $\mathcal{A}$ of anchors. For any $S \in \mathcal{S}$, since S contains an occurrence of at least one anchor $H \in \mathcal{A}$, say $S[j \mathinner{.\,.} (j+|H|-1)] = H$, so any occurrence of S in P can be generated by choosing some occurrence of H in P, say $P[i \mathinner{.\,.} (i+|H|-1)] = H$, and then checking that $S[1 \mathinner{.\,.} (j-1)] = P[(i-j+1) \mathinner{.\,.} (i-1)]$ and $S[(j+|H|) \mathinner{.\,.} |S|] = P[(i+|H|) \mathinner{.\,.} (i+|S|-j)]$. In other words, $S[1 \mathinner{.\,.} (j-1)]$ should be a suffix of $P[1 \mathinner{.\,.} (i-1)]$ and $S[(j+|H|) \mathinner{.\,.} |S|]$ should be a prefix of $P[(i+|H|) \mathinner{.\,.} |P|]$. In such case, we say that the occurrence of S in P is generated by H. By the properties of $\mathcal{A}$, any occurrence of $S \in \mathcal{S}$ is generated by $occ_S \ge 1$ occurrences of anchors, where $\sum_{S \in \mathcal{S}} occ_S = \mathcal{O}(|\mathcal{S}| \log m)$. For every $H \in \mathcal{A}$ we create a separate data structure $D(H)$ responsible for setting $V[i+|S|-1] = 1$, when $U[i-1] = 1$ and $P[i \mathinner{.\,.} (i+|S|-1)] = S$ is generated by H. We now first describe what information is used to initialize each $D(H)$, and how this is later processed to update V.

Initialization. $D(H)$ consists of two compact tries $T(H)$ and $T^r(H)$. For every occurrence of H in P, denoted by $P[i \mathinner{.\,.} (i+|H|-1)] = H$, $T(H)$ should contain a leaf corresponding to $P[(i+|H|) \mathinner{.\,.} |P|]\$$ and $T^r(H)$ should contain a leaf corresponding to $(P[1 \mathinner{.\,.} (i-1)])^r\$$, both decorated with position i. Additionally, $D(H)$ stores a list $L(H)$ of pairs of nodes (u, v), where $u \in T^r(H)$ and $v \in T(H)$ (both nodes might be implicit or explicit). Each such pair corresponds to an occurrence of H in a string $S \in \mathcal{S}$, $S[j \mathinner{.\,.} (j+|H|-1)] = H$, where u is the node of $T^r(H)$ corresponding to $(S[1 \mathinner{.\,.} (j-1)])^r\$$ and v is the node of $T(H)$ corresponding to $S[(j+|H|+1) \mathinner{.\,.} |S|]\$$. We claim that $D(H)$, for all H, can be constructed in $\mathcal{O}(m \log m + N)$ total time.

We first construct the suffix tree ST of $P\$$ and the suffix tree ST^r of $P^r\$$ (again in $\mathcal{O}(m \log m)$ time not to make assumptions on the alphabet). We augment both trees with data for answering both *weighted ancestor* (WA) and *lowest common ancestor* (LCA) queries, that are defined as follows. For a rooted tree T on n nodes with an integer weight $\mathcal{D}(v)$ assigned to every node u, such that the weight of the root is zero and $\mathcal{D}(u) < \mathcal{D}(v)$ if u is the parent of v, we say that a node v is a weighted ancestor of a node v at depth ℓ, denoted by $\mathrm{WA}_T(u, \ell)$, if v is the highest ancestor of u with weight at least ℓ. Such queries can be answered in $\mathcal{O}(\log n)$ time after an $\mathcal{O}(n)$ preprocessing [32]. For a rooted tree T, $\mathrm{LCA}_T(u, v)$ is the lowest node that is an ancestor of both u and v. Such queries can be answered in $\mathcal{O}(1)$ time after an $\mathcal{O}(n)$ preprocessing [12]. Recall that every anchor H is represented by one of its occurrences in P. Using WA queries, we can access in $\mathcal{O}(\log m)$ time the nodes corresponding to H and H^r, respectively, and extract a lexicographically sorted list of suffixes following an occurrence of H in $P\$$ and a lexicographically sorted list of reversed prefixes preceding an occurrence of H in $P^r\$$ in time proportional to the number of such occurrences. Then, by iterating over the lexicographically sorted list of suffixes and using LCA queries on ST we can build $T(H)$ in time proportional to the length of the list, and

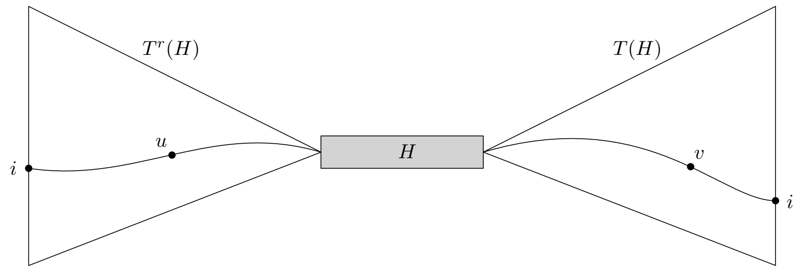

FIG. 1. *An occurrence of S starting at position i in P is generated by H: (u,v) corresponds to $S[j\,..\,(j+|H|-1)] = H$ and i appears in the subtree of $T^r(H)$ rooted at u, as well as in the subtree of $T(H)$ rooted at v.*

similarly we can build $T^r(H)$. To construct $L(H)$ we start by computing, for every $S \in \mathcal{S}$ and $j = 1, \ldots, |S|$, the node of ST^r corresponding to $(S[1\,..\,j])^r$ and the node of ST corresponding to $S([(j+1)\,..\,|S|]$ (the nodes might possibly be implicit). This takes only $\mathcal{O}(|S|)$ time, by using suffix links. We also find, for every length-ℓ substring $S[j\,..\,(j+\ell-1)]$ of S, an anchor $H \in \mathcal{A}$ such that $S[j\,..\,(j+\ell-1)] = H$, if any exists. This can be done by finding the nodes (implicit or explicit) of ST that correspond to the anchors, and then scanning over all length-ℓ substrings while maintaining the node of ST corresponding to the current substring using suffix links in $\mathcal{O}(|S|)$ total time. After having determined that $S[j\,..\,(j+\ell-1)] = H$ we retrieve the previously found nodes u of ST^r and v of ST corresponding to $(S[1\,..\,(j-1)])^r$ and $S[(j+\ell)\,..\,|S|]$, respectively. Then we look up the node $u' \in T^r(H)$ corresponding to u and the node $v' \in T(H)$ corresponding to v, and if they both exist we add (u,v) to $L(H)$. This lookup can be implemented in $\mathcal{O}(\log m)$ time by binary searching over the leaves of the compact tries. By construction, we have the following property, also illustrated in Figure 1.

FACT 2. *A string $S \in \mathcal{S}$ starts at position $i-j+1$ in P if and only if, for some anchor $H \in \mathcal{A}$, $L(H)$ contains a pair (u,v) corresponding to $S[j\,..\,(j+|H|-1)] = H$, such that the subtree of $T^r(H)$ rooted at u and that of $T(H)$ rooted at v contain a leaf decorated with i.*

Note that the overall size of all lists $L(H)$, when summed up over all $H \in \mathcal{A}$, is $\sum_{S\in\mathcal{S}} occ_S = \mathcal{O}(|\mathcal{S}|\log m)$, and since each S is of length at least ℓ this is $\mathcal{O}(N/\ell \cdot \log m)$.

Processing. The goal of processing $D(H)$ is to efficiently process all occurrences generated by H. As a preliminary step, we decompose $T^r(H)$ and $T(H)$ into heavy paths. Then, for every pair of leaves $u \in T^r(H)$ and $v \in T(H)$ decorated by the same i, we consider all heavy paths above u and v. Let $p = u_1 - u_2 - \ldots$ be a heavy path above u in $T^r(H)$ and $q = v_1 - v_2 - \ldots$ be a heavy path above v in $T(H)$, where u_1 is the head of p and v_1 is the head of q, respectively. Further, choose the largest x such that u is in the subtree rooted at u_x, and the largest y such that v is in the subtree rooted at v_y (this is well-defined by the choice of p and q, as u is in the subtree rooted at u_1 and v is in the subtree rooted at v_1). We add $(i, |\mathcal{L}(u_x)|, |\mathcal{L}(v_y)|)$ to an auxiliary list associated with the pair of heavy paths (p,q), where $\mathcal{L}(u)$ denotes the concatenation of the edge labels on the path from the root to node u. In the rest of the processing we work with each such list separately. Notice that the overall size of all auxiliary lists, when summed up over all $H \in \mathcal{A}$, is $\mathcal{O}(m/\ell \cdot \log^3 m)$, because there are at most $\log^2 m$ pairs of heavy paths above u and v decorated by the same i,

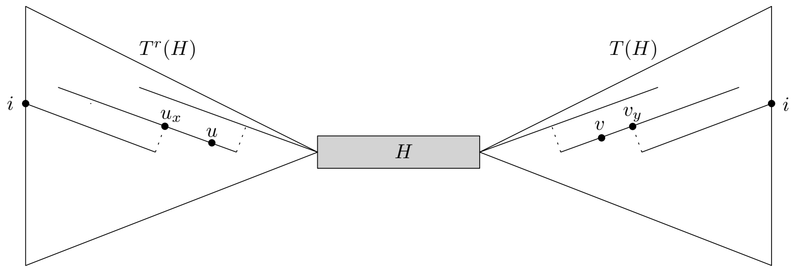

FIG. 2. *An occurrence of S starting at position i in P corresponds to a triple $(i, \mathcal{L}(u_x), \mathcal{L}(v_y))$ on some auxiliary list.*

and the total number of leaves in all trees $T^r(H)$ and $T(H)$ is bounded by the total number of occurrences of all anchors in P, which is $\mathcal{O}(m/\ell \cdot \log m)$. By Fact 2, there is an occurrence of a string $\mathcal{S}$ generated by H and starting at position $i-j+1$ in P if and only if $L(H)$ contains a pair (u, v) corresponding to $S[j \mathinner{.\,.} (j+|H|-1)] = H$ such that, denoting by p the heavy path containing u in $T^r(H)$ and by q the heavy path containing v in $T(H)$, the auxiliary list associated with (p, q) contains a triple (i, x, y) such that $x \geq |\mathcal{L}(u)|$ and $y \geq |\mathcal{L}(v)|$. This is illustrated in Figure 2. Henceforth, we focus on the problem of processing a single auxiliary list associated with (p, q), together with a list of pairs (u, v), such that u belongs to p and v belongs to q.

Processing an auxiliary list can be interpreted geometrically as follows: for every (i, x, y) we create a red point (x, y), and for every (u, v) we create a blue point $(|\mathcal{L}(u)|, |\mathcal{L}(v)|)$. Then, each occurrence of $S \in \mathcal{S}$ generated by H corresponds to a pair of points (p_1, p_2) such that p_1 is red, p_2 is blue, and p_1 dominates p_2. We further reduce this to a collection of simpler instances in which all red points already dominate all blue points. This can be done with a divide-and-conquer procedure which is essentially equivalent to constructing a 2D range tree [13]: we first apply a divide-and-conquer that splits the current set of points along the median x coordinate, and inside every each obtained subproblem consisting of the left and the right part we apply another divide-and-conquer that splits the current set of points along the median y coordinate. The total number of points in all obtained instances increases by a factor of $\mathcal{O}(\log^2 m)$, making the total number of red points in all instances $\mathcal{O}(m/\ell \cdot \log^5 m)$, while the total number of blue points is $\mathcal{O}(N/\ell \cdot \log^3 m)$. There is an occurrence of a string $S \in \mathcal{S}$ generated by H and starting at position $i-j+1$ in P if and only if some simpler instance contains a red point created for some (i, x, y) and a blue point created for some (u, v) corresponding to $S[j \mathinner{.\,.} (j+|H|-1)] = H$. In the following we focus on processing a single simpler instance.

To process a simpler instance we need to check if $U[i-j] = 1$, for a red point created for some (i, x, y) and a blue point created for some (u, v) corresponding to $S[j \mathinner{.\,.} (j+|H|-1)] = H$, and if so set $V[i-j+|S|] = 1$. This has a natural interpretation as an instance of BMM: we create a $\lceil 5/4 \cdot \ell \rceil \times \lceil 5/4 \cdot \ell \rceil$ matrix M such that $M[|S|-j, \lceil 5/4 \cdot \ell \rceil + 1 - j] = 1$ if and only if there is a blue point created for some (u, v) corresponding to $S[j \mathinner{.\,.} (j+|H|-1)] = H$; then for every red point created for some (i, x, y) we construct a bit vector $U_i = U[(i - \lceil 5/4 \cdot \ell \rceil) \mathinner{.\,.} (i-1)]$ (if $i < \lceil 5/4 \cdot \ell \rceil$, we pad U_i with 0s to make its length always equal to $\lceil 5/4 \cdot \ell \rceil$); calculate $V_i = M \times U_i$; and finally set $V[i+j] = 1$ whenever $V_i[j] = 1$ (and $i + j \leq m$).

LEMMA 5.6. *$V_i[k] = 1$ if and only if there is a blue point created for some (u, v) corresponding to $S[j \mathinner{.\,.} (j+|H|-1)] = H$ such that $U[i-j] = 1$ and $k = |S| - j$.*

Proof. By definition of $V_i = M \times U_i$, we have that $V_i[k] = 1$ if and only if $M[k,t] = 1$ for some t such that $U_i[t] = 1$. By definition of U_i, we have that $U_i[t] = 1$ if and only if $U[i - \lceil 5/4 \cdot \ell \rceil + t - 1] = 1$, and hence the previous condition can be rewritten as $M[k,t] = 1$ and $U[i - \lceil 5/4 \cdot \ell \rceil + t - 1] = 1$, or equivalently, by substituting $j = \lceil 5/4 \cdot \ell \rceil + 1 - t$, $M[k, \lceil 5/4 \cdot \ell \rceil + 1 - j] = 1$ and $U[i - j] = 1$. By definition of M, we have that $M[k, \lceil 5/4 \cdot \ell \rceil + 1 - j] = 1$ if and only if there is a blue point created for some (u, v) corresponding to $S[j \mathinner{.\,.} (j + |H| - 1)] = H$ with $k = |S| - j$, which proves the lemma. ∎

The total length of all vectors U_i and V_i is $\mathcal{O}(m \log^5 m)$, so we can afford to extract the appropriate fragment of U and then update the corresponding fragment of V. The bottleneck is computing the matrix-vector product $V_i = M \times U_i$. Since the total number of 1s in all matrices M is bounded by the total number of blue points, a naïve method would take $\mathcal{O}(N/\ell \cdot \log^3 m)$ time; we overcome this by processing together all multiplications concerning the same matrix M, thus amortizing the costs. Let $U_{i_1}, U_{i_2}, \ldots, U_{i_s}$ be all bit vectors that need to be multiplied with M, and let z a parameter to be determined later. We distinguish between two cases: (i) if $s < z$, then we compute the products naïvely by iterating over all 1s in M, and the total computation time, when summed up over all such matrices M, is $\mathcal{O}(N/\ell \cdot \log^3 m \cdot z)$; (ii) if $s \geq z$, then we partition the bit vectors into $\lceil s/z \rceil \leq s/z + 1$ groups of z (padding the last group with bit vectors containing all 0s) and, for every group, we create a single matrix whose columns contain all the bit vectors belonging to the group. Thus, we reduce the problem of computing all matrix-vector products $M \times U_i$ to that of computing $\mathcal{O}(s/z)$ matrix-matrix products of the form $M \times M'$, where M' is an $\lceil 5/4 \cdot \ell \rceil \times z$ matrix. Even if M' is not necessarily a square matrix, we can still apply the fast matrix multiplication algorithm to compute $M \times M'$ using the standard trick of decomposing the matrices into square blocks.

LEMMA 5.7. *If two $\mathcal{N} \times \mathcal{N}$ matrices can be multiplied in $\mathcal{O}(\mathcal{N}^\omega)$ time, then, for any $\mathcal{N} \geq \mathcal{N}'$, an $\mathcal{N} \times \mathcal{N}$ and an $\mathcal{N} \times \mathcal{N}'$ matrix can be multiplied in $\mathcal{O}((\mathcal{N}/\mathcal{N}')^2 \mathcal{N}'^\omega)$ time.*

Proof. We partition both matrices into blocks of size $\mathcal{N}' \times \mathcal{N}'$. There are $(\mathcal{N}/\mathcal{N}')^2$ such blocks in the first matrix and $\mathcal{N}/\mathcal{N}'$ in the second matrix. Then, to compute the product we multiply each block from the first matrix by the appropriate block in the second matrix in $\mathcal{O}(\mathcal{N}'^\omega)$ time, resulting in the claimed complexity. ∎

By applying Lemma 5.7, we can compute $M \times M'$ in $\mathcal{O}(\ell^2 z^{\omega-2})$ time (as long as we later verify that $5/4 \cdot \ell \geq z$), so all products $M \times U_i$ can be computed in $\mathcal{O}(\ell^2 z^{\omega-2} \cdot (s/z+1))$ time. Note that this case can occur only $\mathcal{O}(m/(\ell \cdot z) \cdot \log^5 m)$ times, because all values of s sum up to $\mathcal{O}(m/\ell \cdot \log^5 m)$. This makes the total computation time, when summed up over all such matrices M, $\mathcal{O}(\ell^2 z^{\omega-2} \cdot m/(\ell \cdot z) \cdot \log^5 m) = \mathcal{O}(\ell z^{\omega-3} \cdot m \log^5 m)$. We can now prove our final result for strings of type 1.

THEOREM 5.8. *An instance of the AP problem where all strings are of type 1 can be solved in $\tilde{\mathcal{O}}(m^{\omega-1} + N)$ time.*

Proof. The total time complexity is first $\mathcal{O}(m + N)$ to construct the graph G, then $\mathcal{O}(m \log m + N)$ to solve its corresponding instances of the NodeSelection problem and obtain the set of anchors H. The time to initialize all structures $D(H)$ is $\mathcal{O}(m \log m + N)$. For every $D(H)$, we obtain in $\mathcal{O}(m/\ell \cdot \log^5 m + N/\ell \cdot \log^3 m)$ time a number of simpler instances, and then construct the corresponding Boolean matrices M and bit vectors U_i in additional $\mathcal{O}(m \log^5 m)$ time. Note that some M might be sparse, so we need to represent them as a list of 1s. Then, summing up over all matrices

M and both cases, we spend $\mathcal{O}(N/\ell \cdot \log^3 m \cdot z + \ell z^{\omega-3} \cdot m \log^5 m)$ time. We would like to assume that $\ell \geq \log^3 m$ so that we can set $z = \ell / \log^3 m$. This is indeed possible, because for any t we can switch to a more naïve approach to process all strings of length at most t in $\mathcal{O}(m \log m + mt + N)$ time as described in Lemma 5.1. After applying it with $t = \log^3 m$ in $\mathcal{O}(m \log^3 m + N)$ time, we can set $z = \ell/\log^3 m$ (so that indeed $5/4 \cdot \ell \geq z$ as required in case $s \geq z$) and the overall time complexity for all matrices M and both cases becomes $\mathcal{O}(N + \ell^{\omega-2} \cdot m \log^{5+3(3-\omega)} m)$. Taking the initialization into account we obtain $\mathcal{O}(m \log^5 m + \ell^{\omega-2} \cdot m \log^{5+3(3-\omega)} m + N) = \tilde{\mathcal{O}}(m^{\omega-1} + N)$ total time. ∎

5.2. Type 2 Strings. In this section we show how to solve a restricted instance of the AP problem where every string $S \in \mathcal{S}$ is of type 2, that is, S contains a length-ℓ substring that is not strongly periodic as well as a length-ℓ substring that is strongly periodic, and furthermore $|S| \in [9/8 \cdot \ell, 5/4 \cdot \ell)$ for some $\ell \leq m$.

Similarly as in Section 5.1, we select a set of anchors. In this case, instead of the NodeSelection problem we need to exploit periodicity. We call a string T *ℓ-periodic* if $|T| \geq \ell$ and $\text{per}(T) \leq \ell/4$. We consider all maximal ℓ-periodic substrings of S, that is, ℓ-periodic substrings $S[i\,..\,j]$ such that either $i = 1$ or $\text{per}(S[(i-1)\,..\,j]) > \ell/4$, and $j = |S|$ or $\text{per}(S[i\,..\,(j+1)]) > \ell/4$. We know that S contains at least one such substring (because there exists a length-ℓ substring that is strongly periodic), and that the whole S is not such a substring (because otherwise S would be of type 3). Further, two maximal ℓ-periodic substrings cannot overlap too much, as formalized in the following lemma.

Lemma 5.9. *Any two distinct maximal ℓ-periodic substrings of the same string S overlap by less than $\ell/2$ letters.*

Proof. Assume (by contradiction) the opposite; then we have two distinct ℓ-periodic substrings $S[i\,..\,j]$ and $S[i'\,..\,j']$ such that $i < i' \leq j < j'$ and $j - i' + 1 \geq \ell/2$. Then, both $p = \text{per}(S[i\,..\,j])$ and $p' = \text{per}(S[i'\,..\,j'])$ are periods of $S[i'\,..\,j]$, and hence by Lemma 2.1 we have that $\gcd(p, p')$ is a period of $S[i'\,..\,j]$. If $p \neq p'$ then, because $S[i'\,..\,j]$ contains an occurrence of both $S[i\,..\,(i+p-1)]$ and $S[i'\,..\,(i'+p'-1)]$, we obtain that one of these two substrings is a power of a shorter string, thus contradicting the definition of p or p'. So $p = p'$, but then $p \leq \ell/4$ is actually a period of the whole $S[i\,..\,j']$, meaning that $S[i\,..\,j]$ and $S[i'\,..\,j']$ are not maximal, a contradiction. ∎

By Lemma 5.9, every $S \in \mathcal{S}$ contains exactly one maximal ℓ-periodic substring, and by the same argument P contains $\mathcal{O}(m/\ell)$ such substrings. The set of anchors will be generated by considering the unique maximal ℓ-periodic substring of every $S \in \mathcal{S}$, so we first need to show how to efficiently generate such substrings.

Lemma 5.10. *Given a string S of length at most $5/4 \cdot \ell$, we can generate its (unique) maximal ℓ-periodic substring in $\mathcal{O}(|S|)$ time.*

Proof. We start with observing that any length-ℓ substring of S must contain $S[(\lfloor \ell/2 \rfloor + 1)\,..\,\ell]$ inside. Consequently, we can proceed similarly as in the proof of Lemma 5.4. We compute $p = \text{per}(S[(\lfloor \ell/2 \rfloor + 1)\,..\,\ell])$ in $\mathcal{O}(|S|)$ time. If $p > \ell/4$ then S does not contain any ℓ-periodic substrings. Otherwise, we compute in $\mathcal{O}(|S|)$ time how far the period p extends to the left and to the right; that is, we compute the smallest $i \leq \lfloor \ell/2 \rfloor + 1$ such that $S[k] = S[k+p]$ for every $k = i, i+1, \ldots, \lfloor \ell/2 \rfloor$ and the largest $j \geq \ell$ such that $S[k] = S[k-p]$ for every $k = \ell+1, \ell+2, \ldots, j$. If $j - i + 1 \geq \ell$ then $S[i\,..\,j]$ is a maximal ℓ-periodic substring of S, and, as shown earlier by Lemma 5.9, S cannot contain any other maximal ℓ-periodic substrings. We return

$S[i\,..\,j]$ as the (unique) maximal ℓ-periodic substring of S. ◻

For every $S \in \mathcal{S}$, we apply Lemma 5.10 on S to find its (unique) maximal ℓ-periodic substring $S[i\,..\,j]$ in $\mathcal{O}(|S|)$ time. If $i > 1$ then we designate $S[(i-1)\,..\,(i-1+\ell)]$ as an anchor, and similarly if $j < |S|$ we designate $S[(j+1-\ell)\,..\,(j+1)]$ as an anchor. Observe that because S is of type 2 (and not of type 3) either $i > 1$ or $j < |S|$, so for every $S \in \mathcal{S}$ we designate at least one if its length-$(\ell+1)$ substrings as an anchor. As in Section 5.1, we represent each anchor by one of its occurrences in P, and so need to find its corresponding node in the suffix tree of P (if any). This can be done in $\mathcal{O}(|S|)$ time, so $\mathcal{O}(N)$ overall. During this process we might designate the same string as an anchor multiple times, but we can easily remove the possible duplicates to obtain the set $\mathcal{A}$ of anchors in the end. Then, we generate the occurrences of all anchors in P by accessing their corresponding nodes in the suffix tree of P and iterating over all leaves in their subtrees. We claim that the total number of all these occurrences is only $\mathcal{O}(m/\ell)$. This follows from the following characterization.

LEMMA 5.11. *If $P[x\,..\,(x+\ell)]$ is an occurrence of an anchor then either $P[(x+1)\,..\,y]$ is a maximal ℓ-periodic substring of P, for some $y \geq x+\ell$, or $P[x'\,..\,(x+\ell-1)]$ is a maximal ℓ-periodic substring of P, for some $x' \leq x$.*

Proof. By symmetry, it is enough to consider an anchor H created because of a maximal ℓ-periodic substring $S[i\,..\,j]$ such that $i > 1$, when we add $S[(i-1)\,..\,(i-1+\ell)]$ to $\mathcal{A}$. Thus, $\text{per}(H[2\,..\,|H|]) \leq \ell/4$ and if $P[x\,..\,(x+\ell)] = H$ then $\text{per}(P[(x+1)\,..\,(x+\ell)]) \leq \ell/4$, making $P[(x+1)\,..\,(x+\ell)]$ a substring of some maximal ℓ-periodic substring of $P[(x'+1)\,..\,y]$, where $x' \leq x$ and $y \geq x+\ell$. If $x' < x$ then $\text{per}(H) \leq \ell/4$. But then $H = S[(i-1)\,..\,(i-1+\ell)]$ can be extended to some maximal ℓ-periodic substring $S[i'\,..\,j']$ such that $i' \leq i-1$ and $j' \geq i-1+\ell$. The overlap between $S[i\,..\,j]$ and $S[i'\,..\,j']$ is at least ℓ, so by Lemma 5.9 $i = i'$ and $j = j'$, which is a contradiction. Consequently, $x' = x$ and we obtain the lemma. ◻

By Lemma 5.11, the number of occurrences of all anchors in P is at most two per each maximal ℓ-periodic substrings, so $\mathcal{O}(m/\ell)$ in total. We thus obtain a set of length-$(\ell+1)$ anchors with the following properties:

1. The total number of occurrences of all anchors in P is $\mathcal{O}(m/\ell)$.
2. For every $S \in \mathcal{S}$, at least one of its length-$(\ell+1)$ substrings is an anchor.
3. For every $S \in \mathcal{S}$, at most two of its length-$(\ell+1)$ substrings are anchors.

These properties are even stronger than what we had used in Section 5.1 (except that now we are working with length-$(\ell+1)$ substrings, which is irrelevant), we can now prove our final result also for strings of type 2.

THEOREM 5.12. *An instance of the AP problem where all strings are of type 2 can be solved in $\tilde{\mathcal{O}}(m^{\omega-1}+N)$ time.*

5.3. Type 3 Strings. In this section we show how to solve a restricted instance of the AP problem where every string $S \in \mathcal{S}$ is of type 3, and furthermore $|S| \in [9/8 \cdot \ell, 5/4 \cdot \ell)$ for some $\ell \leq m$. Recall that strings $S \in \mathcal{S}$ are such that every length-ℓ substring of S is strongly periodic and, by Lemma 5.3, in this case, $\text{per}(S) \leq \ell/4$. An occurrence of such S in P must be contained in a maximal ℓ-periodic substring of P. Recall that a string T is called ℓ-periodic if $|T| \geq \ell$ and $\text{per}(T) \leq \ell/4$. For an ℓ-periodic string T, let its *root*, denoted by $\text{root}(T)$, be the lexicographically smallest cyclic shift of $T[1\,..\,\text{per}(T)]$. Because $\text{per}(T) \leq \ell/4$ and $|T| \geq \ell$ by definition, there are at least four repetitions of the period in T, so we can write $T = R[i\,..\,|R|]R^{\alpha}R[1\,..\,j]$, where $R = \text{root}(T)$, for some $i, j \in [1, |R|]$ and $\alpha \geq 2$. It is well known that $\text{root}(T)$

can be computed in $\mathcal{O}(|T|)$ time [31].

EXAMPLE 4. *Let* $T =$ `babababab` *and* $\ell = 8$. *We have* $|T| = 9 \geq \ell = 8$ *and* $\text{per}(T) = 2 \leq \ell/4 = 2$, *so* T *is* ℓ*-periodic. We have* $\text{root}(T) = R =$ `ab`, *and* T *can be written as* $T =$ `b` $\cdot$ (`ab`)$^3 \cdot$ `ab`, *for* $i = 2$ *and* $j = 2$.

We will now make a partition of type 3 strings based on their roots. We start with extracting all maximal ℓ-periodic substrings of P by proceeding similarly as in the proof of Lemma 5.10, and then compute the root of every such substring in $\mathcal{O}(m)$ total time. In more detail, we partition P into blocks of length $\ell/2$, and compute the period of each such block. Any maximal ℓ-periodic substring of P needs to contain at least one such block inside. Therefore, for each block with period at most $\ell/$ we can compute how far its period extends to the left and to the right, and output the corresponding substring if it is long enough. The only difficulty is that we should not extend the period beyond the preceding block. Two maximal ℓ-periodic substrings cannot overlap by more than $\ell/2$ letters, hence their total length is $\mathcal{O}(m)$ and we can compute the root of each such substring in $\mathcal{O}(m)$ total time. We also extract the root of every $S \in \mathcal{S}$ in $\mathcal{O}(N)$ total time. We then partition maximal ℓ-periodic substrings of P and strings $S \in \mathcal{S}$ into groups that have the same root. In the remaining part we describe how to process each such group corresponding to root R in which all maximal ℓ-periodic substrings of P have total length m', and the strings $S \in \mathcal{S}$ have total length N'.

Recall that the bit vector U stores the active prefixes input to the AP problem, and the bit vector V encodes the new active prefixes we aim to compute. For every maximal ℓ-periodic substring of P with root R we extract the corresponding fragment of the bit vector U and need to update the corresponding fragment of the bit vector V. To make the description less cluttered, we assume that each such substring of P is a power of R, that is, R^α for some $\alpha \geq 4$. This can be assumed without loss of generality as it can be ensured by appropriately padding the extracted fragment of U and then truncating the results, while increasing the total length of all considered substrings of P by at most half of their length. In the description below, for simplicity of presentation, U and V denote these padded fragments of the original U and V. When computing V from U we use two different methods for processing the elements $S = R[i \mathinner{.\,.} |R|]R^\beta R[1 \mathinner{.\,.} j]$ of $\mathcal{S}$ depending on their length: either $\beta \geq t$ (large β) or $\beta < t$ (small β), for some parameter t to be chosen later. In both cases, we rely on the observation that $S = R[i \mathinner{.\,.} |R|]R^\beta R[1 \mathinner{.\,.} j]$ occurs R^α at positions $i + \gamma \cdot |R|$, for $\gamma = 0, \dots, \alpha - \beta - 2$. This follows from R being the root and $\beta \geq 1$.

Large β**.** We proceed in phases corresponding to $\beta = t, \dots, \alpha$. In each single phase, we consider all strings $S \in \mathcal{S}$ with $S = R[i \mathinner{.\,.} |R|]R^\beta R[1 \mathinner{.\,.} j]$, for some i and j. Let $C(\beta)$ be the set of the corresponding pairs (i, j), and observe that $\sum_\beta |C(\beta)| \cdot |R^\beta| \leq N'$. This is because the length of R^β is not greater than that of $S = R[i \mathinner{.\,.} |R|]R^\beta R[1 \mathinner{.\,.} j]$, there are $|C(\beta)|$ distinct strings of the latter form in $\mathcal{S}$, and the total length of all $S \in \mathcal{S}$ is N'. The total number of occurrences of a string $S = R[i \mathinner{.\,.} |R|]R^\beta R[1 \mathinner{.\,.} j]$ in R^α is bounded by $\mathcal{O}(\alpha)$, and all such occurrences can be generated in time proportional to their number. Thus, for every $(i, j) \in C(\beta)$, we can generate all occurrences of the corresponding string and appropriately update V in $\mathcal{O}(\alpha \cdot |C(\beta)|)$ total time.

Small β**.** We start by giving a technical lemma on the complexity of multiplying two $r \times r$ matrices whose cells are polynomials of degree up to d.

LEMMA 5.13. *If two* $r \times r$ *matrices over* $\mathbb{Z}$ *can be multiplied in* $\mathcal{O}(r^\omega)$ *time, then two* $r \times r$ *matrices over* $\mathbb{Z}[x]$ *with degrees up to* d *can be multiplied in* $\mathcal{O}(r^\omega d + r^2 d \log d)$

time.

Proof. Let A and B be two $r \times r$ matrices over $\mathbb{Z}[x]$ with degrees up to d. We reduce the product $A \times B = C$ to $(2d+1)$ products of $r \times r$ matrices over $\mathbb{Z}$ as follows. We evaluate the polynomials of each matrix in the complex $(2d+1)$-th roots of unity: let A_i and B_i be the matrices obtained by evaluating the polynomials of A and B in the i-th such root, respectively. We then perform the $2d+1$ products $A_1 \times B_1, \ldots, A_{2d+1} \times B_{2d+1}$ to obtain matrices $C_1, \ldots, C_{2d+1}$: the $2d+1$ values $C_1[i,j], \ldots, C_{2d+1}[i,j]$ are finally interpolated to obtain the coefficient representation of $C[i,j]$, for each $i,j = 1, \ldots, r$, in $\mathcal{O}(d \log d)$ time for each polynomial [27]. Since we perform $2d+1$ products of matrices in $\mathbb{Z}^{r \times r}$, and we evaluate and interpolate r^2 polynomials of degree up to $2d+1$, the overall time complexity is $2d\mathcal{O}(r^\omega) + r^2\mathcal{O}(d \log d) = \mathcal{O}(r^\omega d + r^2 d \log d)$. ◻

Unlike in the large β case, we process $\beta = 2, \ldots, t-1$ simultaneously as follows when $t \geq 3$.

We construct three-dimensional Boolean tables: M with dimensions $|R| \times |R| \times t$ and M' with dimensions $\lceil \alpha/t \rceil \times |R| \times t$. We set $M[i,j,\beta+1] = 1$ if and only if $(i,j) \in C(\beta)$. M can be constructed in time proportional to its size by first precomputing a lexicographically sorted list of triples (β, i, j) corresponding to $S \in \mathcal{S}$ such that $S = R[i\,..\,|R|]R^\beta R[1\,..\,j]$. The lists corresponding to different roots are constructed in $\mathcal{O}(N')$ total time, and we sort them together with radix sort to avoid paying $\mathcal{O}(m)$ per each root. Then, we construct M by considering the prefix of the list consisting of all triples with sufficiently small first coordinates. Next, we set $M'[k,i,\gamma+1] = 1$ if and only if $U[((k-1)t+\gamma)|R|+i-1] = 1$. Finally, we interpret M' and M as matrices over $\mathbb{Z}[x]$ with degrees up to $t-1$, and compute their product $M'' = M' \times M$. That is, we think that $M'[k,i] = \sum_{\gamma=0}^{t-1} M'[k,i,\gamma+1]x^\gamma$ and $M[i,j] = \sum_{\beta=0}^{t-1} M[i,j,\beta+1]x^\beta$, and compute $M''[k,j] = \sum_{i=1}^{|R|} M'[k,i] \cdot M[i,j]$ for every $k = 1, \ldots, \lceil \alpha/t \rceil$ and $j = 1, \ldots, |R|$ (this will be eventually implemented with Lemma 5.13). Note that each $M''[k,j]$ is a polynomial with degree up to $2(t-1)$. We claim that this allows us to recover the updates to V by setting $V[((k-1)t+q+1)|R|+j] = 1$ whenever x^q appears with non-zero coefficient in the polynomial at $M''[k,j]$, for all $k = 1, \ldots, \lceil \alpha/t \rceil$, $j = 1, \ldots, |R|$ and $q = 0, \ldots, 2(t-1)$. Equivalently, we set $V[((k-1)t+\gamma+\beta+1)|R|+j] = 1$ whenever $M'[k,i,\gamma+1] = 1$ and $M[i,j,\beta+1] = 1$, for all $k = 1, \ldots, \lceil \alpha/t \rceil$, $i,j = 1, \ldots, |R|$ and $\gamma, \beta = 0, \ldots, t-1$. This can be rewritten as setting $V[((k-1)t+\gamma+\beta+1)|R|+j] = 1$ whenever $U[((k-1)t+\gamma)|R|+i-1] = 1$ and there exists $S \in \mathcal{S}$ such that $S = R[i\,..\,|R|]R^\beta R[1\,..\,j]$, for all $k = 1, \ldots, \lceil \alpha/t \rceil$, $j = 1, \ldots, |R|$ and $\gamma, \beta = 0, \ldots, t-1$, which is indeed correct as any $x \in \{0, \ldots, \alpha-1\}$ can be written as $x = (k-1)t + \gamma$ for $k \in \{1, \ldots, \lceil \alpha/t \rceil\}$ and $\gamma \in \{0, \ldots, t-1\}$.

We are now in a position to prove the following result for type 3 strings.

THEOREM 5.14. *An instance of the AP problem where all strings are of type 3 can be solved in* $\tilde{\mathcal{O}}(m^{\omega-1} + N)$ *time.*

Proof. Recall that we consider strings S of type 3 with root R and substrings of P with root R together. We first analyze the time to process a single group containing a number of substrings of P of total length m' and a number of strings $S \in \mathcal{S}$ of total length N'. Let us denote by R^{α_h} the h-th considered substring of P and by t_h the value of t used to distinguish between small and large value of β when processing this substring. We partition all substrings into $\log m$ levels, with the k-th level G_k containing h such that $\alpha_h \in [2^k, 2^{k+1})$. We define $\bar{\alpha}_k = \sum_{h \in G_k} \alpha_k$ and choose $t_h = \min(2^{k+1}, \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil)$ for every $h \in G_k$.

For each level k, $h \in G_k$ and $\beta = t_h, \ldots, \alpha_h$, we use the first method and spend $\mathcal{O}(\alpha_h \cdot |C(\beta)|)$ time, where $C(\beta)$ is the set of (i, j) for this specific β. This needs to be done only when $t_h \leq \alpha_h$, that is, $t_h = \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil$. The overall time used for all applications of the first method is thus at most:

$$\begin{aligned}
\sum_k \sum_{h \in G_k} \mathcal{O}(\alpha_h \cdot \sum_{\beta \geq t_h} |C(\beta)|) &= \mathcal{O}(\sum_k \sum_{h \in G_k} \alpha_h / |R^{t_h}| \sum_{\beta \geq t_h} |C(\beta)| \cdot |R^{t_h}|) \\
&= \mathcal{O}(\sum_k \sum_{h \in G_k} a_h / (|R| \cdot t_h) \sum_{\beta \geq t_h} |C(\beta)| \cdot |R^{\beta}|) \\
&= \mathcal{O}(\sum_k \bar{a}_k / (|R| \cdot \bar{\alpha}_k/|R| \cdot \log m) \cdot N') = \mathcal{O}(N'),
\end{aligned}$$

using the fact that $\sum_\beta |C(\beta)| \cdot |R^\beta| \leq N'$ and there are $\log m$ values of k.

For each level k and $h \in G_k$, we process together all $\beta = 2, \ldots, t_h - 1$ using the second method. This requires multiplying two matrices of polynomials of degree up to $t_h - 1$. We observe that the second matrix is in fact the same for all $h \in G_k$, and so we denote the first matrix by M'_h, the second by simply M, and think that the degree of each polynomial in M'_h and M is strictly upper bounded by $d_k = \min(2^{k+1}, \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil)$. M'_h is of size $\lceil \alpha_h / d_k \rceil \times |R|$ while M is of size $|R| \times |R|$. Instead of computing each product $M'_h \times M$ separately, we vertically concatenate all matrices M'_h to obtain a single matrix M'. The number of rows in M' is $r = \sum_{h \in G_k} \lceil \alpha_h / d_k \rceil$. Next, we compute $M' \times M$ with $\lceil r/|R| \rceil$ invocations of Lemma 5.13. We separately analyse the overall time complexity for $d_k = 2^{k+1}$ and $d_k = \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil$.

$d_k = \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil$: Using $\alpha_h \geq 2^k \geq d_k/2$ we bound r as follows:

$$\begin{aligned}
r = \sum_{h \in G_k} \lceil \alpha_h / d_k \rceil &\leq \sum_{h \in G_k} (\alpha_h + d_k)/d_k \leq \sum_{h \in G_k} (\alpha_h + 2\alpha_h)/d_k \\
&\leq 3 \sum_{h \in G_k} \alpha_h / (\bar{\alpha}_k/|R| \cdot \log m) = 3|R|/\log m \leq |R|,
\end{aligned}$$

for sufficiently large m. Thus, one invocation suffices and takes time

$$\mathcal{O}(|R|^\omega d_k + |R|^2 d_k \log d_k) = \mathcal{O}(|R|^{\omega-1} \bar{\alpha}_k \log^2 m)$$

using $d_k \geq 3$ and $d_k \leq 2m$.

$d_k = 2^{k+1}$: Because $\alpha_h \in [2^k, 2^{k+1})$ for each $h \in G_k$, we have $r = |G_k| \leq \bar{\alpha}_k/2^k$. The number of invocations is thus at most $\lceil \bar{\alpha}_k/(2^k \cdot |R|) \rceil \leq \bar{\alpha}_k/(2^k \cdot |R|) + 1$. The total time used by all these invocations is

$$\begin{aligned}
&(\bar{\alpha}_k/(2^k \cdot |R|) + 1)\mathcal{O}(|R|^\omega 2^{k+1} + |R|^2 2^{k+1}(k+1)) \\
&= \mathcal{O}(|R|^{\omega-1} \bar{\alpha}_k \log m + |R|^\omega 2^{k+1} \log m)
\end{aligned}$$

using $2^{k+1} \leq 2m$. Next, because $2^{k+1} \leq \lceil \bar{\alpha}_k/|R| \cdot \log m \rceil$ and $2^{k+1} \geq 2$ we have $2^{k+1} \leq 2\bar{\alpha}_k/|R| \cdot \log m$, so the total time can be further bounded by

$$\begin{aligned}
&\mathcal{O}(|R|^{\omega-1} \bar{\alpha}_k \log m + |R|^\omega 2^{k+1} \log m) \\
&= \mathcal{O}(|R|^{\omega-1} \bar{\alpha}_k \log m + |R|^\omega (\bar{\alpha}_k/|R| \cdot \log m) \log m) \\
&= \mathcal{O}(|R|^{\omega-1} \bar{\alpha}_k \log^2 m).
\end{aligned}$$

Hence in both cases the time used by all multiplications is $\mathcal{O}(|R|^{\omega-1}\bar{\alpha}_k \log^2 m)$. Using $\sum_k \bar{\alpha_k} = m'/|R|$ and $|R| \le m'$, when summed over all $\log m$ levels k this is in fact $\mathcal{O}((m')^{\omega-1}\log^2 m)$. We remark that the matrix M can be built in time proportional to its size assuming $\mathcal{O}(N')$ preprocessing, while the matrix M' can be built in time proportional to its size by just scanning over the corresponding fragment of U.

Finally, summing possibly many groups corresponding to different roots R, because all values of N' sum up to N and all values of m' sum up to $\mathcal{O}(m)$, by convexity of $x^{\omega-1}$ we obtain that the overall time complexity including the preprocessing is $\tilde{\mathcal{O}}(m^{\omega-1} + N)$. ∎

5.4. Wrapping Up. In Sections 5.1, 5.2 and 5.3 we design three $\tilde{\mathcal{O}}(m^{\omega-1}+N)$-time algorithms for an instance of the AP problem where all strings are of type 1, 2 and 3, respectively. Summing up over all values of k and all the types, we thus obtain Theorem 1.2. In every case, the complexity is actually $\tilde{\mathcal{O}}(nm^{\omega-1})+\mathcal{O}(N)$, so using the fact that $\omega < 2.373$ [51, 66] we can hide the polylog factors and obtain the following corollary.

COROLLARY 5.15. *The EDSM problem can be solved on-line in $\mathcal{O}(nm^{1.373} + N)$ time.*

6. Final Remarks.

Our contribution in this paper is twofold. First, we designed an appropriate reduction showing that a combinatorial algorithm solving the EDSM problem in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$, refutes the well-known BMM conjecture. Second, we designed a non-combinatorial $\tilde{\mathcal{O}}(nm^{\omega-1} + N)$ -time algorithm to attack the same problem. By using the fact that $\omega < 2.373$, our algorithm runs in $\mathcal{O}(nm^{1.373} + N)$ time thus circumventing the combinatorial conditional lower bound for the EDSM problem. Let us point out that if $\omega = 2$ then our algorithm for the AP problem is time-optimal up to polylog factors, as any algorithm needs to read the input. As for the EDSM problem, such an argument only shows a lower bound of $\Omega(N)$. However, at the same time we can show that there is no $\mathcal{O}((nm)^{1-\epsilon})$-time algorithm, assuming the Strong Exponential Time Hypothesis (SETH) [19], by the following argument. By prepending and appending a unique letter to both the ED text and the pattern, we can reduce checking membership for a regular expression of type $\cdot|\cdot$, as defined by Backurs and Indyk [10]. Combining this with their reduction from SETH, we immediately obtain the claimed conditional lower bound for the EDSM problem.

We finally remark that, if we use the simple cubic-time matrix multiplication algorithm in our solution then the total time complexity becomes $\tilde{\mathcal{O}}(nm^{\omega-1} + N) = \tilde{\mathcal{O}}(nm^2 + N)$. At the same time, the solution by Aoyama et al. [8], which also does not use fast matrix multiplication, runs in time $\mathcal{O}(nm^{1.5} + N)$. It is thus plausible that one could obtain an $\tilde{\mathcal{O}}(nm^{\omega/2} + N)$-time algorithm for the EDSM problem. We leave this question open for future work.

REFERENCES

[1] A. Abboud, A. Backurs, and V. Williams, *If the current clique algorithms are optimal, so is Valiant's parser*, in 56th IEEE Symposium on Foundations Of Computer Science (FOCS), 2015, pp. 98–117.
[2] A. Abboud and V. Williams, *Popular conjectures imply strong lower bounds for dynamic problems*, in 55th IEEE Symposium on Foundations Of Computer Science (FOCS), 2014, pp. 434–443.
[3] K. Abrahamson, *Generalized string matching*, SIAM J. Comput., 16 (1987), pp. 1039–1051.

[4] J. Alman and V. V. Williams, *A refined laser method and faster matrix multiplication*, in Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), 2021, pp. 522–539.

[5] M. Alzamel, L. Ayad, G. Bernardini, R. Grossi, C. Iliopoulos, N. Pisanti, S. Pissis, and G. Rosone, *Degenerate string comparison and applications*, in 18th Workshop on Algorithms in Bioinformatics (WABI), vol. 113 of LIPIcs, 2018, pp. 21:1–21:14.

[6] M. Alzamel, L. A. K. Ayad, G. Bernardini, R. Grossi, C. S. Iliopoulos, N. Pisanti, S. P. Pissis, and G. Rosone, *Comparing degenerate strings*, Fundam. Informaticae, 175 (2020), pp. 41–58.

[7] A. Amir, M. Lewenstein, and E. Porat, *Faster algorithms for string matching with k mismatches*, J. Algorithms, 50 (2004), pp. 257–275.

[8] K. Aoyama, Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda, *Faster online elastic degenerate string matching*, in 29th Symposium on Combinatorial Pattern Matching (CPM), vol. 105 of LIPIcs, 2018, pp. 9:1–9:10.

[9] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradžev, *On economical construction of the transitive closure of a directed graph*, Soviet Mathematics Doklady, 11 (1970), pp. 1209–1210.

[10] A. Backurs and P. Indyk, *Which regular expression patterns are hard to match?*, in 57th IEEE Symposium on Foundations Of Computer Science (FOCS), 2016, pp. 457–466.

[11] N. Bansal and R. Williams, *Regularity lemmas and combinatorial algorithms*, in 50th IEEE Symposium on Foundations Of Computer Science (FOCS), 2009, pp. 745–754.

[12] M. Bender and M. Farach-Colton, *The LCA problem revisited*, in 4th Latin American symposium on Theoretical INformatics (LATIN), vol. 1776 of Springer LNCS, 2000, pp. 88–94.

[13] J. Bentley, *Multidimensional binary search trees used for associative searching*, Commun. ACM, 18 (1975), pp. 509–517.

[14] G. Bernardini, P. Gawrychowski, N. Pisanti, S. P. Pissis, and G. Rosone, *Even Faster Elastic-Degenerate String Matching via Fast Matrix Multiplication*, in 46th International Colloquium on Automata, Languages, and Programming (ICALP), vol. 132 of LIPIcs, 2019, pp. 21:1–21:15.

[15] G. Bernardini, N. Pisanti, S. Pissis, and G. Rosone, *Pattern matching on elastic-degenerate text with errors*, in 24th International Symposium on String Processing and Information Retrieval (SPIRE), 2017, pp. 74–90.

[16] G. Bernardini, N. Pisanti, S. P. Pissis, and G. Rosone, *Approximate pattern matching on elastic-degenerate text*, Theor. Comput. Sci., 812 (2020), pp. 109–122.

[17] K. Bringmann, F. Grandoni, B. Saha, and V. Williams, *Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product*, in 56th IEEE Symposium on Foundations Of Computer Science (FOCS), 2016, pp. 375–384.

[18] K. Bringmann, A. Grønlund, and K. Larsen, *A dichotomy for regular expression membership testing*, in 58th IEEE Symposium on Foundations Of Computer Science (FOCS), 2017, pp. 307–318.

[19] C. Calabro, R. Impagliazzo, and R. Paturi, *The complexity of satisfiability of small depth circuits*, in Parameterized and Exact Computation, 4th International Workshop (IWPEC), vol. 5917 of Lecture Notes in Computer Science, 2009, pp. 75–85.

[20] T. Chan, *Speeding up the four Russians algorithm by about one more logarithmic factor*, in 26th ACM-SIAM Symposium On Discrete Algorithms (SODA), 2015, pp. 212–217.

[21] Y.-J. Chang, *Hardness of RNA folding problem with four symbols*, in 27th Symposium on Combinatorial Pattern Matching (CPM), vol. 54 of LIPIcs, 2016, pp. 13:1–13:12.

[22] K. Chatterjee, B. Choudhary, and A. Pavlogiannis, *Optimal Dyck reachability for data-dependence and alias analysis*, in 45th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL), 2018, pp. 30:1–30:30.

[23] A. Cislak and S. Grabowski, *Sopang 2: online searching over a pan-genome without false positives*, CoRR, abs/2004.03033 (2020).

[24] A. Cisłak, S. Grabowski, and J. Holub, *SOPanG: online text searching over a pan-genome*, Bioinformatics, 34 (2018), pp. 4290–4292.

[25] P. Clifford and R. Clifford, *Simple deterministic wildcard matching*, Inf. Process. Lett., 101 (2007), pp. 53–54.

[26] R. Cole and R. Hariharan, *Verifying candidate matches in sparse and wildcard matching*, in 34th ACM Symposium on Theory Of Computing (STOC), 2002, pp. 592–601.

[27] J. Cooley and J. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of Computation, 19 (1965), pp. 297–301.

[28] M. Crochemore, C. Hancart, and T. Lecroq, *Algorithms on strings*, Cambridge University

Press, 2007.

[29] M. Crochemore and D. Perrin, *Two-way string matching*, J. ACM, 38 (1991), pp. 651–675.

[30] A. Czumaj and A. Lingas, *Finding a heaviest vertex-weighted triangle is not harder than matrix multiplication*, SIAM J. Comput., 39 (2009), pp. 431–444.

[31] J. Duval, *Factorizing words over an ordered alphabet*, J. Algorithms, 4 (1983), pp. 363–381.

[32] M. Farach-Colton and S. Muthukrishnan, *Perfect hashing for strings: formalization and algorithms*, in 7th Annual Symposium on Combinatorial Pattern Matching (CPM), vol. 1075 of Springer LNCS, 1996, pp. 130–140.

[33] N. Fine and H. Wilf, *Uniqueness theorems for periodic functions*, Proceedings of the American Mathematical Society, 16 (1965), pp. 109–114.

[34] M. Fischer and A. Meyer, *Boolean matrix multiplication and transitive closure*, in 12th IEEE Symposium on Switching and Automata Theory (SWAT/FOCS), 1971, pp. 129–131.

[35] M. Fischer and M. Paterson, *String matching and other products*, SIAM-AMS Proceedings, 7 (1974), pp. 113–125. Complexity of Computation.

[36] M. Furman, *Application of a method of fast multiplication of matrices in the problem of finding the transitive closure of a graph*, Soviet Mathematics Doklady, 11 (1970), p. 1252.

[37] P. Gawrychowski, S. Ghazawi, and G. M. Landau, *On indeterminate strings matching*, in 31st Symposium on Combinatorial Pattern Matching (CPM), vol. 161 of LIPIcs, 2020, pp. 23:1–23:12.

[38] P. Gawrychowski and P. Uznanski, *Towards unified approximate pattern matching for Hamming and L_1 distance*, in 45th International Colloquium on Automata, Languages and Programming (ICALP), vol. 107 of LIPIcs, 2018, pp. 62:1–62:13.

[39] R. Grossi, C. Iliopoulos, C. Liu, N. Pisanti, S. Pissis, A. Retha, G. Rosone, F. Vayani, and L. Versari, *On-line pattern matching on similar texts*, in 28th Symposium on Combinatorial Pattern Matching (CPM), vol. 78 of LIPIcs, 2017, pp. 9:1–9:14.

[40] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, *Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture*, in 47th ACM Symposium on Theory Of Computing (STOC), 2015, pp. 21–30.

[41] J. Holub, W. Smyth, and S. Wang, *Fast pattern-matching on indeterminate strings*, J. Discrete Algorithms, 6 (2008), pp. 37–50.

[42] C. Iliopoulos, R. Kundu, and S. Pissis, *Efficient pattern matching in elastic-degenerate texts*, in 11th International Conference on Language and Automata Theory and Applications (LATA), vol. 10168 of Springer LNCS, 2017, pp. 131–142.

[43] P. Indyk, *Faster algorithms for string matching problems: Matching the convolution bound*, in 39th Symposium on Foundations Of Computer Science (FOCS), 1998, pp. 166–173.

[44] A. Itai and M. Rodeh, *Finding a minimum circuit in a graph*, in 9th ACM Symposium on Theory Of Computing (STOC), 1977, pp. 1–10.

[45] IUPAC-IUB Commission on Biochemical Nomenclature, *Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents*, Biochemistry, 9 (1970), pp. 4022–4027.

[46] A. Kalai, *Efficient pattern-matching with don't cares*, in 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 655–656.

[47] D. Knuth, J. M. Jr., and V. Pratt, *Fast pattern matching in strings*, SIAM J. Comput., 6 (1977), pp. 323–350.

[48] T. Kociumaka, J. Radoszewski, W. Rytter, and T. Wale, *Internal pattern matching queries in a text and applications*, in 26th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2015, pp. 532–551.

[49] T. Kopelowitz and R. Krauthgamer, *Color-distance oracles and snippets*, in 27th Symposium on Combinatorial Pattern Matching (CPM), vol. 54 of LIPIcs, 2016, pp. 24:1–24:10.

[50] K. Larsen, I. Munro, J. Nielsen, and S. Thankachan, *On hardness of several string indexing problems*, Theor. Comput. Sci., 582 (2015), pp. 74–82.

[51] F. Le Gall, *Powers of tensors and fast matrix multiplication*, in 39th International Symposium on Symbolic and Algebraic Computation (ISSAC), 2014, pp. 296–303.

[52] L. Lee, *Fast context-free grammar parsing requires fast boolean matrix multiplication*, J. ACM, 49 (2002), pp. 1–15.

[53] V. Mäkinen, B. Cazaux, M. Equi, T. Norri, and A. I. Tomescu, *Linear time construction of indexable founder block graphs*, in 20th Workshop on Algorithms in Bioinformatics (WABI), vol. 172 of LIPIcs, 2020, pp. 7:1–7:18.

[54] J. Matoušek, *Computing dominances in E^n*, Inf. Process. Lett., 38 (1991), pp. 277–278.

[55] I. Munro, *Efficient determination of the transitive closure of a directed graph*, Inf. Process. Lett., 1 (1971), pp. 56–58.

[56] G. Navarro, *Nr-grep: a fast and flexible pattern-matching tool*, Softw., Pract. Exper., 31

(2001), pp. 1265–1312.

[57] P. Peterlongo, N. Pisanti, F. Boyer, A. P. do Lago, and M. Sagot, *Lossless filter for multiple repetitions with hamming distance*, J. Discrete Algorithms, 6 (2008), pp. 497–509.

[58] N. Pisanti, *On-line pattern matching on D-texts (invited talk)*, in Proceedings of the 32nd Annual Symposium on Combinatorial Pattern Matching (CPM), vol. 191 of LIPIcs, 2021, pp. 3:1–3:2.

[59] S. Pissis and A. Retha, *Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line*, in 17th International Symposium on Experimental Algorithms (SEA), vol. 103 of LIPIcs, 2018, pp. 16:1–16:14.

[60] L. Roditty and U. Zwick, *On dynamic shortest paths problems*, in 12th European Symposium on Algorithms (ESA), vol. 3221 of Springer LNCS, 2004, pp. 580–591.

[61] M. Ružić, *Constructing efficient dictionaries in close to sorting time*, in 35th International Colloquium on Automata, Languages and Programming (ICALP), vol. 5125 of Springer LNCS, 2008, pp. 84–95.

[62] D. Sleator and R. Tarjan, *A data structure for dynamic trees*, J. Comput. Syst. Sci., 26 (1983), pp. 362–391.

[63] The Computational Pan-Genomics Consortium, *Computational pan-genomics: status, promises and challenges*, Briefings in Bioinformatics, 19 (2018), pp. 118–135.

[64] L. Valiant, *General context-free recognition in less than cubic time*, J. Comput. Syst. Sci., 10 (1975), pp. 308–315.

[65] P. Weiner, *Linear pattern matching algorithms*, in 14th IEEE Annual Symposium on Switching and Automata Theory (SWAT/FOCS), 1973, pp. 1–11.

[66] V. Williams, *Multiplying matrices faster than Coppersmith-Winograd*, in 44th ACM Symposium on Theory Of Computing Conference (STOC), 2012, pp. 887–898.

[67] V. Williams and R. Williams, *Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications*, in 38th ACM Symposium on Theory Of Computing Conference (STOC), 2006, pp. 225–231.

[68] V. Williams and R. Williams, *Subcubic equivalences between path, matrix and triangle problems*, in 51st IEEE Symposium on Foundations Of Computer Science (FOCS), 2010, pp. 645–654.

[69] S. Wu and U. Manber, *Agrep – a fast approximate pattern-matching tool*, in USENIX Technical Conference, 1992, pp. 153–162.

[70] H. Yu, *An improved combinatorial algorithm for boolean matrix multiplication*, in 42nd International Colloquium on Automata, Languages, and Programming (ICALP), vol. 9134 of Springer LNCS, 2015, pp. 1094–1105.

[71] H. Yu, *An improved combinatorial algorithm for boolean matrix multiplication*, Inf. Comput., 261 (2018), pp. 240–247.

[72] U. Zwick, *All pairs shortest paths using bridging sets and rectangular matrix multiplication*, J. ACM, 49 (2002), pp. 289–317.