Elastic-Degenerate String Matching with 1 Error*

Giulia Bernardini[1], Esteban Gabory[2], Solon P. Pissis[2,3,4], Leen Stougie[2,3,4], Michelle Sweering[2], and Wiktor Zuba[2]

[1]University of Trieste, Trieste, Italy
[2]CWI, Amsterdam, The Netherlands
[3]Vrije Universiteit, Amsterdam, The Netherlands
[4]INRIA-Erable, France

September 5, 2022

Abstract

An elastic-degenerate (ED) string is a sequence of n finite sets of strings of total length N, introduced to represent a set of related DNA sequences, also known as a *pangenome*. The ED string matching (EDSM) problem consists in reporting all occurrences of a pattern of length m in an ED text. The EDSM problem has recently received some attention by the combinatorial pattern matching community, culminating in an $\tilde{\mathcal{O}}(nm^{\omega-1}) + \mathcal{O}(N)$-time algorithm [Bernardini et al., SIAM J. Comput. 2022], where ω denotes the matrix multiplication exponent and the $\tilde{\mathcal{O}}(\cdot)$ notation suppresses polylog factors. In the k-EDSM problem, the approximate version of EDSM, we are asked to report all pattern occurrences with at most k errors. k-EDSM can be solved in $\mathcal{O}(k^2mG + kN)$ time, under edit distance, or $\mathcal{O}(kmG + kN)$ time, under Hamming distance, where G denotes the total number of strings in the ED text [Bernardini et al., Theor. Comput. Sci. 2020]. Unfortunately, G is only bounded by N, and so even for $k = 1$, the existing algorithms run in $\Omega(mN)$ time in the worst case. In this paper we make progress in this direction. We show that 1-EDSM can be solved in $\mathcal{O}((nm^2 + N)\log m)$ or $\mathcal{O}(nm^3 + N)$ time under edit distance. For the decision version of the problem, we present a faster $\mathcal{O}(nm^2\sqrt{\log m} + N\log\log m)$-time algorithm. We also show that 1-EDSM can be solved in $\mathcal{O}(nm^2 + N\log m)$ time under Hamming distance. Our algorithms for edit distance rely on non-trivial reductions from 1-EDSM to special instances of classic computational geometry problems (2d rectangle stabbing or 2d range emptiness), which we show how to solve efficiently. In order to obtain an even faster algorithm for Hamming distance, we rely on employing and adapting the k-errata trees for indexing with errors [Cole et al., STOC 2004].

1 Introduction

String matching (or pattern matching) is a fundamental task in computer science, for which several linear-time algorithms are known [19]. It consists in finding all occurrences of a short string, known as the *pattern*, in a longer string, known as the *text*. Many representations have been introduced over the years to account for unknown or uncertain letters in the pattern or in the text, a phenomenon that often occurs in real data. In the context of computational biology, for example, the IUPAC notation [29] is used to represent locations of a DNA sequence for which several alternative nucleotides are possible. Such a notation can encode the consensus of a population of DNA sequences [39, 1, 2, 24] in a gapless multiple sequence alignment (MSA).

*The work in this paper is supported in part by: the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 “Optimization for and with Machine Learning (OPTIMAL)” and Gravitation-grant NETWORKS-024.002.003; the PANGAIA and ALPACA projects that have received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively; and the MUR - FSE REACT EU - PON R&I 2014-2020.

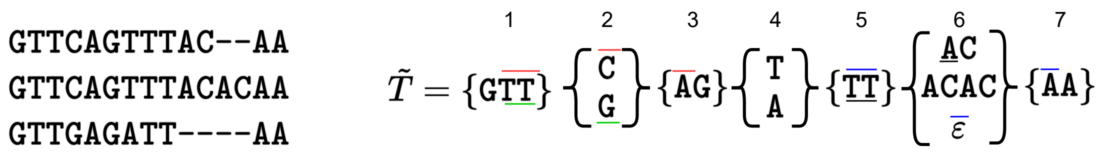

GTTCAGTTTAC--AA
GTTCAGTTTACACAA
GTTGAGATT----AA
1
2
3
4
5
6
7
$\tilde{T}$ = {GTT} {C, G} {AG} {T, A} {TT} {AC, ACAC, ε} {AA}

Figure 1: An MSA of three sequences and its (non-unique) representation $\tilde{T}$ as an ED string of length $n = 7$ and size $N = 20$. The only two *exact* occurrences of $P =$ TTA in $\tilde{T}$ end at positions 6 (black underline) and 7 (blue overline); a *1-mismatch* occurrence of P in $\tilde{T}$ ends at position 2 (green underline); and a *1-error* occurrence of P in $\tilde{T}$ ends at position 3 (red overline). Note that other 1-error and 1-mismatch occurrences of P in $\tilde{T}$ exist (e.g., ending at positions 1 and 5).

EDSM	**Features**	**Running time**
Grossi et al. [27]	Combinatorial	$\mathcal{O}(nm^2 + N)$
Aoyama et al. [5]	Fast Fourier transform	$\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$
Bernardini et al. [7]	Fast matrix multiplication	$\mathcal{O}(nm^{1.381} + N)$
Bernardini et al. [8]	Fast matrix multiplication	$\tilde{\mathcal{O}}(nm^{\omega-1}) + \mathcal{O}(N)$

Table 1: The upper-bound landscape of the EDSM problem. The term "combinatorial" is arguably not well-defined; lower bounds conditioned on Boolean Matrix Multiplication often indicate that other techniques, including fast matrix multiplication, may be employed to obtain improved bounds for a specific problem. This is the case for EDSM.

Iliopoulos et al. generalized these representations in [28] to also encode insertions and deletions (gaps) occurring in MSAs by introducing the notion of elastic-degenerate strings. An *elastic-degenerate* (ED) string $\tilde{T}$ over an alphabet Σ is a sequence of finite subsets of Σ^* (which includes the empty string ε), called *segments*. The total number of segments is the *length* of the ED string, denoted by $n = |\tilde{T}|$; and the total number of letters (including symbol ε) in all segments is the *size* of the ED string, denoted by $N = \|\tilde{T}\|$. Inspect Figure 1 for an example.

A natural problem is to find all occurrences of a standard (non-degenerate) pattern P in an ED text $\tilde{T}$, called the ED string matching (EDSM) problem in the literature. After the simple polynomial-time algorithm proposed by Iliopoulos et al. [28], a series of results have been published for EDSM. The results for EDSM summarized in Table 1 have a *linear dependency* on the size N of the ED text, a highly desirable property. (A different line of research exists, which waives the linear-dependency restriction, and employs bit-vector techniques to speed up the computation specifically for short patterns [27, 34, 16].) In Table 1, m is the length of the pattern, n is the length of the ED text, N is its size, and ω is the matrix multiplication exponent. These algorithms are also *on-line*: the ED text is read segment-by-segment and occurrences are reported as soon as the last segment they overlap is processed. Grossi et al. [27] presented an $\mathcal{O}(nm^2+N)$-time algorithm for EDSM. This was later improved by Aoyama et al. [5], who employed fast Fourier transform to improve the time complexity of EDSM to $\mathcal{O}(nm^{1.5}\sqrt{\log m} + N)$. Bernardini et al. [7] then presented a lower bound conditioned on Boolean Matrix Multiplication suggesting that it is unlikely to solve EDSM by a combinatorial algorithm in $\mathcal{O}(nm^{1.5-\epsilon} + N)$ time, for any $\epsilon > 0$. This was an indication that fast matrix multiplication may improve the time complexity of EDSM. Indeed, Bernardini et al. [7] presented an $\mathcal{O}(nm^{1.381} + N)$-time algorithm, which they subsequently improved to an $\tilde{\mathcal{O}}(nm^{\omega-1}) + \mathcal{O}(N)$-time algorithm [8], both using fast matrix multiplication, thus breaking through the conditional lower bound for EDSM.

Our Results and Techniques In string matching, a single extra or missing letter in the pattern or in a potential occurrence results in missing (many or all) occurrences. Hence, many works are focused on approximate string matching for standard strings [30, 31, 18, 4, 25, 13]. For approximate k-EDSM, Bernardini et al. [9] presented an on-line $\mathcal{O}(k^2mG + kN)$-time algorithm under edit distance and an on-line $\mathcal{O}(kmG + kN)$-time algorithm under Hamming distance, where k is the maximum allowed number of errors (edits) or mismatches, respectively, and G is the total number of strings in all segments. Unfortunately, G is only bounded by N, and so even for $k = 1$, the existing algorithms run in $\Omega(mN)$ time in the worst case.

Approximate EDSM	**Features**	**Running time**
Bernardini et al. [9]	k errors	$\mathcal{O}(k^2mG + kN)$
This work	1 error	$\mathcal{O}(nm^3 + N)$
This work	1 error	$\mathcal{O}((nm^2 + N)\log m)$
This work	1 error (decision)	$\mathcal{O}(nm^2\sqrt{\log m} + N \log\log m)$
Bernardini et al. [9]	k mismatches	$\mathcal{O}(kmG + kN)$
This work	1 mismatch	$\mathcal{O}(nm^3 + N)$
This work	1 mismatch	$\mathcal{O}(nm^2 + N\log m)$

Table 2: The state of the art results for approximate EDSM and our new results for $k = 1$. Note that $n \leq G \leq N$. All algorithms underlying these results are combinatorial and all the reporting algorithms are on-line.

Let us remark that the special case of $k = 1$ is not interesting for approximate string matching on standard strings: the existing algorithms have a polynomial dependency on k and a linear dependency on the length n of the text, and thus for $k = 1$ we trivially obtain $\mathcal{O}(n)$-time algorithms under edit or Hamming distance. However, this is not the case for other string problems, such as text indexing with errors, where the first step was to design a data structure for 1 error [3]. The next step, extending it to k errors, required the development of new highly non-trivial techniques and incurred some exponential factor with respect to k [17]. Interestingly, k-EDSM seems to be the same case, which highlights the main theoretical motivation of this paper. In Table 2, we summarize the state of the art for approximate EDSM and our new results for $k = 1$. Note that the reporting algorithms underlying our results are also *on-line*.

Indeed, to arrive at our main results, we design a rich non-trivial combination of algorithmic techniques. Our algorithms for edit distance rely on non-trivial reductions from 1-EDSM to special instances of classic computational geometry problems (2d rectangle stabbing or 2d range emptiness), which we show how to solve efficiently. In order to obtain an even faster algorithm for Hamming distance, we also rely on employing and adapting the k-errata trees of Cole et al. for text indexing with k errors [17].

The combinatorial algorithms we develop here for approximate EDSM are good in the following sense. First, the running times of our algorithms do not depend on G, a highly desirable property. Specifically, all of our results replace $m \cdot G$ by an $n \cdot \text{poly}(m)$ factor. Second, our $\tilde{\mathcal{O}}(nm^2 + N)$-time algorithms are at most one $\log m$ factor slower than $\mathcal{O}(nm^2 + N)$, the best-known bound obtained by a combinatorial algorithm (not employing fast Fourier transforms) for *exact* EDSM [27]. Notably, for Hamming distance, we show an $\mathcal{O}(nm^2 + N\log m)$-time algorithm. Last, our $\mathcal{O}(nm^3 + N)$-time algorithms have a linear dependency on N, another highly desirable property (at the expense of an extra m-factor).

Other Related Work The main motivation to consider ED strings is that they can be used to represent a *pangenome*: a collection of closely-related genomic sequences that are meant to be analyzed together [39]. Several other pangenome representations have been proposed in the literature, mostly graph-based ones; see [10] for a comprehensive overview by Carletti et al. Compared to these graph-based representations, ED strings have at least two main advantages in the context of string matching, as they support: (i) simple on-line string matching; and (ii) (deterministic) subquadratic in m string matching [5, 7, 8].

Similar in spirit to ED strings, and to the restricted notion of *generalized degenerate* strings, in which strings of different lengths cannot be in the same segment [1, 2], is the representation of pangenomes via *founder graphs*. The idea behind founder graphs is that a multiple alignment of few *founder sequences* can be used to approximate the input MSA, with the feature that each row of the MSA is a recombination of the founders. Unlike ED strings, that are believed not to be efficiently indexable [26] (and indeed their value is to enable fast on-line string matching algorithms), some subclasses of founder graphs are, and a recent line of research is devoted to constructing and indexing such structures [32, 20]. Like founder graphs, ED strings support the recombination of different rows of the MSA between consecutive columns.

Paper Organization In Section 2, we provide the necessary definitions and notation, we describe the basic layout of the developed algorithms, and we formally state our main results. In Section 3, we present our solutions under edit distance; and in Section 4, we present our improvement for the special case of Hamming distance. In Section 5, we conclude this work with some basic open questions for future work.

2 Preliminaries

We start with some basic definitions and notation following [19]. Let $X = X[1] \dots X[n]$ be a *string* of length $|X| = n$ over an ordered alphabet Σ whose elements are called *letters*. The *empty string* is the string of length 0; we denote it by ε. For any two positions i and $j \geq i$ of X, $X[i \,.\,.\, j]$ is the *fragment* of X starting at position i and ending at position j. The fragment $X[i \,.\,.\, j]$ is an *occurrence* of the underlying *substring* $P = X[i] \dots X[j]$; we say that P occurs at *position* i in X. A *prefix* of X is a fragment of the form $X[1 \,.\,.\, j]$ and a *suffix* of X is a fragment of the form $X[i \,.\,.\, n]$. By XY or $X \cdot Y$ we denote the *concatenation* of two strings X and Y, i.e., $XY = X[1] \dots X[|X|]Y[1] \dots Y[|Y|]$. Given a string X we write $X^R = X[|X|] \dots X[1]$ for the *reverse* of X. Given two strings X and Y we write $\mathsf{LCP}(X, Y)$ for the length of their *longest common prefix*, namely for the integer $\max(\{i,\ X[1 \,.\,.\, i] = Y[1 \,.\,.\, i]\})$, or 0 if $X[1] \neq Y[1]$.

An *elastic-degenerate string* (ED string) $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$ over an alphabet Σ is a sequence of $n = |\tilde{T}|$ finite sets, called *segments*, such that for every position i of $\tilde{T}$ we have that $\tilde{T}[i] \subset \Sigma^*$. By $N = ||\tilde{T}||$ we denote the total length of all strings in all segments of $\tilde{T}$, which we call the *size* of $\tilde{T}$; more formally, $N = \sum_{i=1}^{n} \sum_{j=1}^{|\tilde{T}[i]|} |\tilde{T}[i][j]|$, where by $\tilde{T}[i][j]$ we denote the jth string of $\tilde{T}[i]$. (As an exception, we also add 1 to account for empty strings: if $\tilde{T}[i][j] = \varepsilon$, then we have that $|\tilde{T}[i][j]| = 1$.) Given two sets of strings S_1 and S_2, their *concatenation* is $S_1 \cdot S_2 = \{XY \mid X \in S_1, Y \in S_2\}$. For an ED string $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$, we define the *language* of $\tilde{T}$ as $\mathcal{L}(\tilde{T}) = \tilde{T}[1] \cdot \ldots \cdot \tilde{T}[n]$. Given a set S of strings we write S^R for the set $\{X^R \mid X \in S\}$. For an ED string $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$ we write $\tilde{T}^R$ for the ED string $\tilde{T}[n]^R \dots \tilde{T}[1]^R$.

Given a string P and an ED string $\tilde{T}$, we say that P *matches* the fragment $\tilde{T}[j \,.\,.\, j'] = \tilde{T}[j] \dots \tilde{T}[j']$ of $\tilde{T}$, or that an *occurrence* of P *starts* at position j and *ends* at position j' in $\tilde{T}$ if there exist two strings U, V, each of them possibly empty, such that $P = P_j \cdot \ldots \cdot P_{j'}$, where $P_i \in \tilde{T}[i]$, for every $j < i < j'$, $U \cdot P_j \in \tilde{T}[j]$, and $P_{j'} \cdot V \in \tilde{T}[j']$ (or $U \cdot P_j \cdot V \in \tilde{T}[j]$ when $j = j'$). Strings U, V and P_i, for every $j \leq i \leq j'$, specify an *alignment* of P with $\tilde{T}[j \,.\,.\, j']$. For each occurrence of P in $\tilde{T}$, the alignment is, in general, not unique. In Figure 1, $P = \texttt{TTA}$ matches $\tilde{T}[5 \,.\,.\, 6]$ with two alignments: both have $U = \varepsilon$, $P_5 = \texttt{TT}$, $P_6 = \texttt{A}$, and V is either $\texttt{C}$ or $\texttt{CAC}$.

We want to accept matches with edit distance at most 1 according to the following standard definition:

Definition 1. Given two strings P and Q over an alphabet Σ, we define the *edit distance* $d_E(P, Q)$ between P and Q as the length ℓ of a shortest sequence of string operations $\pi_1, \dots, \pi_\ell$ such that $Q = (\Pi_{i=1}^{\ell} \pi_i)(P)$, where each π_i (for $1 \leq i \leq \ell$) is one of the following type:

- *Replacement:* There is $j \in [1, |P|]$ and $\sigma \neq P[j] \in \Sigma$ s.t. $\pi_i(P)[j] = \sigma$ and $\pi_i(P)[j'] = P[j']$ for $j' \neq j$.
- *Deletion:* One has $|\pi_i(P)| = |P| - 1$ and there is $j \in [1, |P|]$ s.t. $\pi_i(P)[j'] = P[j']$ for $1 \leq j' \leq j - 1$ and $\pi_i(P)[j'] = P[j' + 1]$ for $j \leq j' \leq |P| - 1$.
- *Insertion:* One has $|\pi_i(P)| = |P| + 1$ and there is $j \in [1, |P| + 1]$ s.t. $\pi_i(P)[j'] = P[j']$ for $1 \leq j' \leq j - 1$ and $\pi_i(P)[j'] = P[j' - 1]$ for $j + 1 \leq j' \leq |P| + 1$.

Lemma 1 ([19]). *The function d_E is a distance on Σ^*.*

The following lemma follows immediately from Definition 1.

Lemma 2. *If P, Q are two strings with $d_E(P, Q) = 1$, then $P = \pi(Q)$ where π is a replacement, a deletion or an insertion.*

We define the main problem considered in this paper as follows:

1-Error EDSM
Input: A string P of length m and an ED string $\tilde{T}$ of length n and size N.
Output: All positions j' in $\tilde{T}$ such that there is at least one string P' with an occurrence ending at position j' in $\tilde{T}$, and with $d_E(P, P') \leq 1$ (reporting version); or YES if and only if there is at least one string P' with an occurrence in $\tilde{T}$, and with $d_E(P, P') \leq 1$ (decision version).

Let P' be a string starting at position j and ending at position j' in $\tilde{T}$ with $d_E(P, P') = 1$. We call this *an occurrence of P with* 1 *error* (or a *1-error occurrence*); or equivalently, we say that P *matches* $\tilde{T}[j \mathinner{.\,.} j']$ *with* 1 *error*. Let $UP'_j, \ldots, P'_{j'}V$ be an alignment of P' with $\tilde{T}[j \mathinner{.\,.} j']$ and $i \in [j, j']$ be an integer such that the single replacement, insertion, or deletion required to obtain P from $P' = P'_j \cdot \ldots \cdot P'_{j'}$ occurs on P'_i. We then say that the alignment (and the occurrence) *has the* 1 *error in* $\tilde{T}[i]$. (It should be clear that for one alignment we may have multiple different i.) We show the following theorem.

Theorem 1. *Given a pattern P of length m and an ED text $\tilde{T}$ of length n and size N, the reporting version of* 1-Error EDSM *can be solved on-line in $\mathcal{O}(nm^2 \log m + N \log m)$ or $\mathcal{O}(nm^3 + N)$ time. The decision version of* 1-Error EDSM *can be solved off-line in $\mathcal{O}(nm^2\sqrt{\log m} + N \log\log m)$ time.*

Hamming distance, denoted by d_H, is a special case of edit distance in which only replacement operations are allowed (it is therefore defined for two strings of equal length). We define the following problem:

1-Mismatch EDSM
Input: A string P of length m and an ED string $\tilde{T}$ of length n and size N.
Output: All positions j' in $\tilde{T}$ such that there is at least one string P' with an occurrence ending at position j' in $\tilde{T}$, and with $d_H(P, P') \leq 1$.

An occurrence of a string P' as in the problem definition is called an *occurrence of P with* 1 *mismatch*. We call *mismatch* the single position in the support of the replacement π such that $\pi(P) = P'$. We show the following theorem.

Theorem 2. *Given a pattern P of length m and an ED text $\tilde{T}$ of length n and size N,* 1-Mismatch EDSM *can be solved on-line in $\mathcal{O}(nm^2 + N \log m)$ or $\mathcal{O}(nm^3 + N)$ time.*

Definition 2. For a string $P = P[1 \mathinner{.\,.} m]$, an ED string $\tilde{T} = \tilde{T}[1] \ldots \tilde{T}[n]$, a position $1 \leq i \leq n$, and a distance on Σ^*, we define three sets:

- $AP_i \subseteq [1, m]$, such that $j \in AP_i$ if and only if $P[1 \mathinner{.\,.} j]$ is an *active prefix* of P in $\tilde{T}$ ending in the segment $\tilde{T}[i]$, that is, a prefix of P which is also a suffix of a string in $\mathcal{L}(\tilde{T}[1] \ldots \tilde{T}[i])$.
- $AS_i \subseteq [1, m]$, such that $j \in AS_i$ if and only if $P[j \mathinner{.\,.} m]$ is an *active suffix* of P in $\tilde{T}$ starting in the segment $\tilde{T}[i]$, that is, a suffix of P which is also a prefix of a string in $\mathcal{L}(\tilde{T}[i] \ldots \tilde{T}[n])$.
- 1-$AP_i \subseteq [1, m]$, such that $j \in$ 1-AP_i if and only if $P[1 \mathinner{.\,.} j]$ is an *active prefix with* 1 *error* of P in $\tilde{T}$ ending in the segment $\tilde{T}[i]$, that is, a prefix of P which is also at distance at most 1 from a suffix of a string in $\mathcal{L}(\tilde{T}[1] \ldots \tilde{T}[i])$.

For convenience we also define $AP_0 = AS_{n+1} =$ 1-$AP_0 = \emptyset$.

The following lemma shows that the computation of active suffixes can be easily reduced to computing the active prefixes for the reversed strings.

Lemma 3. *Given a pattern $P = P[1 \mathinner{.\,.} m]$ and an ED text $\tilde{T} = \tilde{T}[1 \mathinner{.\,.} n]$, a suffix $P[j \mathinner{.\,.} m]$ of P is an active suffix in $\tilde{T}$ starting in the segment $\tilde{T}[i]$ if and only if the prefix $P^R[1 \mathinner{.\,.} m - j + 1] = (P[j \mathinner{.\,.} m])^R$ of P^R is an active prefix in $\tilde{T}^R$, ending in the segment $\tilde{T}^R[n - i + 1] = (\tilde{T}[i])^R$.*

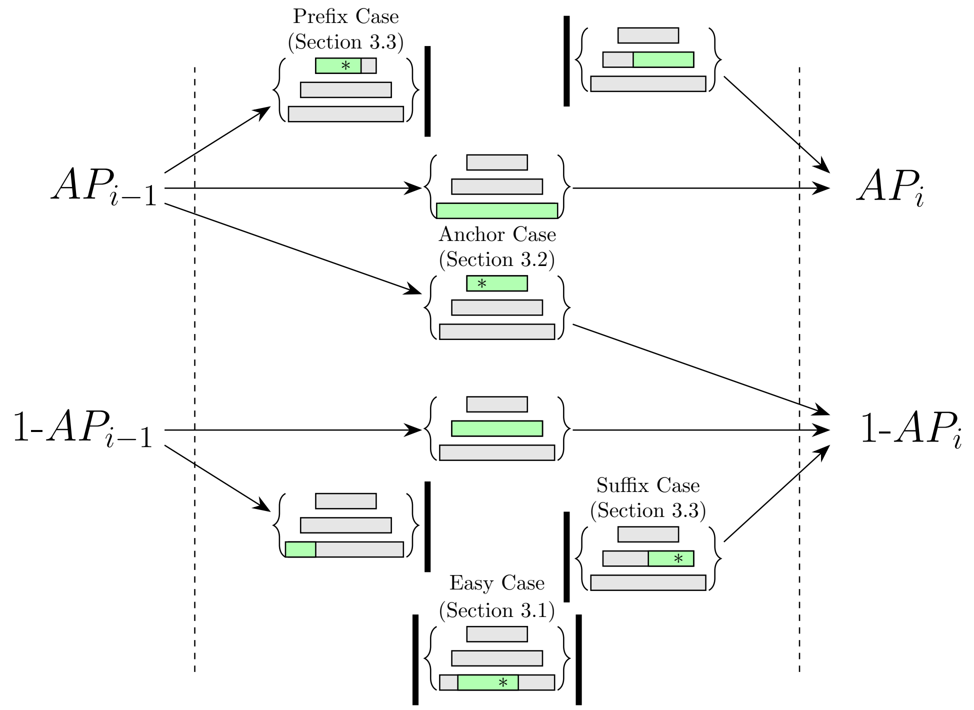


Figure 2: The layout of the algorithms for computing AP_i, 1-AP_i, and reporting occurrences. The green areas correspond to the (partial) matches in $\tilde{T}[i]$, and the symbol $*$ indicates the position of an error. The vertical bold lines indicate the beginning/the end of an occurrence or a 1-error occurrence. The cases without a label allow only exact matches and were already solved by Grossi et al. in [27].

Proof. If $P[j\,..\,m]$ is a prefix of $S \in \mathcal{L}(\tilde{T}[i\,..\,n])$, then $P^R[1\,..\,m-j+1]$ is a suffix of $S^R \in \mathcal{L}(\tilde{T}[1\dots n]^R)$. From the definition of $\tilde{T}^R$ we have $\tilde{T}[i\,..\,n]^R = (\tilde{T[n]})^R \dots (\tilde{T[i]})^R = \tilde{T}^R[1\,..\,n-i+1]$, hence $S^R \in \mathcal{L}(\tilde{T}^R[1\,..\,n-i+1])$.

This proves the forward direction of the lemma; the converse follows from symmetry. ☐

The efficient computation of active prefixes was shown in [27], and constitutes the main part of the combinatorial algorithm for exact EDSM. Similarly, computing the sets 1-AP plays the key role in the reporting version of our algorithm for 1-Error EDSM (see Figure 2). Finding active prefixes (and, by Lemma 3, suffixes) reduces to the following problem, formalized in [7].

Active Prefixes Extension (APE)
Input: A string P of length m, a bit vector U of size m, and 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 $i \in [1, m]$, such that $P[1\,..\,i] \cdot S = P[1\,..\,j]$ and $U[i] = 1$.

Lemma 4 ([27])**.** *The APE problem for a string P of length m and a set $\mathcal{S}$ of strings of total length N can be solved in $\mathcal{O}(m^2 + N)$ time.*

Given an algorithm for the APE problem working in $f(m) + N$ time, we can find *all* active prefixes for a pattern P of length m in an ED text $\tilde{T} = \tilde{T}[1]\dots\tilde{T}[n]$ of size N in $\mathcal{O}(nf(m) + N)$ total time:

Corollary 3 ([27])**.** *For a pattern P of length m and an ED text $\tilde{T} = \tilde{T}[1]\dots\tilde{T}[n]$ of total size N, computing the sets AP_i for all $i \in [1, n]$ takes $\mathcal{O}(nm^2 + N)$ time.*

As depicted in Figure 2, the computation of active prefixes with 1 error (1-AP_i) and the reporting of occurrences with 1 error reduce to a problem where the error can only occur in a single, fixed $\tilde{T}[i]$. In particular, this problem decomposes into 4 cases, which we formalize in the following proposition.

Proposition 4. *Let $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$ be an ED text and P be a pattern that has an occurrence with 1 error (resp. 1 mismatch) in $\tilde{T}$. For each alignment corresponding to such occurrence, at least one of the following is true:*

Easy Case: *P matches $\tilde{T}[i]$ with 1 error (resp. 1 mismatch) for some $1 \le i \le n$.*

Anchor Case: *P matches $\tilde{T}[j \,.\,.\, j']$ with 1 error (resp. 1 mismatch) in $\tilde{T}[i]$ for some $1 \le j < i < j' \le n$. $\tilde{T}[i]$ is called the* anchor *of the alignment.*

Prefix Case: *P matches $\tilde{T}[j \,.\,.\, i]$ with 1 error (resp. 1 mismatch) in $\tilde{T}[i]$ for some $1 \le j < i \le n$, implying an active prefix of P which is a suffix of a string in $\mathcal{L}(\tilde{T}[j \,.\,.\, i-1])$.*

Suffix Case: *P matches $\tilde{T}[i \,.\,.\, j']$ with 1 error (resp. 1 mismatch) in $\tilde{T}[i]$ for some $1 \le i < j' \le n$, implying an active suffix of P which is a prefix of a string in $\mathcal{L}(\tilde{T}[i+1 \,.\,.\, j'])$.*

Proof. Suppose P has a 1-error (resp. 1 mismatch) occurrence matching $\tilde{T}[j \,.\,.\, j']$ with $1 \le j \le j' \le n$. If $j = j'$ we are in the Easy Case. Otherwise, each alignment has an error in some $\tilde{T}[i]$ for $j \le i \le j'$. If $j < i < j'$, we are in the Anchor Case; if $j < i = j'$, we are in the Prefix Case; and if $j = i < j'$, we are in the Suffix Case. □ □

3 1-Error EDSM

In this section, we present algorithms for finding all 1-occurrences of P given by each type of possible alignment described by Proposition 4 (inspect Figure 3). The Prefix and Suffix cases are analogous by Lemma 3; the only difference is in that, while the Suffix Case computes new 1-AP, the Prefix Case is used to actually report occurrences. They are jointly considered in Section 3.3.

We follow two different procedures for the decision and reporting versions. For the decision version, we precompute sets AP_i and AS_i, for all $i \in [1, n]$, using Corollary 3, and we simultaneously compute possible exact occurrences of P. Then we compute 1-error occurrences of P by grouping the alignments depending on the segment i in which the error occurs, and using AP_i and AS_i. For the reporting version, we consider one segment $\tilde{T}[i]$ at a time (on-line) and extend partial exact or 1-error occurrences of P to compute sets AP_i and 1-AP_i using just sets AP_{i-1} and 1-AP_{i-1} computed at the previous step. We design different procedures for the 4 cases of Proposition 4. We can sort all letters of P, assign them rank values from $[1, m]$, and construct a perfect hash table over these letters supporting $\mathcal{O}(1)$-time look-up queries in $\mathcal{O}(m \log m)$ time [35]. Any letter of $\tilde{T}$ not occurring in P can be replaced by the same special letter in $\mathcal{O}(1)$ time. In the rest we thus assume that the input strings are over $[1, m+1]$.

Two problems from computational geometry have a key role in our solutions. We assume the word RAM model with coordinates on the integer grid $[1, n]^d = \{1, 2, \dots, n\}^d$. In the *2d rectangle emptiness* problem, we are given a set $\mathcal{P}$ of n points to be preprocessed, so that when one gives an axis-aligned rectangle as a query, we report YES if and only if the rectangle contains a point from $\mathcal{P}$. In the "dual" *2d rectangle stabbing* problem, we are given a set $\mathcal{R}$ of n axis-aligned rectangles to be preprocessed, so that when one gives a point as a query, we report YES if and only if there exists a rectangle from $\mathcal{R}$ containing the point.

Lemma 5 ([11, 23])**.** *After $\mathcal{O}(n\sqrt{\log n})$-time preprocessing, we can answer 2d rectangle emptiness queries in $\mathcal{O}(\log \log n)$ time.*

Lemma 6 ([15, 36])**.** *After $\mathcal{O}(n \log n)$-time preprocessing, we can answer 2d rectangle stabbing queries in $\mathcal{O}(\log n)$ time.*

In Section 3.4, we note that the 2d rectangle stabbing instances arising from 1-Error EDSM have a special structure. We show how to solve them efficiently thus shaving logarithmic factors from the time complexity.

3.1 Easy Case

The Easy Case can be reduced to approximate string matching with at most 1 error (1-SM):

> 1-SM
> **Input:** A string P of length m and a string T of length n.
> **Output:** All positions j in T such that there is at least one string P' ending at position j in T with $d_E(P, P') \leq 1$.

We have the following well-known results.

Lemma 7 ([31, 18])**.** *Given a pattern P of length m, a text T of length n, and an integer $k > 0$, all positions j in T such that the edit distance of $T[i\,..\,j]$ and P, for some position $i \leq j$ on T, is at most k, can be found in $\mathcal{O}(kn)$ time or in $\mathcal{O}(\frac{nk^4}{m} + n)$ time.*[1] *In particular,* 1*-SM can be solved in $\mathcal{O}(n)$ time.*

We find occurrences of P with at most 1 error that are in the Easy Case for segment $\tilde{T}[i]$ in the following way: we apply Lemma 7 for $k = 1$ and every string of $\tilde{T}[i]$ whose length is at least $m - 1$ (any shorter string is clearly not relevant for this case) as text. If, for any of those strings, we find an occurrence of P, we report an occurrence at position i (inspect Figure 3a). The time for processing a segment $\tilde{T}[i]$ is $\mathcal{O}(N_i)$, where N_i is the total length of all the strings in $\tilde{T}[i]$.

3.2 Anchor Case

Let $\tilde{T}$ be an ED text and P be a pattern with a 1-error occurrence and an alignment in the Anchor Case with anchor $\tilde{T}[i]$. Further let $L = P[1\,..\,\ell]S'$ and $Q = S''P[q\,..\,m]$ be a prefix and a suffix of P, respectively, for some $\ell \in AP_{i-1}, q \in AS_{i+1}$, where S', S'' are a prefix and a suffix of some $S \in \tilde{T}[i]$, respectively (strings S', S'' can be empty). By Lemma 2, a pair L, Q gives a 1-error occurrence of P if one of the following holds:

1 mismatch: $|L| + |Q| + 1 = m$ *and* $|S'| + |S''| + 1 = |S|$ (inspect Figure 3b).

1 deletion in P: $|L| + |Q| = m - 1$ *and* $|S'| + |S''| = |S|$.

1 insertion in P: $|L| + |Q| = m$ *and* $|S'| + |S''| + 1 = |S|$.

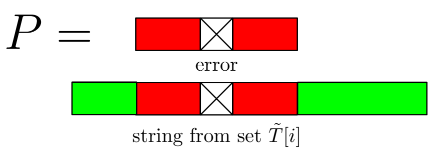
We show how to find such pairs with the use of a geometric approach. For convenience, we only present the Hamming distance (1 mismatch) case. The other cases are handled similarly.

Let $\lambda \in AP_{i-1}$ be the length of an active prefix, and let ρ be the length of an active suffix, that is, $m - \rho + 1 \in AS_{i+1}$. Note that AP_{i-1} and AS_{i+1} can be precomputed, for all i, in $\mathcal{O}(nm^2 + N)$ total time by means of Corollary 3. (In particular, AS_{i+1} is required only for the decision version; for the reporting version, we explain later on how to avoid the precomputation of AS_{i+1} to obtain an on-line algorithm.) We will exhaustively consider all pairs (λ, ρ) such that $\lambda + \rho < m$. Clearly, there are $\mathcal{O}(m^2)$ such pairs.

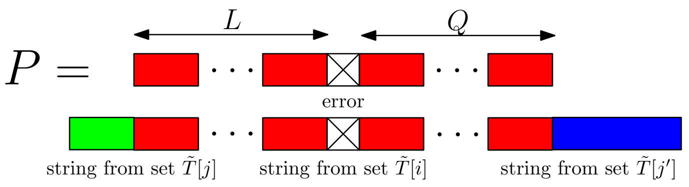
Consider the length $\mu = m - (\lambda + \rho) > 0$ of the substring of P still to be matched for some prefix and suffix of P of lengths (λ, ρ), respectively. We group together all pairs (λ, ρ) such that $m - (\lambda + \rho) = \mu$ by sorting them in $\mathcal{O}(m^2)$ time. We construct, for each such group μ, the compacted trie T_μ of the fragments $P[\lambda + 1\,..\,m - \rho]$, for all (λ, ρ) such that $m - (\lambda + \rho) = \mu$, and analogously the compacted trie T^R_μ of all fragments $P^R[\rho + 1\,..\,m - \lambda]$. For each group μ, this takes $\mathcal{O}(m)$ time [33]. We enhance all nodes with a perfect hash table in $\mathcal{O}(m)$ total time to access edges by the first letter of their label in $\mathcal{O}(1)$ time [22].

We also group all strings in segment $\tilde{T}[i]$ of length less than m by their length μ. The group for length μ is denoted by G_μ. This takes $\mathcal{O}(N_i)$ time. Clearly, the strings in G_μ are the only candidates to extend pairs (λ, ρ) such that $m - (\lambda + \rho) = \mu$. Note that the mismatch can be at any position of any string of G_μ: its position determines a prefix S' of length h and a suffix S'' of length k of the same string S, with $h + k = \mu - 1$, that must match a prefix and a suffix of $P[\lambda + 1\,..\,m - \rho]$, respectively. We will consider

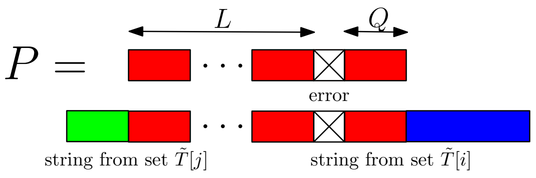
[1]Charalampopoulos et al. have announced an improvement on the exponent of k from 4 to 3.5; specifically they presented an $\mathcal{O}(\frac{nk^{3.5}\sqrt{\log m \log k}}{m} + n)$-time algorithm [14].



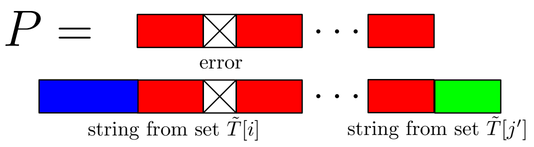
(a) Easy Case: $j = i = j'$.


L
Q
$P =$
error
string from set $\tilde{T}[j]$
string from set $\tilde{T}[i]$
string from set $\tilde{T}[j']$

(b) Anchor Case: $j \neq i, i \neq j'$.


L
Q
$P =$
error
string from set $\tilde{T}[j]$
string from set $\tilde{T}[i]$

(c) Prefix Case: $j \neq i, i = j'$.


$P =$
error
string from set $\tilde{T}[i]$
string from set $\tilde{T}[j']$

(d) Suffix Case: $i = j, i \neq j'$.

Figure 3: Possible alignments of 1-error occurrences of P in $\tilde{T}$. Each occurrence starts at segment $\tilde{T}[j]$, ends at $\tilde{T}[j']$, and the error occurs at $\tilde{T}[i]$.

all such pairs of positions (h, k) whose sum is $\mu - 1$ (intuitively, the minus one is for the mismatch). This guarantees that $L = P[1 \mathinner{.\,.} \lambda]S'$ and $Q = S''P[m - \rho + 1 \mathinner{.\,.} m]$ are such that $|L| + |Q| + 1 = m$. The pairs are $(0, \mu - 1), (1, \mu - 2), \ldots, (\mu - 1, 0)$. This guarantees that L and Q are *one position apart* ($|S'| + |S''| + 1 = |S|$).

The number of these pairs is $\mathcal{O}(\mu) = \mathcal{O}(m)$. Consider one such pair (h, k) and a string $S \in G_\mu$. We treat every such string S separately. We spell $S[1 \mathinner{.\,.} h]$ in T_μ. If the whole $S[1 \mathinner{.\,.} h]$ is successfully spelled ending at a node u, this implies that all the fragments of P corresponding to nodes descending from u share $S[1 \mathinner{.\,.} h]$ as a prefix. We also spell $S^R[1 \mathinner{.\,.} k]$ in T^R_μ. If the whole of $S^R[1 \mathinner{.\,.} k]$ is successfully spelled ending at a node v, then all the fragments of P corresponding to nodes descending from v share $(S^R[1 \mathinner{.\,.} k])^R$ as a suffix. Nodes u and v identify an interval of leaves in T_μ and T^R_μ, respectively. We need to check if these intervals both contain a leaf corresponding to the same fragment of P. If they do, then we obtain an occurrence of P with 1 mismatch (see Figure 4). We now have two different ways to proceed, depending on whether we need to solve the off-line decision version or the on-line reporting version.

Decision Version Let us recall that T_μ, T^R_μ by construction are ordered based on lexicographic ranks. For every pair (T_μ, T^R_μ), we construct a data structure for 2d rectangle emptiness queries on the grid $[1, \ell]^2$, where ℓ is the number of leaves of T_μ (and of T^R_μ), for the set of points (x, y) such that x is the lexicographic rank of the leaf of T_μ representing $P[\lambda + 1 \mathinner{.\,.} m - \rho]$ and y is the rank of the leaf of T^R_μ representing

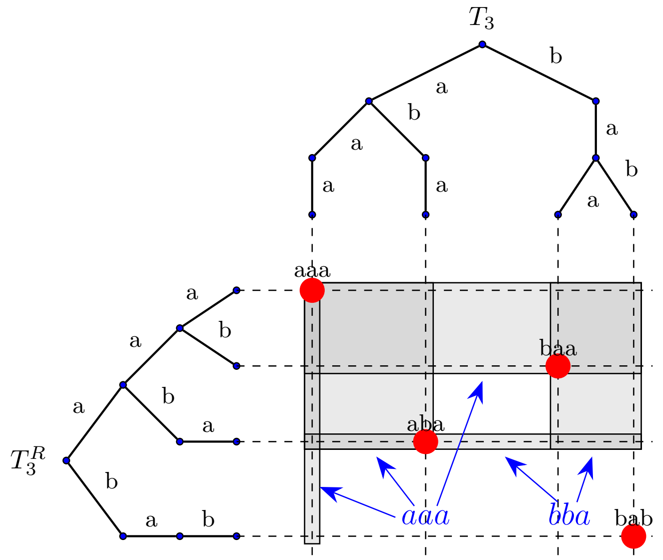


Figure 4: An example of points and rectangles (solid shapes) for the decision version of the Anchor Case with 1 mismatch. Here $P = bbaaaababab b$, $AP_{i-1} = \{1, 2, 4, 7, 8, 9\}$, $AS_{i+1} = \{5, 6, 9, 11, 12\}$, $\mu = 3$, and $\tilde{T}[i] = \{aaa, bba\}$. T_3 and T_3^R are built for 4 strings: $P[2\,..\,4] = baa$, $P[3\,..\,5] = aaa$, $P[8\,..\,10] = aba$, $P[9\,..\,11] = bab$; the 5 rectangles correspond to pairs (ε, aa), (a, a), (aa, ε), (ε, ab), (b, a), namely, the pairs of prefixes and reversed suffixes of aaa and bba (rectangle (bb, ε) does not exist as T_3 contains no node bb).

$P^R[\rho+1\,..\,m-\lambda]$ for the same pair (λ, ρ). This denotes that the two leaves correspond to *the same fragment* of P. For every (T_μ, T_μ^R), this preprocessing takes $\mathcal{O}(m\sqrt{\log m})$ time by Lemma 5, since ℓ is $\mathcal{O}(\mu) = \mathcal{O}(m)$. For all μ groups (they are at most m), the whole preprocessing thus takes $\mathcal{O}(m^2\sqrt{\log m})$ time.

We then ask 2d range emptiness queries that take $\mathcal{O}(\log\log m)$ time each by Lemma 5. Note that all rectangles for S can be collected in $\mathcal{O}(|S|) = \mathcal{O}(\mu)$ time by spelling S through T_μ and S^R through T_μ^R, one letter at a time. Thus the total time for processing all G_μ groups of segment i is $\mathcal{O}(m^2\sqrt{\log m} + N_i \log\log m)$. If any of the queried ranges turns out to be non-empty, then P' such that $d_H(P, P') \leq 1$ appears in $\mathcal{L}(\tilde{T})$ with anchor in $\tilde{T}[i]$; we do not have sufficient information to output its ending position however.

Reporting Version For this version, we do the dual. We construct a data structure for 2d rectangle stabbing queries on the grid $[1, \ell]^2$ for the set of rectangles collected for all strings $S \in G_\mu$. By Lemma 6, for all μ groups, the whole preprocessing thus takes $\mathcal{O}(N_i \log N_i)$ time.

For every (T_μ, T_μ^R), we then ask the following queries: (x, y) is queried if and only if x is the rank of a leaf representing $P[\lambda+1\,..\,m-\rho]$ and y is the rank of a leaf representing $P^R[\rho+1\,..\,m-\lambda]$. For every (T_μ, T_μ^R), this takes $\mathcal{O}(m \log N_i)$ time by Lemma 6 and by the fact that for each group G_μ there are $\mathcal{O}(m)$ pairs (λ, ρ) such that $m - (\lambda + \rho) = \mu$. For all groups G_μ (they are at most m), all the queries thus take $\mathcal{O}(m^2 \log N_i)$ time. Thus the total time for processing all G_μ groups of segment i is $\mathcal{O}((m^2 + N_i)\log N_i)$.

We are not done yet. By performing the above algorithm for active prefixes and active suffixes, we find out which pairs can be completed to a full occurrence of P with at most 1 error. This information is not sufficient to compute where such an occurrence ends (and storing additional information together with the active suffixes may prove costly). To overcome this, we use some ideas from the decision algorithm, appropriately modified to preserve the on-line nature of the reporting algorithm. Instead of iterating ρ over the lengths of precomputed active suffixes, we iterate it over *all* possible lengths in $[0, m]$ (including 0 because we may want to include m in 1-AP_i). A suffix of P of length ρ completes a partial occurrence computed up to segment i exactly when $m - \rho \in$ 1-AP_i (a pair $x \in$ 1-AP_i, $x + 1 \in AS_{i+1}$ corresponds to an occurrence). We thus use the reporting algorithm to compute the part of 1-AP_i coming from the extension of AP_{i-1} (see Figure 2), and defer the reporting to the no-error version of the Prefix Case for the right j'; which was solved by Grossi et al. [27] in linear time.

3.3 Prefix Case

Let $\tilde{T}$ be an ED text and P be a pattern with a 1-error occurrence and an alignment in the Prefix Case with active prefix ending at $\tilde{T}[i-1]$. Let $L = P[1\mathinner{.\,.}\ell]S'$, with $\ell \in AP_{i-1}$, be a prefix of P that is extended in $\tilde{T}[i]$ by S'; and Q be a suffix of P occurring in some string of $\tilde{T}[i]$ (strings S', Q can be empty). By Lemma 2, we have 3 possibilities for any alignment of a 1-error occurrence of P in the Prefix Case:

1 mismatch: $|L| + |Q| + 1 = m$, S' is a prefix of the same string in which Q occurs, and they are one position apart (inspect Figure 3c).

1 deletion in P: $|L| + |Q| = m - 1$, S' is a prefix of the same string in which Q occurs, and they are consecutive.

1 insertion in P: $|L| + |Q| = m$, S' is a prefix of the same string in which Q occurs, and they are one position apart.

For convenience, we only present the method for Hamming distance (1 mismatch). The other possibilities are handled similarly.

The techniques are similar to those for the Anchor Case (Section 3.2). We group the prefixes of all strings in $\tilde{T}[i]$ according to their length $\mu \in [1, m)$. The total number of these prefixes is $\mathcal{O}(N_i)$. The group for length μ is denoted by G_μ. We construct the compacted trie T_{G_μ} of the strings in G_μ, and the compacted trie $T^R_{G_\mu}$ of the reversed strings in G_μ. This can be done in $\mathcal{O}(N_i)$ total time for all compacted tries. To achieve this, we employ the following lemma by Charalampopoulos et al. [12]. (Recall that we have already sorted all letters of P. In what follows, we assume that $N_i \geq m$; if this is not the case, we can sort all letters of $\tilde{T}[i]$ in $\mathcal{O}(m + N_i)$ time.)

Lemma 8 ([12])**.** *Let X be a string of length n over an integer alphabet of size $n^{\mathcal{O}(1)}$. Let I be a collection of intervals $[i, j] \subseteq [1, n]$. We can lexicographically sort the substrings $X[i\mathinner{.\,.}j]$ of X, for all intervals $[i, j] \in I$, in $\mathcal{O}(n + |I|)$ time.*

We concatenate all the strings of $\tilde{T}[i]$ to obtain a single string X of length N_i, to which we apply, for each μ, Lemma 8, with a set I consisting of the intervals over X corresponding to the strings in G_μ. By sorting, in this way, all strings in G_μ (for all μ), and by constructing [21] and preprocessing [6] the generalized suffix tree of the strings in $\tilde{T}[i]$ in $\mathcal{O}(N_i)$ time to support answering lowest common ancestor (LCA) queries in $\mathcal{O}(1)$ time, we can construct all T_{G_μ} in $\mathcal{O}(N_i)$ total time. We handle $T^R_{G_\mu}$, for all μ, analogously. Similar to the Anchor Case we enhance all nodes with a perfect hash table within the same complexities [22].

In contrast to the Anchor Case, we now only consider the set AP_{i-1}: namely, we do not consider AS_{i+1}. Let $\lambda \in AP_{i-1}$ be the length of an active prefix. We treat every such element separately, and they are $\mathcal{O}(m)$ in total. Let $\mu = m - \lambda > 0$ and consider the group G_μ whose strings are all of length μ. The mismatch being at position $h + 1$ in one such string S determines a prefix S' of S of length h that must extend the active prefix of P of length λ, and a fragment Q of S of length $k = \mu - h - 1$ that must match a suffix of P. We will consider all such pairs (h, k) whose sum is $\mu - 1$. The pairs are again $(0, \mu - 1), (1, \mu - 2), \ldots, (\mu - 1, 0)$, and there are clearly $\mathcal{O}(\mu) = \mathcal{O}(m)$ of them.

Consider (h, k) as one such pair. We spell $P[\lambda + 1\mathinner{.\,.}\lambda + h]$ in T_{G_μ}. If the whole $P[\lambda + 1\mathinner{.\,.}\lambda + h]$ is spelled successfully, this implies an interval of leaves of T_{G_μ} corresponding to strings from $\tilde{T}[i]$ that share $P[\lambda + 1\mathinner{.\,.}\lambda + h]$ as a prefix. We spell $P^R[1\mathinner{.\,.}k]$ in $T^R_{G_\mu}$. If the whole $P^R[1\mathinner{.\,.}k]$ is spelled successfully, this implies an interval of leaves of $T^R_{G_\mu}$ corresponding to strings from $\tilde{T}[i]$ that have the same fragment $(P^R[1\mathinner{.\,.}k])^R$. These two intervals form a rectangle in the grid implied by the leaves of T_{G_μ} and $T^R_{G_\mu}$. We need to check if these intervals both contain a leaf corresponding to the same prefix of length μ of a string in $\tilde{T}[i]$. If they do, then we have obtained an occurrence with 1 mismatch in $\tilde{T}[i]$.

To do this we construct, for every $(T_{G_\mu}, T^R_{G_\mu})$, a 2d range data structure for the set of points (x, y) such that x is the rank of a leaf of T_{G_μ}, y is the rank of a leaf of $T^R_{G_\mu}$, and the two leaves correspond to *the same prefix* of length μ of a string in $\tilde{T}[i]$. For every $(T_{G_\mu}, T^R_{G_\mu})$, this takes $\mathcal{O}(|G_\mu|\sqrt{\log |G_\mu|})$ time by Lemma 5. For all G_μ groups, the whole preprocessing takes $\mathcal{O}(N_i\sqrt{\log N_i})$ time.

We then ask 2d range emptiness queries each taking $\mathcal{O}(\log\log |G_\mu|)$ time by Lemma 5. Note that all rectangles for λ can be collected in $\mathcal{O}(m)$ time by spelling $P[\lambda+1 \mathinner{.\,.} \lambda+\mu-1]$ through T_{G_μ} and $P^R[1 \mathinner{.\,.} \mu-1]$ through $T^R_{G_\mu}$, one letter at a time. This gives a total of $\mathcal{O}(m^2 \log\log N_i + N_i\sqrt{\log N_i})$ time for processing all G_μ groups of $\tilde{T}[i]$, because $\sum_\mu |G_\mu| \le N_i$.

To solve the Suffix Case (compute active prefixes with 1 error starting in $\tilde{T}[i]$) we employ the mirror version of the algorithm, but iterating λ over the whole $[0, m]$ instead of AS_{i+1} (like in the reporting version of the Anchor Case).

3.4 Shaving Logs using Special Cases of Geometric Problems

3.4.1 Anchor Case: Simple 2d Rectangle Stabbing

Lemma 9. *We can solve the Anchor Case (i.e., extend AP_{i-1} into 1-AP_i) in $\mathcal{O}(m^3 + N_i)$ time.*

Proof. By Lemma 6, 2d rectangle stabbing queries can be answered in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n \log n)$ space after $\mathcal{O}(n \log n)$-time preprocessing.

Notice that in the case of the 2d rectangle stabbing used in Section 3.2 the rectangles and points are all in a predefined $[1, m] \times [1, m]$ grid. In such a case we can also use an easy folklore data structure of size $\mathcal{O}(m^2)$, which after an $\mathcal{O}(m^2 + |\text{rectangles}|)$-time preprocessing answers such queries in $\mathcal{O}(1)$ time.

Namely, the data structure consists of a $[1, m+1]^2$ grid Γ (a 2d-array of integers) in which for every rectangle $[u, v] \times [w, x]$ we add 1 to $\Gamma[u][w]$ and $\Gamma[v+1][x+1]$ and -1 to $\Gamma[u][x+1]$ and $\Gamma[v+1][w]$. Then we modify Γ to contain the 2d prefix sums of its original values (we first compute prefix sums of each row, and then prefix sums of each column of the result). After these modifications, $\Gamma[x][y]$ stores the number of rectangles containing point (x, y), and hence after $\mathcal{O}(m^2 + |\text{rectangles}|)$-time preprocessing we can answer 2d rectangle stabbing queries in $\mathcal{O}(1)$ time.

In our case we have a total of $\mathcal{O}(m)$ such grid structures, each of $\mathcal{O}(m^2)$ size, and ask $\mathcal{O}(m^2)$ queries, and hence obtain an $\mathcal{O}(m^3 + N_i)$-time and $\mathcal{O}(m^2)$-space solution for computing 1-AP_i from AP_{i-1}. $\square$

3.4.2 Prefix Case: a Special Case of 2d Rectangle Stabbing

Inspect the example of Figure 4 for the Anchor Case. Note that the groups of rectangles for each string have the special property of being composed of *nested intervals*: for each dimension, the interval corresponding to a given node is included in the one corresponding to any of its ancestors. Thus for the Prefix Case, where we only spell fragments of the same string P in both compacted tries, we consider the following special case of off-line 2d rectangle stabbing.

Lemma 10. *Let $p_1, \ldots, p_h$ and $q_1, \ldots, q_h$ be two permutations of $[1, h]$. We denote by Π the set of h points $(p_1, q_1), (p_2, q_2), \ldots, (p_h, q_h)$ on $[1, h]^2$.*

Further let R be a collection of r axis-aligned rectangles $([u_1, v_1], [w_1, x_1]), \ldots, ([u_r, v_r], [w_r, x_r])$, such that

$$[u_r, v_r] \subseteq [u_{r-1}, v_{r-1}] \subseteq \cdots \subseteq [u_1, v_1]$$

and

$$[w_1, x_1] \subseteq [w_2, x_2] \subseteq \cdots \subseteq [w_r, x_r].$$

Then we can find out, for every point from Π, if it stabs any rectangle from R in $\mathcal{O}(h + r)$ total time.

Proof. Let H be a bit vector consisting of h bits, initially all set to zero. We process one rectangle at a time. We start with $([u_1, v_1], [w_1, x_1])$. We set $H[p] = 1$ if and only if $(p, q) \in \Pi$ for $p \in [u_1, v_1]$ and any q. We collect all p such that $(p,q) \in \Pi$ and $q \in [w_1, x_1]$, and then search for these p in H: if for any p, $H[p] = 1$, then the answer is positive for p. Otherwise, we remove from H every p such that $p \in [u_1, v_1]$ and $p \notin [u_2, v_2]$ by setting $H[p] = 0$. We proceed by collecting all p such that $(p,q) \in \Pi$, $q \in [w_2, x_2]$ and $q \notin [w_1, x_1]$, and then search for them in H: if for any p, $H[p] = 1$, then the answer is positive for p. We repeat this until H is empty or until there are no other rectangles to process.

The whole procedure takes $\mathcal{O}(h + r)$ time, because we set at most h bits on in H, we set at most h bits back off in H, we search for at most h points in H, and then we process r rectangles. ☐

Lemma 11. *We can solve the Prefix (resp. Suffix) Case, that is, report* 1*-error occurrences ending in* $\tilde{T}[i]$ *(resp. compute active prefixes with* 1 *error starting in* $\tilde{T}[i]$*) in* $\mathcal{O}(m^2 + N_i)$ *time.*

Proof. We employ Lemma 10 to get rid of the 2d range data structure. The key is that for every length-μ suffix $P[\lambda + 1 \mathinner{..} m]$ of the pattern we can afford to pay $\mathcal{O}(\mu + |G_\mu|)$ time plus the time to construct T_{G_μ} and $T^R_{G_\mu}$ for set G_μ. Because the grid is $[1, |G_\mu|]^2$, we exploit the fact that the intervals found by spelling $P[\lambda + 1 \mathinner{..} \lambda + \mu - 1]$ through T_{G_μ} and $P^R[1 \mathinner{..} \mu - 1]$ through $T^R_{G_\mu}$, one letter at a time, are subset of each other, and querying μ such rectangles is done in $\mathcal{O}(\mu + |G_\mu|)$ time by employing Lemma 10. Since we process at most m distinct length-μ suffixes of P, the total time is $\mathcal{O}(m^2 + N_i)$, because $\sum_\mu |G_\mu| \leq N_i$. ☐

3.5 Wrapping-up

To obtain Theorem 1 for the decision version of the problem we first compute AP_i and AS_i, for all $i \in [1, n]$, in $\mathcal{O}(nm^2 + N)$ total time (Corollary 3). We then compute all the occurrences in the Easy Cases using $\mathcal{O}(N)$ time in total (Section 3.1); and we finally compute all the occurrences in the Prefix and Suffix Cases in $\sum_i \mathcal{O}(m^2 + N_i) = \mathcal{O}(nm^2 + N)$ total time (Lemma 11).

Now, to solve the decision version of the problem, we solve the Anchor Cases with the use of the pre-computed AP_{i-1} and AS_{i+1} for each $i \in [2, n-1]$ in $\mathcal{O}(m^2\sqrt{\log m} + N_i \log\log m)$ time (Section 3.2), which gives $\mathcal{O}(nm^2\sqrt{\log m} + N \log\log m)$ total time for the whole algorithm.

For the reporting version we proceed differently to obtain an on-line algorithm; note that this is possible because we can proceed without AS_i (see Figure 2). We thus consider one segment $\tilde{T}[i]$ at the time, for each $i \in [1, n]$, and do the following. We compute 1-AP_i, as the union of three sets obtained from:

- The Suffix Case for $\tilde{T}[i]$, computed in $\mathcal{O}(m^2 + N_i)$ time (Lemma 11).
- Standard APE with 1-AP_{i-1} as the input bit vector, computed in $\mathcal{O}(m^2 + N_i)$ time (Lemma 4).
- Anchor Case computed from AP_{i-1} in $\mathcal{O}((m^2 + N_i) \log N_i)$ (Section 3.2) or $\mathcal{O}(m^3 + N_i)$ time (Lemma 9).

If $N_i \geq m^3$, the algorithm of Lemma 9 works in the optimal $\mathcal{O}(m^3 + N_i) = \mathcal{O}(N_i)$ time, hence we can assume that the $\mathcal{O}((m^2 + N_i) \log N_i)$-time algorithm is only used when $N_i \leq m^3$, and thus it runs in $\mathcal{O}((m^2 + N_i) \log m)$ time. Therefore over all i the computations require $\mathcal{O}((nm^2 + N) \log m)$ or $\mathcal{O}(nm^3 + N)$ total time. For every segment i we can also check whether an active prefix from 1-AP_{i-1} or from AP_{i-1} can be completed to a full match in $\tilde{T}[i]$ using the algorithms of Grossi et al. from [27] and Prefix Case, respectively, in $\mathcal{O}(m^2 + N_i)$ extra time.

By summing up all these we obtain Theorem 1.

4 1-Mismatch EDSM

In this section, we give an alternative to the construction presented in Section 3.2, in the case of 1-Mismatch EDSM. We do so by finding matches in a tree containing both suffixes of P and elements from the segment $\tilde{T}[i]$, as well as modified versions of those strings. The number of additional strings is bounded by using the *heavy-light decomposition* of Sleator and Tarjan [37]. The construction is directly inspired by the one presented by Thankachan et al. in [38], which is itself inspired by the *k-errata tree* construction introduced by Cole et al. in [17] for indexing with errors. We give an algorithm to find all occurrences of P in $\tilde{T}$

with 1 mismatch by computing sets 1-AP_i under Hamming distance, which, combined with the previously developed techniques, results in solving the 1-Mismatch EDSM problem in $\mathcal{O}(nm^2 + N \log m)$ time.

Let us start with the following basic definition.

Definition 3 ([37])**.** Let $\mathcal{T}$ be a rooted tree. The *heavy path* of $\mathcal{T}$ is the path that starts at the root and at each node descends to the child (called *heavy node*) with the largest number of leaf nodes in its subtree (ties are broken arbitrarily). The *heavy-light decomposition* of $\mathcal{T}$ is defined recursively as a union of the heavy path of $\mathcal{T}$ and the heavy path decompositions of the off-path subtrees of the heavy path. The nodes that are not heavy nodes are called *light nodes* (the root of $\mathcal{T}$ is always a light node). An edge on a heavy path is called *heavy*; and the other edges are called *light*.

A crucial property following from Definition 3 is that any root-to-leaf path crosses $\mathcal{O}(\log |\mathcal{T}|)$ paths. Each light edge on a path from the root decreases the size of the descending subtree by at least half. Thus the number of light edges on a path from any node to the root is $\mathcal{O}(\log |\mathcal{T}|)$.

We use the above properties to efficiently construct a tree $\mathcal{T}_1(P, \tilde{T}[i])$ (for a given ED text $\tilde{T}[1\,..\,n]$ of size N, a pattern $P[1\,..\,m]$ and an index $1 \leq i \leq n$ with $||T[i]|| = N_i$) in three steps (inspect Figure 5):

Step 1 We construct the compacted trie containing the strings in $\tilde{T}[i]$ and suffixes $P[j+1\,..\,m]$ of P for each $j \in AP_{i-1}$. We call this set of suffixes of P $act_{i-1}(P)$. We also add labels $(\iota(X), \#)$ to each node in the tree corresponding to a string X in $act_{i-1}(P) \cup \tilde{T}[i]$, where $\iota(X)$ is a pointer to X and $\#$ is a special label. This takes $\mathcal{O}(m + N_i)$ time and space [21] (we add the suffixes of P in $\mathcal{O}(m)$ total time by constructing the suffix tree of P and truncating the superfluous suffixes). We call $\mathcal{T}_0(P, \tilde{T}[i])$ the tree we obtain from this step. In the next steps it will be extended with new nodes and labels to obtain $\mathcal{T}_1(P, \tilde{T}[i])$.

Step 2 We compute a heavy-light decomposition [37] of $\mathcal{T}_0(P, \tilde{T}[i])$, which takes time linear in its size, namely $\mathcal{O}(m + N_i)$.

Step 3 For each light node u of $\mathcal{T}_0(P, \tilde{T}[i])$ let u' be the leaf on the heavy path starting at u. Leaf u' corresponds to a string X, and for each labeled descendant v of u outside of the heavy path $u \ldots u'$, if Y is the string corresponding to v, we compute $p = 1 + \mathsf{LCP}(X, Y)$ (in $\mathcal{O}(1)$ time after linear-time preprocessing of the tree for LCA queries [6]) and add to $\mathcal{T}_1(P, \tilde{T}[i])$ the string obtained from Y by replacing $Y[p]$ with $X[p]$, with a label $(\iota(Y), p)$ (a given node can store multiple labels). Intuitively, p is the position of a *mismatch* between (a prefix of) X and (a prefix of) Y. Since the tree $\mathcal{T}_0(P, \tilde{T}[i])$ has $\mathcal{O}(m + N_i)$ nodes and each of them has $\mathcal{O}(\log(m + N_i))$ light node ancestors, there are no more than $\mathcal{O}((m + N_i)\log(m + N_i))$ additional nodes and labels. Also the construction of new nodes can be done each time in $\mathcal{O}(1)$ time, because we in fact just copy a subtree of a light node and merge it with the subtree of its heavy sibling. We have thus arrived at the following lemma.

Lemma 12. *The construction of $\mathcal{T}_1(P, \tilde{T}[i])$ takes $\mathcal{O}((m + N_i)\log(m + N_i))$ time and space.*

We now prove that the tree $\mathcal{T}_1(P, \tilde{T}[i])$ satisfies the following property.

Lemma 13. *Let $X \in act_{i-1}(P)$. A string $Y \in \tilde{T}[i]$ is at Hamming distance at most 1 from a prefix of X having length $|Y|$ if and only if $\mathcal{T}_1(P, \tilde{T}[i])$ contains two nodes u, v respectively labeled by $(\iota(X), p)$ and $(\iota(Y), p')$, for some $p, p' \in \mathbb{N} \cup \{\#\}$, such that u is a descendant of v, and one of the following is satisfied:*

- $p = p' \in \mathbb{N}$
- $p = \#$ *or* $p' = \#$.

Proof. For the forward implication, if Y is a prefix of X then the claim is trivial since $\mathcal{T}_0(P, \tilde{T}[i])$ contains nodes with labels $(\iota(X), \#)$ and $(\iota(Y), \#)$, and thus the first node is a descendant of the second one. Now, we assume that Y has one mismatch with $X' = X[1\,..\,|Y|]$ at a position p. Let u, v be nodes in $\mathcal{T}_0(P, \tilde{T}[i])$ respectively corresponding to X and Y, and let w be their lowest common light ancestor. Let Z be the

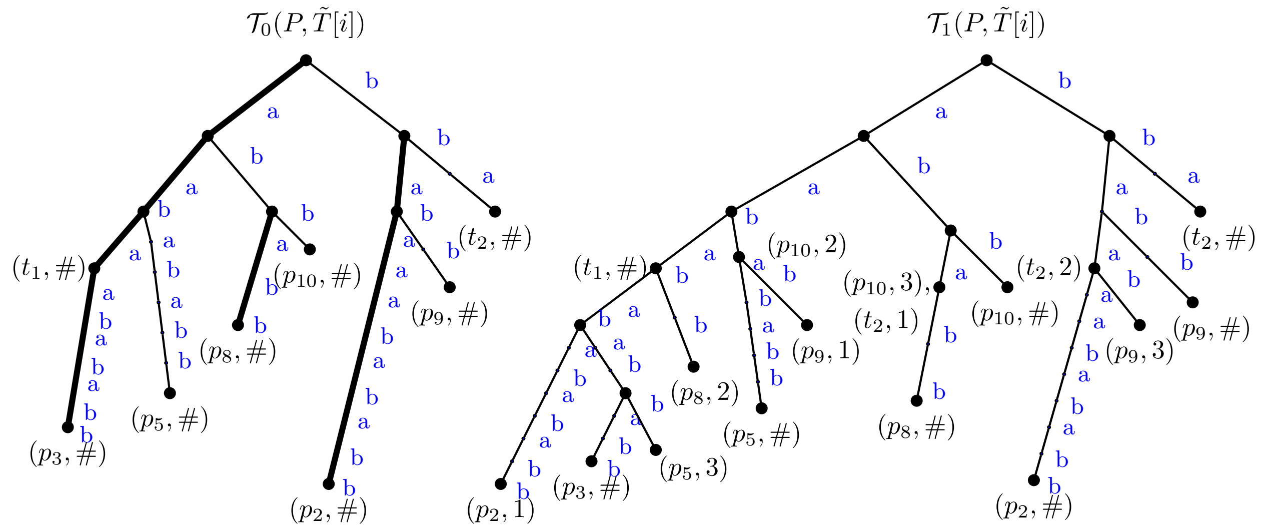


Figure 5: $\mathcal{T}_0(P, \tilde{T}[i])$ and $\mathcal{T}_1(P, \tilde{T}[i])$ for the example from Figure 4 ($P = bbaaaababab b$, $AP_{i-1} = \{1, 2, 4, 7, 8, 9\}$, $\tilde{T}[i] = \{aaa, bba\}$) with labels ($p_j = \iota(P[j \mathinner{.\,.} m]), t_j = \iota(\tilde{T}[i][j])$) and heavy paths.

string corresponding to the leaf on the heavy path starting at the heavy child of w. Since X and Y have a mismatch at position p, at least one of them has a mismatch with Z at position p and there are no mismatches to the left of p. Indeed, suppose towards a contradiction that there exists some $p' < p$ such that $Z[p'] \neq X[p'](= Y[p'])$: then the node corresponding to $X[1 \mathinner{.\,.} p'](= Y[1 \mathinner{.\,.} p'])$ would not be on the heavy path corresponding to Z, but would be a common ancestor of u and v, and thus w would not be the lowest common light ancestor of u and v, a contradiction.

Assume first that $X[p] \neq Z[p]$. Then, there is a node with a label $(\iota(X), p)$ in the tree, which is a descendant of either v, having label $(\iota(Y), \#)$ (if $Y[p] = Z[p]$), or a node having label $(\iota(Y), p)$ (if $Y[p] \neq Z[p]$), because we assumed that X and Y do not have any other mismatch. Finally, if $X[p] = Z[p]$, then $Y[p] \neq Z[p]$ and the node with label $(\iota(Y), p)$ is an ancestor of u, having label $(\iota(X), \#)$.

To prove the reverse implication, let us assume that the consequences are satisfied. Let u be the node whose label contains $(\iota(X), p)$ and v the node whose label contains $(\iota(Y), p')$. We first assume $p = p' \in \mathbb{N}$. Note that, by the construction of $\mathcal{T}_1(P, \tilde{T}[i])$, the node u (resp. v) corresponds to a string obtained by one letter modification on X (resp. on Y) at the same position p. We denote the resulting string $\hat{X}$ (resp $\hat{Y}$). Since v is an ancestor of u in $\mathcal{T}_1(P, \tilde{T}[i])$, $\hat{Y}$ is a prefix of $\hat{X}$. But this exactly means that Y has Hamming distance 1 with the length $|Y|$ prefix of X (or Hamming distance 0 if both replacements replaced the same letter). If the second condition is satisfied, namely if $p = \#$ or $p' = \#$, then it means that one replacement in Y gives $\hat{Y}$ which is a prefix of X, or that Y is a prefix of $\hat{X}$, which is one replacement away from X, therefore we have the claimed result. $\square$

We next formalize how to find nodes satisfying one of the conditions from Lemma 13 and deduce the approximate active prefixes corresponding to the Anchor Case for segment $\tilde{T}[i]$. Let v_1 `OR` v_2 denote a bitwise OR of two vectors, and $v_1 \oplus x$ denote vector v_1 shifted by x positions to the right (the first x positions are set to 0).

Proposition 5. *Algorithm 1 with input $\mathcal{T}_1(P, \tilde{T}[i])$ returns V_{res} such that $V_{res}[p] = 1$ if and only if p is an element of 1-AP_i corresponding to the Anchor Case for segment $\tilde{T}[i]$. Algorithm 1 runs in $\mathcal{O}((m + N_i)\log(m + N_i) + m^2)$ time.*

Proof. We first need the following remark: if $P[1 \mathinner{.\,.} k]$ extends into $P[1 \mathinner{.\,.} k']$ in $\tilde{T}[i]$, that means that some $Y \in \tilde{T}[i]$ is at Hamming distance 1 from the prefix $P[k+1 \mathinner{.\,.} k']$ of $P[k+1 \mathinner{.\,.} |P|]$. Therefore, we are looking

Algorithm 1 Search($\mathcal{T}$)

```
1: Global variables: the set $act_{i-1}(P)$, a segment $\tilde{T}[i]$, and bit vectors $V_{\#}$, $V_1$, $\ldots$, $V_m$, $V_{ANY}$, $V_{res}$ all of
   size $|P|+1$, and initially set to all 0's
2: Input: $\mathcal{T}$ - a subtree of $\mathcal{T}_1(P, \tilde{T}[i])$ with root $r$
3: Output: represented by global bit vector $V_{res}$
4: for each label $(\iota(X), p)$ with $X \in \tilde{T}[i]$ on $r$ do
5:     set $V_p[|X|]$ and $V_{ANY}[|X|]$ to 1
6: for each label $(\iota(X), p)$ with $X \in act_{i-1}(P)$ on $r$ do
7:     if $p = \#$ then
8:         update $V_{res}$ to $V_{res}$ OR $(V_{ANY} \oplus (m - |X|))$.
9:     else update $V_{res}$ to $V_{res}$ OR $((V_p$ OR $V_{\#}) \oplus (m - |X|))$
10: for each $\mathcal{T}'$ a subtree of $\mathcal{T}$ rooted at a child of $r$ do
11:     run Search($\mathcal{T}'$)
12: for each label $(\iota(X), p)$ with $X \in \tilde{T}[i]$ on $r$ do
13:     set $V_p[|X|]$ and $V_{ANY}[|X|]$ to 0
```

for the pairs described in Lemma 13. We show then that $V_{res}[k'] = 1$ after the end of the procedure if and only if there is a pair of nodes u, v in $\mathcal{T}_1(P, \tilde{T}[i])$ satisfying the conditions of Lemma 13 for $X = P[k+1\,..\,|P|]$, $Y \in \tilde{T}[i]$, and $|Y| = k' - k$.

Let us assume the existence of such a pair (u, v). Since the tree is traversed in a DFS, the node v (with a label $(\iota(Y), p')$, $p' \in \mathbb{N} \cup \{\#\}$) is traversed before u, which is its descendant; and at this moment, $V_{p'}[|Y|]$ is set to 1, as well as $V_{ANY}[|Y|]$. Since u is a descendant of v, the vectors are not modified at position $|Y|$ until u is visited: that would mean that v has a strict descendant representing a string of the same length as the string represented by v. When the label $(\iota(X), p)$ for $X \in act_{i-1}(P)$ is visited on u, we set the position $(m - |X|) + |Y| = k'$ of V_{res} to 1 if $V_p[|Y|] = 1$ or $V_{\#}[|Y|] = 1$ (which happens if $p = p' \in \mathbb{N} \cup \{\#\}$ or if $p' = \#$) and when $p = \#$ if $V_{ANY}[|Y|] = 1$.

Vice versa, if after the processing one has $V_{res}[k'] = 1$, this means that at some point in the DFS a node u having a label $(\iota(X), p)$ with $X \in act_{i-1}(P)$ and $p \in \mathbb{N} \cup \#$ was visited, and that at this point, for $k = m - |X|$, one had $V_{ANY}[k' - k] = 1$ or $V_{p'}[k' - k] = 1$ for (p, p') satisfying the conditions from Lemma 13. This one had to be set previously in the DFS at a node v having label $(\iota(Y), p')$ for $Y \in \tilde{T}[i]$ with $|Y| = k' - k$. Finally, an ancestor of u can be chosen as such v, because otherwise, from the DFS traversal order, the corresponding component of the vectors would have been set to 0. Now, the pair of nodes (u, v) satisfy the conditions of Lemma 13, and from our observations that means that there is an active prefix with 1 error of P having length k, extending up to $\tilde{T}[i]$.

The running time follows from the fact that the algorithm visits only $\mathcal{O}((m + N_i)\log(m + N_i))$ labels by Lemma 12, and from the fact that the tree is traversed in a DFS. The analysis of each label consists in reading it and doing a constant number of bit modifications in the stored vectors, and, for $\mathcal{O}(m\log(m+N_i))$ of them (the one corresponding to a suffix of P), doing an OR operation which takes $\mathcal{O}(\frac{m}{\log(N+m)})$ time in the word RAM model. This gives us the required running time. ☐

Corollary 6. 1-Mismatch EDSM *can be solved in* $\mathcal{O}(nm^2 + N\log m)$ *time.*

Proof. We proceed in the same way as in the reporting version of Section 3.5; the only difference is that, when $N_i \leq m^3$, to extend AP_{i-1} into 1-AP_i, instead of using the $\mathcal{O}((m^2 + N_i)\log m)$-time algorithm, we use the one from Proposition 5. Due to this change, the algorithm runs in the desired time. Indeed, notice that when $N_i \leq m^3$, $\mathcal{O}((N_i + m)\log(m + N_i) + m^2) = \mathcal{O}(N_i \log m + m^2)$, and when $N_i \geq m^3$, $\mathcal{O}(m^3 + N_i) = \mathcal{O}(N_i)$. The total time is thus $\mathcal{O}(nm^2 + N\log m)$. ☐

5 Open Questions

While our techniques (Sections 3 and 4) seem to generalize relatively easily to k errors, they would incur some exponential factor with respect to k. We leave the following basic questions open:

1. Can we design an $\mathcal{O}(nm^2 + N)$-time algorithm for 1-EDSM under edit or Hamming distance?
2. Can our techniques be efficiently generalized for $k > 1$ errors or mismatches?
3. Can our Hamming distance improvement for 1 mismatch (Section 4) be extended to edit distance?

References

[1] Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Degenerate string comparison and applications. In Laxmi Parida and Esko Ukkonen, editors, *18th International Workshop on Algorithms in Bioinformatics, WABI 2018, August 20-22, 2018, Helsinki, Finland*, volume 113 of *LIPIcs*, pages 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] Mai Alzamel, Lorraine A. K. Ayad, Giulia Bernardini, Roberto Grossi, Costas S. Iliopoulos, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Comparing degenerate strings. *Fundam. Informaticae*, 175(1-4):41–58, 2020.

[3] Amihood Amir, Dmitry Keselman, Gad M. Landau, Moshe Lewenstein, Noa Lewenstein, and Michael Rodeh. Text indexing and dictionary matching with one error. *J. Algorithms*, 37(2):309–325, 2000.

[4] Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *J. Algorithms*, 50(2):257–275, 2004.

[5] Kotaro Aoyama, Yuto Nakashima, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster online elastic degenerate string matching. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPIcs*, pages 9:1–9:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[6] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000.

[7] Giulia Bernardini, Pawel Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[8] Giulia Bernardini, Paweł Gawrychowski, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Elastic-degenerate string matching via fast matrix multiplication. *SIAM Journal on Computing*, 51(3):549–576, 2022.

[9] Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.*, 812:109–122, 2020.

[10] Vincenzo Carletti, Pasquale Foggia, Erik Garrison, Luca Greco, Pierluigi Ritrovato, and Mario Vento. Graph-based representations for supporting genome data analysis and visualization: Opportunities and challenges. In Donatello Conte, Jean-Yves Ramel, and Pasquale Foggia, editors, *Graph-Based*

Representations in Pattern Recognition - 12th IAPR-TC-15 International Workshop, GbRPR 2019, Tours, France, June 19-21, 2019, Proceedings, volume 11510 of *Lecture Notes in Computer Science*, pages 237–246. Springer, 2019.

[11] Timothy M. Chan, Kasper Green Larsen, and Mihai Patrascu. Orthogonal range searching on the RAM, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011.

[12] Panagiotis Charalampopoulos, Costas S. Iliopoulos, Chang Liu, and Solon P. Pissis. Property suffix array with applications in indexing weighted sequences. *ACM J. Exp. Algorithmics*, 25:1–16, 2020.

[13] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 978–989. IEEE, 2020.

[14] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster pattern matching under edit distance. *CoRR*, abs/2204.03087, 2022. (announced at FOCS 2022).

[15] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

[16] Aleksander Cislak, Szymon Grabowski, and Jan Holub. SOPanG: online text searching over a pan-genome. *Bioinform.*, 34(24):4290–4292, 2018.

[17] Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004.

[18] Richard Cole and Ramesh Hariharan. Approximate string matching: A simpler faster algorithm. *SIAM J. Comput.*, 31(6):1761–1782, 2002.

[19] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings.* Cambridge University Press, 2007.

[20] Massimo Equi, Tuukka Norri, Jarno Alanko, Bastien Cazaux, Alexandru I. Tomescu, and Veli Mäkinen. Algorithms and complexity on indexing elastic founder graphs. In Hee-Kap Ahn and Kunihiko Sadakane, editors, *32nd International Symposium on Algorithms and Computation, ISAAC 2021, December 6-8, 2021, Fukuoka, Japan*, volume 212 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[21] Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997.

[22] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[23] Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 54:1–54:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[24] Pawel Gawrychowski, Samah Ghazawi, and Gad M. Landau. On indeterminate strings matching. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPIcs*, pages 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[25] Pawel Gawrychowski and Przemyslaw Uznanski. Towards unified approximate pattern matching for Hamming and l_1 distance. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[26] Daniel Gibney. An efficient elastic-degenerate text index? not likely. In Christina Boucher and Sharma V. Thankachan, editors, *String Processing and Information Retrieval - 27th International Symposium, SPIRE 2020, Orlando, FL, USA, October 13-15, 2020, Proceedings*, volume 12303 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2020.

[27] Roberto Grossi, Costas S. Iliopoulos, Chang Liu, Nadia Pisanti, Solon P. Pissis, Ahmad Retha, Giovanna Rosone, Fatima Vayani, and Luca Versari. On-line pattern matching on similar texts. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPIcs*, pages 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[28] Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Inf. Comput.*, 279:104616, 2021.

[29] IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970.

[30] Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theor. Comput. Sci.*, 43:239–249, 1986.

[31] Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *J. Comput. Syst. Sci.*, 37(1):63–78, 1988.

[32] Veli Mäkinen, Bastien Cazaux, Massimo Equi, Tuukka Norri, and Alexandru I. Tomescu. Linear time construction of indexable founder block graphs. In Carl Kingsford and Nadia Pisanti, editors, *20th International Workshop on Algorithms in Bioinformatics, WABI 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 172 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[33] Joong Chae Na, Alberto Apostolico, Costas S. Iliopoulos, and Kunsoo Park. Truncated suffix trees and their application to data compression. *Theor. Comput. Sci.*, 304(1-3):87–101, 2003.

[34] Solon P. Pissis and Ahmad Retha. Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*, volume 103 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[35] Milan Ruzic. Constructing efficient dictionaries in close to sorting time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I: Tack A: Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2008.

[36] Qingmin Shi and Joseph F. JáJá. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM J. Comput.*, 34(6):1474–1492, 2005.

[37] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

[38] Sharma V. Thankachan, Alberto Apostolico, and Srinivas Aluru. A provably efficient algorithm for the k-mismatch average common substring problem. *Journal of Computational Biology*, 23(6):472–482, June 2016.

[39] The Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in Bioinformatics*, 19(1):118–135, 2018.