


Faster Approximate Elastic-Degenerate String Matching – Part A


Solon P. Pissis 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski 

Institute of Informatics, University of Warsaw, Poland

Wiktor Zuba 

Institute of Informatics, University of Warsaw, Poland

Abstract

An elastic-degenerate (ED) string \mathbf{T} is a sequence $\mathbf{T} = \mathbf{T}[1] \cdots \mathbf{T}[n]$ of n finite sets of strings. The cardinality m of \mathbf{T} is the total number of strings in $\mathbf{T}[i]$, for all $i \in [1..n]$. The size N of \mathbf{T} is the total length of all m strings of \mathbf{T} . ED strings have been introduced to represent a set of closely-related DNA sequences. Let $P = P[1..p]$ be a pattern of length p and $k > 0$ be an integer. We consider the problem of k -Approximate ED String Matching (EDSM): searching k -approximate occurrences of P in the language of \mathbf{T} . We call k -Approximate EDSM under the Hamming distance, k -Mismatch EDSM; and we call k -Approximate EDSM under edit distance, k -Edit EDSM.

Bernardini et al. (*Theoretical Computer Science*, 2020) showed a simple $\mathcal{O}(kmp + kN)$ -time algorithm for k -Mismatch EDSM and an $\mathcal{O}(k^2mp + kN)$ -time algorithm for k -Edit EDSM. We improve the dependency on k in both results, obtaining an $\tilde{\mathcal{O}}(k^{2/3}mp + \sqrt{k}N)$ -time algorithm for k -Mismatch EDSM and an $\tilde{\mathcal{O}}(kmp + kN)$ -time algorithm for k -Edit EDSM.

Bernardini et al. (*Theory of Computing Systems*, 2024) presented several algorithms for 1-Approximate EDSM working in $\tilde{\mathcal{O}}(np^2 + N)$ time. They have also left the possibility to generalize these solutions for $k > 1$ as an open problem. We improve the runtime of their solution for 1-Mismatch and 1-Edit EDSM from $\tilde{\mathcal{O}}(np^2 + N)$ to $\mathcal{O}(np^2 + N)$. We further show algorithms for k -Approximate EDSM for the Hamming and edit distances working in $\tilde{\mathcal{O}}(np^2 + N)$ time, for any constant $k > 0$.

Finally, we show how our techniques can be applied to improve upon the complexity of the k -Approximate ED String Intersection and k -Approximate Doubly EDSM problems that were introduced very recently by Gabory et al. (*Information and Computation*, 2025).

2012 ACM Subject Classification Theory of computation \rightarrow Pattern matching

Keywords and phrases ED string, approximate string matching, Hamming distance, edit distance

Digital Object Identifier 10.4230/LIPIcs.CPM.2025.28

Funding *Solon P. Pissis*: Supported in part by 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.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2022/46/E/ST6/00463.

1 Introduction

Elastic-degenerate strings (ED strings, in short) were introduced in [23] to represent a set of closely-related DNA sequences. They allow, among others, efficient pattern matching [22, 2, 7] and approximate pattern matching [5, 6, 8], whereas pattern matching in general graph-based representations cannot be done as fast [16, 3]. An ED string usually arises from a multiple sequence alignment (MSA) of the underlying closely-related DNA sequences. It can also be treated as a compacted nondeterministic finite automaton (NFA); see Figure 1.



© Solon P. Pissis, Jakub Radoszewski, and Wiktor Zuba;
licensed under Creative Commons License CC-BY 4.0

36th Annual Symposium on Combinatorial Pattern Matching (CPM 2025).

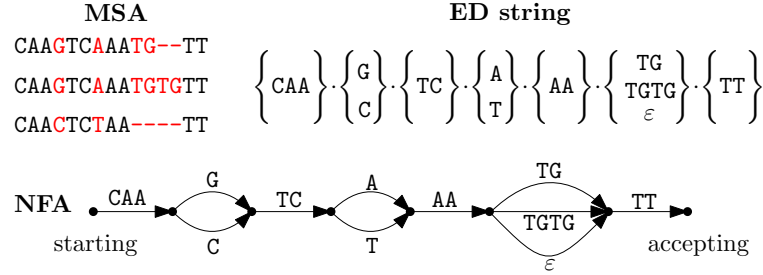
Editors: Paola Bonizzoni and Veli Mäkinen; Article No. 28; pp. 28:1–28:19

Leibniz International Proceedings in Informatics



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

Let Σ be a totally-ordered *alphabet*. An *ED string* \mathbf{T} is a sequence $\mathbf{T} = \mathbf{T}[1] \cdots \mathbf{T}[n]$ of n finite sets, where $\mathbf{T}[i]$ is a subset of Σ^* . The total size of \mathbf{T} is defined as $N = N_\varepsilon + \sum_{i=1}^n \sum_{S \in \mathbf{T}[i]} |S|$, where N_ε is the total number of empty strings in \mathbf{T} . By m we denote the total number of strings in all $\mathbf{T}[i]$, i.e., $m = \sum_{i=1}^n |\mathbf{T}[i]|$. We say that \mathbf{T} has *length* $n = |\mathbf{T}|$, *cardinality* m and *size* N . The *language* $\mathcal{L}(\mathbf{T})$ generated by the ED string \mathbf{T} is $\mathcal{L}(\mathbf{T}) = \{S_1 \cdots S_n : S_i \in \mathbf{T}[i] \text{ for all } i \in [1..n]\}$. A standard string can be represented as an ED string in which each set is a singleton of a character.



■ **Figure 1** An example of an MSA of three strings and the corresponding (non-unique) ED string \mathbf{T} of length $n = 7$, cardinality $m = 11$ and size $N = 20$, and the compacted NFA for \mathbf{T} . The compacted NFA can also be seen as a special case of an edge-labeled directed acyclic graph.

The *Hamming distance* δ_H of two equal-length strings U and V is defined as the number of positions where the two strings do not match, that is, $\delta_H(U, V) = |\{i \in [1..|U|] : U[i] \neq V[i]\}|$. The *edit distance* δ_E (a.k.a. the Levenshtein distance) of U and V is the minimum number of edit operations (single-character insertions, deletions, substitutions) that allow to transform U to V . We say that a substring S of a string text T is a *k-approximate occurrence* of a string pattern P under the Hamming distance (edit distance) if the Hamming (edit) distance of S and P is at most k . The problem in scope is now stated as follows.

k -APPROXIMATE EDSM

Input: An ED string \mathbf{T} (called *text*) of length n , cardinality m and size N , a string P (called *pattern*) of length p over an alphabet of size σ , and a metric (Hamming or edit distance).

Output: **Decision version:** YES if there is a string $S \in \mathcal{L}(\mathbf{T})$ that contains a k -approximate occurrence of P under the respective metric, and NO otherwise.

Reporting version: The set of positions $i \in [1..n]$ for which there exists a string $V \in \mathbf{T}[i]$ and a (possibly empty) string $U \in \mathcal{L}(\mathbf{T}[1..i])$ such that P has a k -approximate occurrence in string UV that ends at some position $j > |U|$ in UV .

We call the k -Approximate EDSM problem under the Hamming distance, *k-Mismatch EDSM*, and under the edit distance, *k-Edit EDSM* (inspect Figure 2 for an example).

Exact EDSM (i.e., 0-Approximate EDSM) was considered by Grossi et al. [22], Aoyama et al. [2], and Bernardini et al. [7], who presented $\mathcal{O}(np^2 + N)$, $\tilde{\mathcal{O}}(np^{1.5} + N)$, and $\tilde{\mathcal{O}}(np^{\omega-1} + N)$ time algorithms, respectively (here ω is the exponent of matrix multiplication). A practical approach using bit parallelism was proposed by Cislak et al. [13]. k -Approximate EDSM under the Hamming and edit distances was considered by Bernardini et al. in [8] (general k) and in [6] ($k = 1$). Below we elaborate on these two results and present our improvements.

Patterns	ED string	Matches
AATTTC	$\mathbf{T} = \left\{ \begin{smallmatrix} \text{C} & \text{A} & \text{A} \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{G} \\ \text{C} \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{T} & \text{C} \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{A} \\ \text{T} \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{A} & \text{A} \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{T} & \text{G} \\ \text{T} & \text{G} & \text{T} & \text{G} \\ \varepsilon \end{smallmatrix} \right\} \cdot \left\{ \begin{smallmatrix} \text{T} & \text{T} \end{smallmatrix} \right\}$	$\delta_E(\text{AATTTC}, \text{AAGTC}) = 2$
ATGAGT		$\delta_H(\text{ATGAGT}, \text{ATGTGT}) = 1$

■ **Figure 2** An example of two approximate occurrences in \mathbf{T} : one for 2-Edit EDSM; and one for 1-Mismatch EDSM. For $P = \text{AATTTC}$, we have a 2-edit occurrence of P ending at $\mathbf{T}[3]$ ($V = \text{TC}$). For $P = \text{ATGAGT}$, we have a 1-mismatch occurrence of P ending at $\mathbf{T}[7]$ ($V = \text{TT}$).

Bernardini et al. [8] showed a simple $\mathcal{O}(kmp + kN)$ -time algorithm for k -Mismatch EDSM and an $\mathcal{O}(k^2mp + kN)$ -time algorithm for the k -Edit EDSM. We improve the dependency on k in both these results, obtaining an $\tilde{\mathcal{O}}(k^{2/3}mp + \sqrt{k}N)$ -time algorithm for k -Mismatch EDSM and an $\tilde{\mathcal{O}}(kmp + kN)$ -time algorithm for k -Edit EDSM.

■ **Table 1** Complexities of solutions for k -Approximate EDSM, suppressing $\log^{\mathcal{O}(1)}(N + p)$ factors.

metric	time complexity	remarks	reference
Hamming	$\tilde{\mathcal{O}}(kmp + kN)$		[8]
	$\tilde{\mathcal{O}}(k^{2/3}mp + \sqrt{k}N)$		Theorem 5
edit	$\tilde{\mathcal{O}}(k^2mp + kN)$		[8]
	$\tilde{\mathcal{O}}(kmp + kN)$		Theorem 10
Hamming	$\tilde{\mathcal{O}}(np^2 + N)$	$k = 1$	[5, 6]
edit			
Hamming	$\tilde{\mathcal{O}}(np^2 + N)$	$k = \mathcal{O}(1)$	Theorem 14
edit			

Bernardini et al. [5, 6] presented several algorithms for 1-Approximate EDSM working in $\tilde{\mathcal{O}}(np^2 + N)$ time and in $\mathcal{O}(np^3 + N)$ time. In [6] they left three open questions, two related to improving the polylogarithmic factors in the complexity of 1-Mismatch and 1-Edit EDSM and one asking for an efficient generalization to $k > 1$ mismatches or errors. We answer all these open questions. We show how to avoid all the polylogarithmic factors in the complexity and present algorithms for 1-Approximate EDSM (both Hamming and edit) that work in $\mathcal{O}(np^2 + N)$ time. We also show an algorithm for k -Approximate EDSM for each of the metrics working in $\tilde{\mathcal{O}}(np^2 + N)$ time, for constant $k > 1$. Actually, there is a $\log^k m$ factor in the complexity; in [5, 6] it was mentioned that having an exponential dependency on k in this form of complexity for k -Approximate EDSM could be inevitable.

Let us note that the complexities of our algorithms for 1-Approximate EDSM match the complexity of the fastest currently known combinatorial algorithm (that is, without using fast Fourier transform (FFT) [2] or fast matrix multiplication [7]) for exact EDSM [22]. Moreover, in [7] a conditional lower bound for combinatorial algorithms for exact EDSM was given according to which exact EDSM cannot be solved in $\mathcal{O}(np^{1.5-\varepsilon} + N)$ time, for constant $\varepsilon > 0$. The $\mathcal{O}(np^2 + N)$ bound for $k = 1$ and the $\tilde{\mathcal{O}}(np^2 + N)$ bound for any constant $k > 1$ that we show here work for the Hamming and the edit distance. In Part B, it is shown how to improve both bounds for the Hamming distance by resorting to FFTs [21].

A summary of the existing results and our results for k -Approximate EDSM is shown in Tables 1 and 2. We consider the word RAM model of computation and assume that the ED string \mathbf{T} and the string P are over an integer alphabet $[0 \dots (N + p)^{\mathcal{O}(1)}]$. All algorithms mentioned in the tables process the text ED string on-line: after each set $\mathbf{T}[i]$ is processed, we report YES if the string pattern P has a k -approximate occurrence ending at set $\mathbf{T}[i]$.

■ **Table 2** Exact complexities of solutions for k -Approximate EDSM assuming integer alphabet.

metric	time complexity	remarks	reference
Hamming	$\mathcal{O}(kmp + kN)$	$+\mathcal{O}(p \log \log p)$ or expected time	[8]
	$\mathcal{O}(k^{2/3}mp \log^{1/3} p + N\sqrt{k \log k})$		Theorem 5
	$\mathcal{O}(k^{2/3}mp \log^{1/3} p + N\sqrt{k})$	expected time	
edit	$\mathcal{O}(k^2mp + kN)$	$+\mathcal{O}(p \log \log p)$	[8]
	$\mathcal{O}(kmp \log k / \log \log k + kN)$	or expected time	Theorem 10
Hamming	$\mathcal{O}(np^2 + N \log p)$	$k = 1$	[6]
	$\mathcal{O}(np^3 + N)$		
edit	$\mathcal{O}((np^2 + N) \log p)$		
	$\mathcal{O}(np^3 + N)$		
	$\mathcal{O}(np^2 \sqrt{\log p} + N \log \log p)$, decision ver.		
Hamming	$\mathcal{O}(np^2 + N)$		Theorem 20
edit			
Hamming edit	$\mathcal{O}(N + (m + np \log \log m + np^2) \log^k m)$	$k = \mathcal{O}(1)$	Theorem 14

Finally, we show how our techniques can be applied to improve upon the complexity of the k -Approximate ED String Intersection (k -Approximate EDSI) and k -Approximate Doubly EDSM problems that were introduced very recently by Gabory et al. [19]. In both problems, the input are two ED strings \mathbf{T}_1 and \mathbf{T}_2 such that \mathbf{T}_i has cardinality m_i and size N_i . In k -Approximate EDSI, we would like to check if there are strings $S_1 \in \mathcal{L}(\mathbf{T}_1)$ and $S_2 \in \mathcal{L}(\mathbf{T}_2)$ at distance (Hamming or edit) at most k . k -Approximate Doubly EDSM, at least in its decision version, consists in k -Approximate EDSM for all string patterns from $\mathcal{L}(\mathbf{T}_2)$ in \mathbf{T}_1 . In [19], both problems were solved in $\mathcal{O}(k(N_1m_2 + N_2m_1))$ time for the Hamming distance and in $\mathcal{O}(k^2(N_1m_2 + N_2m_1))$ time for the edit distance. We obtain $\tilde{\mathcal{O}}(k^{2/3}(N_1m_2 + N_2m_1))$ time and $\tilde{\mathcal{O}}(k(N_1m_2 + N_2m_1))$ time for the Hamming and edit distance, respectively.

Comparison of Techniques. For k -Mismatch EDSM, Bernardini et al. [8] used kangaroo jumps [20, 25], whereas we use efficient k -Mismatch Pattern Matching [1, 11] and an algorithm for computing the so-called k th mismatch [24]. For k -Edit EDSM, Bernardini et al. used the Landau-Vishkin algorithm [26]; we apply the same algorithm and also Tiskin’s seaweeds [29], via an interface that was developed by Charalampopoulos et al. [12].

For 1-Approximate EDSM, Bernardini et al. [5, 6] used special instances of classic computational geometry problems to arrive at a time complexity of $\mathcal{O}((np^2 + N) \log p)$. For 1-Mismatch EDSM, in particular, they also introduced a variant of errata trees [14] to shave a logarithmic factor and obtain a running time of $\mathcal{O}(np^2 + N \log p)$. We first show that their technique based on errata trees can be extended to 1-Edit EDSM thus improving the runtime from $\mathcal{O}((np^2 + N) \log p)$ to $\mathcal{O}(np^2 + N \log p)$. Then we show a simple sorting-based algorithm working in $\mathcal{O}(npt^2 + N)$ time if all strings in \mathbf{T} are of length at most t . A trade-off between the two algorithms lets us shave the remaining $\log p$ factor.

Similarly as in [6], our main tool for extending 1-Approximate EDSM to k -Approximate EDSM for $k > 1$ are errata trees [14]; however, we manage to apply them as a black-box.

Roadmap. In Section 2, we discuss the necessary preliminaries. In Section 3, we prove Theorems 5 and 10. In Section 4, we prove Theorem 14. In Section 5, we prove Theorem 20. In Section 6, we show our results on k -Approximate EDSI and k -Approximate Doubly EDSM.

2 Preliminaries

We use the notion of active prefix (and suffix) that was already present in [7] (exact) and [5] (1-approximate). We generalize this notion in a natural way to a k -approximate version.

► **Definition 1.** We say that $P[1..j]$ is a k -active prefix of string P ending at position i in an ED string \mathbf{T} if $P[1..j]$ is at distance at most k from a suffix of a string in $\mathcal{L}(\mathbf{T}[1..i])$.

Symmetrically, we say that $P[j..p]$ is a k -active suffix of P starting at position i in an ED string \mathbf{T} if $P[j..p]$ is at distance at most k from a prefix of a string in $\mathcal{L}(\mathbf{T}[i..n])$.

By $\text{AP}_i^k(\mathbf{T})[0..p]$ we denote an array of values in $[0..k] \cup \{\infty\}$. We have $\text{AP}_i^k(\mathbf{T})[\ell] \neq \infty$ if and only if $P[1..\ell]$ is a k -active prefix ending at position i in \mathbf{T} and then $\text{AP}_i^k(\mathbf{T})[\ell]$ is the minimum distance between $P[1..\ell]$ and a suffix of a string in $\mathcal{L}(\mathbf{T}[1..i])$. By definition, $\text{AP}_0^k(\mathbf{T}) = (0, \infty, \dots, \infty)$ for the Hamming distance and $\text{AP}_0^k(\mathbf{T}) = (0, 1, 2, \dots, k, \infty, \dots, \infty)$ (or $(0, 1, \dots, p)$ for $p \leq k$) for the edit distance. Symmetrically, we define $\text{AS}_i^k(\mathbf{T})$ as an array of values in $[0..k] \cup \{\infty\}$ such that $\text{AS}_i^k(\mathbf{T})[\ell] \neq \infty$ if and only if $P[\ell..p]$ is a k -active suffix starting at position i in \mathbf{T} and then $\text{AS}_i^k(\mathbf{T})[\ell]$ is the minimum distance between $P[\ell..p]$ and a prefix of a string in $\mathcal{L}(\mathbf{T}[i..n])$. We make the following fundamental observation.

► **Observation 2.** A string pattern P of length p has a k -approximate occurrence (under the Hamming or the edit distance) ending at position i of an ED string \mathbf{T} if P has a k -approximate occurrence (under the respective distance) in one of the strings in $\mathbf{T}[i]$ or there is some index $\ell \in [0..p]$ such that $\text{AP}_{i-1}^k(\mathbf{T})[\ell] + \text{AS}_1^k(\mathbf{T}[i])[\ell+1] \leq k$ (again, under the respective distance).

By Observation 2, there are four essential steps to design an on-line algorithm for k -Approximate EDSM:

- (1) For every $i \in [1..n]$ given on-line, check if the string pattern P has a k -mismatch occurrence in one of the strings in $\mathbf{T}[i]$.
- (2) For every $i \in [1..n]$ given on-line, compute the arrays $\text{AP}_1^k(\mathbf{T}[i])$, $\text{AS}_1^k(\mathbf{T}[i])$ of active prefixes and suffixes produced by only this set of the ED string.
- (3) For every $i \in [1..n]$ given on-line, compute the array $\text{AP}_i^k(\mathbf{T})$ using $\text{AP}_1^k(\mathbf{T}[i])$ and $\text{AP}_{i-1}^k(\mathbf{T})$ together with the set of k -approximate occurrences of strings from $\mathbf{T}[i]$ in P .
- (4) For every $i \in [1..n]$ given on-line, check if there is an index $\ell \in [0..p]$ such that $\text{AP}_{i-1}^k(\mathbf{T})[\ell] + \text{AS}_1^k(\mathbf{T}[i])[\ell+1] \leq k$. This step takes just $\mathcal{O}(p)$ time, for a total of $\mathcal{O}(np)$.

Similar steps were considered in previous work on k -Approximate EDSM. For instance, Bernardini et al. [6] introduced four cases for 1-Approximate EDSM: Easy Case that is exactly step (1), Suffix Case that is covered by step (2), Prefix Case covered by steps (2) and (4), and Anchor Case covered by step (3).

Let $m_i = |\mathbf{T}[i]|$ and $N_i = \sum_{S \in \mathbf{T}[i]} \max(|S|, 1)$. (Thus $m = \sum_{i=1}^n m_i$ and $N = \sum_{i=1}^n N_i$.)

Often we would like to construct a data structure for the strings in a given set $\mathbf{T}[i]$ and the string P that requires that the strings in $\mathbf{T}[i] \cup \{P\}$ are over an integer alphabet of size $(N_i + p)^{\mathcal{O}(1)}$ (this could be, for example, the generalized suffix tree; see [17]). However, in k -Approximate EDSM we are only guaranteed that the strings are over an integer alphabet of size $(N + p)^{\mathcal{O}(1)}$. As we aim at an on-line algorithm, we use a technique discussed in [8] in which an index of all characters of P is constructed. With the index we renumber all characters of P with integers in $[1..p]$ and then, for every character in a string in $\mathbf{T}[i]$, we check if it is contained in the index. If so, we assign the character the number of the character in the pattern in $[1..p]$, and if not, we assign it a number $p+1$. The index can be constructed using perfect hashing [18] in $\mathcal{O}(p)$ expected time or using a deterministic dictionary [27] in $\mathcal{O}(p \log \log p)$ time; in both cases, an index query is answered in $\mathcal{O}(1)$ time.

3 Approximate EDSM via Approximate Overlaps

Our solutions to k -Approximate EDSM depend on two building blocks: k -Approximate Pattern Matching; and computing k -approximate overlaps of two strings.

Let us consider the Hamming or edit distance. In the k -Approximate Pattern Matching (k -Approximate PM) problem, we are given two strings, a text T and a pattern P , and we are to compute all k -approximate occurrences $T[i..j]$ of P , i.e., such that P and $T[i..j]$ are at distance at most k (under the respective metric). In particular, we have $j - i = |P|$ under the Hamming distance and $j - i - |P| \in [-k..k]$ for the edit distance. The solutions to k -Approximate PM also report the distance (Hamming or edit) between a k -approximate occurrence $T[i..j]$ and P ; in this sense, they solve a more general k -Bounded PM problem.

We say that two strings U and V have a k -approximate overlap of length ℓ under the Hamming or edit distance if the length- ℓ prefix $U[1.. \ell]$ of U is at (Hamming or edit) distance at most k from some suffix of V . We define the k -Bounded Overlaps problem as follows.

k -BOUNDED OVERLAPS

Input: Two strings U and V and a metric (Hamming or edit distance).

Output: For every length $\ell \in [0.. \min(|U|, |V| + k)]$, the smallest $k' \leq k$ such that U and V have a k' -approximate overlap of length ℓ or NO if no such k' exists.

We impose a restriction $\ell \leq |V| + k$ since otherwise the answer for the edit distance would be clearly NO. Under the Hamming distance, we can further restrict the problem to $\ell \leq |V|$.

3.1 k -Mismatch EDSM

The k -Approximate PM problem under the Hamming distance is called k -Mismatch PM. For a string text T of length n , if P and T are over an integer alphabet of size $n^{\mathcal{O}(1)}$, k -Mismatch PM can be solved in $\mathcal{O}(n\sqrt{k \log k})$ time with a classic result by Amir et al. [1] or by a recent Monte Carlo algorithm of Chan et al. [11] in $\mathcal{O}(n\sqrt{k})$ expected time with high probability.

We call k -approximate overlaps under the Hamming distance k -mismatch overlaps. The k -Bounded Overlaps problem under Hamming distance was present implicitly in [8] where an $\mathcal{O}(nk)$ -time solution using kangaroo jumps was presented. We solve the k -Bounded Overlaps problem under Hamming distance more efficiently using the algorithm for the k th mismatch computation of Kaplan et al. [24].

In the k th Mismatch problem, we are given two strings P and T , $|P| \leq |T|$, and we are to determine, for every $i \in [1.. |T| - |P| + 1]$, if $T[i..i + |P|)$ matches P with at most k mismatches and, if not, to report the position of the k th mismatch in P . Kaplan et al. [24] presented a solution to the k th Mismatch problem working in $\mathcal{O}(|T|k^{2/3} \log^{1/3} |P|)$ time for strings P and T over an integer alphabet.

► **Lemma 3.** *The k -Bounded Overlaps problem under Hamming distance for two strings U and V such that $\min(|U|, |V|) = n$ and U, V are over an integer alphabet of size $n^{\mathcal{O}(1)}$ can be solved in $\mathcal{O}(nk^{2/3} \log^{1/3} n)$ time.*

Proof. Let U' be the prefix of U of length n and V' be the suffix of V of length n . By definition, if ℓ is a length of k -mismatch overlap of U and V , then $\ell \leq n$ and the approximately matching prefix of U and suffix of V are a prefix of U' and a suffix of V' , respectively. We construct strings $P = U' \#^{n+1}$ and $T = V' \$^{2n+1}$, where $\#, \$$ are symbols not occurring in U and V , and solve the $(k_0 + 1)$ th Mismatch problem for P and T , where $k_0 = \min(k, n)$.

▷ **Claim 4.** For every $\ell \in [1..n]$, $k' \in [0..k]$ is the smallest nonnegative integer such that U and V have a k' -mismatch overlap of length ℓ if and only if $k' \leq k_0$ and the $(k_0 + 1)$ th mismatch between $T' := T[|V'| + 1 - \ell .. |V'| + 1 - \ell + |P|]$ and P is at position $\ell + k_0 + 1 - k'$ in P .

Proof. (\Rightarrow) Assume that $\ell \in [1..|V'|]$ and

$$\delta_H(U[1..\ell], V[|V| - \ell + 1..|V|]) = \delta_H(U'[1..\ell], V'[|V'| - \ell + 1..|V'|]) = k',$$

with $k' \leq k$. We have $k' \leq \ell \leq n$, so $k' \leq k_0$. Then

$$\delta_H(P[1..\ell + k_0 + 1 - k'], V[|V| - \ell + 1..|V|]\$^{k_0+1-k'}) = k_0 + 1$$

where the second string ends with a $\$$ and $P[\ell + k_0 + 1 - k'] \neq \$$. The two above strings are prefixes of P and T' , respectively. Hence, the $(k_0 + 1)$ th mismatch between P and T' is at position $\ell + k_0 + 1 - k'$ in P , as required.

(\Leftarrow) Assume that $k' \leq k_0$ and the $(k_0 + 1)$ th mismatch between T' and P is at position $\ell + k_0 + 1 - k'$. By definition, the last $k_0 + 1 - k' \geq 1$ positions of T' and P do not match. This means that the remaining prefixes of the two strings, i.e., $U[1..\ell]$ and $V[|V| - \ell + 1..|V|]$, are at Hamming distance $k_0 + 1 - (k_0 + 1 - k') = k'$, which concludes the proof. ◀

By the claim, the solution to $(k_0 + 1)$ th Mismatch problem for P and T allows us to recover in $\mathcal{O}(n)$ time the answer to k -Bounded Overlap problem. We have $|P| + |T| = \mathcal{O}(n)$. The complexity follows by [24]. ◀

▶ **Theorem 5.** k -Mismatch EDSM can be solved on-line in $\mathcal{O}(pmk^{2/3} \log^{1/3} p + N\sqrt{k \log k})$ worst-case time or $\mathcal{O}(pmk^{2/3} \log^{1/3} p + N\sqrt{k})$ expected time with high probability.

Proof. We apply the general scheme of steps (1)–(4), according to Observation 2. In step (1), we check if the string pattern P has a k -mismatch occurrence in a string $S \in \mathbf{T}[i]$. If $|S| \geq p$, we apply the k -Mismatch PM algorithm for string pattern P and string text S . After the $\mathcal{O}(p \log \log p)$ -time preprocessing on P , both strings can be assumed to be over alphabet $[1..p + 1]$. Over all strings S , this takes $\mathcal{O}(N_i \sqrt{k \log k})$ worst-case time [1] or $\mathcal{O}(N_i \sqrt{k})$ expected time with high probability [11], which sums up over all $i \in [1..n]$ to $\mathcal{O}(N\sqrt{k \log k})$ worst case or $\mathcal{O}(N\sqrt{k})$ expected time.

In step (2), the array $\mathbf{AP}_1^k(\mathbf{T}[i])$ of active prefixes of a single set is computed using k -Bounded Overlaps. More precisely, for $U = P$ and every string $V \in \mathbf{T}[i]$, for every ℓ such that U and V have a k -mismatch overlap of length ℓ , we set $\mathbf{AP}_1^k(\mathbf{T}[i])[\ell]$ to $\delta_H(P[1..\ell], V[|V| - \ell + 1..|V|])$; otherwise $\mathbf{AP}_1^k(\mathbf{T}[i])[\ell]$ is set to ∞ . By Lemma 3, the time complexity, over all $V \in \mathbf{T}[i]$, is $\mathcal{O}(p \cdot m_i \cdot k^{2/3} \log^{1/3} p)$. We point out that renumbering the alphabet to integer to satisfy the requirements of Lemma 3, over all calls to k -Bounded Overlaps, takes $\mathcal{O}(mp \log \log p)$ time. The array $\mathbf{AS}_1^k(\mathbf{T}[i])$ of active suffixes is computed using a symmetric application of k -Bounded Overlaps in the same time complexity. This step is performed for all $i \in [1..n]$ in $\mathcal{O}(pmk^{2/3} \log^{1/3} p)$ total time.

In step (3), we compute $\mathbf{AP}_i^k(\mathbf{T})$, for $i \in [1..n]$ on-line. Initially $\mathbf{AP}_i^k(\mathbf{T}) = \mathbf{AP}_1^k(\mathbf{T}[i])$. For a given i , for all $S \in \mathbf{T}[i]$ such that $|S| \leq p$, we compute all k -mismatch occurrences of string pattern S in string text P . Using k -Mismatch PM, this takes $\mathcal{O}(pm_i \sqrt{k \log k})$ worst-case time [1], for a total of $\mathcal{O}(pm\sqrt{k \log k})$ time for all $i \in [1..n]$.

Whenever a k -mismatch occurrence $P[a..b]$ of $S \in \mathbf{T}[i]$ is discovered, we set $\mathbf{AP}_i^k(\mathbf{T})[b] := \min(\mathbf{AP}_i^k(\mathbf{T})[b], k' + k'')$ where $k' = \mathbf{AP}_{i-1}^k(\mathbf{T})[a - 1]$ and $k'' = \delta_H(P[a..b], S)$. In total, this takes $\mathcal{O}(pm_i)$ time for a given i , and $\mathcal{O}(pm)$ time overall.

Finally, in step (4) we check the condition of Observation 2 for each position $i \in [1..n]$ in $\mathcal{O}(np) = \mathcal{O}(mp)$ time. ◀

3.2 k -Edit EDSM

The k -Approximate PM problem under the edit distance is called k -Edit PM. For a string text of length n , if P and T are over an integer alphabet of size $n^{\mathcal{O}(1)}$, k -Edit PM can be solved in $\mathcal{O}(nk)$ time using the Landau-Vishkin algorithm [26].

We call k -approximate overlaps under the edit distance *k -edit overlaps*. We compute k -Bounded Overlaps under edit distance using a modification of the All- k -LPAM problem of Charalampopoulos et al. [12] (LPAM stands for Longest Prefix Approximate Match). In the original All- k -LPAM problem we are given a string text T of length n and a string pattern P of length p and we are to compute, for each $k' \in [0..k]$ and $i \in [0..n]$, the length of the longest prefix of P that matches a prefix of $T[i..n]$ with at most k' edits. We introduce a similar but different problem, All- k -BO (BO stands for Bounded Overlap), in which, given a string text T of length n and a string pattern P of length p , we are to compute for each $\ell \in [0..p]$ and $\ell' \in [\max(0, \ell - k) .. \min(n, \ell + k)]$, the edit distance between a length- ℓ prefix of P and a length- ℓ' suffix of T provided that it is at most k or state otherwise.

The output size for each of the problems All- k -LPAM and All- k -BO is $\mathcal{O}((n+p)k)$. The All- k -LPAM problem can be solved in $\mathcal{O}(nk \log^3 k)$ time [12]. We show that All- k -BO can be solved in $\tilde{\mathcal{O}}(nk)$ time using the techniques from [12] that we recall next.

The *deletion distance* $\delta_D(U, V)$ of two strings U and V is the minimum number of character insertions and deletions required to transform U to V (i.e., substitutions are not allowed). For a string S , by $S_\$$ we denote the string $S[1]\$S[2]\$ \cdots S[|S|]\$$. By the following fact, we can consider the deletion distance instead of the edit distance.

► **Fact 6** ([12, Fact 24]). *For any two strings U and V that do not contain the character $\$$, we have $2 \cdot \delta_E(U, V) = \delta_D(U_\$, V_\$)$.*

A *permutation matrix* is a square matrix over $\{0, 1\}$ that contains exactly one 1 in each row and in each column. A permutation matrix P of size $s \times s$ corresponds to a permutation π of $[0..s]$ such that $P[i, j] = 1$ if and only if $\pi(i) = j$. For two permutations π_1, π_2 and their corresponding permutation matrices P_1, P_2 , by $\Delta(\pi_1, \pi_2) = \Delta(P_1, P_2)$ we denote a shortest sequence of transpositions f_1, \dots, f_q of neighboring elements such that $f_q \circ \cdots \circ f_1 \circ \pi_1 = \pi_2$ (that is, each transposition swaps two adjacent columns of the maintained permutation matrix). Such a transposition of $\pi(i)$ and $\pi(i+1)$ is *ordered* if $\pi(i) < \pi(i+1)$. For an $s \times s$ matrix A , we denote by A^Σ an $(s+1) \times (s+1)$ matrix such that $A^\Sigma[i, j] = \sum_{i' \geq i} \sum_{j' < j} A[i', j']$ for $i, j \in [0..s]$.

We refer to a lemma from [12] that relates computing the k -bounded deletion distance between a prefix of P and a substring of T with a certain family of matrices (actually, Monge matrices [10]) that can be stored efficiently.

► **Lemma 7** ([12, Observation 27 and Lemma 28]). *Let P and T be strings of length p and n , respectively, and $q \in [1..n]$. There exists a sequence of $(3k+2) \times (3k+2)$ matrices $D_t(P, T, q, k)$, for $t \in [0..p]$, that satisfies the following conditions.*

- (a) *Let $t \in [0..p]$, $i \in [q..q+k]$, $j \in [i-k..i+k] \cap [i-t..n+1-t]$, and $k' \in [0..k]$. Then, $\delta_D(T[i..j+t], P[1..t]) = k'$ if and only if $D_t(P, T, q, k)[i-q+k+1, j-q+k+1] = k'$.*
- (b) *For each $t \in [0..p]$, there is a $(3k+1) \times (3k+1)$ permutation matrix P_t such that $D_t(P, T, q, k)[i, j] = 2P_t^\Sigma[i, j] + i - j$ holds for all $i, j \in [0..3k+1]$. Matrix P_0 is an identity permutation matrix (i.e., $D_0(P, T, q, k)[a, b] = |a - b|$). Moreover, the sequence $\Delta(P_0, P_1), \dots, \Delta(P_{p-1}, P_p)$ contains at most $3k(3k+1)/2$ ordered transpositions of neighboring elements in total and all its non-empty elements can be computed in $\mathcal{O}(k^2 \log \log k)$ time plus $\mathcal{O}(k^2)$ longest common prefix (LCP) queries on pairs of suffixes of P and T .*

We will also use a dynamic partial sums data structure that stores an integer array of size s and allows to update its elements and query for its prefix sums. We use a dynamic partial sums data structure with $\mathcal{O}(\log s / \log \log s)$ -time queries and $\mathcal{O}(s)$ space [9].

We are ready to describe our algorithm for All- k -BO.

► **Lemma 8.** *If string P , with $p = |P|$, and string T , with $n = |T|$, are over an integer alphabet of size $(n + p)^{\mathcal{O}(1)}$, All- k -BO can be solved in $\mathcal{O}(p + nk \log k / \log \log k)$ time.*

Proof. We perform in $\mathcal{O}(n + p)$ time a preprocessing on $P_{\S} \# T_{\S}$, where $\#$ is a symbol that does not occur in P_{\S} and T_{\S} , after which longest common prefix (LCP) queries on suffixes of P_{\S} and T_{\S} can be answered in $\mathcal{O}(1)$ time [4, 17] (we use the assumption on the alphabet).

The following computations are made for $\mathcal{O}(n/k)$ values of q such that the intervals $[q \dots q + 2k]$ cover $[1 \dots 2n]$. We will treat the sequence of matrices $D_t(P_{\S}, T_{\S}, q, 2k)$ for $t \in [0 \dots 2p]$ as a dynamic matrix D . Each ordered transposition of adjacent columns in the maintained matrix P_t corresponds to a sub-column increment in $D_t(P_{\S}, T_{\S}, q, 2k)$ (increasing the entries in the sub-column by 2). The i th column of the dynamic matrix D will be stored with the aid of a dynamic partial sums data structure on array A_i . Initially, the arrays A_i are zeroed. If the elements $D[\ell, i], \dots, D[r, i]$ are to be increased by α , in the array A_i corresponding to the i th column, we increase $A_i[\ell]$ by α and decrease $A_i[r + 1]$ by α . Then

$$D_t(P_{\S}, T_{\S}, q, 2k)[j, i] = D_0(P_{\S}, T_{\S}, q, 2k)[j, i] + \sum_{a=0}^j A_i[a] = |i - j| + \sum_{a=0}^j A_i[a] \quad (1)$$

can indeed be computed by a single dynamic partial sum query. By Lemma 7(b), the number of updates will be $\mathcal{O}(k^2)$ and the updates can be computed in $\mathcal{O}(k^2 \log \log k)$ time after $\mathcal{O}(n + p)$ -time preprocessing for LCP queries. The updates for the partial sums data structure are performed in $\mathcal{O}(k^2 \log k / \log \log k)$ total time [9].

Recall that position x in T translates to position $2x - 1$ in T_{\S} and Fact 6. For every $\ell \in [0 \dots p]$ and $\ell' \in [\max(0, \ell - k) \dots \min(n, \ell + k)]$, we will compute the k -bounded value of

$$\delta_E(P[1 \dots \ell], T[n + 1 - \ell' \dots n]) = \frac{1}{2} \delta_D(P_{\S}[1 \dots 2\ell], T_{\S}[2(n + 1 - \ell') - 1 \dots 2n]) \quad (2)$$

using $\gamma := \frac{1}{2} D_{2\ell}(P_{\S}, T_{\S}, q, 2k)[2(n + 1 - \ell') - 1 - q + 2k + 1, (2n - 2\ell + 1) - q + 2k + 1]$ for q such that $2(n + 1 - \ell') - 1 \in [q \dots q + 2k]$. More precisely, by Lemma 7(a) (with $i = 2(n + 1 - \ell') - 1$, $t = 2\ell$, $j + t - 1 = 2n$, and threshold $2k$), if $2\gamma \leq 2k$, then (2) equals γ , and otherwise (2) is greater than k . We will process all pairs (ℓ, ℓ') of interest as follows.

For each of the $\mathcal{O}(n/k)$ given values of q (that satisfy $[q \dots q + 2k] \subseteq [1 \dots 2n]$), we consider all lengths $\ell \in [0 \dots p]$ such that the interval

$$L := [2(n + 1 - (\ell + k)) - 1 \dots 2(n + 1 - (\ell - k)) - 1] \cap [q \dots q + 2k] \quad (3)$$

is non-empty. For each such ℓ , we compute in $\mathcal{O}(1)$ time the interval L from (3) and, for each odd $r \in L$, assuming $2(n + 1 - \ell') - 1 = r$, compute $\ell' = n + 1 - (r + 1)/2$. This way, for q we consider $\mathcal{O}(k^2)$ pairs (ℓ, ℓ') . For each of them, γ can be computed in $\mathcal{O}(\log k / \log \log k)$ time using a dynamic prefix sum as in (1), which gives $\mathcal{O}(k^2 \log k / \log \log k)$ total time.

The time complexity is $\mathcal{O}(n + p + (n/k) \cdot k^2 \log k / \log \log k) = \mathcal{O}(p + nk \log k / \log \log k)$. ◀

► **Lemma 9.** *The k -Bounded Overlaps problem under edit distance for two strings U and V such that $\min(|U|, |V|) = n$ and U, V are over an integer alphabet of size $n^{\mathcal{O}(1)}$ can be solved in $\mathcal{O}((n + k)k \log k / \log \log k)$ time.*

Proof. Let U' be the prefix of U of length $\min(|U|, |V| + k)$ and V' be the suffix of V of length $\min(|V|, |U| + k)$. If ℓ is a length of a k -edit overlap of U and V , then, by definition, $\ell \leq |U'|$. Further, the approximately overlapping prefix of U and suffix of V are a prefix of U' and a suffix of V' , respectively. We have $|U'|, |V'| \leq n + k$.

We solve the All- k -BO problem for string pattern $P = U'$ and string text $T = V'$ using Lemma 8 in $\mathcal{O}((n + k)k \log k / \log \log k)$ time. For each $\ell \in [0 \dots |P|]$, we can then compute the k -bounded minimum edit distance between $P[1 \dots \ell]$ and a length- ℓ' suffix of T , over all ℓ' such that the distance can be at most k , i.e., for all $\ell' \in [\max(0, \ell - k) \dots \min(n, \ell + k)]$, in $\mathcal{O}(nk + k^2)$ total time. This produces the desired output. \blacktriangleleft

► **Theorem 10.** *k -Edit EDSM can be solved on-line in $\mathcal{O}(pmk \log k / \log \log k + Nk)$ time, plus $\mathcal{O}(p \log \log p)$ time preprocessing or in expectation.*

Proof. We apply the general scheme of steps (1)–(4), according to Observation 2. In step (1), we check if the string pattern P has a k -edit occurrence in a string $S \in \mathbf{T}[i]$. We apply the k -Edit PM algorithm for string pattern P and string text S . Over all strings S , this takes $\mathcal{O}(N_i k)$ time using the Landau-Vishkin algorithm [26] which sums up to $\mathcal{O}(Nk)$ time.

In step (2), the array $\mathbf{AP}_1^k(\mathbf{T}[i])$ of active prefixes of a single set is computed using k -Bounded Overlaps. More precisely, for $U = P$ and every string $V \in \mathbf{T}[i]$, for every ℓ such that U and V have a k -edit overlap of length ℓ , we set $\mathbf{AP}_1^k(\mathbf{T}[i])[\ell]$ to the smallest k' such that U and V have a k' -edit overlap of length ℓ ; otherwise $\mathbf{AP}_1^k(\mathbf{T}[i])[\ell]$ is set to ∞ . By Lemma 9, the time complexity, over all $V \in \mathbf{T}[i]$, is $\mathcal{O}((p + k) \cdot m_i \cdot k \log k / \log \log k)$ where $p = |P|$. We can assume that $k < p$ since otherwise P occurs at every position in \mathbf{T} . The array $\mathbf{AS}_1^k(\mathbf{T}[i])$ of active suffixes is computed symmetrically. This step is performed for all $i \in [1 \dots n]$ in $\mathcal{O}(pmk \log k / \log \log k)$ total time.

In step (3), we compute $\mathbf{AP}_i^k(\mathbf{T})$, for $i \in [1 \dots n]$ on-line. Initially $\mathbf{AP}_i^k(\mathbf{T}) = \mathbf{AP}_1^k(\mathbf{T}[i])$. For a given i , for all $S \in \mathbf{T}[i]$ such that $|S| \leq p$, we compute all k -edit occurrences of string pattern S in string text P . Using Landau-Vishkin k -Edit PM, this takes $\mathcal{O}(pk)$ time [26], for a total of $\mathcal{O}(pmk)$ time for all $i \in [1 \dots n]$ and $S \in \mathbf{T}[i]$.

Whenever a k -edit occurrence $P[a \dots b]$ of $S \in \mathbf{T}[i]$ is discovered, we set $\mathbf{AP}_i^k(\mathbf{T})[b] := \min(\mathbf{AP}_i^k(\mathbf{T})[b], k' + k'')$ where $k' = \mathbf{AP}_{i-1}^k(\mathbf{T})[a - 1]$ and $k'' = \delta_E(P[a \dots b], S)$. There are $\mathcal{O}(pmk)$ such occurrences, so the time complexity is $\mathcal{O}(pmk)$.

Finally, step (4) is performed in $\mathcal{O}(np) = \mathcal{O}(mp)$ time. \blacktriangleleft

4 Approximate EDSM via Approximate Dictionary Matching

Bernardini et al. [6] used the technique behind errata trees [14] to design their solution for 1-Approximate EDSM. We also use the data structure of [14], but basically as a black-box; overall, we design a general solution for k -Approximate EDSM for a constant k .

In Approximate Dictionary Matching, we are given a dictionary \mathcal{D} of d strings of total length s and an integer parameter $k > 0$. Our goal is to preprocess \mathcal{D} so that later, given a string text T of length t , we can list all the k -approximate occurrences of the patterns from \mathcal{D} in T , under the Hamming or edit distance. Lemma 11 below follows from Cole et al. [14]. Minor tweaks are needed in their approach to obtain the exact complexity as stated in the lemma; the original complexity would be $\mathcal{O}(s \log s + (d + t \log \log d) \log^k d + \text{occ})$ in the worst case, where occ is the number of pairs (string from \mathcal{D} , substring of T at distance at most k).

► **Lemma 11** ([14]). *If $k = \mathcal{O}(1)$ and all strings are over an integer alphabet, Approximate Dictionary Matching can be solved in $\mathcal{O}(s + (d + t \log \log d + t^2) \log^k d)$ time.*

Proof. Using the approach of Cole et al. [14], one first constructs the generalized suffix tree of all the patterns from the dictionary \mathcal{D} in $\mathcal{O}(s)$ expected time or $\mathcal{O}(s \log s)$ worst-case time. This time complexity is required to implement an oracle with $\mathcal{O}(1)$ -time access to a child of a given node with a given first character of the edge; Cole et al. either use perfect hashing or a slower deterministic solution. The sole purpose of the oracle is to be able to extend the suffix tree with all suffixes of the string text T in $\mathcal{O}(t)$ time, as in Ukkonen's on-line suffix tree construction [30]. We can, instead, construct the generalized suffix tree of all patterns from \mathcal{D} and the text T in $\mathcal{O}(s + t)$ worst-case time [17], as we consider only one text T .

With the aid of the suffix tree, Cole et al. construct a data structure composed of several compacted tries in $\mathcal{O}(s + d \log^k d)$ total time. Each explicit node of each compacted trie stores a (possibly empty) list of patterns from \mathcal{D} . We do not alter this part of their construction.

In the query algorithm, for a text T , after all suffixes of T have been located in the suffix tree as discussed above, for each suffix T' of T , $\mathcal{O}(\log^k d)$ paths in various compacted tries are computed in $\mathcal{O}(\log^k d \log \log d)$ total time. For each such path, all patterns from \mathcal{D} stored in the lists of the explicit nodes on the path are returned. If pattern $S \in \mathcal{D}$ is returned, this means that suffix T' of T has a prefix that matches S with (Hamming or edit) distance at most k . From the path information one can easily extract a corresponding prefix T'' of T' such that the (Hamming or edit) distance between T'' and S is at most k .

In the worst case, $\Omega(dt)$ approximate occurrences can be reported in total, which we would like to avoid. Each path computed in the query algorithm has length at most t . If instead of reporting all elements of a list of an explicit node, we report one single arbitrarily chosen element, we report only $\mathcal{O}(t^2 \log^k d)$ approximate occurrences in total. ◀

We generalize the Active Prefixes problem that was introduced in the exact variant in [7]. A solution to this problem is the non-trivial part of step (3).

k -APPROXIMATE ACTIVE PREFIXES (k -APPROXIMATE AP)

Input: A string P of length p , an array $W[0..p]$ containing values in $[0..k] \cup \{\infty\}$, a set \mathcal{S} of d strings of total length s , and a metric (Hamming or edit distance).

Output: An array $V[0..p]$ containing values in $[0..k] \cup \{\infty\}$ such that $V[j] = k'$, for $j \in [1..p]$, is the smallest value $k' \in [0..k]$ for which there exist $k'' \in [0..k]$, $S \in \mathcal{S}$ and $j' \in [1..p]$, such that $P[j' + 1..j]$ is at distance (Hamming or edit) at most k'' from S and $W[j'] = k' - k''$, or ∞ if such a value k' does not exist.

We obtain k -Mismatch AP and k -Edit AP problems for the respective distances. The k -Approximate AP problem can be solved directly using Approximate Dictionary Matching.

► **Lemma 12.** *If $k = \mathcal{O}(1)$ and all strings are over an integer alphabet, k -Approximate AP under Hamming or edit distance can be solved in $\mathcal{O}(s + (d + p \log \log d + p^2) \log^k d)$ time.*

Proof. We construct an instance of Approximate Dictionary Matching for dictionary \mathcal{S} of d strings of total length s , for every $k'' \in [0..k]$. The text in the Approximate Dictionary Matching will be P . By Lemma 11, for every k'' , all k'' -approximate occurrences of strings in \mathcal{S} in P can be computed in $\mathcal{O}(s + (d + p \log \log d + p^2) \log^k d)$ time.

Initially, the array V is filled with ∞ . For every $k'' \in [0..k]$ and every k'' -approximate occurrence $P[j' + 1..j]$ of a string from \mathcal{S} , we set $V[j] := \min(V[j], W[j'] + k'')$. If $V[j] > k$, we set it to ∞ . This step takes $\mathcal{O}(p^2)$ time as there are $\mathcal{O}(p^2)$ distinct occurrences. ◀

► **Lemma 13.** *If $k = \mathcal{O}(1)$, k -Approximate EDMS under the Hamming or edit distance can be solved on-line in $\mathcal{O}(N + np + p \log \log p + \sum_{i=1}^n T_k(N_i, m_i, p))$ time, where $T_k(s, d, p)$ is the time complexity of a solution to k -Approximate AP with parameters s, d, p .*

Proof. We follow steps (1)–(4) according to Observation 2. In step (1), we need to check if the string pattern P has a k -mismatch occurrence in one of the strings in $\mathbf{T}[i]$. In case of the Hamming distance, we can use the algorithm by Amir et al. [1] that works in $\mathcal{O}(N_i \sqrt{k} \log k)$ time, and in case of edit distance, the Landau-Vishkin algorithm [26] that works in $\mathcal{O}(N_i k)$ time. After $\mathcal{O}(p \log \log p)$ -time preprocessing on P , all strings in $\mathbf{T}[i] \cup \{P\}$ can be assumed to be over alphabet $[1..p+1]$. Overall, for $k = \mathcal{O}(1)$ the time complexity is $\mathcal{O}(N)$ under each of the distance metrics.

Assuming the Hamming distance, in step (2) the array $\text{AP}_1^k(\mathbf{T}[i])$ of active prefixes of a single set is computed using k -Bounded Overlaps for $U = P$ and every string $V \in \mathbf{T}[i]$. Using a solution to k -Bounded Overlaps with kangaroo jumps, the time complexity, over all $V \in \mathbf{T}[i]$, is $\mathcal{O}(N_i k)$. The array $\text{AS}_1^k(\mathbf{T}[i])$ of active suffixes is computed using a symmetric application of k -Bounded Overlaps in the same time complexity. This step is performed for all $i \in [1..n]$ in $\mathcal{O}(Nk)$ total time, which is $\mathcal{O}(N)$ for constant k .

Step (2) under edit distance is solved using k -Bounded Overlaps exactly as in Theorem 10. However, this time we use Lemma 9 to bound the complexity for a single string $S \in \mathbf{T}[i]$ by $\mathcal{O}(|S| + k)k \log k$, which yields $\mathcal{O}((N_i + m_i k)k \log k)$ time over all $S \in \mathbf{T}[i]$. For all $i \in [1..n]$, we get just $\mathcal{O}(N)$ time as k is constant.

Step (3) requires to compute the array $\text{AP}_i^k(\mathbf{T})$ using dynamic programming. We start with $\text{AP}_i^k(\mathbf{T}) = \text{AP}_1^k(\mathbf{T}[i])$. Then we solve an instance of k -Approximate AP problem with $W = \text{AP}_{i-1}^k(\mathbf{T})$, $\mathcal{S} = \mathbf{T}[i]$ and k . For a given i , we have $s = N_i$, $d = m_i$. Over all $i \in [1..n]$, the time complexity is $\mathcal{O}(\sum_{i=1}^n T_k(N_i, m_i, p))$.

Finally, step (4) is performed in $\mathcal{O}(np)$ time. ◀

We plug in the solution to k -Approximate AP from Lemma 12 to obtain the next theorem.

► **Theorem 14.** *If $k = \mathcal{O}(1)$, k -Approximate EDSM under the Hamming or edit distance can be solved on-line in $\mathcal{O}(N + (m + np \log \log m + np^2) \log^k m)$ time.*

5 1-Approximate EDSM in $\mathcal{O}(np^2 + N)$ Time

In this section, we solve the other two open problems stated by Bernardini et al. [6]. We design a combinatorial $\mathcal{O}(np^2 + N)$ -time algorithm for 1-Approximate EDSM under Hamming distance *and* also show how this can be extended to work for edit distance. In Section 5.1, we extend the $\mathcal{O}(np^2 + N \log p)$ -time algorithm presented in [6] for Hamming to edit distance. In Section 5.2, we show how we can shave the $\log p$ factor from the time complexity.

We introduce the 01-Approximate AP problem, in which the array $W[0..p]$ contains values in $\{0, \infty\}$ and the array $V[0..p]$ is to contain values in $\{1, \infty\}$ such that $V[j] = 1$ for $j \in [1..p]$ if and only if there exist $S \in \mathcal{S}$ and $j' \in [1..p]$, such that $P[j' + 1..j]$ is at distance (Hamming, edit) at most 1 from S and $W[j'] = 0$. The 1-Approximate AP problem reduces to two instances of exact (i.e., 0-Approximate) AP problem, one in which we have $W[j'] = V[j] = 0$ and one with $W[j'] = V[j] = 1$, and an instance of 01-Approximate AP.

5.1 1-Edit AP in $\mathcal{O}(s + d \log p + p^2)$ Time using 1-Errata Tree

In [6], the 01-Mismatch AP problem is solved using a version of 1-errata tree, where the copied nodes of the input trie are *explicitly inserted into the tree*. This way the obtained errata tree is an actual trie and hence allows to apply standard tree traversal algorithms. In the next lemma, we generalize the construction in [6] to edit distance.

► **Lemma 15.** *Let us consider an instance of the 01-Edit AP problem. In $\mathcal{O}(s + (d+p) \log(d+p))$ time and space, one can construct a 1-errata tree \mathcal{T}_1 satisfying the following: $V[b] \leq 1$ if and only if \mathcal{T}_1 contains a pair of nodes u, v such that u has label (S, ℓ_1) , it is an ancestor of node v with label $(P[a..p], \ell_2)$, where $W[a] = 0$, $S \in \mathcal{S}$, $a + \text{depth}(u) = b$ and one of the following is satisfied: $\ell_1 = \ell_2 = \text{sub}_k$ for some integer $k \geq 0$ or $\ell_1 = \#$ or $\ell_2 = \#$.*

Proof. We start with the description of the 1-errata tree \mathcal{T}_1 , and then show that it satisfies the conditions stated in Lemma 15.

For a general compacted trie \mathcal{T}_0 with t leaf nodes, we can construct its 1-errata tree \mathcal{T}_1 as follows. We compute the heavy-light decomposition [28] of \mathcal{T}_0 in $\mathcal{O}(t)$ time and make all direct children of branching nodes explicit. A node of the trie reached by reading the string X is labeled with the reference to this string (a single node can have multiple labels; e.g., if the same string appears in the collection multiple times); here we just use the string itself for simplicity of the description, while in practice one would use a more succinct representation. For every light node u at depth d that is reached from $\text{parent}(u)$ with a character a we make three copies of its subtree:

- in one copy, for each label X of each node, we add sub_d (the label becomes (X, sub_d)) and merge it with the subtree of the heavy sibling of u (just like in [5, 6], but using sub_d instead of d);
- in one copy, for each label X of each node, we add del_d (the label becomes (X, del_d)) and merge it with the subtree of $\text{parent}(u)$;
- in one copy, for each label X of each node, we add ins_d (the label becomes (X, ins_d)) and merge it with the subtree of the node reached by reading a from the heavy sibling of u (this node can be implicit, or even non-existent in \mathcal{T}_0 , in which case we simply make it explicit/add it).

We change the original labels of nodes (the ones which were already present in \mathcal{T}_0) from X to $(X, \#)$ so that each label forms a pair. We denote the resulting tree by \mathcal{T}_1 .

▷ **Claim 16.** Let u and v be two nodes of \mathcal{T}_0 labeled with X and Y respectively, for two strings X and Y . String Y is at edit distance at most 1 from a prefix of X if and only if \mathcal{T}_1 contains two nodes, u' labeled with (X, ℓ_1) and its ancestor v' labeled with (Y, ℓ_2) , and

- $\ell_1 = \ell_2 = \text{sub}_k$ for some integer $k \geq 0$; or
- $\ell_1 = \#$ or $\ell_2 = \#$.

Proof. We assume that $|X| \geq |Y| + 2$ to reduce the number of special cases – if such an assumption cannot be made we can simply modify the construction by appending the pattern X with 3 characters \$ out of the alphabet, which does not change the performance.

(\Rightarrow) Let node $x = \text{LCA}(u, v)$ be the lowest common ancestor of u and v in \mathcal{T}_0 . If $x = v$, then Y is a prefix of X and hence the pair u, v is such a pair (v with label $(Y, \#)$ is an ancestor of u labeled with $(X, \#)$). Otherwise $d = \text{depth}(x) + 1$ is the position of the error (substitution, deletion, or insertion) in some alignment (sometimes a string can be obtained in a few ways for example by removing different characters of a unary run). Now it is enough to consider the three possible cases:

- Y is obtained from a prefix of X by a substitution at position d – in the heavy child of x there exist nodes u', v' with the descendant ancestor relation in \mathcal{T}_1 with labels (X, ℓ_1) , (Y, ℓ_2) , respectively, such that $\ell_1, \ell_2 \in \{\#, \text{sub}_d\}$.
- Y is obtained from a prefix of X by a deletion of the character at position d – if the node u is in the subtree of the heavy child of x in \mathcal{T}_0 , then in \mathcal{T}_1 the node u with label $(X, \#)$ is a descendant of v' with label (Y, ins_d) , otherwise in \mathcal{T}_1 a node u' with label (X, del_d) is a descendant of $(Y, \#)$.

- Y is obtained from a prefix of X by an insertion of a character between positions $d - 1$ and d – if the node v is in the subtree of the heavy child of x in \mathcal{T}_0 , then in \mathcal{T}_1 the node v with label $(Y, \#)$ is an ancestor of u' with label (X, ins_d) , otherwise in \mathcal{T}_1 a node v' with label (Y, del_d) is an ancestor of $(X, \#)$.

(\Leftarrow) Let us assume that the consequences are satisfied. Let u' be the node whose label contains (X, ℓ_1) and v' the node whose label contains (Y, ℓ_2) , such that v' is an ancestor of u' . We first assume $\ell_1 = \ell_2 = \text{sub}_d$ for some $d \in \mathbb{N}$, then (like in the Hamming case) the prefix of X of length $|Y|$ can be obtained from Y by replacing its d th character with $X[d]$. The same applies when $\ell_1 = \#$ and $\ell_2 = \text{sub}_d$ or $\ell_1 = \text{sub}_d$ and $\ell_2 = \#$.

If $\ell_1 = \#$ and $\ell_2 = \text{del}_d$ or $\ell_1 = \text{ins}_d$ and $\ell_2 = \#$ then the length- $(|Y| - 1)$ prefix of X can be obtained from Y by removing the character on its d th position.

If $\ell_1 = \#$ and $\ell_2 = \text{ins}_d$ or $\ell_1 = \text{del}_d$ and $\ell_2 = \#$ then Y can be obtained from the length- $(|Y| + 1)$ prefix of X by removing the character on its d th position. \triangleleft

Now, Lemma 15 is a direct application of Claim 16 to the tree \mathcal{T}_0 representing the set of the strings in \mathcal{S} plus the set of the suffixes of the pattern $P[j'..p]$ such that $W[j'] = 0$. This tree can be constructed in $\mathcal{O}(s + p)$ time and has $\mathcal{O}(d + p)$ nodes, hence its 1-errata tree \mathcal{T}_1 can be constructed in $\mathcal{O}(s + (d + p) \log(d + p))$ time in the same way as in [6, Lemma 4.2]. (Actually, in [6, Lemma 4.2] the complexity is stated as $\mathcal{O}((s + p) \log(d + p))$, but it can be readily verified that the algorithm presented there achieves the claimed complexity; cf. [31] for a more general construction with such a bound.) \blacktriangleleft

This property of our 1-errata tree for edit distance is exactly the same as the property of the 1-errata tree for the Hamming distance in [6, Lemma 4.3]. At the same time it is a simple special case of the property shown in [31] (together with the errata tree construction). Moreover, we can assume that $\log d = \mathcal{O}(\log p)$ because when $d > p^3$ we use a $\mathcal{O}(s + p^3)$ -time algorithm that was shown in [6] (such an algorithm also follows by Lemma 18 below as we can assume that $t \leq p$). We can now employ the algorithm from [6] without any modification (only the size of the tree and the number of bit-vectors used therein is up to 3 times bigger).

► **Lemma 17.** *The 1-Edit AP problem can be solved in $\mathcal{O}(s + d \log p + p^2)$ time.*

5.2 Shaving the $\log p$ Factor

To shave the $\log p$ factor from Lemma 17, we need to apply it for $d = \mathcal{O}(s / \log p)$ strings of \mathcal{S} . By the pigeonhole principle, we have $d \leq s / \log p$ strings of length greater than $\log p$ in \mathcal{S} . For the remaining short strings, we employ a simple sorting-based algorithm.

In the next lemma, in the case of Hamming distance, instead of performing every possible substitution in pattern substrings, which would incur an additional σ factor in the complexity, we replace every possible character in pattern substrings *as well as* in every string $S \in \mathcal{S}$ by a wildcard character $\#$. We then sort these strings using existing techniques. If two such modified strings are equal, the original strings were at Hamming distance at most 1. This is efficient *only for 1 mismatch*, because then we obtain only s modified strings from \mathcal{S} .

► **Lemma 18.** *Let us consider an instance of the 1-Approximate AP problem in which all strings in \mathcal{S} have length at most t . The 1-Approximate AP problem under the Hamming or edit distance can be solved in $\mathcal{O}(s + pt^2)$ time.*

Proof. We first solve the 01-Approximate AP problem; the 0-Approximate AP problem is easier. Our algorithm for 01-Approximate AP uses a technique based on sorting. We will describe it for the Hamming distance but edit distance works largely in the same manner. Consider the following grouping; for every $\ell \in [1..t]$, we construct group G_ℓ consisting of:

- For all pairs (j', j) , with $j' \in [0..p]$ such that $W[j'] = 0$ and $j \in [j' + 1..j' + t]$, the modified substrings of P , such that the ℓ th character of every substring $P[j' + 1..j]$ is replaced by a special character $\#$ that does not occur in P .

■ The modified strings $S \in \mathcal{S}$, $|S| \geq \ell$, such that the ℓ th character of S is replaced by $\#$. To simulate a single insertion or deletion operation, we *delete* the ℓ th character from $P[j' + 1..j]$ or from S . For technical purposes, we assume that every such modified string is concatenated with another special character $\$$ that is lexicographically the smallest.

For efficiency, we never construct these strings explicitly. We rather encode them using a constant number of substrings from P or from strings in \mathcal{S} (substrings are represented by intervals). The key observation is that the total number of strings in G_ℓ , for all $\ell \in [1..t]$, is $\mathcal{O}(s + pt^2)$. We then sort the strings in G_ℓ and compute the (longest common prefix) LCP between every pair of successive strings in this sorted list; we denote the sorted list by L_ℓ . This can be done in $\mathcal{O}(|G_\ell|)$ time, for all (j', j) , using the *gapped suffix array* [15], because all strings in group G_ℓ have character $\#$ at the same ℓ th position.

The strings in L_ℓ , whose unmodified version are within Hamming distance 1 and the mismatching position is ℓ , form a *consecutive sublist* of L_ℓ . In particular, two elements x, y are in a consecutive sublist if and only if $x = y$. This can be checked in $\mathcal{O}(1)$ time because the length of their LCP is $|x|$. It suffices to look if, in any such consecutive sublist, we have one element that originates from $\mathbf{T}[i]$ and one element that originates from P . By thus traversing all L_ℓ and looking at the LCP values of consecutive entries, we can find if any string from $\mathbf{T}[i]$ is within Hamming distance 1 from any $P[j' + 1..j]$. If so, we set $V[j]$ to 1. The time for processing one L_ℓ is $\mathcal{O}(pt + |L_\ell|)$. Thus the total time is $\mathcal{O}(s + pt^2)$.

For 0-Approximate AP, we construct just one group G that contains all the $\mathcal{O}(pt)$ considered substrings of P and all strings in \mathcal{S} . The complexity is $\mathcal{O}(pt)$. ◀

► **Lemma 19.** *1-Approximate AP under Hamming or edit distance can be solved in $\mathcal{O}(s + p^2)$ time.*

Proof. For short strings of length at most $\log p$, we employ Lemma 18 working in $\mathcal{O}(s + pt^2) = \mathcal{O}(s + p \log^2 p)$ time. For long strings of length greater than $\log p$ we use the $\mathcal{O}(s + d \log p + p^2)$ -time algorithm for Hamming distance from [6] and for edit distance we use Lemma 17. By the pigeonhole principle, $d \leq s / \log p$ and the second complexity becomes $\mathcal{O}(s + p^2)$. ◀

We plug Lemma 19 into the scheme of Lemma 13 to obtain the following result.

► **Theorem 20.** *1-Approximate EDSM under the Hamming or edit distance can be solved on-line in $\mathcal{O}(N + np^2)$ time.*

6 k -Approximate Problems on Two ED Strings

We proceed to our results on k -Approximate EDSI and k -Approximate Doubly EDSM.

► **Theorem 21.** *Given an ED string \mathbf{T}_1 of cardinality m_1 and size N_1 , an ED string \mathbf{T}_2 of cardinality m_2 and size N_2 , and an integer $k > 0$, we can check whether a pair of strings $S_1 \in \mathcal{L}(\mathbf{T}_1), S_2 \in \mathcal{L}(\mathbf{T}_2)$ with $\delta_H(S_1, S_2) \leq k$ ($\delta_E(S_1, S_2) \leq k$, respectively) exists in $\mathcal{O}(k^{2/3}(N_1 m_2 + N_2 m_1) \log^{1/3}(N_1 + N_2))$ time ($\mathcal{O}(k(N_1 m_2 + N_2 m_1) \log k / \log \log k)$ time, respectively) and, if that is the case, return a pair with the smallest distance.*

Proof. Let n_1 and n_2 denote the lengths of \mathbf{T}_1 and \mathbf{T}_2 , respectively. We use the approach from [19, Theorem 8.2] that performs the following steps for either of the distance metrics:

1. For $a \in \{1, 2\}$, compute all k -approximate occurrences of each string pattern P from each set $\mathbf{T}_a[i]$ in each string text T from each set $\mathbf{T}_{3-a}[j]$, for $i \in [1 \dots n_a]$ and $j \in [1 \dots n_{3-a}]$. Together with each approximate occurrence, report also the actual minimal distance (that is at most k).
2. For $a \in \{1, 2\}$, for each string U from each set $\mathbf{T}_a[i]$ and for each string V from each set $\mathbf{T}_{3-a}[j]$, compute all pairs (U', V') such that U' is a prefix of U , V' is a suffix of V and the distance between U' and V' is at most k . Together with each prefix-suffix pair, report also the actual minimal distance (that is at most k).
3. Use the computed outputs to construct a graph of size $\mathcal{O}(N_1 m_2 + N_2 m_1)$ in case of the Hamming distance and of size $\mathcal{O}(k(N_1 m_2 + N_2 m_1))$ in case of edit distance [19].
4. The solution to k -Approximate EDSI (that includes a witness pair) is obtained using a linear-time traversal of the graph constructed in step 3 [19].

Steps 3 and 4 are the same as in Gabory et al. [19]. In steps 1 and 2, Gabory et al. [19] achieved $\mathcal{O}((N_1 m_2 + N_2 m_1)k)$ time for the Hamming distance and $\mathcal{O}((N_1 m_2 + N_2 m_1)k^2)$ time for the edit distance. We use the techniques from Section 3 to improve these complexities.

In step 1, in each string text T in \mathbf{T}_{3-a} , we perform k -Approximate PM for m_a patterns. Each k -Mismatch PM is performed in $\mathcal{O}(|T|\sqrt{k \log k})$ time using the algorithm of Amir et al. [1] and each k -Edit PM is performed in $\mathcal{O}(|T|k)$ time using the Landau-Vishkin algorithm [26]. Over all patterns and texts, we obtain $\mathcal{O}(m_a N_{3-a} \sqrt{k \log k})$ time for the Hamming distance and $\mathcal{O}(m_a N_{3-a} k)$ for the edit distance. We recall that each algorithm returns the actual minimal distance of the occurrence if it is at most k .

In step 2 for the Hamming distance, we solve the k -Bounded Overlaps problem for every string U from \mathbf{T}_a and V from \mathbf{T}_{3-a} . The total number of prefix-suffix pairs is bounded by the number of prefixes, which is at most N_a , times the number of strings V , which is m_{3-a} . With Lemma 3, the time complexity is bounded by $\mathcal{O}(N_a m_{3-a} k^{2/3} \log^{1/3}(N_a + N_{3-a}))$. For the edit distance, each prefix of each string U from \mathbf{T}_a can match up to $2k - 1$ suffixes of each string from \mathbf{T}_{3-a} . Hence, the number of prefix-suffix pairs is bounded by $\mathcal{O}(N_a m_{3-a} k)$. The approximately matching prefix-suffix pairs are computed using the All- k -BO problem (Lemma 8) in $\mathcal{O}((|U| + |V|)k \log k / \log \log k)$ time. For all U and V , we achieve $\mathcal{O}((N_1 m_2 + N_2 m_1)k \log k / \log \log k)$ time. Thus the complexities of step 2 dominate the respective complexities in step 1.

The time complexities of steps 3 and 4, that is, $\mathcal{O}(N_1 m_2 + N_2 m_1)$ for the Hamming distance and $\mathcal{O}((N_1 m_2 + N_2 m_1)k)$ for the edit distance, are also dominated. ◀

In [19, Corollary 8.3], it was shown that the complexity of k -Approximate EDSI carries on to k -Approximate Doubly EDSM, which we define next. We therefore obtain Corollary 22.

k -APPROXIMATE DOUBLY EDSM

Input: An ED string \mathbf{T}_1 of cardinality m_1 and size N_1 (called text) and an ED string \mathbf{T}_2 of cardinality m_2 and size N_2 (called pattern).

Output: The set of positions i for which there exists a string $P \in \mathcal{L}(\mathbf{T}_2)$ whose k -approximate occurrence in \mathbf{T}_2 starts in $\mathbf{T}_2[i]$.

► **Corollary 22.** *k -Approximate Doubly EDSM problem can be solved in $\mathcal{O}(k^{2/3}(N_1 m_2 + N_2 m_1) \log^{1/3}(N_1 + N_2))$ time and in $\mathcal{O}(k(N_1 m_2 + N_2 m_1) \log k / \log \log k)$ time for the Hamming and edit distance, respectively.*

References

- 1 Amihoud Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004. doi:10.1016/S0196-6774(03)00097-X.
- 2 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. doi:10.4230/LIPICS.CPM.2018.9.
- 3 Rocco Ascone, Giulia Bernardini, Alessio Conte, Massimo Equi, Estéban Gabory, Roberto Grossi, and Nadia Pisanti. A unifying taxonomy of pattern matching in degenerate strings and founder graphs. In Solon P. Pissis and Wing-Kin Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 312 of *LIPIcs*, pages 14:1–14:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.14.
- 4 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. doi:10.1007/10719839_9.
- 5 Giulia Bernardini, Estéban Gabory, Solon P. Pissis, Leen Stougie, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string matching with 1 error. In Armando Castañeda and Francisco Rodríguez-Henríquez, editors, *LATIN 2022: Theoretical Informatics - 15th Latin American Symposium, Guanajuato, Mexico, November 7-11, 2022, Proceedings*, volume 13568 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2022. doi:10.1007/978-3-031-20624-5_2.
- 6 Giulia Bernardini, Estéban Gabory, Solon P. Pissis, Leen Stougie, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string matching with 1 error or mismatch. *Theory of Computing Systems*, 68:1442–1467, 2024. doi:10.1007/s00224-024-10194-8.
- 7 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. doi:10.1137/20M1368033.
- 8 Giulia Bernardini, Nadia Pisanti, Solon P. Pissis, and Giovanna Rosone. Approximate pattern matching on elastic-degenerate text. *Theoretical Computer Science*, 812:109–122, 2020. doi:10.1016/J.TCS.2019.08.012.
- 9 Gerth Stølting Brodal, Pooya Davoodi, and S. Srinivasa Rao. Path minima queries in dynamic weighted trees. In Frank Dehne, John Iacono, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, volume 6844 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2011. doi:10.1007/978-3-642-22300-6_25.
- 10 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996. doi:10.1016/0166-218X(95)00103-X.
- 11 Timothy M. Chan, Ce Jin, Virginia Vassilevska Williams, and Yinzhan Xu. Faster algorithms for text-to-pattern Hamming distances. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 2188–2203. IEEE, 2023. doi:10.1109/FOCS57990.2023.00136.
- 12 Panagiotis Charalampopoulos, Tomasz Kociumaka, Jakub Radoszewski, Solon P. Pissis, Wojciech Rytter, Tomasz Waleń, and Wiktor Zuba. Approximate circular pattern matching. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 35:1–35:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ESA.2022.35.

- 13 Aleksander Cislak, Szymon Grabowski, and Jan Holub. SOPanG: online text searching over a pan-genome. *Bioinformatics*, 34(24):4290–4292, 2018. doi:10.1093/BIOINFORMATICS/BTY506.
- 14 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. doi:10.1145/1007352.1007374.
- 15 Maxime Crochemore and German Tischler. The gapped suffix array: A new index structure for fast approximate matching. In Edgar Chávez and Stefano Lonardi, editors, *String Processing and Information Retrieval - 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings*, volume 6393 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2010. doi:10.1007/978-3-642-16321-0_37.
- 16 Massimo Equi, Veli Mäkinen, Alexandru I. Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Transactions on Algorithms*, 19(3):21:1–21:25, 2023. doi:10.1145/3588334.
- 17 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.
- 18 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*, 31(3):538–544, 1984. doi:10.1145/828.1884.
- 19 Estéban Gabory, Moses Njagi Mwaniki, Nadia Pisanti, Solon P. Pissis, Jakub Radoszewski, Michelle Sweering, and Wiktor Zuba. Elastic-degenerate string comparison. *Information and Computation*, 304:105296, 2025. doi:10.1016/j.ic.2025.105296.
- 20 Zvi Galil and Raffaele Giancarlo. Parallel string matching with k mismatches. *Theoretical Computer Science*, 51:341–348, 1987. doi:10.1016/0304-3975(87)90042-9.
- 21 Paweł Gawrychowski, Adam Górkiewicz, Pola Marciniak, Solon P. Pissis, and Karol Pokorski. Faster approximate elastic-degenerate string matching – Part B. In Paola Bonizzoni and Veli Mäkinen, editors, *36th Annual Symposium on Combinatorial Pattern Matching, CPM 2025, June 17-19, 2025, Milan, Italy*, volume 331 of *LIPICs*, pages 29:1–29:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICs.CPM.2025.29.
- 22 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. doi:10.4230/LIPICs.CPM.2017.9.
- 23 Costas S. Iliopoulos, Ritu Kundu, and Solon P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Information and Computation*, 279:104616, 2021. doi:10.1016/J.IC.2020.104616.
- 24 Haim Kaplan, Ely Porat, and Nira Shafrir. Finding the position of the k -mismatch and approximate tandem repeats. In Lars Arge and Rusins Freivalds, editors, *Algorithm Theory - SWAT 2006, 10th Scandinavian Workshop on Algorithm Theory, Riga, Latvia, July 6-8, 2006, Proceedings*, volume 4059 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2006. doi:10.1007/11785293_11.
- 25 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. doi:10.1016/0304-3975(86)90178-7.
- 26 Gad M. Landau and Uzi Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989. doi:10.1016/0196-6774(89)90010-2.
- 27 Milan Ruzic. Constructing efficient dictionaries in close to sorting time. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, 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. doi:10.1007/978-3-540-70575-8_8.
- 28 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.
- 29 Alexandre Tiskin. Semi-local string comparison: algorithmic techniques and applications. *CoRR*, abs/0707.3619, 2007. arXiv:0707.3619.
- 30 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. doi:10.1007/BF01206331.
- 31 Wiktor Zuba, Grigorios Loukides, Solon P. Pissis, and Sharma V. Thankachan. Approximate suffix-prefix dictionary queries. In Rastislav Královic and Antonín Kucera, editors, *49th International Symposium on Mathematical Foundations of Computer Science, MFCS 2024, August 26-30, 2024, Bratislava, Slovakia*, volume 306 of *LIPICs*, pages 85:1–85:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.MFCS.2024.85.