


# Elastic-Degenerate String Matching with 1 Error

Giulia Bernardini<sup>1</sup> , Esteban Gabory<sup>2</sup> , Solon P. Pissis<sup>2,3,4</sup> ,  
Leen Stougie<sup>2,3,4</sup> , Michelle Sweering<sup>2</sup>, and Wiktor Zuba<sup>2</sup>  

<sup>1</sup> University of Trieste, Trieste, Italy

`giulia.bernardini@units.it`

<sup>2</sup> CWI, Amsterdam, The Netherlands

`{esteban.gabory,solon.pissis,leen.stougie,michelle.sweering,  
wiktor.zuba}@cwi.nl`

<sup>3</sup> Vrije Universiteit, Amsterdam, The Netherlands

<sup>4</sup> INRIA-Érable, Villeurbanne, France

**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  $\tilde{O}(nm^{\omega-1}) + \mathcal{O}(N)$ -time algorithm [Bernardini et al., SIAM J. Comput. 2022], where  $\omega$  denotes the matrix multiplication exponent and the  $\tilde{O}(\cdot)$  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, 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 algorithm runs in  $\Omega(mN)$  time in the worst case. Here 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. Our algorithms rely on non-trivial reductions from 1-EDSM to special instances of classic computational geometry problems (2d rectangle stabbing or range emptiness), which we show how to solve efficiently.

**Keywords:** String algorithms · Approximate string matching · Edit distance · Degenerate strings · Elastic-degenerate strings

---

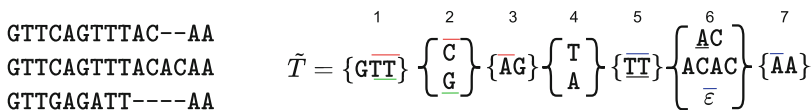
The work in this paper is supported in part by: the Netherlands Organisation for Scientific Research (NWO) through project OCENW.GROOT.2019.015 “Optimization for and with Machine Learning (OPTIMAL)” and Gravitation-grant NETWORKS-024.002.003; the PANGAIA and ALPACA projects that have received funding 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.

# 1 Introduction

String matching (or pattern matching) is a fundamental task in computer science, for which several linear-time algorithms are known [18]. 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 [26] 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 [1, 2, 22, 32] in a gapless multiple sequence alignment (MSA).

Iliopoulos et al. generalized these representations in [25] 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  $\Sigma$  is a sequence of finite subsets of  $\Sigma^*$  (which includes the empty string  $\varepsilon$ ), called *segments*. The number of segments is the *length* of the ED string, denoted by  $n = |\tilde{T}|$ ; and the total number of letters (including symbol  $\varepsilon$ ) in all segments is the *size* of the ED string, denoted by  $N = \|\tilde{T}\|$ . Inspect Fig. 1 for an example.

In Table 1,  $m$  is the length of the pattern,  $n$  is the length of the ED text,  $N$  is its size, and  $\omega$  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. [24] 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. [8] 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. [8] 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 [9], both using fast matrix multiplication, thus breaking through the conditional lower bound for EDSM.



**Fig. 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  $P = \text{TTA}$  in  $\tilde{T}$  end at positions 6 (black underline) and 7 (blue overline); a *1-error* occurrence of  $P$  in  $\tilde{T}$  ends at position 2 (green underline); and another *1-error* occurrence of  $P$  in  $\tilde{T}$  ends at position 3 (red overline). Note that other 1-error occurrences of  $P$  in  $\tilde{T}$  exist (e.g., ending at positions 1 and 5). (Color figure online)

**Table 1.** The upper-bound landscape of the EDSM problem.

EDSM	Features	Running time
Grossi et al. [24]	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. [8]	Fast matrix multiplication	$\mathcal{O}(nm^{1.381} + N)$
Bernardini et al. [9]	Fast matrix multiplication	$\tilde{\mathcal{O}}(nm^{\omega-1}) + \mathcal{O}(N)$

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

$k$ -EDSM	Features	Running time
Bernardini et al. [10]	$k$ errors	$\mathcal{O}(k^2mG + kN)$
<b>This work</b>	1 error	$\mathcal{O}(nm^3 + N)$
<b>This work</b>	1 error	$\mathcal{O}((nm^2 + N) \log m)$
<b>This work</b>	1 error (decision)	$\mathcal{O}(nm^2\sqrt{\log m} + N \log \log m)$

*Our Results and Techniques.* In string matching, a single extra or missing letter in a potential occurrence results in missing (many or all) occurrences. Hence, many works are focused on approximate string matching for standard strings [4, 13, 17, 23, 27, 28]. For approximate EDSM ( $k$ -EDSM), Bernardini et al. [7, 10] gave 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.

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  $k$  [16]. Interestingly,  $k$ -EDSM seems to be the same case, which highlights the main theoretical motivation for this paper. In Table 2, we summarize the state of the art result for  $k$ -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 combination of algorithmic techniques. Our algorithms 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.

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  $\tilde{O}(nm^2 + N)$ -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 [24]. Last, our  $\mathcal{O}(nm^3 + N)$ -time algorithm has a linear dependency on  $N$ , another highly desirable property (at the expense of an extra  $m$ -factor).

*Paper Organization.* In Sect. 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 Sect. 3, we present our solutions under edit distance. In Sect. 4, we conclude with some basic open questions for future work.

## 2 Preliminaries

We start with some basic definitions and notation following [18]. Let  $X = X[1] \dots X[n]$  be a *string* of length  $|X| = n$  over an ordered alphabet  $\Sigma$  whose elements are called *letters*. The *empty string* is the string of length 0; we denote it by  $\varepsilon$ . 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  $X$ .

An *elastic-degenerate string* (ED string)  $\tilde{T} = \tilde{T}[1] \dots \tilde{T}[n]$  over an alphabet  $\Sigma$  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  $j$ th 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  $\mathcal{L}(\tilde{T}) = \tilde{T}[1] \cdot \dots \cdot \tilde{T}[n]$ . 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  $P = P_j \cdot \dots \cdot P_{j'}$ , 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 Fig. 1,  $P = \text{TTA}$  matches  $\tilde{T}[5..6]$  with two alignments: both have  $U = \varepsilon$ ,  $P_5 = \text{TT}$ ,  $P_6 = \text{A}$ , and  $V$  is either  $\text{C}$  or  $\text{CAC}$ .

We will refer to  $P$  as the *pattern* and to  $\tilde{T}$  as the *ED text*. We want to accept matches with edit distance at most 1.

**Definition 1.** *Given two strings  $P$  and  $Q$  over an alphabet  $\Sigma$ , we define the edit distance  $d_E(P, Q)$  between  $P$  and  $Q$  as the length of a shortest sequence of letter replacements, deletions, and insertions, to obtain  $P$  from  $Q$ .*

**Lemma 1** ([18]). *The function  $d_E$  is a distance on  $\Sigma^*$ .*

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  $\tilde{T}[j..j']$  with 1 error*. Let  $UP'_j, \dots, P'_j, V$  be an alignment of  $P'$  with  $\tilde{T}[j..j']$  and  $i \in [j, j']$  be an integer such that the single replacement, insertion, or deletion required to obtain  $P$  from  $P' = P'_j \cdot \dots \cdot P'_j$ , 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.*

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

- $AP_i \subseteq [1, m]$ , such that  $j \in AP_i$  if and only if  $P[1..j]$  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  $\mathcal{L}(\tilde{T}[1] \dots \tilde{T}[i])$ .
- $AS_i \subseteq [1, m]$ , such that  $j \in AS_i$  if and only if  $P[j..m]$  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  $\mathcal{L}(\tilde{T}[i] \dots \tilde{T}[n])$ .
- $1-AP_i \subseteq [1, m]$ , such that  $j \in 1-AP_i$  if and only if  $P[1..j]$  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 edit distance at most 1 from a suffix of a string in  $\mathcal{L}(\tilde{T}[1] \dots \tilde{T}[i])$ .

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

*Proof.* If  $P[j..m]$  is a suffix of  $S \in \mathcal{L}(\tilde{T}[i..n])$ , then  $P^R[1..m-j+1]$  is a suffix of  $S^R \in \mathcal{L}(\tilde{T}[1..n]^R)$ . 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.  $\square$

The efficient computation of active prefixes was shown in [24], 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 Fig. 2). Finding active prefixes (and, by Lemma 2, suffixes) reduces to the following problem, formalized in [8].

**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 3** ([24]). *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 1** ([24]). *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.*

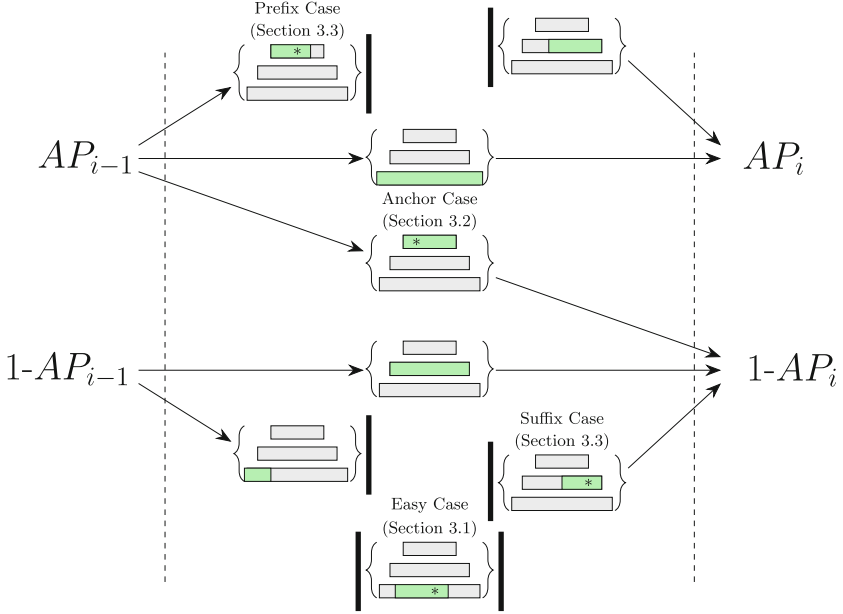
As depicted in Fig. 2, the computation of active prefixes with 1 error (1- $AP_i$ ) and the reporting of occurrences with 1 error reduce to a problem where the error can only occur in a single, fixed  $\tilde{T}[i]$ . In particular, this problem decomposes into 4 cases, which we formalize in the following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

**Proposition 1.** *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 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 for some  $1 \leq i \leq n$ .

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

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



**Fig. 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 the 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 [24]. (Color figure online)

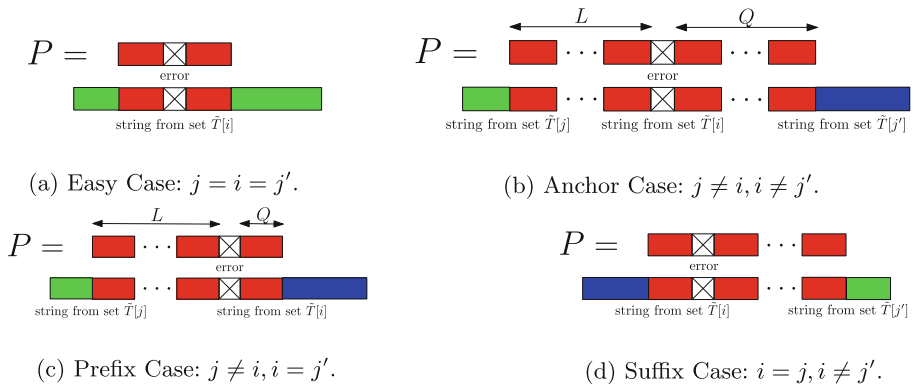
**Suffix Case:**  $P$  matches  $\tilde{T}[i..j']$  with 1 error in  $\tilde{T}[i]$  for some  $1 \leq i < j' \leq n$ , implying an active suffix of  $P$  which is a prefix of a string in  $\mathcal{L}(\tilde{T}[i+1..j'])$ .

*Proof.* Suppose  $P$  has a 1-error occurrence matching  $\tilde{T}[j..j']$  with  $1 \leq j \leq j' \leq n$ . If  $j = j'$  we are in the Easy Case. Otherwise, each alignment has an error in some  $\tilde{T}[i]$  for  $j \leq i \leq j'$ . 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.  $\square$

### 3 1-Error EDSM

In this section, we present algorithms for finding all 1-error occurrences of  $P$  given by each type of possible alignment described by Proposition 1 (inspect Fig. 3). The Prefix and Suffix Cases are analogous by Lemma 2; 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 Sect. 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 1, and we simultaneously compute possible exact occurrences of  $P$ . Then we compute 1-error occurrences of  $P$  by grouping the alignments



**Fig. 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  $\tilde{T}[i]$

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 1. 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 [30]. 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 4** ([11, 21]). *After  $\mathcal{O}(n\sqrt{\log n})$ -time preprocessing, we can answer 2d rectangle emptiness queries in  $\mathcal{O}(\log \log n)$  time.*

**Lemma 5** ([15, 31]). *After  $\mathcal{O}(n \log n)$ -time preprocessing, we can answer 2d rectangle stabbing queries in  $\mathcal{O}(\log n)$  time.*

In Sect. 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), for which we have the following well-known results.

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  $d_E(P, P') \leq 1$ .

**Lemma 6** ([17, 28]). *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.<sup>1</sup> 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 6 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 Fig. 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 definition of the edit distance, 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 Fig. 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  $\rho$  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 1. (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

<sup>1</sup> 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].

an on-line algorithm.) We will exhaustively consider all pairs  $(\lambda, \rho)$  such that  $\lambda + \rho < m$ . Clearly, there are  $\mathcal{O}(m^2)$  such pairs.

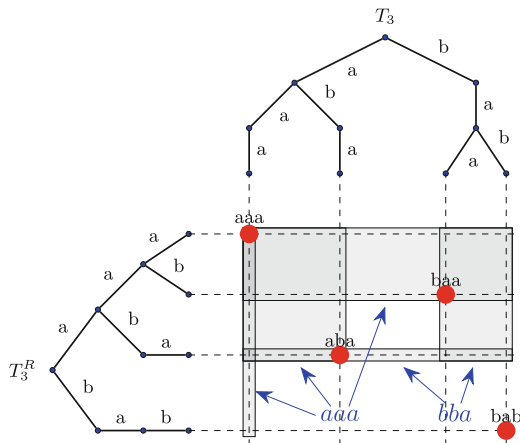
Consider the length  $\mu = m - (\lambda + \rho) > 0$  of the substring of  $P$  still to be matched for some prefix and suffix of  $P$  of lengths  $(\lambda, \rho)$ , respectively. We group together all pairs  $(\lambda, \rho)$  such that  $m - (\lambda + \rho) = \mu$  by sorting them in  $\mathcal{O}(m^2)$  time. We construct, for each such group  $\mu$ , the compacted trie  $T_\mu$  of the fragments  $P[\lambda + 1 .. m - \rho]$ , for all  $(\lambda, \rho)$  such that  $m - (\lambda + \rho) = \mu$ , and analogously the compacted trie  $T_\mu^R$  of all fragments  $P^R[\rho + 1 .. m - \lambda]$ . For each group  $\mu$ , this takes  $\mathcal{O}(m)$  time [29]. 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 [20].

We also group all strings in segment  $\tilde{T}[i]$  of length less than  $m$  by their length  $\mu$ . The group for length  $\mu$  is denoted by  $G_\mu$ . This takes  $\mathcal{O}(N_i)$  time. Clearly, the strings in  $G_\mu$  are the only candidates to extend pairs  $(\lambda, \rho)$  such that  $m - (\lambda + \rho) = \mu$ . Note that the mismatch can be at any position of any string of  $G_\mu$ : 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 all such pairs of positions  $(h, k)$  whose sum is  $\mu - 1$  (intuitively, the minus one is for the mismatch). This guarantees that  $L = P[1 .. \lambda]S'$  and  $Q = S''P[m - \rho + 1 .. m]$  are such that  $|L| + |Q| + 1 = m$ . The pairs are  $(0, \mu - 1), (1, \mu - 2), \dots, (\mu - 1, 0)$ . 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  $S[1 .. h]$  in  $T_\mu$ . If the whole  $S[1 .. h]$  is successfully spelled ending at a node  $u$ , this implies that all the fragments of  $P$  corresponding to nodes descending from  $u$  share  $S[1 .. h]$  as a prefix. We also spell  $S^R[1 .. k]$  in  $T_\mu^R$ . If the whole of  $S^R[1 .. k]$  is successfully spelled ending at a node  $v$ , then all the fragments of  $P$  corresponding to nodes descending from  $v$  share  $(S^R[1 .. k])^R$  as a suffix. Nodes  $u$  and  $v$  identify an interval of leaves in  $T_\mu$  and  $T_\mu^R$ , 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 Fig. 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.** Recall that  $T_\mu, T_\mu^R$  by construction are ordered based on lexicographic ranks. For every pair  $(T_\mu, T_\mu^R)$ , we construct a data structure for 2d rectangle emptiness queries on the grid  $[1, \ell]^2$ , where  $\ell$  is the number of leaves of  $T_\mu$  (and of  $T_\mu^R$ ), for the set of points  $(x, y)$  such that  $x$  is the lexicographic rank of the leaf of  $T_\mu$  representing  $P[\lambda + 1 .. m - \rho]$  and  $y$  is the rank of the leaf of  $T_\mu^R$  representing  $P^R[\rho + 1 .. m - \lambda]$  for the same pair  $(\lambda, \rho)$ . This denotes that the two leaves correspond to the *same fragment* of  $P$ . For every  $(T_\mu, T_\mu^R)$ , this preprocessing takes  $\mathcal{O}(m\sqrt{\log m})$  time by Lemma 4, since  $\ell$  is  $\mathcal{O}(\mu) = \mathcal{O}(m)$ . For all  $\mu \leq m$  groups, 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 4. Note that all rectangles for  $S$  can be collected in  $\mathcal{O}(|S|) = \mathcal{O}(\mu)$  time



**Fig. 4.** An example of points and rectangles (solid shapes) for the decision version of the Anchor Case with 1 mismatch. Here  $P = bbaaaabababb$ ,  $AP_{i-1} = \{1, 2, 4, 7, 8, 9\}$ ,  $AS_{i+1} = \{5, 6, 9, 11, 12\}$ ,  $\mu = 3$ , 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  $(\varepsilon, aa)$ ,  $(a, a)$ ,  $(aa, \varepsilon)$ ,  $(\varepsilon, ab)$ ,  $(b, a)$ , namely, the pairs of prefixes and reversed suffixes of  $aaa$  and  $bba$  (rectangle  $(bb, \varepsilon)$  does not exist as  $T_3$  contains no node  $bb$ ).

by spelling  $S$  through  $T_\mu$  and  $S^R$  through  $T_\mu^R$ , one letter at a time. Thus the total time for processing all  $G_\mu$  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 5, for all  $\mu$  groups, the whole preprocessing thus takes  $\mathcal{O}(N_i \log N_i)$  time.

For every  $(T_\mu, T_\mu^R)$ , 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  $(T_\mu, T_\mu^R)$ , this takes  $\mathcal{O}(m \log N_i)$  time by Lemma 5 and by the fact that for each group  $G_\mu$  there are  $\mathcal{O}(m)$  pairs  $(\lambda, \rho)$  such that  $m - (\lambda + \rho) = \mu$ . For all groups  $G_\mu$  (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  $G_\mu$  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  $\rho$  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  $\rho$  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 Fig. 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. [24] 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  $L = P[1.. \ell]S'$ , 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 definition of the edit distance, 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 Fig. 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 (Sect. 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  $\mu$  is denoted by  $G_\mu$ . We construct the compacted trie  $T_{G_\mu}$  of the strings in  $G_\mu$ , and the compacted trie  $T_{G_\mu}^R$  of the reversed strings in  $G_\mu$ . 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 7** ([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  $X[i..j]$  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  $\mu$ , Lemma 7, with a set  $I$  consisting of the intervals over  $X$  corresponding to the strings in  $G_\mu$ . By sorting, in this way, all strings in  $G_\mu$  (for all  $\mu$ ), and by constructing [19] 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  $T_{G_\mu}$  in  $\mathcal{O}(N_i)$

total time. We handle  $T_{G_\mu}^R$ , for all  $\mu$ , analogously. Similar to the Anchor Case we enhance all nodes with a perfect hash table within the same complexities [20].

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  $G_\mu$  whose strings are all of length  $\mu$ . 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  $\lambda$ , 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  $(0, \mu - 1), (1, \mu - 2), \dots, (\mu - 1, 0)$ , and there are clearly  $\mathcal{O}(\mu) = \mathcal{O}(m)$  of them.

Consider  $(h, k)$  as one such pair. We spell  $P[\lambda + 1 \dots \lambda + h]$  in  $T_{G_\mu}$ . If the whole  $P[\lambda + 1 \dots \lambda + h]$  is spelled successfully, this implies an interval of leaves of  $T_{G_\mu}$  corresponding to strings from  $\tilde{T}[i]$  that share  $P[\lambda + 1 \dots \lambda + h]$  as a prefix. We spell  $P^R[1 \dots k]$  in  $T_{G_\mu}^R$ . If the whole  $P^R[1 \dots k]$  is spelled successfully, this implies an interval of leaves of  $T_{G_\mu}^R$  corresponding to strings from  $\tilde{T}[i]$  that have the same fragment  $(P^R[1 \dots k])^R$ . These two intervals form a rectangle in the grid implied by the leaves of  $T_{G_\mu}$  and  $T_{G_\mu}^R$ . We need to check if these intervals both contain a leaf corresponding to the same prefix of length  $\mu$  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  $(T_{G_\mu}, T_{G_\mu}^R)$ , a 2d range data structure for the set of points  $(x, y)$  such that  $x$  is the rank of a leaf of  $T_{G_\mu}$ ,  $y$  is the rank of a leaf of  $T_{G_\mu}^R$ , and the two leaves correspond to *the same prefix* of length  $\mu$  of a string in  $\tilde{T}[i]$ . For every  $(T_{G_\mu}, T_{G_\mu}^R)$ , this takes  $\mathcal{O}(|G_\mu| \sqrt{\log |G_\mu|})$  time by Lemma 4. For all  $G_\mu$  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 4. Note that all rectangles for  $\lambda$  can be collected in  $\mathcal{O}(m)$  time by spelling  $P[\lambda + 1 \dots \lambda + \mu - 1]$  through  $T_{G_\mu}$  and  $P^R[1 \dots \mu - 1]$  through  $T_{G_\mu}^R$ , 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  $G_\mu$  groups of  $\tilde{T}[i]$ , because  $\sum_\mu |G_\mu| \leq N_i$ .

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  $\lambda$  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

#### Anchor Case: Simple 2d Rectangle Stabbing

**Lemma 8.** *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 5, 2d rectangle stabbing queries can be answered in  $\mathcal{O}(\log n)$  time after  $\mathcal{O}(n \log n)$ -time preprocessing.

Notice that in the case of the 2d rectangle stabbing used in Sect. 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  $\Gamma$  (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  $\Gamma$  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  $AP_{i-1}$ .  $\square$

**Prefix Case: A Special Case of 2d Rectangle Stabbing.** Inspect the example of Fig. 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 9.** *Let  $p_1, \dots, p_h$  and  $q_1, \dots, q_h$  be two permutations of  $[1, h]$ . We denote by  $\Pi$  the set of  $h$  points  $(p_1, q_1), (p_2, q_2), \dots, (p_h, q_h)$  on  $[1, h]^2$ . Further let  $R$  be a collection of  $r$  axis-aligned rectangles  $([u_1, v_1], [w_1, x_1]), \dots, ([u_r, v_r], [w_r, x_r])$ , such that*

$$[u_r, v_r] \subseteq [u_{r-1}, v_{r-1}] \subseteq \dots \subseteq [u_1, v_1] \quad \text{and} \quad [w_1, x_1] \subseteq [w_2, x_2] \subseteq \dots \subseteq [w_r, x_r].$$

*Then we can find out, for every point from  $\Pi$ , 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.  $\square$

**Lemma 10.** *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 9 to get rid of the 2d range data structure. The key is that for every length- $\mu$  suffix  $P[\lambda + 1 .. m]$  of the pattern we can afford to pay  $\mathcal{O}(\mu + |G_\mu|)$  time plus the time to construct  $T_{G_\mu}$  and  $T_{G_\mu}^R$  for set  $G_\mu$ . Because the grid is  $[1, |G_\mu|]^2$ , we exploit the fact that the intervals found by spelling  $P[\lambda + 1 .. \lambda + \mu - 1]$  through  $T_{G_\mu}$  and  $P^R[1 .. \mu - 1]$  through  $T_{G_\mu}^R$ , one letter at a time, are subset of each other, and querying  $\mu$  such rectangles is done in  $\mathcal{O}(\mu + |G_\mu|)$  time by employing Lemma 9. Since we process at most  $m$  distinct length- $\mu$  suffixes of  $P$ , the total time is  $\mathcal{O}(m^2 + N_i)$ , because  $\sum_\mu |G_\mu| \leq N_i$ .  $\square$

### 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 1). We then compute all the occurrences in the Easy Cases using  $\mathcal{O}(N)$  time in total (Sect. 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 10).

Now, to solve the decision version of the problem, we solve the Anchor Cases with the use of the precomputed  $AP_{i-1}$  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 (Sect. 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 Fig. 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 10).
- Standard APE with  $1-AP_{i-1}$  as the input bit vector, computed in  $\mathcal{O}(m^2 + N_i)$  time (Lemma 3).
- Anchor Case computed from  $AP_{i-1}$  in  $\mathcal{O}((m^2 + N_i) \log N_i)$  (Sect. 3.2) or  $\mathcal{O}(m^3 + N_i)$  time (Lemma 8).

If  $N_i \geq m^3$ , the algorithm of Lemma 8 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 [24] and Prefix Case, respectively, in  $\mathcal{O}(m^2 + N_i)$  extra time.

By summing up all these we obtain Theorem 1.

## 4 Open Questions

We leave the following basic questions open for future investigation:

1. Can we design an  $\mathcal{O}(nm^2 + N)$ -time algorithm for 1-EDSM?
2. Can our techniques be efficiently generalized for  $k > 1$  errors?

## References

1. t al Alzamel, M., e.: Degenerate string comparison and applications. In: Parida, L., Ukkonen, E. (eds.) 18th International Workshop on Algorithms in Bioinformatics, WABI 2018, Helsinki, Finland, 20–22 August 2018, LIPIcs, vol. 113, pp. 21:1–21:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). <https://doi.org/10.4230/LIPIcs.WABI.2018.21>
2. Alzamel, M., et al.: Comparing degenerate strings. *Fundam. Informaticae* **175**(1–4), 41–58 (2020). <https://doi.org/10.3233/FI-2020-1947>
3. Amir, A., Keselman, D., Landau, G.M., Lewenstein, M., Lewenstein, N., Rodeh, M.: Text indexing and dictionary matching with one error. *J. Algorithms* **37**(2), 309–325 (2000). <https://doi.org/10.1006/jagm.2000.1104>
4. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with  $k$  mismatches. *J. Algorithms* **50**(2), 257–275 (2004). [https://doi.org/10.1016/S0196-6774\(03\)00097-X](https://doi.org/10.1016/S0196-6774(03)00097-X)
5. Aoyama, K., Nakashima, Y., Inenaga, T., Inenaga, S., Bannai, H., Takeda, M.: Faster online elastic degenerate string matching. In: Navarro, G., Sankoff, D., Zhu, B. (eds.) Annual Symposium on Combinatorial Pattern Matching, CPM 2018, Qingdao, China, 2–4 July 2018, LIPIcs, vol. 105, pp. 9:1–9:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). <https://doi.org/10.4230/LIPIcs.CPM.2018.9>
6. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000). [https://doi.org/10.1007/10719839\\_9](https://doi.org/10.1007/10719839_9)
7. Bernardini, G., Pisanti, N., Pissis, S., Rosone, G.: Pattern matching on elastic-degenerate text with errors. In: 24th International Symposium on String Processing and Information Retrieval (SPIRE), pp. 74–90 (2017). [https://doi.org/10.1007/978-3-319-67428-5\\_7](https://doi.org/10.1007/978-3-319-67428-5_7)
8. Bernardini, G., Gawrychowski, P., Pisanti, N., Pissis, S.P., Rosone, G.: Even faster elastic-degenerate string matching via fast matrix multiplication. In: Baier, C., Chatzigiannakis, I., Flocchini, P., Leonardi, S. (eds.) 46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, Patras, Greece, 9–12 July 2019, LIPIcs, vol. 132, pp. 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). <https://doi.org/10.4230/LIPIcs.ICALP.2019.21>
9. Bernardini, G., Gawrychowski, P., Pisanti, N., Pissis, S.P., Rosone, G.: Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput.* **51**(3), 549–576 (2022). <https://doi.org/10.1137/20M1368033>
10. Bernardini, G., Pisanti, N., Pissis, S.P., Rosone, G.: Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.* **812**, 109–122 (2020). <https://doi.org/10.1016/j.tcs.2019.08.012>
11. Chan, T.M., Larsen, K.G., Patrascu, M.: Orthogonal range searching on the RAM, revisited. In: Hurtado, F., van Kreveld, M.J. (eds.) Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, 13–15 June 2011, pp. 1–10. ACM (2011). <https://doi.org/10.1145/1998196.1998198>
12. Charalampopoulos, P., Iliopoulos, C.S., Liu, C., Pissis, S.P.: Property suffix array with applications in indexing weighted sequences. *ACM J. Exp. Algorithmics* **25**, 1–16 (2020). <https://doi.org/10.1145/3385898>
13. Charalampopoulos, P., Kociumaka, T., Wellnitz, P.: Faster approximate pattern matching: a unified approach. In: Irani, S. (ed.) 61st IEEE Annual Symposium on



- Foundations of Computer Science, FOCS 2020, Durham, NC, USA, 16–19 November 2020, pp. 978–989. IEEE (2020). <https://doi.org/10.1109/FOCS46700.2020.00095>
14. Charalampopoulos, P., Kociumaka, T., Wellnitz, P.: Faster pattern matching under edit distance. CoRR **abs/2204.03087** (2022). <https://doi.org/10.48550/arXiv.2204.03087>, (announced at FOCS 2022)
  15. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput. **17**(3), 427–462 (1988). <https://doi.org/10.1137/0217026>
  16. Cole, R., Gottlieb, L., Lewenstein, M.: Dictionary matching and indexing with errors and don’t cares. In: Babai, L. (ed.) Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 13–16 June 2004, pp. 91–100. ACM (2004). <https://doi.org/10.1145/1007352.1007374>
  17. Cole, R., Hariharan, R.: Approximate string matching: a simpler faster algorithm. SIAM J. Comput. **31**(6), 1761–1782 (2002). <https://doi.org/10.1137/S0097539700370527>
  18. Crochemore, M., Hancart, C., Lecroq, T.: Algorithms on Strings. Cambridge University Press, Cambridge (2007). <https://doi.org/10.1017/CBO9780511546853>
  19. Farach, M.: Optimal suffix tree construction with large alphabets. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, pp. 137–143. IEEE Computer Society (1997). <https://doi.org/10.1109/SFCS.1997.646102>
  20. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with  $O(1)$  worst case access time. J. ACM **31**(3), 538–544 (1984). <https://doi.org/10.1145/828.1884>
  21. Gao, Y., He, M., Nekrich, Y.: Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In: Grandoni, F., Herman, G., Sanders, P. (eds.) 28th Annual European Symposium on Algorithms, ESA 2020, Pisa, Italy (Virtual Conference), 7–9 September 2020, LIPIcs, vol. 173, pp. 54:1–54:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). <https://doi.org/10.4230/LIPIcs.ESA.2020.54>
  22. Gawrychowski, P., Ghazawi, S., Landau, G.M.: On indeterminate strings matching. In: Gørtz, I.L., Weimann, O. (eds.) 31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, Copenhagen, Denmark, 17–19 June 2020, LIPIcs, vol. 161, pp. 14:1–14:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). <https://doi.org/10.4230/LIPIcs.CPM.2020.14>
  23. Gawrychowski, P., Uznanski, P.: Towards unified approximate pattern matching for Hamming and  $l_1$  distance. In: Chatzigiannakis, I., Kaklamanis, C., Marx, D., Sannella, D. (eds.) 45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, Prague, Czech Republic, 9–13 July 2018, LIPIcs, vol. 107, pp. 62:1–62:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). <https://doi.org/10.4230/LIPIcs.ICALP.2018.62>
  24. Grossi, R., Iliopoulos, C.S., Liu, C., Pisanti, N., Pissis, S.P., Retha, A., Rosone, G., Vayani, F., Versari, L.: On-line pattern matching on similar texts. In: Kärkkäinen, J., Radoszewski, J., Rytter, W. (eds.) 28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, Warsaw, Poland, 4–6 July 2017, LIPIcs, vol. 78, pp. 9:1–9:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017). <https://doi.org/10.4230/LIPIcs.CPM.2017.9>
  25. Iliopoulos, C.S., Kundu, R., Pissis, S.P.: Efficient pattern matching in elastic-degenerate strings. Inf. Comput. **279**, 104616 (2021). <https://doi.org/10.1016/j.ic.2020.104616>

26. IUPAC-IUB Commission on Biochemical Nomenclature: Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry* **9**(20), 4022–4027 (1970). [https://doi.org/10.1016/0022-2836\(71\)90319-6](https://doi.org/10.1016/0022-2836(71)90319-6)
27. Landau, G.M., Vishkin, U.: Efficient string matching with  $k$  mismatches. *Theor. Comput. Sci.* **43**, 239–249 (1986). [https://doi.org/10.1016/0304-3975\(86\)90178-7](https://doi.org/10.1016/0304-3975(86)90178-7)
28. Landau, G.M., Vishkin, U.: Fast string matching with  $k$  differences. *J. Comput. Syst. Sci.* **37**(1), 63–78 (1988). [https://doi.org/10.1016/0022-0000\(88\)90045-1](https://doi.org/10.1016/0022-0000(88)90045-1)
29. Na, J.C., Apostolico, A., Iliopoulos, C.S., Park, K.: Truncated suffix trees and their application to data compression. *Theor. Comput. Sci.* **304**(1–3), 87–101 (2003). [https://doi.org/10.1016/S0304-3975\(03\)00053-7](https://doi.org/10.1016/S0304-3975(03)00053-7)
30. Ružić, M.: Constructing efficient dictionaries in close to sorting time. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008. LNCS*, vol. 5125, pp. 84–95. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70575-8\\_8](https://doi.org/10.1007/978-3-540-70575-8_8)
31. Shi, Q., JáJá, J.F.: Novel transformation techniques using  $q$ -heaps with applications to computational geometry. *SIAM J. Comput.* **34**(6), 1474–1492 (2005). <https://doi.org/10.1137/S0097539703435728>
32. The Computational Pan-Genomics Consortium: Computational pan-genomics: status, promises and challenges. *Brief. Bioinf.* **19**(1), 118–135 (2018). <https://doi.org/10.1093/bib/bbw089>