

# Streaming $k$ -mismatch with error correcting and applications

Jakub Radoszewski<sup>1</sup> and Tatiana Starikovskaya<sup>2</sup>

<sup>1</sup> Institute of Informatics, University of Warsaw, Warsaw, Poland, jrad@mimuw.edu.pl

<sup>2</sup> DIENS, École normale supérieure, PSL Research University, Paris, 75005, France,  
tat.starikovskaya@gmail.com

## Abstract

We present a new streaming algorithm for the  $k$ -MISMATCH problem, one of the most basic problems in pattern matching. Given a pattern and a text, the task is to find all substrings of the text that are at the Hamming distance at most  $k$  from the pattern. Our algorithm is enhanced with an important new feature called ERROR CORRECTING, and its complexities for  $k = 1$  and for a general  $k$  are comparable to those of the solutions for the  $k$ -MISMATCH problem by Porat and Porat (FOCS 2009) and Clifford et al. (SODA 2016). In parallel to our research, a yet more efficient algorithm for the  $k$ -MISMATCH problem with the ERROR CORRECTING feature was developed by Clifford et al. (SODA 2019). Using the new feature and recent work on streaming MULTIPLE PATTERN MATCHING we develop a series of streaming algorithms for pattern matching on weighted strings, which are a commonly used representation of uncertain sequences in molecular biology. We also show that these algorithms are space-optimal up to polylog factors.

A preliminary version of this work was published at DCC 2017 conference [24].

## 1 Introduction

In this work we design efficient streaming algorithms for a number of problems of approximate pattern matching. In this class of problems we are given a pattern and a text and wish to find all substrings of the text that are “similar” to the pattern. We assume that the text arrives as a stream, one symbol at a time.

The first small-space streaming algorithms for pattern matching were suggested in the pioneering paper by Porat and Porat in FOCS 2009 [22]. In particular, they showed an algorithm for the PATTERN MATCHING problem, where one must find all exact occurrences of the pattern in the text. For a pattern of length  $m$  their algorithm takes only  $\mathcal{O}(\log m)$  space and  $\mathcal{O}(\log m)$  time per each symbol of the text, and reports all exact occurrences of the pattern in the text as they occur. In 2010, Ergün et al. [14] presented a slightly simpler version of the algorithm, and finally in 2011 the running time of the algorithm was improved to constant by Breslauer and Galil [6]. All of the aforementioned results are randomised Monte Carlo algorithms.

The next logical step was to study the complexity of approximate pattern matching in the streaming model in which we are to find all substrings of the text that are at a small distance from the pattern. The most popular distances are the Hamming distance,  $L_1$ ,  $L_2$ ,  $L_\infty$ -distances, and the edit distance. Unfortunately, the general task of computing these distances for each alignment of the pattern and of the text precisely requires at least  $\Omega(m)$  space. This lower bound holds even for randomised algorithms [11].

However, this is not the case if we allow approximation or if it is sufficient to compute the exact distances only when they are small. Here we focus on approximate pattern matching under the Hamming distance. Let  $m$  be the length of the pattern and  $n$  be the length of the text. If we are interested in computing a  $(1 + \varepsilon)$ -approximation of the Hamming distance for all alignments of the pattern and of the text, then this can be done in  $\mathcal{O}(\varepsilon^{-5} \sqrt{m} \log^4 m)$  space and  $\mathcal{O}(\varepsilon^{-4} \log^3 m)$  time per arriving symbol [13]. If we are only interested in computing the Hamming distances at the alignments where they do not exceed a given threshold  $k$ , which we call the  $k$ -MISMATCH problem, then Porat and Porat [22] showed a randomised streaming algorithm that solves this problem in  $\mathcal{O}(k^3 \log^7 m / \log \log m)$  space and  $\mathcal{O}(k^2 \log^5 m / \log \log m)$  time per arriving symbol; they also presented an algorithm for a special case of

$k = 1$  with  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per arriving symbol. Their result was improved (in terms of the dependency on  $k$ ) to  $\mathcal{O}(k^2 \log^{11} m / \log \log m)$  space and  $\mathcal{O}(\sqrt{k} \log k + \log^5 m)$  time per arriving symbol by Clifford et al. [9].<sup>1</sup> The error probability of all these solutions is at most  $1/\text{poly}(m)$ .

Our first contribution is a new streaming algorithm for the  $k$ -MISMATCH problem. The crucial feature of our algorithm is that, for each alignment where the Hamming distance is at most  $k$ , it can also output the differences of symbols of the pattern and of the text in the mismatching positions. This is particularly surprising as we are not allowed to store a copy of the pattern or of the text in the streaming setting. The  $k$ -MISMATCH problem extended with computing this additional characteristic is called here the  $k$ -MISMATCH WITH ERROR CORRECTING problem. We first develop a solution for  $k = 1$ .

**Theorem 1.1.** *1-MISMATCH WITH ERROR CORRECTING can be solved in  $\mathcal{O}(\log^5 m / \log \log m)$  space and  $\mathcal{O}(\log^5 m / \log \log m)$  time per arrival. The probability of error is at most  $1/\text{poly}(n)$ .*

As a corollary we obtain a  $k$ -MISMATCH WITH ERROR CORRECTING algorithm for arbitrary  $k$  via an existing randomised reduction to the case of  $k = 1$  [22, 9].

**Theorem 1.2.**  *$k$ -MISMATCH WITH ERROR CORRECTING can be solved in  $\mathcal{O}(k^2 \log^{10} m / \log \log m)$  space and  $\mathcal{O}(k \log^8 m / \log \log m)$  time per arrival. The probability of error is at most  $1/\text{poly}(m)$ .*

A comparison of the complexities of our algorithm and the previous algorithms can be found in Table 1. Since in the  $k$ -MISMATCH WITH ERROR CORRECTING problem we need at least  $\Omega(k)$  time per arrival to list the symbol differences, the dependency of the running time of our algorithm on  $k$  is optimal. The time complexity of our  $k$ -MISMATCH WITH ERROR CORRECTING algorithm is better than the time complexity of the  $k$ -MISMATCH algorithm of [22] and worse than that of [9]. Our algorithm also uses less space than the  $k$ -MISMATCH algorithm of [22] (in terms of  $k$ ) and the  $k$ -MISMATCH algorithm of [9]. For the former, it is explained by the fact that we use a more efficient reduction. For the latter, it is because we do not need the involved time-saving techniques of Clifford et al. [9] (as the running time of our algorithm is already almost optimal). We note that a similar problem was considered previously by Porat and Lipsky [23]. They introduced a small-space representation of a stream called a “sketch” that can be efficiently maintained under a symbol change and can be used to compute the Hamming distance between two streams as well as to correct the errors between them. Unfortunately, it is not clear whether their sketches can be efficiently maintained over a sliding window (i.e. for substrings of the text) and therefore we cannot use them in our model.

Algorithm	Space	Time per symbol	ERR. CORR.
Porat and Porat [22]	$\mathcal{O}(k^3 \log^4 m / \log \log m)$	$\mathcal{O}(k^2 \log^9 m / \log \log m)$	No
Clifford et al. [9]	$\mathcal{O}(k^2 \log^{11} m / \log \log m)$	$\mathcal{O}(\sqrt{k} \log k + \log^5 m)$	No
This paper	$\mathcal{O}(k^2 \log^{10} m / \log \log m)$	$\mathcal{O}(k \log^8 m / \log \log m)$	Yes
Clifford et al. [12]	$\mathcal{O}(k \log \frac{n}{k})$	$\mathcal{O}(k + \log \frac{n}{k} (\sqrt{k} \log k + \log^3 n))$	Yes

Table 1: Comparison of the previous solutions to the  $k$ -MISMATCH problem with our solution to the  $k$ -MISMATCH WITH ERROR CORRECTING problem and a parallel contribution by Clifford et al. [12].

In parallel to our research, Clifford et al. [12] presented a new streaming algorithm for the  $k$ -MISMATCH WITH ERROR CORRECTING problem with space complexity  $\mathcal{O}(k \log \frac{n}{k})$  which spends  $\mathcal{O}(k + \log \frac{n}{k} (\sqrt{k} \log k + \log^3 n))$  time per text symbol. Their algorithm introduces a new, much more elaborate way to sketch the pattern and the text. In particular, in contrast to the approach used by Porat and Porat [22] and Clifford et al. [9] their algorithm does not reduce the  $k$ -MISMATCH problem to the 1-MISMATCH problem, but solves the problem immediately for an arbitrary  $k$ .

The ERROR CORRECTING feature is a powerful tool. We demonstrate this by using it to develop efficient streaming algorithms for the problem of pattern matching on weighted strings. A *weighted string* (also known as weighted sequence, position probability matrix or position weight matrix, PWM) is a sequence of probability distributions on the alphabet. Weighted strings are a commonly used

<sup>1</sup>The logarithmic factors are hidden in [9], but can be restored easily from [22, 9].

representation of uncertain sequences in molecular biology. In particular, they are used to model motifs and have become an important part of many software tools for computational motif discovery; see e.g. [25, 26]. In the WEIGHTED PATTERN MATCHING problem we are given a text and a pattern, both of which can be either weighted or regular strings. If either only the text or only the pattern are weighted, the task is to find all alignments of the text and of the pattern where they match with probability above a given threshold  $1/z$ . In the most general case, when both the pattern and the text is weighted, we must find all alignments of the text and of the pattern where there exists a regular string that matches both the text and the pattern with probability at least  $1/z$ . We assume here that the distributions at the respective positions are independent.

As previously, let  $m$  be the length of the pattern,  $n$  be the length of the text and assume a constant-sized alphabet. We are the first to consider the WEIGHTED PATTERN MATCHING problem in the streaming setting. In the offline setting the most commonly studied variant, when the text is a weighted string and the pattern is a regular string, can be solved in  $\mathcal{O}(n \log n)$  time via the Fast Fourier Transform [7] or in  $\mathcal{O}(n \log z)$  time using the suffix array and lookahead scoring [20]. This variant has been also considered in the indexing setting, in which we are to preprocess a weighted text to be able to answer queries for multiple patterns; see [1, 3, 5, 18]. The symmetric variant of the WEIGHTED PATTERN MATCHING problem, when only the pattern is weighted, is closely related to the problem of profile matching [21] and admits  $\mathcal{O}(n \log n)$ -time and  $\mathcal{O}(n \log z)$ -time solutions as well. Finally, the variant when both the text and the pattern are weighted was introduced in [4], where an  $\mathcal{O}(nz^2 \log z)$ -time solution was presented. Later a more efficient  $\mathcal{O}(n\sqrt{z} \log \log z)$ -time solution was devised in [20]. The offline algorithms use  $\Omega(m)$  space and the best indexing solution uses  $\Omega(nz)$  space. A problem of computing Hamming and edit distances for weighted strings has been also considered [2].

We consider each of the three variants of the WEIGHTED PATTERN MATCHING problem. If  $z \geq m$ , the offline algorithms listed above [20, 7] and the black box transformation [8] give  $\mathcal{O}(m)$ -space on-line algorithms. The time complexities are  $\mathcal{O}(\log^2 m)$  for the variants where only the pattern or only the text are weighted, and  $\mathcal{O}(\sqrt{z} \log m \log \log z)$  for the variant when both the pattern and the text are weighted. Below we assume  $z \leq m$ . For the two variants of WEIGHTED PATTERN MATCHING where the text is weighted, our solutions are approximate. Namely, at each alignment we output either “Yes” or “No”. If the pattern matches the fragment of the text, we output “Yes”. If the match probability is between  $(1 - \varepsilon)/z$  and  $1/z$ , we output either “Yes” or “No”, and otherwise we output “No”. If we output “Yes”, we also output a  $(1 - \varepsilon)$ -approximation of the match probability between  $P$  and  $T$ .

We show two series of streaming algorithms for the WEIGHTED PATTERN MATCHING problem. The first one is based on the  $k$ -MISMATCH WITH ERROR CORRECTING problem. Let  $\mathcal{S}_{\log z}$ ,  $\mathcal{T}_{\log z}$ , and  $\mathcal{P}_{\log z}$  be the space and the time complexities and error probability for the  $k$ -MISMATCH WITH ERROR CORRECTING for  $k = \log z$ , a pattern of length  $m$ , and a text of length  $n$ . With Theorem 1.2 we obtain space  $\mathcal{S}_{\log z} = \mathcal{O}(\log^2 z \cdot \log^{10} m / \log \log m)$ , time  $\mathcal{T}_{\log z} = \mathcal{O}(\log z \cdot \log^8 m / \log \log m)$ , and error probability  $\mathcal{P}_{\log z} = 1/\text{poly}(m)$  (the new work of Clifford et al. [12] yields space  $\mathcal{S}_{\log z} = \mathcal{O}(\log z \log m)$ , time  $\mathcal{T}_{\log z} = \mathcal{O}(\log z + \log m(\sqrt{\log z \log \log z} + \log^3 m))$ , and error probability  $\mathcal{P}_{\log z} = 1/\text{poly}(n)$ ).

**Theorem 1.3.** *Assume that  $z \leq m$ . If only the pattern is weighted, there is a streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(\log^2 z) + \mathcal{T}_{\log z}$  time per arrival. If only the text is weighted, the problem can be solved  $(1 - \varepsilon)$ -approximately in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{T}_{\log z}$  time. At each arrival, the algorithms can err with probability  $\mathcal{P}_{\log z}$ .*

Second, we show three streaming algorithms for the WEIGHTED PATTERN MATCHING problem that are based on the recent breakthroughs for the streaming MULTIPLE PATTERN MATCHING problem [10, 17, 16], the last two carried out in parallel with our research. In the MULTIPLE PATTERN MATCHING problem we are given a set of strings (a *dictionary*) and a set of texts that arrive in a streaming fashion. When a new symbol of a text arrives, the algorithm must decide if the current text ends with an occurrence of a string in the dictionary. In the weighted-pattern-regular-text version of WEIGHTED PATTERN MATCHING, we use a single-text-stream version of MULTIPLE PATTERN MATCHING for which the current best algorithm is by Golan and Porat [16], which for a dictionary  $D$  of  $m$ -length strings takes  $\mathcal{O}(|D| \log m)$  space and  $\mathcal{O}(1)$  time per symbol. In the weighted-text-regular-pattern case we apply the PATTERN MATCHING algorithm of Breslauer and Galil [6]. Finally, in the both-weighted case we apply the most general version of MULTIPLE PATTERN MATCHING for which the algorithm of Golan et

al. [16] uses  $\mathcal{O}(|D| \log m)$  shared memory,  $\mathcal{O}(\log |D| \log m)$  space per stream, and  $\mathcal{O}(\log m)$  time per each arriving character of the text.

**Theorem 1.4.** *Assume that  $z \leq m$ . If only the pattern is weighted, there is a streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z \log m)$  space and  $\mathcal{O}(1)$  time per arrival. If only the text is weighted, the problem can be solved  $(1 - \varepsilon)$ -approximately in  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log m))$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  time. Finally, when both the pattern and the text are weighted, there is a  $(1 - \varepsilon)$ -approximation streaming algorithm with space complexity  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log z \log m))$  that uses  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log z \log m))$  time per arrival. At each arrival, the algorithms can err with probability  $1/\text{poly}(n)$ .*

Finally, in Section 3.5 we show that the space complexity of our algorithms is almost optimal.

**Proposition 1.5.** *If  $z \leq m$ , then any streaming algorithm, exact or  $(1 - \varepsilon)$ -approximate, solving one of the three variants of the WEIGHTED PATTERN MATCHING problem must use  $\Omega(z)$  space.*

This is an extended version of a paper that was published at DCC 2017 conference [24].

**Model of computation** We assume that we receive the pattern first and can preprocess it by reading it in a streaming fashion several times. After having preprocessed the pattern we receive the text that arrives as a stream, one symbol at a time. We account for all the space that is used after the preprocessing and cannot afford to store a copy of the text or of the pattern. The indicated error probabilities are per arrival. We assume a constant-sized alphabet  $\Sigma$ . A symbol of a weighted string is a vector of probabilities of the letters in which all entries are of the form  $c^{p/2^{dw}}$ , where  $w$  is the machine word size,  $c$  and  $d$  are constants, and  $p$  is an integer that fits in a constant number of machine words (log-probability model). Additionally, the probability 0 has a special representation. The only operations on probabilities in our algorithms are multiplications and divisions, which can be performed exactly in  $\mathcal{O}(1)$  time in this model.

## 2 $k$ -Mismatch with Error Correcting

In this section we give our solution to the  $k$ -MISMATCH WITH ERROR CORRECTING problem for  $k = 1$  (Theorem 1.1) and for a general value of  $k$  (Theorem 1.2).

### 2.1 Proof of Theorem 1.1: $k = 1$

Let us start with a brief overview of the algorithm. Assume that letters of a string are numbered starting from 1. We reduce the 1-MISMATCH WITH ERROR CORRECTING problem to  $\mathcal{O}(\log m)$  instances of a special case of this problem where the mismatch is required to belong to the *second half* of the pattern. More formally, consider  $\lceil \log m \rceil + 1$  partitions  $P = P_i S_i$ , where  $P_i$  is a prefix of length  $\min\{2^i, m\}$  and  $S_i$  is the remaining suffix, for  $i = 0, 1, \dots, \lceil \log m \rceil$ . We say that a substring of  $T$  is a *right-half 1-mismatch occurrence* of  $P_i$  if either  $i = 0$  and  $P_0$  does not match, or  $i \geq 1$  and the mismatch is at position  $j > 2^{i-1}$  in  $P_i$ .

**Observation 2.1.** *Any 1-mismatch occurrence of  $P$  in  $T$  is a right-half 1-mismatch occurrence of some  $P_i$  followed by an exact occurrence of  $S_i$ .*

We run two parallel processes for each  $i$ . The first process searches for right-half 1-mismatch occurrences of  $P_i$ . After having found a right-half 1-mismatch occurrence, it passes the information about it to the second process that decides if it is followed by an exact occurrence of  $S_i$ .

#### 2.1.1 Preliminaries

**Rabin–Karp fingerprints and 1-mismatch sketches** The Rabin–Karp fingerprint of a string  $X = X[1] \dots X[\ell]$  is defined as  $\phi(X) = (\sum_{i=1}^{\ell} X[i] \cdot r^i) \bmod p$ , where  $p$  is a prime and  $r$  is a random integer in  $\mathbb{F}_p$ . If we choose  $p$  to be large enough, then the collision probability for any two  $\ell$ -length strings  $X$  and  $Y$ , where  $\ell \leq m$ , will be at most  $1/\text{poly}(m)$  [19]. We will also need the following fact which follows immediately from the definition.

**Fact 2.1.** Let  $X, Y$  be two strings and  $Z = XY$  be their concatenation. Assuming that together with a Rabin–Karp fingerprint of a string of length  $\ell$  we store  $r^\ell \bmod p$  and  $r^{-\ell} \bmod p$ , from the Rabin–Karp fingerprints of two of the strings  $X, Y, Z$  one can compute the Rabin–Karp fingerprint of the third string in  $\mathcal{O}(1)$  time using the formula:

$$\phi(Z) = (\phi(X) + r^{|X|} \cdot \phi(Y)) \bmod p.$$

We now introduce a notion of 1-mismatch sketches and show its basic properties.

**Definition 2.2** (1-mismatch sketch). For a prime  $q$ , the 1-mismatch sketch of a string  $X$  is a vector of length  $q$ , where the  $j$ -th element is the Rabin–Karp fingerprint of the subsequence  $X[j]X[j+q]X[j+2q]\dots$

This notion was implicitly used by Porat and Porat [22] in their 1-MISMATCH algorithm. In particular, they showed the following lemma.

**Lemma 2.3** ([22]). The Hamming distance between two  $m$ -length strings  $X$  and  $Y$  is equal to 1 if and only if for each prime  $q \in [\log m, 2 \log m]$  there is exactly one  $j \leq q$  such that subsequences  $X[j]X[j+q]X[j+2q]\dots$  and  $Y[j]Y[j+q]Y[j+2q]\dots$  are not equal. Moreover, knowing which subsequences are not equal, we can determine the mismatch position between  $X$  and  $Y$  in  $\mathcal{O}(\log m)$  time.

Let us list a few simple properties of 1-mismatch sketches that generalise Fact 2.1.

**Lemma 2.4.** Let  $X, Y$  be two strings and  $Z = XY$  be their concatenation. Consider the 1-mismatch sketches of  $X, Y$ , and  $Z$  defined for a prime  $q$ . Then:

- (i) If  $X = Y$ , then their 1-mismatch sketches are equal;
- (ii) Given the 1-mismatch sketches of  $X$  and  $Y$ , we can compute the 1-mismatch sketch of  $Z$  in  $\mathcal{O}(q)$  time;
- (iii) Given the 1-mismatch sketches of  $X$  and  $Z$ , we can compute the 1-mismatch sketch of  $Y$  in  $\mathcal{O}(q)$  time;
- (iv) Given the 1-mismatch sketch of  $X$ , we can compute the 1-mismatch sketch of  $X^\alpha$  in  $\mathcal{O}(q \log \alpha)$  time, where  $\alpha$  is an integer and  $X^\alpha$  is a concatenation of  $\alpha$  copies of  $X$ .

*Proof.* Property (i) is a direct corollary of the definitions of Rabin–Karp fingerprints and 1-mismatch sketches. As for Property (ii), note that we need to compute the Rabin–Karp fingerprints of the at most  $q$  concatenations of pairs of strings. Similarly, for Property (iii) we only need to compute the Rabin–Karp fingerprints of the at most  $q$  strings constructed from  $Y$  given the Rabin–Karp fingerprints of the at most  $q$  strings constructed from  $X$  and their concatenations (in  $Z$ ). Thus, Properties (ii) and (iii) follow from Fact 2.1. Finally, Property (iv) is implied by Property (ii) as we can compute the 1-mismatch sketch of a square of any string given its 1-mismatch sketch in  $\mathcal{O}(q)$  time.  $\square$

**Periodicity in strings** For a string  $Q$ , by  $Q[i, j]$  we denote a substring of  $Q$  equal to  $Q[i] \dots Q[j]$ . We say that a string  $Q$  has a *period*  $p$  if  $Q[i] = Q[i + p]$  for all  $i = 1, \dots, |Q| - p$  (equivalently, if  $Q[1, |Q| - p] = Q[p + 1, |Q|]$ ). If  $Q$  has a period  $p$ , then the string  $Q[1, p]$  is the corresponding *string period* of  $Q$ .

**Lemma 2.5** (Fine and Wilf’s periodicity lemma [15]). If  $Q$  has two periods  $p$  and  $q$  such that  $p + q \leq |Q|$ , then  $Q$  also has a period  $\gcd(p, q)$ .

We will need the following two corollaries of this lemma. First, all the periods of  $Q$  not greater than the half of the length of  $Q$  are multiplicities of the shortest period of  $Q$ . And secondly, if  $Q$  has at least 3 occurrences in a string  $P$  such that  $|P| \leq 2|Q|$ , then all the occurrences of  $Q$  in  $P$  form an arithmetic progression with the difference equal to the shortest period of  $Q$ .

**Streaming algorithm for Pattern Matching** Let us now recall a streaming PATTERN MATCHING algorithm [22, 14, 6] for a pattern  $Q$  and text  $T$  that uses  $\mathcal{O}(\log |Q|)$  space and takes  $\mathcal{O}(\log |Q|)$  time per symbol. The algorithm stores  $\mathcal{O}(\log |Q|)$  levels of positions of  $T$ . Positions in level  $j$  are occurrences of  $Q[1, 2^j]$  in the suffix of the current text  $T$  of length  $2^{j+1}$ . The algorithm stores the Rabin–Karp fingerprints of strings from the beginning of  $T$  up to each of these positions. If there are at least 3 such positions at one level, then, by Lemma 2.5, all the positions form a single arithmetic progression whose difference equals the length of the minimal period of  $Q[1, 2^j]$ . This allows to store the aforementioned information very compactly, using only  $\mathcal{O}(\log |Q|)$  space in total. Finally, the algorithm stores the Rabin–Karp fingerprint of the current text and of all prefixes  $Q[1, 2^j]$ . When a new symbol  $T[q]$  arrives, the algorithm considers the leftmost position  $\ell_j$  in each level  $j$ . If  $q - \ell_j + 1$  is smaller than  $2^{j+1}$ , the algorithm does nothing. Otherwise if the fingerprints imply that  $\ell_j$  is an occurrence of  $Q[1, 2^{j+1}]$ , the algorithm promotes it to the next level, and if  $\ell_j$  is not an occurrence of  $Q[1, 2^{j+1}]$ , the algorithm discards it. When a position reaches the top level, it is an occurrence of  $Q$  and the algorithm outputs it. We note that the algorithm above is not the fastest known solution, but it gives the desired time bounds and is conceptually simple. For more details, see [22, 14, 6].

We now describe our 1-MISMATCH WITH ERROR CORRECTING algorithm in detail.

### 2.1.2 Right-half 1-mismatch occurrences of $P_i$

We start by describing the first process that detects right-half 1-mismatch occurrences of  $P_i$ . When the process finds a new occurrence  $T[q - |P_i| + 1, q]$ , it also computes the 1-mismatch sketches of  $T[q - |P_i| + 1, q]$  and sends them to the second process.

Suppose a new symbol  $T[q]$  has arrived. If  $i = 0$ ,  $P_i = P[1]$  is a single letter string. The process compares  $T[q]$  and  $P[1]$ , and if they are not equal, outputs  $q$  and 1-mismatch sketches of  $T[q]$ .

Suppose now that  $i > 0$ . During the preprocessing phase, we compute the 1-mismatch sketches of  $P_i$  for all primes in  $[\log m, 2 \log m]$ . During the main phase, we run two algorithms for  $P_i$  and  $T$ : the 1-MISMATCH algorithm [22] and the PATTERN MATCHING algorithm. The 1-MISMATCH algorithm [22] identifies 1-mismatch occurrences of  $P_i$  in  $T$ , and for each of them returns the mismatch position. The algorithm uses  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per symbol. However, the algorithm does not know the difference of symbols at the mismatch position, and we will need this information in order to compute the 1-mismatch sketches. To extend the algorithm with the required functionality, we use the PATTERN MATCHING algorithm. If  $T[q - |P_i| + 1, q]$  is a right-half 1-mismatch occurrence of  $P_i$  for  $i > 0$ , then  $P_i[1, 2^{i-1}]$  matches at the position  $q - |P_i| + 1$  exactly. It follows that at time  $q$  the position  $q - |P_i| + 1$  is stored at level  $i - 1$  of the PATTERN MATCHING algorithm, and we know the Rabin–Karp fingerprints of  $T[1, q - |P_i|]$  and  $T[1, q]$ . Therefore, we can compute the Rabin–Karp fingerprint of  $T[q - |P_i| + 1, q]$  in  $\mathcal{O}(1)$  time using Fact 2.1. Let  $j$  be the mismatch position and  $\phi(P_i)$  and  $\phi(T[q - |P_i| + 1, q])$  be the Rabin–Karp fingerprints of  $P_i$  and  $T[q - |P_i| + 1, q]$ , respectively. We use the fingerprints to compute the difference of symbols of  $P_i$  and  $T[q - |P_i| + 1, q]$  at position  $j$ .

**Lemma 2.6.** *Assume that  $X$  and  $Y$  are two strings of length  $m$  that differ only at position  $j$ . Knowing the Rabin–Karp fingerprints of  $X$  and  $Y$ , we can compute  $X[j] - Y[j]$  in  $\mathcal{O}(\log m)$  time.*

*Proof.* Let  $\phi(X)$  and  $\phi(Y)$  be the Rabin–Karp fingerprints of  $X$  and  $Y$ , respectively. Then  $X[j] - Y[j]$  is equal to  $(\phi(X) - \phi(Y)) \cdot r^{-j} \pmod{p}$ , where  $p$  and  $r$  are the integers used in the definition of Rabin–Karp fingerprints. Finally,  $r^{-j} \pmod{p}$  can be computed in  $\mathcal{O}(\log m)$  time.  $\square$

Now that we know the mismatch position and the letter difference, we can compute the 1-mismatch sketches of  $T[q - |P_i| + 1, q]$  in  $\mathcal{O}(\log^2 m / \log \log m)$  time from the 1-mismatch sketches of  $P_i$ .

### 2.1.3 Exact occurrences of $S_i$

If  $i = \lceil \log m \rceil$ ,  $S_i$  is the empty string and therefore the second process is not necessary. Henceforth we assume  $0 \leq i < \lceil \log m \rceil$ . The second process is built on top of the PATTERN MATCHING algorithm for the pattern  $Q = S_i$  and  $T$ . Since for each new position  $q$  the first process tells whether it is preceded by a right-half 1-mismatch occurrence of  $P_i$ , all we need is to carry this information from the level 0 of the PATTERN MATCHING algorithm to the top level. We claim that it suffices to store the 1-mismatch

sketches for a constant number of positions in each level. Using them, we will be able to infer the remaining unstored information.

Consider level  $j$ . Let us assume that the algorithm has read  $T[1, q]$  so far. The progression that the PATTERN MATCHING algorithm currently stores for this level can be a part of a longer progression of occurrences of  $S_i[1, 2^j]$  in  $T$ . More precisely, we consider the maximal arithmetic progression  $R_j$  of occurrences of  $S_i[1, 2^j]$  in  $T[1, q]$  with difference  $\rho_{ij}$  being the shortest period of  $S_i[1, 2^j]$ . Then, if the PATTERN MATCHING algorithm stores at least three occurrences, they form a suffix of the progression  $R_j$ . Otherwise there are at most two occurrences stored, so only the first occurrence may not belong to  $R_j$  but to a previous such maximal progression.

Let  $\ell$  be some occurrence of  $S_i[1, 2^j]$  for which we would like to figure out whether it is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If  $\ell$  is far from the start of the current progression  $R_j$ , then the text preceding  $\ell$  is periodic with period  $\rho_{ij}$  and we can use this fact to infer the 1-mismatch information. So our main concern is the positions  $\ell$  that are at the distance of at most  $|P_i| = 2^i$  from the start of the progression  $R_j$ . We define four positions  $\ell_j^a$ ,  $a = 1, 2, 3, 4$ , relative to the progression  $R_j$  that help us to restore the information in this case. Note that we can easily determine the moment when a new progression  $R_j$  starts, as this is precisely the moment when the difference between two consecutive positions in level  $j$  becomes greater than  $\rho_{ij}$ . We define  $\ell_j^1$  as the first position preceded by a right-half 1-mismatch occurrence of  $P_i$  that was added to level  $j$  since that moment. We further define  $\ell_j^a$  as the leftmost terms in  $R_j$  preceded by a right-half 1-mismatch occurrence of  $P_i$  in  $[\ell_j^1 + (a-1) \cdot 2^{i-2}, \ell_j^1 + a \cdot 2^{i-2}]$  for  $a = 2, 3, 4$ . (If such terms exist; otherwise they are left undefined). A schematic view is given in Fig. 1. The algorithm stores 1-mismatch sketches of  $T[1, \ell_j^a - 2^i - 1]$  and  $T[1, \ell_j^a - 1]$  for each  $a = 1, 2, 3, 4$ .

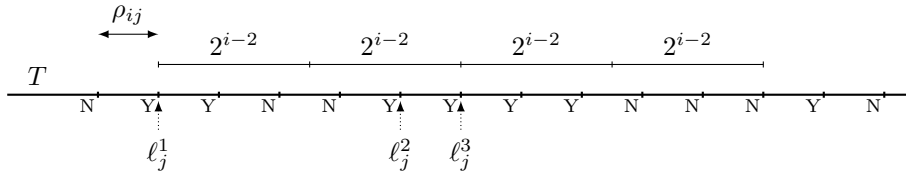


Figure 1: A progression  $R_j$ ; the letter Y represents a 1-mismatch occurrence of  $P_i$  preceding a given position from  $R_j$  and the letter N means there is no such occurrence. The special positions  $\ell_j^1, \ell_j^2, \ell_j^3$  are marked whereas  $\ell_j^4$  is undefined.

Let us mention that when the first element of a new progression  $R_j$  arrives, the algorithm stores all the data for the previous progression  $R_j$  as well. This accounts for the special case of just two occurrences of  $S_i[1, 2^j]$  in the suffix of  $T[1, q]$  belonging to two different progressions. Afterwards the data for the previous progression can be safely discarded.

Besides, the algorithm stores a number of 1-mismatch sketches for the pattern, which are computed during the preprocessing step and in total occupy  $\mathcal{O}(\log^3 m / \log \log m)$  space (for a fixed  $i$ ). First, it stores the 1-mismatch sketches of the shortest string period  $P_i[1, \gamma_i]$  of  $P_i[1, 2^{i-1}]$ . Second, it stores the 1-mismatch sketches of the shortest string period  $S_i[1, \rho_{ij}]$  of  $S_i[1, 2^j]$  for each  $j = 0, 1, \dots, \lfloor \log |S_i| \rfloor$ . Finally, the algorithm stores the 1-mismatch sketches of  $S_i[1, \delta_{ij}]$ , where  $\delta_{ij}$  is the remainder of  $-2^i$  modulo  $\rho_{ij}$ . To compute the periods and the sketches during the preprocessing step we make three passes over the pattern. Over the first two passes we compute the minimal periods of  $P_i[1, 2^{i-1}]$  and  $S_i[1, 2^j]$  for all  $i$  and  $j$  using  $\mathcal{O}(\log^2 m)$  instances of the streaming algorithm [14]. The algorithm requires  $\mathcal{O}(\log^2 m)$  space and computes the minimal period in one pass if the period is smaller than  $m/2$ , and in two passes if otherwise. During the third pass we compute the 1-mismatch sketches.

**Lemma 2.7.** *The algorithm can decide if a position  $\ell \in R_j$  is preceded by a right-half 1-mismatch occurrence of  $P_i$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time and, if this is the case, it can also compute the 1-mismatch sketches of  $T[1, \ell - 2^i - 1]$  and  $T[1, \ell - 1]$ .*

*Proof.* We consider three cases based on the relationship between  $\ell$  and the positions  $\ell_j^a$ .

**Case 1:**  $\ell < \ell_j^1$  or  $\ell_j^a + 2^{i-2} \leq \ell < \ell_j^{a+1}$  for some  $a \in \{1, 2, 3\}$  In this case  $\ell$  cannot be preceded by a 1-mismatch occurrence of  $P_i$  by definition. If  $\ell_j^{a+1}$  is undefined, as the upper bound on  $\ell$  we take  $\ell_j^b$  for

the smallest  $b > a + 1$  that is defined or, if no such  $b > a + 1$  exists,  $\ell_j^1 + 2^i$ .

**Case 2:**  $\ell_j^a \leq \ell < \ell_j^a + 2^{i-2}$  for some  $a \in \{1, 2, 3, 4\}$  Recall that  $\gamma_i$  is the minimal period of  $P_i[1, 2^{i-1}]$ . Let us first show that if  $\ell$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ , then it is also preceded by a 1-mismatch occurrence of  $(P_i[1, \gamma_i])^\alpha P_i$ , where  $\alpha = (\ell - \ell_j^a)/\gamma_i$ . To this end, recall that  $\ell_j^a$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If the same property holds for  $\ell$ , then  $T[\ell_j^a - 2^i, \ell_j^a - 2^{i-1} - 1]$  and  $T[\ell - 2^i, \ell - 2^{i-1} - 1]$  are two occurrences of  $P_i[1, 2^{i-1}]$  that overlap by at least  $2^{i-2}$  positions; see Fig. 2. Therefore  $T[\ell_j^a - 2^i, \ell - 2^i - 1]$  is a power of the minimal period of  $P_i[1, \gamma_i]$  by Lemma 2.5.

Below we explain how to compute the 1-mismatch sketches of  $T[\ell_j^a - 2^i, \ell - 1]$  and of  $(P_i[1, \gamma_i])^\alpha P_i$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time. The algorithm can then use Lemma 2.3 to determine in  $\mathcal{O}(\log^2 m / \log \log m)$  time whether  $\ell$  is preceded by a 1-mismatch occurrence of  $(P_i[1, \gamma_i])^\alpha P_i$  and, if so, to determine the mismatch position. Hence, it can check whether  $\ell$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ . Moreover, if this is the case, the algorithm can compute the 1-mismatch sketch of  $T[1, \ell - 2^i - 1]$  using sketches of  $T[1, \ell_j^a - 2^i - 1]$  and  $(P_i[1, \gamma_i])^\alpha$ .

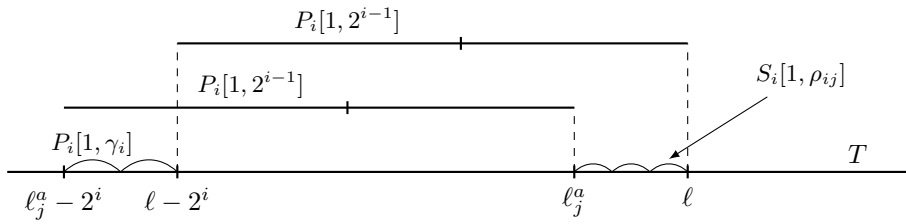


Figure 2: Case 2 of Lemma 2.7. Position  $\ell$  is close to one of the positions  $\ell_j^a$ , which implies that  $T[\ell_j^a - 2^i, \ell - 2^i - 1]$  is a power of  $P_i[1, \gamma_i]$ .

Recall that the algorithm stores the 1-mismatch sketches of  $T[1, \ell_j^a - 1]$ . The algorithm can compute the 1-mismatch sketches of  $T[1, \ell - 1]$  using the 1-mismatch sketches of  $T[1, \ell_j^a - 1]$  and  $S_i[1, \rho_{ij}]$  in  $\mathcal{O}(\log^3 m / \log \log m)$  time via Lemma 2.4(iv) and Lemma 2.4(ii). Finally, the algorithm computes the 1-mismatch sketches of  $T[\ell_j^a - 2^i, \ell - 1]$  in  $\mathcal{O}(\log^2 m / \log \log m)$  time by applying Lemma 2.4(iii) for the 1-mismatch sketches of  $T[1, \ell_j^a - 2^i - 1]$  and  $T[1, \ell - 1]$ . The 1-mismatch sketches of  $(P_i[1, \gamma_i])^\alpha P_i$  are computed from sketches of  $P_i[1, \gamma_i]$  and  $P_i$  using Lemma 2.4(iv) and Lemma 2.4(ii).

**Case 3:**  $\ell \geq \ell_j^1 + 2^i$  In this case  $T[\ell - 2^i, \ell - 1]$  is a suffix of  $T[\ell_j^1, \ell - 1]$ , which is a power of the minimal string period of  $S_i[1, 2^j]$ , that is,  $S_i[1, \rho_{ij}]$ . As we know the 1-mismatch sketches of  $T[1, \ell_j^1 - 1]$ ,  $S_i[1, \rho_{ij}]$ , and  $S_i[1, \delta_{ij}]$  (recall that  $\delta_{ij}$  is the remainder of  $-2^i$  modulo  $\rho_{ij}$ ), the algorithm can use Lemma 2.4 to compute the 1-mismatch sketches of  $T[1, \ell - 1]$  (by extending  $T[1, \ell_j^1 - 1]$  with a power of  $S_i[1, \rho_{ij}]$ ) and  $T[1, \ell - 2^i - 1]$  (by subtracting the sketches of a power of  $S_i[1, \rho_{ij}]$  of exponent  $\lceil 2^i / \rho_{ij} \rceil$  and adding the sketches of  $S_i[1, \delta_{ij}]$ ) in  $\mathcal{O}(\log^3 m / \log \log m)$  time; see Fig. 3. It can then use them to determine whether  $T[\ell - 2^i, \ell - 1]$  is a 1-mismatch occurrence of  $P_i$ .

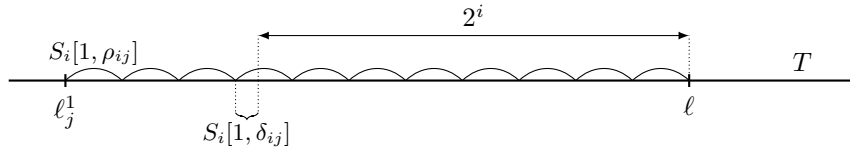


Figure 3: Case 3 of Lemma 2.7. Position  $\ell$  is far enough from  $\ell_j^1$ .

□

The algorithm uses the lemma both to compile the output and to update the levels. When the algorithm encounters a position  $\ell$  in the top level that is preceded by a right-half 1-mismatch occurrence



of  $P_i$  and followed by an exact occurrence of  $S_i$ , the algorithm outputs it together with the required difference of symbols, which is computed from the 1-mismatch sketches of  $T[\ell - 2^i + 1, \ell]$  using Lemma 2.3 to determine the mismatch position and then Lemma 2.6. Let us show how the algorithm updates the levels. When a new symbol  $T[q]$  arrives and  $T[q] = S_i[1]$ , the algorithm adds  $q$  to the level 0. If we know from the first process that  $q$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ , then the algorithm also tries to update the  $\ell_0^a$  values and possibly retains the 1-mismatch sketches of  $T[1, q - 2^i - 1]$  and  $T[1, q - 1]$  output by the first process. The algorithm then updates each of the remaining levels in turn. To update level  $j$ , it considers the leftmost position  $\ell_j$  in this level. If the Rabin–Karp fingerprints imply that it is an occurrence of  $S_i[1, 2^{j+1}]$ , the algorithm promotes it to the next level, and otherwise discards it. In the former case the algorithm uses Lemma 2.7 to check whether  $\ell_j$  is preceded by a right-half 1-mismatch occurrence of  $P_i$ . If it is, it computes the 1-mismatch sketches and updates the positions  $\ell_{j+1}^a$ .

#### 2.1.4 Complexity

For each  $i$  we run the two processes in parallel. The bottleneck of the first process is the 1-MISMATCH algorithm; it uses  $\mathcal{O}(\log^4 m / \log \log m)$  space and  $\mathcal{O}(\log^3 m / \log \log m)$  time per symbol. The time complexity of the second process is bounded by  $\mathcal{O}(\log m)$  applications of the algorithm of Lemma 2.7 (one per level) and is  $\mathcal{O}(\log^4 m / \log \log m)$  per symbol. The space complexity of the second process is  $\mathcal{O}(\log^3 m / \log \log m)$ . Therefore, both the space and the time that our 1-MISMATCH WITH ERROR CORRECTING algorithm uses per symbol is  $\mathcal{O}(\log^5 m / \log \log m)$ .

## 2.2 Proof of Theorem 1.2: general value of $k$

Theorem 1.2 follows from Theorem 1.1 via a randomised reduction. The first variant of this reduction, which was deterministic, was presented by Porat and Porat [22], who used it to reduce the  $k$ -MISMATCH problem to the 1-MISMATCH problem. It was further made more space-efficient at a return of slightly higher error probability by Clifford et al. [9]. The main idea of this reduction is to consider a number of partitions of the pattern into  $\mathcal{O}(k \log^2 m)$  subpatterns. By defining the partitions appropriately, we can guarantee that at each alignment where the Hamming distance is small, each mismatch will correspond to a 1-mismatch occurrence of some subpattern. This lets us apply the 1-MISMATCH WITH ERROR CORRECTING algorithm to find all such alignments and to restore the data for them. We give a full description of the reduction below.

We start by filtering out the locations where the Hamming distance is large. To this end, we use a randomised algorithm for the following *2-approximate  $k$ -mismatch problem*. Let  $H$  be the true Hamming distance at a particular alignment of the pattern and the text. The algorithm outputs “Yes” if  $H \leq k$ , either “Yes” or “No” if  $k < H \leq 2k$ , and “No” if  $H > 2k$ .

**Lemma 2.8** ([9]). *Given a pattern  $P$  of length  $m$  and a streaming text arriving one symbol at a time, there is a randomised  $\mathcal{O}(k^2 \log^6 m)$ -space algorithm which takes  $\mathcal{O}(\log^5 m)$  worst-case time per arriving symbol and solves the 2-approximate  $k$ -mismatch problem. The probability of error is at most  $1/m^2$ .*

Next, we consider  $\ell = \lfloor \log m \rfloor$  partitions of the pattern into equispaced subpatterns. More precisely, we select  $\ell$  random primes from the interval  $[k \log^2 m, 34k \log^2 m]$ . For each prime  $p_i$  the pattern  $P$  is then partitioned into  $p_i$  subpatterns  $P_{i,j} = P[j]P[p_i + j]P[2p_i + j] \dots$ , where  $j = 1, \dots, p_i$ . Consider a particular location  $q$  in the text and the set of all mismatches between  $P$  and  $T[q - m + 1, q]$ . Let us call a mismatch *isolated* if it is the only mismatch between some subpattern  $P_{i,j}$  and the corresponding subsequence of  $T[q - m + 1, q]$ , and let  $M_q$  be the set of all isolated mismatches at an alignment  $q$ .

**Lemma 2.9** ([9]). *If  $|M_q| \leq 2k$ , the set  $M_q$  contains all mismatches between  $P$  and  $T[q - m + 1, q]$  with probability at least  $1 - 1/m^2$ .*

To finalise the reduction, we partition the text  $T$  into  $p_i$  equispaced substreams  $T_{i,j}$ , where  $j = 1, \dots, p_i$ , for each prime  $p_i$ . We run the 1-MISMATCH WITH ERROR CORRECTING algorithm for all  $\sum_i p_i^2 = \mathcal{O}(k^2 \log^5 m)$  subpattern/substream pairs  $(P_{i,j_1}, T_{i,j_2})$ . At each location where the algorithm for the 2-approximate problem outputs “Yes”, we retrieve all isolated mismatches and the data using the appropriate  $p_i$  instances of the 1-MISMATCH WITH ERROR CORRECTING problem for each  $i$ . This completes the description of our solution to the  $k$ -MISMATCH WITH ERROR CORRECTING problem.

In total, we use  $\mathcal{O}(k^2 \log^{10} m / \log \log m)$  space. To analyse the time complexity, note that when a new symbol of  $T$  arrives, we need to send it to  $\mathcal{O}(\log m)$  substreams only (one for each prime) and run the next step for each of the  $\mathcal{O}(k \log^2 m)$  instances of the 1-MISMATCH WITH ERROR CORRECTING problem on each of these substreams, which requires  $\mathcal{O}(k \log^8 m / \log \log m)$  time per symbol.

### 3 Applications — Weighted Pattern Matching

Below we present the proofs of Theorems 1.3 and 1.4 that give two series of streaming algorithms for the WEIGHTED PATTERN MATCHING problem. We present the proofs in parallel. We conclude in Section 3.5 with a proof of Proposition 1.5 that states an  $\Omega(z)$  lower bound on the space complexities of such algorithms.

#### 3.1 Preliminaries

We first introduce a notion of a heavy string  $\mathcal{H}(P)$  of a weighted string  $P$ .

**Definition 3.1.** For a weighted string  $P$ , by  $\mathcal{H}(P)$  we denote a regular string obtained from  $P$  by choosing at each position the symbol with the maximum probability (ties are handled arbitrarily).

The observation below, initially introduced in [20], shows a key property of this notation.

**Observation 3.2.** If a string  $S$  matches a weighted string  $P$  with probability at least  $1/z$ , then the number of mismatches between  $\mathcal{H}(P)$  and  $S$  is at most  $\log z$ .

*Proof.* It follows from the fact that at each mismatch position the probability of  $P$  and  $S$  to match is at most  $1/2$ .  $\square$

**Observation 3.3.** The total number of strings that match a weighted string  $P$  with probability at least  $1/z$  is at most  $z$ .

*Proof.* The sum of the match probabilities over all such strings cannot exceed 1. The claim follows.  $\square$

#### 3.2 Case 1 — only pattern is weighted

We start by presenting two algorithms for the WEIGHTED PATTERN MATCHING problem for a weighted pattern and a regular text.

##### 3.2.1 Solution via $k$ -Mismatch with Error Correcting

Alongside Observation 3.2, the main idea of our solution is to find all  $\log z$ -occurrences of  $\mathcal{H}(P)$  in  $T$ , and then to filter out those corresponding to the alignments where the match probability is too small.

Let  $D$  be the set of all (regular) strings that match the pattern with probability at least  $1/z$ . If a substring of the text  $T$  matches  $P$  with probability  $\geq 1/z$ , it must belong to the set  $D$ . During the preprocessing phase, we do not compute the set  $D$  itself, but would like to compute a set  $M$  of mismatches between  $\mathcal{H}(P)$  and the strings in  $D$ . More formally:

$$M = \{(i, a) : a = S[i] \neq \mathcal{H}(P)[i], S \in D\}.$$

**Observation 3.4.**  $|M| \leq z - 1$ .

*Proof.* From each mismatch  $(i, a) \in M$  one can produce a regular string that matches  $P$  with probability  $\geq 1/z$  by taking  $\mathcal{H}(P)$  and replacing  $\mathcal{H}(P)[i]$  by  $a$ . Due to Observation 3.3 and the fact that  $\mathcal{H}(P) \in D$ , there are at most  $z - 1$  such strings in  $D$ .  $\square$

**Example 3.5.** Consider the weighted pattern  $P$  corresponding to the weighted string  $X$  from Table 2. We have  $\mathcal{H}(P) = \text{ABAB}$ . Let  $z = 8$ .

We have  $D = \{\text{ABAB}, \text{ABBB}, \text{CBAB}\}$  with the probabilities  $\frac{1}{6}$ ,  $\frac{1}{8}$ , and  $\frac{1}{8}$ , respectively. Thus  $M = \{(1, \text{C}), (3, \text{B})\}$  (the mismatches are underlined).

$i$	1	2	3	4
probability of A	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{6}$
probability of B	$\frac{1}{8}$	1	$\frac{3}{8}$	$\frac{2}{3}$
probability of C	$\frac{3}{8}$	0	$\frac{1}{8}$	$\frac{1}{6}$

Table 2: A weighted string  $X$  of length 4 over  $\Sigma = \{A, B, C\}$ .

To perform the preprocessing step in a streaming fashion with small space usage, we compute a slightly larger set  $M'$  that consists of  $z - 1$  pairs  $(i, a)$  for  $i \in \{1, \dots, m\}$  and  $a \in \Sigma$  with the greatest value of  $p(i, a) = \frac{\Pr[P[i]=a]}{\Pr[P[i]=\mathcal{H}(P)[i]]}$  (if there are less than  $z - 1$  such pairs in total, the set  $M'$  contains all such pairs).

**Observation 3.6.**  $M \subseteq M'$ .

*Proof.* Assume to the contrary that  $(i, a) \in M \setminus M'$ . This implies that  $|M'| = \lfloor z \rfloor - 1$ . Using the same argument as in the proof of Observation 3.4, from every pair in  $M' \cup (i, a)$  one can obtain a different string from  $D$ . This would yield  $|D| > z - 1$ , which contradicts Observation 3.3.  $\square$

**Example 3.7.** For the weighted pattern from Example 3.5, one might have

$$M' = \{(1, C, \frac{3}{4}), (3, B, \frac{3}{4}), (1, B, \frac{1}{4}), (3, C, \frac{1}{4}), (4, A, \frac{1}{4}), (4, C, \frac{1}{4}), (2, A, 0)\}.$$

The elements of  $M'$  are ordered by  $p(i, a)$ .

Each element of  $M'$  is stored as a triple  $(i, \mathcal{H}(P)[i] - a, p(i, a))$  in a priority queue ordered by  $p(i, a)$ . To construct the set  $M'$ , for  $i = 1, \dots, m$  we insert all  $(i, a)$  with  $a \neq \mathcal{H}(P)[i]$  into the priority queue. We keep the size of  $M'$  not greater than  $z - 1$  by removing the elements with the smallest  $p(i, a)$  if needed. In the end we compute a balanced binary search tree that indexes all the mismatches from  $M'$  with pairs  $(i, \mathcal{H}(P)[i] - a)$ . We also compute and store the probability  $\pi$  that  $\mathcal{H}(P)$  matches  $P$ . The time needed for the preprocessing step is  $\mathcal{O}(z \log z)$  and the total space consumption is  $\mathcal{O}(z)$ .

During the main phase, we run a  $k$ -MISMATCH WITH ERROR CORRECTING algorithm for  $k = \log z$ ,  $\mathcal{H}(P)$ , and  $T$ . From Observation 3.2 it follows that the algorithm will report all alignments where  $P$  and  $T$  match with probability at least  $1/z$ , but it might report some other alignments, and we need to filter them out. Recall that every occurrence found by the  $k$ -MISMATCH WITH ERROR CORRECTING algorithm applied for  $\mathcal{H}(P)$  and  $T$  with  $k = \log z$  is reported together with the at most  $k$  mismatch positions and the corresponding letter differences. Therefore, we can use the balanced binary search tree that stores the set  $M'$  to compute the match probability by updating  $\pi$  with the probabilities of the at most  $k = \log z$  mismatches. If any of the mismatches is not present in  $M'$ , then by Observation 3.6 the candidate does not belong to the set  $D$  and its match probability is certainly below the  $1/z$  threshold. Thus the time needed to retrieve the match probability (provided that it is at least  $1/z$ ) is  $\mathcal{O}(\log^2 z)$ . With this we arrive at the first algorithm of Theorem 1.3:

**Proposition 3.8.** *If only the pattern is weighted, there is a streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(\log^2 z) + \mathcal{T}_{\log z}$  time per arrival. At each arrival, the algorithm can err with probability  $\mathcal{P}_{\log z}$ .*

Let us note that, in the considered case of  $z < m$ ,  $\mathcal{T}_{\log z} = \Omega(\log^2 z)$  both for the algorithm of Theorem 1.2 and the algorithm of Clifford et al. [12].

### 3.2.2 Solution via Multiple Pattern Matching

As in the previous subsection, let  $D$  be the set of all (regular) strings that match the pattern with probability at least  $1/z$ . Let us recall that if a substring of the text  $T$  matches  $P$  with probability  $\geq 1/z$ , it must belong to the set  $D$ . To identify such substrings, we will use the streaming MULTIPLE PATTERN MATCHING algorithm for the set  $D$  and the text  $T$ . The current best algorithm is by Golan and Porat [16], which for a dictionary  $D$  of  $m$ -length strings takes  $\mathcal{O}(|D| \log m)$  space and  $\mathcal{O}(1)$  time per symbol. The algorithm is randomised and has error probability  $1/\text{poly}(n)$ .

We arrive at the first algorithm of Theorem 1.4:

**Proposition 3.9.** *If only the pattern is weighted, there is a streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z \log m)$  space and  $\mathcal{O}(1)$  time per arrival. At each arrival, the algorithm can err with probability  $1/\text{poly}(n)$ .*

### 3.3 Case 2 — only text is weighted

In this section we show two  $(1 - \varepsilon)$ -approximate solutions to the WEIGHTED PATTERN MATCHING problem for a regular pattern  $P$  and a weighted text  $T$ . We assume that  $\varepsilon < 1/2$ . If  $\varepsilon > 1/2$ , we can use the  $1/2$ -approximation algorithm that has the same asymptotic complexity and better approximation factor. We start by giving a definition of maximal matching suffixes that will play a crucial role in both algorithms.

**Definition 3.10.** *A maximal matching suffix of a weighted string  $T$  is a (regular) string  $S$  such that  $S$  matches  $T[|T| - |S| + 1, |T|]$  with probability at least  $1/(2z)$  and either  $|S| = |T|$  or any string  $aS$ , for  $a \in \Sigma$ , matches  $T[|T| - |S|, |T|]$  with probability smaller than  $1/(2z)$ .*

**Remark 3.11.** *The reason for selecting the cut-off value of  $1/(2z)$  will become clear later (in Lemma 3.16, where we need the cut-off to be at most  $(1 - \varepsilon)/z$ ).*

We will use the observation below; intuitively, it follows from the fact that the sum of probabilities of maximal matching suffixes of a string does not exceed 1.

**Observation 3.12** ([1]). *A weighted string has at most  $2z$  maximal weighted suffixes.*

**Example 3.13.** *For the weighted text  $T$  corresponding to the weighted string  $X$  from Table 2 and  $z = 8$ , the set of maximal matching suffixes is:*

$$\{\text{ABAB } (\frac{1}{6}), \text{ABBB } (\frac{1}{8}), \text{CBAB } (\frac{1}{8}), \text{CBBB } (\frac{3}{32}), \text{BCB } (\frac{1}{12}), \text{BAA } (\frac{1}{12}), \text{BAC } (\frac{1}{12}), \text{BBA } (\frac{1}{16}), \text{BBC } (\frac{1}{16})\}.$$

*Mismatches with  $\mathcal{H}(T)$  are underlined and matching probabilities of each suffix are listed.*

Importantly, if  $P$  matches  $T[q - m + 1, q]$  with probability  $\geq 1/z$ , then there is a maximal matching suffix  $S$  of  $T[1, q]$  that ends with  $P$ . Moreover, the match probability between  $P$  and  $T[q - m + 1, q]$  is equal to the match probability between the  $m$ -length suffix of  $S$  and  $T[q - m + 1, q]$ . To be able to compute the latter, we introduce a new problem which we refer to as the SLIDING WINDOW PRODUCT problem. In this problem we are given a stream of numbers from  $[0, 1]$  and an integer  $m$ . Each time a new number arrives, we must update and output the product of numbers in the  $m$ -length suffix of the stream. We will develop a  $(1 - \varepsilon)$ -approximation solution to the SLIDING WINDOW PRODUCT problem.

We summarize our two solutions as Algorithm 1. The only difference between them is how we find the maximal matching suffixes that end with  $P$ . In the following subsections we explain each step in detail.

---

#### Algorithm 1 WEIGHTED PATTERN MATCHING — only text is weighted

---

```

for each new text symbol  $T[q + 1]$  do
  Update the set of maximal matching suffixes
  for each maximal matching suffix  $S$  do
    Run the next step of SLIDING WINDOW PRODUCT
  end for
  if there is a maximal matching suffix  $S$  that ends with  $P$  then
    Use the output of SLIDING WINDOW PRODUCT for  $S$  to compute a  $(1 - \varepsilon)$ -approximation  $p$  of
    the match probability between  $P$  and  $T$ 
    If  $p \geq (1 - \varepsilon)/z$ , report an occurrence
  end if
end for

```

---

### 3.3.1 Computing maximal matching suffixes

We maintain each maximal matching suffix  $S$  of  $T[1, q]$  as a stream. The stream represents letters  $s_i$ ,  $i \leq q$ , that correspond to  $S[i + |S| - q]$  and probabilities  $x_i$  that are equal to the match probability between  $s_i$  and  $T[i]$ . The letters  $s_i$  for  $1 \leq i \leq q - |S|$  can be arbitrary (they will be implied by the previous steps of the algorithm) and probabilities  $x_i$  for  $1 \leq i \leq q - |S|$  correspond to the matching probability of these letters and  $T[i]$ . Each of the streams will be stored using only  $\mathcal{O}(\log_{1/(1-\varepsilon)} z)$  space (in particular, we do not store all  $s_i$  and  $x_i$  explicitly).

We call a position  $i$  in the stream a *mismatch position* if  $s_i \neq \mathcal{H}(T)[i]$ . Let  $r$  be the rightmost mismatch position in the stream such that  $\prod_{i \geq r} x_i < 1/(2z)$ . If such a position does not exist, we put  $r = 0$ . By Observation 3.2, there are at most  $\log z + 1$  mismatch positions to the right of  $r$ . We index the stream by these mismatch positions and the differences between  $s_i$  and  $\mathcal{H}(T)[i]$  in these positions. We also store the product of the probabilities located to the right of each of these at most  $\log z + 1$  mismatch positions.

When a new text symbol  $T[q + 1]$  arrives, we first create  $|\Sigma|$  copies of each stream. We then add  $\Pr[T[q + 1] = b]$  for each  $b \in \Sigma$  to the  $b$ -th copy of each stream and update the mismatches, the indices, and the related information in  $\mathcal{O}(\log z)$  time using the stored products of probabilities. At this moment the value  $r$  in some streams might increase. Furthermore, there might appear “duplicate” streams with equal indices. Duplicate streams correspond to a single maximal matching suffix and possibly its suffixes (which are not maximal matching suffixes of  $T[1, q + 1]$ ). We sort the streams by building a trie on their indices and delete the duplicates, leaving for each stream index one stream with the smallest position  $r$ . This takes  $\mathcal{O}(z \log z)$  space and time in total.

**Example 3.14.** *Let us consider an extension of the weighted string from Table 2 by a single position that is shown in Table 3.*

$i$	1	2	3	4	5
probability of A	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{2}{3}$
probability of B	$\frac{1}{8}$	1	$\frac{3}{8}$	$\frac{2}{3}$	$\frac{1}{3}$
probability of C	$\frac{3}{8}$	0	$\frac{1}{8}$	$\frac{1}{6}$	0

Table 3: A weighted string  $X'$  of length 5 over  $\Sigma = \{A, B, C\}$ .

Table 4 shows how the first five maximal matching suffixes from Example 3.13 are extended with position 5. Letters in gray are not part of a maximal matching suffix, but they are still present in the stream. The matching probability of the matching suffix is given for reference (in brackets); this value is not stored.

Amongst the streams with stream index (5, B), the first one has  $r = 0$ , the second has  $r = 1$ , and the third has  $r = 3$  (and, notably, it does not correspond to a maximal matching suffix). Thus, the first one of these streams is retained.

Finally, for each of the streams we store a data structure for the approximate solution of SLIDING WINDOW PRODUCT. It takes  $\mathcal{O}(\log_{1/(1-\varepsilon)} z)$  space per stream, so storing and updating all of them upon a letter arrival takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  space and time.

### 3.3.2 Approximate solution to Sliding Window Product

Recall that in the SLIDING WINDOW PRODUCT problem we are given a stream of numbers from  $[0, 1]$  and an integer  $m$ . Each time a new number arrives, we must update and output the product of numbers in the  $m$ -length suffix of the stream. We give a  $(1 - \varepsilon)$ -approximation algorithm for the problem. The algorithm may output either a number or “No”. If it outputs a number  $y$ , then the product of the numbers in the  $m$ -length suffix of the stream is between  $y$  and  $y/(1 - \varepsilon)$ . Otherwise, the product is less than  $(1 - \varepsilon)/z$ .

**Lemma 3.15.** *For a stream of numbers  $\{x_i\}_{i=1}^{\infty}$ , where  $x_i \in [0, 1]$ , arriving one at a time, and a window width  $m$ , there is a deterministic  $(1 - \varepsilon)$ -approximation algorithm that takes  $\mathcal{O}(\log_{1/(1-\varepsilon)} z)$  space and  $\mathcal{O}(1)$  time per arrival and solves the SLIDING WINDOW PRODUCT problem.*

before extension					after extension					stream index	
1	2	3	4		1	2	3	4	5		
$A\frac{1}{2}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$(\frac{1}{6})$	$A\frac{1}{2}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$A\frac{2}{3}$	$(\frac{1}{9})$	$\emptyset$
					$A\frac{1}{2}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$\underline{B}\frac{1}{3}$	$(\frac{1}{9})$	(5, B)
$A\frac{1}{2}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$(\frac{1}{8})$	$A\frac{1}{2}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$A\frac{2}{3}$	$(\frac{1}{12})$	(3, B)
					$A\frac{1}{2}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$\underline{B}\frac{1}{3}$	$(\frac{1}{12})$	(3, B), (5, B)
$\underline{C}\frac{3}{8}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$(\frac{1}{8})$	$\underline{C}\frac{3}{8}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$A\frac{2}{3}$	$(\frac{1}{12})$	(1, C)
					$\underline{C}\frac{3}{8}$	B1	$A\frac{1}{2}$	$B\frac{2}{3}$	$\underline{B}\frac{1}{3}$	$(\frac{1}{9})$	(5, B), duplicate
$\underline{C}\frac{3}{8}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$(\frac{3}{32})$	$\underline{C}\frac{3}{8}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$A\frac{2}{3}$	$(\frac{1}{16})$	(1, C), (3, B)
					$\underline{C}\frac{3}{8}$	B1	$\underline{B}\frac{3}{8}$	$B\frac{2}{3}$	$\underline{B}\frac{1}{3}$	$(\frac{1}{12})$	(3, B), (5, B), duplicate
$A\frac{1}{2}$	B1	$\underline{C}\frac{1}{2}$	$B\frac{2}{3}$	$(\frac{1}{12})$	$A\frac{1}{2}$	B1	$\underline{C}\frac{1}{8}$	$B\frac{2}{3}$	$A\frac{2}{3}$	$(\frac{1}{16})$	(3, C)
					$A\frac{1}{2}$	B1	$\underline{C}\frac{1}{8}$	$B\frac{2}{3}$	$\underline{B}\frac{1}{3}$	$(\frac{2}{9})$	(5, B), duplicate

Table 4: Extensions of maximal matching suffixes from Example 3.13 by position 5 with letter probabilities  $A: \frac{2}{3}, B: \frac{1}{3}, C: 0$ .

*Proof.* At time  $q$  the algorithm maintains a queue that consists of at most  $M(z) = 2 \lceil \log_{1-\varepsilon}((1-\varepsilon)/z) \rceil$  intervals  $[i_1, i_2 - 1], [i_2, i_3 - 1], \dots, [i_{k(q)}, i_{k(q)+1} - 1]$  where  $i_{k(q)+1} = q + 1$ , such that  $1 \leq i_1 < i_2 < \dots < i_{k(q)} < i_{k(q)+1}$ , which obeys the following invariant.

- (i) All intervals  $[i_j, i_{j+1} - 1]$  for  $j \geq 2$  are subintervals of  $[q - m + 1, q]$ , and  $[i_1, i_2 - 1]$  has a non-empty intersection with  $[q - m + 1, q]$ ;
- (ii) If  $x_{i_j} < 1 - \varepsilon$ , then  $i_{j+1} = i_j + 1$ ;
- (iii) Otherwise,  $x_{i_j} \cdot x_{i_{j+1}} \cdot \dots \cdot x_{i_{j+1}-1} \geq 1 - \varepsilon$  and either  $j = k(q)$  or  $x_{i_j} \cdot x_{i_{j+1}} \cdot \dots \cdot x_{i_{j+1}} < 1 - \varepsilon$ .

The algorithm stores the product of numbers in each interval and the total product of numbers in all intervals. When  $x_{q+1}$  arrives, it updates the family of intervals in the following way. Let  $\pi$  be the product of numbers in the interval  $[i_{k(q)}, q]$ . If  $\pi \cdot x_{q+1} \geq 1 - \varepsilon$ , then the algorithm extends the interval  $[i_{k(q)}, q]$  by the element  $x_{q+1}$ . Otherwise, it creates a new interval  $[q + 1, q + 1]$ . If the number of intervals becomes larger than  $M(z)$  or if  $i_2 \leq q - m$ , the algorithm deletes the leftmost interval  $[i_1, i_2 - 1]$ . Finally, it updates the total product of numbers in the intervals. Let us now explain how the algorithm exploits the intervals.

Assume that  $q \geq m$ . Note that the product of numbers in each two consecutive intervals is at most  $1 - \varepsilon$ . Therefore, if  $q - m + 1 < i_1$ , then the product of the last  $m$  numbers is smaller than

$$(1 - \varepsilon)^{\lfloor \frac{1}{2} M(z) \rfloor} = (1 - \varepsilon)^{\lceil \log_{1-\varepsilon}((1-\varepsilon)/z) \rceil} \leq (1 - \varepsilon)^{\log_{1-\varepsilon}((1-\varepsilon)/z)} = \frac{1-\varepsilon}{z}$$

and the algorithm outputs “No”. Otherwise from the invariant it follows that  $q - m + 1 \in [i_1, i_2 - 1]$ . In this case we output the total product of the numbers in the intervals. If  $[i_1, i_2 - 1]$  is a singleton interval (i.e.  $i_1 = i_2 - 1$ ), then  $[i_1, i_{k(q)+1} - 1] = [q - m + 1, q]$ , and the output equals  $x_{q-m+1} \cdot x_{q-m+2} \cdot \dots \cdot x_q$  exactly. If  $[i_1, i_2 - 1]$  is not a singleton interval, then the product of numbers in it is at least  $1 - \varepsilon$ . Therefore, the output will be a  $(1 - \varepsilon)$ -approximation of  $x_{q-m+1} \cdot x_{q-m+2} \cdot \dots \cdot x_q$ . The space complexity follows from the fact that  $M(z) = \mathcal{O}(\log_{1/(1-\varepsilon)} z)$ .  $\square$

**Lemma 3.16.** *Consider the stream  $\{x_i\}_{i=1}^q$  related to maximal matching suffix  $S$  of  $T[1, q]$  and let  $q \geq m$ . If the algorithm for SLIDING WINDOW PRODUCT returns “No” or a number that is smaller than  $(1 - \varepsilon)/z$ , then either  $|S| < m$  or the probability  $\pi$  that  $S[|S| - m + 1, |S|]$  matches  $T[q - m + 1, q]$  is below  $1/z$ . Otherwise,  $|S| \geq m$  and the result  $y$  of the algorithm satisfies  $y \leq \pi \leq y/(1 - \varepsilon)$ .*

*Proof.* Let us denote  $\pi' = x_{q-m+1} \cdot \dots \cdot x_q$ . If the SLIDING WINDOW PRODUCT algorithm returns “No”, then  $\pi' < (1 - \varepsilon)/z$ . Then, indeed, either  $|S| < m$  or  $\pi = \pi' < (1 - \varepsilon)/z$ .

Assume that the SLIDING WINDOW PRODUCT algorithm returns a number  $y$ . Then  $y \leq \pi' \leq y/(1-\varepsilon)$ . If  $y < (1-\varepsilon)/z$ , then  $\pi' < 1/z$ . Again this means that either  $|S| < m$  or  $\pi = \pi' < 1/z$ .

Finally, consider the case that  $y \geq (1-\varepsilon)/z$ . Since  $\pi' \geq (1-\varepsilon)/z \geq 1/(2z)$ , we have  $|S| \geq m$ . This concludes that  $\pi = \pi'$  and  $y \leq \pi \leq y/(1-\varepsilon)$ , as required.  $\square$

### 3.3.3 Finding a maximal matching suffix that ends with $P$

We give two different methods for this task. The first method is based on a  $k$ -MISMATCH algorithm. Recall that if  $P$  matches  $T$  at some alignment  $q$  with probability at least  $1/z$ , then  $\mathcal{H}(T)[q-m+1, q]$  is a  $\log z$ -mismatch occurrence of  $P$  (see Observation 3.2). We use the  $k$ -MISMATCH algorithm with  $k = \log z$  for  $P$  and  $\mathcal{H}(T)$  to find all such alignments. When we identify a  $\log z$ -mismatch occurrence of  $P$  in  $\mathcal{H}(T)$ , we find a stream that corresponds to a maximal matching suffix that ends with  $P$ , if any (using the indexing trie). Suppose we have found a maximal matching suffix  $S$  that ends with  $P$ . We then use the algorithm of Lemma 3.15 to compute a  $(1-\varepsilon)$ -approximation of the product of the last  $m$  probabilities in the stream associated with  $S$ . By Lemma 3.16, we can output it as an answer.

By Observation 3.12, the number of maximal matching suffixes of  $T[1, q]$  never exceeds  $2z$ . Therefore, the total number of streams is  $\mathcal{O}(z)$ . Updating the streams, including removing the duplicates, takes  $\mathcal{O}(z \log z)$  space and time per position. For each of the streams we run the SLIDING WINDOW PRODUCT algorithm, which takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  space and time per position (Lemma 3.15). For  $k = \log z$ , the  $k$ -MISMATCH algorithm takes  $\mathcal{S}_{\log z}$  space and  $\mathcal{T}_{\log z}$  time per arriving symbol. Finally, to find the right stream we need just  $\mathcal{O}(\log z)$  additional time per position. In total, this is  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{T}_{\log z}$  time per position. Our algorithm can output an incorrect answer only when the  $k$ -MISMATCH algorithm errs, which happens with probability  $\mathcal{P}_{\log z}$ . We arrive at the second claim of Theorem 1.3:

**Proposition 3.17.** *If only the text is weighted, there is a  $(1-\varepsilon)$ -approximation streaming algorithm that solves WEIGHTED PATTERN MATCHING in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + \mathcal{T}_{\log z}$  time per arrival. At each arrival, the algorithm can err with probability  $z \cdot \mathcal{P}_{\log z}$ .*

The second method is based on a straightforward application of the streaming PATTERN MATCHING algorithm of Breslauer and Galil [6] for the pattern  $P$  and each of the maximal matching suffixes streams. The streaming PATTERN MATCHING algorithm of Breslauer and Galil [6] uses  $\mathcal{O}(\log m)$  space and takes  $\mathcal{O}(1)$  time per arrival. Once we have found a maximal matching suffix that ends with  $P$ , we apply the algorithm of Lemma 3.15 to compute a  $(1-\varepsilon)$ -approximation of the product of the last  $m$  probabilities in the stream associated with  $S$  and output it as an answer. In total, the second approach requires  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z + z \log m)$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  time per position. The error probability of the PATTERN MATCHING algorithm and, therefore, of our approach is  $1/\text{poly}(n)$  per position. We arrive at the second claim of Theorem 1.4:

**Proposition 3.18.** *If only the text is weighted, there is a  $(1-\varepsilon)$ -approximation streaming algorithm that solves WEIGHTED PATTERN MATCHING in  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log m))$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  time per arrival. At each arrival, the algorithm can err with probability  $1/\text{poly}(n)$ .*

## 3.4 Case 3 — both the text and the pattern are weighted

We conclude by showing two solutions for the case when both the pattern and the text are weighted. Our solutions combine the ideas of the two previous sections. Recall that at each alignment the algorithm must output “Yes” if there is a regular string that matches the text and the pattern with probability above  $1/z$ . The algorithm may also output “Yes” if there is a string that matches the pattern with probability at least  $1/z$  and the text with probability between  $(1-\varepsilon)/z$  and  $1/z$ . Otherwise it outputs “No”.

In the algorithm we maintain a set of at most  $2z$  streams for the maximal matching suffixes of the text  $T$ . For each of the streams we run the SLIDING WINDOW PRODUCT algorithm (Lemma 3.15). Let us recall that the set  $D$  contains at most  $z$  regular strings that match  $P$  with probability  $\geq 1/z$ . There is a regular string that matches the pattern and the text with probability  $\geq 1/z$  if and only if the following two conditions hold:

- (i) One of the maximal matching suffixes ends with a string in  $D$ ;

(ii) The product of the last  $m$  probabilities in the suffix's stream is  $\geq 1/z$ .

We check the condition (ii) approximately using the SLIDING WINDOW PRODUCT algorithm. For the condition (i), we again discuss two methods.

The first method uses a  $k$ -MISMATCH algorithm. If  $P$  matches  $T$  at some alignment  $q$ , then  $\mathcal{H}(T)[q - m + 1, q]$  must be a  $\log z$ -occurrence of one of the strings we generated for the pattern. We use the  $k$ -MISMATCH algorithm with  $k = \log z$  for each of the generated strings and  $\mathcal{H}(T)$  to find all such occurrences. At each alignment  $q$  we consider  $\log z$ -occurrences of the generated strings and build a trie of their indices, where the indices are defined as in Section 3.3. If a maximal matching suffix ends with one of these strings, its index must be in the trie. We can perform a search for each index in the trie in  $\mathcal{O}(z \log z)$  time in total. The time required to update the suffix streams is  $\mathcal{O}(z \log z)$  as there are  $\mathcal{O}(z)$  of them [1]. For each of the streams we run the SLIDING WINDOW PRODUCT algorithm, which takes  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z)$  space and time. For  $k = \log z$  the  $k$ -MISMATCH algorithm takes  $\mathcal{S}_{\log z}$  space and  $\mathcal{T}_{\log z}$  time per symbol. Building the trie of indices takes  $\mathcal{O}(z \log z)$  time. Searching for the indices in the trie takes  $\mathcal{O}(z \log z)$  time. In total, this is  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \cdot \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \cdot \mathcal{T}_{\log z}$  time. The algorithm can output an incorrect answer only when one of the  $k$ -MISMATCH algorithms errs, which happens with probability  $z/\text{poly}(m) = 1/\text{poly}(m)$ . We arrive at the following proposition. It is not included in Theorem 1.3 since the complexities resulting for the currently best values of  $\mathcal{S}_{\log z}$  and  $\mathcal{T}_{\log z}$  are worse than if the second method is used.

**Proposition 3.19.** *When both the pattern and the text are weighted, there is a  $(1 - \varepsilon)$ -approximation streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \mathcal{S}_{\log z}$  space and  $\mathcal{O}(z \log_{1/(1-\varepsilon)} z) + z \mathcal{T}_{\log z}$  time per arrival. At each arrival, the algorithm can err with probability  $z \cdot \mathcal{P}_{\log z}$ .*

The second method is to run the MULTIPLE PATTERN MATCHING algorithm for multiple streams [16]. This algorithm takes a dictionary and a set of streaming texts as an input. When a new symbol of a text arrives, the algorithm must determine whether the current text ends with a string from the dictionary. We run this algorithm for the dictionary  $D$  and each of the maximal matching suffixes streams. The algorithm uses  $\mathcal{O}(z \log m)$  shared memory,  $\mathcal{O}(\log m \log z)$  space per stream, and  $\mathcal{O}(\log m)$  time per each arriving character of a stream. In our case the total space is thus  $\mathcal{O}(z \log m \log z)$  and, since the maximal matching suffix streams are handled in a dynamic way, the algorithm takes  $\mathcal{O}(z \log m \log z)$  time per each arriving character of the text. The error probability is  $1/\text{poly}(n)$ . We therefore arrive at the third claim of Theorem 1.4:

**Proposition 3.20.** *When both the pattern and the text are weighted, there is a  $(1 - \varepsilon)$ -approximation streaming algorithm that solves the WEIGHTED PATTERN MATCHING problem in  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log z \log m))$  space and  $\mathcal{O}(z(\log_{1/(1-\varepsilon)} z + \log z \log m))$  time per arrival. At each arrival, the algorithm can err with probability  $1/\text{poly}(n)$ .*

### 3.5 Space lower bound — Proof of Proposition 1.5

The Yao minimax principle [27] implies that the expected space complexity of the optimal deterministic algorithm for an arbitrarily chosen input distribution  $\pi$  is a lower bound on the space complexity of the optimal Monte Carlo randomized algorithm. Therefore, it suffices to show that for some input distribution  $\pi$  any deterministic  $(1 - \varepsilon)$ -approximate streaming algorithm for the WEIGHTED PATTERN MATCHING problem requires  $\Omega(z)$  space on average (the lower bound for the exact algorithm follows).

**Case 1: Only pattern is weighted** Let us consider the following communication problem. Alice is given a weighted pattern  $P$  of length  $m$  drawn from some distribution and the prefix  $T[1, m]$  of the regular text  $T$  of length  $2m - 1$ . Bob is given the suffix  $T[m + 1, 2m - 1]$ . Alice and Bob are given a threshold  $1/z$  and Bob must distinguish all alignments of  $P$  and  $T$  where the match probability is at most  $(1 - \varepsilon)/z$  and at least  $1/z$ . The communication complexity of the problem is defined to be the expectation of the minimum number of bits that Alice must send to Bob so that he can achieve his mission.

The lower bound on the communication complexity of this problem is a lower bound on the space complexity of any streaming algorithm for the WEIGHTED PATTERN MATCHING by a standard reduction.



Namely, Alice can run the algorithm on the pattern and her half of the text and then send the memory of the algorithm to Bob. The space complexity of the algorithm will be equal to the communication complexity of the problem.

We select a string  $S$  of length  $m$  over an alphabet  $\{0, 1\}$  uniformly at random and build a weighted pattern over an alphabet  $\{A, B, C\}$ . For  $1 \leq i \leq m$ , if  $S[i] = 0$ , then  $P[i] = A$  with probability  $1 - \frac{1}{m}$ ,  $B$  with probability  $\frac{2}{3m}$ , and  $C$  with probability  $\frac{1}{3m}$ . If  $S[i] = 1$ , then  $P[i] = A$  with probability  $1 - \frac{1}{m}$ ,  $B$  with probability  $\frac{1}{3m}$ , and  $C$  with probability  $\frac{2}{3m}$ . The text is defined to be  $A^{m-1}BA^{m-1}$  and  $1/z = \frac{2}{3m}(1 - \frac{1}{m})^{m-1} = \Theta(\frac{1}{m})$ . We choose  $\varepsilon$  to be an arbitrary constant in  $[0, \frac{1}{2}]$ . If Bob can distinguish all alignments of  $P$  and  $T$  where the match probability is at most  $(1-\varepsilon)/z$  or at least  $1/z$ , he can also restore  $S$ . Indeed, if the match probability between  $P$  and  $T[i, i+m-1]$  is equal to  $\frac{2}{3m}(1 - \frac{1}{m})^{m-1} = 1/z$ , then  $S[m+1-i] = 0$ , and if the match probability between  $P$  and  $T[i, i+m-1]$  is equal to  $\frac{1}{3m}(1 - \frac{1}{m})^{m-1} \leq (1-\varepsilon)/z$ , then  $S[m+1-i] = 1$ . Therefore, from information theoretic ideas, the communication complexity of the problem must be  $\Omega(m) = \Omega(z)$ .

**Case 2: Only text is weighted** For the case when only the text is weighted, we use the same reduction but define  $P$  and  $T$  differently. Let  $S$  be a random string in  $\{0, 1\}^m$  as above.  $P$  is defined to be  $BA^{m-1}$  and  $T$  is defined over  $\{A, B, C\}$  as follows. For  $1 \leq i \leq m$ , if  $S[i] = 0$ , then  $T[i] = A$  with probability  $1 - \frac{1}{m}$ ,  $T[i] = B$  with probability  $\frac{2}{3m}$ , and  $T[i] = C$  with probability  $\frac{1}{3m}$ . If  $S[i] = 1$ , then  $T[i] = A$  with probability  $1 - \frac{1}{m}$ ,  $T[i] = B$  with probability  $\frac{1}{3m}$ , and  $T[i] = C$  with probability  $\frac{2}{3m}$ . For  $m+1 \leq i \leq 2m-1$ ,  $T[i] = A$  with probability  $1 - \frac{1}{m}$  and  $T[i] = B$  with probability  $\frac{1}{m}$ . Again, if Bob can distinguish all alignments of  $P$  and  $T$  where the match probability is at most  $(1-\varepsilon)/z$  or at least  $1/z$ , he can also restore  $S$ . Therefore, any streaming algorithm for this variant must use  $\Omega(z)$  space as well.

**Case 3: Both the text and the pattern are weighted** As a corollary we immediately obtain that any exact or  $(1-\varepsilon)$ -approximate streaming algorithm for the weighted pattern, weighted text case must use  $\Omega(z)$  space.

**Remark 3.21.** *We gave the proof for  $z = \Theta(m)$ . However, we can modify the examples so that it holds for  $z \leq m$ : it suffices to prepend the pattern and the text with  $A^k$ , for sufficiently large  $k$ , which for a weighted sequence means adding  $k$  positions with probability of  $A$  equal to 1.*

## 4 Conclusions

In this work we present the first efficient solutions to the problems of  $k$ -MISMATCH WITH ERROR CORRECTING and WEIGHTED PATTERN MATCHING in the streaming model. In parallel to our work, a yet more efficient algorithm for  $k$ -MISMATCH WITH ERROR CORRECTING was developed [12] that near-matches time and space lower bounds. We provide lower bounds for streaming solutions to WEIGHTED PATTERN MATCHING that show that the space complexity of our solutions is nearly optimal. It is an interesting open problem if the two variants of WEIGHTED PATTERN MATCHING where the text is weighted can be solved exactly using  $\mathcal{O}(z^{1-\varepsilon}) \cdot \log^{O(1)} m$  time per arrival, for any  $\varepsilon > 0$ .

**Acknowledgements** This work was supported by the ‘‘Algorithms for text processing with errors and uncertainties’’ project carried out within the HOMING programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund.

## References

- [1] Amihhood Amir, Eran Chencinski, Costas S. Iliopoulos, Tsvi Kopelowitz, and Hui Zhang. Property matching and weighted matching. *Theor. Comput. Sci.*, 395(2-3):298–310, 2008. doi:10.1016/j.tcs.2008.01.006.
- [2] Amihhood Amir, Costas Iliopoulos, Oren Kapah, and Ely Porat. Approximate matching in weighted sequences. In *CPM’06: Proc. of the 17th Annual Symposium on Combinatorial Pattern Matching*, volume 4009 of *LNCS*, pages 365–376, 2006. doi:10.1007/11780441\_33.

- [3] Carl Barton, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Efficient index for weighted sequences. In *CPM'16: Proc. of the 27th Annual Symposium on Combinatorial Pattern Matching*, LIPIcs, pages 4:1–4:13, 2016. doi:10.4230/LIPIcs.CPM.2016.4.
- [4] Carl Barton, Chang Liu, and Solon P. Pissis. Linear-time computation of prefix table for weighted strings & applications. *Theor. Comput. Sci.*, 656:160–172, 2016. doi:10.1016/j.tcs.2016.04.029.
- [5] Sudip Biswas, Manish Patil, Sharma V. Thankachan, and Rahul Shah. Probabilistic threshold indexing for uncertain strings. In *EDBT'16: Proc. of the 19th International Conference on Extending Database Technology*, pages 401–412, 2016. doi:10.5441/002/edbt.2016.37.
- [6] Dany Breslauer and Zvi Galil. Real-time streaming string-matching. *ACM Trans. Algorithms*, 10(4):22:1–22:12, 2014. doi:10.1145/2635814.
- [7] Manolis Christodoulakis, Costas S. Iliopoulos, Laurent Mouchard, and Kostas Tsichlas. Pattern matching on weighted sequences. In *CompBioNets'04: Proc. of the 1st International Conference on Algorithms and Computational Methods for Biochemical and Evolutionary Networks*, KCL publications, 2004.
- [8] Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. *Inf. Comput.*, 209(4):731–736, 2011. doi:10.1016/j.ic.2010.12.007.
- [9] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The  $k$ -mismatch problem revisited. In *SODA'16: Proc. of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2039–2052, 2016. doi:10.1137/1.9781611974331.ch142.
- [10] Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana A. Starikovskaya. Dictionary matching in a stream. In *ESA'15: Proc. of the 23rd Annual European Symposium on Algorithms*, volume 9294 of *LNCS*, pages 361–372, 2015. doi:10.1007/978-3-662-48350-3\_31.
- [11] Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. *Theor. Comput. Sci.*, 483:58–74, 2013. doi:10.1016/j.tcs.2012.06.012.
- [12] Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming  $k$ -mismatch problem. In *SODA'19: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1106–1125, 2019. doi:10.1137/1.9781611975482.68.
- [13] Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In *ICALP'16: Proc. of the 43rd International Colloquium on Automata, Languages, and Programming*, LIPIcs, pages 20:1–20:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.
- [14] Funda Ergün, Hossein Jowhari, and Mert Sağlam. Periodicity in streams. In *APPROX-RANDOM'10: Proc. of the 14th International Workshop on Approximation, Randomization, and Combinatorial Optimization*, volume 6302 of *LNCS*, pages 545–559, 2010. doi:10.1007/978-3-642-15369-3\_41.
- [15] N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.
- [16] Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *ICALP'18: Proc. of the 45th International Colloquium on Automata, Languages, and Programming*, pages 65:1–65:16, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.
- [17] Shay Golan and Ely Porat. Real-time streaming multi-pattern search for constant alphabet. In *ESA'17: 25th Annual European Symposium on Algorithms*, pages 41:1–41:15, 2017. doi:10.4230/LIPIcs.ESA.2017.41.
- [18] Costas S. Iliopoulos, Christos Makris, Yannis Panagis, Katerina Perdikuri, Evangelos Theodoridis, and Athanasios K. Tsakalidis. The weighted suffix tree: An efficient data structure for handling molecular weighted sequences and its applications. *Fundam. Inform.*, 71(2-3):259–277, 2006. URL: <http://content.iospress.com/articles/fundamenta-informaticae/fi71-2-3-07>.

- [19] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. of Research and Development*, 31(2):249–260, 1987.
- [20] Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Pattern matching and consensus problems on weighted sequences and profiles. *Theory Comput. Syst.*, 63(3):506–542, 2019. doi:10.1007/s00224-018-9881-2.
- [21] Cinzia Pizzi and Esko Ukkonen. Fast profile matching algorithms - A survey. *Theor. Comput. Sci.*, 395(2-3):137–157, 2008. doi:10.1016/j.tcs.2008.01.015.
- [22] Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS'09: Proc. of the 50th Annual Symposium on Foundations of Computer Science*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.
- [23] Ely Porat and Ohad Lipsky. Improved sketching of Hamming distance with error correcting. In *CPM'07: Proc. of the 18th Annual Conference on Combinatorial Pattern Matching*, volume 4580 of *LNCS*, pages 173–182, 2007. doi:10.1007/978-3-540-73437-6\_19.
- [24] Jakub Radoszewski and Tatiana A. Starikovskaya. Streaming k-mismatch with error correcting and applications. In *DCC'17: Proc of the 2017 Data Compression Conference*, pages 290–299, 2017. doi:10.1109/DCC.2017.14.
- [25] Albin Sandelin, Wynand Alkema, Pär Engström, Wyeth W. Wasserman, and Boris Lenhard. JASPAR: An open-access database for eukaryotic transcription factor binding profiles. *Nucl. Acids Res.*, 32(1):D91–D94, 2004. doi:10.1093/nar/gkh012.
- [26] Xuhua Xia. Position weight matrix, Gibbs sampler, and the associated significance tests in motif characterization and prediction. *Scientifica*, 2012. Article ID 917540. doi:10.6064/2012/917540.
- [27] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity. In *FOCS'18: Proc. of the 18th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977. doi:10.1109/SFCS.1977.24.