Communication and Streaming Complexity of Approximate Pattern Matching

Tatiana Starikovskaya

Communication and Streaming Complexity of Approximate Pattern Matching

Tatiana Starikovskaya

Université Paris-Diderot – Paris 7, Paris, France
tat.starikovskaya@gmail.com

Abstract

We consider the approximate pattern matching problem. Given a text T of length $2n$ and a pattern P of length n, the task is to decide for each prefix $T[1, j]$ of T if it ends with a string that is at the edit distance at most k from P. If this is the case, we must output the edit distance and the corresponding edit operations. We first look at the communication complexity of the problem. We show the following:

- If Alice and Bob both share the pattern and Alice holds the first half of the text and Bob the second half, then the deterministic one-way communication complexity of the problem is $\Theta(k \log n)$.
- If Alice holds the first half of the text, Bob the second half of the text, and Charlie the pattern, then there is a deterministic one-way communication protocol that uses $\mathcal{O}(k\sqrt{n} \log n)$ bits.

We then develop the first sublinear-space streaming algorithm for the problem.

- There exists a streaming algorithm that solves the problem in $\mathcal{O}(k^8\sqrt{n} \log^6 n)$ space. The worst-case time complexity of the algorithm $\mathcal{O}((k^2\sqrt{n}+k^{13})\cdot\log^4 n)$ per arrival. The algorithm is randomised with error probability at most $1/\text{poly}(n)$.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases approximate pattern matching, edit distance, randomised algorithms, streaming algorithms, communication complexity

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.13

1 Introduction

In this work we study the famous approximate pattern matching problem. Recall that the edit distance between two strings S_1, S_2 is the minimum number of insertions, deletions, and substitutions required to transform S_1 to S_2. Assume we are given a pattern P and a text T. We say that a substring S of T is a k-mismatch occurrence of P if the edit distance between S and P is at most k. In the approximate pattern matching problem we must find all prefixes $T[1, j]$ of T that end with a k-mismatch occurrence of P. The problem has numerous applications in bioinformatics, signal processing, text retrieval, and has received a lot of attention in the literature.

1.1 Our results

We first study the communication complexity of the problem, namely, we consider the following setting. Let T be a text of length $2n$ and P be a pattern of length n. Let Alice hold the information about the first half of the text, and let Bob hold the information about the second half of the text. Alice sends Bob a message, and Bob's task is to find all prefixes $T[1, j]$ that end with a k-mismatch occurrence of P and the edit operations that transform the occurrence into P using only Alice's message and his half of the text. The minimal size of

Alice's message that allows Bob to complete the task is called the communication complexity of the problem.

It is not hard to see that if both Bob and Alice have access to the pattern, the communication complexity is $\Theta(k \log n)$. Indeed, from the information theoretic lower bound it follows that Alice has to send at least $k \log n$ bits. On the other hand we can consider the following (deterministic) protocol. Alice first finds the smallest i such that the edit distance between $T[i, n]$ and some prefix $P[1, j]$ of the pattern is at most k, and then sends the edit operations and j to Bob. Bob uses the message from Alice to restore $T[i, n]$. He then knows both $T[i, 2n]$ and P and therefore can compute all outputs. (Note that the edit distance between P and any substring of T that starts in $[1, i]$ and ends in $[n + 1, 2n]$ is at least k, and therefore Bob does not need any information about $T[1, i]$). However, the situation is different when only the third party, Charlie, knows the pattern, as in this case Alice can no longer use the pattern to encode her half of the text. We show the following theorem.

▶ **Theorem 1.** *When both P and T are binary, the one-way deterministic communication complexity of the approximate pattern matching problem for three parties is $\mathcal{O}(k\sqrt{n} \log n)$.*

The main idea of the proof is that if k-mismatch occurrences of the pattern in the text are rare, Alice can send them all to Bob. If on the other hand there are many k-mismatch occurrences of the pattern, two of them will be located close to each other and therefore the underlying text will be weakly periodic, which will allow to encode it in small space.

Our motivation to study the communication complexity of the problem is twofold. First, it can be viewed as a generalisation of the document exchange problem, where we have two parties Alice and Bob, Alice holds one string and Bob holds the other string, and Bob's task is to decide the edit distance between their strings using the message Alice sends to him and his half of the text. If the distance is at most k, Bob must output the edit operations that transform his string into Alice's string. Otherwise, he may simply say that the distance is too large. In the paper we will refer to Alice's message as document exchange sketch. The problem has been studied both in deterministic and randomised settings [1, 4, 3, 8, 11, 15]. The protocol shown by Orlitsky in 1991 [15] has optimal complexity $\Theta(k \log n)$ and is deterministic. However, Bob needs $n^{\mathcal{O}(k)}$ time to compute the distance. Recently, Belazzougui showed a new deterministic protocol [1]. It has complexity $\mathcal{O}(k^2 + k \log^2 n)$ and much lower computation time of $n \cdot \text{poly}(\log n)$. The best randomised protocol is due to [2] and has $\mathcal{O}(k \cdot (\log^2 k + \log n))$ complexity and $n \cdot \text{poly}(\log n)$ computation time.

The second reason to study the communication complexity of the problem is its relation to streaming algorithms. Let us first remind the setting. Consider a pattern P of length n and a text T of length $2n$ arriving as a stream, one symbol at a time. When a new symbol arrives we must decide if the current text ends with a k-mismatch occurrence of P and if so output the edit operations that transform the occurrence into P. We assume the standard RAM model of computation. The time complexity is defined in the usual way, and the space complexity is defined as all the space used by the algorithm. In particular, if we store a copy of the pattern or of the text we must account for it. It is well-known that a communication complexity lower bound implies a similar space lower bound for a streaming algorithm. However, upper bounds provide some insight as well. Imagine that the algorithm processes the stream in non-overlapping blocks, then it needs an efficient way to encode the edit distances in each of the blocks, and one possible approach is to use the message that Alice sends to Bob in the communication complexity protocol. We will use this idea to show the first sublinear-space streaming algorithm for the problem.

▶ **Theorem 2.** *Assume that both P and T are binary and that $k < n^{1/c}$ for a sufficiently large constant $c > 0$. There is a streaming algorithm that solves the approximate pattern matching problem in $\mathcal{O}(k^8\sqrt{n}\log^6 n)$ space and $\mathcal{O}((k^2\sqrt{n}+k^{13})\cdot\log^4 n)$ worst-case time per symbol. The algorithm is correct with probability $1-1/\mathrm{poly}(n)$.*

The main advance has become possible due to the result of Belazzougui and Zhang [2], who showed a sketch that can be used to compute the exact value of the edit distance between two strings if it is at most k. Our algorithm maintains such sketches for $\mathcal{O}(\sqrt{n})$ suffixes of the text. To compute the edit distance between the pattern and the text the algorithm divides the pattern into two parts, a short prefix and a suffix aligned with one of the sketched suffixes of the text. The edit distance between the short prefix and the text is computed beforehand using dynamic programming and stored very compactly using the communication complexity approach. The edit distance between the suffix and the text is computed with the help of the sketches. We note that the requirement on the text length is not restrictive. Indeed, if the text's length is larger than $2n$, then one can split it into blocks of length $2(n+k)$ which overlap by $n+k$ symbols (the last block can be shorter) and run the algorithm of Theorem 2 independently for each of the blocks. For each k-mismatch occurrence of P there is a block containing it and therefore the algorithm is correct. The complexity of the algorithm and the error probability do not change.

As we have already mentioned the problem has been extensively studied in the literature. For a survey of previous solutions see [14]. The solutions can be roughly classified into four main types: dynamic programming algorithms, automata-based algorithms, filtering algorithms, and bit-parallelism. To the best of author's knowledge, all previously known solutions require at least $\Omega(n/\log n)$ space, and thus our result exhibits a remarkable improvement in space complexity. On the other hand, the running time of our algorithm is rather large. This is because the nature of the sketches is very complex and we have to maintain them independently. We give further details in Sections 3 and 4.

1.2 Related work

Lower bounds. In this work we focus on computing small edit distances between a pattern and a stream. If however we were interested in computing all edit distances, we would have to spend at least $n^{1-\varepsilon}$ amortised time per output for any constant $\varepsilon > 0$ unless the strong Exponential Time Hypothesis is false. (The original bound was given for computing the edit distance between two strings, and our problem is harder.) The best unconditional time lower bound was shown by Clifford et al. [6] who considered the problem in the cell-probe model, where the time complexity of algorithm is measured as the number of cells that must be accessed to compute the output. This model is particularly strong and any lower bounds that hold in it hold in the RAM model as well. Clifford et al. showed that the expected amortised time of any randomised algorithm that solves the edit distance problem is $\Omega(\sqrt{\log n}/(\log\log n)^{3/2})$ per output.

Approximate pattern matching in a stream. Another formalisation of approximate pattern matching is the k-mismatch problem, where one must find all substrings of the text such that the Hamming distance between them and the pattern is at most k. The first streaming algorithm for this problem was given in [16]. It used $\mathcal{O}(k^3\log^7 n/\log\log n)$ space and $\mathcal{O}(k^2\log^5 n/\log\log n)$ time per arriving symbol. In [5] this result was improved in terms of the dependency on k to $\mathcal{O}(k^2\log^{11} n/\log\log n)$ space and $\mathcal{O}(\sqrt{k}\log k+\log^5 n)$ time per arriving symbol. Finally, in [7] the authors studied communication and streaming complexities of computing approximate values of all Hamming distances between the pattern and the text.

2 Communication complexity

In this section we show Theorem 1. Recall that Alice holds the first half of the text, Bob the second half of the text, and only Charlie holds the pattern. Bob must find all prefixes $T[1, j]$ of T that end with a k-mismatch occurrence of P and output the edit operations that convert the occurrence into P.

2.1 Periodicity under edit distance

We start by introducing a notion of approximate period for the edit distance. The idea is that two close k-mismatch occurrences of the pattern imply weak periodicity of the text. We will use this property of the text to encode it in small space.

▶ **Definition 3.** The α-period of a string S is a minimal integer $\ell > \alpha$ such that the edit distance between some prefix of S and $S[\ell + 1, n]$ is at most α.

▶ **Example 4.** The 1-period of a string $S = bbaabb$ is 3. This is because $S[3, 6] = aabb$ cannot be transformed into a prefix of S using just one edit operation, while the edit distance between $S[4, 6] = abb$ and $S[1, 2]$ is exactly one.

The condition $\ell > \alpha$ is essential as any suffix $S[\ell + 1, n]$ can be transformed into S by ℓ insertions. We now show that the α-period can be used to encode the pattern in an efficient way similar to the way the period of a string can be used to encode it.

▶ **Lemma 5.** *If the $4k$-period of a string S of length n is $\rho > 4k$, then S can be encoded in $\mathcal{O}(\rho + k \log n)$ bits.*

Proof. The encoding will occupy $\mathcal{O}(\rho + k \log n)$ bits and contain the prefix and the suffix of S of length ρ (both taking $\mathcal{O}(\rho)$ bits to store), and the at most $4k$ edit operations that transform a prefix S' of S into $S[\rho + 1, n]$. The information about the edit operations will include the type of the operation (insertion, deletion, substitution), the position, and the symbol itself.

We now show that the encoding is lossless. Consider the first ρ symbols of S'. Let $i_1 \leq 4k$ be the number of these symbols that must be deleted. It follows that the remaining $\rho - i_1$ symbols of S' must be aligned against the symbols of $S[\rho + 1, n]$. Therefore, using the encoding, we can restore (at least) the first $\rho - i_1$ symbols of $S[\rho + 1, n]$ and consequently $S'[1, 2\rho - i_1]$. (Recall that insertions and replacements are stored in the encoding explicitly.) We then consider $S'[\rho, 2\rho - i_1]$. Let i_2, $i_1 + i_2 \leq 4k$, be the number of symbols in $S'[\rho, 2\rho - i_1]$ that must be deleted. We can then use the remaining symbols to restore the first $2\rho - i_1 - i_2$ symbols of $S[\rho + 1, n]$ and consequently $S'[1, 3\rho - i_1 - i_2]$. We continue in a similar way until we reach the end of S'. At this point, we will restore all symbols of S except for maybe the last ρ symbols which we already know from the encoding. ◀

2.2 Communication complexity protocol

We first explain what Charlie sends to Alice, and what Alice sends to Bob. Let $B = k\sqrt{n} \log n$ and $n_B = \lceil n/B \rceil$. Charlie sends to Alice document exchange sketches for each prefix $P[1, (n_B - i) \cdot B]$ and for each suffix $P[(n_B - i) \cdot B + 1, n]$. We use deterministic document exchange sketches of size $\mathcal{O}(k^2 + k \log^2 n)$ bits [1]. (We note that using $\mathcal{O}(k \log n)$-space sketches [15] would not improve the complexity but would drastically increase the computation time for Alice and Bob. For this reason, even though time is not the focus of this work, we

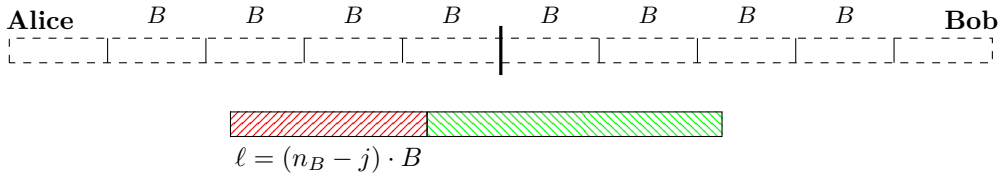

Figure 1 Let i be the first block containing two k-mismatch occurrences of $P[1, (n_B - i) \cdot B]$ that start at least $2k$ positions apart. To compute the edit distances in a block $j < i$ Bob divides the pattern into two parts, a prefix $P[1, \ell]$ and the suffix $P[\ell + 1, n]$, and computes the distance for each of the two parts separately.

prefer the sketches [1].) Alice starts by dividing her half of the text into non-overlapping blocks of length B except for the last one which may be shorter, that is in total there are n_B blocks.

▶ **Definition 6.** A position p of a block i is k-good if it is the left endpoint of a k-mismatch occurrence of $P[1, (n_B - i) \cdot B]$.

Alice considers each block i in turn and finds all k-good positions in the block using the pattern sketches. Suppose first that all k-good positions in the block are at distance $< 4k$. In this case all k-mismatch occurrences of $P[1, (n_B - i) \cdot B]$ that start in these positions end in an interval of length at most $6k$. For each position in this interval Alice finds the substring that ends in it and has the smallest edit distance from $P[1, (n_B - i) \cdot B]$ (using the pattern sketches again) and sends the distance and the corresponding edit operations to Bob. In total this information occupies $\mathcal{O}(k^2 \log n)$ bits per block. Suppose now that block i contains two k-good positions p_1, p_2, where $p_2 - p_1 > 4k$, and let i be the first such block. Let $\ell = (n_B - i) \cdot B$ and let ED be the edit distance between two strings.

▶ **Lemma 7.** *The $4k$-period of $T[p_1, p_2 + \ell - 1]$ is at most B.*

Proof. By the definition both p_1 and p_2 are starting positions of k-mismatch occurrences of $P[1, \ell]$. Therefore, $ED(T[p_1, p_1 + \ell - 1], P[1, \ell]) \leq 2k$ and $ED(T[p_2, p_2 + \ell - 1], P[1, \ell]) \leq 2k$. From the triangle inequality it follows that $ED(T[p_1, p_1 + \ell], T[p_2, p_2 + \ell - 1]) \leq 4k$ and from the definition of approximate periods it follows that the $4k$-period of $T[p_1, p_2 + \ell - 1]$ is at most B. ◀

By Lemma 5 the substring $T[p_1, p_2 + \ell - 1]$ and therefore $T[p_1, n - B]$ can be encoded in $\mathcal{O}(B + k \log n)$ bits. Alice sends the encoding to Bob (note that she only does it for the first block containing distant k-good positions). Finally, she sends Bob the last $(B + k)$ symbols of her half of the text and also forwards the sketches received from Charlie. The total size of Alice's message is $\mathcal{O}((n/B) \cdot k^2 \log^2 n + B) = \mathcal{O}(k\sqrt{n} \log n)$ bits.

We now explain how Bob computes the distances. Suppose that he wants to compute the edit distance between the pattern a substring starting to the left of position p_1. Using the encoding of $T[p_1, n - B]$, the last B symbols of Alice's half of the text, and his half of the text he can restore all symbols of $T[p_1, 2n]$. He can then use the pattern sketch to compute the edit distance and operations. Consider now the case when the substring starts in a block $j < i$ (see Fig. 1). Let S be the substring for which Bob wants to compute the edit distance and $\ell = (n_B - j) \cdot B$. Bob starts by dividing the pattern into two parts, a prefix $P[1, \ell]$ and the suffix $P[\ell + 1, n]$. The following observation is a corollary of the definition of the edit distance.

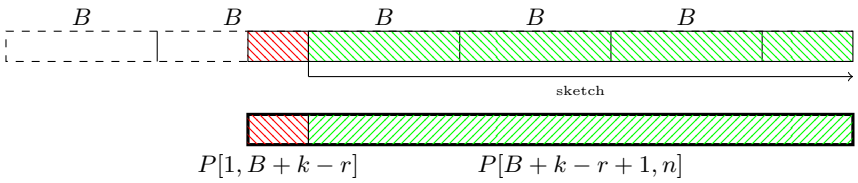

Figure 2 The algorithm processes the text in blocks of size B. To decide whether the current stream ends with a k-mismatch occurrence of P, the algorithm divides the pattern into two parts, a prefix of length at most $B + k$ and the remaining suffix and computes the edit distance for each of the parts separately.

▶ **Observation 8.** *Let* $\Delta = \min_{\ell' \in [\ell-k, \ell+k]}\{ED(P[1,\ell], S[1,\ell']) + ED(P[\ell+1,n], S[\ell'+1,n])\}$. *If* $\Delta > k$, *then the edit distance between* S *and* P *is larger than* k, *and otherwise it is equal to* Δ.

Since $j < i$, Bob knows all positions ℓ' of S for which there exists a k-mismatch occurrence of $P[1,\ell]$ ending at this position (and also the edit operations that convert the occurrence into $P[1,\ell]$). On the other hand, since Bob knows the last $B+k$ symbols of Alice's half of the text, he knows $S[\ell'+1,n]$ and can use the sketch of $P[\ell+1,n]$ to compute the edit distance and the edit operations between the two. He can therefore decide if S is a k-mismatch occurrence of P and the edit operations that transform S into P.

3 Streaming

We now show a streaming algorithm for approximate pattern matching. As soon as a new symbol arrives we must decide if the current stream ends with a k-mismatch occurrence of P and output the edit operations between P and the occurrence. The algorithm processes the text by blocks of size $B = \sqrt{n}$ (see Fig. 2). Suppose that the text ends with a k-mismatch occurrence of the pattern P. This occurrence can be divided into two parts, a prefix of length at most B, and a suffix that starts at a block border. From Observation 8 it follows that there exists a position $i \in [1, B+k]$ such that the prefix of the occurrence must be aligned with $P[1,i]$, and the suffix of the occurrence must be aligned with $P[i+1,n]$. The algorithm will therefore need to be able to compute the edit distances between each block and prefixes $P[1,i]$, and the edit distances between suffixes of the text starting at block borders and suffixes $P[i+1,n]$.

3.1 Prefixes

Consider a block of the text T. For each i such that the block ends with a k-mismatch occurrence of $P[1,i]$ we define S_i to be the suffix of the block with the smallest edit distance from $P[1,i]$. Below we will show a hybrid dynamic programming algorithm that computes all suffixes S_i, the corresponding edit distances and edit operations in $\mathcal{O}((B+k)\cdot k)$ space and in $\mathcal{O}(k)$ time per symbol of the block. But first, let us explain how we apply it. Note that the suffixes S_i, the distances and the operations will be used only n/B blocks later. A naive approach would be to compute all this information and to store it explicitly until that time. However, the total space requirement of this approach is too large. Instead, we develop a different approach which runs the algorithm twice. Upon having received a new text block, we run the algorithm for the first time and compute suffixes S_i for all $i \in [1, B+k]$. Let

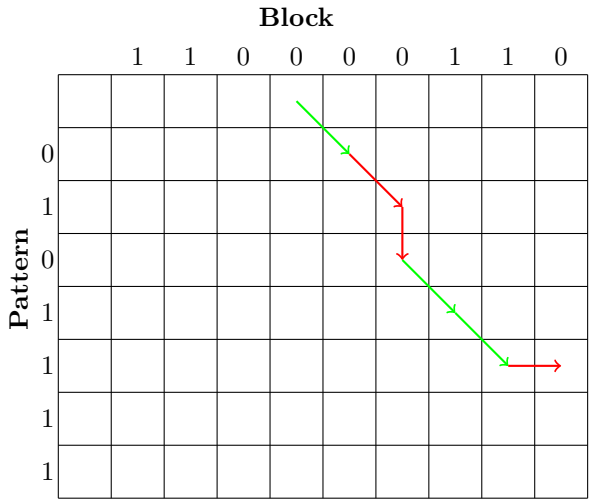

Figure 3 The graph shows a 3-path that encodes the edit operations between $P[1,5] = 01011$ and a suffix 00110 of the block. The three red arrows show the edit operations: a replacement, an insertion, and a deletion.

$S^\star = S_j$ be the longest of the retrieved suffixes. We encode the block as a tuple consisting of the position j, and the at most k edit operations that transform $P[1, j]$ into $S^\star$ (see also Introduction). After having read $n/B-2$ more blocks we use the encoding and $P[1, B+k]$ to restore $S^\star$ and then run the algorithm on $S^\star$ to compute the suffixes S_i and the corresponding edit operations.

We now describe our algorithm. The algorithm uses the same approach as the hybrid dynamic programming algorithms for the approximate pattern matching problem [12, 13] (see also [10, Chapter 12.2.4]). We assume that $P[1, B+k]$ is stored explicitly. The algorithm receives as an input a text block of length $\leq B$. The algorithm starts by preprocessing the $P[1, B+k]$ and the block for longest common extension queries. For a pair of positions (p_1, p_2), a longest common extension query finds the longest substring starting at position p_1 of the block that matches a substring starting at position p_2 of $P[1, B+k]$. The preprocessing phase takes $\mathcal{O}(B+k)$ time and space [9]. The algorithm then considers a table of size $(B+k+1) \times (B+1)$ and builds a set of paths from the first row to the last column of the table. Each such path will correspond to a suffix of the block that is a k-mismatch occurrence of $P[1, i]$ and encode the edit operations that transform the suffix into $P[1, i]$.

The algorithm runs in k rounds. In round m, $1 \leq m \leq k$, it processes each of the diagonals of the table in turn and finds a path that corresponds to at most m edit operations (m-path) and ends in the lowest cell in the current diagonal. Each of the paths starts in one of the cells in the first row of the table. From a cell (p_1, p_2) a path can go either to a cell (p_1+1, p_2), or to (p_1, p_2+1), or to (p_1+1, p_2+1). Let a be the (p_1+1)-th symbol of the block and b be the (p_2+1)-th symbol of the pattern. A move to (p_1+1, p_2) corresponds to deletion of a, a move to (p_1, p_2+1) to insertion of b, and a move to (p_1+1, p_2+1) to a replacement of a by b if $a \neq b$. If symbols a, b are not edited, the path makes a diagonal step as well. Suppose that in round m, $m \leq k$, a path reaches a cell (B, i) of the last column of the table for the first time. From construction it follows that this path corresponds to the suffix S_i.

It remains to explain how the algorithm finds the m-paths. Consider a diagonal i. To find the m-path that ends in the lowest cell in the diagonal, the algorithm tries to extend the $(m-1)$-paths for diagonals $i-1$, i, and $i+1$. Consider first the $(m-1)$-path for

diagonal i. Suppose that it ends in a cell $(j, j+i)$. The algorithm makes a step from $(j, j+i)$ to $(j+1, j+i+1)$ that corresponds to a replacement of a symbol and then tries to extend the path further down along the diagonal until it meets the next pair of mismatching symbols. Note that this step can be performed in $\mathcal{O}(1)$ time using a longest common extension query. The $(m-1)$-paths in diagonals $i-1$ and $i+1$ are extended in a similar fashion, except that from the end of the $(m-1)$-path in diagonal $i+1$ the algorithm makes a horizontal step (corresponds to a deletion of a symbol of the block) and from the end of the $(m-1)$-path in diagonal $i+1$ the algorithms makes a vertical step (corresponds to an insertion of a symbol). It is not hard to see that in this way the algorithm finds the end of the m-path for a fixed diagonal in $\mathcal{O}(1)$ time, meaning that overall the algorithm uses $\mathcal{O}((B+k)\cdot k)$ time and $\mathcal{O}((B+k)\cdot k)$ space per block.

▶ Remark. Note that the running time of the algorithm can be de-amortised to spend $\mathcal{O}(k)$ time per arrival in the worst case: When we apply the algorithm to a block i for the first time, we de-amortise its running timer over block $i+1$ by running $\Omega(k)$ steps of the algorithm each time a new block symbol arrives, and when we run the algorithm for the second time we de-amortise its running time over block $i+n/B-2$.

3.2 Suffixes

To compute the distances from suffixes of the pattern to the text the algorithm uses sketches by Belazzougui and Zhang [2, Theorem 13].

▶ **Theorem 9** ([2]). *Assume $k < n^{1/c}$ for some sufficiently large constant $c > 0$. There is a sketch of size $\mathcal{O}(k^8 \log^5 n)$ that can be used to compute the edit distance between two binary strings of length at most n in $\mathcal{O}(k^{12} \log^3 n)$ time correctly with probability 0.9. Given a string arriving as a stream its sketch can be constructed in $\mathcal{O}(k^2 \log^4 n)$ amortised time per symbol.*

The space and time bounds are not given in [2, Theorem 13] but can be derived from its proof. We will show the following corollary.

▶ **Corollary 10.** *Assume $k < n^{1/c}$ for some sufficiently large constant $c > 0$. There is a sketch of size $\mathcal{O}(k^8 \log^6 n)$ that can be used to compute the edit distance between two binary strings of length at most n in $\mathcal{O}(k^{12} \log^4 n)$ time correctly with probability $1 - \mathrm{poly}(n)$. Given a string arriving as a stream its sketch can be constructed in $\mathcal{O}(k^2 \log^4 n)$ worst-case time per symbol.*

We boost the probability of Theorem 13 [2] from 0.9 to $1 - \mathrm{poly}(n)$ in a standard way, by repeating the computation independently $\mathcal{O}(\log n)$ times and taking the smallest edit distance as an answer, which yields the extra $\log n$ factors in the complexities.

For completeness and to explain how to de-amortise the time bound, we give the definition of the sketches. The sketches are constructed using a random walk embedding from edit to Hamming distance [4]. The embedding maps strings of length n onto strings of length $3n$. Consider a string S and set a pointer to $S[1]$. At each step, the embedding copies the symbol at which the pointer is currently at to the resulting string $E(S)$ and either moves the pointer to the right or stays in place. After having reached the end of S it stops, and if the length of $E(S)$ is $\ell < 3n$, it appends $3n - \ell$ zeros to it. The moves of the pointer are defined by a random string $R \in \{0,1\}^{6n}$. If i is the current position of the pointer in S, and j is the length of $E(S)$, then the pointer moves to the right if $R[S[i] + 2j] = 1$ and otherwise it stays in place.

▶ **Theorem 11** ([4]). *For every constant $c > 0$ and every pair of binary strings S_1, S_2 of length at most n, the Hamming distance between $E(S_1), E(S_2)$ is at most $c \cdot \big(ED(S_1, S_2)\big)^2$ with probability at least $1 - 12/\sqrt{c}$.*

The intuition behind the proof is that the difference between the pointers' positions as they move along two strings S_1, S_2 behaves as a one-dimensional random walk. In more details, since R is a random binary string, at each time moment when the difference is not zero and there is a mismatch between $E(S_1)$ and $E(S_2)$ the difference does not change with probability $1/2$, increases by one with probability $1/4$, and decreases by one with probability $1/4$.

The mismatched symbols of $E(S_1)$ and $E(S_2)$ and their respective positions in S_1 and S_2 can be used to construct a set of edit operations that transform S_1 to S_2. The set might be not optimal, but it gives some evidence of which positions in S_1 and S_2 must be edited. Belazzougui and Zhang first developed sketches of the embeddings $E(S_1), E(S_2)$ that allow to retrieve both the mismatched symbols and their positions in S_1 and S_2. Their sketches are based on the Hamming distance sketches of Porat and Lipsky [17] and can be constructed in $\mathcal{O}(\log^2 n)$ worst-case time per symbol of an embedding. They further suggested to consider $\mathcal{O}(k^2 \log^2 n)$ independent random walk embeddings and showed that they give enough information to derive the optimal set of edit operations.

To de-amortise the time bound of Theorem 13 [2] we notice that in the random walk embedding a pointer advances by at least one position of the initial string each $3 \log n$ steps with probability at least $1 - 1/n^3$. Therefore if the sketch construction algorithm gets stuck at some position for more than $3 \log n$ steps, we can simply abandon it. This incomplete sketch might result in erroneous outputs, but the probability of this event is small.

3.3 Algorithm

We are now ready to give a full description of the algorithm. We assume that the algorithm first receives the pattern and preprocesses it in a streaming fashion. Namely, it remembers the first $B + k$ symbols of the pattern and also computes sketches of each suffix $P[i, n]$, $i \in [1, B + k]$. The sketches occupy $\mathcal{O}((B + k) \cdot k^8 \log^6 n)$ space in total.

After a new block of the text has arrived, the algorithm computes its encoding defined in Section 3.1. In total all block encodings occupy $\mathcal{O}((n/B) \cdot k \log n)$ space. Also, while reading block i, the algorithm decodes block $i + 2 - n/B$ and runs the algorithm of Section 3.1 to compute the edit distances for the prefixes of P. Recall that this step can be de-amortised to take $\mathcal{O}(k)$ worst-case time per arrival. Finally, the algorithm considers each of the suffixes of the current text that starts at a block border as a separate stream and computes its sketch in a streaming manner. That is, when a new symbol $T[j]$ arrives the algorithm updates each of the $\mathcal{O}(n/B)$ suffix streams and each of its sketches in $\mathcal{O}((n/B) \cdot k^2 \log^4 n)$ time. The suffix sketches occupy $\mathcal{O}((n/B) \cdot k^8 \log^6 n)$ space in total.

We finally explain how the algorithm computes an output for a new arrival $T[j]$ in a block i. Recall that the task is to decide if $T[1, j]$ ends with a k-mismatch occurrence of P and if so to output the edit operations between the pattern and the occurrence. The length of the occurrence must be in $[n - k, n + k]$. It therefore starts either in block $i - n/B$ or in block $i + 1 - n/B$. The two cases are analogous and we consider only the case when the occurrence starts in block $i - n/B$. Let S be the suffix of $T[1, j]$ starting at the right border of block $i - n/B$ (in Fig. 2 the suffix is shown in green). S must be aligned with one of the $2k$ suffixes of the pattern of length in $[|S| - k, |S| + k]$. Using the sketches, we compute the edit distances (and the edit operations) from each of these suffixes to S. Consider a suffix

$P[i+1, n]$. If it is aligned with S, the prefix $P[1, i]$ must be aligned with some suffix of block $i - n/B$ and we have computed the minimal edit distance from $P[1, i]$ to the block or we know that it is larger than k. For each i, we sum the edit distances for the prefix and for the suffix and take the minimum. If the minimum is smaller than k, then by Observation 8 $T[1, j]$ ends with a k-mismatch occurrence of the pattern P and we can output the edit distance and the edit operations. In total, this step takes $\mathcal{O}(k^{13} \log^4 n)$ time.

We choose $B = \sqrt{n}$. The space complexity of the algorithm is then $\mathcal{O}(k^8 \sqrt{n} \log^6 n)$. The time for updating the sketches is $\mathcal{O}(k^2 \sqrt{n} \log^4 n)$ per arrival, and the time for computing the edit distance is $\mathcal{O}(k^{13} \log^4 n)$, meaning that the total time complexity is $\mathcal{O}((k^2 \sqrt{n} + k^{13}) \cdot \log^4 n)$ per arrival.

4 Conclusion

In this work we studied the approximate pattern matching problem. In particular we showed the first sublinear-space streaming algorithm for the problem. The space complexity of our algorithm is $\mathcal{O}(k^8 \sqrt{n} \log^6 n)$, which is significantly better than that of the previously known solutions. We note that on the other hand the time complexity of our algorithm is quite large as we have to update sketches of $\sqrt{n}$ text suffixes each time a new symbol arrives. One possibility to improve the time complexity is to maintain sketches of the blocks of the text rather than sketches of the suffixes (this way, the algorithm will need to update only one sketch per arrival). However, it is not clear whether the block sketches can be used to compute suffix sketches and therefore the edit distance. This is because the moves of a pointer in a suffix' blocks are not independent, in other words the image of a block under the random walk embedding depends on all preceding blocks. We leave this challenging question for further research.

References

1 Djamal Belazzougui. Efficient deterministic single round document exchange for edit distance, 2015. arXiv:1511.09229.

2 Djamal Belazzougui and Qin Zhang. Edit distance: Sketching, streaming, and document exchange. In Irit Dinur, editor, *Proceedings of the 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2016)*, pages 51–60. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.15.

3 Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. In Robert Krauthgamer, editor, *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 1884–1892. SIAM, 2016. doi:10.1137/1.9781611974331.ch132.

4 Diptarka Chakraborty, Elazar Goldenberg, and Michal Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2016)*, pages 712–725. ACM, 2016. doi:10.1145/2897518.2897577.

5 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In Robert Krauthgamer, editor, *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.ch142.

6 Raphaël Clifford, Markus Jalsenius, and Benjamin Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In Piotr Indyk, editor, *Proceedings of the*

26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015), pages 552–561. SIAM, 2015. doi:10.1137/1.9781611973730.37.

7 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

8 Graham Cormode, Mike Paterson, Süleyman Cenk Sahinalp, and Uzi Vishkin. Communication complexity of document exchange. In David B. Shmoys, editor, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000)*, pages 197–206. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338252.

9 Johannes Fischer and Volker Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM 2006)*, volume 4009 of *LNCS*, pages 36–48. Springer, 2006. doi:10.1007/11780441_5.

10 Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

11 Hossein Jowhari. Efficient communication protocols for deciding edit distance. In Leah Epstein and Paolo Ferragina, editors, *Proceedings of the 20th Annual European Symposium on Algorithms (ESA 2012)*, volume 7501 of *LNCS*, pages 648–658. Springer, 2012. doi:10.1007/978-3-642-33090-2_56.

12 Gad M. Landau and Uzi Vishkin. Introducing efficient parallelism into approximate string matching and a new serial algorithm. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC 1986)*, pages 220–230. ACM, 1986. doi:10.1145/12130.12152.

13 Eugene W. Myers. An $O(nd)$ difference algorithm and its variations. *Algorithmica*, 1(2):251–266, 1986. doi:10.1007/BF01840446.

14 Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. doi:10.1145/375360.375365.

15 Alon Orlitsky. Interactive communication: Balanced distributions, correlated files, and average-case complexity. In Michael Sipser, editor, *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS 1991)*, pages 228–238. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185373.

16 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In Daniel A. Spielman, editor, *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 315–323. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.11.

17 Ely Porat and Ohad Lipsky. Improved sketching of Hamming distance with error correcting. In Bin Ma and Kaizhong Zhang, editors, *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*, volume 4580 of *LNCS*, pages 173–182. Springer, 2007. doi:10.1007/978-3-540-73437-6_19.