Asymmetric Streaming Approximate Pattern Matching

Wojciech Janczewski[1]
Institute of Computer Science, University of Wrocław, Poland

Tatiana Starikovskaya
DIENS, École normale supérieure de Paris, PSL Research University, France

Abstract

We study the space complexity of pattern matching in the *asymmetric* streaming model, focusing on approximate pattern matching under the Hamming and edit distances. In this problem, we are given an m-length pattern and an n-length text and must compute, for every position of the text, the smallest distance between the pattern and a substring of the text which ends at this position. In the asymmetric streaming model, we assume to have constant-time random access to the pattern, while the text arrives as a stream, one letter at a time.

It is known that computing all distances exactly in the asymmetric streaming model requires $\Omega(m)$ space (for the edit distance see Li and Zheng [FSTTCS 2021]). Hence, to achieve sublinear space, a relaxation of the problem is necessary. One possible variant is to consider the small distance regime, where the algorithm must compute only those distances that are bounded by a small integer parameter k. In this case, existing algorithms in a more restrictive fully streaming model (Kociumaka, Clifford, Porat [SODA'19], Bhattacharya, Koucký [ICALP'23]) straightforwardly imply the existence of $\mathrm{poly}(k, \log n)$-space asymmetric streaming algorithms. Another possible relaxation is computing *all distances* approximately. For this variant, we don't have small-space algorithms in the fully streaming model: the best known algorithm solves pattern matching under the Hamming distance $(1+\epsilon)$-approximately using $\tilde{\mathcal{O}}(\epsilon^{-2}\sqrt{m})$ space (Starikovskaya, Svagerka, Uznański [APPROX'20]).[2] For the edit distance, no efficient approximation algorithms are known.

In this work, we show approximation algorithms for pattern matching under the Hamming and edit distances in the asymmetric streaming model for any constant $\epsilon > 0$:

1. We show that there is a simple randomised asymmetric streaming algorithm that solves approximate pattern matching under the Hamming distance $(1+\epsilon)$-approximately using $\mathcal{O}(\epsilon^{-3}\log^3 n)$ bits.
2. As our second and main contribution, we extend the result of Cheng et al. [ICALP 2021] and show that for any integer k there is a deterministic asymmetric streaming algorithm that solves pattern matching under the edit distance $(2^k - 1 + \epsilon)$-approximately using $\tilde{\mathcal{O}}(m^{1/k})$ space.

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Pattern matching

Keywords and phrases Asymmetric streaming, Pattern matching, Approximation, Edit distance, Hamming distance

Digital Object Identifier 10.4230/LIPIcs.CPM.2026.19

Funding Partially funded by a grant ANR-20-CE48-0001 from the French National Research Agency and by the Polish National Science Centre grant number 2023/51/B/ST6/01505.

[1] During the preparation of this paper, Wojciech Janczewski was a postdoctoral researcher at DI ENS, École normale supérieure, Paris, PSL University, France.
[2] Hereafter, $\tilde{\mathcal{O}}()$ hides factors polylogarithmic in n.

1 Introduction

In this work, we study the space complexity of pattern matching in the streaming setting. We focus on approximate pattern matching under the Hamming and edit distances. For the Hamming distance and a pattern of length m, the algorithm should output the Hamming distance to the pattern for each length-m window of the text. For the edit distance, we should find, for every position j of text T, the distance between pattern P and a substring $T[i, j]$ of T for a position i minimising this value.

In the fully streaming setting, we assume that P arrives first, the algorithm preprocesses it, and then receives T as a stream. We account for all the space used to store the information about the pattern (which is not available via random access) and the text. This work is focused on the more recent *asymmetric* streaming model proposed by Saks and Seshadhri [18] in the context of the longest common subsequence problem (see also the work of Andoni, Krauthgamer, and Onak [3]).

In this model, we are given constant-time random access to the pattern P, which arrives first, and the text T arrives as a stream. Thus, we have read-only access to the pattern and do not account for the space used to store it. This is a natural variant of the fully streaming setting, as two parts of the input (pattern/text) are inherently different and it is easy to imagine some fixed relatively short pattern residing in working memory, while fully storing the whole text is out of reach. As such, the asymmetric streaming model represents a practically interesting intermediate point between the read-only setting (where both the pattern and the text are read-only) and the fully streaming setting.

It is known that computing all distances exactly in the approximate pattern matching problem, by information-theoretic argument, requires $\Omega(m)$ space, even in the asymmetric streaming model. Hence, to achieve sublinear space one must consider a relaxation of the problem.

One possible variant is to consider the small distance regime, where one must compute only those distances that are bounded by a small integer parameter k. In such a case, efficient fully streaming algorithms exist, and they carry on to the asymmetric streaming model. The study of fully streaming algorithms for pattern matching under the Hamming distance in the small-distance regime was initiated by Porat and Porat [17], who showed that the problem can be solved in $\tilde{\mathcal{O}}(k^3)$ space and $\tilde{\mathcal{O}}(k^2)$ time per letter of the text. Subsequent improvements [7, 14] led to the current best solution [8], which solves the problem in $\mathcal{O}(k \log n)$ space and $\tilde{\mathcal{O}}(\sqrt{k})$ time per letter of the text. [13] studied trade-offs between streaming space and time. For the edit distance, two algorithms are known [15, 4], the best [4] by Bhattacharya and Koucký taking $\tilde{\mathcal{O}}(k^2)$ space and time in the streaming model.

Another possible relaxation is computing *all distances* approximately. For this variant, we don't have algorithms using polylogarithmic space in the fully streaming model: the best known algorithm solves pattern matching under the Hamming distance $(1+\epsilon)$-approximately using $\tilde{\mathcal{O}}(\epsilon^{-2}\sqrt{n})$ space [9, 19]. For the edit distance, no efficient approximation algorithms are known.

1.1 Our results

In this work, we study the approximate pattern matching and show that the asymmetric streaming model does allow for space-efficient algorithms.

Hamming distance. As our first result, we give an efficient algorithm for the approximate pattern matching under the Hamming distance, in the asymmetric streaming model:

▶ **Theorem 1.** *Let $\epsilon > 0$ be a constant, P be a read-only pattern of length m, and T be a streaming text of length n on a polynomial-size alphabet. There is an asymmetric streaming algorithm that solves approximate pattern matching under the Hamming distance $(1+\epsilon)$-approximately using $\mathcal{O}(\epsilon^{-3}\log^3 n)$ bits and polynomial time. The algorithm is randomised and is correct with high probability.*[3]

By the lower bound for counting ones in a sliding window [10] (see also [9]), any asymmetric streaming algorithm for this problem must use $\Omega(\epsilon^{-1}\log^2 m)$ bits.

The main idea of our algorithm is to divide the text, for every natural $k \leq \log m$, into blocks of varying length but each at a distance roughly 2^k from the pattern. For any level k and each of the $\lceil \epsilon^{-1} \rceil$ latest blocks on this level, we store a sketch which uses $\mathcal{O}(\epsilon^{-2}\log^2 n)$ bits and allows to compute $(1+\epsilon)$-approximation of the Hamming distance. We then show that if the Hamming distance between P and $T[j-m+1, j]$ is at most $\epsilon^{-1}2^k$, where $T[j]$ is the latest arrived letter of the text, then whole $T[j-m+1, j]$ is covered by the stored blocks on level k, and we can use the sketches to compute an approximation of the Hamming distance, with some additive error which is reduced by the chosen parameters.

Edit distance. As our second and main contribution, we show a space-efficient asymmetric streaming algorithm for approximating the edit distances between the pattern and text substrings. Namely, for each position j of the text T, we must compute an approximate value of the smallest possible edit distance between P and some substring $T[i, j]$.

Asymmetric query complexity for edit distance was first considered in [3], with free access to one string and the goal of minimising the number of queries for letters in the other string. Saks and Seshadhri [18] furthered this idea and developed the asymmetric streaming model, along with the first asymmetric streaming algorithm for the longest common subsequence problem (LCS). The merged paper of Cheng, Farhadi, Hajiaghayi, Jin, Li, Rubinstein, Seddighin, and Zheng [5] (full versions [11], [6]) revisited the problem of computing the edit distance, and showed that there is (i) a deterministic asymmetric streaming algorithm which computes $\mathcal{O}(2^k)$-approximation of the edit distance between two strings using $\tilde{\mathcal{O}}(km^{1/k})$ space and polynomial time, and (ii) $(1+\epsilon)$-approximation of the edit distance between two strings using $\tilde{\mathcal{O}}(\epsilon^{-1}\sqrt{n})$ space for any constant $\epsilon > 0$. Li and Zheng [16] improved the space complexity of the $\mathcal{O}(2^k)$-approximation algorithm to $\tilde{\mathcal{O}}(kd^{1/k})$, where d is the edit distance between the input strings. They also showed that any asymmetric streaming algorithm (even randomised) computing the edit distance exactly must use $\Omega(n)$ bits, and that computing the edit distance $(1+\epsilon)$-approximately requires $\Omega(\epsilon^{-1})$ bits.

For the problem of approximate pattern under the edit distance, we show the following:

▶ **Theorem 2.** *Let $\epsilon > 0$ be a constant, k a constant integer, P a read-only pattern of length m, and T a streaming text on a polynomial-size alphabet. There is an asymmetric streaming algorithm that solves approximate pattern matching under the edit distance $(2^k - 1 + \epsilon)$-approximately, using $\tilde{\mathcal{O}}(m^{1/k})$ bits and polynomial time. The algorithm is deterministic.*

Therefore, we are able to extend the result of [5] from string-to-string edit distance to approximate pattern matching under the edit distance, in the same space complexity $\tilde{\mathcal{O}}(m^{1/k})$, losing only an additional ϵ factor in the (large) constant-factor approximation.

The structure of our solution for the edit distance is similar to that for the Hamming distance; in particular, we still divide text into blocks, for multiple levels parametrised by distances. However, unlike the Hamming distance, there are no small-space sketches for

[3] By high probability we mean probability that is bigger than $1 - 1/n^c$ for a constant $c \geq 1$.

approximating the edit distance between strings, which is the main source of difficulty. In their solution for string-to-string edit distance computation, [5] suggested approximating the blocks of the text with the closest substrings of the pattern (to which we have constant-time random access). We extend their techniques to pattern matching. There are some challenges in this approach: first, [5] works with a static decomposition of a fixed-length string, while we must maintain the decomposition for appropriate suffixes of the streamed longer text. To this end, we maintain a number of the latest static blocks in a queue, but then a substring $T[i, j]$, where j is the latest arrived letter of the text, is rarely fully aligned with the chosen blocks. The first and last blocks need to be trimmed, introducing a small additive error, which nevertheless can be transformed into a multiplicative one.

Open questions. Our results lead to intriguing open problems. First, what is the best time complexity one can achieve for approximate pattern matching under the Hamming distance while using polylogarithmic space? Ideally, the time per letter of the text would also be polylogarithmic, but in our solution complexity is dominated by computing distances to all substrings of given length, which takes quadratic time per letter due to sketch properties. Second, can pattern matching under the Hamming distance be solved with a deterministic algorithm more efficiently than under the edit distance? Our space-efficient algorithm uses randomised sketches.

For the pattern matching with edits, can we achieve $(1 + \epsilon)$-approximation in $\tilde{\mathcal{O}}(\epsilon^{-1})$ space? Note that even for only string-to-string edit distance, the best existing solution takes $\tilde{\mathcal{O}}(\epsilon^{-1}\sqrt{n})$ bits, so the first step would be to match this for pattern matching. Finally, can randomisation help to achieve a better space or time complexity for the edit distance? The time complexity of known deterministic algorithms is large.

1.2 Preliminaries

Throughout the paper, we index strings starting from 1. $S[i, j]$ is a substring of S from index i to j, which is the same as $S[i, j + 1)$ or $S(i - 1, j]$. $S_1 \circ S_2$ denotes the concatenation of two strings. $\mathrm{ED}(S_1, S_2)$ is the edit distance between two strings, that is, the minimal number of single-letter insertions, deletions, or substitutions necessary to transform S_1 into S_2. $\mathrm{HD}(S_1, S_2)$ denotes the Hamming distance between a pair of strings of equal length, that is, the number of letter substitutions necessary to transform S_1 into S_2. By k-approximation of value v we will always mean a number in the range $[v, kv]$.

The distance of string S from pattern P is the minimum distance between S and any substring of P.

2 Approximate Hamming Distance

We first assume that the alphabet is binary and show how to lift this assumption later. For the Hamming distance, one can adapt randomised so-called *tug-of-war sketches* of polylogarithmic size, developed for estimating frequency moments in the stream [2, 12]. We will use the following sketches, designed by Achlioptas:

▶ **Corollary 3** ([1, Theorem 1.1], see also [9, 19]). *There is a sketching function $f\colon \{0,1\}^{\le n} \to \{0,1\}^{\mathcal{O}(\epsilon^{-2}\log^2 n)}$, such that for any two strings S_1, S_2 of equal lengths at most n, we can with high probability obtain $(1 + \epsilon)$-approximation of $\mathrm{HD}(S_1, S_2)$ using $f(S_1), f(S_2)$, in $\mathcal{O}(\epsilon^{-2}\log n)$ time. The sketch $f(S)$ of a string $S \in \{0,1\}^{\le n}$ can be maintained in streaming using $\mathcal{O}(\epsilon^{-2}\log^2 n)$ bits and in $\mathcal{O}(\epsilon^{-2}\log n)$ time per letter.*

Let us define an *approximate block* to be a tuple storing a Hamming distance sketch for some text substring $T[i, j]$, length $j - i + 1$, as well as a pair of indices describing a pattern substring $P[i', j']$ (which will be close to $T[i, j]$). We say that such a block is an approximation of $T[i, j]$.

Queues Construction. For each value 2^t, where $0 \leq t \leq \lceil \log m \rceil$, we create a queue Q_t storing up to $\lceil \epsilon^{-1} \rceil$ approximate blocks. Assume that the last block in the queue Q_t is an approximation of a substring of T ending at a position $i - 1$ (if Q_t is empty, set $i = 1$). We maintain the Hamming distance sketch of a substring starting at the index i. When a new letter $T[j]$ arrives, we update the sketch and use it to compute the minimum of approximate Hamming distances from $T[i, j]$ to substrings of P of length $j - i + 1$. If this minimum is at least $(1+\epsilon)2^t$, we create a new approximate block, where the stored indices are the endpoints of a substring of P realising the minimum approximate distance to $T[i, j]$, and add the block to Q_t. By Corollary 3, with high probability $T[i, j]$ is at distance at least 2^t from P, and the distance between the block and the stored substring of P is in range $[2^t, (1 + \epsilon)2^t]$. It might also be the case that we read m letters from the stream without achieving the threshold of distance from P, if the substring of the text is very similar to the pattern; then we also finalise a block. If the size of Q_t exceeds $\lceil \epsilon^{-1} \rceil$, the oldest block is removed from the queue.

Note that the last processed index of T is usually further than the end of the last block in Q_t, as the new approximate block covering the current suffix of the streamed text has not yet been created. Nevertheless, the algorithm maintains a sketch of this suffix. We have:

▶ **Definition 4.** *The index k in T is* covered *by the queue of approximate blocks Q if the first block in Q is an approximation of $T[i, j]$ and $i \leq k$.*

▶ **Proposition 5.** *Let the last processed index of T be j, and $i = j - m + 1$. For any t, if $\mathrm{HD}(P, T[i, j]) < \epsilon^{-1}2^t$, then with high probability i is covered by Q_t.*

The above follows from the fact that Q_t holds up to $\lceil \epsilon^{-1} \rceil$ last blocks, and each block is an approximation of some $T[q_1, q_2]$ at distance at least 2^t from P. Assume that $\mathrm{HD}(P, T[i, j]) < \epsilon^{-1}2^t$, then we observe that Q_t cannot contain $\lceil \epsilon^{-1} \rceil$ such blocks starting at indices larger than i. This means the first block in Q_t starts before i, and $T[i, j]$ is covered by Q_t.

Approximating the distance. Let $T' = T[i, j]$ be the current m-length suffix of T, so $i = j - m + 1$. Assume that $\epsilon^{-1}2^{t-1} \leq \mathrm{HD}(P, T') < \epsilon^{-1}2^t$, so i is covered by Q_t. Let $B[1, b]$ be a substring of P stored in a block in Q_t approximating $T[q_1, q_2]$ with $q_1 \leq i$ and maximal q_1. For all blocks following B in Q_t and the last unfinished block, we have sketches and strings lengths, therefore we can obtain $(1+\epsilon)$-approximation of $\mathrm{HD}(P[q_2 - i + 2, m], T'[q_2 - i + 2, m])$ by partitioning $P[q_2 - i + 2, m]$ into substrings with lengths equal to consecutive blocks lengths, computing sketches of these substrings, and finally summing approximate block-to-substring distances. Let the resulting value be D. For the prefix of T' there is no sketch available, but letting $k = q_2 - i + 1$, by construction $B(b - k, b]$ is at a distance at most $(1 + \epsilon)2^t$ from $T'[1, k]$, thus by the triangle inequality:

$$\mathrm{HD}(P[1, k], T'[1, k]) - (1+\epsilon)2^t \leq \mathrm{HD}(P[1, k], B(b-k, b]) \leq \mathrm{HD}(P[1, k], T'[1, k]) + (1+\epsilon)2^t$$

We can compute $\mathrm{HD}(P[1, k], B(b - k, b]) + (1 + \epsilon)2^t$ exactly, and finally obtain that

$$\begin{aligned}
&\mathrm{HD}(P, T') \leq D + \mathrm{HD}(P[1, k], B(b - k, b]) + (1 + \epsilon)2^t \\
&\leq (1 + \epsilon)\mathrm{HD}(P[q_2 - i + 2, m], T'[q_2 - i + 2, m]) + \mathrm{HD}(P[1, k], T'[1, k]) + 2(1 + \epsilon)2^t \\
&\leq (1 + \epsilon)\mathrm{HD}(P, T') + 2(1 + \epsilon)2^t < (1 + 6\epsilon)\mathrm{HD}(P, T'),
\end{aligned}$$

since $\epsilon^{-1}2^{t-1} \leq \mathrm{HD}(P, T')$, and for $\epsilon < 1/2$.

Complexity analysis. We use sketches of length $\mathcal{O}(\epsilon^{-2}\log^2 n)$, for every of the $\mathcal{O}(\log m)$ queues, each storing ϵ^{-1} blocks, resulting in space complexity of $\mathcal{O}(\epsilon^{-3}\log^3 n)$ bits. With high probability, the output is correct for every alignment. Time complexity is $\tilde{\mathcal{O}}(m^2\epsilon^{-3})$ per letter, dominated by using sketches to compute distances to all substrings of fixed lengths.

Extension to polynomial-size alphabets. To handle any polynomial-size alphabet, we can use a known method described for example in [19]. For the alphabet $\Sigma = \{0, 1, \ldots, k\}$ we use mapping $h : \Sigma \to \{0,1\}^{k+1}$ such that $h(x) = 0^x 10^{k-x}$, which exactly doubles the Hamming distance. Given string S over Σ, we instead run the algorithm on $h(S)$ over the binary alphabet. Then in Corollary 3 sketches use $\Theta(\epsilon^{-2}\log^2(n|\Sigma|))$ bits, which is asymptotically the same for a polynomial-size alphabet. Time complexity is not affected either, due to the sketching algorithm performing operations only for letters 1 in the text, see [19].

3 Approximate Edit Distance

In this section, we prove Theorem 2. We will use the following basic facts:

▶ **Proposition 6.** *For any integer z, consider arbitrary partitions of two strings into z substrings as $S_1 = A_1 \circ A_2 \circ \ldots \circ A_z$ and $S_2 = B_1 \circ B_2 \circ \ldots \circ B_z$. It holds that* $\mathrm{ED}(S_1, S_2) \leq \sum_{l=1}^{z} \mathrm{ED}(A_l, B_l)$. *Moreover, there exists a partition of S_2 into $S_2 = B'_1 \circ B'_2 \circ \ldots \circ B'_z$ such that* $\mathrm{ED}(S_1, S_2) = \sum_{l=1}^{z} \mathrm{ED}(A_l, B'_l)$.

We will also use redefined approximate blocks:

▶ **Definition 7.** *k-approximate block for a text substring $T[i, j]$ and the pattern P is a pair of indices describing $P[i', j']$ and an* additive value *d such that for any substring P^* of the pattern it holds that* $\mathrm{ED}(P^*, T[i, j]) \leq \mathrm{ED}(P^*, P[i', j']) + d \leq k\mathrm{ED}(P^*, T[i, j])$.

In other words, by leveraging the read-only pattern, we can store just $\mathcal{O}(\log n)$ bits, providing us with a k-approximation of the edit distance between some text substring and any substring of the pattern. For brevity, we will often say that some $P[i', j']$ is an approximate block, omitting stored additive factor d. The following observation using the triangle inequality is a cornerstone of algorithms for asymmetric streaming edit distance:

▶ **Proposition 8** ([11])**.** *Let P' be a substring of P with the smallest edit distance to some text substring $T[i, j]$. Then P' with an additive value of* $\mathrm{ED}(P', T[i, j])$ *is a 3-approximate block for $T[i, j]$. (See Figure 1.)*

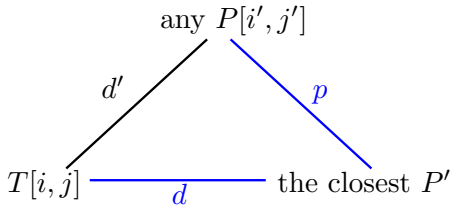

Figure 1 We have $d \leq d' \leq d + p \leq 3d'$, so the closest substring in P provides 3-approximation for edit distance d' between T and substrings of P. We will not compute the edit distance to strings outside of P.

For any string S, we will refer to a substring of P with the smallest edit distance to S as its *closest substring*. Recall that the distance of S from P is the distance between S and its closest substring. We will also use:

▶ **Fact 9** ([5, Theorem 3]). *For any constants $\delta < 1/2$ and $\epsilon < 1$, there is an algorithm that outputs $(1+\epsilon)$-approximation of edit distance between two given read-only strings using $\tilde{\mathcal{O}}(n^\delta/\epsilon)$ space and polynomial time.*

3.1 First Step: $(3+\epsilon)$-approximation in $\tilde{\mathcal{O}}(m^{1/2})$ space

We start by showing the following result:

▶ **Theorem 10.** *There is an asymmetric streaming algorithm providing a $(3+\epsilon)$-approximation for pattern matching with edits in $\tilde{\mathcal{O}}(m^{1/2})$ space and polynomial time.*

Theorem 2 is then obtained by applying this algorithm recursively, as described in the next subsection. Let us set $r = m^{1/2}$. The goal is to maintain a queue Q of up to r 3-approximate blocks, each at the distance $\Theta(r)$ from P.

To this end, consider some index i in T, initially $i = 1$, and read $T[i, i+r)$ from the stream. Then compute, in space $\tilde{\mathcal{O}}(r)$, the closest substring P' along with the full list of d edit operations necessary to transform P' into $T[i, i+r)$. Next, remove $T[i, i+r)$ from the working space and read $T[i+r, i+2r)$. Observe that we still have access to all letters of $T[i, i+2r)$, via P' and the list of edits. Therefore, we can compute the closest substring P'' of $T[i, i+2r)$, again with the full list of edit operations. If $\mathrm{ED}(P'', T[i, i+2r)) > r$, we stop and place P'' in the queue Q as a 3-approximate (by Proposition 8) block for the processed fragment of text, clearly at the distance of $\Theta(r)$ from P. In the other case, the process continues iteratively until the closest substring is finally at a distance larger than r. Then we push the block on the queue, removing the oldest element from Q if the queue already contained r blocks. Until the block is finished, we retain full access to the corresponding suffix of T. This is presented in Algorithm 1. See also Figure 2.

Algorithm 1 Computing a single block in a queue.

1: **function** NEXT-BLOCK(P, T, i, Q, r)
2: **Input:** read-only pattern P, index i in streamed T, queue Q, parameter r.
3: **Output:** Q is updated with a new block.
4: $O, S \leftarrow \emptyset$
5: $S' \leftarrow T[i, i+r), i \leftarrow i+r$ ▷ Read new fragment into buffer
6: Compute $P[l_1, l_2]$, the closest substring of S'
7: **while** $\mathrm{ED}(S \circ S', P[l_1, l_2]) \leq r$ **do**
8: Store edit operations transforming $P[l_1, l_2]$ into $S \circ S'$ in O
9: Represent $S \circ S'$ as a tuple (l_1, l_2, O) and store as a new S
10: $S' \leftarrow T[i, i+r), i \leftarrow i+r$ ▷ Read new fragment into buffer
11: Compute $P[l_1, l_2]$, the closest substring of $S \circ S'$
12: **if** $|Q| = r$ **then**
13: Remove the first block from Q
14: Push $(l_1, l_2, \mathrm{ED}(S \circ S', P[l_1, l_2]))$ on Q

The set of edit operations O can be stored sorted by indices, allowing access to $S \circ S'$ while using $\mathcal{O}(r \log m)$ bits. We can compute the edit distance in line 7 using $\mathcal{O}(r \log m)$ bits, even though the two strings can be much longer. This is since the distance cannot be larger than $2r$, thus computing $\mathcal{O}(r)$ central diagonals in the standard dynamic programming for the edit distance is enough:

▶ **Proposition 11.** *For two read-only strings S_1, S_2, if $\mathrm{ED}(S_1, S_2) = \mathcal{O}(r)$, we can compute $\mathrm{ED}(S_1, S_2)$ using only $\tilde{\mathcal{O}}(r)$ additional space.*

The set of edit operations can be obtained similarly. Let us now consider a useful property of the computed queue.

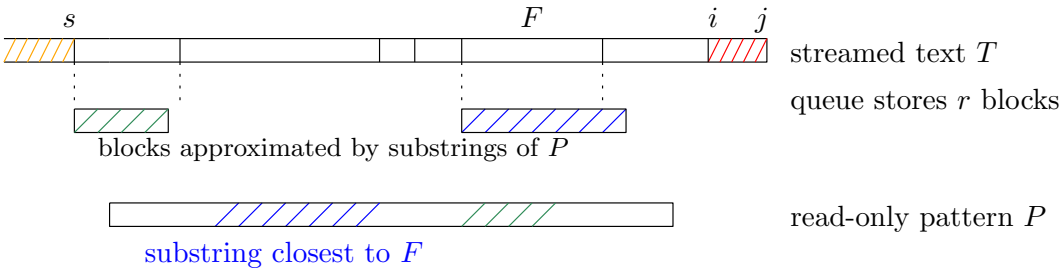

Figure 2 A queue stores up to r approximate blocks. The last read letter of the text is $T[j]$. A suffix $T[i, j]$, marked red, is currently being processed and is not yet represented in the queue as a full block. For $T[s, i)$, the queue stores 3-approximate blocks representing its consecutive fragments. For example, a substring F of $T[s, i)$ is represented by its closest substring in the pattern P (of possibly different length, and marked blue). $T[1, s)$, marked orange, is not covered anymore as the queue stores no more than r blocks, but any index larger than s is covered.

▶ **Proposition 12.** *Let the last processed index of T be j. For any $i \leq j$, if $\mathrm{ED}(P, T[i, j]) \leq r^2 = m$, then i is covered by Q.*

Proof. The above follows from the fact that Q holds up to r latest blocks, and each block is an approximation of some $T[q_1, q_2]$ at distance from P exceeding r.

Assume $\mathrm{ED}(P, T[i, j]) \leq r^2 = m$, and i is not covered by Q, thus the queue must be full. For the k-th block in Q for $k \in [1, r]$, denote that it stores $P[a_k, a'_k]$ as an approximation of $T[b_k, b_{k+1})$. As i is not covered, we have $b_1 > i$. By Proposition 6, consider a partition of $P = P_0 \circ P_1 \circ \ldots \circ P_{r+1}$ such that $\mathrm{ED}(P, T[i, j]) = \mathrm{ED}(P_0, T[i, b_1)) + \sum_{k=1}^{r} \mathrm{ED}(P_k, T[b_k, b_{k+1})) + \mathrm{ED}(P_{r+1}, T[b_{r+1}, j])$. By definition of blocks stored in Q, they are at distance larger than r from P, so for any k $\mathrm{ED}(P_k, T[b_k, b_{k+1})) > r$. But this contradicts $\mathrm{ED}(P, T[i, j]) \leq r^2 = m$, as there are r blocks at a distance larger than r. ◀

It remains to bound the approximation we can obtain using the queue. For some index i, assume that $\mathrm{ED}(P, T[i, j]) \leq m$, with the last processed index of T being j. By Proposition 12, $T[i, j]$ is covered by Q. We consider a string T' constructed as follows, in order to approximate $T[i, j]$ (which is not stored explicitly) with blocks from the queue.

Let $B = [1, b]$ be a block in Q being approximation of $T[q_1, q_2]$ with $q_1 \leq i$ and maximal q_1. Recall B is a substring of P. Set $T' = B(i - q_1, b]$, as B is an approximate block for a substring of T extending too far to the left of i. Since we are interested in the edit distance to just $T[i, q_2]$, we drop a prefix of B of length $i - q_1$, using the fact that removing the first letters from two strings does not increase the edit distance between them. That is, $\mathrm{ED}(S_1, S_2) \geq \mathrm{ED}(S_1[q, |S_1|], S_2[q, |S_2|])$, for any $q < |S_1|, |S_2|$. Note that we need to know q_1, but it can be stored for any approximate block without affecting space complexity (we omit this detail in the pseudocode).

Then, append to T' all blocks following B in the queue, and finally append the suffix of T which is still being processed by the queue procedure. Let D_l be an additive value for the l-th concatenated full approximate block, and $D = \sum D_l$. See Figure 3.

We do not build T' explicitly, as it can be much more than r letters. It is just represented as a sequence of indices to the read-only pattern, and for the unprocessed suffix, additionally a list of edit operations and a buffer of the most recent letters. Next, we show that T' can be used to approximate distance between $T[i,j]$ and arbitrary substring of the pattern:

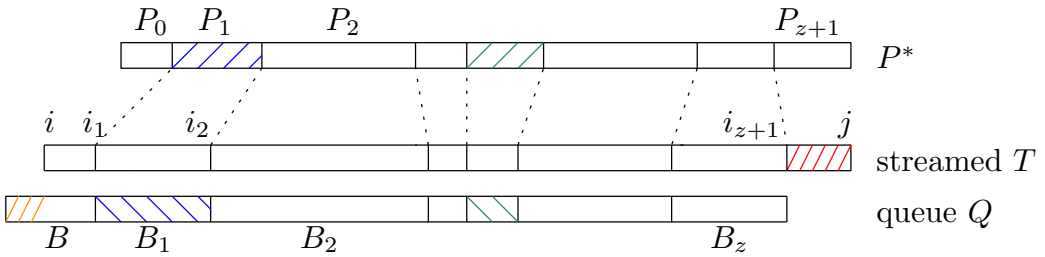

Figure 3 We want to approximate edit distance between $T[i,j]$ and P^*, a suffix of T processed so far and an arbitrary substring of P, using Q which covers i. $T[i,j]$ is naturally split into $z+2$ fragments according to blocks from Q. P^* can then be optimally split into $z+2$ fragments such that $\mathrm{ED}(P^*,T[i,j])$ is obtained by summing edit distances between corresponding pairs of substrings, for example between $T[i_1,i_2)$ and P_1. Text fragments are approximated by blocks stored in Q, so B_1 is 3-approximate block for $T[i_1,i_2)$, allowing us to compare $T[i_1,i_2)$ (not stored in the memory) to P_1. Block B extends to the left of i and will be trimmed (the orange part). After the last block B_z, the next block is not yet constructed, but this means we can access all letters of $T[i_{z+1},j]$ (the red part). All blocks are directly represented as substrings of P, with stored additive values.

▶ **Lemma 13.** *Let the last processed index of T be j, P^* be any substring of P such that $\mathrm{ED}(P^*,T[i,j]) = d \le m$, and $\mathrm{ED}(P^*,T') = d'$. We can compute in polynomial time $(1+\epsilon)$-approximation of d', and a value v such that $d \le d' + v \le 3d + 4r$.*

Proof. Recall that $B = [1,b]$ is a block in Q being approximation of $T[q_1,q_2]$ with $q_1 \le i$ and maximal q_1. Let $i_1, i_2, \ldots, i_{z+1}$ be the sequence of indices such that l-th block B_l in Q after initial block B is an approximation of $T[i_l, i_{l+1})$. Note that:

$$T' = B(i-q_1,b] \circ B_1 \circ \ldots \circ B_z \circ T[i_{z+1},j].$$

By Proposition 6, P^* can be partitioned into consecutive substrings $P_0,\ldots,P_{z+1}$ such that $\mathrm{ED}(P^*,T[i,j]) = \mathrm{ED}(P_0,T[i,i_1)) + \sum_{l=1}^{z} \mathrm{ED}(P_l,T[i_l,i_{l+1})) + \mathrm{ED}(P_{z+1},T[i_{z+1},j])$. Note this is not necessarily a partition optimal for computing $\mathrm{ED}(P^*,T')$ with selected blocks.

We have $\mathrm{ED}(T[i,i_1), B(i-q_1,b]) \le 2r$, which means that:

$$\mathrm{ED}(S,T[i,i_1)) - 2r \le \mathrm{ED}(S,B(i-q_1,b]) \le \mathrm{ED}(S,T[i,i_1)) + 2r \tag{1}$$

for any string S, by the triangle inequality. We cannot claim 3-approximation here, as $B(i-q_1,b]$ is not the closest substring of $T[i,i_1)$, but only a trimmed closest substring of $T[q_1,q_2]$, which was at distance at most $2r$.

Moreover, from definition of 3-approximate blocks, for any l it holds that $\mathrm{ED}(P_l,T[i_l,i_{l+1})) \le \mathrm{ED}(P_l,B_l) + D_l \le 3\mathrm{ED}(P_l,T[i_l,i_{l+1}))$. We want to avoid guessing the optimal partition of P^* necessary for block-by-block computation, so instead need to tie $\mathrm{ED}(P^*,T')$, relatively easy to compute, to $\mathrm{ED}(P^*,T[i,j]) = d$. The algorithm will return value $\mathrm{ED}(P^*,T') + 2r + D$. As an upper bound for this estimate of d, we get:

▷ Claim 14. $\mathrm{ED}(P^*,T') + 2r + D \le 3d + 4r$.

Proof.

$$\mathrm{ED}(P^*, T') + 2r + D \tag{2}$$

$$\leq \mathrm{ED}(P_0, B(i-q_1, b]) + 2r + \sum_{l=1}^{z} (\mathrm{ED}(P_l, B_l) + D_l) + \mathrm{ED}(P_{z+1}, T[i_{z+1}, j]) \tag{3}$$

$$\leq \mathrm{ED}(P_0, T[i, i_1)) + 4r + \sum_{l=1}^{z} 3\mathrm{ED}(P_l, T[i_l, i_{l+1})) + \mathrm{ED}(P_{z+1}, T[i_{z+1}, j]) \tag{4}$$

$$\leq 3d + 4r. \tag{5}$$

(3) holds as we partition P^* into $P_0, P_1, \ldots$, and T' into blocks. (4) is by using (1) and the blocks being 3-approximate. (5) follows just by the chosen partition of P^* optimal for computing $\mathrm{ED}(P^*, T[i, j])$. ◁

To obtain a lower bound for the returned estimate, let us consider another partition of P^* into $z+2$ consecutive substrings $P'_0, \ldots, P'_{z+1}$ such that $\mathrm{ED}(P^*, T') = \mathrm{ED}(P'_0, B(i-q_1, b]) + \sum_{l=1}^{z} \mathrm{ED}(P'_l, B_l) + \mathrm{ED}(P'_{z+1}, T[i_{z+1}, j])$.

▷ Claim 15. $d \leq \mathrm{ED}(P^*, T') + 2r + D$.

Proof.

$$d \leq \mathrm{ED}(P'_0, T[i, i_1)) + \sum_{l=1}^{z} \mathrm{ED}(P'_l, T[i_l, i_{l+1})) + \mathrm{ED}(P'_{z+1}, T[i_{z+1}, j]) \tag{6}$$

$$\leq \mathrm{ED}(P'_0, B(i-q_1, b]) + 2r + \sum_{l=1}^{z} (\mathrm{ED}(P'_l, B_l) + D_l) + \mathrm{ED}(P'_{z+1}, T[i_{z+1}, j]) \tag{7}$$

$$= \mathrm{ED}(P^*, T') + 2r + D \tag{8}$$

(6) follows from $P'_0, P'_1, \ldots$ being a partition of P^*. (7) is by using (1) and the blocks being 3-approximate. (8) holds by the chosen partition of P^* optimal for computing $\mathrm{ED}(P^*, T')$. ◁

By Claims 14 and 15, $d \leq \mathrm{ED}(P^*, T') + 2r + D \leq 3d + 4r$. The algorithm can compute $(1+\epsilon)$-approximation of $\mathrm{ED}(P^*, T')$, using Fact 9, as we have access to all letters of T'. $2r + D$ is known to the algorithm. ◀

By Lemma 13, for any constant ϵ we can compute $(3+\epsilon)$-approximation of $\mathrm{ED}(P^*, T[i, j])$ but with an additive factor $\mathcal{O}(r)$. To get rid of it, we create an additional queue Q_0 storing blocks at a constant distance from P, augmented with the full list of edit operations transforming a substring of P into a substring of T; so these are in fact not approximate blocks, but provide exact access to substrings of T. This can be done by a variant of Algorithm 1, replacing the inequality in line 7 with < 1 instead of $< r$, and additionally reading only a single letter into the buffer. This ensures that the distance of a block from P is always 1. Q_0 stores up to $r \log m$ augmented blocks, and we have that:

▶ **Proposition 16.** *Let the last processed index of T be j. For any i, if $\mathrm{ED}(P, T[i, j]) \leq r \log m$, then i is covered by Q_0.*

This allows us to compute the edit distance exactly for suffixes of T close to the pattern. Otherwise, we can assume that $\mathrm{ED}(P, T[i, j]) = d \geq r \log m$. By Lemma 13 and Proposition 12, we can obtain estimate w such that $d \leq w < (1+\epsilon)(3d+4r) \leq (1+\epsilon)3d(1+4/(3\log m))$. As $1/\log m = o(1)$, Algorithm 2 can compute $(3+\epsilon)$-approximation of $\min_i \mathrm{ED}(T[i, j], P)$ for any constant ϵ, for m large enough.

The total space used by the algorithm is $\mathcal{O}(m^{1/2}\log^2 m)$ bits, as Q_0 uses $\mathcal{O}(r\log^2 m)$ bits for storing blocks and lists of edit operations. Q uses a smaller number of $\mathcal{O}(r\log m)$ bits to store the blocks and one list of edit operations for the unfinished block. $\mathcal{O}(r\log m)$ bits is also enough to compute this list of edit operations by Proposition 11. Using Fact 9 requires asymptotically fewer bits. This finishes the proof of Theorem 10.

Algorithm 2 Algorithm to obtain approximate edit distance for the best suffix.

```
function ED(P, T, j)
    Input: read-only pattern P, the current index j in streamed T.
    Output: approximate ED(T[i, j], P)

    R ← ∞
    Let i be the smallest index covered by Q
    for i' ← i to j do
        if T[i', j] is covered by Q_0 then
            Let R' be ED(P, T[i', j]) obtained using Q_0
        else
            Let R' be approximate ED(P, T[i', j]) obtained using Q
        R ← min(R, R')
    return R
```

3.2 Second Step: $(2^k - 1 + \epsilon)$-approximation in $\tilde{\mathcal{O}}(m^{1/k})$ space

In this section, we apply the previous approach iteratively, arriving at our main result of Theorem 2. We need the following extension of Proposition 8:

▶ **Proposition 17.** *Let d be the distance from some text substring $T[i, j]$ to its closest substring in P. Let P' be a substring of P, and v be a value for which $\mathrm{ED}(P', T[i, j]) \leq v \leq g \cdot d$ holds, for some g. Then P' with an additive value v is $(2g + 1)$-approximate block for $T[i, j]$.*

Proof. Let P^* be any substring of P, by assumption $\mathrm{ED}(P^*, T[i, j]) \geq d$. $\mathrm{ED}(P^*, T[i, j]) \leq v + \mathrm{ED}(P^*, P')$ by the triangle inequality. On the other hand, $v + \mathrm{ED}(P^*, P') \leq v + \mathrm{ED}(P^*, T[i, j]) + ED(P', T[i, j]) \leq 2v + \mathrm{ED}(P^*, T[i, j]) \leq (2g + 1)\mathrm{ED}(P^*, T[i, j])$. ◀

For an integer k, set $r = m^{1/k}$ as roughly the space complexity we are aiming for. We assume r is a power of two, otherwise we can take the largest power of two smaller than $m^{1/k}$. The algorithm maintains the set of queues Q_t for $t \in [0, k)$, each queue holding up to $r \log m$ approximate blocks (assume $\log m$ is an integer). The index t is called a level of the queue Q_t. We will be using Fact 9 with some small constant ϵ' to be specified later.

The base of the algorithm is formed by Q_0, constructed exactly as in the previous section, and providing access to the suffix of the streamed text which is very close to the pattern. The queues on further levels cannot be constructed in the same manner, since we do not want to maintain in a buffer a suffix of T which is at a distance larger than r from P. Instead, we will maintain Q_t using Q_{t-1}, so there are k steps in the inductive structure. Q_1 could also be built directly, but not any of the further queues, so we choose not to distinguish Q_1. We construct queues with the following properties:

▶ **Lemma 18.** *Let $0 \le t < k$, and $q = (1+\epsilon')(1+8/r)$, for some ϵ' to be defined later depending on the desired final approximation factor. The following holds for Q_t:*

1. *Q_t stores $q^t(2^{t+1}-1)$-approximate blocks, up to $r \log m$ of them.*
2. *For any block in Q_t, which is an approximation of some text substring $T[i,j]$, the distance from $T[i,j]$ to P is in range $[r^t, 2r^t q^t(2^{t+1}-1)]$.*
3. *If the last processed index of T is j, for any i, if $\mathrm{ED}(P, T[i,j]) \le (r \log m) r^t$, then i is covered by Q_t.*

Proof. Let $A_t = q^t(2^{t+1}-1)$. We will proceed by induction on levels. As in Lemma 13, Q_t can be used to compute distance to any substring of P. For some long suffix of T we possibly can no longer compute *the closest* substring in P, but still can obtain an approximate closest substring, compounding the approximation factor for each consecutive level, by Proposition 17. This is described in Algorithm 3; Q_0 works as in the previous section.

For the ease of presentation, in the pseudocode each queue moves the current index j in T, while in fact this process should be synchronised between all queues processing the next letter from the stream, starting from level 0. The details of the algorithm in lines 5 or 9 are described below. A temporary block $tempB_t$ is a substring of P with the smallest approximate distance to the current suffix of T on level t; only in Q_0 we could afford direct access to this suffix. As previously, we create a new full block only when the (approximate) distance from P exceeds some threshold.

Algorithm 3 Computing a single block for queue Q_t, $t > 0$.

```
1:  function NEXT-BLOCK(P, T, i, 𝒬, t, r)
2:      Input: read-only pattern P, index i in streamed T, queues set 𝒬, parameter r.
3:      Output: Q_t is updated with a new block.

4:      j ← i + 1
5:      Compute approximate distance d' of T[i, j] from P using Q_{t−1},
6:      and store a temporary block tempB_t
7:      while d' < 2r^t A_t and i is covered by Q_{t−1} do          ▷ Still close to P
8:          j ← j + 1
9:          Compute approximate distance d' of T[i, j] from P using Q_{t−1}, store tempB_t
10:     if |Q_t| = r log m then
11:         Remove the first block from Q_t
12:     Let P[l_1, l_2] be the closest substring of T[i, j] according to approximation via Q_{t−1},
13:     and d' be the approximate distance between P[l_1, l_2] and T[i, j]
14:     Push (l_1, l_2, d') on Q_t
```

We need to inductively bound the approximation factor, as in the lemma statement. All properties hold for the base of $t = 0$, so assume they hold for t and consider Q_{t+1}. Q_{t+1} is constructing blocks using Q_t and its A_t-approximate blocks. Let the current suffix be $T[i,j]$, meaning the last block in Q_{t+1} ended at $i-1$ and the current index is j, and consider any substring P^* of the pattern.

The algorithm can approximate $\mathrm{ED}(P^*, T[i,j]) = d$ as follows, similarly to Lemma 13. Let $B[1,b]$ be a block in Q_t being approximation of $T[q_1,q_2]$ with $q_1 \le i$ and maximal q_1. Such a block exists since i is covered by Q_t, and recall that B is a substring of P. Set $T' = B(i-q_1, b]$, as B is an approximate block for a substring of T extending too far to the left. Then, append to T' all blocks following B from Q_t, and finally append $tempB_t$, the temporary block approximating the suffix which is still being processed by the queue

procedure in Q_t. Again, T' is not built explicitly, but as it is a concatenation of fragments of P, we can access any of its letters. Let D_l be the additive value for the l-th concatenated full approximate block B_l, and $D = \sum_{l=1}^{z} D_l$. Let $i_1, i_2, \ldots, i_{z+1}$ be the sequence of indices such that the l-th block in the Q_t after B is an approximation of $T[i_l, i_{l+1})$.

Recall $A_t = q^t(2^{t+1} - 1)$ is the approximation ratio of blocks in Q_t. By Proposition 6, P^* can be partitioned into $z + 2$ consecutive substrings $P_0, P_1, \ldots, P_{z+1}$ such that $d = \mathrm{ED}(P_0, T[i, i_1)) + \sum_{l=1}^{z} \mathrm{ED}(P_l, T[i_l, i_{l+1})) + \mathrm{ED}(P_{z+1}, T[i_{z+1}, j])$. From the second property of Q_t we have $\mathrm{ED}(T[i, i_1), B(i - q_1, b]) \leq 2r^t A_t$, thus by the triangle inequality for any string S it holds that:

$$\mathrm{ED}(S, T[i, i_1)) - 2r^t A_t \leq \mathrm{ED}(S, B(i - q_1, b]) \leq \mathrm{ED}(S, T[i, i_1)) + 2r^t A_t.$$

For any l, $\mathrm{ED}(P_l, T[i_l, i_{l+1})) \leq \mathrm{ED}(P_l, B_l) + D_l \leq A_t \mathrm{ED}(P_l, T[i_l, i_{l+1}))$, by the first property and the definition of approximate blocks. By Line 7 of Algorithm 3, we have that $\mathrm{ED}(tempB_t, T[i_{z+1}, j]) \leq 2r^t A_t$ is enforced directly, so again for any S:

$$\mathrm{ED}(S, tempB_t) - 2r^t A_t \leq \mathrm{ED}(S, T[i_{z+1}, j]) \leq \mathrm{ED}(S, tempB_t) + 2r^t A_t.$$

Since the algorithm have no access to $T[i, j]$, it estimates $\mathrm{ED}(P^*, T[i, j])$ with value of $\mathrm{ED}(P^*, T') + 4r^t A_t + D$. We can obtain the following upper bound:

$$\begin{aligned}
&\mathrm{ED}(P^*, T') + 4r^t A_t + D \leq \\
&\leq \mathrm{ED}(P_0, B(i - q_1, b]) + 4r^t A_t + \sum_{l=1}^{z} (\mathrm{ED}(P_l, B_l) + D_l) + \mathrm{ED}(P_{z+1}, tempB_t) \\
&\leq \mathrm{ED}(P_0, T[i, i_1)) + 8r^t A_t + \sum_{l=1}^{z} A_t \mathrm{ED}(P_l, T[i_l, i_{l+1})) + \mathrm{ED}(P_{z+1}, T[i_{z+1}, j]) \\
&\leq A_t \cdot d + 8r^t A_t,
\end{aligned}$$

where steps are analogous to those in Lemma 13. To get a lower bound, consider another partition of P^* into substrings $P'_0, \ldots, P'_{z+1}$ with $\mathrm{ED}(P^*, T') = \mathrm{ED}(P'_0, B(i - q_1, b]) + \sum_{l=1}^{z} \mathrm{ED}(P'_l, B_l) + \mathrm{ED}(P'_{z+1}, tempB_t)$. We then have:

$$\begin{aligned}
d &\leq \mathrm{ED}(P'_0, T[i, i_1)) + \sum_{l=1}^{z} \mathrm{ED}(P'_l, T[i_l, i_{l+1})) + \mathrm{ED}(P'_{z+1}, T[i_{z+1}, j]) \\
&\leq \mathrm{ED}(P'_0, B(i - q_1, b]) + 4r^t A_t + \sum_{l=1}^{z} (\mathrm{ED}(P'_l, B_l) + D_l) + \mathrm{ED}(P'_{z+1}, tempB_t) \\
&= \mathrm{ED}(P^*, T') + 4r^t A_t + D.
\end{aligned}$$

Therefore, $d \leq \mathrm{ED}(P^*, T') + 4r^t A_t + D \leq A_t d + 8r^t A_t$. Algorithm 3 uses $\mathrm{ED}(P^*, T') + 4r^t A_t + D$ as d', with $\mathrm{ED}(P^*, T')$ approximated using Fact 9, incurring additional $1 + \epsilon'$ error.

Consider the second property of the lemma. For $T[i, j]$ at distance at most r^{t+1} from P, by the third property i is covered by Q_t when j is the current index. Moreover, for any P^* with $d = \mathrm{ED}(P^*, T[i, j]) < r^{t+1}$, using T' as above we have:

$$(1 + \epsilon')\mathrm{ED}(P^*, T') + 4r^t A_t + D < (1 + \epsilon') A_t d + 9r^t A_t < r^{t+1} A_t (1 + \epsilon' + 9/r) < 2r^{t+1} A_t$$

assuming $r > 10$ and $\epsilon' < 1/10$. All of this ensures that the block in Q_{t+1} is not created until the distance from $T[i, j]$ to P is at least r^{t+1}.

For the first property, we can now assume that Q_{t+1} creates blocks only for suffixes at distance from P at least r^{t+1}. Consider P^* with $r^{t+1} \leq \mathrm{ED}(P^*, T[i,j]) = d$, and so we have $A_t d + 8r^t A_t \leq A_t d(1 + 8/r)$. We use Fact 9 to obtain a $(1+\epsilon')$-approximation of $\mathrm{ED}(P^*, T')$. Now if we take P^* with the minimal reported approximate distance to $T[i,j]$, since $2A_t(1+\epsilon')(1+8/r)+1 = 2q^t(2^{t+1}-1)(1+\epsilon')(1+8/r)+1 < q^{t+1}(2^{t+2}-1) = A_{t+1}$, by Proposition 17 we obtain A_{t+1}-approximate block, satisfying the first property.

Regarding the third property, the proof is analogous to the Proposition 12, as any $T[i,j]$ at distance at most $r^{t+2}\log m$ from P can be partitioned into $r\log m$ parts each at distance at most r^{t+1} from P, and by the second property blocks in Q_{t+1} approximate substrings at distances at least r^{t+1} from P. This finishes the proof of the lemma. ◀

Now, to obtain Theorem 2, we try all the queues and indices covered by them, then take the minimum reported edit distance. Consider $T[i,j]$ at distance d from P. Q_t will provide desired approximation for $d \in (r^t \log m, r^{t+1}\log m]$. Indeed, for values in this range Q_t covers i, thus we can construct T' and compute a value which is at most

$$\begin{aligned}(1+\epsilon')A_t d + 9r^t A_t &\leq A_t d(1+\epsilon'+9/\log m)\\ &< (2^k-1)(1+\epsilon')^k(1+8/r)^k d(1+\epsilon'+9/\log m).\end{aligned}$$

Let $\delta = \epsilon/2^k$. We can set $\epsilon' = k^{-1}\ln(1+\delta/4)$ so $(1+\epsilon')^k < 1+\delta/4$. For m large enough, $(1+8m^{-1/k})^k$ is smaller than $1+\delta/4$ for any constant ϵ. Similarly $(1+\epsilon'+9/\log m) < 1+\delta/4$, for m large enough and ϵ' as above. Overall, we will get $(2^k-1)(1+\delta) < (2^k-1+\epsilon)$-approximate solution.

As $r = m^{1/k}$, space complexity of the algorithm is $\mathcal{O}(m^{1/k}\log^2 m)$ bits with a constant number of queues, each storing $r\log m$ blocks, and time complexity for each letter read remains polynomial in m. Parameters in Fact 9 can be set so that it has space complexity $\tilde{\mathcal{O}}(m^{1/(k+1)}\epsilon^{-1})$, which is asymptotically smaller for any constant ϵ.

We note that the space complexity of Theorem 2 can be achieved for the Hamming distance, as the triangle inequality holds and we have a trivial small-space algorithm computing the distance of two read-only strings. This gives a solution much less efficient than Theorem 1, but deterministic.

References

1 Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003. doi:10.1016/S0022-0000(03)00025-4.

2 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999. doi:10.1006/JCSS.1997.1545.

3 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, pages 377–386. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.43.

4 Sudatta Bhattacharya and Michal Koucký. Streaming k-edit approximate pattern matching via string decomposition. In *ICALP*, volume 261 of *LIPIcs*, pages 22:1–22:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICALP.2023.22.

5 Kuan Cheng, Alireza Farhadi, MohammadTaghi Hajiaghayi, Zhengzhong Jin, Xin Li, Aviad Rubinstein, Saeed Seddighin, and Yu Zheng. Streaming and small space approximation algorithms for edit distance and Longest Common Subsequence. In *ICALP*, volume 198 of *LIPIcs*, pages 54:1–54:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.54.

6 Kuan Cheng, Zhengzhong Jin, Xin Li, and Yu Zheng. Space efficient deterministic approximation of string measures. *CoRR*, abs/2002.08498, 2020. arXiv:2002.08498.

7 Raphaël Clifford, Allyx Fontaine, Ely Porat, Benjamin Sach, and Tatiana Starikovskaya. The k-mismatch problem revisited. In *SODA*, pages 2039–2052. SIAM, 2016. doi:10.1137/1.9781611974331.CH142.

8 Raphaël Clifford, Tomasz Kociumaka, and Ely Porat. The streaming k-mismatch problem. In *SODA*, pages 1106–1125. SIAM, 2019. doi:10.1137/1.9781611975482.68.

9 Raphaël Clifford and Tatiana Starikovskaya. Approximate Hamming distance in a stream. In *ICALP*, volume 55 of *LIPIcs*, pages 20:1–20:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.20.

10 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31(6):1794–1813, 2002. doi:10.1137/S0097539701398363.

11 Alireza Farhadi, MohammadTaghi Hajiaghayi, Aviad Rubinstein, and Saeed Seddighin. Asymmetric streaming algorithms for edit distance and LCS. *CoRR*, abs/2002.11342, 2020.

12 Joan Feigenbaum, Sampath Kannan, Martin Strauss, and Mahesh Viswanathan. An approximate L1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32(1):131–151, 2002. doi:10.1137/S0097539799361701.

13 Shay Golan, Tomasz Kociumaka, Tsvi Kopelowitz, and Ely Porat. The streaming k-mismatch problem: Tradeoffs between space and total time. In *CPM*, volume 161 of *LIPIcs*, pages 15:1–15:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CPM.2020.15.

14 Shay Golan, Tsvi Kopelowitz, and Ely Porat. Towards optimal approximate streaming pattern matching by matching multiple patterns in multiple streams. In *ICALP*, volume 107 of *LIPIcs*, pages 65:1–65:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.65.

15 Tomasz Kociumaka, Ely Porat, and Tatiana Starikovskaya. Small-space and streaming pattern matching with k edits. In *FOCS*, pages 885–896. IEEE, 2021.

16 Xin Li and Yu Zheng. Lower bounds and improved algorithms for asymmetric streaming edit distance and Longest Common Subsequence. In *FSTTCS*, volume 213 of *LIPIcs*, pages 27:1–27:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.27.

17 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS*, pages 315–323. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.11.

18 Michael E. Saks and C. Seshadhri. Space efficient streaming algorithms for the distance to monotonicity and asymmetric edit distance. In *SODA*, pages 1698–1709. SIAM, 2013. doi:10.1137/1.9781611973105.122.

19 Tatiana Starikovskaya, Michal Svagerka, and Przemyslaw Uznanski. Lp pattern matching in a stream. In *APPROX-RANDOM*, volume 176 of *LIPIcs*, pages 35:1–35:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.APPROX/RANDOM.2020.35.