Approximate Online Pattern Matching in Sublinear Time

Diptarka Chakraborty
National University of Singapore, Singapore
diptarka@comp.nus.edu.sg

Debarati Das
University of Copenhagen, Denmark
debaratix710@gmail.com

Michal Koucký
Computer Science Institute of Charles University, Czech Republic
koucky@iuuk.mff.cuni.cz

Abstract

We consider the approximate pattern matching problem under edit distance. In this problem we are given a pattern P of length m and a text T of length n over some alphabet Σ, and a positive integer k. The goal is to find all the positions j in T such that there is a substring of T ending at j which has edit distance at most k from the pattern P. Recall, the edit distance between two strings is the minimum number of character insertions, deletions, and substitutions required to transform one string into the other. For a position t in $\{1, ..., n\}$, let k_t be the smallest edit distance between P and any substring of T ending at t. In this paper we give a constant factor approximation to the sequence $k_1, k_2, ..., k_n$. We consider both offline and online settings.

In the offline setting, where both P and T are available, we present an algorithm that for all t in $\{1, ..., n\}$, computes the value of k_t approximately within a constant factor. The worst case running time of our algorithm is $\tilde{O}(nm^{3/4})$.

In the online setting, we are given P and then T arrives one symbol at a time. We design an algorithm that upon arrival of the t-th symbol of T computes k_t approximately within $O(1)$-multiplicative factor and $m^{8/9}$-additive error. Our algorithm takes $\tilde{O}(m^{1-(7/54)})$ amortized time per symbol arrival and takes $\tilde{O}(m^{1-(1/54)})$ additional space apart from storing the pattern P. Both of our algorithms are randomized and produce correct answer with high probability. To the best of our knowledge this is the first algorithm that takes worst-case sublinear (in the length of the pattern) time and sublinear extra space for the online approximate pattern matching problem. To get our result we build on the technique of Chakraborty, Das, Goldenberg, Koucký and Saks [FOCS'18] for computing a constant factor approximation of edit distance in sub-quadratic time.

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

Keywords and phrases Approximate Pattern Matching, Online Pattern Matching, Edit Distance, Sublinear Algorithm, Streaming Algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2019.10

Related Version A full version of the paper is available at `https://arxiv.org/abs/1810.03664`.

Funding The research leading to these results is partially supported by the Grant Agency of the Czech Republic under the grant agreement no. 19-27871X and by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement no. 616787.

Acknowledgements Authors would like to thank anonymous reviewers for many helpful suggestions and comments on an earlier version of this paper.

39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019).
Editors: Arkadev Chattopadhyay and Paul Gastin; Article No. 10; pp. 10:1–10:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Finding the occurrences of a pattern in a larger text is one of the fundamental problems in computer science. Due to its immense applications this problem has been studied extensively under several variations [25, 21, 5, 17, 23, 18, 24, 31, 26]. One of the most natural variations is where we are allowed to have a small number of errors while matching the pattern. This problem of pattern matching while allowing errors is known as *approximate pattern matching*. The kind of possible errors varies with the applications. Generally we capture the amount of errors by the metric defined over the set of strings. One common and widely used distance measure is the edit distance (aka *Levenshtein distance*) [28]. The edit distance between two strings T and P denoted by $d_{\text{edit}}(T, P)$ is the minimum number of character insertions, deletions, and substitutions required to transform one string into the other. In this paper we focus on the approximate pattern matching problem under edit distance. This problem has various applications ranging from computational biology, signal transmission, web searching, text processing to many more.

Given a pattern P of length m and a text T of length n over some alphabet Σ, and an integer k we want to identify all the substrings of T at edit distance at most k from P. As the number of such substrings might be quadratic in n and one wants to obtain efficient algorithms, one focuses on finding the set of all right-end positions in T of those substrings at distance at most k. More specifically, for a position t in T, we let k_t be the smallest edit distance of a substring of T ending at t-th position in T. (We number positions in T and P from 1.) The goal is to compute the sequence $k_1, k_2, \ldots, k_n$ for P and T. Using basic dynamic programming paradigm we can solve this problem in $O(nm)$ time [33]. Later Masek and Paterson [29] shaved a $\log n$ factor from the above running time bound. Despite of a long line of research, this running time remains the best till now. Recently, Backurs and Indyk [8] indicate that this $O(nm)$ bound cannot be improved significantly unless the Strong Exponential Time Hypothesis (SETH) is false. Moreover Abboud et al. [3] showed that even shaving an arbitrarily large polylog factor would imply that NEXP does not have non-uniform NC^1 circuits which is likely but hard to prove conclusion. More hardness results can be found in [2, 9, 1, 4].

In this paper we focus on finding an approximation to the sequence $k_1, k_2, \ldots, k_n$ for P and T. For reals $c, a \geq 0$, a sequence $\tilde{k}_1, \ldots, \tilde{k}_n$ is (c, a)-approximation to $k_1, \ldots, k_n$, if for each $t \in \{1, \ldots, n\}$, $k_t \leq \tilde{k}_t \leq c \cdot k_t + a$. Hence, c is the multiplicative error and a is the additive error of the approximation. An algorithm computes (c, a)-approximation to approximate pattern matching if it outputs a (c, a)-approximation of the true sequence $k_1, k_1, \ldots, k_n$ for P and T. We refer to $(c, 0)$-approximation simply as c-approximation. Our main theorem is the following.

▶ **Theorem 1.** *There is a constant $c \geq 1$ and there is a randomized algorithm that computes a c-approximation to approximate pattern matching in time $\tilde{O}(n \cdot m^{3/4})$ with probability at least $(1 - 1/n^3)$.*

In the recent past researchers also studied the approximate pattern matching problem in the online setting. The online version of this pattern matching problem mostly arises in real life applications that require matching pattern in a massive data set, like in telecommunications, monitoring Internet traffic, building firewall to block viruses and malware connections and many more. The online approximate pattern matching is as follows: we are given a pattern P first, and then the text T is coming symbol by symbol. Upon receipt of the t-th symbol we should output the corresponding k_t. The online algorithm runs in *amortized time* $O(\ell)$ if it runs in total time $O(n \cdot \ell)$. We also say that the online algorithm uses *extra space* $O(s)$ if in addition to storing the pattern P it uses at most $O(s)$ cells of memory at any time.

▶ **Theorem 2.** *There is a constant $c \geq 1$ and there is a randomized online algorithm that computes $(c, m^{8/9})$-approximation to approximate pattern matching in amortized time $\tilde{O}(m^{1-(7/54)})$ and extra space $\tilde{O}(m^{1-(1/54)})$ with probability at least $1 - 1/poly(m)$.*

To the best of our knowledge this is the first online approximation algorithm that takes sublinear (in the length of the pattern) running time and sublinear extra space for the approximate pattern matching problem. Designing algorithm that uses small extra space is quite natural from the practical point of view and has been considered for many problems including pattern matching, e.g. [32, 22].

To prove our result we use the technique developed by Chakraborty, Das, Goldenberg, Koucký and Saks in [10, 11], where they provide a sub-quadratic time constant factor approximation algorithm for the edit distance problem. In particular, in [11] authors describe a constant factor approximation algorithm that given two strings of length n runs in time $\tilde{O}(n^{12/7})$. Now suppose we only have a black-box access to that approximation algorithm for computing the edit distance. Then we claim that we get $O(1)$-approximation to the offline approximate pattern matching problem in time $\tilde{O}(nm^{6/7})$. Let us first set a parameter $k = m^{6/7}$. Now in the first phase we run $O(nk)$ time algorithm by Landau and Vishkin [27] and get all the values of k_t which are at most k. In the next phase we divide the text T into overlapping substrings of length m with overlap of $m - k$. In other words for every t that are multiple of k consider the substring $T_{t-m+1,t}$ (that starts at $(t - m + 1)$-th symbol and ends at t-th symbol). For all the positions t that are multiple of k and $k_t > k$ (as identified by the first phase) we use the edit distance algorithm of [11] to get $O(1)$-approximation of $d_{\text{edit}}(T_{t-m+1,t}, P)$ and output that value as $\tilde{k}_t$. Since $k_t \leq d_{\text{edit}}(T_{t-m+1,t}, P) \leq 2k_t$, the output $\tilde{k}_t$ is an $O(1)$-approximation of k_t. For all the remaining values of t (that are not multiples of k, and $k_t > k$) we output $\tilde{k}_{t'} + (t - t')$ where $t' = \lfloor \frac{t}{k} \rfloor \cdot k$, as an estimate of k_t. Since $k_{t'} - (t - t') \leq k_t \leq k_{t'} + (t - t')$, for all t such that $k_t > k$ we get $O(1)$-approximation of k_t. Note, the above described process takes $\tilde{O}(nm^{6/7})$ time and thus breaks $O(nm)$ barrier for the offline approximate pattern matching problem for constant factor approximation. However our claimed running time in Theorem 1 is better than that of this black-box algorithm.

In this paper we first design an offline algorithm by building upon the technique used in [11]. To do this we exploit the similarity between the "dynamic programming graphs" (see Section 2) for approximate pattern matching problem and the edit distance problem. To get $\tilde{O}(nm^{3/4})$ time algorithm for the offline approximate pattern matching problem still requires careful modifications to the edit distance algorithm. However the scenario becomes much more involved if one wants to design an online algorithm using only a small amount of extra space. The approximation algorithm for edit distance in [11] works in two phases: first a covering algorithm is used to discover a suitable set of shortcuts in the pattern matching graph, and then a min-cost path algorithm on a grid graph with the shortcuts yields the desired result. In the online setting we carefully interleave all of the above phases. However that by itself is not sufficient since the first phase, i.e., the covering algorithm used in [11] essentially relies on the fact that both of the strings are available at any point of time. We modify the covering technique so that it can also be implemented in the situation when we cannot see the full text. We show that if we store the pattern P then we need only $O(m^{1-\gamma})$ extra space (for some small constant $\gamma > 0$) to perform the sampling. Furthermore, the min-cost path algorithm in [11] takes $O(m)$ space. We modify that algorithm too in a way so that it also works using only $O(m^{1-\gamma})$ space (for some small constant $\gamma > 0$). We describe our algorithm in more details in Section 5.

1.1 Related work

The approximate pattern matching problem is one of the most extensively studied problems in modern computer science due to its direct applicability to data driven applications. In contrast to the exact pattern matching here a text location has a match if the distance between the pattern and the text is within some tolerated limit. In our work we study the approximate pattern matching under edit distance metric. The very first $O(nm)$-time algorithm was given by Sellers [33] in 1980. Masek and Paterson [29] proposed an $O(nm/\log n)$-time $O(n)$-space algorithm using Four Russians [7] technique. Later [30, 27, 20] gave $O(nk)$-time algorithms where k is the upper limit of allowed edit operations. All of these algorithms use either $O(m^2)$ or $O(n)$ space. However [19, 36] note that achieving $O(m)$ space is also possible while maintaining the running time. A faster (for small values of k) algorithm was given by Cole and Hariharan [16], which has running time $O(n(1 + k^4/m))$. We refer the interested readers to a beautiful survey by Navarro [31] for a comprehensive treatment on this topic. We have already seen in the previous section that any c-approximation algorithm for the edit distance problem can be transformed into an $O(c)$-approximation algorithm for the approximate pattern matching problem. We get $(\log m)^{O(1/\epsilon)}$-approximation to the approximate pattern matching problem in time $O(nm^{\frac{1}{2}+\epsilon})$ (for every $\epsilon > 0$) from the edit distance algorithm of Andoni *et al.* [6]. To achieve the same running time while having only constant factor approximation is an important open problem.

All the above mentioned algorithms assume that the entire text is available from the very beginning of the process. However in the online version, the pattern is given at the beginning and the text arrives in a stream, one symbol at a time. Clifford *et al.* [12] gave a "black-box algorithm" for online approximate matching where the supported distance metrics are Hamming distance, matching with wildcards, L_1 and L_2 norm. Their algorithm has running time $O(\sum_{j=1}^{\log_2 m} T(n, 2^{j-1})/n)$ per symbol arrival, where $T(n, m)$ is the running time of the best offline algorithm. This result was extended in [14] by introducing an algorithm solving online approximate pattern matching under edit distance metric in time $O(k \log m)$ per symbol arrival. This algorithm uses $O(m)$-space. In [15] the running time was further improved to $O(k)$ per symbol. However none of these algorithms for edit distance metric is black-box and they highly depend on the specific structure of the corresponding offline algorithm. Recently, Starikovskaya [34] gave a randomized algorithm which has a worst case time complexity of $O((k^2\sqrt{m} + k^{13}) \log^4 m)$ and uses space $O(k^8\sqrt{m} \log^6 m)$. Although her algorithm takes both sublinear time and sublinear space for small values of k, heavy dependency on k in the complexity terms makes it much worse than the previously known algorithms in the high regime of k. On the lower bound side, Clifford, Jalsenius and Sach [13] showed in the *cell-probe model* that expected amortized running time of any randomized algorithm solving online approximate pattern matching problem must be $\Omega(\sqrt{\log m}/(\log \log m)^{3/2})$ per output.

2 Preliminaries

We recall some basic definitions of [11]. Consider the text T of length n to be aligned along the horizontal axis and the pattern P of length m to be aligned along the vertical axis. For $i \in \{1, \ldots, n\}$, T_i denotes the i-th symbol of T and for $j \in \{1, \ldots, m\}$, P_j denotes the j-th symbol of P. $T_{s,t}$ is the substring of T starting by the s-th symbol and ending by the t-th symbol of T. For any interval $I \subseteq \{0, \ldots, n\}$, T_I denotes the substring of T indexed by $I \setminus \{\min(I)\}$ and for $J \subseteq \{0, \ldots, m\}$, P_J denotes the substring of P indexed by $J \setminus \{\min(J)\}$.

Edit distance and pattern matching graphs

For a text T of length n and a pattern P of length m, the *edit distance graph* $G_{T,P}$ is a directed weighted graph called a grid graph with vertex set $\{0,\cdots,n\} \times \{0,\cdots,m\}$ and following three types of edges: $(i-1,j) \to (i,j)$ (H-steps), $(i,j-1) \to (i,j)$ (V-steps) and $(i-1,j-1) \to (i,j)$ (D-steps). Each H-step or V-step has cost 1 and each D-step costs 0 if $T_i = P_j$ and 1 otherwise. The *pattern matching graph* $\tilde{G}_{T,P}$ is the same as the edit distance graph $G_{T,P}$ except for the cost of horizontal edges $(i,0) \to (i+1,0)$ which is zero.

For intervals $I \subseteq \{0,\ldots,n\}$ and $J \subseteq \{0,\ldots,m\}$, $G_{T,P}(I \times J)$ is the subgraph of $G_{T,P}$ induced on $I \times J$. Clearly, $G_{T,P}(I \times J) \cong G_{T_I,P_J}$. We define the cost of a path τ in G_{T_I,P_J}, denoted by $\mathrm{cost}_{G_{T_I,P_J}}(\tau)$, as the sum of the costs of its edges. We also define the cost of a graph G_{T_I,P_J}, denoted by $\mathrm{cost}(G_{T_I,P_J})$, as the cost of the cheapest path from $(\min I, \min J)$ to $(\max I, \max J)$.

The following is well known in the literature (e.g. see [33]).

▶ **Proposition 3.** *Consider a pattern P of length m and a text T of length n, and let $G = \tilde{G}_{T,P}$. For any $t \in \{1,\ldots,n\}$, let $I = \{0,\cdots,t\}$, and $J = \{0,\cdots,m\}$. Then $k_t = cost(G(I \times J)) = \min_{i \le t} d_{edit}(T_{i,t}, P)$.*

A similar proposition is also true for the edit distance graph.

▶ **Proposition 4.** *Consider a pattern P of length m and a text T of length n, and let $G = G_{T,P}$. For any $i_1 \le i_2 \in \{1,\cdots,n\}$, $j_1 \le j_2 \in \{1,\cdots,m\}$ let $I = \{i_1-1,\cdots,i_2\}$ and $J = \{j_1-1,\cdots,j_2\}$. Then $cost(G(I \times J)) = d_{edit}(T_{i_1,i_2}, P_{j_1,j_2})$.*

Let G be a grid graph on $I \times J$ and $\tau = (i_1,j_1),\ldots,(i_l,j_l)$ be a path in G. *Horizontal projection* of a path τ is the set $\{i_1,\ldots,i_l\}$. Let I' be an interval contained in the horizontal projection of τ, then $\tau_{I'}$ denotes the (unique) minimal subpath of τ with horizontal projection I'. Let $G' = G(I' \times J')$ be a subgraph of G. For $\delta \ge 0$ we say that $I' \times J'$ $(1-\delta)$*-covers* the path τ if the initial and the final vertex of $\tau_{I'}$ are at a vertical distance of at most $\delta(|I'|-1)$ from $(\min(I'), \min(J'))$ and $(\max(I'), \max(J'))$, resp..

A *certified box* of G is a pair $(I' \times J', \ell)$ where $I' \subseteq I$, $J' \subseteq J$ are intervals, and $\ell \in \mathbb{N} \cup \{0\}$ such that $\mathrm{cost}(G(I' \times J')) \le \ell$. At high level, our goal is to approximate each path τ in G by a path via the corner vertices of certified boxes. For that we want that a substantial portion of the path τ goes via those boxes and that the sum of the costs of the certified boxes is not much larger than the actual cost of the path. The next definition makes our requirements precise. Let $\sigma = \{(I_1 \times J_1, \ell_1), (I_2 \times J_2, \ell_2), \ldots, (I_s \times J_s, \ell_s)\}$ be a sequence of certified boxes in G. Let τ be a path in $G(I \times J)$ with horizontal projection I. For any $k, \zeta \ge 0$, we say that σ (k,ζ)*-approximates* τ if the following three conditions hold:

1. $I_1,\ldots,I_s$ is a decomposition of I, i.e., $I = \bigcup_{i \in [s]} I_i$, and for all $i \in [s-1]$, $\min(I_{i+1}) = \max(I_i)$.
2. For each $i \in [s]$, $I_i \times J_i$ $(1 - \ell_i/(|I_i|-1))$-covers τ.
3. $\sum_{i \in [s]} \ell_i \le k \cdot \mathrm{cost}(\tau) + \zeta$.

3 Offline approximate pattern matching

3.1 Technical Overview

To prove Theorem 1 we design an algorithm as follows. For $k = 2^j$, $j = 0,\ldots,\log m^{3/4}$, we run the standard $O(nk)$ algorithm by Landau and Vishkin [27] to identify all t such that $k_t \le k$. To identify positions with $k_t \le k$ for $k > m^{3/4}$ where k is a power of two we will use the technique of [11] to compute $(O(1), O(m^{3/4}))$-approximation of $k_1, \dots, k_n$. The obtained information can be combined in a straightforward manner to get a single $O(1)$-approximation to $k_1, \dots, k_n$: For each t, if for some $2^j \leq m^{3/4}$, k_t is at most 2^j (as determined by the former algorithm) then output the exact value of k_t using the algorithm of [27], otherwise output the approximation of k_t found by the latter algorithm. This way, for $k_t \leq m^{3/4}$ we will get the exact value, and for $k > m^{3/4}$ we will get an $O(1)$-approximation. We will now elaborate on the latter algorithm based on [11]. The edit distance algorithm of [11] has two phases which we will also use. The first phase (*covering phase*) identifies a set of *certified boxes*, subgraphs of the pattern matching graph with good upper bounds on their cost. These certified boxes should cover the min-cost paths of interest. Then the next phase runs a min-cost path algorithm on these boxes to obtain the output sequence. We will show that both of these phases take $\widetilde{O}(nm^{3/4})$ time and so the overall running time will be $\widetilde{O}(nm^{3/4})$.

We next describe the two phases of the algorithm. The algorithm will use the following parameters: $w_1 = m^{1/4}$, $w_2 = m^{1/2}$, $d = m^{1/4}$, $\theta = m^{-1/4}$. The meaning of the parameters is essentially the same as in [11] though their setting is different. Let $c_0, c_1 \geq 0$ be the large enough constants from [11]. For simplicity we will assume without loss of generality that w_1 and w_2 are powers of two (by rounding them down to the nearest powers of two), θ is a reciprocal of a power of two (by decreasing θ by at most a factor of two), $w_2 | m$ (by chopping off a small suffix from P which will affect the approximation by a negligible additive error as $m^{3/4} \gg w_2$), and $m|n$ (if not we can run the algorithm twice: on the largest prefix of T of length divisible by m and then on the largest suffix of T of length divisible by m). The algorithm will not explicitly compute k_t for all t but only for t where t is a multiple of w_2, and then it will use the same value for each block of w_2 consecutive k_t's. Again, this will affect the approximation by a negligible additive error.

3.2 Covering phase

We describe the first phase of the algorithm now. First, we partition the text T into substrings $T^0_1, \dots, T^0_{n_0}$ of length m, where $n_0 = n/m$. Then we process each of the parts independently. Let T' be one of the parts. We partition T' into substrings $T^1_1, T^1_2, \dots, T^1_{n_1}$ of length w_1, and we also partition T' into substrings $T^2_1, T^2_2, \dots, T^2_{n_2}$ of length w_2, where $n_1 = m/w_1$ and $n_2 = m/w_2$. For a substring u of v starting by i-th symbol of v and ending by j-th symbol of v, we let $\{i-1, i, i+1, \dots, j\}$ be its *span.* Moreover for $\delta \in (0,1)$ we call u to be (δ)-aligned if both $i-1$ and $j-1$ are divisible by $\delta(j-i)$. The covering algorithm proceeds in phases $j = 0, \dots, \lceil \log 1/\theta \rceil$ associated with $\epsilon_j = 2^{-j}$. Similar to the edit distance algorithm, here also each phase has two parts, namely the *dense substrings* and the *extension sampling.* Then the covering algorithm proceeds as follows:

Dense substrings

In this part the algorithm aims to identify for each ϵ_j, a set of substrings T^1_i that are similar (i.e., up to "small" edit distance) to more than d $(\epsilon_j/8)$-aligned, w_1 length substrings of P. We identify each T^1_i by testing a random sample of relevant substrings of P. If we determine with high confidence that there are at least $\Omega(d)$ substrings of P similar to T^1_i, we add T^1_i into a set D_j of such strings, and we also identify all $T^1_{i'}$ that are similar to T^1_i. By triangle inequality we would also expect them to be similar to many relevant substrings of P. So we add these $T^1_{i'}$ to D_j as well as we will not need to process them anymore. We output the set of certified boxes of edit distance $O(\epsilon_j w_1)$ found this way. More formally:

For $j = \lceil \log 1/\theta \rceil, \dots, 0$, the algorithm maintains sets D_j of substrings T^1_i. These sets are initially empty.

Step 1. For each $i = 1, \dots, n_1$ and $j = \lceil \log 1/\theta \rceil, \dots, 0$, if T^1_i is in D_j then we continue with the next i and j. Otherwise we process it as follows.

Step 2. Set $\epsilon_j = 2^{-j}$. Independently at random, sample $8c_0 \cdot m \cdot (\epsilon_j w_1 d)^{-1} \cdot \log n$ many $(\epsilon_j/8)$-aligned substrings of P of length w_1. For each sampled substring u check if its edit distance from T^1_i is at most $\epsilon_j w_1$. If less than $\frac{1}{2} \cdot c_0 \cdot \log n$ of the samples have their edit distance from T^1_i below $\epsilon_j w_1$ then we are done with processing this i and j and we continue with the next pair.

Step 3. Otherwise we identify all substrings $T^1_{i'}$ that are not in D_j and are at edit distance at most $2\epsilon_j w_1$ from T^1_i, and we let X to be the set of their spans relative to the whole T.

Step 4. Then we identify all $(\epsilon_j/8)$-aligned substrings of P of length w_1 that are at edit distance at most $3\epsilon_j w_1$ from T^1_i, and we let Y to be the set of their spans. (We also allow some $(\epsilon_j/8)$-aligned substrings of P of edit distance at most $6\epsilon_j w_1$ to be included in the set Y as some might be misidentified to have the smaller edit distance from T^1_i by our procedure that searches for them, see further.)

Step 5. For each pair of spans (I, J) from $X \times Y$ we output corresponding certified box $(I \times J, 8\epsilon_j w_1)$. We add substrings corresponding to X into D_j and continue with the next pair i and j.

Once we process all pairs of i and j, we proceed to the next phase: *extension sampling*.

Extension sampling

In this part for every $\epsilon_j = 2^{-j}$ and every substring T^2_i, which does not have all its substrings T^1_ℓ contained in D_j we randomly sample a set of such T^1_ℓ's. For each sampled T^1_ℓ we determine all relevant substrings of P at edit distance at most $\epsilon_j w_1$ from T^1_ℓ. There should be $O(d)$-many such substrings of P. We extend each such substring into a substring of size $|T^2_i|$ within P and we check the edit distance of the extended string from T^2_i. For each extended substring of edit distance at most $3\epsilon_j w_2$ we output a set of certified boxes.

Here we define the appropriate extension of substrings. Let u be a substring of T of length less than $|P|$, and let v be a substring of u starting by the i-th symbol of u. Let v' be a substring of P of the same length as v starting by the j-th symbol of P. The *diagonal extension u' of v' in P with respect to u and v*, is the substring of P of length $|u|$ starting at position $j - i$. If $(j - i) \leq 0$ then the extension u' is the prefix of P of length $|u|$, and if $j - i + |P| > |P|$ then the extension u' is the suffix of P of length $|u|$.

Step 6. Process all pairs $i = 1, \dots, n_2$ and $j = \lceil \log 1/\theta \rceil, \dots, 0$.

Step 7. Independently at random, sample $c_1 \cdot \log^2 n \cdot \log m$ substrings T^1_ℓ that are part of T^2_i and that are not in D_j. (If there is no such substring continue for the next pair of i and j.)

Step 8. For each T^1_ℓ, find all $(\epsilon_j/8)$-aligned substrings v' of P of length w_1 that are at edit distance at most $\epsilon_j w_1$ from T^1_ℓ.

Step 9. For each v' determine its diagonal extension u' with respect to T^2_i and T^1_ℓ. Check if the edit distance of u' and T^2_i is less than $3\epsilon_j w_2$. If so, compute it and denote the distance by c. Let I' be the span of T^2_i relative to T, and J' be the span of u' in P. For all powers a and b of two, $m^{3/4} \leq a \leq b \leq m$, output the certified box $(I' \times J', c + a + b)$. Proceed for the next i and j.

This ends the covering algorithm which outputs various certified boxes.

To implement the above algorithm we will use Ukkonen's [35] $O(nk)$-time algorithm to check whether the edit distance of two strings of length w_1 is at most $\epsilon_j w_1$ in time $O(w_1^2 \epsilon_j)$. Given the edit distance is within this threshold the algorithm can also output its precise value. We use this algorithm in Step 3. To identify all substrings of length w_1 at edit distance at most $\epsilon_j w_1$ of S from a given string R (where S is the pattern P of length m and R is one of the T^1_i of length w_1), in Step 4, we use the $O(nk)$-time pattern matching algorithm of Landau and Vishkin [27]. For a given threshold k, this algorithm determines for each position t in S, whether there is a substring of edit distance at most k from R ending at that position in S. If the algorithm reports such a position t then we know by the following proposition that the substring $S_{t-|R|+1,t}$ is at edit distance at most $2k$. At the same time we are guaranteed to identify all the substrings of S of length w_1 at edit distance at most k from R. Hence in Step 4, finding all the substrings at distance $3\epsilon_j w_1$ with perhaps some extra substrings of edit distance at most $6\epsilon_j w_1$ can be done in time $O(m w_1 \epsilon_j)$.

▶ **Proposition 5.** *For strings S and R, and integers $t \in \{1, \ldots, |S|\}$, $k \geq 0$, if $\min_{i \leq t} d_{edit}(S_{i,t}, R) \leq k$ then $d_{edit}(S_{t-|R|+1,t}, R) \leq 2k$.*

Proof. Let $S_{i,t}$ be the best match for R ending by the t-th symbol of S. Hence, $k = d_{\text{edit}}(S_{i,t}, R)$. If $S_{i,t}$ is by ℓ symbols longer than R then $k \geq \ell$ and $d_{\text{edit}}(S_{t-|R|+1,t}, R) \leq k + \ell \leq 2k$ by the triangle inequality. The same is true if $S_{i,t}$ is shorter by ℓ symbols. ◀

3.3 Correctness of the covering algorithm

▶ **Lemma 6.** *Let $t \geq 1$ be such that t is a multiple of w_2. Let τ_t be the min-cost path between vertex $(t - m, 0)$ and (t, m) in the edit distance graph $G = G_{T,P}$ of T and P of cost at least $m^{3/4} \geq \theta m$. The covering algorithm outputs a set of weighted boxes $\mathcal{R}$ such that every $(I \times J, \ell) \in \mathcal{R}$ is correctly certified i.e., cost$(G(I \times J)) \leq \ell$ and there is a subset of $\mathcal{R}$ that $(O(1), O(k_t))$-approximates τ_t with probability at least $1 - 1/n^7$.*

It is clear from the description of the covering algorithm that it outputs only correct certified boxes from the edit distance graph of T and P, that is for each box $(I \times J, \ell)$, $\text{cost}(G(I \times J)) \leq \ell$.

The cost of τ_t corresponds to the edit distance between P and $T_{t-m+1,t}$ and it is bounded by $2k_t$ by Proposition 5. Let k'_t be the smallest power of two $\geq k_t$. We claim that by essentially the same argument as in Proposition 3.8 and Theorem 3.9 of [11] the algorithm outputs with high probability a set of certified boxes that $(O(1), O(k'_t))$-approximates τ_t. Therefore instead of repeating the whole proof, here we sketch the differences between the current covering algorithm with that of [11] and argue about how to handle them.

The main substantial difference is that the algorithm in [11] searches for certified boxes located only within $O(k_t)$ diagonals along the main diagonal of the edit distance graph. (This rests on the observation of Ukkonen [35] that a path of cost $\leq k_t$ must pass only through vertices on those diagonals.) Here we process certified boxes in the whole matrix as each t requires a different "main" diagonal. Except for this difference and the order of processing various pieces the algorithms are the same.

The discovery of certified boxes depends on the number (*density*) of relevant substrings of P similar to a given T^1_i. In the edit distance algorithm in [11] this density is measured only in the $O(k_t)$-width strip along the main diagonal of the edit distance graphs whereas here it is measured within the whole P. (So the actual classification of substrings T^1_i on *dense* (in D_j) and *sparse* (not in D_j) might differ between the two algorithms.) Hence, one could think (though technically not quite correct) that the certified boxes output by the current algorithm form a superset of boxes output by the edit distance algorithm of [11]. However, this difference is immaterial for the correctness argument in Theorem 3.9 of [11].

Another difference is that in Step 4 we use $O(mw_1\epsilon_j)$-time algorithm to search for all the similar substrings. This algorithm will report all the substrings we were looking for and additionally it might report some substrings of up to twice the required edit distance. This necessitates the upper bound $8\epsilon_j w_1$ in certified boxes in Step 5. It also means a loss of factor of at most two in the approximation guarantee as the boxes of interest are reported with the cost $8\epsilon_j w_1$ instead of the more accurate $5\epsilon_j w_1$ of the original algorithm in [11] which would give a $(45, 15\text{cost}(\tau_t))$-approximation. (In that theorem θm represents an (arbitrary) upper bound on the cost of τ_t provided it satisfies certain technical conditions requiring that θ is large enough relative to m. This is satisfied by requiring that $\text{cost}(\tau_t) \geq m^{3/4} \geq \theta m$.)

Another technical difference is that the path τ_t might pass through two edit distance graphs $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$, where $t \in [(\ell-1)m+1, \ell m]$. This means that one needs to argue separately about restriction of τ_t to $G_{T^0_{\ell-1},P}$ and $G_{T^0_\ell,P}$. However, the proof of Theorem 3.9 in [11] analyses approximation of the path in separate parts restricted to substrings of T of size w_2. As both t and m are multiples of w_2, the argument for each piece applies in our setting as well.

3.4 Time complexity of the covering algorithm

By analyzing the running time we get the following.

▶ **Lemma 7.** *The covering algorithm runs in time $\widetilde{O}(nm^{3/4})$ with probability at least $1-1/n^8$.*

We analyse the running time of the covering algorithm for each $T' = T^0_i$ separately. We claim that the running time on T' is $\widetilde{O}(m^{7/4})$ so the total running time is $\widetilde{O}((n/m)m^{7/4}) = \widetilde{O}(nm^{3/4})$.

In Step 1, for every $i = 1, \dots, n_1$ and $j = 0, \dots, \log m^{1/4}$, we might sample $O(\frac{m}{\epsilon_j w_1 d} \cdot \log n)$ substrings of P of length w_1 and check whether their edit distance from T^1_i is at most $\epsilon_j w_1$. This takes time at most $\widetilde{O}(\frac{m}{\epsilon_j w_1 d} \cdot \frac{m}{w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(m^2/d) = \widetilde{O}(m^{7/4})$ in total.

We say that a bad event happens either if some substring T^1_i has more than d relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample less than $\frac{1}{2} \cdot c_0 \log n$ of them, or if some substring T^1_i has less than $d/4$ relevant substrings of P having distance at most $\epsilon_j w_1$ but we sample more than $\frac{1}{2} \cdot c_0 \log n$ of them. By Chernoff bound, the probability of a bad event happening during the whole run of the covering algorithm is bounded by $\exp(-O(\log n)) \leq 1/n^8$, for sufficiently large constant c_0. Assuming no bad event happens we analyze the running time of the algorithm further.

Each substring T^1_i that reaches Step 3 can be associated with a set of its relevant substrings in P of edit distance at most $\epsilon_j w_1$ from it. The number of these substrings is at least $d/4$ many. These substrings must be different for different strings T^1_i that reach Step 3 as if they were not distinct then the two substrings T^1_i and $T^1_{i'}$ would be at edit distance at most $2\epsilon_j w_1$ from each other and one of them would be put into D_j in Step 5 while processing the other one so it could not reach Step 3. Hence, we can reach Steps 3–5 for at most $\frac{8m}{\epsilon_j w_1} \cdot \frac{4}{d}$ strings T^1_i. For a given j and each T^1_i that reaches Step 3, the execution of Steps 3 and 4 takes $O(mw_1\epsilon_j)$ time, hence we will spend in them $\widetilde{O}(m^2/d) = \widetilde{O}(m^{7/4})$ time in total.

Step 5 can report for each j at most $\frac{8m}{\epsilon_j w_1} \cdot \frac{m}{w_1}$ certified boxes, so the total time spent in this step is $\widetilde{O}(m^2/w_1) = \widetilde{O}(m^{7/4})$ as $\epsilon_j w_1 \geq 1/4$.

Step 7 takes order less time than Step 8. In Step 8 we use Ukkonen's [35] $O(nk)$-time edit distance algorithm to check the distance of strings of length w_1. We need to check $\widetilde{O}(n_2 \cdot \frac{m}{\epsilon_j w_1})$ pairs for the total cost $\widetilde{O}(\frac{m}{w_2} \cdot \frac{m}{\epsilon_j w_1} \cdot w_1^2 \epsilon_j) = \widetilde{O}(m^{7/4})$ per j.

As no bad event happens, for each T^1_ℓ, there will be at most $d/4$ strings v' processed in Step 9. We will spend $O(w_2^2 \epsilon_j)$ time on each of them to check for edit distance and $O(\log^2 n)$ to output the certified boxes. Hence, for each j we will spend here $\widetilde{O}(\frac{m}{w_2} \cdot d w_2^2 \epsilon_j)$ time, which is $\widetilde{O}(m w_2 d)$ in total.

Thus, the total time spent by the algorithm in each of the steps is $\widetilde{O}(m^{7/4})$ as required.

4 Min-cost Path in a Grid Graph with Shortcuts

In this section we explain how we use certified boxes to calculate the approximation of k_t's. Consider any grid graph G. A *shortcut* in G is an additional edge $(i,j) \to (i',j')$ with cost ℓ, where $i < i'$ and $j < j'$.

Let $G_{T,P}$ be the edit distance graph for T and P. Let $(I \times J, \ell)$ be a certified box in $G_{T,P}$ with $|I| = |J|$. If $\ell < 1/2(|I|-1)$ add a shortcut edge $e_{I,J}$ from vertex $(\min I, \min J + \ell)$ to vertex $(\max I, \max J - \ell)$ with cost 3ℓ. Do this for all certified boxes output by the covering algorithm to obtain a graph $G'_{T,P}$. Note, if $\ell \geq 1/2(|I|-1)$ we do not add any shortcut edge for the corresponding certified box. Next remove all the diagonal edges (D-steps) of cost 0 or 1 from graph $G'_{T,P}$ and obtain graph $G''_{T,P}$.

▶ **Proposition 8.** *If τ is a path from $(t-m, 0)$ to (t,m) in $G_{T,P}$ which is (k,ζ)-approximated by a subset of certified boxes σ by the covering algorithm then there is a path from $(t-m,0)$ to (t,m) in $G''_{T,P}$ of cost at most $5 \cdot (k \cdot cost_{G_{T,P}}(\tau) + \zeta)$ consisting of shortcut edges corresponding to σ and H and V steps.*

We provide the proof of proposition 8 in the full version. By Lemma 6 and Proposition 8, for t where $w_2 | t$, the cost of a shortest path from $(t-m, 0)$ to (t,m) in $G''_{T,P}$ is bounded by $O(k_t)$. At the same time, any path in $G''_{T,P}$ from $(i,0)$ to (t,m), $i \leq t$, has cost at least k_t. So we only need to find the minimal cost of a shortest path from any $(i,0)$ to (t,m) in $G''_{T,P}$ to get an approximation of k_t.

To find the minimal cost, we reset to zero the cost of all horizontal edges $(i,0) \to (i+1,0)$ in $G''_{T,P}$ to get a graph G. The graph G corresponds to taking the pattern matching graph $\tilde{G}_{T,P}$, removing from it all its diagonal edges and adding the shortcut edges. The cost of a path from $(0,0)$ to (t,m) in G is the minimum over $i \leq t$ of the cost of a shortest path from $(i,0)$ to (t,m) in $G''_{T,P}$.

Hence, we want to calculate the cost of the shortest path from $(0,0)$ to (t,m) for all t.[1] For this we will use a simple algorithm that will make a single sweep over the shortcut edges sorted by their origin and calculate the distances for $t = 0, \ldots, n$. The algorithm will maintain a data structure that at time t will allow to answer efficiently queries about the cost of the shortest path from $(0,0)$ to (t,j) for any $j \in \{0,\ldots,m\}$.

The data structure will consist of a binary tree with $m+1$ leaves. Each node is associated with a subinterval of $\{0,\ldots,m\}$ so that the j-th leaf (counting from left to right) corresponds to $\{j\}$, and each internal node corresponds to the union of all its children. We denote by

[1] Although, we really care only about t where $w_2|t$, as for all the other values of t we will approximate k_t by the value equal to $\tilde{k}_{t'} + (t - t')$, where $t' = \lfloor \frac{t}{w_2} \rfloor$. Recall, $\tilde{k}_{t'}$ is the estimated value of $k_{t'}$.

I_v the interval associated with a node v. The depth of the tree is at most $1 + \log(m+1)$. At time t, query to the node v of the data structure will return the cost of the shortest path from $(0,0)$ to $(t, \max I_v)$ that uses some shortcut edge $(i,j) \to (i',j')$, where $j' \in I_v$. Each node v of the data structure stores a pair of numbers (c_v, t_v), where c_v is the cost of the relevant shortest path from $(0,0)$ to $(t_v, \max I_v)$ and t_v is the time it was updated the last time. (Initially this is set to $(\infty, 0)$.) At time $t \geq t_v$, the query to the node v returns $c_v + (t - t_v)$.

At time t to find the cost of the shortest path from $(0,0)$ to (t,j) we traverse the data structure from the root to the leaf j. Let $v_1, \ldots, v_\ell$ be the left children of the nodes along the path in which we continue to the right child. We query nodes $v_1, \ldots, v_\ell$ to get answers $a_1, \ldots, a_\ell$. The cost of the shortest paths from $(0,0)$ to (t,j) is $a = \min\{j, a_1 + (j - \max I_{v_1}), a_2 + (j - \max I_{v_2}), \ldots, a_\ell + (j - \max I_{v_\ell})\}$. As each query takes $O(1)$ time to answer, computing the shortest path to (t,j) takes $O(\log m)$ time.

The algorithm that outputs the cheapest cost of any path from $(0,0)$ to (t,m) in G will process the shortcut edges $(i,j) \to (i',j')$ one by one in the order of increasing i. The algorithm will maintain lists $L_0, \ldots, L_n$ of updates to the data structure to be made before time t. At time t the algorithm first outputs the cost of the shortest path from $(0,0)$ to (t,m). Then it takes each shortcut edge $(t,j) \to (t',j')$ one by one, $t < t'$. (The algorithm ignores shortcut edges where $t = t'$.) Using the current state of the data structure it calculates the cost c of a shortest path from $(0,0)$ to (t,j) and adds $(c+d, j')$ to list $L_{t'}$, where d is the cost of the shortcut edge $(t,j) \to (t',j')$.

After processing all edges starting at $(t, \cdot)$ the algorithm performs updates to the data structure according to the list L_{t+1}. Update (c,j) consists of traversing the tree from the root to the leaf j and in each node v updating its current values (c_v, t_v) to the new values $(c'_v, t+1)$, where $c'_v = \min\{c_v + t + 1 - t_v, c + \max I_v - j\}$. Then the algorithm increments t and continues with further edges.

If the number of shortcut edges is m then the algorithm runs in time $O(n + m(\log m + \log m))$. First, it has to set-up the data structure, sort the edges by their origin and then it processes each edge. Processing each edge will require $O(\log m)$ time to find the min-cost path to the originating vertex and then later at time t' it will require time $O(\log m)$ to update the data structure. As there are $\widetilde{O}(\frac{n}{m} \cdot \frac{m}{\theta w_1} \cdot \frac{m}{w_1}) \leq \widetilde{O}(nm^{3/4})$ certified boxes in total the running time of the algorithm is as required.

The correctness of the algorithm is immediate from its description.

5 Online approximate pattern matching

In this section we describe the online algorithm from Theorem 2. In the online setting the pattern P is given while the text T arrives in online fashion. The main challenge of this setting is that at any point of time (other than the pattern) we are allowed to store a substring of the text of length just sublinear in m. To overcome this situation the online algorithm is based on interleaved execution of the cover and min-cost path algorithms from Sections 3.2 and 4. Moreover we need to maintain some extra data structure in a clever manner for the covering algorithm. Also to get the required space bound we use a slightly modified tree data structure for the min-cost path algorithm. For the online setting we use the same parameters as the offline one, but we set their values slightly differently: $w_1 = m^{11/18}$, $w_2 = m^{20/27}$, $d = m^{7/54}$, $\theta = m^{-1/9}$. Next, we describe the data structure used in the covering algorithm and the modified tree data structure for the min-cost path algorithm.

Covering algorithm data structure. For each substring T^0_r of m consecutive input symbols of text T, and $j = \lceil \log 1/\theta \rceil, \ldots, 0$ the algorithm will maintain a set D'_j that stores the content of strings T^1_i that reached Step 3 of the covering algorithm during processing of T^0_r. Moreover for each of such string T^1_i the algorithm will also store a set $Y_{i,j}$ that contains the spans obtained in Step 4 while processing T^1_i. This is done as the whole m length string T^0_r can't be stored at once. Moreover to bound the size of D'_j and $Y_{i,j}$, before adding a new T^1_i that reached Step 3 of the covering algorithm to D'_j, we first ensure that no string close to T^1_i is already contained in D'_j. Also after finishing each T^0_r we discard all the information associated with it.

Modified tree data structure. Here we describe the modified tree data structure used for the min-cost path algorithm. Notice, every shortcut edge corresponds to some certified box. Our covering algorithm has $\log 1/\theta$ rounds where in any round the total number of possible vertical positions, where the bottom left corner or the top right corner point of any certified box might lie is bounded by $\frac{m}{\theta w_1}$. Next, we round up all the edit distance estimates to powers of two, hence in any certified box there are at most $2 \log m$ positions from which a shortcut edge might start or end. Therefore, the number of distinct vertical positions where these shortcut edges might originate from or lead to is bounded by $q = \frac{2m}{\theta w_1} \cdot \log 1/\theta \cdot \log m$. Thus the tree data structure of the min-cost path algorithm will ever perform updates to at most $q \log m$ distinct nodes. We do not need to store the nodes that are never updated, so the tree data structure will occupy only space $\widetilde{O}(\frac{m}{\theta w_1})$.

5.1 The online algorithm

Now we explain how to interleave the two phases to achieve required time and space bound. The algorithm processes the input text T in batches of w_2 symbols. Upon receipt of the t-th symbol we buffer the symbol, if t is not divisible by w_2 then the algorithm outputs the estimated value for $(t-1)$-th position plus one, i.e., $\tilde{k}_{t-1} + 1$ as the current value k_t and waits for the next symbol. Otherwise we received batch T^2_ℓ of next w_2 symbols, for $\ell = t/w_2$, and we will proceed as follows.

The covering algorithm in the online setting is similar to the covering algorithm offline setting. However here, we will execute the covering algorithm twice on each T^2_ℓ where during the first execution the only thing that we will send to the min-cost path algorithm are the certified boxes produced at Step 9, all other modifications to data structures will be discarded. During the second run of the algorithm on T^2_ℓ, we will preserve all modifications to D'_j's and other data structures except we will discard the certified boxes produced at Step 9 (we will not send them to the min-cost path algorithm as they are already sent in the first pass).

Covering algorithm. We now describe how the covering algorithm executes on each T^2_ℓ. The algorithm maintain sets S_j, $j = \lceil \log 1/\theta \rceil, \ldots, 0$ that are empty at the beginning. We partition T^2_ℓ into $T^1_g, \ldots, T^1_h$ of length w_1, where $g = (\ell - 1) \cdot \frac{w_2}{w_1} + 1$ and $h = g + \frac{w_2}{w_1} - 1$. For $i = g, \ldots, h$ we do the following. For each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, set $\epsilon_j = 2^{-j}$. Check, whether T^1_j is at edit distance at most $2\epsilon_j w_1$ from some string $T^1_{i'}$ in D'_j. If it is then send the set of all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T^1_i in T and $J \in Y_{i',j}$. If it is not close to any string in D'_j then sample the relevant substring in P as in Step 2 and see how many of them are at edit distance $\leq \epsilon_j w_1$ from T^1_i. If at most $\frac{1}{2} \cdot c_0 \cdot \log n$ of the samples have their edit distance from T^1_i below $\epsilon_j w_1$ then put index i into S_j and continue for another j and then the next i. Otherwise we execute Step 4 of the algorithm to find set Y. (We always skip Step 3.) We put T^1_i into D_j and set $Y_{i,j}$ to

Y. During the first execution of the covering algorithm, upon processing all j and i we will directly proceed to the sparse extension sampling part whereas after the second execution of the covering algorithm, we send all the certified boxes $(I, J, 8\epsilon_j w_1)$ to the min-cost path algorithm, where I is the span of T_i^1 and $J \in Y_{i,j}$.

In the extension sampling part for each $j = \lceil \log 1/\theta \rceil, \ldots, 0$, we sample from the set S_j the strings T_ℓ^1 in Step 7, and we proceed for them as in Steps 8–9. During the first execution of the covering algorithm, for each certified box (I, J, ℓ') produced in Step 9 round up ℓ' to the nearest larger or equal power of two and send the box to the min-cost path algorithm.

Min-cost path algorithm. The min-cost path algorithm receives certified boxes from the covering algorithm and it converts them into corresponding shortcut edges. The algorithm receives the certified boxes at two distinct phases.

Shortcut edges generated after the first execution of the covering algorithm correspond to boxes that were produced at Step 9. These edges are sorted by their originating vertex, stored, and processed at appropriate time steps during the next phase.

During the next phase the algorithm receives boxes $(I, J, 8\epsilon_j w_1)$, where I is the span of some T_i^1 and $J \in Y_{i,j}$. It converts them into edges and upon receiving all the edges for a particular T_i^1, it sorts them according to their originating vertex. Then the min-cost path algorithm proceeds from time steps $(i-1) \cdot w_1$ to $i \cdot w_1 - 1$, and processes all stored shortcut edges that originate in these time steps. During these time steps it also updates its tree data structure as in the offline case. Again we use lists for storing pending updates. At any moment of time, the number of unprocessed edges and updates is bounded by the number of edges produced in Step 9 and edges produced for a particular string T_i^1. This is at most $\widetilde{O}(\frac{m}{\theta w_1})$. We conclude by the following lemma:

▶ **Lemma 9.** *Let n and m be large enough integers. Let P be the pattern of length m, T be the text of length n (arriving online one symbol at a time), $1/m \leq \theta \leq 1$ be a real. Let $\theta w_1 \geq 1$, $w_1 \leq \theta w_2$, $w_1 | w_2$ and $w_2 | n$. With probability at least $1 - 1/poly(n)$ the online algorithm for pattern matching runs in amortized time $\widetilde{O}(\frac{m}{d} + \frac{m w_1}{w_2} + d w_2 + \frac{m}{w_1})$ per symbol and in extra space $\widetilde{O}(w_2 + \frac{m}{d\theta} + \frac{m}{w_1 \theta} + \frac{m^2}{\theta^2 w_1^2 d} + d)$.*

We defer the proof of the above lemma to the full version. We instantiate the above lemma for the parameters: $w_1 = m^{11/18}$, $w_2 = m^{20/27}$, $d = m^{7/54}$, $\theta = m^{-1/9}$, to get the following:

▶ **Theorem 10** (Restatement of Theorem 2). *There is a constant $c \geq 1$ so that there is a randomized online algorithm that computes $(c, m^{8/9})$-approximation to approximate pattern matching in amortized time $\tilde{O}(m^{1-(7/54)})$ and extra space $\tilde{O}(m^{1-(1/54)})$ with probability at least $1 - 1/poly(m)$.*

6 Discussion

For our online pattern matching algorithm it can be noticed that there is a clear trade off among the running time, the extra space used by the algorithm and the additive part of the approximation factor. Keeping the running time fixed, decreasing the additive part of the approximation factor (by changing the value of parameter θ) would increase the extra space used, and also keeping the additive error part fixed, decreasing the running time would increase the extra space used.

Open Problem. The online algorithm presented in this paper has non-trivial time and space complexity only for the case when the edit distance between the pattern and the text is high. Therefore, it will be nice to extend our online approximation algorithm for the full range of edit distance, which will be interesting from both theoretical and practical perspectives.

References

1 Amir Abboud and Arturs Backurs. Towards Hardness of Approximation for Polynomial Time Problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 11:1–11:26, 2017.

2 Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.

3 Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016.

4 Amir Abboud and Aviad Rubinstein. Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018.

5 Karl Abrahamson. Generalized String Matching. *SIAM J. Comput.*, 16(6):1039–1051, December 1987.

6 Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic Approximation for Edit Distance and the Asymmetric Query Complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386, 2010.

7 V. L. Arlazarov, E. A. Dinic, M. A. Konrod, and L. A. Faradzev. On economic construction of the transitive closure of a directed graph. *Dokl. Akad*, Nauk SSSR 194:487–488, 1970. [in Russian]. English translation: Soviet. Math. Dokl. 11 No. 5 (1970), 1209–1210.

8 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (Unless SETH is False). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. ACM.

9 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015.

10 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating Edit Distance within Constant Factor in Truly Sub-Quadratic Time. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 979–990, 2018.

11 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time. *CoRR*, abs/1810.03664, 2018. `arXiv:1810.03664`.

12 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A Black Box for Online Approximate Pattern Matching. In *Combinatorial Pattern Matching, 19th Annual Symposium, CPM 2008, Pisa, Italy, June 18-20, 2008, Proceedings*, pages 143–151, 2008.

13 Raphaël Clifford, Markus Jalsenius, and Benjamin Sach. Cell-probe bounds for online edit distance and other pattern matching problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 552–561, 2015.

14 Raphaël Clifford and Benjamin Sach. Online Approximate Matching with Non-local Distances. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Lille, France, June 22-24, 2009, Proceedings*, pages 142–153, 2009.

15 Raphaël Clifford and Benjamin Sach. Pseudo-realtime Pattern Matching: Closing the Gap. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 101–111, 2010.

16 Richard Cole and Ramesh Hariharan. Approximate String Matching: A Simpler Faster Algorithm. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California, USA.*, pages 463–472, 1998.

17 Maxime Crochemore. String-Matching on Ordered Alphabets. *Theor. Comput. Sci.*, 92(1):33–47, 1992.

18 Maxime Crochemore, Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. Two-Dimensional Pattern Matching in Linear Time and Small Space. In *STACS*, pages 181–192, 1995.

19 Zvi Galil and Raffaele Giancarlo. Data structures and algorithms for approximate string matching. *J. Complexity*, 4(1):33–72, 1988.

20 Zvi Galil and Kunsoo Park. An Improved Algorithm for Approximate String Matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

21 Zvi Galil and Joel Seiferas. Time-space-optimal String Matching (Preliminary Report). In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 106–113, New York, NY, USA, 1981. ACM.

22 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving succinct data structures for parameterized pattern matching and related problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 397–407, 2017.

23 Leszek Gasieniec, Wojciech Plandowski, and Wojciech Rytter. The Zooming Method: A Recursive Approach to Time-Space Efficient String-Matching. *Theor. Comput. Sci.*, 147(1&2):19–30, 1995.

24 Piotr Indyk. Faster Algorithms for String Matching Problems: Matching the Convolution Bound. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA*, pages 166–173, 1998.

25 Donald E. Knuth, James H. Morris Jr., and Vaughan R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6(2):323–350, 1977.

26 Tsvi Kopelowitz and Ely Porat. A Simple Algorithm for Approximating the Text-To-Pattern Hamming Distance. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 10:1–10:5, 2018.

27 Gad M. Landau and Uzi Vishkin. Fast Parallel and Serial Approximate String Matching. *Journal of Algorithms*, 10(2):157–169, 1989.

28 VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

29 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

30 G. Myers. Incremental alignment algorithms and their applications. *Technical Report*, 1986.

31 Gonzalo Navarro. A Guided Tour to Approximate String Matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

32 Mihai Patrascu. Succincter. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 305–313, 2008.

33 Peter H. Sellers. The Theory and Computation of Evolutionary Distances: pattern recognition. *Journal of Algorithms*, pages 1:359–373, 1980.

34 Tatiana A. Starikovskaya. Communication and Streaming Complexity of Approximate Pattern Matching. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, pages 13:1–13:11, 2017.

35 Esko Ukkonen. Algorithms for Approximate String Matching. *Inf. Control*, 64(1-3):100–118, March 1985.

36 Esko Ukkonen and Derick Wood. Approximate String Matching with Suffix Automata. *Algorithmica*, 10(5):353–364, 1993.