Algorithmica (2019) 81:2633–2652
https://doi.org/10.1007/s00453-019-00548-x

Longest Common Substring with Approximately *k* Mismatches

Tomasz Kociumaka[1] · **Jakub Radoszewski**[1] · **Tatiana Starikovskaya**[2]

Received: 22 December 2017 / Accepted: 17 January 2019 / Published online: 16 February 2019

Abstract

In the longest common substring problem, we are given two strings of length n and must find a substring of maximal length that occurs in both strings. It is well known that the problem can be solved in linear time, but the solution is not robust and can vary greatly when the input strings are changed even by one character. To circumvent this, Leimeister and Morgenstern introduced the problem of the longest common substring with k mismatches. Lately, this problem has received a lot of attention in the literature. In this paper, we first show a conditional lower bound based on the SETH hypothesis implying that there is little hope to improve existing solutions. We then introduce a new but closely related problem of the longest common substring with approximately k mismatches and use locality-sensitive hashing to show that it admits a solution with strongly subquadratic running time. We also apply these results to obtain a strongly subquadratic-time 2-approximation algorithm for the longest common substring with k mismatches problem and show conditional hardness of improving its approximation ratio.

Keywords Randomised algorithms · String similarity measures · Longest common substring · Sketching · Locality-sensitive hashing · Binary jumbled indexing

This is a full and extended version of the conference paper [31].

✉ Jakub Radoszewski
jrad@mimuw.edu.pl

Tomasz Kociumaka
kociumaka@mimuw.edu.pl

Tatiana Starikovskaya
tat.starikovskaya@gmail.com

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland

[2] DIENS, École Normale Supérieure, PSL University, Paris, France

1 Introduction

Understanding how similar two strings are and what they share in common is a central task in stringology. The significance of this task is witnessed by the 50,000+ citations of the paper introducing BLAST [3], a heuristic algorithmic tool for comparing biological sequences. This task can be formalised in many different ways, from the longest common substring problem to the edit distance problem. The longest common substring problem can be solved in optimal linear time and space, while the best known algorithms for the edit distance problem require $n^{2-o(1)}$ time, which makes the longest common substring problem an attractive choice for many practical applications. On the other hand, the longest common substring problem is not robust and its solution can vary greatly when the input strings are changed even by one character. To overcome this issue, recently a new problem has been introduced called the longest common substring with k mismatches. In this paper, we continue this line of research.

1.1 Related Work

Let us start with a precise statement of the longest common substring problem.

Problem 1.1 (LCS) Given two strings T_1, T_2 of length n, find a maximum-length substring of T_1 that occurs in T_2.

The suffix tree of T_1 and T_2, a data structure containing all suffixes of T_1 and T_2, allows to solve this problem in linear time and space [17,21,36], which is optimal as any algorithm needs $\Omega(n)$ time to read and $\Omega(n)$ space to store the strings. However, if we only account for "additional" space, the space the algorithm uses apart from the space required to store the input, then the suffix tree-based solution is not optimal and has been improved in a series of publications [5,26,32].

The major disadvantage of the longest common substring problem is that its solution is not robust. Consider, for example, two pairs of strings: $a^{2m+1}, a^{2m}b$ and $a^m ba^m, a^{2m}b$. The longest common substring of the first pair of strings is almost twice as long as the longest common substring of the second pair of strings, although we changed only one character. This makes the longest common substring unsuitable to be used as a measure of similarity of two strings: Intuitively, changing one character must not change the measure of similarity much. To overcome this issue, it is natural to allow the substring to occur in T_1 and T_2 not exactly but with a small number of mismatches.

Problem 1.2 (LCS with k Mismatches) Given two strings T_1, T_2 of length n and an integer k, find a maximum-length substring of T_1 that occurs in T_2 with at most k mismatches.

The problem can be solved in quadratic time and space by a dynamic-programming algorithm, but more efficient solutions have also been shown. The longest common substring with one mismatch problem was first considered in [6], where an $O(n^2)$-time and $O(n)$-space solution was given. This result was further improved by Flouri et al. [14], who showed an $O(n \log n)$-time and $O(n)$-space solution.

For a general value of k, the problem was first considered by Leimeister and Morgenstern [29], who suggested a greedy heuristic algorithm. Flouri et al. [14] showed that LCS with k Mismatches admits a quadratic-time algorithm which takes constant (additional) space. Grabowski [16] presented two output-dependent algorithms with running times $O(n((k+1)(\ell_0+1))^k)$ and $O(n^2k/\ell_k)$, where ℓ_0 is the length of the longest common substring of T_1 and T_2 and ℓ_k is the length of the longest common substring with k mismatches of T_1 and T_2. Thankachan et al. [35] gave an $O(n)$-space, $O(n\log^k n)$-time solution for $k = O(1)$. Very recently, Charalampopoulos et al. [10] extended the underlying techniques and developed an $O(n)$-time algorithm for the case of $\ell_k = \Omega(\log^{2k+2} n)$. Finally, Abboud et al. [1] applied the polynomial method to develop a $k^{1.5}n^2/2^{\Omega(\sqrt{(\log n)/k})}$-time randomised solution to the problem. In fact, their algorithm was developed for a more general problem of computing the longest common substring with k edits, but it can be adapted to LCS with k Mismatches as well. The problem of computing the longest common substring with k edits was also considered in [34], where an $O(n\log^k n)$-time solution was given for constant k.

1.2 Our Contribution

Our contribution is as follows. In Sect. 2, we show that existence of a strongly subquadratic-time algorithm for LCS with k Mismatches on strings over binary alphabet for $k = \Omega(\log n)$ refutes the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, and Zane [23,24]; see also [11, Chapter 14]:

Hypothesis 1.3 *(SETH) For every $\delta > 0$, there exists an integer q such that SAT on q-CNF formulas with m clauses and n variables cannot be solved in $m^{O(1)}2^{(1-\delta)n}$ time.*

This conditional lower bound implies that there is little hope to improve existing solutions to LCS with k Mismatches. To this end, we introduce a new problem, inspired by the work of Andoni and Indyk [4].

Problem 1.4 (LCS with Approximately k Mismatches) Two strings T_1, T_2 of length n, an integer k, and a constant $\varepsilon > 0$ are given. If ℓ_k is the length of the longest common substring with k mismatches of T_1 and T_2, return a substring of T_1 of length at least ℓ_k that occurs in T_2 with at most $(1+\varepsilon)\cdot k$ mismatches.

Let $d_H(S_1, S_2)$ denote the Hamming distance between equal-length strings S_1 and S_2, that is, the number of mismatches between them. Then we are to find the substrings S_1 and S_2 of T_1 and T_2, respectively, of length at least ℓ_k such that $d_H(S_1, S_2) \leq (1+\varepsilon)\cdot k$.

Although the problem statement is not standard, it makes perfect sense from the practical point of view. It is also more robust than the LCS with k Mismatches problem, as for most applications it is not important whether a returned substring occurs in T_1 and T_2 with, for example, 10 or 12 mismatches. The result is also important from the theoretical point of view as it improves our understanding of the big picture of string comparison. In their work, Andoni and Indyk used the technique of locality-sensitive hashing to develop a space-efficient randomised index for a variant of the approximate

pattern matching problem. We extend their work with new ideas in the construction and the analysis to develop a randomised subquadratic-time solution to Problem 1.4. This result is presented in Sect. 3.

In Sect. 4, we consider approximation algorithms for the length of the LCS with k Mismatches. By applying previous techniques, we show a strongly subquadratic-time 2-approximation algorithm and show that no strongly subquadratic-time $(2-\varepsilon)$-approximation algorithm exists for any $\varepsilon > 0$ unless SETH fails.

Finally, in Sect. 5 we show a strongly subcubic-time solution for LCS with k Mismatches for all k by reducing it (for arbitrary alphabet size) to Binary Jumbled Indexing. Namely, we show that LCS with k Mismatches for all $k = 1, \ldots, n$ can be solved in $O(n^{2.859})$ expected time or in $O(n^{2.864})$ deterministic time, improving upon naive computation performed for every k separately.

2 LCS with k Mismatches is SETH-Hard

Recall that the Hamming distance of two strings U and V of the same length, denoted as $d_H(U, V)$, is simply the number of mismatches. Our proof is based on conditional hardness of the following problem.

Problem 2.1 (Orthogonal Vectors) Given a set A of N vectors from $\{0, 1\}^d$ each, does there exist a pair of vectors $U, V \in A$ that is orthogonal, i.e., $\sum_{h=1}^{d} U[h]V[h] = 0$?

Williams showed a conditional lower bound for an equivalent problem called cooperative subset queries [37, Section 5.1], which immediately implies the following fact:

Fact 2.2 *Suppose there is $\varepsilon > 0$ such that for all constant c, Orthogonal Vectors on a set of N vectors of dimension $d = c \log N$ can be solved in $2^{o(d)} \cdot N^{2-\varepsilon}$ time. Then SETH is false.*

We treat vectors from $\{0, 1\}^d$ as binary strings of length d. Let us introduce two morphisms, μ and τ:

$$\mu(0) = 011\,1000, \quad \mu(1) = 000\,1000, \quad \tau(0) = 001\,1000, \quad \tau(1) = 111\,1000.$$

We will use the following two observations.

Observation 2.3 *We have $d_H(\mu(0), \tau(0)) = d_H(\mu(0), \tau(1)) = d_H(\mu(1), \tau(0)) = 1$, and $d_H(\mu(1), \tau(1)) = 3$.*

Observation 2.4 *Let $x, y, z \in \{0, 1\}$. Then the string 1000 has exactly two occurrences in $1000xyz1000$.*

Let us also introduce a string gadget $H = \gamma^d$, where $\gamma = 100\,1000$. Note that $\gamma \neq \mu(x)$ and $\gamma \neq \tau(x)$ for $x \in \{0, 1\}$. Further, note that $|H| = |\mu(U)| = |\tau(U)| = 7d$ for any $U \in A$.

Lemma 2.5 *Consider a set of vectors $A = \{U_1, \ldots, U_N\}$ from $\{0,1\}^d$, the strings:*

$$T_1 = H^q \mu(U_1) H^q \cdots \mu(U_N) H^q, \quad T_2 = H^q \tau(U_1) H^q \cdots \tau(U_N) H^q$$

for some positive integer q, and $k = d$. Then:

(a) *If the set A contains two orthogonal vectors, then the* LCS with k Mismatches *problem for T_1 and T_2 has a solution of length at least $\ell = (14q + 7)d$.*
(b) *If the set A does not contain two orthogonal vectors, then all the solutions for the* LCS with k Mismatches *problem for T_1 and T_2 have length smaller than $\ell' = (7q + 14)d$.*

Proof (a) Assume that U_i and U_j are a pair of orthogonal vectors. T_1 contains a substring $H^q \mu(U_i) H^q$ and T_2 contains a substring $H^q \tau(U_j) H^q$. Both substrings have length ℓ and, by Observation 2.3, their Hamming distance is exactly $k = d$.

(b) Assume to the contrary that there are indices a and b for which the substrings $S_1 = T_1[a, a + \ell' - 1]$ and $S_2 = T_2[b, b + \ell' - 1]$ have at most k mismatches. First, let us note that $7 \mid a - b$. Indeed, otherwise S_1 would contain at least $\lfloor (\ell' - 3)/7 \rfloor = (q + 2)k - 1 \geq k + 1$ substrings of the form 1000 which, by Observation 2.4, would not be aligned with substrings 1000 in S_2. Hence, they would account for more than k mismatches between S_1 and S_2.

Let us call all the substrings of T_1 and T_2 that come from the 3-character prefixes of $\mu(0), \mu(1), \tau(0), \tau(1)$, and γ the *core substrings*, with core substrings that come from γ being *gadget core substrings*. We have already established that the core substrings of S_1 and S_2 are aligned. Moreover, S_1 and S_2 contain at least $\lfloor (\ell' - 2)/7 \rfloor = (q + 2)k - 1$ core substrings each. Amongst every $(q + 2)k - 1$ consecutive core substrings in S_1, some k consecutive must come from $\mu(U_i)$ for some index i; a symmetric property holds for S_2 and $\tau(U_j)$. Moreover, as only the gadget core substrings in S_1 and S_2 can match exactly, at most k core substrings that are contained in S_1 and S_2 can be non-gadget. Hence, S_1 and S_2 contain exactly k non-gadget core substrings each. If they were not aligned, they would have produced more than k mismatches in total with the gadget core substrings.

Therefore, S_1 and S_2 must contain, as aligned substrings, $\mu(U_i)[1, 7d - 4]$ and $\tau(U_j)[1, 7d - 4]$ for some $i, j \in \{1, \ldots, N\}$, respectively. Hence, $d_H(U_i, U_j) \leq k$. By Observation 2.3, we conclude that U_i and U_j are orthogonal. □

Theorem 2.6 *Suppose there is $\varepsilon > 0$ such that* LCS with k Mismatches *can be solved in $O(n^{2-\varepsilon})$ time on strings over binary alphabet for $k = \Omega(\log n)$. Then SETH is false.*

Proof The reduction of Lemma 2.5 with $q = 1$ constructs, for an instance of the Orthogonal Vectors problem with N vectors of dimension d, an equivalent instance of the LCS with k Mismatches problem with strings of length $n = 7d(2N + 1)$ and $k = d$. Thus, assuming that LCS with k Mismatches can be solved in $O(n^{2-\varepsilon})$ time for $k = \Omega(\log n)$, the constructed instance can be solved in $O(N^{2-\varepsilon} d^{O(1)})$ time if $d = c \log N$. This, by Fact 2.2, contradicts SETH. □

3 LCS with Approximately *k* Mismatches

In this section, we prove the following theorem.

Theorem 3.1 *Let $\varepsilon \in (0, 2)$ and $\delta \in (0, 1)$ be arbitrary constants. The LCS with Approximately k Mismatches problem can be solved in $O(n^{1+1/(1+\varepsilon)})$ space and $O(n^{1+1/(1+\varepsilon)} \log^2 n)$ time with error probability δ.*

3.1 Overview of the Proof

The classic solution to the longest common substring problem is based on two observations. The first observation is that the longest common substring of T_1 and T_2 is in fact the longest common prefix of some suffix of T_1 and some suffix of T_2. The second observation is that the maximal length of the longest common prefix of a fixed suffix S of T_1 and suffixes of T_2 is reached by one of the two suffixes of T_2 that are closest to S in the lexicographic order. This suggests the following algorithm: First, build a suffix tree of T_1 and T_2, which contains all suffixes of T_1 and T_2 ordered lexicographically. Second, compute the longest common prefix of each suffix of T_1 and the two suffixes of T_2 closest to S in the lexicographic order, one from the left and one from the right. The problem of computing the longest common prefix has been extensively studied in the literature and a number of very efficient deterministic and randomised solutions exist [7,8,12,19,22]; for example, one can use a Lowest Common Ancestor (LCA) data structure, which can be constructed in linear time and space and answers longest common prefix queries in $O(1)$ time [12,19].

Our solution to the longest common substring with approximately k mismatches problem is somewhat similar. Instead of the lexicographic order, we will consider $\Theta(n^{1/(1+\varepsilon)})$ different orderings on the suffixes of T_1 and T_2. To define these orderings, we will use the locality-sensitive hashing technique, which was initially introduced for the needs of computational geometry [18] and later adapted for substrings with Hamming distance [4]. In more detail, we will choose $\Theta(n^{1/(1+\varepsilon)})$ hash functions, where each function can be considered as a projection of a string of length n onto a random subset of its positions. By choosing the size of the subset appropriately, we will be able to guarantee that the hash function is locality-sensitive: For any two strings at the Hamming distance at most k, the values of the hash functions on them will be equal with reasonably high probability, while the values of the hash functions on any pair of strings at the Hamming distance bigger than $(1 + \varepsilon) \cdot k$ will be equal with low probability. For each hash function, we will sort the suffixes of T_1 and T_2 by the lexicographic order on their hash values. As a corollary of the locality-sensitive property, if two suffixes of T_1 and T_2 have a long common prefix with at most k mismatches, they are likely to be close to each other in at least one of the orderings.

However, we will not be able to compute the longest common prefix with $(1 + \varepsilon)k$ mismatches for all candidate pairs of suffixes exactly (the best data structure, based on the kangaroo method [15,28], has query time $\Theta((1 + \varepsilon)k)$ which is $\Theta(n)$ in the worst case). We will use this method for only one pair of suffixes chosen at random from a carefully preselected set of candidate pairs. For other candidate pairs, we will use $\mathrm{LCP}_{\tilde{k}}$ queries. In an $\mathrm{LCP}_{\tilde{k}}$ query, we are given two suffixes S_1, S_2 of T_1 and T_2, respectively,

and must output any integer ℓ such that $\mathrm{LCP}_k(S_1, S_2) \le \ell \le \mathrm{LCP}_{(1+\varepsilon)k}(S_1, S_2)$, where LCP_k and $\mathrm{LCP}_{(1+\varepsilon)k}$ denote the longest common prefix with at most k and at most $(1+\varepsilon)k$ mismatches, respectively. In Sect. 3.2, we show the following lemma based on the sketching techniques by Kushilevitz et al. [27]:

Lemma 3.2 *For given k and ε, after $O(n \log^3 n)$-time and $O(n \log^2 n)$-space preprocessing of strings T_1, T_2, any $\mathrm{LCP}_{\tilde{k}}$ query can be answered in $O(\log^2 n)$ time. With probability at least $1-1/n^3$, the preprocessing produces a data structure that correctly answers all $\mathrm{LCP}_{\tilde{k}}$ queries.*

The key idea is to compute sketches for all power-of-two length substrings of T_1 and T_2. The sketches will have logarithmic length (so that we will be able to compare them very fast) and the Hamming distance between them will be roughly proportional to the Hamming distance between the original substrings. Once the sketches are computed, we use binary search to answer $\mathrm{LCP}_{\tilde{k}}$ queries in polylogarithmic time.

3.2 Proof of Lemma 3.2

During the preprocessing stage, we compute sketches [27] of all substrings of the strings T_1 and T_2 of lengths $\ell = 1, 2, 4, \ldots, 2^{\lfloor \log n \rfloor}$, which can be defined in the following way. Without loss of generality, assume that the alphabet is $\Sigma = \{0, 1, \ldots, p-1\}$, where p is a prime number. For a fixed ℓ, choose $\lambda = \lceil 3 \ln n / \gamma^2 \rceil$ vectors r_ℓ^i of length ℓ, where γ is a constant to be defined later, such that the values $r_\ell^i[j]$ across $i = 1, 2, \ldots, \lambda$ and $j = 1, 2, \ldots, \ell$ are independent and identically distributed so that for every $a \in \Sigma$:

$$\Pr[r_\ell^i[j] = a] = \begin{cases} 1 - \frac{p-1}{2kp} & \text{if } a = 0, \\ \frac{1}{2kp} & \text{otherwise.} \end{cases}$$

For a string X of length ℓ, we define the sketch $\mathrm{sk}(X)$ to be a vector of length λ, where $\mathrm{sk}(X)[i] = r_\ell^i \cdot X \pmod p$. For each $i = 1, 2, \ldots, \lambda$, we compute the inner product of r_ℓ^i with all length-ℓ substrings of T_1 and T_2 in $O(n \log n)$ time by running the Fast Fourier Transform (FFT) algorithm in the field $\mathbb{Z}_p$ [13]. As a result, we obtain the sketches of each length-ℓ substring of T_1 and T_2. We repeat this step for all specified values of ℓ. One instance of the FFT algorithm takes $O(n \log n)$ time, and we run an instance for each $i = 1, 2, \ldots, \lambda$ and for each $\ell = 1, 2, 4, \ldots, 2^{\lfloor \log n \rfloor}$, which takes $O(n \log^3 n)$ time in total. The sketches occupy $O(n \log^2 n)$ space. Each string S can be decomposed uniquely as $X_1 X_2 \ldots X_g$, where $g = O(\log n)$ and $|X_1| > |X_2| > \ldots > |X_g|$ are powers of two; we define a sketch $\mathrm{sk}(S) = \sum_q \mathrm{sk}(X_q) \pmod p$. Let $\delta_1 = \frac{p-1}{p}(1 - (1 - \frac{1}{2k})^k)$ and $\delta_2 = \frac{p-1}{p}(1 - (1 - \frac{1}{2k})^{(1+\varepsilon)\cdot k})$.

Lemma 3.3 (see [27]) *Let S_1, S_2 be strings of the same length. For each $i = 1, \ldots, \lambda$:*

- *if $d_H(S_1, S_2) \le k$, then $\mathrm{sk}(S_1)[i] \ne \mathrm{sk}(S_2)[i]$ with probability at most δ_1;*
- *if $d_H(S_1, S_2) \ge (1+\varepsilon) \cdot k$, then $\mathrm{sk}(S_1)[i] \ne \mathrm{sk}(S_2)[i]$ with probability at least δ_2.*

Proof We use a different interpretation of r_ℓ^i that defines the same distribution. We start with the zero vector and sample positions with probability $\frac{1}{2k}$. For each sampled position j, we decide on the value $r_\ell^i[j] \in \Sigma$ independently and uniformly at random. Let $m = d_H(S_1, S_2)$ and $a_1, \ldots, a_m$ be the positions of the mismatches between the two strings. If none of the positions $a_1, \ldots, a_m$ are sampled, then $\mathrm{sk}(S_1)[i] = \mathrm{sk}(S_2)[i]$. Otherwise, if $a_{j_1}, \ldots, a_{j_g}$ are sampled, for each $r_\ell^i[a_{j_1}], \ldots, r_\ell^i[a_{j_{g-1}}]$ exactly one of the p choices of $r_\ell^i[a_{j_g}]$ results in $\mathrm{sk}(S_1)[i] = \mathrm{sk}(S_2)[i]$ (because p is prime). Hence, the probability that $\mathrm{sk}(S_1)[i] \neq \mathrm{sk}(S_2)[i]$ is equal to $\frac{p-1}{p}(1 - (1 - \frac{1}{2k})^m)$, which is at most δ_1 if $d_H(S_1, S_2) \leq k$, and at least δ_2 if the Hamming distance is at least $(1 + \varepsilon) \cdot k$. □

We set $\Delta = \frac{\delta_1+\delta_2}{2} \cdot \lambda$ and $\gamma = \frac{\delta_2-\delta_1}{2}$. Observe that

$$\gamma = \tfrac{p-1}{2p}(1 - \tfrac{1}{2k})^k \left(1 - (1 - \tfrac{1}{2k})^{\varepsilon k}\right) \geq \tfrac{1}{8}\left(1 - e^{-\varepsilon/2}\right) = \Omega(\varepsilon^{-1})$$

because $(1 - \frac{1}{2k})^k$ is an increasing function of k bounded from above by $e^{1/2}$. Consequently, if ε is a constant, then γ is a constant as well.

Lemma 3.4 *For all strings S_1 and S_2 of the same length, the following claims hold with probability at least $1 - n^{-6}$:*

- *if $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) > \Delta$, then $d_H(S_1, S_2) > k$;*
- *if $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) \leq \Delta$, then $d_H(S_1, S_2) < (1 + \varepsilon) \cdot k$.*

Proof Let χ_i be an indicator random variable that is equal to one if and only if $\mathrm{sk}(S_1)[i] \neq \mathrm{sk}(S_2)[i]$. The claim follows immediately from Lemma 3.3 and the following Chernoff–Hoeffding bounds [20, Theorem 1]. For λ independently and identically distributed binary variables $\chi_1, \chi_2, \ldots, \chi_\lambda$, we have

$$\Pr\left[\frac{1}{\lambda}\sum_{i=1}^{\lambda}\chi_i > \mu + \gamma\right] \leq e^{-2\lambda\gamma^2} \quad \text{and} \quad \Pr\left[\frac{1}{\lambda}\sum_{i=1}^{\lambda}\chi_i \leq \mu - \gamma\right] \leq e^{-2\lambda\gamma^2},$$

where $\mu = \Pr[\chi_i = 1]$. Recall that $\gamma = \frac{\delta_2-\delta_1}{2}$, so we obtain that the error probability is at most $e^{-2\lambda\gamma^2} \leq n^{-6}$.

If $d_H(S_1, S_2) \leq k$, Lemma 3.3 asserts that $\mu \leq \delta_1$. By the first of the above inequalities, we have that $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) \leq \Delta$ with probability at least $1 - n^{-6}$. Hence, if $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) > \Delta$, then $d_H(S_1, S_2) > k$ with the same probability.

If $d_H(S_1, S_2) \geq (1 + \varepsilon) \cdot k$, Lemma 3.3 asserts that $\mu \geq \delta_2$. By the second inequality, we have that $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) > \Delta$ with probability at least $1 - n^{-6}$. Hence, if $d_H(\mathrm{sk}(S_1), \mathrm{sk}(S_2)) \leq \Delta$, then $d_H(S_1, S_2) < (1 + \varepsilon) \cdot k$ with the same probability. □

Suppose we wish to answer an $\mathrm{LCP}_{\tilde{k}}$ query on two suffixes S_1, S_2. It suffices to find the longest prefixes of S_1, S_2 such that the Hamming distance between their sketches is at most Δ. As mentioned above, these prefixes can be represented uniquely as a

concatenation of strings of power-of-two lengths $\ell_1 > \ell_2 > \ldots > \ell_g$. To compute ℓ_1, we initialise it with the biggest power of two not exceeding n and compute the Hamming distance between the sketches of the corresponding substrings. If it does not exceed Δ, we have found ℓ_1; otherwise, we divide ℓ_1 by two and continue. Suppose that we already know $\ell_1, \ell_2, \ldots, \ell_i$ and the sketches $\mathrm{sk}(S_1[1, d_i])$ and $\mathrm{sk}(S_2[1, d_i])$, where $d_i = \ell_1 + \cdots + \ell_i$. To determine ℓ_{i+1}, we initialise it with $\frac{1}{2}\ell_i$ and then divide it by two until $d_H(\mathrm{sk}(S_1[1, d_i + \ell_{i+1}]), \mathrm{sk}(S_2[1, d_i + \ell_{i+1}])) \leq \Delta$. These two sketches can be computed in $O(\lambda) = O(\log n)$ time by combining $\mathrm{sk}(S_1[1, d_i])$ and $\mathrm{sk}(S_2[1, d_i])$ with the precomputed sketches $\mathrm{sk}(S_1[d_i + 1, d_i + \ell_{i+1}])$ and $\mathrm{sk}(S_2[d_i + 1, d_i + \ell_{i+1}])$, respectively. Consequently, the query procedure takes $O(\log^2 n)$ time. It errs on at least one query with probability at most n^{-3} (Lemma 3.4 is only applied for pairs of same-length substrings of T_1 and T_2, so we estimate error probability by the union bound). This completes the proof of Lemma 3.2.

3.3 Proof of Theorem 3.1

We start by preprocessing T_1 and T_2 as described in Lemma 3.2. In the main phase of the algorithm, we construct a family $\mathbf{H}$ of hash functions based on four parameters $m, s, t, w \in \mathbb{Z}$ to be specified later.

Let Π be the set of all projections of strings of length n onto a single position, i.e. the value $\pi_i(S)$ of the i-th projection on a string S is simply its i-th character $S[i]$. More generally, for a string S of length n and a function $h = (\pi_{a_1}, \ldots, \pi_{a_q}) \in \Pi^q$, we define $h(S)$ as $S[a_{p_1}]S[a_{p_2}]\cdots S[a_{p_q}]$, where p is a permutation such that $a_{p_1} \leq \cdots \leq a_{p_q}$. If $|S| < n$, we define $h(S) := h(S \cdot \$^{n-|S|})$, where $\$ \notin \Sigma$ is a special gap-filling character.

Each hash function $h \in \mathbf{H}$ is going to be a uniformly random element of Π^{mt}; however, the individual hash functions are *not* chosen independently in order to ensure faster running time for the algorithm. Nevertheless, $\mathbf{H}$ will be composed of s independent subfamilies $\mathbf{H}_i$, each of size $\binom{w}{t}$. To construct $\mathbf{H}_i$, we choose w functions $u_{i,1}, \ldots, u_{i,w} \in \Pi^m$ independently and uniformly at random. Each hash function $h \in \mathbf{H}_i$ is defined as an unordered t-tuple of distinct functions $u_{i,r}$. Formally,

$$\mathbf{H}_i = \{(u_{i,r_1}, u_{i,r_2}, \ldots, u_{i,r_t}) \in \Pi^{mt} : 1 \leq r_1 < r_2 < \cdots < r_t \leq w\}.$$

Consider the set of all suffixes $S_1, S_2, \ldots, S_{2n}$ of T_1 and T_2. For each $h \in \mathbf{H}$, we define an ordering $\prec_h$ of the suffixes $S_1, \ldots, S_{2n}$ according to the lexicographic order of the values $h(S_j)$ of the hash function and, in case of ties, according to the lengths $|S_j|$. To construct it, we build a compact trie[1] on strings $h(S_1), h(S_2), \ldots, h(S_{2n})$.

Theorem 3.5 *Functions $u_{i,r}$ for $i = 1, \ldots, s$ and $r = 1, \ldots, w$ can be preprocessed in $O(n^{4/3}\log^{4/3} n)$ time and $O(n)$ space each, i.e., in $O(swn^{4/3}\log^{4/3} n)$ time and $O(swn)$ space in total, so that afterwards, for each $h \in \mathbf{H}$, a trie on $h(S_1), \ldots, h(S_{2n})$*

[1] Recall that a compact trie stores only explicit nodes, that is, the root, the leaves, and nodes with at least two children. Its size is linear in the number of strings that are stored. Henceforth we call a compact trie simply a trie.

can be constructed in $O(tn \log n)$ time and $O(n)$ space. The preprocessing errs with probability $O(1/n)$ for each $u_{i,r}$, i.e., $O(sw/n)$ in total.

Let us defer the proof of the theorem until we complete the description of the algorithm and derive Theorem 3.1. We preprocess functions $u_{i,r}$ and build a trie on $h(S_1), \ldots, h(S_{2n})$ for each $h \in \mathbf{H}_i$. We then augment the trie with an LCA data structure, which can be done in linear time and space [12,19]. The latter can be used to find in constant time the longest common prefix of any two strings $h(S_j)$ and $h(S_{j'})$.

Consider a function $h \in \mathbf{H}$ and a positive integer $\ell \leq n$. We define $h\big|_{[\ell]}$ so that

$$h\big|_{[\ell]}(S) = \begin{cases} h(S[1,\ell]) & \text{if } |S| \geq \ell, \\ h(S) & \text{otherwise.} \end{cases}$$

In other words, if h is a projection onto positions from a multiset P, then $h\big|_{[\ell]}$ is a projection onto positions from the multiset $\{p \in \mathsf{P} : p \leq \ell\}$, extended with \$'s to length mt. Consequently, $h\big|_{[\ell]}(S) = h\big|_{[\ell]}(S')$ if and only if the longest common prefix of $h(S)$ and $h(S')$ is at least $|\{p \in \mathsf{P} : p \leq \ell\}|$ characters long.

We define the family of *collisions* $\mathbf{C}^{\mathbf{H}}_{\ell}$ as a set of triples (S, S', h) such that S and S' are suffixes of T_1 and T_2, respectively, both of length at least ℓ, and $h \in \mathbf{H}$ is such that the suffixes collide on $h\big|_{[\ell]}$, that is, $h\big|_{[\ell]}(S) = h\big|_{[\ell]}(S')$. Note that the families of collisions are nested: $\mathbf{C}^{\mathbf{H}}_0 \supseteq \cdots \supseteq \mathbf{C}^{\mathbf{H}}_{\ell} \supseteq \mathbf{C}^{\mathbf{H}}_{\ell+1} \supseteq \cdots \supseteq \mathbf{C}^{\mathbf{H}}_n$.

For a fixed function h, we define the *ℓ-neighbourhood* of S as the set of suffixes S' of T_2 such that $(S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell}$. We observe that the ℓ-neighbourhood of S forms a contiguous range in the sequence of suffixes of T_2 ordered according to $\prec_h$, and this range can be identified in $O(\log n)$ time using binary search and LCA queries on the trie constructed for h. Consequently, an $O(n|\mathbf{H}|)$-space representation of $\mathbf{C}^{\mathbf{H}}_{\ell}$, with one range for every ℓ-neighbourhood of each suffix S, can be constructed in $O(n|\mathbf{H}| \log n)$ time.

Algorithm 1 Longest common substring with approximately k mismatches.

```
Preprocess T_1, T_2 for LCP_k̃ queries
for i = 1, 2, ..., s do
    for r = 1, 2, ..., w do
        Choose a function u_{i,r} ∈ Π^m uniformly at random and preprocess it using Theorem 3.5
    end for
    for all h = (u_{i,r_1}, u_{i,r_2}, ..., u_{i,r_t}) do
        Build a trie on h(S_1), h(S_2), ..., h(S_{2n}) and augment it with an LCA data structure
        Add h to H
    end for
end for
Find the largest ℓ such that |C^H_ℓ| ≥ 2n|H|
for all (S, S', h) ∈ C^H_{ℓ+1} do
    Compute LCP_k̃(S, S') and update the answer
end for
Pick (S̄, S̄', h̄) ∈ C^H_ℓ uniformly at random
Compute LCP_{(1+ε)k}(S̄, S̄') and update the answer
```

In the algorithm, we find the largest ℓ such that $|\mathbf{C}_\ell^{\mathbf{H}}| \geq 2n|\mathbf{H}|$; using a binary search, this takes $O(n|\mathbf{H}|\log^2 n)$ time. For each $(S, S', h) \in \mathbf{C}_{\ell+1}^{\mathbf{H}}$, we compute the longest common prefix with approximately k mismatches $\mathrm{LCP}_{\tilde{k}}(S, S')$ (Lemma 3.2). Additionally, we pick a single element $(\bar{S}, \bar{S}', \bar{h}) \in \mathbf{C}_\ell^{\mathbf{H}}$ uniformly at random and compute the longest common prefix with at most $(1+\varepsilon)k$ mismatches $\mathrm{LCP}_{(1+\varepsilon)k}(\bar{S}, \bar{S}')$ naively in $O(n)$ time. The longest of the retrieved prefixes is returned as an answer.

The algorithm is summarised in the pseudocode above. We will now proceed to the analysis of its complexity and correctness.

3.4 Complexity and Correctness

To ensure the complexity bounds and correctness of the algorithm, we must carefully choose the parameters s, t, w, and m. Let $p_1 = 1 - k/n$, $p_2 = 1 - (1+\varepsilon)\cdot k/n$, and $\rho = \log p_1 / \log p_2$. The intuition behind these values is that if S and S' are two strings of length n and $d_H(S, S') \leq k$, then p_1 is a lower bound for the probability of $S[i] = S'[i]$ for a uniformly random position i. On the other hand, p_2 is an upper bound for the same probability if $d_H(S, S') \geq (1+\varepsilon)\cdot k$. Based on these values, we define

$$t = \left\lceil \sqrt{\log n} \right\rceil, \quad m = \left\lceil \tfrac{1}{t} \log_{p_2} \tfrac{1}{n} \right\rceil, \quad w = t^2 + \lceil p_1^{-m} \rceil, \quad \text{and} \quad s = \Theta(t!).$$

We assume that $(1+\varepsilon)k < n$ in order to guarantee $p_1 > p_2 > 0$. Note that if $(1+\varepsilon)k \geq n$, the problem is trivial.

3.4.1 Complexity

To show the complexity of the algorithm, we will start with a simple observation and a more involved fact.

Observation 3.6 *We have $s = n^{o(1)}$ and $w = n^{o(1)}$.*

Proof First, observe

$$s = O(t!) = 2^{O(t\log t)} = 2^{O(\sqrt{\log n}\log\log n)} = 2^{o(\log n)} = n^{o(1)}.$$

Similarly,

$$\begin{aligned} w &= t^2 + \lceil p_1^{-m} \rceil \leq t^2 + 1 + p_1^{-m} = O(\log n) + p_1^{-O(\frac{1}{t}\log_{p_2}\frac{1}{n})} \\ &= O(\log n) + 2^{O(\rho\sqrt{\log n})}. \end{aligned}$$

Moreover, $p_1 > p_2$ yields $\log p_1 > \log p_2$ and therefore $\rho = \frac{\log p_1}{\log p_2} < 1$. Consequently, $w = O(\log n) + 2^{O(\sqrt{\log n})} = n^{o(1)}$, which concludes the proof. □

Fact 3.7 *We have $|\mathbf{H}| = O(n^{1/(1+\varepsilon)})$.*

Proof Observe that $|\mathbf{H}| = s\binom{w}{t} = O(t!\binom{w}{t}) = O(w^t)$. To estimate the latter, we consider two cases. If $w \leq 3t^3$, then

$$w^t = \left(3t^3\right)^t = 2^{O(t \log t)} = 2^{O(\sqrt{\log n} \log\log n)} = 2^{o(\log n)} = n^{o(1)} = O(n^{1/(1+\varepsilon)}).$$

Otherwise, $p_1^{-m} \geq w - 1 - t^2 \geq 3t^3 - t^2 \geq t^3 + t$. Consequently,

$$\begin{aligned} w^t &= (t^2 + \lceil p_1^{-m} \rceil)^t \leq (t^2 + 1 + p_1^{-m})^t = p_1^{-mt} \left(1 + \tfrac{t^2+1}{p_1^{-m}}\right)^t \\ &\leq p_1^{-mt} \left(1 + \tfrac{1}{t}\right)^t \leq p_1^{-mt} \cdot e. \end{aligned}$$

Thus, it suffices to prove that $p_1^{-mt} = O(n^{1/(1+\varepsilon)})$. We have

$$\log(p_1^{-mt}) = -t \left\lceil \tfrac{1}{t} \log_{p_2} \tfrac{1}{n} \right\rceil \log p_1 \leq (-\log_{p_2} \tfrac{1}{n} - t) \log p_1 = \rho \log n - t \log p_1.$$

Moreover, due to $(1+\varepsilon)k < n$ and $\varepsilon = \Theta(1)$, we have

$$-t \log p_1 = -t \log(1 - \tfrac{k}{n}) = t \log \tfrac{n}{n-k} = O\left(t \tfrac{k}{n-k}\right) = O\left(t \tfrac{1+\varepsilon}{\varepsilon} \tfrac{k}{n}\right) = O\left(\tfrac{k}{n}\sqrt{\log n}\right).$$

On the other hand, taking the Taylor's expansion of $f(x) = \frac{\log(1-x)}{\log(1-(1+\varepsilon)x)}$, which is concave for $0 \leq x < \frac{1}{1+\varepsilon}$, we obtain

$$\rho = \frac{\log\left(1 - \frac{k}{n}\right)}{\log\left(1 - (1+\varepsilon)\frac{k}{n}\right)} \leq \tfrac{1}{1+\varepsilon} - \tfrac{\varepsilon}{2(\varepsilon+1)} \tfrac{k}{n} = \tfrac{1}{1+\varepsilon} - \Theta(\tfrac{k}{n}).$$

Consequently,

$$\begin{aligned} \log(p_1^{-mt}) \leq \rho \log n - t \log p_1 &\leq \tfrac{\log n}{1+\varepsilon} - \Theta\left(\tfrac{k}{n} \log n\right) \\ &\quad + O\left(\tfrac{k}{n}\sqrt{\log n}\right) = \tfrac{\log n}{1+\varepsilon} - \Theta\left(\tfrac{k}{n} \log n\right). \end{aligned}$$

Thus, $p_1^{-mt} \leq n^{1/(1+\varepsilon)}$ holds for sufficiently large n and therefore $|\mathbf{H}| = O(w^t) = O(e \cdot p_1^{-mt}) = O(n^{1/(1+\varepsilon)})$, which concludes the proof. □

Lemma 3.8 *The running time of the algorithm is* $O(n^{1+1/(1+\varepsilon)} \log^2 n)$.

Proof Preprocessing for $\mathrm{LCP}_{\tilde{k}}$ queries takes $O(n \log^3 n)$ time (Lemma 3.2), whereas functions $u_{i,r}$ are processed in $O(ws \cdot n^{4/3} \log^{4/3} n)$ overall time using Theorem 3.5. Afterwards, for each hash function $h \in \mathbf{H}$ we can build a trie and an LCA data structure on strings $h(S_1), \ldots, h(S_{2n})$ in $O(tn \log n)$ time, which is $O(|\mathbf{H}| tn \log n)$ in total. Next, the value ℓ and the family $|\mathbf{C}^{\mathbf{H}}_{\ell+1}|$ are computed in $O(|\mathbf{H}| n \log^2 n)$ time. The time for $|\mathbf{C}^{\mathbf{H}}_{\ell+1}| < 2n|\mathbf{H}|$ $\mathrm{LCP}_{\tilde{k}}$ queries is bounded by the same function. Finally, we answer one $\mathrm{LCP}_{(1+\varepsilon)k}$ query, which takes $O(n)$ time. The overall running time is

$$O(n \log^3 n + ws \cdot n^{4/3} \log^{4/3} n + |\mathbf{H}| n \log n (t + \log n)) = O(n^{4/3+o(1)} + n^{1+1/(1+\varepsilon)} \log^2 n)$$

due to Observation 3.6 and Fact 3.7. We can hide the first term because of $\varepsilon < 2$. □

Lemma 3.9 *The space complexity of the algorithm is* $O(n^{1+1/(1+\varepsilon)})$.

Proof The data structure for $\mathrm{LCP}_{\tilde{k}}$ queries requires $O(n \log^2 n)$ space. Preprocessing functions $u_{i,r}$ requires $O(swn) = O(n^{1+o(1)})$ space and the tries occupy $O(|\mathbf{H}| \cdot n) = O(n^{1+1/(1+\varepsilon)})$ space. □

3.4.2 Correctness

First, let us focus on two suffixes which yield the longest common substring with exactly k mismatches.

Lemma 3.10 *Let S and S' be suffixes of T_1 and T_2, respectively, that maximise $\mathrm{LCP}_k(S, S')$, i.e., such that $\mathrm{LCP}_k(S, S') = \ell_k$. For each $i \in \{1, \ldots, s\}$, with probability $\Omega(1/t!)$ there exists $h \in \mathbf{H}_i$ such that $h\big|_{[\ell_k]}(S) = h\big|_{[\ell_k]}(S')$.*

Proof By definition of ℓ_k, we have $d_H(S[1, \ell_k], S'[1, \ell_k]) \le k$. Moreover, for any hash function h we have that $h\big|_{[\ell_k]}(S) = h\big|_{[\ell_k]}(S')$ if and only if $h(S[1, \ell_k]) = h(S'[1, \ell_k])$. Let us recall that each hash function $h \in \mathbf{H}_i$ is a t-tuple of functions $u_{i,r} \in \Pi^m$. Consequently, $h(S[1, \ell_k]) = h(S'[1, \ell_k])$ for some $h \in \mathbf{H}_i$ if and only if the strings $S[1, \ell_k]\$^{n-\ell_k}$ and $S'[1, \ell_k]\$^{n-\ell_k}$ collide on at least t out of w functions $u_{i,r}$. We shall give a lower bound on the probability μ of this event. Individual collisions are independent and each of them holds with the same probability $q = p_1^m$. Moreover, μ may only increase as we increase q, so we can replace q by a lower bound $\frac{1}{w}$. (Note that $w = t^2 + \lceil p_1^{-m} \rceil \ge \lceil q^{-1} \rceil \ge q^{-1}$.) We have

$$\mu = \sum_{i=t}^{w} \binom{w}{i} q^i (1-q)^{w-i} \ge \sum_{i=t}^{w} \binom{w}{i} \tfrac{1}{w^i} (1 - \tfrac{1}{w})^{w-i} \ge \binom{w}{t} \tfrac{1}{w^t} (1 - \tfrac{1}{w})^w \ge \tfrac{1}{t!} (\tfrac{w-t+1}{w})^t (\tfrac{w-1}{w})^w.$$

Hence,

$$\tfrac{1}{\mu t!} \le (\tfrac{w}{w-t+1})^t (\tfrac{w}{w-1})^w = (1 + \tfrac{t-1}{w-t+1})^t (1 + \tfrac{1}{w-1})^w \le \exp\left(\tfrac{t(t-1)}{w-t+1} + \tfrac{w}{w-1}\right) = O(1),$$

where the latter is true because $w \ge t^2$ and $w \ge 2$. Consequently, $\mu = \Omega(1/t!)$. □

As a corollary, we can choose a constant in the number of steps $s = \Theta(t!)$ so that $(S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell_k}$ for some $h \in \mathbf{H}$ holds with probability at least $\frac{3}{4}$. If additionally $\ell_k > \ell$, then $(S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell+1}$, so $\mathrm{LCP}_{\tilde{k}}(S, S')$ will be called and with high probability will return a substring of length $\ge \ell_k$. Otherwise, $|\mathbf{C}^{\mathbf{H}}_{\ell_k}| \ge 2n|\mathbf{H}|$ and we claim that a uniformly random $(\bar{S}, \bar{S}', \bar{h}) \in \mathbf{C}^{\mathbf{H}}_{\ell}$ satisfies $\mathrm{LCP}_{(1+\varepsilon)k}(\bar{S}, \bar{S}') \ge \ell \ge \ell_k$ with

probability at least $\frac{1}{2}$. To prove this, we first introduce a family $\mathbf{B^H}$ of *bad collisions*: triples (S, S', h) which belong to $\mathbf{C}^{\mathbf{H}}_{\ell}$ for some $\ell > \mathrm{LCP}_{(1+\varepsilon)k}(S, S')$, and bound its expected size.

Lemma 3.11 *The expected number of bad collisions satisfies* $\mathbb{E}[|\mathbf{B^H}|] \leq n|\mathbf{H}|$.

Proof Let us bound the probability that $(S, S', h) \in \mathbf{B^H}$ for fixed suffixes S and S' (of T_1 and T_2, respectively) and fixed $h = (u_{i,r_1}, \ldots, u_{i,r_t})$. Equivalently, we shall bound $\Pr[(S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell}]$ for $\ell = \mathrm{LCP}_{(1+\varepsilon)k}(S, S') + 1$.

If $|S| < \ell$ or $|S'| < \ell$, the probability is 0 by the definition of $\mathbf{C}^{\mathbf{H}}_{\ell}$. Otherwise, we observe that $d_H(S[1, \ell], S'[1, \ell]) > (1 + \varepsilon)k$ and that h can be considered (due to its marginal distribution) as a projection onto mt uniformly random positions. Therefore,

$$\Pr[h\big|_{[\ell]}(S) = h\big|_{[\ell]}(S')] = \Pr[h(S[1, \ell]\$^{n-\ell}) = h(S'[1, \ell]\$^{n-\ell})] \leq p_2^{mt} \leq \frac{1}{n},$$

where the last inequality follows from the definition of m, which yields $mt \geq \log_{p_2} \frac{1}{n}$.

In total, we have $n^2|\mathbf{H}|$ possible triples (S, S', h) so by linearity of expectation, we conclude that the expected number of bad collisions is at most $\frac{1}{n}n^2|\mathbf{H}| = n|\mathbf{H}|$. □

Corollary 3.12 *Let* (S, S', h) *be a uniformly random element of* $\mathbf{C}^{\mathbf{H}}_{\ell}$*, where* ℓ *is a* random variable *which always satisfies* $|\mathbf{C}^{\mathbf{H}}_{\ell}| \geq 2n|\mathbf{H}|$. *We have* $\Pr[(S, S', h) \in \mathbf{B^H}] \leq \frac{1}{2}$.

Proof More formally, we shall prove that $\Pr[(S, S', h) \in \mathbf{B^H} \mid (S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell}] \leq \frac{1}{2}$ holds for a uniformly random triple (S, S', h). Indeed:

$$\Pr[(S, S', h) \in \mathbf{B^H} \mid (S, S', h) \in \mathbf{C}^{\mathbf{H}}_{\ell}] = \mathbb{E}\left[\frac{|\mathbf{B^H} \cap \mathbf{C}^{\mathbf{H}}_{\ell}|}{|\mathbf{C}^{\mathbf{H}}_{\ell}|}\right] \leq \mathbb{E}\left[\frac{|\mathbf{B^H}|}{2n|\mathbf{H}|}\right] \leq \frac{1}{2}.$$ □

Below, we combine the previous results to prove that with constant probability Algorithm 1 correctly solves the Approximate LCS with k Mismatches problem. Note that we can reduce the error probability to an arbitrarily small constant $\delta > 0$: it suffices to repeat the algorithm a constant number of times and among the resulting pairs, choose the longest substrings successfully verified to be at Hamming distance at most $(1 + \varepsilon)k$; verification can be implemented naively in $O(n)$ time.

Corollary 3.13 *With non-zero constant probability, Algorithm 1 succeeds — it reports a substring of* T_1 *and a substring of* T_2 *at Hamming distance at most* $(1 + \varepsilon)k$*, both of length at least* ℓ_k*, where* ℓ_k *is the length of the longest common substring with* k *mismatches.*

Proof We will prove that the algorithm succeeds conditioned on the following events:

- the preprocessing of Lemma 3.2 succeeds,
- the preprocessing of Theorem 3.5 succeeds for each function $u_{i,r}$,
- $\mathbf{C}^{\mathbf{H}}_{\ell_k}$ contains (S, S', h) such that $\mathrm{LCP}_k(S, S') = \ell_k$ (see Lemma 3.10),
- the randomly chosen $(\bar{S}, \bar{S}', \bar{h}) \in \mathbf{C}^{\mathbf{H}}_{\ell}$ does not belong to $\mathbf{B^H}$ (see Corollary 3.12).

This assumption holds with probability $\Omega(1)$, because probability of the complementary event can be bounded as follows using the union bound applied on the top of Lemma 3.2, Theorem 3.5, Lemma 3.10, and Corollary 3.12:

$$\tfrac{1}{n^3} + O(\tfrac{ws}{n}) + \tfrac{1}{4} + \tfrac{1}{2} = \tfrac{3}{4} + o(1) = 1 - \Omega(1).$$

Successful preprocessing of functions $u_{i,r}$ guarantees that the value ℓ and the families $\mathbf{C}^{\mathbf{H}}_{\ell}$ and $\mathbf{C}^{\mathbf{H}}_{\ell+1}$ have been computed correctly. If $\ell_k > \ell$, then $\mathbf{C}^{\mathbf{H}}_{\ell+1}$ contains (S, S', h) such that $\mathrm{LCP}_k(S, S') = \ell_k$. The correctness of $\mathrm{LCP}_{\tilde{k}}$ queries asserts that $\mathrm{LCP}_{\tilde{k}}(S, S') \geq \ell_k$, so the algorithm considers prefixes of S and S' of length at least ℓ_k as candidates for the resulting substrings. If $\ell_k \leq \ell$, on the other hand, then the randomly chosen $(\bar{S}, \bar{S}', \bar{h}) \in \mathbf{C}^{\mathbf{H}}_{\ell}$ satisfies $\mathrm{LCP}_{(1+\varepsilon)k}(\bar{S}, \bar{S}') \geq \ell \geq \ell_k$, so the algorithm considers prefixes of $\bar{S}$ and $\bar{S}'$ of length at least $\ell \geq \ell_k$. In either case, a pair of substrings of length at least ℓ_k and at Hamming distance at most $(1+\varepsilon)k$ is among the considered candidates. The resulting substrings also satisfy these conditions, because we return the longest candidates and the correctness of $\mathrm{LCP}_{\tilde{k}}$ queries asserts that no substrings at distance more than $(1+\varepsilon)k$ are considered. □

3.5 Proof of Theorem 3.5

Recall that each $h \in \mathbf{H}$ is a t-tuple of functions $u_{i,r}$, i.e. $h = (u_{i,r_1}, u_{i,r_2}, \ldots, u_{i,r_t})$, where $1 \leq i \leq s$ and $1 \leq r_1 < r_2 < \cdots < r_t \leq w$. We will show a preprocessing of functions $u_{i,r}$ after which we will be able to compute the longest common prefix of any two strings $u_{i,r}(S_j), u_{i,r}(S_{j'})$ in $O(1)$ time. As a result, we will be able to compute the longest common prefix of $h(S_j), h(S_{j'})$ in $O(t)$ time. It also follows that we will be able to compare any two strings $h(S_j), h(S_{j'})$ in $O(t)$ time as the order $\prec_h$ is defined by the character following the longest common prefix (or by the lengths $|S_j|$ and $|S_{j'}|$ if $h(S_j) = h(S_{j'})$). Therefore, we can sort strings $h(S_1), h(S_2), \ldots, h(S_{2n})$ in $O(tn \log n)$ time and $O(n)$ space and then compute the longest common prefix of each two adjacent strings in $O(tn)$ time. The trie on $h(S_1), h(S_2), \ldots, h(S_{2n})$ can then be built in $O(n)$ time by imitating its depth-first traverse.

It remains to explain how we preprocess individual functions $u_{i,r}$. For each function, it suffices to build a trie on strings $u_{i,r}(S_1), u_{i,r}(S_2), \ldots, u_{i,r}(S_{2n})$ and to augment it with an LCA data structure [12,19]. We will consider two different methods for constructing the trie with time dependent on m. No matter what the value of m is, one of these methods will have $O(n^{4/3} \log^{4/3} n)$ running time. Let $u_{i,r}$ be a projection onto a multiset P of positions $1 \leq a_1 \leq a_2 \leq \cdots \leq a_m \leq n$ and denote $T = T_1\$^n T_2\$^n$.

Lemma 3.14 *The trie on $u_{i,r}(S_1), \ldots, u_{i,r}(S_{2n})$ can be constructed in $O(\sqrt{mn} \log n)$ time and $O(n)$ space correctly with error probability at most $1/n$.*

Proof Without loss of generality assume that $\sqrt{m}$ is integer. Let us partition P into subsets $B_1, \ldots, B_{\sqrt{m}}$, where

$$B_\ell = \left\{a_{\ell,1}, a_{\ell,2}, \ldots, a_{\ell,\sqrt{m}}\right\} = \left\{a_{(\ell-1)\sqrt{m}+q} \mid q \in [1, \sqrt{m}]\right\}.$$

Now $u_{i,r}$ can be represented as a $\sqrt{m}$-tuple of projections $b_1, b_2, \ldots, b_{\sqrt{m}}$ onto the subsets $B_1, B_2, \ldots, B_{\sqrt{m}}$, respectively. We will build the trie by layers to avoid space overhead. Suppose that we have built the trie for a function $(b_1, b_2, \ldots, b_{\ell-1})$ and we want to extend it to the trie for $(b_1, b_2, \ldots, b_{\ell-1}, b_\ell)$.

Let p be a prime of value $\Omega(n^5)$. With error probability inverse polynomial in n, we can find such p in $O(\log^{O(1)} n)$ time; see [2,33]. We choose a uniformly random $r \in \mathbb{F}_p$ and create a vector χ of length n. We initialise χ as a zero vector and for each position $a_{\ell,q} \in B_\ell$, we increase $\chi[a_{\ell,q}]$ by r^q. We then run the FFT algorithm for χ and T in the field $\mathbb{Z}_p$ [13]. The output of the FFT algorithm contains the inner products of χ and all suffixes $S_1, S_2, \ldots, S_{2n}$. The inner product of χ and a suffix S_j is the Karp–Rabin fingerprint [25] $\varphi_{\ell,j}$ of $b_\ell(S_j)$, where

$$\varphi_{\ell,j} = \sum_{q=1}^{\sqrt{m}} S_j[a_{\ell,q}] \cdot r^q \pmod{p}.$$

If the fingerprints of $b_\ell(S_j)$ and $b_\ell(S_{j'})$ are equal, then $b_\ell(S_j)$ and $b_\ell(S_{j'})$ are equal with probability at least $1 - 1/n^4$, and otherwise they differ (for a proof, see e.g. [30]).

For a fixed leaf of the trie for $(b_1, b_2, \ldots, b_{\ell-1})$, we first sort all the suffixes that end in it by fingerprints $\varphi_{\ell,j}$. Second, we lexicographically sort the strings $b_\ell(S_j)$ with distinct fingerprints. For this, we need to be able to compare $b_\ell(S_j)$ and $b_\ell(S_{j'})$ and to find the first character where they differ. We compare $b_\ell(S_j)$ and $b_\ell(S_{j'})$ character-by-character in $O(\sqrt{m})$ time. We then append the leaf of the trie for $(b_1, b_2, \ldots, b_{\ell-1})$ with a trie on strings $b_\ell(S_j)$ that can be built by imitating its depth-first traverse.

By the union bound, the error probability is at most $\frac{1}{n^4} \cdot n^2\sqrt{m} \leq \frac{1}{n}$. We now analyse the complexity of the algorithm. For each of the $\sqrt{m}$ layers, the FFT algorithm takes $O(n \log n)$ time. The sort by fingerprints takes $O(n \log n)$ time per layer, or $O(\sqrt{m}n \log n)$ time in total. We finally need to estimate the total number of character-by-character comparisons in all the layers. We claim that it can be upper bounded by $O(n \log n)$. The reason for that is as follows: if we consider the resulting trie for $u_{i,r}(S_1), \ldots, u_{i,r}(S_{2n})$, it has size $O(n)$. Imagine that the layers cut this trie into a number of smaller tries. The total size of these tries is still $O(n)$, and we build each of these tries using character-by-character comparisons. For a trie of size x, we need $O(x \log x)$ comparisons, which in total is $O(n \log n)$. Therefore, the character-by-character comparisons take $O(\sqrt{m}n \log n)$ time in total. □

The second method builds the trie using the algorithm described in the first paragraph of this section: we only need to give a method for computing the longest common prefix of $u_{i,r}(S_j)$ and $u_{i,r}(S_{j'})$ (or, equivalently, the first position where $u_{i,r}(S_j)$ and $u_{i,r}(S_{j'})$ differ). The following lemma shows that this query can be answered in $O(n \log n/m)$ time, which gives $O(n^2 \log^2 n/m)$ time complexity of the trie construction.

Lemma 3.15 (see [4]) *After $O(n)$-time and space preprocessing the first position where two strings $u_{i,r}(S_j)$ and $u_{i,r}(S_{j'})$ differ can be found in $O(n \log n/m)$ time correctly with error probability at most $1/n^3$.*

Proof For $m = O(\log n)$ the conclusion is trivial. Assume otherwise. We start by building the suffix tree for the string T which takes $O(n)$ time and space [17,36]. Furthermore, we augment the suffix tree with an LCA data structure in $O(n)$ time [12, 19].

Let $\ell = \lceil 3n \ln n/m \rceil$. We can find the first ℓ positions $q_1 < q_2 < \cdots < q_\ell$ where S_j and $S_{j'}$ differ in $O(\ell) = O(n \log n/m)$ time using the kangaroo method [15,28]. We set $q_r = \infty$ if a given position does not exist. The idea of the kangaroo method is as follows. We can find q_1 by one query to the LCA data structure in $O(1)$ time. After removing the first q_1 positions of S_j and $S_{j'}$, we obtain suffixes S_{j+q_1}, $S_{j'+q_1}$ and find q_2 by another query to the LCA data structure, and so on. If at least one of the positions $q_1, q_2, \ldots, q_\ell$ belongs to P, then we return the first such position as an answer, and otherwise we say that $u_{i,r}(S_j) = u_{i,r}(S_{j'})$. The multiset P can be stored as an array of multiplicities so that testing if an element belongs to it can be done in constant time.

Let us show that if p is the first position where $u_{i,r}(S_j)$ and $u_{i,r}(S_{j'})$ differ, then p belongs to $\{q_1, q_2, \ldots, q_\ell\}$ with high probability. Because $q_1 < q_2 < \cdots < q_\ell$ are the first ℓ positions where S_j and $S_{j'}$ differ, it suffices to show that at least one of these positions belongs to P. We rely on the fact that positions of P are independent and uniformly random elements of $[1, n]$. Consequently, we have $\Pr[q_1, \ldots, q_\ell \notin \mathsf{P}] = (1 - \ell/n)^m \le (1 - 3 \ln n/m)^m \le \frac{1}{e^{3 \ln n}} = 1/n^3$. □

By Lemmas 3.14 and 3.15, the trie on strings $u_{i,r}(S_1), \ldots, u_{i,r}(S_{2n})$ can be built in $O(\min\{\sqrt{m}, n \log n/m\} \cdot n \log n) = O(n^{4/3} \log^{4/3} n)$ time and $O(n)$ space correctly with high probability which implies Theorem 3.5 as explained in the beginning of this section.

4 Approximate LCS with *k* Mismatches

In this section, we consider an approximate variant of the LCS with k Mismatches problem, defined as follows.

Problem 4.1 (Approximate LCS with k Mismatches) Two strings T_1, T_2 of length n, an integer k, and a constant $z > 1$ are given. If ℓ_k is the length of the longest common substring with k mismatches of T_1 and T_2, return a substring of T_1 of length at least ℓ_k/z that occurs in T_2 with at most k mismatches.

Theorem 4.2 (a) *The Approximate LCS with k Mismatches problem for $z = 2$ can be solved in $O(n^{1.5} \log^2 n)$ time and $O(n^{1.5})$ space.*

(b) *Suppose there exist $0 < \varepsilon < 1$ and $\delta > 0$ such that the Approximate LCS with k Mismatches problem for $z = 2 - \varepsilon$ and a binary alphabet can be solved in $O(n^{2-\delta})$ time. Then SETH is false.*

Proof (a) The algorithm of Theorem 3.1 for $\varepsilon = 1$ computes a pair of substrings of length at least ℓ_k of T_1 and T_2 that have Hamming distance at most $2k$. Either the first halves or the second halves of the strings have Hamming distance at most k.

(b) We use the gap that exists in Lemma 2.5 for $q > 1$. Assume that there is such an algorithm for some ε and δ. We will run it for strings T_1 and T_2 from that lemma. Let

$q = \lceil \frac{3}{\varepsilon} \rceil - 2$; then $\ell/\ell' \geq 2 - \varepsilon$. If the Orthogonal Vectors problem has a solution, by Lemma 2.5(a), the algorithm produces a longest common substring of length at least $\ell/(2-\varepsilon) \geq \ell'$. Otherwise, by Lemma 2.5(b), its result has length smaller than ℓ'. This concludes that the conjectured approximation algorithm can be used to solve the Orthogonal Vectors problem.

The lengths of the selected strings are $n = N(7dq + 7d) + 7dq = O(Nd)$ for $d = c \log N$. Hence, the running time is $O(n^{2-\delta}) = O(N^{2-\delta} d^{O(1)})$, which, by Fact 2.2, contradicts SETH. □

5 LCS with *k* Mismatches for all *k*

The following problem has received a considerable attention in the recent years; see [9] and the references therein.

Problem 5.1 (Binary Jumbled Indexing) Construct a data structure over a binary string S of length n that, given positive integers ℓ and q, can compute if there is a substring of S of length ℓ containing exactly q ones.

A simple combinatorial argument shows that it suffices to compute the minimal and maximal number of ones in a substring of S of length ℓ, as for every intermediate number of ones a substring of S of this length exists as well. As a result, the Binary Jumbled Indexing problem can be solved in linear space and with constant-time queries. It turns out that the index can also be constructed in strongly subquadratic time.

Lemma 5.2 (Chan and Lewenstein [9]) *The index for Binary Jumbled Indexing of $O(n)$ size and with $O(1)$-time queries can be constructed in $O(n^{1.859})$ expected time or in $O(n^{1.864})$ deterministic time.*

We use this result to solve the LCS with k Mismatches problem for all values of k simultaneously.

Theorem 5.3 *LCS with k Mismatches for all k can be solved in $O(n^{2.859})$ expected time or in $O(n^{2.864})$ deterministic time.*

Proof Note that, equivalently, we can compute, for all $\ell = 1, \ldots, n$, what is the minimal Hamming distance between substrings of length ℓ in T_1 and T_2.

Let M be an $n \times n$ Boolean matrix such that $M[i, j] = 0$ if and only if $T_1[i] = T_2[j]$. We construct $2n - 1$ binary strings corresponding to the diagonals of M: the string number p, for $p \in \{-n, \ldots, n\}$, corresponds to the diagonal $M[i, j] : j - i = p$. For each of the strings, we construct the jumbled index using Lemma 5.2.

Each diagonal corresponds to one of the possible alignments of T_1 and T_2. In the jumbled index we compute, in particular, for each value of ℓ what is the minimal number of 1s (which correspond to mismatches between the corresponding positions in T_1 and T_2) in a string of length ℓ. To compute the global minimum for a given ℓ, we only need to take the minimum across all the jumbled indexes.

By Lemma 5.2, all the jumbled indexes can be constructed in $O(n^{2.859})$ expected time or in $O(n^{2.864})$ time deterministically. □

Acknowledgements Jakub Radoszewski was supported by the "Algorithms for text processing with errors and uncertainties" project carried out within the HOMING programme of the Foundation for Polish Science co-financed by the European Union under the European Regional Development Fund. Funding was provided by Foundation for Polish Science (Grant No. Homing/2016-2/16).

Funding Open access funding approved by Polish ICM agreement.

References

1. Abboud, A., Williams, R.R., Yu, H.: More applications of the polynomial method to algorithm design. In: Indyk P. (ed.) 26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, pp. 218–230. SIAM (2015). https://doi.org/10.1137/1.9781611973730.17
2. Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Ann. Math. **160**(2), 781–793 (2004). https://doi.org/10.4007/annals.2004.160.781
3. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. J. Mol. Biol. **215**(3), 403–410 (1990). https://doi.org/10.1016/s0022-2836(05)80360-2
4. Andoni, A., Indyk, P.: Efficient algorithms for substring near neighbor problem. In: 17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 1203–1212. SIAM (2006). https://doi.org/10.1145/1109557.1109690
5. Babenko, M.A., Starikovskaya, T.: Computing longest common substrings via suffix arrays. In: Hirsch, E.A., Razborov, A.A., Semenov, A.L., Slissenko, A. (eds.) Computer Science Symposium in Russia, CSR 2008, LNCS, vol. 5010, pp. 64–75. Springer (2008). https://doi.org/10.1007/978-3-540-79709-8_10
6. Babenko, M.A., Starikovskaya, T.: Computing the longest common substring with one mismatch. Probl. Inf. Transm. **47**(1), 28–33 (2011). https://doi.org/10.1134/S0032946011010030
7. Bille, P., Gørtz, I.L., Kristensen, J.: Longest common extensions via fingerprinting. In: Dediu, A., Martín-Vide, C. (eds.) Language and Automata Theory and Applications, LATA 2012, LNCS, vol. 7183, pp. 119–130. Springer (2012). https://doi.org/10.1007/978-3-642-28332-1_11
8. Bille, P., Gørtz, I.L., Sach, B., Vildhøj, H.W.: Time-space trade-offs for longest common extensions. J. Discrete Algorithms **25**, 42–50 (2014). https://doi.org/10.1016/j.jda.2013.06.003
9. Chan, T.M., Lewenstein, M.: Clustered integer 3SUM via additive combinatorics. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th Annual ACM Symposium on Theory of Computing, STOC 2015, pp. 31–40. ACM (2015). https://doi.org/10.1145/2746539.2746568
10. Charalampopoulos, P., Crochemore, M., Iliopoulos, C.S., Kociumaka, T., Pissis, S.P., Radoszewski, J., Rytter, W., Waleń, T.: Linear-time algorithm for long LCF with k mismatches. In: Navarro, G., Sankoff, D., Zhu, B. (eds.) Combinatorial Pattern Matching, CPM 2018, LIPIcs, vol. 105, pp. 23:1–23:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.CPM.2018.23
11. Cygan, M., Fomin, F.V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015). https://doi.org/10.1007/978-3-319-21275-3
12. Fischer, J., Heun, V.: Space-efficient preprocessing schemes for range minimum queries on static arrays. SIAM J. Comput. **40**(2), 465–492 (2011). https://doi.org/10.1137/090779759
13. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Karp, R.M. (ed.) Complexity of Computation, SIAM-AMS Proceedings, vol. 7, pp. 113–125. AMS, Providence, RI (1974)
14. Flouri, T., Giaquinta, E., Kobert, K., Ukkonen, E.: Longest common substrings with k mismatches. Inf. Process. Lett. **115**(6–8), 643–647 (2015). https://doi.org/10.1016/j.ipl.2015.03.006
15. Galil, Z., Giancarlo, R.: Parallel string matching with k mismatches. Theor. Comput. Sci. **51**, 341–348 (1987). https://doi.org/10.1016/0304-3975(87)90042-9
16. Grabowski, S.: A note on the longest common substring with k-mismatches problem. Inf. Process. Lett. **115**(6–8), 640–642 (2015). https://doi.org/10.1016/j.ipl.2015.03.003
17. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997). https://doi.org/10.1017/cbo9780511574931
18. Har-Peled, S., Indyk, P., Motwani, R.: Approximate nearest neighbor: towards removing the curse of dimensionality. Theory Comput. **8**(1), 321–350 (2012). https://doi.org/10.4086/toc.2012.v008a014
19. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984). https://doi.org/10.1137/0213024
20. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963). https://doi.org/10.1080/01621459.1963.10500830

21. Hui, L.C.K.: Color set size problem with application to string matching. In: Apostolico, A., Crochemore, M., Galil, Z., Manber, U. (eds.) Combinatorial Pattern Matching, CPM 1992, LNCS, vol. 644, pp. 230–243. Springer (1992). https://doi.org/10.1007/3-540-56024-6_19
22. Ilie, L., Navarro, G., Tinta, L.: The longest common extension problem revisited and applications to approximate string searching. J. Discrete Algorithms **8**(4), 418–428 (2010). https://doi.org/10.1016/j.jda.2010.08.004
23. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727
24. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001). https://doi.org/10.1006/jcss.2001.1774
25. Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. IBM J. Res. Dev. **31**(2), 249–260 (1987). https://doi.org/10.1147/rd.312.0249
26. Kociumaka, T., Starikovskaya, T., Vildhøj, H.W.: Sublinear space algorithms for the longest common substring problem. In: Schulz, A.S., Wagner, D. (eds.) Algorithms, ESA 2014, *LNCS*, vol. 8737, pp. 605–617. Springer (2014). https://doi.org/10.1007/978-3-662-44777-2_50
27. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM J. Comput. **30**(2), 457–474 (2000). https://doi.org/10.1137/S0097539798347177
28. Landau, G.M., Vishkin, U.: Efficient string matching with k mismatches. Theor. Comput. Sci. **43**, 239–249 (1986). https://doi.org/10.1016/0304-3975(86)90178-7
29. Leimeister, C., Morgenstern, B.: kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. Bioinformatics **30**(14), 2000–2008 (2014). https://doi.org/10.1093/bioinformatics/btu331
30. Porat, B., Porat, E.: Exact and approximate pattern matching in the streaming model. In: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, pp. 315–323. IEEE Computer Society (2009). https://doi.org/10.1109/FOCS.2009.11
31. Starikovskaya, T.: Longest common substring with approximately k mismatches. In: Grossi, R., Lewenstein, M. (eds.) Combinatorial Pattern Matching, CPM 2016, *LIPIcs*, vol. 54, pp. 21:1–21:11. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.CPM.2016.21
32. Starikovskaya, T., Vildhøj, H.W.: Time-space trade-offs for the longest common substring problem. In: Fischer, J., Sanders, P., (eds.) Combinatorial Pattern Matching, CPM 2013, LNCS, vol. 7922, pp. 223–234. Springer (2013). https://doi.org/10.1007/978-3-642-38905-4_22
33. Tao, T., Croot III, E., Helfgott, H.: Deterministic methods to find primes. Math. Comput. **81**(278), 1233–1246 (2012). https://doi.org/10.1090/S0025-5718-2011-02542-1
34. Thankachan, S.V., Aluru, C., Chockalingam, S.P., Aluru, S.: Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis. In: Raphael, B.J. (ed.) Research in Computational Molecular Biology, RECOMB 2018, LNCS, vol. 10812, pp. 211–224. Springer (2018). https://doi.org/10.1007/978-3-319-89929-9_14
35. Thankachan, S.V., Apostolico, A., Aluru, S.: A provably efficient algorithm for the k-mismatch average common substring problem. J. Comput. Biol. **23**(6), 472–482 (2016). https://doi.org/10.1089/cmb.2015.0235
36. Weiner, P.: Linear pattern matching algorithms. In: 14th Annual Symposium on Switching and Automata Theory, SWAT 1973, pp. 1–11. IEEE Computer Society, Washington, DC, USA (1973). https://doi.org/10.1109/SWAT.1973.13
37. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**(2–3), 357–365 (2005). https://doi.org/10.1016/j.tcs.2005.09.023