Approximating Longest Common Substring with k mismatches: Theory and Practice

Garance Gourdel
ENS Paris Saclay, France
garance.gourdel@ens-paris-saclay.fr

Tomasz Kociumaka
Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland
Samsung R&D Institute, Warsaw, Poland
jrad@mimuw.edu.pl

Tatiana Starikovskaya
DIENS, École normale supérieure, PSL Research University, France
tat.starikovskaya@gmail.com

Abstract

In the problem of the longest common substring with k mismatches we are given two strings X, Y and must find the maximal length ℓ such that there is a length-ℓ substring of X and a length-ℓ substring of Y that differ in at most k positions. The length ℓ can be used as a robust measure of similarity between X, Y. In this work, we develop new approximation algorithms for computing ℓ that are significantly more efficient that previously known solutions from the theoretical point of view. Our approach is simple and practical, which we confirm via an experimental evaluation, and is probably close to optimal as we demonstrate via a conditional lower bound.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases approximation algorithms, string similarity, LSH, conditional lower bounds

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.16

Related Version A full version of the paper is available at `https://arxiv.org/abs/2004.13389`.

Supplementary Material To ensure the reproducibility of our results, our complete experimental setup, including data files, is available at `https://github.com/fnareoh/LCS_Approx_k_mis`.

Funding *Tomasz Kociumaka*: Supported by ISF grants no. 1278/16 and 1926/19, a BSF grant no. 2018364, and an ERC grant MPM (no. 683064) under the EU's Horizon 2020 Research and Innovation Programme.
Jakub Radoszewski: Supported by the Polish National Science Center, grant number 2018/31/D/ST6/03991.

1 Introduction

For decades, the edit distance and its variants remained the most relevant measure of similarity between biological sequences. However, there is strong evidence that the edit distance cannot be computed in strongly subquadratic time [7]. One possible approach to overcoming the quadratic time barrier is computing the edit distance approximately, and last year in the breakthrough paper Chakraborty et al. [8] showed a constant-factor approximation algorithm that computes the edit distance between two strings of length n in time $\tilde{\mathcal{O}}(n^{2-2/7})$. Nevertheless, the algorithm is highly non-trivial and because of that is likely to be impractical.

A different approach is to consider alignment-free measures of similarities. Ideally, we want the measure to be robust and simple enough so that we could compute it efficiently. One candidate for such a measure is the length of the longest common substring with k mismatches. Formally, given two strings X, Y of lengths at most n and an integer k, we want to find the maximal length $\mathrm{LCS}_k(X, Y)$ of a substring of X that occurs in Y with at most k mismatches. Computing this value constitutes the LCS with k Mismatches problem.

The LCS with k Mismatches problem was first considered for $k = 1$ [6, 13], with current best algorithm taking $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space. The first algorithm for the general value of k was shown by Flouri et al. [13]. Their simple approach used quadratic time and linear space. Grabowski [15] focused on a data-dependent approach, namely, he showed two linear-space algorithms with running times $\mathcal{O}(n((k+1)(\mathrm{LCS}+1))^k)$ and $\mathcal{O}(n^2 k/\mathrm{LCS}_k)$, where LCS is the length of the longest common substring of X and Y and LCS_k, similarly to above, is the length of the longest common substring with k mismatches of X and Y. Abboud et al. [1] showed a $k^{1.5} n^2 / 2^{\Omega(\sqrt{(\log n)/k})}$-time randomised solution to the problem via the polynomial method. Thankachan et al. [24] presented an $\mathcal{O}(n \log^k n)$-time, $\mathcal{O}(n)$-space solution for constant k. This approach was recently extended by Charalampopoulos et al. [10] to develop an $\mathcal{O}(n)$-time and $\mathcal{O}(n)$-space algorithm for the case of $\mathrm{LCS}_k = \Omega(\log^{2k+2} n)$.

On the other hand, Kociumaka, Radoszewski, and Starikovskaya [19] showed that there is $k = \Theta(\log n)$ such that the LCS with k Mismatches problem cannot be solved in strongly subquadratic time, even for the binary alphabet, unless the Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi, and Zane [16] is false. This conditional lower bound implies that there is little hope to improve existing solutions to LCS with k Mismatches. To overcome this barrier, they introduced an approximation approach to LCS with k Mismatches, inspired by the work of Andoni and Indyk [4].

▶ **Problem 1** (LCS with Approximately k Mismatches). *Two strings X, Y of length at most n, an integer k, and a constant $\varepsilon > 0$ are given. Return a substring of X of length at least $\mathrm{LCS}_k(X, Y)$ that occurs in Y with at most $(1+\varepsilon) \cdot k$ mismatches.*

Kociumaka, Radoszewski, and Starikovskaya [19] also showed that for any $\varepsilon \in (0, 2)$ the LCS with Approximately k Mismatches problem can be solved in $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log^2 n)$ time and $\mathcal{O}(n^{1+1/(1+\varepsilon)})$ space. Besides for superlinear space, their solution uses a very complex class of hash functions which requires $n^{4/3+o(1)}$-time preprocessing, and that is the underlying reason for the bounds on ε. In this work, we significantly improve the complexity of the LCS with Approximately k Mismatches problem and show the following results.

▶ **Theorem 2.** *Let $\varepsilon > 0$ be an arbitrary constant. The LCS with Approximately k Mismatches problem can be solved correctly with high probability:*

1) *In $\mathcal{O}(n^{1+1/(1+2\varepsilon)+o(1)})$ time and $\mathcal{O}(n^{1+1/(1+2\varepsilon)+o(1)})$ space assuming a constant-size alphabet;*

2) *In $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log^3 n)$ time and $\mathcal{O}(n)$ space for alphabets of arbitrary size.*

Our first solution uses the Approximate Nearest Neighbour data structure [5] as a black box. The definition of this data structure is extremely involved, and we view this result as more of a theoretical interest. On the other hand, our second solution is simple and practical, which we confirm by experimental evaluation (see Section 4 for details).

As a final remark, we note that a construction similar to the one used to show a lower bound for the LCS with k Mismatches problem [19] gives a lower bound for LCS with Approximately k Mismatches. A proof of the following fact can be found in Section 5.

▶ **Fact 3.** *Assuming SETH, for every constant $\delta > 0$, there exists a constant $\varepsilon = \varepsilon(\delta)$*[1] *such that any randomised algorithm that solves the LCS with Approximately k Mismatches problem for given X and Y of length at most n correctly with constant probability uses $\Omega(n^{2-\delta})$ time.*

Related work. In 2014, Leimester and Morgenstern [20] introduced a related similarity measure, *the k-macs distance.* Let $\mathrm{LCP}_k(X_i, Y_j) = \max\{\ell : d_H(X[i, i+\ell-1], Y[j, j+\ell-1]) \le k\}$, where d_H stands for Hamming distance, i.e. the number of mismatches between two strings. We have $\mathrm{LCS}_k = \max_{i,j} \mathrm{LCP}_k(X_i, Y_j)$. The k-macs distance, on the other hand, is defined as a normalised average of these values. Leimeister and Morgenstern [20] showed a heuristic algorithm for computing the k-macs distance, with no theoretical guarantees for the precision of the approximation; other heuristic approaches for computing the k-macs distance include [25, 26]. The only algorithm with provable theoretical guarantees is [24] and it computes the k-macs distance in $\mathcal{O}(n \log^k n)$ time and $\mathcal{O}(n)$ space.

2 Preliminaries

We assume that the alphabet of the strings X, Y is $\Sigma = \{1, \dots, \sigma\}$, where $\sigma = n^{\mathcal{O}(1)}$.

Karp–Rabin fingerprints. The Karp–Rabin fingerprint [18] of a string $S = s_1 s_2 \dots s_\ell$ is defined as

$$\varphi(S) = \left(\sum_{i=1}^{\ell} r^{i-1} s_i\right) \bmod q,$$

where $q = \Omega(\max\{n^5, \sigma\})$ is a prime number, and $r \in \mathbb{F}_q$ is chosen uniformly at random. Obviously, if $S_1 = S_2$, then $\varphi(S_1) = \varphi(S_2)$. Furthermore, for any $\ell \le n$, if the fingerprints of two ℓ-length strings S_1, S_2 are equal, then S_1, S_2 are equal with probability at least $1 - 1/n^4$ (for a proof, see e.g. [21]).

Dimension reduction. We will exploit a computationally efficient variant of the Johnson–Lindenstrauss lemma [17] which describes a low-distortion embedding from a high-dimensional Euclidean space into a low-dimensional one. Let $\|\cdot\|$ be the Euclidean (L_2) norm of a vector. We will exploit the following claim which follows immediately from [2, Theorem 1.1]:

▶ **Lemma 4.** *Let P be a set of n vectors in $\mathbb{R}^\ell$, where $\ell \le n$. Given $\alpha = \alpha(n) > 0$ and a constant $\beta > 0$, there is $d = \Theta(\alpha^{-2} \log n)$ and a scalar $c > 0$ such that the following holds. Let M be a $d \times \ell$ matrix filled with i.u.d. ± 1 random variables. For all $U \in P$, define $\mathrm{sk}_\alpha(U) = c \cdot MU$. Then for all $U, V \in P$ there is $\|U - V\|^2 \le \|\mathrm{sk}_\alpha(U) - \mathrm{sk}_\alpha(V)\|^2 \le (1+\alpha)\|U - V\|^2$ with probability at least $1 - n^{-\beta}$.*

Since the Hamming distance between binary strings U, V is equal to $\|U - V\|^2$, the matrix M defines a low-distortion embedding from an ℓ-dimensional into a d-dimensional Hamming space as well. For non-binary strings, an extra step is required. Let the alphabet be $\Sigma = \{1, 2, \dots, \sigma\}$ and consider a morphism $\mu : \Sigma \to \{0,1\}^\sigma$, where $\mu(a) = 0^{a-1}10^{\sigma-a}$ for all $a \in \Sigma$. We extend μ to strings in a natural way. Note that for two strings U, V over the alphabet Σ the Hamming distance between $\mu(U), \mu(V)$ is exactly twice the Hamming distance between U, V. We therefore obtain:

[1] Here δ is a function of ε for which the explicit form is not known (a condition inherited from [22]).

▶ **Corollary 5.** *Let P be a set of n strings in Σ^ℓ, where $\ell \leq n$. Given $\alpha = \alpha(n) > 0$ and a constant $\beta > 0$, there is $d = \Theta(\alpha^{-2} \log n)$ and a scalar $c > 0$ such that the following holds. Let M be a $d \times (\sigma \cdot \ell)$ matrix filled with i.u.d. ± 1 random variables. For all $U \in P$, define $\mathrm{sk}_\alpha(U) = c \cdot M\mu(U)$. Then for all $U, V \in P$ there is $d_H(U,V) \leq \|\mathrm{sk}_\alpha(U) - \mathrm{sk}_\alpha(V)\|^2 \leq (1+\alpha) d_H(U,V)$ with probability at least $1 - n^{-\beta}$.*

We will use the corollary for dimension reduction, and also to design a simple test that checks whether the Hamming distance between two strings is at most k.

▶ **Corollary 6.** *Let P be a set of n strings in Σ^ℓ, where $\ell \leq n$. With probability at least $1 - n^{-\beta}$, for all $U, V \in P$:*
1) *if $\|\mathrm{sk}_\alpha(U) - \mathrm{sk}_\alpha(V)\|^2 \leq (1+\alpha)k$, then $d_H(U,V) \leq (1+\alpha) \cdot k$;*
2) *if $\|\mathrm{sk}_\alpha(U) - \mathrm{sk}_\alpha(V)\|^2 > (1+\alpha)k$, then $d_H(U,V) \geq k$.*

2.1 The Twenty Questions game

Consider the following version of the classic game "Twenty Questions". There are two players: Paul and Carole; Carole thinks of two numbers A, B between 0 and N, and Paul must return some number in $[A, B]$. He is allowed to ask questions of form "Is $x \leq A$?", for any $x \in [0, N]$. If $x \leq A$, Carole must return YES; If $A < x \leq B$, she can return anything; and if $B < x$, she must return NO. Paul must return the answer after having asked at most Q questions where Carole can tell at most $\lceil \rho Q \rceil$ lies, and only in the case when $x \leq A$.

We show that Paul has a winning strategy for $Q = \Theta(\log n)$ and any $\rho < 1/3$ by a black-box reduction to the result of Dhagat, Gács, and Winkler [11] who showed a winning strategy for $A = B$.

▶ **Theorem 7** ([11]). *For $A = B$, Paul has a winning strategy for all $\rho < \frac{1}{3}$ asking $Q = \lceil \frac{8 \log N}{(1-3\rho)^2} \rceil$ questions.*

This result is obtained by maintaining a stack of trusted intervals. Once Paul knows that A is between ℓ and r, where $\ell \leq r$, he checks whether A is in the left or the right half of the interval $[\ell, r]$. If no inconsistencies appear (like $A < \ell$ or $r < A$), he pushes the new interval to the stack, else he removes the interval $[\ell, r]$ from the stack of trusted intervals. After Q rounds, Paul returns the only number in the top interval in the stack, which is guaranteed to have length 1 and to contain A. We give the pseudocode of Paul's strategy in Algorithm 1. By *Carole*(x), we denote the answer of Carole for a question "Is $x \leq A$?".

Algorithm 1 The Twenty Questions game.

```
1:  $Q \leftarrow \lceil \frac{8 \log N}{(1-3\rho)^2} \rceil$
2:  $S \leftarrow \{[0, N]\}$
3:  for $i = 1, 2, \ldots, Q/2$ do
4:      $I = [\ell, r] \leftarrow S.top()$
5:      $mid \leftarrow \lceil \frac{\ell + r}{2} \rceil$
6:      if $Carole(mid)$ then
7:          if $Carole(r)$ then $S.pop()$          ▷ The answer is inconsistent with $I$; remove $I$.
8:          else $S.push([mid, r])$
9:      else
10:         if $Carole(\ell)$ then $S.push([\ell, mid-1])$
11:         else $S.pop()$                         ▷ The answer is inconsistent with $I$; remove $I$.
```

We now a show a winning strategy for our variant of the game.

▶ **Corollary 8.** *For $A \leq B$, Paul has a winning strategy for all $\rho < \frac{1}{3}$ asking $Q = \frac{8 \log N}{(1-3\rho)^2}$ questions.*

Proof. We introduce just one change to Algorithm 1, namely, we return the argument of the largest YES obtained in the course of the algorithm. From the problem statement it follows that the answer is at most B. We shall now prove that the answer is at least A. If Carole ever returned YES for $A < x \leq B$, then it is obviously the case. Otherwise, Carole actually behaved as if she had $A = B$ in mind: apart from the small fraction of erroneous answers, she returned YES for $x \leq A$, and NO for $x > A$. Thus, the strategy of Dhagat, Gács, and Winkler ends up with A as the answer (and this must be due to a YES for $x = A$). ◀

3 LCS with Approximately k Mismatches

In this section, we prove Theorem 2. Let us first introduce a decision variant of the LCS with Approximately k Mismatches problem.

▶ **Problem 9.** *Two strings X, Y of length at most n, integers k, ℓ, and a constant $\varepsilon > 0$ are given. We must return:*
1. *YES if $\ell \leq \mathrm{LCS}_k(X, Y)$;*
2. *Anything if $\mathrm{LCS}_k(X, Y) < \ell \leq \mathrm{LCS}_{(1+\varepsilon)k}(X, Y)$;*
3. *NO if $\mathrm{LCS}_{(1+\varepsilon)k}(X, Y) < \ell$.*

If we return YES, we must also give a witness pair *of length-ℓ substrings S_1 and S_2 of X and Y, respectively, such that $d_H(S_1, S_2) \leq (1 + \varepsilon)k$.*

The decision variant of the LCS with Approximately k Mismatches problem can be reduced to the following (c, r)-Approximate Near Neighbour problem.

▶ **Problem 10.** *In the (c, r)-Approximate Near Neighbour problem with failure probability f, the aim is, given a set P of n points in $\mathbb{R}^d$, to construct a data structure supporting the following queries: given any point $q \in \mathbb{R}^d$, if there exists $p \in P$ such that $\|p - q\| \leq r$, then return some point $p' \in P$ such that $\|p' - q\| \leq cr$ with probability at least $1 - f$.*

Using the reduction, we will show our first solution to the LCS with Approximately k Mismatches decision problem based on the result of Andoni and Razenshteyn [5], who showed that for any constant f, there is a data structure for the (c, r)-Approximate Near Neighbour problem that has $\mathcal{O}(n^{1+\rho+o(1)} + d \cdot n)$ size, $\mathcal{O}(d \cdot n^{\rho+o(1)})$ query time, and $\mathcal{O}(d \cdot n^{1+\rho+o(1)})$ preprocessing time, where $\rho = 1/(2c^2 - 1)$.

▶ **Lemma 11.** *Assume an alphabet of constant size σ. The decision variant of the LCS with Approximately k Mismatches problem can be solved in space $\mathcal{O}(n^{1+1/(1+2\varepsilon)+o(1)})$ and time $\mathcal{O}(n^{1+1/(1+2\varepsilon)+o(1)})$. The answer is correct with constant probability.*

Proof. Let P be the set of all length-ℓ substrings of X and Q be the set of all length-ℓ substrings of Y, all encoded in binary using the morphism μ (see Section 2). We start by applying the dimension reduction procedure of Corollary 5 to P and Q with $\alpha = 1/(\log \log n)^{\Theta(1)}$ and $\beta = 2$ to obtain sets P' and Q'. We can implement the procedure in $\mathcal{O}(\sigma n \log^2 n (\log \log n)^{\Theta(1)}) = \mathcal{O}(n \log^{2+o(1)} n)$ time by encoding X, Y using μ and running the FFT algorithm [12] for each of the $\mathcal{O}(\log^{1+o(1)} n)$ rows of the matrix and $\mu(X), \mu(Y)$.

To solve the decision variant of LCS with Approximately k Mismatches, we build the data structure of Andoni and Razenshteyn [5] for $(\sqrt{(1+\varepsilon)(1-\alpha)}, \sqrt{(1+\alpha)k})$-Approximate Near Neighbour over Q'. We make a query for each string in P'. If, queried for $\mathrm{sk}_\alpha(S_1) \in P'$, where S_1 is a length-ℓ substring of X, the data structure outputs $\mathrm{sk}_\alpha(S_2) \in Q'$, where S_2 is a length-ℓ substring of Y, then we compute $\|\mathrm{sk}_\alpha(S_1) - \mathrm{sk}_\alpha(S_2)\|^2$. If it is at most $(1+\varepsilon)k$, we output YES and the witness pair (S_1, S_2) of substrings. As the length of vectors in P', Q' is $d = \mathcal{O}(\log^{1+o(1)} n)$, we obtain the desired complexity.

To show that the algorithm is correct, suppose that there are length-ℓ substrings S_1 and S_2 of X and Y, respectively, with $d_H(S_1, S_2) \le k$. By Corollary 5, $\|\mathrm{sk}_\alpha(S_1), \mathrm{sk}_\alpha(S_2)\| \le \sqrt{(1+\alpha)k}$ holds with probability at least $1 - 1/n$. Then, when querying for $\mathrm{sk}_\alpha(S_1)$, with constant probability the data structure will output a string $\mathrm{sk}_\alpha(S_2')$ such that $\|\mathrm{sk}_\alpha(S_1) - \mathrm{sk}_\alpha(S_2')\|^2 \le (1+\varepsilon)(1-\alpha^2)k \le (1+\varepsilon)k$. Then, our algorithm will return YES.

On the other hand, if we output YES with a witness pair (S_1, S_2), then $\|\mathrm{sk}_\alpha(S_1) - \mathrm{sk}_\alpha(S_2)\|^2 \le (1+\varepsilon)k$ implies $d_H(S_1, S_2) \le (1+\varepsilon)k$ with high probability by Corollary 5. ◀

While this solution is very fast, it uses quite a lot of space. Furthermore, the data structure of [5] that we use as a black box applies highly non-trivial techniques. To overcome these two disadvantages, we will show a different solution based on a careful implementation of ideas first introduced in [4] that showed a data structure for approximate text indexing with mismatches. In [19], the authors developed these ideas further to show an algorithm that solves the LCS with Approximately k Mismatches problem in $\mathcal{O}(n^{1+1/(1+\varepsilon)})$ space and $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log^2 n)$ time for $\varepsilon \in (0, 2)$ with constant error probability. In this work, we significantly improve and simplify the approach to show the following result:

▶ **Theorem 12.** *Assume an alphabet of arbitrary size $\sigma = n^{\mathcal{O}(1)}$. The decision variant of LCS with Approximately k Mismatches can be solved in $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log^2 n)$ time and $\mathcal{O}(n)$ space. The answer is correct with constant probability.*

Let us defer the proof of the theorem until Section 3.1 and start by explaining how we use Lemma 11 and Theorem 12 and the Twenty Questions game to show Theorem 2.

Proof of Theorem 2. We will rely on the modified version of the Twenty Questions game that we described in Section 2.1. In our case, $A = \mathrm{LCS}_k(X, Y)$ and $B = \mathrm{LCS}_{(1+\varepsilon)k}(X, Y)$. For Carole, we use either the algorithm of Lemma 11, or the algorithm of Theorem 12, with an additional procedure verifying the witness pair (S_1, S_2) character by character to check that it indeed satisfies $d_H(S_1, S_2) \le (1+\varepsilon)k$. We output the longest pair of (honest) witness substrings found across all iterations. We will return a correct answer assuming that the fraction of errors is $\rho < \frac{1}{3}$. Recall that the algorithm solves the decision variant of the LCS with Approximately k Mismatches problem incorrectly with probability not exceeding a constant δ, and we can ensure $\delta < \frac{1}{3}$ by repeating it a constant number of times. It means that Carole can answer an individual question erroneously with probability less than $\frac{1}{3}$. Therefore, for a sufficiently large constant in the number of queries $Q = \Theta(\log n)$, the fraction of erroneous answers is $\rho < \frac{1}{3}$ with high probability by Chernoff–Hoeffding bounds. The claim of the theorem follows immediately from Lemma 11 and Theorem 12. ◀

3.1 Proof of Theorem 12

We first give an algorithm for the decision version of the LCS with Approximately k Mismatches problem that uses $\mathcal{O}(n \log n)$ space and $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log n + \sigma n \log^2 n)$ time, and then we improve the space and time complexity.

We assume to have fixed a Karp–Rabin fingerprinting function φ for a prime $q = \Omega(\max\{n^5, \sigma\})$ and an integer $r \in \mathbb{Z}_q$. With error probability inverse polynomial in n, we can find such q in $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ time; see [23, 3].

Let Π be the set of all projections of strings of length ℓ onto a single position, i.e., the value $\pi_i(S)$ of the i-th projection on a string S of length ℓ is simply its i-th character $S[i]$. More generally, for a length-ℓ string S and a function $h = (\pi_{a_1}, \dots, \pi_{a_m}) \in \Pi^m$, we define $h(S)$ as $S[a_1]S[a_2]\cdots S[a_m]$.

Let $p_1 = 1 - k/\ell$ and $p_2 = 1 - (1+\varepsilon)k/\ell$. We assume that $(1+\varepsilon)k < \ell$ in order to guarantee $p_1 > p_2 > 0$; the problem is trivial if $(1+\varepsilon)k \ge \ell$. Further, let $m = \lceil \log_{p_2} \frac{1}{n} \rceil$.

We choose a set $\mathcal{H}$ of $L = \Theta(n^{1/(1+\varepsilon)})$ hash functions in Π^m uniformly at random. Let $C_\ell^{\mathcal{H}}$ be the mutliset of all collisions of length-ℓ substrings of X and Y under the functions from $\mathcal{H}$, i.e. $C_\ell^{\mathcal{H}} = \{(X[i, i+\ell-1], Y[j, j+\ell-1], h) : \varphi(h(X[i, i+\ell-1])) = \varphi(h(Y[j, j+\ell-1])), 1 \le i \le |X| - \ell, 1 \le j \le |Y| - \ell\}$.

We will perform two tests. The first test chooses an arbitrary subset $C' \subseteq C_\ell^{\mathcal{H}}$ of size $|C'| = \min\{4nL, |C_\ell^{\mathcal{H}}|\}$ and, for each collision $(S_1, S_2, h) \in C'$, computes $\|\mathrm{sk}_\varepsilon(S_1) - \mathrm{sk}_\varepsilon(S_2)\|^2$. If this value is at most $(1+\varepsilon)k$, then the algorithm returns YES and the pair (S_1, S_2) as a witness. The second test chooses a collision $(S_1, S_2, h) \in C_\ell^{\mathcal{H}}$ uniformly at random and computes the Hamming distance between S_1 and S_2 character by character in $\mathcal{O}(\ell) = \mathcal{O}(n)$ time. If the Hamming distance is at most $(1+\varepsilon)k$, the algorithm returns YES and the witness pair (S_1, S_2). Otherwise, the algorithm returns NO. See Algorithm 2.

Algorithm 2 LCS with Approximately k Mismatches (decision variant).

```
1: Choose a set $\mathcal{H}$ of $L$ functions from $\Pi^m$ uniformly at random
2: $C_\ell^{\mathcal{H}} = \{(S_1, S_2, h) : S_1, S_2$ – length-$\ell$ substrings of $X, Y$ resp. and $\varphi(h(S_1)) = \varphi(h(S_2))\}$
3: Choose an arbitrary subset $C' \subseteq C_\ell^{\mathcal{H}}$ of size $\min\{4nL, |C_\ell^{\mathcal{H}}|\}$
4: Compute $\mathrm{sk}_\varepsilon(\cdot)$ sketches for all length-$\ell$ substrings of $X, Y$
5: for $(S_1, S_2, h) \in C'$ do
6:     if $\|\mathrm{sk}_\varepsilon(S_1) - \mathrm{sk}_\varepsilon(S_2)\|^2 \le (1+\varepsilon)k$ then return (YES, $(S_1, S_2)$)
7: Draw a collision $(S_1, S_2, h) \in C_\ell^{\mathcal{H}}$ uniformly at random
8: if $d_H(S_1, S_2) \le (1+\varepsilon)k$ then return (YES, $(S_1, S_2)$)
9: return NO
```

We must explain how we compute $C_\ell^{\mathcal{H}}$ and choose the collisions that we test. We consider each hash function $h \in \mathcal{H}$ in turn. Let $h = (\pi_{a_1}, \dots, \pi_{a_m})$. Recall that for a string S of length ℓ we define $h(S)$ as $S[a_1]S[a_2]\cdots S[a_m]$. Consequently, $\varphi(h(S)) = (\sum_{i=1}^m r^{i-1} S[a_i]) \bmod q$. We create a vector U of length ℓ where each entry is initialised with 0. For each i, we add $r^{i-1} \bmod q$ to the a_i-th entry of U. Finally, we run the FFT algorithm [12] for U and X, Y in the field $\mathbb{Z}_q$, and sort the resulting values. We obtain a list of sorted values that we can use to generate the collisions. Namely, consider some fixed value z. Assume that there are x substrings of X and y substrings of Y of length ℓ such that the fingerprint of their projection is equal to z. The value z then gives xy collisions, and we can generate each one of them in constant time. This explains how to choose the subset C' in $\mathcal{O}(nL \log n)$ time.

To draw a collision from $C_\ell^{\mathcal{H}}$ uniformly at random, we could simply compute the total number of collisions across all functions $h \in \mathcal{H}$, draw a number in $[1, |C_\ell^{\mathcal{H}}|]$, and generate the corresponding collision. However, this would require to generate the collisions twice. Instead, we use the weighted reservoir sampling algorithm [9]. We divide all collisions into subsets according to the values of fingerprints. We assume that the weighted reservoir sampling algorithm receives the fingerprint values one-by-one, as well as the number of corresponding collisions. At all times, the algorithm maintains a “reservoir” containing one fingerprint value and a random collision corresponding to this value. When a new value z with xy collisions arrives, the algorithm replaces the value in the reservoir with z and a random collision with some probability. Note that to select a random collision it suffices to choose a pair from $[1, x] \times [1, y]$ uniformly at random. It is guaranteed that if for a value z we have xy collisions, the algorithm will select z with probability $xy/|C_\ell^{\mathcal{H}}|$. Consequently, after processing all values, the reservoir will contain a collision chosen from $C_\ell^{\mathcal{H}}$ uniformly at random.

▶ **Lemma 13.** *Algorithm 2 uses $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log n + \sigma n \log^2 n)$ time and $\mathcal{O}(n \log n)$ space.*

Proof. Computing the sketches (Line 4) takes $\mathcal{O}(\sigma n \log^2 n)$ time and $\mathcal{O}(n \log n)$ space. Computing the collisions and choosing the collisions to test takes $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log n)$ time and $\mathcal{O}(n)$ space in total. Testing $\min\{4nL, |C_\ell^{\mathcal{H}}|\}$ collisions (Line 5) takes $\mathcal{O}(n^{1+1/(1+\varepsilon)} \log n)$ time and constant space. Computing the Hamming distance for a random collision (Line 8) takes $\mathcal{O}(\ell) = \mathcal{O}(n)$ time and constant space. ◀

▶ **Lemma 14.** *Let S_1 and S_2 be two length-ℓ substrings of X and Y, respectively, with $d_H(S_1, S_2) \le k$. If $L = \Theta(n^{1/(1+\varepsilon)})$ is large enough, then, with probability at least $3/4$, there exists a function $h \in \mathcal{H}$ such that $h(S_1) = h(S_2)$.*

Proof. Consider a function $h = (\pi_{a_1}, \dots, \pi_{a_m})$ drawn from Π^m uniformly at random. The probability of $h(S_1) = h(S_2)$ is at least p_1^m. Due to $p_1 \le 1$, we have

$$p_1^m = p_1^{\lceil \log_{p_2} \frac{1}{n} \rceil} \ge p_1^{1+\log_{p_2} \frac{1}{n}} = p_1 \cdot n^{-\frac{\log p_1}{\log p_2}}.$$

Moreover, $p_1 = 1 - \frac{k}{\ell}$ and $(1+\varepsilon)k < \ell$ yield $p_1 > 1 - \frac{1}{1+\varepsilon} = \frac{\varepsilon}{1+\varepsilon}$, whereas Bernoulli's inequality implies $p_2 = 1 - (1+\varepsilon)\frac{k}{\ell} \le (1 - \frac{k}{\ell})^{1+\varepsilon} = p_1^{1+\varepsilon}$, i.e., $\log p_2 \le (1+\varepsilon) \log p_1$. Therefore,

$$p_1^m \ge p_1 \cdot n^{-\frac{\log p_1}{\log p_2}} \ge \frac{\varepsilon}{1+\varepsilon} \cdot n^{-\frac{1}{1+\varepsilon}}.$$

Hence, we can choose the constant in $L = |\mathcal{H}|$ so that the claim of the lemma holds. ◀

▶ **Lemma 15.** *If $|C_\ell^{\mathcal{H}}| > 4nL$ and (S_1, S_2, h) is a uniformly random element of $C_\ell^{\mathcal{H}}$, then $\Pr[d_H(S_1, S_2) \ge (1+\varepsilon)k] \le \frac{1}{2}$.*

Proof. Consider length-ℓ substrings S_1, S_2 of X, Y, respectively, such that $d_H(S_1, S_2) \ge (1+\varepsilon)k$, and a hash function h. Let us bound the probability of $(S_1, S_2, h) \in C_\ell^{\mathcal{H}}$. There two possible cases: either $h(S_1) \ne h(S_2)$ but $\varphi(h(S_1)) = \varphi(h(S_2))$, or $h(S_1) = h(S_2)$. The probability of the first event is bounded by the collision probability of Karp–Rabin fingerprints, which is at most $1/n$. Let us now bound the probability of the second event. Since $d_H(S_1, S_2) \ge (1+\varepsilon)k$, we have $\Pr[h(S_1) = h(S_2)] \le p_2^m \le 1/n$, where the last inequality follows from the definition of m. Therefore, the probability that for some function $h \in \mathcal{H}$ we have $\varphi(h(S_1)) = \varphi(h(S_2))$ is at most $2/n$.

In total, we have $n^2|\mathcal{H}|$ possible triples (S_1, S_2, h) so by linearity of expectation, we conclude that the expected number of such triples is at most $\frac{2}{n} n^2 L = 2nL$. Therefore the probability to hit a triple (S_1, S_2, h) such that $d_H(S_1, S_2) \ge (1+\varepsilon)k$ when drawing from $C_\ell^{\mathcal{H}}$ uniformly at random is at most $2nL/|C_\ell^{\mathcal{H}}| \le 2nL/4nL = 1/2$. ◀

Below, we combine the previous results to prove that, with constant probability, Algorithm 2 correctly solves the decision variant of the LCS with Approximately 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.

▶ **Corollary 16.** *With non-zero constant probability, Algorithm 2 solves the decision variant of LCS with Approximately k Mismatches correctly.*

Proof. Suppose first that $\ell \le \mathrm{LCS}_k(X,Y)$, which means that there are two length-ℓ substrings S_1, S_2 of X, Y such that $d_H(S_1,S_2) \le k$. By Lemma 14, with probability at least 3/4, there exists a function $h \in \mathcal{H}$ such that $h(S_1) = h(S_2)$. In other words, $(S_1,S_2,h) \in C_\ell^{\mathcal{H}}$ with probability at least $\frac{3}{4}$. If $|C_\ell^{\mathcal{H}}| < 4nL$, we will find this triple and it will pass the test with probability at least $1 - n^{-6}$. If $|C_\ell^{\mathcal{H}}| \ge 4nL$, then by Lemma 15 the Hamming distance between S_1, S_2, where (S_1,S_2,h) was drawn from $C_\ell^{\mathcal{H}}$ uniformly at random, is at most $(1+\varepsilon)k$ with probability $\ge 1/2$, and therefore this pair will pass the test with probability $\ge 1/2$. It follows that in this case the algorithm outputs YES with constant probability.

Suppose now that $\ell > \mathrm{LCS}_{(1+\varepsilon)k}(X,Y)$. In this case, the Hamming distance between any pair of length-ℓ substrings of X and Y is at least $(1+\varepsilon)k$, so none of them will ever pass the second test and none of them will pass the first test with constant probability. ◀

We now improve the space of the algorithm to linear. Note that the only reason why we needed $\mathcal{O}(n \log n)$ space is that we precompute and store the sketches for the Hamming distance. Below we explain how to overcome this technicality.

First, we do not precompute the sketches. Second, we process the collisions in C' in batches of size n. Consider one of the batches, $\mathcal{B}$. For each collision $(S_1,S_2,h) \in \mathcal{B}$ we must compute $\|\mathrm{sk}_\varepsilon(S_1) - \mathrm{sk}_\varepsilon(S_2)\|^2$. We initialize a counter for every collision, setting it to zero initially. The number of rounds in the algorithm will be equal to the length of the sketches, and, in round i, the counter for a collision $(S_1,S_2,h) \in \mathcal{B}$ will contain the squared L_2 distance between the length-i prefixes of $\mathrm{sk}_\varepsilon(S_1)$ and $\mathrm{sk}_\varepsilon(S_2)$. In more detail, let $\mathcal{S}$ be the set of all substrings of X, Y that participate in the collisions in $\mathcal{B}$. Recall that all these substrings have length ℓ. At round i, we compute the i-th coordinate of the sketches of the substrings in $\mathcal{S}$. By definition, the i-th coordinate is the dot product of the i-th row of $c \cdot M$, where c and M are as in Corollary 5, and a substring encoded using μ. Hence, we can compute the coordinate using the FFT algorithm [12] in $\mathcal{O}(\sigma n \log n)$ time and $\mathcal{O}(n)$ space. When we have the coordinate i computed, we update the counters for the collisions and repeat.

At any time, the algorithm uses $\mathcal{O}(n)$ space. Compared to the time consumption proven in Lemma 13, the algorithm spends an additional $\mathcal{O}(\sigma n^{1+1/(1+\varepsilon)} \log^2 n)$ time for computing the coordinates of the sketches. Therefore, in total the algorithm uses $\mathcal{O}(\sigma n^{1+1/(1+\varepsilon)} \log^2 n) = \mathcal{O}(n^{1+1/(1+\varepsilon)} \log^2 n)$ time and $\mathcal{O}(n)$ space. For constant-size alphabets, this completes the proof of Theorem 12. For alphabets of arbitrary size, we replace the sketches from Section 2 with the sketches defined in [19] to achieve the desired complexity. We note that we could use the sketches [19] for small-size alphabets as well, but their lengths hide a large constant.

4 Experiments

We now present results of experimental evaluation of the second solution of Theorem 2.

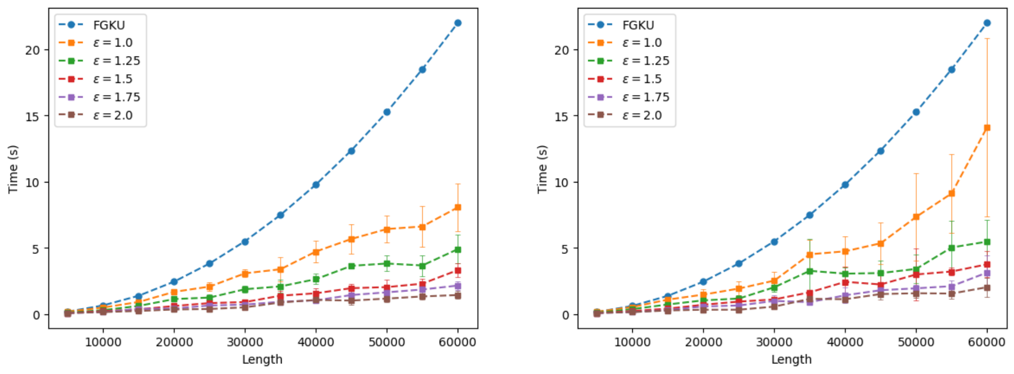
Methodology and test environment. The baselines and our solution are written in `C++11` and compiled with optimizations using gcc 7.4.0. The experimental results were generated on an Intel Xeon E5-2630 CPU using 128 GiB RAM. To ensure the reproducibility of our results, our complete experimental setup, including data files, is available at `https://github.com/fnareoh/LCS_Approx_k_mis`.

Baseline. The only other solution to the LCS with Approximately k Mismatches problem was presented in [19]. However, it has a worse complexity and is likely to be unpractical because it uses a very complex class of hash functions. We therefore chose to compare our algorithm against algorithms for the LCS with k Mismatches problem. To the best of our knowledge, none of the existing algorithms has been implemented. We implemented the solution to LCS with k Mismatches by Flouri et al., which we refer to as FGKU [13]. (The other algorithms seem too complex to be efficient in practice.) The main idea of the algorithm of Flouri et al. is that if we know that the longest common substring with k mismatches is obtained by a substring of X that starts at a position p and a substring of Y that starts at a position $p+i$, then we can find it by scanning X and $Y[i,|Y|]$ in linear time.

Details of implementation. We made several adjustments to the theoretical algorithm we described. First, we use the fact that $A = \mathrm{LCS}(X,Y) + k \leq \mathrm{LCS}_k(X,Y) \leq B = (k+1)\cdot \mathrm{LCS}(X,Y) + k$ to bound the interval in the Twenty Questions game. We also treated the number of questions in the Twenty Questions game and L, the size of the set of hash functions $\mathcal{H}$, as parameters that trade time for accuracy, and put the number of questions to $2\log(B-A)$ in the Twenty Questions game and $L = n^{1/(1+\varepsilon)}/16$. In Line 6 of Algorithm 2, we used sketches to estimate the Hamming distance. In practice, we computed the Hamming distance via character-by-character comparison when ℓ is small compared to k and via kangaroo jumps [14] otherwise. Also, when $\ell \leq 2\log n$ in Algorithm 2, we computed the hash values of the length-ℓ substrings of S_1 and S_2 naively, instead of using the FFT algorithm [12].

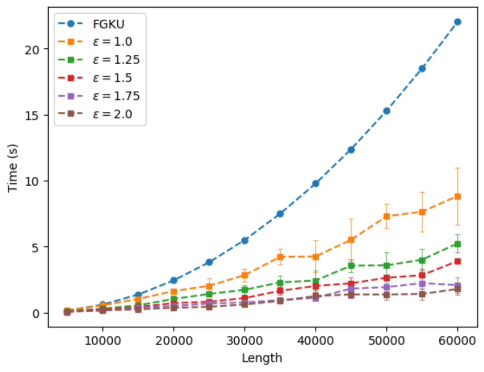
Data sets and results. We considered $k \in \{10, 25, 50\}$ and $\varepsilon \in \{1.0, 1.25, 1.5, 1.75, 2.0\}$. We tested the algorithms on pairs of random strings (each character is selected independently and uniformly from a four-character alphabet $\{A, T, G, C\}$) and on pairs of strings extracted at random from the E. coli genome. The lengths of the strings in each pair are equal and vary from 0 to 60000 with a step of 5000. All timings reported are averaged over ten runs. Figures 1–3 show the results for $k = 10, 25, 50$. We note that for $\varepsilon = 1$ and $k = 10, 25$, the standard deviation of the running time on the E. coli data set is quite large, which is probably caused by our choice of the method to compute the Hamming distance between substrings, but for all other parameter combinations it is within the standard range. We can see that the time decreases when ε grows, which is coherent with the theoretical complexity.

As for the accuracy, note that our algorithm cannot return a pair of strings at Hamming distance more than $(1+\varepsilon)k$, and so the only risk is returning strings which are too short. Consequently, we measured the accuracy of our implementation by the ratio of the length

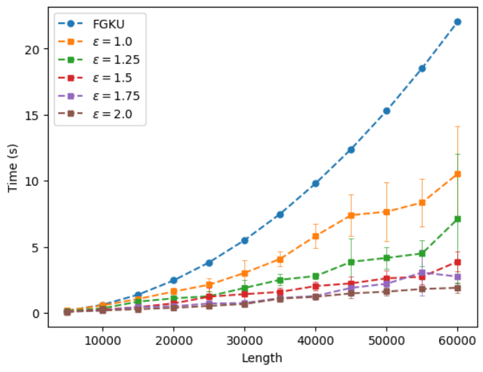


(a) Random, $k = 10$. **(b)** E. coli, $k = 10$.

Figure 1 Comparison of the FGKU algorithm versus our algorithm for $k = 10$ and different values of ε. Large standard deviation for length 60000 is caused by an outlier with very long longest common substring with k mismatches.

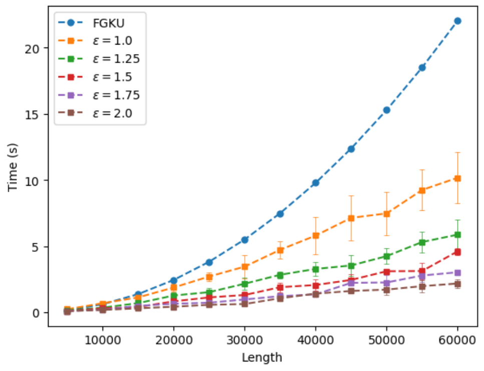


(a) Random, $k = 25$.

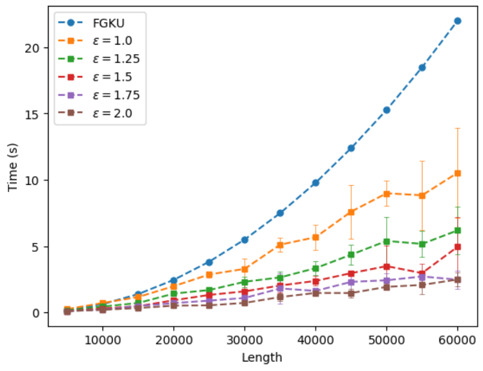


(b) E. coli, $k = 25$.

Figure 2 Comparison of the FGKU algorithm versus our algorithm for $k = 25$ and different values of ε.



(a) Random, $k = 50$.



(b) E. coli, $k = 50$.

Figure 3 Comparison of the FGKU algorithm versus our algorithm for $k = 50$ and different values of ε.

Table 1 Accuracy of the LCS with Approximately k Mismatches algorithm. For each k and ε, we show $r_{\min}(\varepsilon, k)$, $r_{\max}(\varepsilon, k)$, as well as the error rate.

	Random						E. coli					
	$k = 10$		$k = 25$		$k = 50$		$k = 10$		$k = 25$		$k = 50$	
$\varepsilon = 1.0$	0.95	1.41	1.12	1.46	1.27	1.54	0.89	1.34	0.94	1.48	0.97	1.59
	error = 3%		error = 0%		error = 0%		error = 33%		error = 13%		error = 3%	
$\varepsilon = 1.25$	0.97	1.47	1.15	1.63	1.44	1.78	0.88	1.48	0.98	1.56	0.99	1.73
	error = 1%		error = 0%		error = 0%		error = 28%		error = 5%		error = 3%	
$\varepsilon = 1.5$	1.05	1.57	1.37	1.76	1.55	1.91	0.88	1.45	0.96	1.67	0.99	1.89
	error = 0%		error = 0%		error = 0%		error = 17%		error = 3%		error = 3%	
$\varepsilon = 1.75$	1.02	1.69	1.46	1.86	1.72	2.12	0.88	1.58	0.95	1.84	1.02	2.15
	error = 0%		error = 0%		error = 0%		error = 17%		error = 2%		error = 0%	
$\varepsilon = 2.0$	1.10	1.72	1.59	2.00	1.89	2.24	0.91	1.77	1.01	2.10	1.00	2.19
	error = 0%		error = 0%		error = 0%		error = 9%		error = 0%		error = 1%	

$\mathrm{LCS}_{\tilde{k}}(X,Y)$ returned by our algorithm divided by $\mathrm{LCS}_k(X,Y)$ computed by the dynamic programming. We estimate $r_{\min}(\varepsilon,k) = \min_{X,Y}(\mathrm{LCS}_{\tilde{k}}(X,Y)/\mathrm{LCS}_k(X,Y))$ and $r_{\max}(\varepsilon,k) = \max_{X,Y}(\mathrm{LCS}_{\tilde{k}}(X,Y)/\mathrm{LCS}_k(X,Y))$ by computing $\mathrm{LCS}_{\tilde{k}}(X,Y)$ and $\mathrm{LCS}_k(X,Y)$ for 10 pairs of strings for each length from 5000 to 60000 with step of 5000, as well as the error rate, i.e., the percentage of experiments where $\mathrm{LCS}_{\tilde{k}}(X,Y) < \mathrm{LCS}_k(X,Y)$ (see Table 1). Not surprisingly, $r_{\min}$ and $r_{\max}$ grow as k and ε grow, while the error rate drops. Even though there is no theoretical upper bound on $r_{\max}$, the latter is at most 2.24 at all times. We also note that even in the cases when the error rate is non-negligible, $\mathrm{LCS}_{\tilde{k}} \geq 0.86 \cdot \mathrm{LCS}_k$; in other words, our algorithm returns a reasonable approximation of LCS_k.

5 Proof of Fact 3

We now show the lower bound of Fact 3 by a reduction from the $(1+\gamma)$-approximate Bichromatic Closest Pair problem.

▶ **Problem 17** ($(1+\gamma)$-approximate Bichromatic Closest Pair). *Given a constant $\gamma > 0$ and two sets of binary strings $(U_i)_{i\in[1,N]}$ and $(V_j)_{j\in[1,N]}$, each of length $d = \mathcal{O}(\log N)$, if the smallest Hamming distance between a pair $(U_i,V_j)_{i,j\in[1,N]}$ is h, we must output (possibly another) pair of binary strings (U_i,V_j) with Hamming distance in $[h,(1+\gamma)h]$.*

Rubinstein [22] proved that for every constant $\delta > 0$, there exists $\gamma = \gamma(\delta)$ such that any randomised algorithm that solves $(1+\gamma)$-approximate Bichromatic Closest Pair correctly with constant probability requires $\mathcal{O}(N^{2-\delta})$ time assuming SETH:

▶ **Hypothesis 18** (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 $m^{\mathcal{O}(1)}2^{(1-\delta)n}$ time even by a Monte-Carlo randomised algorithm (with error probability bounded by a small constant)*[2].

We show the lower bound by reducing a single instance of $(1+\gamma)$-approximate Bichromatic Closest Pair to a polylogarithmic number of instances of LCS with Approximately k Mismatches. We assume that U_i, V_j are over the alphabet $\{0,1\}$. Let us introduce a string $H = (\mathtt{a}^d\mathtt{b})^{d+1}$ and construct $X = HU_1HU_2H\dots HU_NH$ and $Y = HV_1HV_2H\dots HV_NH$.

▶ **Observation 19.** *For every integer $k \geq 0$, if there exist $i,j \in [1,N]$ such that $d_H(U_i,V_j) \leq k$, then $\mathrm{LCS}_k(X,Y) \geq 2(d+1)^2 + d$.*

Proof. If $d_H(U_i,V_j) \leq k$ for some i,j, then $d_H(HU_iH,HV_jH) \leq k$ and $\mathrm{LCS}_k(X,Y) \geq |HU_iH| = 2(d+1)^2 + d$. ◀

▶ **Lemma 20.** *For every integer $0 \leq k \leq d$, if $\mathrm{LCS}_k(X,Y) \geq 2(d+1)^2 + d$, then there exist $i,j \in [1,N]$ such that $d_H(U_i,V_j) \leq k$.*

Proof. By the assumption of the lemma, there exist substrings S_1 and S_2 of X and Y, respectively, with $|S_1| = |S_2| \geq 2(d+1)^2 + d$ and $d_H(S_1,S_2) \leq k$. The substring S_2 contains either HV_j or V_jH for some j. Without loss of generality, we can assume that S_2 contains a copy of H followed by V_j for some j. Let us consider the substring S of X aligned with the copy of H in S_2. Below we will prove that $S = H$, and since S is followed by U_i for some i, this will imply that $d_H(HU_iH,HV_jH) \leq k$.

[2] Impagliazzo, Paturi, and Zane [16] stated the hypothesis for deterministic algorithms only, but nowadays it is common to extend SETH to allow randomisation. If we condition on the classic version of the hypothesis, we will obtain a lower bound for deterministic algorithms. See [27] for more discussion.

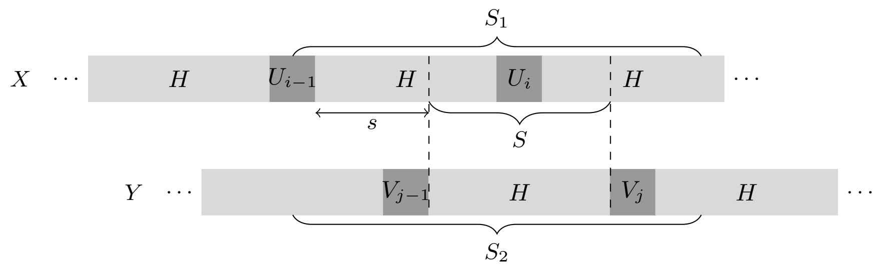


Figure 4 Substrings S_1 and S_2 of X and Y, respectively, substring S aligned with a copy of H in S_2, and the shift s.

Suppose that $S \neq H$, and let $0 < s < (d+1)^2 + d$ be the distance between the starting positions of S and the nearest copy of H from the left. If $s < d+1$ or $(d+1)^2 < s$, then each occurrence of $\mathtt{b}$ in H creates a mismatch. There are $d+1 > k$ of them, a contradiction. If $d+1 \leq s \leq (d+1)^2$, then S contains U_i, creating d mismatches with H. Since $|U_i| = d$ and $|H| = (d+1)^2$, we will have at least one more mismatch from the alignment of the copy of H in Y and the copies of H in X that surround U_i. Therefore, in total there are at least $d+1 > k$ mismatches, a contradiction. To conclude, both cases are impossible, and hence $s = 0$. The lemma follows as explained above. ◀

With this lemma, we can now proceed to prove Fact 3. Define $\varepsilon = \gamma/3$ and consider all $k = 1, (1+\varepsilon), (1+\varepsilon)^2, (1+\varepsilon)^3, \ldots$ until $d/(1+\varepsilon)$. For each k, we run $\log\log n$ independent instances of an algorithm for LCS with Approximately k Mismatches. Let k_0 be the smallest k such that the identified longest common substring with approximately k mismatches has length at least $2(d+1)^2 + d$.

By the definition of k_0, Observation 19 and Lemma 20, there do not exist $i, j \in [1, N]$ such that $d_H(U_i, V_j) \leq k_0/(1+\varepsilon)$, but there exist $i, j \in [1, N]$ such that $d_H(U_i, V_j) \leq k_0(1+\varepsilon)$. In the $(1+\gamma)$-approximate Bichromatic Closest Pair problem, this translates to $k_0/(1+\varepsilon) < h \leq k_0(1+\varepsilon)$, where h is the minimal distance between all pairs U_i, V_j. This is equivalent to

$$h \leq k_0(1+\varepsilon) < h(1+\varepsilon)^2 = h(1 + \tfrac{2}{3}\gamma + \tfrac{1}{9}\gamma^2) \leq h(1+\gamma),$$

which means that the pair (U_i, V_j) found by the algorithm for k_0 is a valid solution for $(1+\gamma)$-approximate Bichromatic Closest Pair. It follows that, for some k, the algorithm for LCS with Approximately k Mismatches must spend $\Omega(N^{2-\delta}/\log_{1+\varepsilon}\log N)$ time. We have $n = |X| = |Y| = \mathcal{O}(d^2 N) = \mathcal{O}(N \log^2 N)$, which implies $N = \Omega(n/\log^2 n)$. Fact 3 follows.

References

1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *SODA'15*, pages 218–230, 2015. `doi:10.1137/1.9781611973730.17`.

2 Dimitris Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. `doi:10.1016/S0022-0000(03)00025-4`.

3 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. `doi:10.4007/annals.2004.160.781`.

4 Alexandr Andoni and Piotr Indyk. Efficient algorithms for substring near neighbor problem. In *SODA'06*, pages 1203–1212, 2006. `doi:10.1145/1109557.1109690`.

5 Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *STOC'15*, pages 793–801, 2015. doi:10.1145/2746539.2746553.

6 Maxim Babenko and Tatiana Starikovskaya. Computing the longest common substring with one mismatch. *Problems of Information Transmission*, 47(1):28–33, 2011. doi:10.1134/S0032946011010030.

7 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. doi:10.1137/15M1053128.

8 Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *FOCS'18*, pages 979–990, 2018. doi:10.1109/FOCS.2018.00096.

9 Min-Te Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, December 1982. doi:10.2307/2336002.

10 Panagiotis Charalampopoulos, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Linear-time algorithm for long LCF with k mismatches. In *CPM'18*, pages 23:1–23:16, 2018. doi:10.4230/LIPIcs.CPM.2018.23.

11 Aditi Dhagat, Peter Gács, and Peter Winkler. On playing "Twenty questions" with a liar. In *SODA'92*, pages 16–22, 1992. URL: http://dl.acm.org/citation.cfm?id=139404.139409.

12 Michael J. Fischer and Michael S. Paterson. String matching and other products. In *Complexity of Computation*, pages 113–125, 1974.

13 Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643–647, 2015.

14 Zvi Galil and Raffaele Giancarlo. Improved string matching with k mismatches. *SIGACT News*, 17(4):52–54, March 1986. doi:10.1145/8307.8309.

15 Szymon Grabowski. A note on the longest common substring with k-mismatches problem. *Information Processing Letters*, 115(6-8):640–642, 2015.

16 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

17 William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, 1984.

18 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987. doi:10.1147/rd.312.0249.

19 Tomasz Kociumaka, Jakub Radoszewski, and Tatiana Starikovskaya. Longest common substring with approximately k mismatches. *Algorithmica*, 81(6):2633–2652, 2019. doi:10.1007/s00453-019-00548-x.

20 Chris-Andre Leimeister and Burkhard Morgenstern. kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics*, 30(14):2000–2008, 2014. doi:10.1093/bioinformatics/btu331.

21 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS'09*, pages 315–323, 2009. doi:10.1109/FOCS.2009.11.

22 Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *STOC'18*, pages 1260–1268, 2018. doi:10.1145/3188745.3188916.

23 Terence Tao, Ernest Croot III, and Harald Helfgott. Deterministic methods to find primes. *AMS Mathematics of Computation*, 81(278):1233–1246, 2012. doi:10.1090/S0025-5718-2011-02542-1.

24 Sharma V. Thankachan, Alberto Apostolico, and Srinivas Aluru. A provably efficient algorithm for the k-mismatch average common substring problem. *Journal of Computational Biology*, 23(6):472–482, 2016. doi:10.1089/cmb.2015.0235.

25 Sharma V. Thankachan, Sriram P. Chockalingam, Yongchao Liu, Alberto Apostolico, and Srinivas Aluru. ALFRED: A practical method for alignment-free distance computation. *Journal of Computational Biology*, 23(6):452–460, 2016. doi:10.1089/cmb.2015.0217.

26 Sharma V. Thankachan, Sriram P. Chockalingam, Yongchao Liu, Ambujam Krishnan, and Srinivas Aluru. A greedy alignment-free distance estimator for phylogenetic inference. *BMC Bioinformatics*, 18(8):238, June 2017. doi:10.1186/s12859-017-1658-0.

27 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *ICM'18*, pages 3447–3487, 2018. doi:10.1142/9789813272880_0188.