Linear-Time Algorithm for Long LCF with k Mismatches

Panagiotis Charalampopoulos
Department of Informatics, King's College London, London, UK
panagiotis.charalampopoulos@kcl.ac.uk
https://orcid.org/0000-0002-6024-1557

Maxime Crochemore
Department of Informatics, King's College London, London, UK
maxime.crochemore@kcl.ac.uk
https://orcid.org/0000-0003-1087-1419

Costas S. Iliopoulos
Department of Informatics, King's College London, London, UK
costas.iliopoulos@kcl.ac.uk

Tomasz Kociumaka
Institute of Informatics, University of Warsaw, Warsaw, Poland
kociumaka@mimuw.edu.pl
https://orcid.org/0000-0002-2477-1702

Solon P. Pissis
Department of Informatics, King's College London, London, UK
solon.pissis@kcl.ac.uk
https://orcid.org/0000-0002-1445-1932

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Warsaw, Poland
jrad@mimuw.edu.pl
https://orcid.org/0000-0002-0067-6401

Wojciech Rytter
Institute of Informatics, University of Warsaw, Warsaw, Poland
rytter@mimuw.edu.pl

Tomasz Waleń
Institute of Informatics, University of Warsaw, Warsaw, Poland
walen@mimuw.edu.pl
https://orcid.org/0000-0002-7369-3309

Abstract

In the Longest Common Factor with k Mismatches (LCF_k) problem, we are given two strings X and Y of total length n, and we are asked to find a pair of maximal-length factors, one of X and the other of Y, such that their Hamming distance is at most k. Thankachan et al. [27] show that this problem can be solved in $\mathcal{O}(n \log^k n)$ time and $\mathcal{O}(n)$ space for constant k. We consider the $\text{LCF}_k(\ell)$ problem in which we assume that the sought factors have length at least ℓ. We use difference covers to reduce the $\text{LCF}_k(\ell)$ problem with $\ell = \Omega(\log^{2k+2} n)$ to a task involving $m = \mathcal{O}(n/\log^{k+1} n)$ *synchronized* factors. The latter can be solved in $\mathcal{O}(m \log^{k+1} m)$ time, which results in a linear-time algorithm for $\text{LCF}_k(\ell)$ with $\ell = \Omega(\log^{2k+2} n)$. In general, our solution to the $\text{LCF}_k(\ell)$ problem for arbitrary ℓ takes $\mathcal{O}(n + n \log^{k+1} n/\sqrt{\ell})$ time.

2012 ACM Subject Classification Theory of computation → Pattern matching

29th Annual Symposium on Combinatorial Pattern Matching (CPM 2018).
Editors: Gonzalo Navarro, David Sankoff, and Binhai Zhu; Article No. 23; pp. 23:1–23:16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Keywords and phrases longest common factor, longest common substring, Hamming distance, heavy-light decomposition, difference cover

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.23

Related Version A full version of the paper is available at `http://arxiv.org/abs/1802.06369`.

Funding Jakub Radoszewski is 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. Wojciech Rytter and Tomasz Waleń are supported by the Polish National Science Center, grant no. 2014/13/B/ST6/00770.

1 Introduction

The longest common factor (LCF) problem is a classical and well-studied problem in theoretical computer science. It consists in finding a maximal-length factor of a string X occurring in another string Y. When X and Y are over a linearly-sortable alphabet, the LCF problem can be solved in the optimal $\mathcal{O}(n)$ time and space [17, 15], where n is the total length of X and Y. Considerable efforts have thus been made on improving the *additional* working space; namely, the space required for computations, not taking into account the space providing read-only access to X and Y. We refer the interested reader to [25, 21].

In many bioinformatics applications and elsewhere, it is relevant to consider potential alterations within the pair of input strings (e.g. DNA sequences). It is thus natural to define the LCF problem under a distance metric model. The problem then consists in finding a pair of maximal-length factors of X and Y whose distance is at most k. In fact, this problem has received much attention recently, in particular due to its applications in alignment-free sequence comparison [29, 22].

Under the Hamming distance model, the problem is known as the Longest Common Factor with at most k Mismatches (LCF_k) problem. The restricted case of $k = 1$ was first considered in [4], where an $\mathcal{O}(n^2)$-time and $\mathcal{O}(n)$-space solution was given. It was later improved by Flouri et al. [12], who built heavily on a technique by Crochemore et al. [11] to obtain $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.

For a general value of k, the problem can be solved in $\mathcal{O}(n^2)$ time and space by a dynamic programming algorithm, but more efficient solutions have been devised. Leimeister and Morgenstern [22] first suggested a greedy heuristic algorithm. Flouri et al. [12] proposed an $\mathcal{O}(n^2)$-time algorithm that uses $\mathcal{O}(1)$ additional space. Grabowski [13] presented two algorithms with running times $\mathcal{O}(n((k+1)(\ell_0+1))^k)$ and $\mathcal{O}(n^2k/\ell_k)$, where ℓ_0 and ℓ_k are, respectively, the length of an LCF of X and Y and the length of an LCF of X and Y with at most k mismatches. Thankachan et al. [27] proposed an $\mathcal{O}(n \log^k n)$-time and $\mathcal{O}(n)$-space algorithm (for any constant k).

Abboud et al. [1] employed the polynomial method to obtain a $k^{1.5}n^2/2^{\Omega(\sqrt{\frac{\log n}{k}})}$-time randomized algorithm. Kociumaka et al. [20] showed that a strongly subquadratic-time algorithm for the LCF_k problem, for binary strings and $k = \Omega(\log n)$, refutes the Strong Exponential Time Hypothesis [19, 18]. Thus, subquadratic-time solutions for approximate variants of the problem have been developed [20, 24]. The average-case complexity of this problem has also been considered [28, 2, 3].

1.1 Our Contribution

We consider the following variant of the Longest Common Factor with at most k Mismatches problem in which the result is constrained to have at least a given length. Let $\mathsf{LCF}_k(X, Y)$ denote the length of the longest common factor of X and Y with at most k mismatches.

> LCF of Length at Least ℓ with at most k Mismatches ($\mathrm{LCF}_k(X, Y, \ell)$)
> **Input:** Two strings X and Y of total length n and integers $k \geq 0$ and $\ell \geq 1$
> **Output:** $\mathsf{LCF}_k(X, Y)$ if it is at least ℓ, and "NONE" otherwise.

We focus on a special case of this problem with $\ell = \Omega(\log^{2k+2} n)$. Apart from its theoretical interest, solutions to the $\mathrm{LCF}_k(X, Y, \ell)$ problem may prove to be useful from a practical standpoint. The LCF_k length has been used as a measure of sequence similarity [29, 22]. It is thus assumed that similar sequences share relatively long factors with k mismatches.

We show an $\mathcal{O}(n)$-time algorithm for the $\mathrm{LCF}_k(X, Y, \ell)$ problem with $\ell = \Omega(\log^{2k+2} n)$. Moreover, we prove that the $\mathrm{LCF}_k(X, Y, \ell)$ problem can be solved in $\mathcal{O}(n + n \log^{k+1} n/\sqrt{\ell})$ time for arbitrary ℓ and constant k. In the final section we discuss the complexity for $k = \mathcal{O}(\log n)$. This unveils that the $\mathcal{O}(\cdot)$ notation hides a multiplicative factor that is actually subconstant in k.

For simplicity, we only describe how to compute the length $\mathsf{LCF}_k(X, Y)$. It is straightforward to amend our solution so that it extracts the corresponding factors of X and Y.

Toolbox. We use the following algorithmic tools:

- Difference covers (see, e.g., [23, 8]) let us reduce the $\mathrm{LCF}_k(X, Y, \ell)$ problem to searching for longest common prefixes and suffixes with at most k mismatches (LCP_k, LCS_k) at positions belonging to sets A in X and B in Y such that $|A|, |B| = \mathcal{O}(n/\sqrt{\ell})$.
- We use a technique of recursive heavy-path decompositions by Cole et al. [9], already applied in the context of the LCF_k problem by Thankachan et al. [27], to reduce computing LCP_k, LCS_k to computing LCP, LCS in sets of modified prefixes and suffixes starting at positions in A and B. Modifications consist in at most k changes and increase the size of the problem by a factor of $\mathcal{O}(\log^k n)$. We adjust the original technique of Cole et al. [9] so that all modified strings are stored in one compacted trie.
- Finally we apply to the compacted trie a solution to a problem on colored trees that is the cornerstone of the previous $\mathcal{O}(n \log n)$-time solution for the LCF_1 problem by Flouri et al. [12] (and originates from efficient merging of AVL trees [7]).

In total we arrive at $\mathcal{O}(n + n \log^{k+1} n/\sqrt{\ell})$ time complexity for the $\mathrm{LCF}_k(X, Y, \ell)$ problem.

2 Preliminaries

Henceforth we denote the input strings by X and Y and their common length by n.

The i-th letter of a string U, for $1 \leq i \leq |U|$, is denoted by $U[i]$. By $[i \,..\, j]$ we denote the integer interval $\{i, \ldots, j\}$ and by $U[i \,..\, j]$ we denote the string $U[i] \ldots U[j]$ that we call a factor of U. For simplicity, we denote $U[..\, i] = U[1 \,..\, i]$ and $U[i \,..] = U[i \,..\, |U|]$. By U^R we denote the mirror image of U.

For a pair of strings U and V such that $|U| = |V|$, we define their Hamming distance as $d_H(U, V) = |\{1 \leq i \leq |U| \,:\, U[i] \neq V[i]\}|$. For two strings U, V and a non-negative integer d, we define

$$\mathsf{LCP}_d(U, V) \;=\; \max\{p \leq |U|, |V| \,:\, d_H(U[1 \,..\, p], V[1 \,..\, p]) \leq d\}.$$

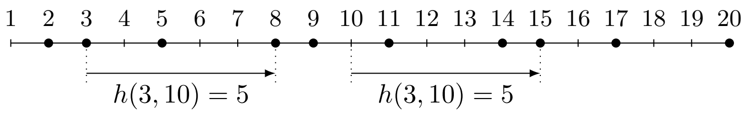

Figure 1 An example of a 6-cover $\mathbf{S}_{20}(6) = \{2, 3, 5, 8, 9, 11, 14, 15, 19, 20\}$, with the elements marked as black circles. For example, we may have $h(3, 10) = 5$ since $3 + 5,\ 10 + 5 \in \mathbf{S}_{20}(6)$.

Let T be the trie of a collection of strings $\mathcal{F}$. The *compacted trie* of $\mathcal{F}$, $\mathcal{T}(\mathcal{F})$, contains the root, the branching nodes, and the terminal nodes of T. Each edge of the compacted trie may represent several edges of $\mathcal{T}$ and is labeled by a factor of one of the strings F_i, stored in $\mathcal{O}(1)$ space. The edges outgoing from a node are labeled by the first letter of the respective strings. The size of a compacted trie is $\mathcal{O}(|\mathcal{F}|)$. The best-known example of a compacted trie is the suffix tree of a string; see [10].

2.1 Difference covers

We say that a set $\mathbf{S}(d) \subseteq \mathbb{Z}_+$ is a *d-cover* if there is a constant-time computable function h such that for $i, j \in \mathbb{Z}_+$ we have $0 \le h(i, j) < d$ and $i + h(i, j), j + h(i, j) \in \mathbf{S}(d)$ (see Figure 1). The following fact synthesizes a well-known construction implicitly used in [8], for example.

▶ **Fact 1** ([23, 8]). *For each $d \in \mathbb{Z}_+$ there is a d-cover $\mathbf{S}(d)$ such that $\mathbf{S}_n(d) := \mathbf{S}(d) \cap [1\,..\,n]$ is of size $\mathcal{O}(\frac{n}{\sqrt{d}})$ and can be constructed in $\mathcal{O}(\frac{n}{\sqrt{d}})$ time.*

2.2 Colored Trees Problem

As a component of our solution we use the following problem for colored trees:

Colored Trees Problem
Input: Two trees T_1 and T_2 containing blue and red leaves such that each internal node is branching (except for, possibly, the root). Each leaf has a number between 1 and m. Each tree has at most one red leaf and at most one blue leaf with a given number. The nodes of T_1 and T_2 are weighted such that children are at least as heavy as their parent.
Output: A node v_1 of T_1 and a node v_2 of T_2 with maximum total weight such that v_1 and v_2 have at least one blue leaf of the same number and at least one red leaf of the same number in their subtrees.

This abstract problem lies at the heart of the algorithm of Flouri et al. [12] for the Longest Common Factor with 1 Mismatch problem. They solve it in $\mathcal{O}(m \log m)$ time applying a solution inspired by an algorithm of Crochemore et al. [11] finding the longest repeat with a block of k don't cares, which, in turn, is based on the fact that two AVL trees can be merged efficiently [7].

▶ **Fact 2** ([11, 12]). Colored Trees Problem *can be solved in $\mathcal{O}(m \log m)$ time.*

In our solution we actually use the following problem related to families of strings represented on a compacted trie. It reduces to the Colored Trees Problem.

Two String Families LCP Problem
Input: A compacted trie $\mathcal{T}(\mathcal{F})$ of a family of strings $\mathcal{F}$ and two sets $\mathcal{P}, \mathcal{Q} \subseteq \mathcal{F}^2$
Output: The value maxPairLCP$(\mathcal{P}, \mathcal{Q})$, defined as
maxPairLCP$(\mathcal{P}, \mathcal{Q}) = \max\{\mathsf{LCP}(P_1, Q_1) + \mathsf{LCP}(P_2, Q_2) : (P_1, P_2) \in \mathcal{P}$ and $(Q_1, Q_2) \in \mathcal{Q}\}$

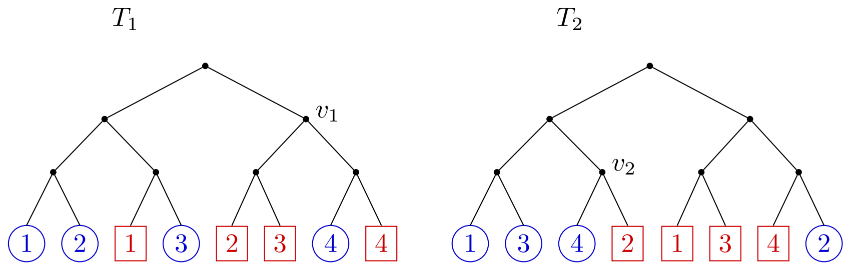

Figure 2 Example instance for COLORED TREES PROBLEM. Assuming that each node has weight equal to the distance from the root, the optimal solution is a pair of nodes (v_1, v_2) as shown in the figure. Both v_1 and v_2 have as a descendant a blue leaf with number 4 and a red leaf with number 2.

▶ **Lemma 3.** *The* TWO STRING FAMILIES LCP PROBLEM *can be solved in* $\mathcal{O}(|\mathcal{F}| + N \log N)$ *time, where* $N = |\mathcal{P}| + |\mathcal{Q}|$.

Proof. First, we create two copies $\mathcal{T}_1$ and $\mathcal{T}_2$ of the tree $\mathcal{T}(\mathcal{F})$, removing the edge labels but preserving the node weights $w(v)$ equal to the sum of lengths of edges on the path to the root.

Next, for each $(P_1, P_2) \in \mathcal{P}$ we attach a blue leaf to the terminal node of $\mathcal{T}_1$ representing P_1 and to the terminal of $\mathcal{T}_2$ representing P_2. We label these two blue leaves with a unique label, denoted here $L_{\mathcal{P}}(P_1, P_2)$. Similarly, for each $(Q_1, Q_2) \in \mathcal{Q}$, we attach red leaves to the terminal node of $\mathcal{T}_1$ representing Q_1 and the terminal node of $\mathcal{T}_2$ representing Q_2. We label these two red leaves with a unique label $L_{\mathcal{Q}}(Q_1, Q_2)$. Finally, in both $\mathcal{T}_1$ and $\mathcal{T}_2$ we remove all nodes which do not contain any colored leaf in their subtrees and dissolve all nodes with exactly one child (except for the roots). This way, each tree $\mathcal{T}_i$ contains $\mathcal{O}(|\mathcal{P}| + |\mathcal{Q}|)$ nodes, including $|\mathcal{P}| + |\mathcal{Q}|$ leaves, each with a distinct label.

Observe that for $(P_1, P_2) \in \mathcal{P}$, $(Q_1, Q_2) \in \mathcal{Q}$, and $j \in \{1, 2\}$, the value $\mathsf{LCP}(P_j, Q_j)$ is the weight of the lowest common ancestor (LCA) in $\mathcal{T}_j$ of the two leaves with labels $L_{\mathcal{P}}(P_1, P_2)$ and $L_{\mathcal{Q}}(Q_1, Q_2)$. Consequently, our task can be formulated as follows: Find a pair of internal nodes $v_1 \in \mathcal{T}_1$ and $v_2 \in \mathcal{T}_2$ of maximal total weight $w(v_1) + w(v_2)$ so that the subtrees rooted at v_1 and v_2 contain blue leaves with the same label and red leaves with the same label. This is exactly the COLORED TREES PROBLEM that can be solved in $\mathcal{O}(m \log m)$ time, where $m = |\mathcal{T}_1| + |\mathcal{T}_2| = \mathcal{O}(|\mathcal{P}| + |\mathcal{Q}|)$ (Fact 2). ◀

3 Reduction of $\mathsf{LCF}_k(\ell)$ problem to multiple synchronized LCP_k's

Let U be a string of length n. We denote:

$$\mathsf{Pairs}_\ell(U) = \{((U[..i-1])^R, U[i..]) \; : \; i \in \mathbf{S}_n(\ell)\}.$$

Observe that $|\mathsf{Pairs}_\ell(U)| = |\mathbf{S}_n(\ell)| = \mathcal{O}(n/\sqrt{\ell})$.

▶ **Lemma 4.** *If* $\mathsf{LCF}_k(X, Y) \geq \ell$, *then*

$$\mathsf{LCF}_k(X, Y) = \max_{p+q=k} \{\mathsf{LCP}_p(U_1, V_1) + \mathsf{LCP}_q(U_2, V_2) \; : \; (U_1, U_2) \in \mathsf{Pairs}_\ell(X), \, (V_1, V_2) \in \mathsf{Pairs}_\ell(Y)\}.$$

Proof. First, assume that $(U_1, U_2) \in \mathsf{Pairs}_\ell(X)$, $(V_1, V_2) \in \mathsf{Pairs}_\ell(Y)$, and $k = p + q$. Let $\tilde{U}_1$ and $\tilde{V}_1$ be prefixes of U_1 and V_1 (respectively) of length $\mathsf{LCP}_p(U_1, V_1)$, and let $\tilde{U}_2$ and $\tilde{V}_2$ be

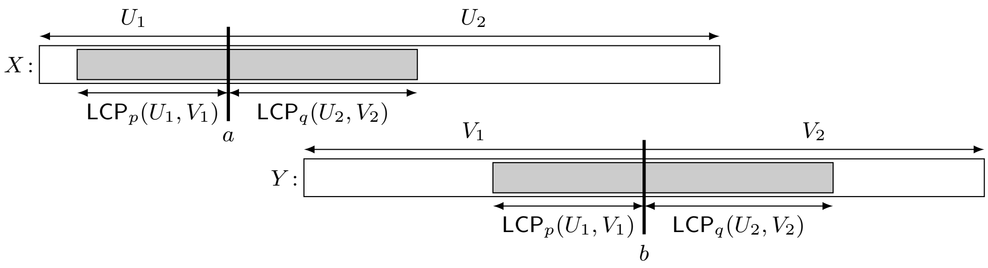

Figure 3 If $\mathsf{LCF}_k(X,Y) \geq \ell$, then there exist $(U_1,U_2) \in \mathsf{Pairs}_\ell(X)$ and $(V_1,V_2) \in \mathsf{Pairs}_\ell(Y)$ such that $\mathsf{LCF}_k(X,Y) = \mathsf{LCP}_p(U_1,U_2) + \mathsf{LCP}_q(V_1,V_2)$ for some $p+q=k$.

prefixes of U_2 and V_2 (respectively) of length $\mathsf{LCP}_q(U_2,V_2)$. Observe that $\tilde{U}_1^R\tilde{U}_2$ is a factor of $X = U_1^R U_2$ and $\tilde{V}_1^R\tilde{V}_2$ is a factor of $Y = V_1^R V_2$. Moreover,

$$d_H(\tilde{U}_1^R\tilde{U}_2, \tilde{V}_1^R\tilde{V}_2) = d_H(\tilde{U}_1,\tilde{U}_2) + d_H(\tilde{V}_1,\tilde{V}_2) \leq p+q = k.$$

Consequently,

$$\mathsf{LCF}_k(X,Y) \geq |\tilde{U}_1^R\tilde{U}_2| = |\tilde{U}_1| + |\tilde{U}_2| = \mathsf{LCP}_p(U_1,V_1) + \mathsf{LCP}_q(U_2,V_2).$$

This concludes the proof of the lower bound on $\mathsf{LCF}_k(X,Y)$.

Next, let $X[i\,..\,i']$ and $Y[j\,..\,j']$ be an optimal pair of factors; see Figure 3. They satisfy

$$|X[i\,..\,i']| = |Y[j\,..\,j']| = \mathsf{LCF}_k(X,Y) \geq \ell \quad \text{and} \quad d_H(X[i\,..\,i'], Y[j\,..\,j']) \leq k.$$

Denote $a = i + h(i,j)$ and $b = j + h(i,j)$, where h is the shift function associated with the l-cover $\mathbf{S}(\ell)$. Note that $a \in [i\,..\,i'] \cap \mathbf{S}(\ell)$ and $b \in [j\,..\,j'] \cap \mathbf{S}(\ell)$. Consequently, $(U_1,U_2) := ((X[..\,a-1])^R, X[a\,..]) \in \mathsf{Pairs}_\ell(X)$ and $(V_1,V_2) := ((Y[..\,b-1])^R, Y[b\,..]) \in \mathsf{Pairs}_\ell(Y)$. Moreover,

$$k \geq d_H(X[i\,..\,i'], Y[j\,..\,j']) = d_H(X[i\,..\,a-1], Y[j\,..\,b-1]) + d_H(X[a\,..\,i'], Y[b\,..\,j']).$$

Therefore, for $p = d_H(X[i\,..\,a-1], Y[j\,..\,b-1])$ and $q = k-p$, we have

$$\mathsf{LCP}_p(U_1,V_1) + \mathsf{LCP}_q(V_2,U_2) \geq |X[i\,..\,a-1]| + |X[a\,..\,i']| = |X[i\,..\,i']| = \mathsf{LCF}_k(X,Y).$$

This concludes the proof. ◀

4 The case of $k=0$ and of $k=1$ and $\sigma=2$

In this section, as a warm-up, we show how the Two String Families LCP Problem can be used to solve two special cases of the $\text{LCF}_k(X,Y,\ell)$ problem. Then in Section 6 we explain how it can be used to solve the problem in full generality.

In order to solve the $\text{LCF}_k(X,Y,\ell)$ problem for $k=0$, we observe that, by Lemma 4, if $\mathsf{LCF}_0(X,Y) \geq \ell$, then $\mathsf{LCF}_0(X,Y) = \text{maxPairLCP}(\mathsf{Pairs}_\ell(X), \mathsf{Pairs}_\ell(Y))$. Thus, we simply build the joint suffix tree $\mathcal{T}$ of X, Y, X^R, and Y^R, and we solve the appropriate instance of Two String Families LCP Problem.

The preprocessing time is clearly $\mathcal{O}(n)$, while solving the Two String Families LCP Problem takes $\mathcal{O}(n + n\log n/\sqrt{\ell})$ time, which is $\mathcal{O}(n)$ provided that $\ell = \Omega(\log^2 n)$.

For $k \geq 1$, we would ideally like to extend the family $\mathsf{Pairs}_\ell(S)$ to $\mathsf{Pairs}_\ell^{(k)}(S)$ replacing the suffixes and reversed prefixes of S with their approximate copies so that

$$\mathsf{LCF}_k(X, Y) = \text{maxPairLCP}(\mathsf{Pairs}_\ell^{(k)}(X), \mathsf{Pairs}_\ell^{(k)}(Y)).$$

A very naive solution would be to extend the alphabet Σ to $\Sigma_\$$ adding a symbol $\$ \notin \Sigma$, and for each $(S_1, S_2) \in \mathsf{Pairs}_\ell(S)$ to replace an arbitrary subset of k symbols with \$'s. However, this results in $\binom{n}{k}$ copies of each $(S_1, S_2) \in \mathsf{Pairs}_\ell(S)$, which is by far too much.

Our approach is therefore based on the technique of Cole et al. [9], which has already been used in the context of the LONGEST COMMON FACTOR WITH AT MOST k MISMATCHES problem by Thankachan et al. [27]. It allows us to reduce the number of approximate copies of each $(S_1, S_2) \in \mathsf{Pairs}_\ell(S)$ to $\mathcal{O}(\log^k n)$. However, the sets $\mathsf{Pairs}_\ell^{(k)}(X)$ and $\mathsf{Pairs}_\ell^{(k)}(Y)$ cannot be constructed independently, and we actually have to build several pairs of such sets rather just one.

Below, we explain the main points for $k = 1$ and $\sigma = 2$.

Let $\mathcal{F}$ be a family consisting of the suffixes of X, X^R, Y, and Y^R, appearing in $\mathsf{Pairs}_\ell(X)$ or $\mathsf{Pairs}_\ell(Y)$. We apply the heavy-light decomposition on the compacted trie $\mathcal{T}(F)$; this technique can be summarized as follows:

▶ **Fact 5** (Tarjan [26]). *If T is a rooted tree, then in linear time we can mark some edges in T as* light *so that:*
- *each node has at most one outgoing edge which is* not *light,*
- *each root-to-leaf path contains $\mathcal{O}(\log |T|)$ light edges.*

Next, for each string $F \in \mathcal{F}$, we construct a set $N(F)$ consisting of F and any string which can be obtained from F by flipping the first symbol on one of the light edges on the path representing F in $\mathcal{T}(\mathcal{F})$. By Fact 5, we have $|N(F)| = \mathcal{O}(\log |\mathcal{F}|) = \mathcal{O}(\log n)$.

Let us introduce two subsets of $N(F)$: $N_0(F) = \{F\}$ and $N_1(F) = N(F)$. These sets have been constructed so that they enjoy the following crucial property:

▶ **Lemma 6.** *If $F, G \in \mathcal{F}$, then*

$$\mathsf{LCP}_1(F, G) = \max_{d_1+d_2=1} \{\mathsf{LCP}(F', G') : F' \in N_{d_1}(F), G' \in N_{d_2}(G)\}.$$

Proof. First, let us bound $\mathsf{LCP}_1(F, G)$ from below. Let $p = \mathsf{LCP}(F', G')$ be the maximum on the right-hand side, We have

$$d_H(F[..p], G[..p]) = d_H(F[..p], F'[..p]) + d_H(G'[..p], G[..p]) \leq$$
$$\leq d_H(F, F') + d_H(G', G) \leq d_1 + d_2 = 1.$$

Consequently, $\mathsf{LCP}_1(F, G) \geq p$ as claimed.

To bound $\mathsf{LCP}_1(F, G)$ from above, let us consider terminal nodes v_F and v_G in $\mathcal{T}(\mathcal{F})$ representing F and G, respectively, and their LCA v. If $v = v_F$ or $v = v_G$, then $\mathsf{LCP}_1(F, G) = \mathsf{LCP}(F, G)$ and the claimed bound holds due to $F \in N_0(F)$ and $G \in N_1(G)$ (and vice versa). Otherwise, the edge from v towards v_F or the edge from v towards v_G has to be light (according to Fact 5). If the former edge is light, then $N_1(F)$ contains a string F' obtained from F by flipping the first letter on that edge. Such a string F' satisfies $\mathsf{LCP}_1(F, G) = \mathsf{LCP}(F', G)$, so the claimed bound holds due to $G \in N_0(G)$. Symmetrically, if the edge towards v_G is light, then $\mathsf{LCP}_1(F, G) = \mathsf{LCP}(F, G')$ for some $G' \in N_1(G)$. ◀

For $S \in \{X, Y\}$ and $d \in \{0, 1\}$, let us define

$$\mathsf{Pairs}_\ell^{(d)}(S) = \bigcup_{\substack{(U_1,U_2)\in\mathsf{Pairs}_\ell(S)\\ d_1+d_2=1}} \{(U_1', U_2') : U_1' \in N_{d_1}(U_1), U_2' \in N_{d_2}(U_2)\}.$$

Observe that $\mathsf{Pairs}_\ell^{(0)}(S) = \mathsf{Pairs}_\ell(S)$, whereas the set $\mathsf{Pairs}_\ell^{(1)}(S)$ satisfies $|\mathsf{Pairs}_\ell^{(1)}(S)| = \mathcal{O}(|\mathsf{Pairs}_\ell(S)| \log |\mathcal{F}|) = \mathcal{O}(n \log n/\sqrt{\ell})$. Lemmas 4 and 6 yield the following

▶ **Corollary 7.** *If* $\mathsf{LCF}_1(X, Y) \geq \ell$*, then*

$$\mathsf{LCF}_1(X, Y) = \max_{k_1+k_2=1} \mathrm{maxPairLCP}(\mathsf{Pairs}_\ell^{(k_1)}(X), \mathsf{Pairs}_\ell^{(k_2)}(Y)).$$

Proof. By Lemma 4, we have $\mathsf{LCF}_1(X, Y) = \mathsf{LCP}_p(U_1, V_1)+\mathsf{LCP}_q(U_2, V_2)$ for some $(U_1, U_2) \in \mathsf{Pairs}_\ell(X)$, $(V_1, V_2) \in \mathsf{Pairs}_\ell(Y)$, and $p + q = 1$. Lemma 6 yields that $\mathsf{LCP}_p(U_1, V_1) = \mathsf{LCP}(U_1', V_1')$ for some $U_1' \in N_{p_1}(U_1)$ and $V_1' \in N_{p_2}(V_1)$ such that $p = p_1 + p_2$. Similarly, $\mathsf{LCP}_q(U_2, V_2) = \mathsf{LCP}(U_2', V_2')$ for some $U_2' \in N_{q_1}(U_2)$ and $V_2' \in N_{q_2}(V_2)$. Observe that $(U_1', U_2') \in \mathsf{Pairs}_\ell^{(p_1+q_1)}(X)$ and $(V_1', V_2') \in \mathsf{Pairs}_\ell^{(p_2+q_2)}(Y)$, so

$$\mathsf{LCF}_1(X, Y) \leq \mathrm{maxPairLCP}(\mathsf{Pairs}_\ell^{(k_1)}(X), \mathsf{Pairs}_\ell^{(k_2)}(Y))$$

for $k_i = p_i + q_1$ (which satisfy $k_1 + k_2 = p + q = 1$, as claimed).

Next, suppose that $(U_1', U_2') \in \mathsf{Pairs}_\ell^{(k_1)}(X)$ and $(V_1', V_2') \in \mathsf{Pairs}_\ell^{(k_2)}(Y)$. We shall prove that $\mathsf{LCF}_1(X, Y) \geq \mathsf{LCP}(U_1', V_1') + \mathsf{LCP}(U_2', V_2')$. Note that $U_1' \in N_{p_1}$ and $U_2' \in N_{q_1}(U_2)$ for some $p_1 + q_1 = k_1$ and $(U_1, U_2) \in \mathsf{Pairs}_\ell(X)$; symmetrically, $V_1' \in N_{p_2}$ and $V_2' \in N_{q_2}(V_2)$ for some $p_2 + q_2 = k_2$ and $(V_1, V_2) \in \mathsf{Pairs}_\ell(Y)$. By Lemma 6, $\mathsf{LCP}(U_1', V_1') \leq \mathsf{LCP}_{p_1+p_2}(U_1, V_1)$ and $\mathsf{LCP}(U_2', V_2') \leq \mathsf{LCP}_{q_1+q_2}(U_2, V_2)$. Hence, the claimed bound holds due to Lemma 4:

$$\mathsf{LCF}_1(X, Y) \geq \mathsf{LCP}_{p_1+p_2}(U_1, V_1) + \mathsf{LCP}_{q_1+q_2}(U_2, V_2) \geq \mathsf{LCP}(U_1', V_1') + \mathsf{LCP}(U_2', V_2').$$

This concludes the proof. ◀

Consequently, it suffices to solve two instances of Two String Families LCP Problem, with $(\mathcal{P}, \mathcal{Q})$ equal to $(\mathsf{Pairs}_\ell^{(0)}(X), \mathsf{Pairs}_\ell^{(1)}(Y))$ and $(\mathsf{Pairs}_\ell^{(1)}(X), \mathsf{Pairs}_\ell^{(0)}(Y))$, respectively.

▶ **Proposition 8.** *The* $\mathrm{LCF}_k(X, Y, \ell)$ *problem for* $k = 1$ *and binary alphabet can be solved in* $\mathcal{O}(n + n \log^2 n/\sqrt{\ell})$ *time. If* $\ell = \Omega(\log^4 n)$*, this running time is* $\mathcal{O}(n)$*.*

Proof. First, we build the sets $\mathsf{Pairs}_\ell(X)$ and $\mathsf{Pairs}_\ell(Y)$. Next, we construct the joint suffix tree of strings X, Y, X^R, Y^R (along with a component for constant-time LCA queries [5]) and we extract the compacted trie $\mathcal{T}(\mathcal{F})$ of the family $\mathcal{F}$. Then, we process light edges on $\mathcal{T}(\mathcal{F})$ (determined by Fact 5) to construct the sets $N(F)$ as defined above for each $F \in \mathcal{F}$. We initialize each set $N(F)$ with F; then, for every light edge e, we traverse the subtree below e and for each terminal node (representing $F \in \mathcal{F}$), we insert to $N(F)$ a string F' obtained from F by flipping the first letter represented by e. Technically, in $N(F)$ we just store the set of positions for which F should be flipped to obtain F'.

To compute the compacted trie $\mathcal{T}(\mathcal{F}')$ of a family $\mathcal{F}' = \bigcup_{F\in\mathcal{F}} N(F)$, we sort the strings in $F' \in \mathcal{F}'$ using a comparison-based algorithm. Next, we extend the representation of $N(F)$ so that each $F' \in N(F)$ stores a pointer to the corresponding terminal node in $\mathcal{T}(\mathcal{F}')$. This way, we can generate sets $\mathsf{Pairs}_\ell^d(S)$ for $d \in \{0, 1\}$ and $S \in \{X, Y\}$ with strings represented as pointers to terminal nodes of $\mathcal{T}(\mathcal{F}')$. Finally, we solve two instances of Two String Families LCP Problem according to Corollary 7.

We conclude with the running-time analysis. In the preprocessing, we spend $\mathcal{O}(n)$ time construct the joint suffix tree. Then, applying Fact 5 to build the sets $N(F)$ for $F \in \mathcal{F}$ takes $\mathcal{O}(|\mathcal{F}| \log |\mathcal{F}|) = \mathcal{O}(n \log n/\sqrt{\ell})$ time. We spend further $\mathcal{O}(|\mathcal{F}'| \log |\mathcal{F}'|) = \mathcal{O}(n \log^2 n/\sqrt{\ell})$ time to construct $\mathcal{T}(\mathcal{F}')$. Since $|\mathsf{Pairs}_\ell^{(d)}(S)| = \mathcal{O}(|\mathcal{F}| \log |\mathcal{F}|)$ for $d \in \{0,1\}$ and $S \in \{X,Y\}$, the time to solve both instances of the Two String Families LCP Problem is also $\mathcal{O}(n \log^2 n/\sqrt{\ell})$ (see Lemma 3). Hence, the overall time complexity is $\mathcal{O}(n + n \log^2 n/\sqrt{\ell})$. ◀

5 Arbitrary k and σ

In this section, we describe the core concepts of our solution for arbitrary number of mismatches k and alphabet size σ. They depend heavily on the ideas behind the $\mathcal{O}(n \log^k n)$-time solution to the LCF$_k$ problem [27], which originate in approximate indexing [9].

▶ **Definition 9.** Consider strings $U, V \in \Sigma^*$ and an integer $d \geq 0$. We say that strings $U', V' \in \Sigma_\$^*$ form a $(U,V)_d$-pair if

- $|U'| = |U|$ and $|V'| = |V|$;
- if $i > \mathsf{LCP}_d(U,V)$ or $U[i] = V[i]$, then $U'[i] = U[i]$ and $V'[i] = V[i]$;
- otherwise, $U'[i] = V'[i] \in \{U[i], V[i], \$\}$.

For a string $S \in \Sigma_\$^*$ let us define $\#_\$(S) = |\{1 \leq i \leq |S| : S[i] = \$\}|$. The following observation specifies key properties of $(U,V)_d$ pairs.

▶ **Observation 10.** *Consider strings $U, V \in \Sigma^*$ and an integer $d \geq 0$. If strings $U', V' \in \Sigma_\$^*$ form a $(U,V)_d$-pair, then the following conditions hold:*

(a) $\mathsf{LCP}(U', V') = \mathsf{LCP}_d(U,V)$,

(b) $\#_\$(U') = \#_\$(V')$,

(c) $d = d_H(U,U') - \frac{1}{2}\#_\$(U') + d_H(V,V') - \frac{1}{2}\#_\$(V')$.

▶ **Definition 11.** Consider a finite family of strings $\mathcal{F} \subseteq \Sigma^*$. We say that sets $N(F) \subseteq \Sigma_\$^*$ for $F \in \mathcal{F}$ form a *k-complete family* if for every $U, V \in \mathcal{F}$ and $0 \leq d \leq k$, there exists a $(U,V)_d$-pair U', V' with $U' \in N(U)$ and $V' \in N(V)$.

▶ Remark. A simple (yet inefficient) way to construct a k-complete family is to include in $N(F)$ all strings which can be obtained from F by replacing up to k letters with \$'s. An example of a more efficient family is shown in Table 1.

The following lemma states a property of k-complete families that we will use in the algorithm. For $F \in \mathcal{F}$ and $0 \leq d \leq k$, let us define $N_d(F) = \{F' \in N(F) : d_H(F,F') \leq d\}$. Moreover, for a half integer[1] d', $0 \leq d' \leq d$, let

$$N_{d,d'}(F) = \{F' \in N_d(F) : d_H(F,F') - \tfrac{1}{2}\#_\$(F') \leq d'\},$$

▶ **Lemma 12.** *Let $N(F)$ for $F \in \mathcal{F}$ be a k-complete family. If $F_1, F_2 \in \mathcal{F}$ and $0 \leq d \leq k$, then*

$$\mathsf{LCP}_d(F_1,F_2) = \max_{\substack{d_1+d_2=d \\ F_i' \in N_{d,d_i}(F_i)}} \mathsf{LCP}(F_1', F_2') = \max_{\substack{d_1+d_2<d+1 \\ F_i' \in N_{k,d_i}(F_i)}} \mathsf{LCP}(F_1', F_2').$$

[1] Here, a half integer is a number of the form $\frac{a}{2}$, where a is an integer.

Table 1 A sample 1-complete family for $\mathcal{F} = \{\texttt{abacb}, \texttt{bacb}, \texttt{acb}, \texttt{cb}, \texttt{b}\}$ (the suffixes of abacb) is $N(\texttt{b}) = \{\texttt{a}, \texttt{b}, \$\}$, $N(\texttt{cb}) = \{\texttt{ab}, \texttt{cb}, \$\texttt{b}\}$, $N(\texttt{acb}) = \{\texttt{abb}, \texttt{acb}\}$, $N(\texttt{bacb}) = \{\texttt{aacb}, \texttt{bacb}, \$\texttt{acb}\}$, and $N(\texttt{abacb}) = \{\texttt{abacb}\}$. The $(U, V)_1$-pairs for all $U, V \in \mathcal{F}$ are illustrated in the table above. Observe that $\mathsf{LCP}_1(U, V) = \mathsf{LCP}(U', V')$ for the corresponding $(U, V)_1$-pair (U', V'). Also, note that $\mathsf{LCP}_1(\texttt{acb}, \texttt{cb}) = 1$ even though $\texttt{abb} \in N(\texttt{acb})$, $\texttt{ab} \in N(\texttt{cb})$, and $\mathsf{LCP}(\texttt{abb}, \texttt{ab}) = 2$.

	b	cb	acb	bacb	abacb
abacb	a abacb	ab abacb	abb abacb	aacb abacb	abacb abacb
bacb	b bacb	\$b \$acb	acb aacb	bacb bacb	abacb aacb
acb	a acb	ab acb	acb acb	aacb acb	abacb abb
cb	\$ \$b	cb cb	acb ab	\$acb \$b	abacb ab
b	b b	\$b \$	acb a	bacb b	abacb a

Proof. We shall prove that

$$\max_{\substack{d_1+d_2=d\\ F_i'\in N_{d,d_i}(F_i)}} \mathsf{LCP}(F_1', F_2') \geq \mathsf{LCP}_d(F_1, F_2) \geq \max_{\substack{d_1+d_2<d+1\\ F_i'\in N_{k,d_i}(F_i)}} \mathsf{LCP}(F_1', F_2').$$

This is sufficient due to the fact that $N_{d,d'}(F)$ is monotone with respect to both d and d'.

For the first inequality, observe that (by definition of a k-complete family) the sets $N(F_1)$ and $N(F_2)$ contain an $(F_1, F_2)_d$-pair (F_1', F_2'). By Observation 10, we have

$$d \geq d_H(F_1[..|P|], F_2[..|P|]) = d_H(F_1, F_1') - \tfrac{1}{2}\#_\$(F_1') + d_H(F_2, F_2') - \tfrac{1}{2}\#_\$(F_2').$$

Consequently, $F_i' \in \mathcal{N}_{d,d_i}(F_i)$ for $d_i = d_H(F_i, F_i') - \frac{1}{2}\#_\$(F_i')$. If $d > d_1 + d_2$, we may increase d_1 or d_2.

For the second inequality, suppose that $F_i' \in N_{k,d_i}(F_i)$ for $d_1 + d_2 < d + 1$. Let P be the longest common prefix of F_1' and F_2'. Then

$$\begin{aligned} d_H(F_1[..|P|], F_2[..|P|]) &\leq d_H(F_1[..|P|], P) + d_H(F_2[..|P|], P) - \#_\$(P) \leq \\ &\leq d_H(F_1, F_1') - \#_\$(F_1') + d_H(F_2, F_2') - \#_\$(F_2') + \#_\$(P) \leq d_1 + d_2. \end{aligned}$$

Consequently, $d_H(F_1[..|P|], F_2[..|P|]) \leq d_1 + d_2 < d + 1$, i.e., $d_H(F_1[..|P|], F_2[..|P|]) \leq d$, as claimed. ◀

In the algorithms, we represent a k-complete family using the compacted trie $\mathcal{T}(\mathcal{F}')$ of the union $\mathcal{F}' = \bigcup_{F\in\mathcal{F}} N(F)$. Its terminal nodes F' are marked with a subset of strings $F \in \mathcal{F}$ for which $F' \in N(F)$; for convenience we also store $\#_\$(F')$ and $d_H(F, F')$. Each edge is labeled by a factor of $F \in \mathcal{F}$, perhaps prepended by \$.

Our construction of a k-complete family is based on the results of [9, 27], but below we provide a self-contained proof.

▶ **Proposition 13** (see also [9, 27]). *Let $\mathcal{F} \subseteq \Sigma^*$ be a finite family of strings and let $k \geq 0$ be an integer. There exists a k-complete family N such that $|N_d(F)| \leq 2^d\binom{\log|\mathcal{F}|+d}{d}$ for each $F \in \mathcal{F}$ and $0 \leq d \leq k$. Moreover, the compacted trie $\mathcal{T}(\mathcal{F}')$ can be constructed in $\mathcal{O}(2^k|\mathcal{F}|\binom{\log|\mathcal{F}|+k+1}{k+1})$ time provided constant-time* LCP *queries for suffixes of the strings $F \in \mathcal{F}$.*

Algorithm 1: A recursive procedure inserting strings with prefix P to sets $N(F)$.

Function `Generate`$(P, \mathcal{F}_P)$ **is**
  $h :=$ a most frequent element of $\{S[1] : (S,F,b) \in \mathcal{F}_P$ and $S \neq \varepsilon\}$;
  foreach $(S,F,b) \in \mathcal{F}_P$ **do** // $b = k - d_H(F, PS) \geq 0$
    if $S = \varepsilon$ **then** $N(F) := N(F) \cup \{P\}$;
    else
      $c := S[1]$;
      $\mathcal{F}_{Pc} := \mathcal{F}_{Pc} \cup \{\, (S[2\,.\,.], F, b)\, \}$;
      if $c \neq h$ **and** $b > 0$ **then**
        $\mathcal{F}_{Ph} := \mathcal{F}_{Ph} \cup \{\, (S[2\,.\,.], F, b-1)\, \}$;
        $\mathcal{F}_{P\$} := \mathcal{F}_{P\$} \cup \{\, (S[2\,.\,.], F, b-1)\, \}$;
  foreach $c \in \Sigma \cup \{\$\}$ **such that** $\mathcal{F}_{Pc} \neq \emptyset$ **do**
    `Generate`$(Pc, \mathcal{F}_{Pc})$;

Proof. We apply a recursive procedure that builds the subtree rooted at the node representing P. The input $\mathcal{F}_P$ consists of tuples (S,F,b) such that $F \in \mathcal{F}$, S is a suffix of F of length $|S| = |F| - |P|$, and $b = k - d_H(F, PS) \geq 0$. Intuitively, the parameter b can be seen as a "budget" of remaining symbol changes in the string that prevents exceeding the number k of mismatches. In the first call we have $P = \varepsilon$ and $\mathcal{F}_P = \{(F,F,k) : F \in \mathcal{F}\}$.

In the pseudocode below we state this procedure in an abstract way; afterwards we explain how to implement it efficiently. The 1-complete family from Table 1 is a subset of the family constructed by that procedure.

Correctness of Algorithm 1 is relatively easy to derive. Due to space constrains, the proof of the following claim can be found in the full version.

▶ **Claim 14.** *For every $S,T \in \mathcal{F}$ and $0 \leq d \leq k$, there exists an $(S,T)_d$-pair (S',T') with $S' \in N(S)$ and $T' \in N(T)$.*

We also refer to the full version for a complete proof of the following bound on $N_d(F)$.

▶ **Claim 15.** *For each $F \in \mathcal{F}$, we have $|N_d(F)| \leq 2^d \binom{\log|\mathcal{F}|+d}{d}$.*

The idea is to define $N_{d,P}(F) = \{F' \in N_d(F) : P$ is a prefix of $F'\}$ for each $P \in \Sigma_\$^*$ and to prove the following bound by induction on decreasing $|P|$:

$$|N_{d,P}(F)| \leq \begin{cases} 2^b \binom{\log|\mathcal{F}_P|+b}{b} & \text{if } (S,F,b+k-d) \in \mathcal{F}_P \text{ and } b \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Below, we show that the k-complete family N represented as a trie T_N can be constructed in $\mathcal{O}(|\mathcal{F}|2^k\binom{\log|\mathcal{F}|+k+1}{k+1})$ time provided constant-time LCP queries for suffixes of strings $F \in \mathcal{F}$.

To a tuple $(S,F,b) \in \mathcal{F}_P$ we assign a number of tokens:

$$\mathit{Tokens}_P(S,F,b) = C(2^{b+1}-1)\binom{\log|\mathcal{F}_P|+b+1}{b+1}$$

where C is a sufficiently large constant. We shall inductively prove that `Generate` $(P, \mathcal{F}_P)$ can be implemented in time $\sum_{(S,F,b)\in\mathcal{F}_P} \mathit{Tokens}_P(S,F,b)$.

In the implementation of the procedure we use finger search trees [14], which maintain subsets of a linearly-ordered universe supporting constant-time queries. Among many applications (see [6] for a survey), they support the following two operations [16, 6]:

- insert an element into a set A, which takes $\mathcal{O}(\log|A|)$ time,
- for a given key t, split the set A into $A_{\le t} = \{a \in A : a \le t\}$ and $A_{>t} = \{a \in A : a > t\}$, which takes $\mathcal{O}(\log\min(|A_{\le t}|, |A_{>t}|))$ time.

We are now ready to specify how the arguments to the procedure `Generate` $(P, \mathcal{F}_P)$ are given. The string P is represented by the corresponding node of the constructed trie T_N; we also explicitly store $|P|$ and $\#_\$(P)$. The set $\mathcal{F}_P$ is stored in a finger search tree with tuples (S, F, b) ordered by S. However, S is not stored itself as it is uniquely specified as a suffix of F of length $|F| - |P|$. Thus each element in the tree is stored in $\mathcal{O}(1)$ space.

First, we process tuples (S, F, b) with $S = \varepsilon$, conveniently located at the beginning of $\mathcal{F}_P$. We remove these tuples from $\mathcal{F}_P$ and store F at the current node of T_N. This simulates inserting P to $N(F)$; we also store auxiliary values $d_H(P, F) = k - b$ and $\#_\$(P)$.

Next, we compute the length of longest common prefix P' of non-empty strings S with $(S, F, b) \in \mathcal{F}_P$. For this, we make an LCP query for the smallest and the largest of these suffixes. If the longest common prefix P' is non-empty, we observe that $\mathcal{F}_{PP'} = \mathcal{F}_P$ (with the stored representation unchanged) and Algorithm 1 does not explore any other branch. Hence, we immediately call `Generate` $(PP', \mathcal{F}_{PP'})$ which corresponds to creating a complete compacted edge of the resulting trie. This step takes $\mathcal{O}(1)$ time, but it guarantees that `Generate` $(PP', \mathcal{F}_{PP'})$ outputs or branches. Hence, this time gets amortized.

If $P' = \varepsilon$, we partition $\mathcal{F}_P$ into at most σ finger search trees $\mathcal{F}_{P,c}$ each storing tuples sharing the letter $S[1] = c$, and we identify the heavy letter h by choosing the largest $\mathcal{F}_{P,c}$. For this, we iteratively split out the tree with the smallest unprocessed $S[1]$, which takes time proportional to $\sum_{c \ne h} \log|\mathcal{F}_{P,c}|$.

The sets $\mathcal{F}_{Pc}$ for $c \ne h$ already represented by $\mathcal{F}_{P,c}$ (note that the order does not change, and the tuples need not be altered since the "budget" b remains the same and S is stored implicitly). Similarly, we can build $\mathcal{F}_{Ph}$ by inserting new tuples into $\mathcal{F}_{P,h}$.

Thus, we define $\mathcal{L}_P := \{(S, F, b) \in \mathcal{F}_P : S \ne \varepsilon \text{ and } S[1] \ne h\}$ and insert to $\mathcal{F}_{Ph}$ and $\mathcal{F}_{P\$}$ tuples $(S[2\,.\,.], F, b-1)$ for $(S, F, b) \in \mathcal{L}_P$ with $b > 0$, which takes $\mathcal{O}(\log|\mathcal{F}_P|)$ time per element.

In total, the processing time is $\mathcal{O}(1)$ for each element of $\mathcal{L}_P$ with $b = 0$, and $\mathcal{O}(\log|\mathcal{F}_P|)$ when $b > 0$. Additionally, we may spend $\mathcal{O}(1)$ time for a tuple with $S = \varepsilon$. Let us check that the difference in the number of tokens is sufficient to cover the running time of these operations.

The tuples with $S = \varepsilon$ do not appear in future computations. Hence, we spend all their tokens on the computations related to them. It is indeed sufficient:

$$\mathit{Tokens}_P(\varepsilon, F, b) = C(2^{b+1} - 1)\binom{\log|\mathcal{F}_P| + b + 1}{b+1} \ge C\binom{\log|\mathcal{F}_P| + 1}{1} = C(\log|\mathcal{F}_P| + 1) \ge C.$$

We don't spend any time on tuples with $S[1] = h$, and number of tokens for such a tuple does not increase:

$$\begin{aligned}&\mathit{Tokens}_P(S, F, b) - \mathit{Tokens}_{Ph}(S, F, b) = \\ &C(2^{b+1} - 1)\binom{\log|\mathcal{F}_P| + b + 1}{b+1} - C(2^{b+1} - 1)\binom{\log|\mathcal{F}_{Ph}| + b + 1}{b+1} \ge 0.\end{aligned}$$

Finally, for a tuple with $S[1] \ne h$ (i.e., in $\mathcal{L}_p$) the difference in the number of tokens is

$$\begin{aligned}&\mathit{Tokens}_P(S, F, b) - \mathit{Tokens}_{Pc}(S', F, b) - \mathit{Tokens}_{Ph}(S', F, b-1) - \mathit{Tokens}_{P\$}(S', F, b-1) = \\ &= C(2^{b+1} - 1)\binom{\log|\mathcal{F}_P| + b + 1}{b+1} - C(2^{b+1} - 1)\binom{\log|\mathcal{F}_{Pc}| + b + 1}{b+1} - C(2^b - 1)\binom{\log|\mathcal{F}_{Ph}| + b}{b} \\ &\qquad - C(2^b - 1)\binom{\log|\mathcal{F}_{P\$}| + b}{b} \ge C\binom{\log|\mathcal{F}_P| + b}{b}\end{aligned}$$

where $c = S[1]$ and $S' = S[2\,.\,.]$. It is sufficient since we spend constant time for $b = 0$ and $\mathcal{O}(\log |\mathcal{F}_P|)$ time for $b \geq 1$.

The claimed bound on the overall running time follows. ◀

6 Main Result

Let $\mathcal{F}$ be a family of suffixes and reverse prefixes of X and Y occurring in $\mathsf{Pairs}_\ell(X)$ or $\mathsf{Pairs}_\ell(Y)$, and let us fix a k-complete family $N(F) : F \in \mathcal{F}$. For a half integer k', $0 \leq k' \leq k$, and a string $S \in \{X, Y\}$ let us define

$$\mathsf{Pairs}_\ell^{(k,k')}(S) = \bigcup_{(U_1,U_2)\in \mathsf{Pairs}_\ell(S)} \{(U_1', U_2') : U_i' \in N_{d_i,d_i'}(U_i), k = d_1 + d_2, k' = d_1' + d_2'\}.$$

Intuitively, we extend $(U_1, U_2) \in \mathsf{Pairs}_\ell(S)$, arbitrarily splitting the budgets k and k' between U_1 and U_2. To bound the size of $\mathsf{Pairs}_\ell^{(k,k')}(S)$, we observe that for $d_1 + d_2 = k$ and $k = \mathcal{O}(\log n)$

$$|N_{d_1}(U_1)| \cdot |N_{d_2}(U_2)| \leq 2^k \binom{\log |\mathcal{F}| + d_1}{d_1} \binom{\log |\mathcal{F}| + d_2}{d_2} = \frac{2^{\mathcal{O}(k)} \log^k |\mathcal{F}|}{k^k}.$$

Hence, $|\mathsf{Pairs}_\ell^{(k,k')}(S)| = \frac{2^{\mathcal{O}(k)} |\mathcal{F}| \log^k |\mathcal{F}|}{k^k \sqrt{\ell}}$. Combining Lemmas 4 and 12, we obtain the following.

▶ **Corollary 16.** *If* $\mathsf{LCF}_k(X, Y) \geq \ell$*, then*

$$\mathsf{LCF}_k(X, Y) = \max_{k_1+k_2=k} \mathrm{maxPairLCP}(\mathsf{Pairs}_\ell^{(k,k_1)}(X), \mathsf{Pairs}_\ell^{(k,k_2)}(Y)).$$

Proof. By Lemma 4, there exist $(U_1, U_2) \in \mathsf{Pairs}_\ell(X)$, $(V_1, V_2) \in \mathsf{Pairs}_\ell(Y)$, and $p + q = k$ such that $\mathsf{LCF}_k(X, Y) = \mathsf{LCP}_p(U_1, V_1) + \mathsf{LCP}_q(U_2, V_2)$. Lemma 12 further yields the existence of half integers $p_1' + p_2' \leq p$ and $q_1' + q_2' \leq q$ such that $\mathsf{LCF}_k(X, Y) = \mathsf{LCP}(U_1', V_1') + \mathsf{LCP}(U_2', V_2')$ for some $U_1' \in N_{p,p_1'}(U_1)$, $V_1' \in N_{p,p_2'}(V_1)$, $U_2' \in N_{q,q_1'}(U_2)$, and $V_2' \in N_{q,q_2'}(V_2)$.

We set $k_1' = p_1' + q_1'$ and $k_2' = k - k_1' \geq p_2' + q_2'$ so that $(U_1', U_2') \in \mathsf{Pairs}_\ell^{(k,k_1')}(X)$ and $(V_1', V_2') \in \mathsf{Pairs}_\ell^{(k,p_2'+q_2')}(Y) \subseteq \mathsf{Pairs}_\ell^{(k,k_2')}(Y)$. Consequently,

$$\mathsf{LCF}_k(X, Y) \leq \mathrm{maxPairLCP}(\mathsf{Pairs}_\ell^{(k,k_1')}(X), \mathsf{Pairs}_\ell^{(k,k_2')}(Y)),$$

which concludes the proof of the upper bound on $\mathsf{LCF}_k(X, Y)$.

For the lower bound, we shall prove that $\mathsf{LCF}_k(X, Y) \geq \mathsf{LCP}(U_1', V_1') + \mathsf{LCP}(U_2', V_2')$ for all $(U_1', U_2') \in \mathsf{Pairs}_\ell^{(k,k_1')}(X)$ and $(V_1', V_2') \in \mathsf{Pairs}_\ell^{(k,k_2')}(Y)$ such that $k_1' + k_2' \leq k$. By definition of $\mathsf{Pairs}_\ell^{(k,k_1')}$, there exist $(U_1, U_2) \in \mathsf{Pairs}_\ell(X)$ such that $U_1' \in N_{k,p_1'}(U_1)$ and $U_2' \in N_{k,q_1'}(U_2)$ for half integers $p_1' + q_1' \leq k_1'$. Similarly, there exist $(V_1, V_2) \in \mathsf{Pairs}_\ell(Y)$ such that $V_1' \in N_{k,p_2'}(V_1)$ and $V_2' \in N_{k,q_2'}(V_2)$ for half integers $p_2' + q_2' \leq k_2'$. We set $p = \lfloor p_1' + p_2' \rfloor$ and $q = \lfloor q_1' + q_2' \rfloor$, and observe that $\mathsf{LCP}_p(U_1, V_1) \geq \mathsf{LCP}(U_1', V_1')$ as well as $\mathsf{LCP}_q(U_2, V_2) \geq \mathsf{LCP}(U_2', V_2')$ due to Lemma 12. Now, Lemma 4 yields $\mathsf{LCF}_k(X, Y) \geq \mathsf{LCP}_p(U_1, V_1) + \mathsf{LCP}_q(U_2, V_2) \geq \mathsf{LCP}(U_1', V_1') + \mathsf{LCP}(U_2', V_2')$, as desired. ◀

▶ **Theorem 17.** *For* $k = \mathcal{O}(\log n)$*, the* $\mathrm{LCF}_k(X, Y, \ell)$ *problem can be solved in time* $\mathcal{O}(n + \frac{2^{\mathcal{O}(k)} n \log^{k+1} n}{k^k \sqrt{\ell}})$*. For* $k = \mathcal{O}(1)$*, this running time becomes* $\mathcal{O}(n + \frac{n \log^{k+1} n}{\sqrt{\ell}})$*.*

Proof. First, we build the joint suffix tree of X, X^R, Y, and Y^R, as well as the family $\mathcal{F}$. A component for the LCA queries on the suffix tree lets us compare any suffixes of $F \in \mathcal{F}$ in constant time [5]. This allows us to build the k-complete family $N(F) : F \in \mathcal{F}$, represented as a compacted trie of $\mathcal{F}' := \bigcup\{N(F) : F \in \mathcal{F}\}$ using Proposition 13. Next, we construct the sets $\mathsf{Pairs}_\ell^{(k,k')}(X) \subseteq (\mathcal{F}')^2$ and $\mathsf{Pairs}_\ell^{(k,k')}(Y) \subseteq (\mathcal{F}')^2$ for $k' = 0, \frac{1}{2}, \ldots, k - \frac{1}{2}, k$, and solve the $2k+1$ instances of Two String Families LCP Problem, as specified in Corollary 16.

We conclude with running-time analysis. Preprocessing takes $\mathcal{O}(n)$ time, and the procedure of Proposition 13 runs in $\mathcal{O}(2^k|\mathcal{F}|\binom{\log|\mathcal{F}|+k+1}{k+1}) = \frac{2^{\mathcal{O}(k)} n \log^{k+1} n}{k^k\sqrt{\ell}}$ time. We have $\mathsf{Pairs}_\ell^{k,k'}(X) = \frac{2^{\mathcal{O}(k)} n \log^{k} n}{k^k\sqrt{\ell}}$, so solving all instances of Two String Families LCP Problem also takes $\frac{2^{\mathcal{O}(k)} n \log^{k+1} n}{k^k\sqrt{\ell}}$ time (Lemma 3). The overall running time is therefore as claimed. ◀

In particular, for $k = \mathcal{O}(\log n)$, there exists $\ell_0 = \frac{2^{\mathcal{O}(k)} \log^{2k+2} n}{k^{2k}}$ such that the $\mathrm{LCF}_k(X, Y, \ell)$ problem can be solved in $\mathcal{O}(n)$ time for $\ell \geq \ell_0$. For $k = \mathcal{O}(1)$, we have $\ell_0 = \mathcal{O}(\log^{2k+2} n)$, while for $k = o(\log n)$, we have $\ell_0 = n^{o(1)}$. We arrive at the main result.

▶ **Corollary 18.** *The $\mathrm{LCF}_k(X, Y, \ell)$ problem with $\ell = \Omega(\log^{2k+2} n)$ can be solved in $\mathcal{O}(n)$ time.*

References

1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

2 Hayam Alamro, Lorraine A. K. Ayad, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Solon P. Pissis. Longest common prefixes with k-mismatches and applications. In A Min Tjoa, Ladjel Bellatreche, Stefan Biffl, Jan van Leeuwen, and Jirí Wiedermann, editors, *SOFSEM 2018: Theory and Practice of Computer Science - 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, January 29 - February 2, 2018, Proceedings*, volume 10706 of *Lecture Notes in Computer Science*, pages 636–649. Springer, 2018. doi:10.1007/978-3-319-73117-9_45.

3 Lorraine A. K. Ayad, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Solon P. Pissis. Longest common prefixes with k-errors and applications. *CoRR*, abs/1801.04425, 2018. arXiv:1801.04425.

4 Maxim A. Babenko and Tatiana A. Starikovskaya. Computing the longest common substring with one mismatch. *Probl. Inf. Transm.*, 47(1):28–33, 2011. doi:10.1134/S0032946011010030.

5 Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.

6 Gerth Stølting Brodal. Finger search trees. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications.* Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch11.

7 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *J. ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.

8 Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In Ricardo A. Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, volume 2676 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2003. doi:10.1007/3-540-44888-8_5.

9 Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 91–100. ACM, 2004. `doi:10.1145/1007352.1007374`.

10 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

11 Maxime Crochemore, Costas S. Iliopoulos, Manal Mohamed, and Marie-France Sagot. Longest repeats with a block of k don't cares. *Theor. Comput. Sci.*, 362(1-3):248–254, 2006. `doi:10.1016/j.tcs.2006.06.029`.

12 Tomás Flouri, Emanuele Giaquinta, Kassian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Inf. Process. Lett.*, 115(6-8):643–647, 2015. `doi:10.1016/j.ipl.2015.03.006`.

13 Szymon Grabowski. A note on the longest common substring with k-mismatches problem. *Inf. Process. Lett.*, 115(6-8):640–642, 2015. `doi:10.1016/j.ipl.2015.03.003`.

14 Leonidas J. Guibas, Edward M. McCreight, Michael F. Plass, and Janet R. Roberts. A new representation for linear lists. In John E. Hopcroft, Emily P. Friedman, and Michael A. Harrison, editors, *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 49–60. ACM, 1977. `doi:10.1145/800105.803395`.

15 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

16 Kurt Hoffman, Kurt Mehlhorn, Pierre Rosenstiehl, and Robert Endre Tarjan. Sorting jordan sequences in linear time using level-linked search trees. *Information and Control*, 68(1-3):170–184, 1986. `doi:10.1016/S0019-9958(86)80033-X`.

17 Lucas Chi Kwong Hui. Color set size problem with application to string matching. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, Third Annual Symposium, CPM 92, Tucson, Arizona, USA, April 29 - May 1, 1992, Proceedings*, volume 644 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 1992. `doi:10.1007/3-540-56024-6_19`.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. `doi:10.1006/jcss.2000.1727`.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. `doi:10.1006/jcss.2001.1774`.

20 Tomasz Kociumaka, Jakub Radoszewski, and Tatiana A. Starikovskaya. Longest common substring with approximately k mismatches. *CoRR*, abs/1712.08573, 2017. `arXiv:1712.08573`.

21 Tomasz Kociumaka, Tatiana A. Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2014. `doi:10.1007/978-3-662-44777-2_50`.

22 Chris-André 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`.

23 Mamoru Maekawa. A square root N algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2):145–159, 1985.

24 Tatiana A. Starikovskaya. Longest common substring with approximately k mismatches. In Roberto Grossi and Moshe Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel*, volume 54 of *LIPIcs*, pages 21:1–21:11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CPM.2016.21.

25 Tatiana A. Starikovskaya and Hjalte Wedel Vildhøj. Time-space trade-offs for the longest common substring problem. In Johannes Fischer and Peter Sanders, editors, *Combinatorial Pattern Matching, 24th Annual Symposium, CPM 2013, Bad Herrenalb, Germany, June 17-19, 2013. Proceedings*, volume 7922 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2013. doi:10.1007/978-3-642-38905-4_22.

26 Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979. doi:10.1145/322154.322161.

27 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.

28 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.

29 Igor Ulitsky, David Burstein, Tamir Tuller, and Benny Chor. The average common substring approach to phylogenomic reconstruction. *Journal of Computational Biology*, 13(2):336–350, 2006. doi:10.1089/cmb.2006.13.336.