


Faster Algorithms for Longest Common Substring

Panagiotis Charalampopoulos ✉ 

The Interdisciplinary Center Herzliya, Israel


Tomasz Kociumaka ✉ 

University of California, Berkeley, CA, USA

Solon P. Pissis ✉ 

CWI, Amsterdam, The Netherlands

Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski ✉ 

Institute of Informatics, University of Warsaw, Poland

Samsung R&D, Warsaw, Poland

Abstract

In the classic longest common substring (LCS) problem, we are given two strings S and T , each of length at most n , over an alphabet of size σ , and we are asked to find a longest string occurring as a fragment of both S and T . Weiner, in his seminal paper that introduced the suffix tree, presented an $\mathcal{O}(n \log \sigma)$ -time algorithm for this problem [SWAT 1973]. For polynomially-bounded integer alphabets, the linear-time construction of suffix trees by Farach yielded an $\mathcal{O}(n)$ -time algorithm for the LCS problem [FOCS 1997]. However, for small alphabets, this is not necessarily optimal for the LCS problem in the word RAM model of computation, in which the strings can be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time. We show that, in this model, we can compute an LCS in time $\mathcal{O}(n \log \sigma / \sqrt{\log n})$, which is sublinear in n if $\sigma = 2^{o(\sqrt{\log n})}$ (in particular, if $\sigma = \mathcal{O}(1)$), using optimal space $\mathcal{O}(n \log \sigma / \log n)$.

We then lift our ideas to the problem of computing a k -mismatch LCS, which has received considerable attention in recent years. In this problem, the aim is to compute a longest substring of S that occurs in T with at most k mismatches. Flouri et al. showed how to compute a 1-mismatch LCS in $\mathcal{O}(n \log n)$ time [IPL 2015]. Thankachan et al. extended this result to computing a k -mismatch LCS in $\mathcal{O}(n \log^k n)$ time for $k = \mathcal{O}(1)$ [J. Comput. Biol. 2016]. We show an $\mathcal{O}(n \log^{k-1/2} n)$ -time algorithm, for any constant integer $k > 0$ and *irrespective* of the alphabet size, using $\mathcal{O}(n)$ space as the previous approaches. We thus notably break through the well-known $n \log^k n$ barrier, which stems from a recursive heavy-path decomposition technique that was first introduced in the seminal paper of Cole et al. [STOC 2004] for string indexing with k errors.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases longest common substring, k mismatches, wavelet tree

Digital Object Identifier 10.4230/LIPIcs.ESA.2021.30

Related Version *Full Version:* <https://arxiv.org/abs/2105.03106>

Funding *Panagiotis Charalampopoulos:* Supported by the Israel Science Foundation grant 592/17.

Tomasz Kociumaka: Partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

Solon P. Pissis: This paper is part of the PANGAIA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 872539. This paper is also part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956229.

Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.



© Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski; licensed under Creative Commons License CC-BY 4.0

29th Annual European Symposium on Algorithms (ESA 2021).

Editors: Petra Mutzel, Rasmus Pagh, and Grzegorz Herman; Article No. 30; pp. 30:1–30:17

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In the classic longest common substring (LCS) problem, we are given two strings S and T , each of length at most n , over an alphabet of size σ , and we are asked to find a longest string occurring as a fragment of both S and T . The problem was conjectured by Knuth to require $\Omega(n \log n)$ time until Weiner, in his seminal paper introducing the suffix tree [62], showed that the LCS problem can be solved in $\mathcal{O}(n)$ time when σ is constant via constructing the suffix tree of string $S\#T$, for a sentinel letter $\#$. Later, Farach showed that if σ is not constant, the suffix tree can be constructed in linear time in addition to the time required for sorting its letters [33]. This yielded an $\mathcal{O}(n)$ -time algorithm for the LCS problem in the word RAM model for polynomially-bounded integer alphabets. While Farach's algorithm for suffix tree construction is *optimal* for all alphabets (the suffix tree by definition has size $\Theta(n)$), the same does not hold for the LCS problem. We were thus motivated to answer the following basic question:

Can the LCS problem be solved in $o(n)$ time when $\log \sigma = o(\log n)$?

We consider the word RAM model and assume an alphabet $[0, \sigma)$. Any string of length n can then be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time. Note that if $\log \sigma = \Theta(\log n)$, one requires $\Theta(n)$ time to read the input. We answer this basic question positively when $\log \sigma = o(\sqrt{\log n})$:

► **Theorem 1.** *Given two strings S and T , each of length at most n , over an alphabet $[0, \sigma)$, we can solve the LCS problem in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n / \log_\sigma n)$ space.*

We also consider the following generalisation of the LCS problem that allows mismatches.

k -MISMATCH LONGEST COMMON SUBSTRING (k -LCS)

Input: Two strings S and T , each of length at most n , over an integer alphabet and an integer $k > 0$.

Output: A pair S', T' of substrings of S and T , respectively, with Hamming distance (i.e., number of mismatches) at most k and maximal length.

Flouri et al. presented an $\mathcal{O}(n \log n)$ -time algorithm for the 1-LCS problem [35]. (Earlier work on this problem includes [6].) This was generalised by Thankachan et al. [59] to an algorithm for the k -LCS problem that works in $\mathcal{O}(n \log^k n)$ time if $k = \mathcal{O}(1)$. Both algorithms use $\mathcal{O}(n)$ space. In [24], Charalampopoulos et al. presented an $\mathcal{O}(n + n \log^{k+1} n / \sqrt{\ell})$ -time algorithm for k -LCS with $k = \mathcal{O}(1)$, where ℓ is the length of a k -LCS. For general k , Flouri et al. presented an $\mathcal{O}(n^2)$ -time algorithm that uses $\mathcal{O}(1)$ additional space [35]. Grabowski [40] presented two algorithms with running times $\mathcal{O}(n((k+1)(\ell_0+1))^k)$ and $\mathcal{O}(n^2 k / \ell_k)$, where ℓ_0 and ℓ_k are, respectively, the length of an LCS of S and T and the length of a k -LCS of S and T . Abboud et al. [1] employed the polynomial method to obtain a $k^{1.5} n^2 / 2^{\Omega(\sqrt{\log n/k})}$ -time randomised algorithm. In [49], Kociumaka et al. showed that, assuming the Strong Exponential Time Hypothesis (SETH) [45, 46], no strongly subquadratic-time solution for k -LCS exists for $k = \Omega(\log n)$. The authors of [49] additionally presented a subquadratic-time 2-approximation algorithm for k -LCS for general k .

Analogously to Weiner's solution to the LCS problem via suffix trees, the algorithm of Thankachan et al. [59] builds upon the ideas of the k -errata tree, which was introduced by Cole et al. [29] in their seminal paper for indexing a string of length n with the aim of answering pattern matching queries with up to k mismatches. For constant k , the size of the k -errata tree is $\mathcal{O}(n \log^k n)$. (Note that computing a k -LCS using the k -errata tree directly is not straightforward as opposed to computing an LCS using the suffix tree.)

We show the following result, breaking through the $n \log^k n$ barrier, for any constant integer $k > 0$ and irrespective of the alphabet size. Recall that, in the word RAM, the letters of S and T can be renumbered in $\mathcal{O}(n \log \log n)$ time [42] so that they belong to $[0, \sigma)$.

► **Theorem 2.** *Given two strings S and T , each of length at most n , over an integer alphabet and a constant integer $k > 0$, the k -LCS problem can be solved in $\mathcal{O}(n \log^{k-1/2} n)$ time using $\mathcal{O}(n)$ space.*

Notably, on the way to proving the above theorem, we improve upon [24] by showing an $\mathcal{O}(n + n \log^{k+1} n / \ell)$ -time algorithm for k -LCS with $k = \mathcal{O}(1)$, where ℓ is the length of a k -LCS. (Our second summand is smaller by a $\sqrt{\ell}$ multiplicative factor compared to [24].)

Our Techniques

At the heart of our approaches lies the following TWO STRING FAMILIES LCP PROBLEM. (Here, the length of the longest common prefix of two strings U and V is denoted by $\text{LCP}(U, V)$; see Preliminaries for a precise definition of compacted tries.)

TWO STRING FAMILIES LCP PROBLEM

Input: Compacted tries $\mathcal{T}(\mathcal{F}_1), \mathcal{T}(\mathcal{F}_2)$ of $\mathcal{F}_1, \mathcal{F}_2 \subseteq \Sigma^*$ and two sets $\mathcal{P}, \mathcal{Q} \subseteq \mathcal{F}_1 \times \mathcal{F}_2$, with $|\mathcal{P}|, |\mathcal{Q}|, |\mathcal{F}_1|, |\mathcal{F}_2| \leq N$.

Output: The value

$$\text{maxPairLCP}(\mathcal{P}, \mathcal{Q}) = \max\{\text{LCP}(P_1, Q_1) + \text{LCP}(P_2, Q_2) : (P_1, P_2) \in \mathcal{P}, (Q_1, Q_2) \in \mathcal{Q}\}.$$

This abstract problem was introduced in [24]. Its solution, shown in the lemma below, is directly based on a technique that was used in [19, 31] and then in [35] to devise an $\mathcal{O}(n \log n)$ -time solution for 1-LCS. In particular, Lemma 3 immediately implies an $\mathcal{O}(n \log n)$ -time algorithm for 1-LCS.

► **Lemma 3** ([24, Lemma 3]). *The TWO STRING FAMILIES LCP PROBLEM can be solved in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ space.¹*

In the algorithm underlying Lemma 3, for each node v of $\mathcal{T}(\mathcal{F}_1)$ we try to identify a pair of elements, one from \mathcal{P} and one from \mathcal{Q} , whose first components are descendants of v and the LCP of their second components is maximised. The algorithm traverses $\mathcal{T}(\mathcal{F}_1)$ bottom-up and uses mergeable height-balanced trees with $\mathcal{O}(N \log N)$ total merging time to store elements of pairs; see [20].

An $o(N \log N)$ time solution to the TWO STRING FAMILIES LCP PROBLEM is not known and devising such an algorithm seems difficult. The key ingredient of our algorithms is an efficient solution to the following special case of the problem. We say that a family of string pairs \mathcal{P} is an (α, β) -family if each $(U, V) \in \mathcal{P}$ satisfies $|U| \leq \alpha$ and $|V| \leq \beta$.

► **Lemma 4.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which \mathcal{P} and \mathcal{Q} are (α, β) -families can be solved in time $\mathcal{O}(N(\alpha + \log N)(\log \beta + \sqrt{\log N}) / \log N)$ and space $\mathcal{O}(N + N\alpha / \log N)$.*

The algorithm behind this solution uses a wavelet tree of the first components of $\mathcal{P} \cup \mathcal{Q}$.

¹ The original formulation of [24, Lemma 3] does not include a claim about the space complexity. However, it can be readily verified that the underlying algorithm, described in [31, 35], uses only linear space.

Solution to LCS. For the LCS problem, we design three different algorithms depending on the length of the solution. For short LCS ($\leq \frac{1}{3} \log_\sigma n$), we employ a simple tabulation technique. For long LCS ($\geq \log^4 n$), we employ the technique of Charalampopoulos et al. [24] for computing a long k -LCS, plugging in the sublinear longest common extension (LCE) data structure of Kempa and Kociumaka [47]. Both of these solutions work in $\mathcal{O}(n/\log_\sigma n)$ time.

As for medium-length LCS, let us first consider a case when the strings do not contain highly periodic fragments. In this case, we use the string synchronising sets of Kempa and Kociumaka [47] to select a set of $\mathcal{O}(\frac{n}{\tau})$ anchors over S and T , where $\tau = \Theta(\log_\sigma n)$, such that for any common substring U of S and T of length $\ell \geq 3\tau - 1$, there exist occurrences $S[i^S .. j^S]$ and $T[i^T .. j^T]$ of U , for which we have anchors $a^S \in [i^S, j^S]$ and $a^T \in [i^T, j^T]$ with $a^S - i^S = a^T - i^T \leq \tau$. For each anchor a in S , we add a string pair $((S[a - \tau .. a])^R, S[a .. a + \beta])$ to \mathcal{P} (and similarly for T and \mathcal{Q}). This lets us apply Lemma 4 with $N = \mathcal{O}(n/\tau)$, $\alpha = \mathcal{O}(\tau)$, and $\beta = \mathcal{O}(\log^4 n)$. In the periodic case, we cannot guarantee that $a^S - i^S = a^T - i^T$ is small, but we can obtain a different set of anchors based on maximal repetitions (runs) that yields multiple instances of the TWO STRING FAMILIES LCP PROBLEM, which have extra structure leading to a linear-time solution.

Solution to k -LCS. In this case we also obtain a set of $\mathcal{O}(n/\ell)$ anchors, where ℓ is the length of k -LCS. If the common substring is far from highly periodic, we use a synchronising set for $\tau = \Theta(\ell)$, and otherwise we generate anchors using a technique of misperiods that was initially introduced for k -mismatch pattern matching [18, 26]. Now the families \mathcal{P} , \mathcal{Q} need to consist not simply of substrings of S and T , but rather of modified substrings generated by an approach that resembles k -errata trees [29]. This requires combining the ideas of Thankachan et al. [59] and Charalampopoulos et al. [24]; this turns out to be technically challenging in order to stay within linear space. We apply Lemma 3 or Lemma 4 depending on the length ℓ , which allows us to break through the $n \log^k n$ barrier for k -LCS.

Other Related Work

A large body of work has been devoted to exploiting bit-parallelism in the word RAM model for string matching [7, 57, 55, 36, 37, 48, 14, 21, 9, 15, 11, 17, 39, 12, 16].

Other results on the LCS problem include the linear-time computation of an LCS of several strings over an integer alphabet [43], trade-offs between the time and the working space for computing an LCS of two strings [13, 50, 56], and the dynamic maintenance of an LCS [2, 3, 25]. Very recently, a strongly sublinear-time quantum algorithm and a lower bound for the quantum setting were shown [38]. The k -LCS problem has also been studied under edit distance and subquadratic-time algorithms for $k = o(\log n)$ are known [58, 4].

The problem of indexing a string of length n over an alphabet $[0, \sigma]$ in sublinear time in the word RAM model, with the aim of answering pattern matching queries, has attracted significant attention. Since by definition the suffix tree occupies $\Theta(n)$ space, alternative indexes have been sought. The state of the art is an index that occupies $\mathcal{O}(n \log \sigma / \log n)$ space and can be constructed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time [47, 53]. Interestingly, the running time of our algorithm (Theorem 1) matches the construction time of this index. Note that, in contrast to suffix trees, such indexes *cannot be used directly* for computing an LCS. Intuitively, these indexes sample suffixes of the string to be indexed, and upon a pattern matching query, they have to treat separately the first $\mathcal{O}(\log_\sigma n)$ letters of the pattern.

As for k -mismatch indexing, for $k = \mathcal{O}(1)$, a k -errata tree occupies $\mathcal{O}(n \log^k n)$ space, can be constructed in $\mathcal{O}(n \log^{k+1} n)$ time, and supports pattern matching queries with at most k mismatches in $\mathcal{O}(m + \log^k n \log \log n + occ)$ time, where m is the length of the pattern and occ

is the number of the reported pattern occurrences. Other trade-offs for this problem, in which the product of space and query time is still $\Omega(n \log^{2k} n)$, were shown in [23, 60], and solutions with $\mathcal{O}(n)$ space but $\Omega(\min\{n, \sigma^k m^{k-1}\})$ -time queries were presented in [22, 27, 44, 61]. More efficient solutions for $k = 1$ are known (see [10] and references therein). Cohen-Addad et al. [28] showed that, under SETH, for $k = \Theta(\log n)$ any indexing data structure that can be constructed in polynomial time cannot have $\mathcal{O}(n^{1-\delta})$ query time, for any $\delta > 0$. They also showed that in the pointer machine model, for $k = o(\log n)$, exponential dependency on k either in the space or in the query time cannot be avoided (for the reporting version of the problem). We hope that our techniques can fuel further progress in k -mismatch indexing.

2 Preliminaries

Strings. Let $T = T[1]T[2] \cdots T[n]$ be a *string* (or *text*) of length $n = |T|$ over an alphabet $\Sigma = [0, \sigma)$. The elements of Σ are called *letters*.

By ε we denote the *empty string*. For two positions i and j of T , we denote by $T[i..j]$ the *fragment* of T that starts at position i and ends at position j (the fragment is empty if $i > j$). A fragment of T is represented using $\mathcal{O}(1)$ space by specifying the indices i and j . We define $T[i..j] = T[i..j-1]$ and $T(i..j) = T[i+1..j]$. The fragment $T[i..j]$ is an *occurrence* of the underlying *substring* $P = T[i] \cdots T[j]$. We say that P occurs at *position* i in T . A *prefix* of T is a fragment of T of the form $T[1..j]$ and a *suffix* of T is a fragment of T of the form $T[i..n]$. We denote the *reverse string* of T by T^R , i.e., $T^R = T[n]T[n-1] \cdots T[1]$. By UV we denote the *concatenation* of two strings U and V , i.e., $UV = U[1]U[2] \cdots U[|U|]V[1]V[2] \cdots V[|V|]$.

A positive integer p is called a *period* of a string T if $T[i] = T[i+p]$ for all $i \in [1, |T| - p]$. We refer to the smallest period as *the period* of the string, and denote it by $\text{per}(T)$. A string T is called *periodic* if $\text{per}(T) \leq |T|/2$ and *aperiodic* otherwise. A *run* in T is a periodic substring that cannot be extended (to the left nor to the right) without an increase of its shortest period. All runs in a string can be computed in linear time [8, 51, 32].

► **Lemma 5** (Periodicity Lemma (weak version) [34]). *If a string S has periods p and q such that $p + q \leq |S|$, then $\text{gcd}(p, q)$ is also a period of S .*

Tries. Let \mathcal{M} be a finite set containing $m > 0$ strings over Σ . The *trie* of \mathcal{M} , denoted by $\mathcal{R}(\mathcal{M})$, contains a node for every distinct prefix of a string in \mathcal{M} ; the root node is ε ; the set of leaf nodes is \mathcal{M} ; and edges are of the form $(u, \alpha, u\alpha)$, where u and $u\alpha$ are nodes and $\alpha \in \Sigma$. The *compacted trie* of \mathcal{M} , denoted by $\mathcal{T}(\mathcal{M})$, contains the root, the branching nodes, and the leaf nodes of $\mathcal{R}(\mathcal{M})$. Each maximal branchless path segment from $\mathcal{R}(\mathcal{M})$ is replaced by a single edge, and a fragment of a string $M \in \mathcal{M}$ is used to represent the label of this edge in $\mathcal{O}(1)$ space. The best known example of a compacted trie is the suffix tree [62]. Throughout our algorithms, \mathcal{M} always consists of a set of substrings or modified substrings with $k = \mathcal{O}(1)$ modifications (see Section 5 for a definition) of a reference string. The value $\text{val}(u)$ of a node u is the concatenation of labels of edges on the path from the root to u , and the *string-depth* of u is the length of $\text{val}(u)$. The size of $\mathcal{T}(\mathcal{M})$ is $\mathcal{O}(m)$. We use the following well-known construction (cf. [30]).

► **Lemma 6.** *Given a sorted list of N strings and the longest common prefixes between pairs of consecutive strings, the compacted trie of the strings can be constructed in $\mathcal{O}(N)$ time.*

Packed strings. We assume the unit-cost word RAM model with word size $w = \Theta(\log n)$ and a standard instruction set including arithmetic operations, bitwise Boolean operations, and shifts. We count the space complexity of our algorithms in machine words used by the

algorithm. The *packed representation* of a string T over alphabet $[0, \sigma)$ is a list obtained by storing $\Theta(\log_\sigma n)$ letters per machine word thus representing T in $\mathcal{O}(|T|/\log_\sigma n)$ machine words. If T is given in the packed representation we simply say that T is a *packed string*.

String synchronising sets. Our solution uses the string synchronising sets introduced by Kempa and Kociumaka [47]. Informally, in the simpler case that T is cube-free, a τ -synchronising set of T is a small set of *synchronising positions* in T such that each length- τ fragment of T contains at least one synchronising position, and the leftmost synchronising positions within two sufficiently long matching fragments of T are consistent.

Formally, for a string T and a positive integer $\tau \leq \frac{1}{2}n$, a set $A \subseteq [1, n - 2\tau + 1]$ is a τ -synchronising set of T if it satisfies the following two conditions:

1. If $T[i..i + 2\tau] = T[j..j + 2\tau]$, then $i \in A$ if and only if $j \in A$.
2. For $i \in [1, n - 3\tau + 2]$, $A \cap [i, i + \tau] = \emptyset$ if and only if $\text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau$.

► **Theorem 7** ([47, Proposition 8.10, Theorem 8.11]). *For a string $T \in [0, \sigma)^n$ with $\sigma = n^{\mathcal{O}(1)}$ and $\tau \leq \frac{1}{2}n$, there exists a τ -synchronising set of size $\mathcal{O}(n/\tau)$ that can be constructed in $\mathcal{O}(n)$ time or, if $\tau \leq \frac{1}{5}\log_\sigma n$, in $\mathcal{O}(n/\tau)$ time if T is given in a packed representation.*

As in [47], for a τ -synchronising set A , let $\text{succ}_A(i) := \min\{j \in A \cup \{n - 2\tau + 2\} : j \geq i\}$.

► **Lemma 8** ([47, Fact 3.2]). *If $p = \text{per}(T[i..i + 3\tau - 2]) \leq \frac{1}{3}\tau$, then $T[i.. \text{succ}_A(i) + 2\tau - 1]$ is the longest prefix of $T[i..|T|]$ with period p .*

► **Lemma 9** ([47, Fact 3.3]). *If a string U with $|U| \geq 3\tau - 1$ and $\text{per}(U) > \frac{1}{3}\tau$ occurs at positions i and j in T , then $\text{succ}_A(i) - i = \text{succ}_A(j) - j \leq |U| - 2\tau$.*

A τ -run R is a run of length at least $3\tau - 1$ with period at most $\frac{1}{3}\tau$. The *Lyndon root* of R is the lexicographically smallest cyclic shift of $R[1.. \text{per}(R)]$. A proof of the following lemma resembles an argument given in [47, Section 6.1.2]; its proof can be found in the full version of this paper.

► **Lemma 10.** *For a positive integer τ , a string $T \in [0, \sigma)^n$ contains $\mathcal{O}(n/\tau)$ τ -runs. Moreover, if $\tau \leq \frac{1}{5}\log_\sigma n$, given a packed representation of T , we can compute all τ -runs in T and group them by their Lyndon roots in $\mathcal{O}(n/\tau)$ time. Within the same complexities, for each τ -run, we can compute the two leftmost occurrences of its Lyndon root.*

► **Theorem 11** ([47, Theorem 4.3]). *Given a packed representation of a string $T \in [0, \sigma)^n$ and a τ -synchronising set A of T of size $\mathcal{O}(n/\tau)$ for $\tau = \mathcal{O}(\log_\sigma n)$, we can compute in $\mathcal{O}(n/\tau)$ time the lexicographic order of all suffixes of T starting at positions in A .*

We often want to preprocess T to be able to answer queries $\text{LCP}(T[i..n], T[j..n])$ [52]. For this case, there exists an optimal data structure that applies synchronising sets.

► **Theorem 12** ([47, Theorem 5.4]). *Given a packed representation of a string $T \in [0, \sigma)^n$, LCP queries on T can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$ -time preprocessing.*

3 Sublinear-Time LCS

We provide different solutions depending on the length ℓ of an LCS. Lemmas 13, 14, and 17 directly yield Theorem 1.

The proof of the following lemma, for the case where an LCS is short, i.e., of length $\ell \leq \frac{1}{3}\log_\sigma n$, uses tabulation and can be found in the full version of this paper.

► **Lemma 13.** *The LCS problem can be solved in $\mathcal{O}(n/\log_\sigma n)$ time if $\ell \leq \frac{1}{3} \log_\sigma n$.*

The proof of the following lemma, for the case where an LCS is long, i.e., of length $\ell = \Omega(\frac{\log^4 n}{\log^2 \sigma})$, uses difference covers and the $\mathcal{O}(N \log N)$ -time solution to the TWO STRING FAMILIES LCP PROBLEM. This proof closely follows [24] and can be found in the full version of this paper.

► **Lemma 14.** *The LCS problem can be solved in $\mathcal{O}(n/\log_\sigma n)$ time if $\ell = \Omega(\frac{\log^4 n}{\log^2 \sigma})$.*

Solution for medium-length LCS. We now give a solution to the LCS problem for $\ell \in [\frac{1}{3} \log_\sigma n, 2\sqrt{\log n}]$. We first construct three subsets of positions in $S\$T$, where $\$ \notin \Sigma$, of size $\mathcal{O}(n/\log_\sigma n)$ as follows. For $\tau = \lfloor \frac{1}{9} \log_\sigma n \rfloor$, let A_I be a τ -synchronising set of $S\$T$. For each τ -run in $S\$T$, we insert to A_{II} the starting positions of the first two occurrences of the Lyndon root of the τ -run and to A_{III} the last position of the τ -run. The elements of A_{II} and A_{III} store the τ -run they originate from. Finally, we denote $A_j^S = A_j \cap [1, |S|]$ and $A_j^T = \{a - |S| - 1 : a \in A_j, a > |S| + 1\}$ for $j = I, II, III$. The following lemma shows that there exists an LCS of S and T for which $A_I \cup A_{II} \cup A_{III}$ is a set of *anchors* that satisfies certain distance requirements.

► **Lemma 15.** *If an LCS of S and T has length $\ell \geq 3\tau$, then there exist positions $i^S \in [1, |S|]$, $i^T \in [1, |T|]$, a shift $\delta \in [0, \ell)$, and $j \in \{I, II, III\}$ such that $S[i^S \dots i^S + \ell] = T[i^T \dots i^T + \ell]$, $i^S + \delta \in A_j^S$, $i^T + \delta \in A_j^T$, and*

- *if $j = I$, then $\delta \in [0, \tau)$;*
- *if $j = II$, then $S[i^S \dots i^S + \ell]$ is contained in the τ -run from which $i^S + \delta \in A^S$ originates;*
- *if $j = III$, then $\delta \geq 3\tau - 1$ and $S[i^S \dots i^S + \delta]$ is a suffix of the τ -run from which $i^S + \delta \in A^S$ originates.*

Proof. By the assumption, there exist $i^S \in [1, |S|]$ and $i^T \in [1, |T|]$ such that $S[i^S \dots i^S + \ell] = T[i^T \dots i^T + \ell]$. Let us choose any such pair (i^S, i^T) minimising the sum $i^S + i^T$. We have the following cases.

1. If $\text{per}(S[i^S \dots i^S + 3\tau - 2]) > \frac{1}{3}\tau$, then, by the definition of a τ -synchronising set, in this case there exist some elements $a^S \in A_I^S \cap [i^S, i^S + \tau)$ and $a^T \in A_I^T \cap [i^T, i^T + \tau)$. Let us choose the smallest such elements. By Lemma 9, we have $a^S - i^S = a^T - i^T$.
2. Else, $p = \text{per}(S[i^S \dots i^S + 3\tau - 2]) \leq \frac{1}{3}\tau$. We have two subcases.
 - a. If $p = \text{per}(S[i^S \dots i^S + \ell])$, then, by the choice of i^S and i^T there exists a τ -run R_S in S that starts at position in $(i^S - p \dots i^S)$ and a τ -run R_T in T that starts at position in $(i^T - p \dots i^T)$. Moreover, by Lemma 5, both runs have equal Lyndon roots. For each $X \in \{S, T\}$, let us choose a^X as the leftmost starting position of a Lyndon root of R_X that is $\geq i^X$. We have $a^S - i^S = a^T - i^T \in [0, \frac{1}{3}\tau)$. Each position a^X will be the starting position of the first or the second occurrence of the Lyndon root of R_S , so $a^S \in A_{II}^S$ and $a^T \in A_{II}^T$.
 - b. Else, $p \neq \text{per}(S[i^S \dots i^S + \ell])$. We have $d = \min\{b \geq p : S[i^S + b] \neq S[i^S + b - p]\} < \ell$ (and $d \geq 3\tau - 1$). In this case, $a^S = i^S + d - 1$ and $a^T = i^T + d - 1$ are the ending positions of τ -runs with period p in S and T , respectively, so $a^S \in A_{III}^S$ and $a^T \in A_{III}^T$. ◀

Case $j = I$ from the above lemma corresponds to the TWO STRING FAMILIES LCP PROBLEM with \mathcal{P} and \mathcal{Q} being $(\tau, 2\sqrt{\log n})$ -families. Let us introduce a variant of the TWO STRING FAMILIES LCP PROBLEM that intuitively corresponds to the case $j \in \{II, III\}$. A family of string pairs \mathcal{P} is called a *prefix family* if there exists a string Y such that, for each

$(U, V) \in \mathcal{P}$, U is a prefix of Y . We arrive at this special case with first components of \mathcal{P} and \mathcal{Q} being prefixes of some cyclic shift of a power of a (common) Lyndon root of τ -runs. The proof of the following lemma can be found in the full version of this paper.

► **Lemma 16.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which $\mathcal{P} \cup \mathcal{Q}$ is a prefix family can be solved in $\mathcal{O}(N)$ time.*

We are now ready to state the main result of this subsection.

► **Lemma 17.** *The LCS problem can be solved in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time using $\mathcal{O}(n / \log_\sigma n)$ space if $\ell \in [\frac{1}{3} \log_\sigma n, 2^{\sqrt{\log n}}]$.*

Proof. Recall that $\tau = \lfloor \frac{1}{9} \log_\sigma n \rfloor$. The set of anchors $A = A_I \cup A_{II} \cup A_{III}$ consists of a τ -synchronising set and of $\mathcal{O}(1)$ positions per each τ -run in $S\$T$. Hence, $|A| = \mathcal{O}(n/\tau)$ and A can be constructed in $\mathcal{O}(n/\tau)$ time by Theorem 7 and Lemma 10.

We construct sets of pairs of substrings of $X = S\$_1T\$_2S^R\$_3T^R$. First, for $\Delta = \lfloor 2^{\sqrt{\log n}} \rfloor$:

$$\mathcal{P}_I = \{((S[a - \tau..a])^R, S[a..a + \Delta]) : a \in A_I^S\}.$$

Then, for each group \mathcal{G} of τ -runs in S and T with equal Lyndon root, we construct the following set of string pairs:

$$\mathcal{P}_{II}^{\mathcal{G}} = \{((S[x..a])^R, S[a..y]) : a \in A_{II}^S \text{ that originates from } \tau\text{-run } S[x..y] \in \mathcal{G}\}.$$

We define the *tail* of a τ -run $S[i..j]$ with period p and Lyndon root $S[i'..i' + p]$ as $(j + 1 - i') \bmod p$ (and same for τ -runs in T). For each group of τ -runs in S and T with equal Lyndon roots, we group the τ -runs belonging to it by their tails. This can be done in $\mathcal{O}(n/\tau)$ time using tabulation, since the tail values are up to $\frac{1}{3}\tau$. For each group \mathcal{G} of τ -runs in S and T with equal Lyndon root and tail, we construct the following set of string pairs:

$$\mathcal{P}_{III}^{\mathcal{G}} = \{((S[x..y])^R, S[y..|S|]) : S[x..y] \in \mathcal{G}\}.$$

Simultaneously, we create sets \mathcal{Q}_I , $\mathcal{Q}_{II}^{\mathcal{G}}$ and $\mathcal{Q}_{III}^{\mathcal{G}}$ defined with T instead of S .

Now, it suffices to output the maximum of $\text{maxPairLCP}(\mathcal{P}_I, \mathcal{Q}_I)$, $\text{maxPairLCP}(\mathcal{P}_{II}^{\mathcal{G}}, \mathcal{Q}_{II}^{\mathcal{G}})$, and $\text{maxPairLCP}(\mathcal{P}_{III}^{\mathcal{G}}, \mathcal{Q}_{III}^{\mathcal{G}})$, where \mathcal{G} ranges over groups of τ -runs. Computing any individual maxPairLCP value can be expressed as an instance of the TWO STRING FAMILIES LCP PROBLEM provided that all the first and second components of families are represented as nodes of compacted tries. We will use Lemma 6 to construct these compacted tries. LCP queries can be answered efficiently due to Theorem 12, so it suffices to be able to sort all the first and second components of each pair of string pair sets lexicographically. Each of the sets \mathcal{P}_I , \mathcal{Q}_I can be ordered by the second components using Theorem 11 since A_I is a τ -synchronising set, and by the first components with easy preprocessing using the fact that the number of possible τ -length strings is $\sigma^\tau = \mathcal{O}(n^{1/9})$. In a set $\mathcal{P}_{II}^{\mathcal{G}}$, both all first and all second components are prefixes of a single string (a power of the common Lyndon root). Hence, they can be sorted simply by comparing their lengths. This sorting is performed simultaneously for all the families $\mathcal{P}_{II}^{\mathcal{G}}$, $\mathcal{Q}_{II}^{\mathcal{G}}$ in $\mathcal{O}(n/\tau)$ time via radix sort. Finally, to sort the second components of the sets $\mathcal{P}_{III}^{\mathcal{G}}$, $\mathcal{Q}_{III}^{\mathcal{G}}$, instead of comparing strings of the form $S[y..|S|]$ (and same for T), we can equivalently compare strings $S[y - 2\tau + 1..|S|]$ which are known to start at positions from a τ -synchronising set by Lemma 8. This sorting is done across all groups using radix sort and Theorem 11. The correctness follows by Lemma 15.

Finally, we observe that $(\mathcal{P}_I, \mathcal{Q}_I)$ is a (τ, Δ) -family of size $N = \mathcal{O}(n/\tau)$, and thus $\text{maxPairLCP}(\mathcal{P}_I, \mathcal{Q}_I)$ can be computed in $\mathcal{O}(n \log \sigma / \sqrt{\log n})$ time and $\mathcal{O}(n/\log_\sigma n)$ space using Lemma 4. On the other hand, $(\mathcal{P}_{II}^g, \mathcal{Q}_{II}^g)$ and $(\mathcal{P}_{III}^g, \mathcal{Q}_{III}^g)$ are prefix families of total size $\mathcal{O}(n/\tau)$, so the corresponding instances of the TWO STRING FAMILIES LCP PROBLEM can be solved in $\mathcal{O}(n/\log_\sigma n)$ total time using Lemma 16. ◀

4 Proof of Lemma 4 via Wavelet Trees

Wavelet trees. For an arbitrary alphabet Σ , the *skeleton tree* for Σ is a full binary tree \mathcal{T} together with a bijection between Σ and the leaves of \mathcal{T} . For a node $v \in \mathcal{T}$, let Σ_v denote the subset of Σ that corresponds to the leaves in the subtree of v .

The \mathcal{T} -shaped *wavelet tree* of a string $T \in \Sigma^*$ consists of bit vectors assigned to internal nodes of \mathcal{T} (inspect Figure 1(a)). For an internal node v with children v_L and v_R , let T_v denote the maximal subsequence of T that consists of letters from Σ_v ; the bit vector $B_v[1..|T_v|]$ is defined so that $B_v[i] = 0$ if $T_v[i] \in \Sigma_{v_L}$ and $B_v[i] = 1$ if $T_v[i] \in \Sigma_{v_R}$.

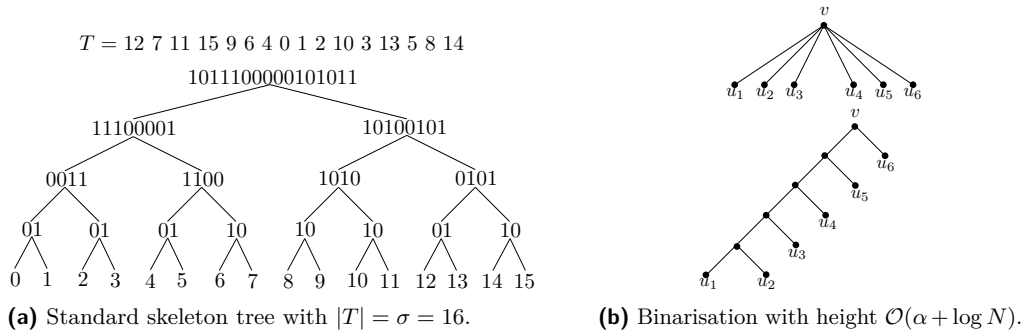


Figure 1 (a) Let v be left child of the root node. Then $\Sigma_v = \{0, 1, \dots, 7\}$, $T_v = 7, 6, 4, 0, 1, 2, 3, 5$ and so $B_v = 11100001$: 7, 6, 4, 5 belong to the right subtree of v and 0, 1, 2, 3 to the left. (b) For each i , let the size of the subtree rooted at u_i be 2^i . The binarisation from [5] leads to height $\mathcal{O}(\alpha + \log N)$, favouring heavier children.

Wavelet trees were introduced in [41], whereas efficient construction algorithms were presented in [54, 5].

► **Theorem 18** (see [54, Theorem 2]). *Given the packed representation of a string $T \in [0, \sigma]^n$ and a skeleton tree \mathcal{T} of height h , the \mathcal{T} -shaped wavelet tree of T takes $\mathcal{O}(nh/\log n + \sigma)$ space and can be constructed in $\mathcal{O}(nh/\sqrt{\log n} + \sigma)$ time.*

Wavelet trees are sometimes constructed for sequences $T \in \mathcal{M}^*$ over an alphabet $\mathcal{M} \subseteq \Sigma^*$ that itself consists of strings (see e.g. [47]). In this case, the skeleton tree \mathcal{T} is often chosen to resemble the compacted trie of \mathcal{M} . Formally, we say that a skeleton tree \mathcal{T} for \mathcal{M} is *prefix-consistent* if each node $v \in \mathcal{T}$ admits a label $\text{val}(v) \in \Sigma^*$ such that:

- if v is a leaf, then $\text{val}(v)$ is the corresponding string in \mathcal{M} ;
- if v is a node with children v_L, v_R , then, for all leaves u_L, u_R in the subtrees of v_L and v_R , respectively, the string $\text{val}(v)$ is the longest common prefix of $\text{val}(u_L)$ and $\text{val}(u_R)$.

Observe that if $\mathcal{M} \subseteq \{0, 1\}^\alpha$ for some integer α , then the compacted trie $\mathcal{T}(\mathcal{M})$ is a prefix-consistent skeleton tree for \mathcal{M} . For larger alphabets, we binarise $\mathcal{T}(\mathcal{M})$ as follows:

► **Lemma 19.** *Given the compacted trie $\mathcal{T}(\mathcal{M})$ of a set $\mathcal{M} \subseteq \Sigma^\alpha$, a prefix-consistent skeleton tree of height $\mathcal{O}(\alpha + \log |\mathcal{M}|)$ can be constructed in $\mathcal{O}(|\mathcal{M}|)$ time, with each node v associated to a node v' of $\mathcal{T}(\mathcal{M})$ such that $\text{val}(v) = \text{val}(v')$.*

Proof. We use [5, Corollary 3.2], where the authors showed that any rooted tree of size m and height h can be binarised in $\mathcal{O}(m)$ time so that the resulting tree is of height $\mathcal{O}(h + \log m)$; see Figure 1(b). For $\mathcal{T}(\mathcal{M})$, we obtain height $\mathcal{O}(\alpha + \log |\mathcal{M}|)$ and time $\mathcal{O}(|\mathcal{M}|)$. ◀

► **Lemma 4.** *An instance of the TWO STRING FAMILIES LCP PROBLEM in which \mathcal{P} and \mathcal{Q} are (α, β) -families can be solved in time $\mathcal{O}(N(\alpha + \log N)(\log \beta + \sqrt{\log N})/\log N)$ and space $\mathcal{O}(N + N\alpha/\log N)$.*

Proof. By traversing $\mathcal{T}(\mathcal{F}_2)$ we can compute in $\mathcal{O}(N)$ time a list \mathcal{R} being a union of sets \mathcal{P} and \mathcal{Q} in which the second components are ordered lexicographically. We also store a bit vector G of length $|\mathcal{R}|$ that determines, for each element of \mathcal{R} , which of the sets \mathcal{P} , \mathcal{Q} it originates from. We construct the wavelet tree of the sequence of strings being the first components of pairs from \mathcal{R} using Theorem 18 and the skeleton tree from Lemma 19. Before the wavelet tree is constructed, we pad each string with a letter $\$ \notin \Sigma$ to make them all of length α ; we will ignore the nodes of the wavelet tree with a path-label containing a $\$$.

For a sublist $\mathcal{X} = (U_1, V_1), \dots, (U_m, V_m)$ of \mathcal{R} , by $\text{LCPs}(\mathcal{X})$ we denote the representation of the list $0, \text{LCP}(V_1, V_2), \dots, \text{LCP}(V_{m-1}, V_m)$ as a packed string over alphabet $[0, \beta]$ in space $\mathcal{O}(N/\log_\beta N)$. Together with each $\text{LCPs}(\mathcal{X})$ we also store the bit vector $G(\mathcal{X})$ of origins of elements of \mathcal{X} without increasing the complexity. The list $\text{LCPs}(\mathcal{R})$ can be computed in $\mathcal{O}(N)$ time when constructing \mathcal{R} . For each node v of the wavelet tree, we wish to compute $\mathcal{L}_v = \text{LCPs}(\mathcal{R}_v)$, where \mathcal{R}_v is the sublist of \mathcal{R} composed of elements whose first component is in the leaf list Σ_v of v . We will construct the lists $\text{LCPs}(\mathcal{R}_v)$ without actually computing \mathcal{R}_v .

Computation of LCPs lists. The lists are computed recursively using the bit vectors from the wavelet tree. Assume we have computed \mathcal{L}_u and wish to compute \mathcal{L}_v for the left child v of u – the computations for the right child are symmetric.

Let $c \in (0, 1)$ be a constant. Let us partition \mathcal{L}_u into blocks of $\lambda = \max(1, \lfloor c \log_\beta N \rfloor)$ LCP values. We will process the blocks in order, constructing \mathcal{L}_v . Each block of \mathcal{L}_u can be represented in a single word and this representation can be extracted from the packed representation of \mathcal{L}_u in $\mathcal{O}(1)$ time. For each block $W = \mathcal{L}_u(a\lambda \dots (a+1)\lambda)$, we retrieve in $\mathcal{O}(1)$ time the corresponding block $D = B_u(a\lambda \dots (a+1)\lambda)$ in the bit vector from the wavelet tree. Further, we store $\mu_a = \min \mathcal{L}_v(p_a \dots a\lambda)$, where $p_a = \max\{i \in [0, a\lambda) : i = 0 \text{ or } B_u[i] = 0\}$. Let us show how, given W , D and μ_a , we can determine μ_{a+1} and the chunk of \mathcal{L}_v that corresponds to $i \in [1, \lambda]$ such that $D[i] = 0$. The calculations are based on the following well-known fact.

► **Fact 20.** *If U_1, U_2, U_3 are strings such that $U_1 \leq U_2 \leq U_3$, then we have $\text{LCP}(U_1, U_3) = \min(\text{LCP}(U_1, U_2), \text{LCP}(U_2, U_3))$.*

For each $i \in [1, \lambda]$ such that $D[i] = 0$, in increasing order, if a previous position j with $D[j] = 0$ exists, then $\min W(i' \dots i)$ should be appended to \mathcal{L}_v , where i' is the predecessor of i satisfying $D[i'] = 0$, and otherwise $\min(\{\mu_a\} \cup W[1 \dots i])$ should. Then, for the last position $i \in [1, \lambda]$ such that $D[i] = 0$, $\mu_{a+1} = \min W(i \dots \lambda)$, and if no such position exists, then $\mu_{a+1} = \min(\{\mu_a\} \cup W[1 \dots \lambda])$.

If $\lambda = 1$, the calculations can be performed in $\mathcal{O}(1)$ time. Otherwise, we make use of preprocessing: for every possible combination of (W, D, μ_a) , i.e., up to $2c \log N + \log \beta < 3c \log N$ bits, precompute the block to be appended to \mathcal{L}_v and μ_{a+1} , i.e., up to $c \log N + \log \beta < 2c \log N$ bits. We can choose $c > 0$ small enough to make the preprocessing $\mathcal{O}(N^{1-\varepsilon})$ for some $\varepsilon > 0$. Thus, the computation takes $\mathcal{O}(|\mathcal{L}_u|/\lambda)$ time. Within this time, we can also populate the bit vector of origins for v .

Application of LCPs lists. For each node u of the wavelet tree, we must extract the maximum LCP between suffixes of different origins, add the string-depth $|\text{val}(u)|$, and compare the result with the stored candidate. The former can be computed in $\mathcal{O}(|\mathcal{L}_u|/\log_\beta N)$ time as follows. Due to Fact 20, the answer will be the LCP between a pair of second components of consecutive elements of \mathcal{R}_u that originate from different sets, i.e. $\max\{\mathcal{L}_u[i] : i \in [2, |\mathcal{R}_u|], G(\mathcal{R}_u)[i-1] \neq G(\mathcal{R}_u)[i]\}$. We can cover all pairs of consecutive elements of \mathcal{L}_u using $\Theta(1 + |\mathcal{L}_u|/\log_\beta N)$ blocks of $\max(2, \lfloor c \log_\beta N \rfloor)$ LCP values. Each such block, augmented with its corresponding bit vector of origins, consists of at most $2c \log N$ bits. We can thus precompute all possible answers in $\mathcal{O}(N^{1-\epsilon})$ time, and then process each node u in $\Theta(1 + |\mathcal{L}_u|/\log_\beta N)$ time.

Time complexity. The wavelet tree can be built in $\mathcal{O}(Nh/\sqrt{\log N})$ time using space $\mathcal{O}(Nh/\log N)$ by Theorem 18, where $h = \mathcal{O}(\alpha + \log N)$. Computing LCPs for children of a single node u takes $\mathcal{O}(1 + |\mathcal{L}_u|/\log_\beta N)$ time; over all nodes, this is $\mathcal{O}(Nh \log \beta / \log N)$. ◀

5 Faster k -LCS

In this section, we outline our $\mathcal{O}(n \log^{k-1/2} n)$ -time algorithm for the k -LCS problem with $k = \mathcal{O}(1)$, that underlies Theorem 2. For simplicity, we focus on computing the length of a k -LCS; an actual pair of strings forming a k -LCS can be recovered easily from our approach. If the length of an LCS of S and T is d , then the length of a k -LCS of S and T is in $[d, (k+1)d + k]$. Below, we show how to compute a k -LCS provided that it belongs to an interval $(\ell/2, \ell]$ for a specified ℓ ; it is sufficient to call this subroutine $\mathcal{O}(\log k) = \mathcal{O}(1)$ times.

Similarly to our solutions for long and medium-length LCS, we first distinguish anchors $A^S \subseteq [1, |S|]$ in S and $A^T \subseteq [1, |T|]$ in T , as summarised in the following lemma.

► **Lemma 21.** *Consider an instance of the k -LCS problem for $k = \mathcal{O}(1)$ and let $\ell \in [1, n]$. In $\mathcal{O}(n)$ time, one can construct sets $A^S \subseteq [1, |S|]$ and $A^T \subseteq [1, |T|]$ of size $\mathcal{O}(\frac{n}{\ell})$ satisfying the following condition: If a k -LCS of S and T has length $\ell' \in (\ell/2, \ell]$, then there exist positions $i^S \in [1, |S|]$, $i^T \in [1, |T|]$ and a shift $\delta \in [0, \ell')$ such that $i^S + \delta \in A^S$, $i^T + \delta \in A^T$, and the Hamming distance between $S[i^S \dots i^S + \ell')$ and $T[i^T \dots i^T + \ell')$ is at most k .*

Proof. As in [26], we say that position a in a string X is a *misperiod* with respect to a substring $X[i \dots j]$ if $X[a] \neq X[b]$, where b is the unique position such that $b \in [i, j]$ and $(j-i) \mid (b-a)$; for example, $j-i$ is a period of X if and only if there are no misperiods with respect to $X[i \dots j]$. We define the set $\text{LeftMisper}_k(X, i, j)$ as the set of k maximal misperiods that are smaller than i and $\text{RightMisper}_k(X, i, j)$ as the set of k minimal misperiods that are not smaller than j . Either set may have fewer than k elements if the corresponding misperiods do not exist. Further, let us define $\text{Misper}_k(X, i, j) = \text{LeftMisper}_k(X, i, j) \cup \text{RightMisper}_k(X, i, j)$ and $\text{Misper}(X, i, j) = \bigcup_{k=0}^{\infty} \text{Misper}_k(X, i, j)$.

Similar to Lemma 15, we construct three subsets of positions in $Y = \#S\$T$, where $\#, \$ \notin \Sigma$. For $\tau = \lfloor \ell / (6(k+1)) \rfloor$, let A_I be a τ -synchronising set of Y . Let $Y[i \dots j]$ be a τ -run with period p and assume that the first occurrence of its Lyndon root is at a position q of Y . Then, for $Y[i \dots j]$, for each $x \in \text{LeftMisper}_{k+1}(Y, i, i+p)$, we insert to A_{II} the two smallest positions in $[x+1, |Y|]$ that are equivalent to $q \pmod{p}$. Moreover, we insert to A_{III} the positions in $\text{Misper}_{k+1}(Y, i, i+p)$. Finally, we denote $A = A_I \cup A_{II} \cup A_{III}$, as well as $A^S = \{a-1 : a \in A \cap [2, |S|+1]\}$ and $A^T = \{a-|S|-2 : a \in A \cap [|S|+3, |Y|]\}$. The proof of the following claim, that can be found in the full version of this paper, resembles that of Lemma 15.

▷ **Claim 22.** The sets A^S and A^T satisfy the condition stated in Lemma 21.

It remains to show that the sets A^S and A^T can be constructed efficiently. A τ -synchronising set can be computed in $\mathcal{O}(n)$ time by Theorem 7 and all the τ -runs, together with the position of the first occurrence of their Lyndon root, can be computed in $\mathcal{O}(n)$ time [8]. After an $\mathcal{O}(n)$ -time preprocessing, for every τ -run, we can compute the set of the $k + 1$ misperiods of its period to either side in $\mathcal{O}(1)$ time; see [26, Claim 18]. ◀

The next step in our solutions to long LCS and medium-length LCS was to construct an instance of the TWO STRING FAMILIES LCP PROBLEM. To adapt this approach, we generalise the notions of LCP and maxPairLCP so that they allow for mismatches. By $\text{LCP}_k(U, V)$, for $k \in \mathbb{Z}_{\geq 0}$, we denote the maximum length ℓ such that $U[1.. \ell]$ and $V[1.. \ell]$ are at Hamming distance at most k .

► **Definition 23.** *Given two sets $\mathcal{U}, \mathcal{V} \subseteq \Sigma^* \times \Sigma^*$ and two integers $k_1, k_2 \in \mathbb{Z}_{\geq 0}$, we define $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V}) = \max\{\text{LCP}_{k_1}(U_1, V_1) + \text{LCP}_{k_2}(U_2, V_2) : (U_1, U_2) \in \mathcal{U}, (V_1, V_2) \in \mathcal{V}\}$.*

Note that $\text{maxPairLCP}(\mathcal{U}, \mathcal{V}) = \text{maxPairLCP}_{0, 0}(\mathcal{U}, \mathcal{V})$.

By Lemma 21, if a k -LCS of S and T has length $\ell' \in (\ell/2, \ell]$, then

$$\ell' = \max_{k'=0}^k \text{maxPairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V}), \text{ for } \mathcal{U} = \{((S[a - \ell.. a])^R, S[a.. a + \ell]) : a \in A^S\},$$

$$\mathcal{V} = \{((T[a - \ell.. a])^R, T[a.. a + \ell]) : a \in A^T\}.$$

Here, k' bounds the number of mismatches between $S[i^S.. i^S + \delta]$ and $T[i^T.. i^T + \delta]$, whereas $k - k'$ bounds the number of mismatches between $S[i^S + \delta.. i^S + \ell']$ and $T[i^T + \delta.. i^T + \ell']$. The following theorem, whose full proof can be found in the full version of this paper, is the most technical part of our contribution, and allows for efficiently computing the values $\text{maxPairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V})$.

► **Theorem 24.** *Consider two (ℓ, ℓ) -families \mathcal{U}, \mathcal{V} of total size N consisting of pairs of substrings of a given length- n text. For any non-negative integers $k_1, k_2 = \mathcal{O}(1)$, the value $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V})$ can be computed:*

- in $\mathcal{O}(n + N \log^{k_1+k_2+1} N)$ time and $\mathcal{O}(n + N)$ space if $\ell > \log^{3/2} N$,
- in $\mathcal{O}(n + N \ell \log^{k_1+k_2-1/2} N)$ time and $\mathcal{O}(n + N \ell / \log N)$ space if $\log N < \ell \leq \log^{3/2} N$,
- in $\mathcal{O}(n + N \ell^{k_1+k_2} \sqrt{\log N})$ time and $\mathcal{O}(n + N)$ space if $\ell \leq \log N$.

Proof Outline. We reduce the computation of $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V})$ into multiple computations of $\text{maxPairLCP}(\mathcal{U}', \mathcal{V}')$ across a family \mathbf{P} of pairs $(\mathcal{U}', \mathcal{V}')$ with $\mathcal{U}', \mathcal{V}' \subseteq \Sigma^* \times \Sigma^*$. Each pair $(U'_1, U'_2) \in \mathcal{U}'$ is associated to a pair $(U_1, U_2) \in \mathcal{U}$, with the string U'_i represented as a pointer to the source U_i and up to k_i substitutions needed to transform U_i to U'_i . Similarly, each pair $(V'_1, V'_2) \in \mathcal{V}'$ consists of *modified strings* with sources $(V_1, V_2) \in \mathcal{V}$. In order to guarantee $\text{maxPairLCP}_{k_1, k_2}(\mathcal{U}, \mathcal{V}) = \max_{(\mathcal{U}', \mathcal{V}') \in \mathbf{P}} \text{maxPairLCP}(\mathcal{U}', \mathcal{V}')$, we require $\text{LCP}(U'_i, V'_i) \leq \text{LCP}_{k_i}(U_i, V_i)$ for every $(U'_1, U'_2) \in \mathcal{U}'$ and $(V'_1, V'_2) \in \mathcal{V}'$ with $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ and that, for every $(U_1, U_2) \in \mathcal{U}$ and $(V_1, V_2) \in \mathcal{V}$, there exists $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ with $(U'_1, U'_2) \in \mathcal{U}'$ and $(V'_1, V'_2) \in \mathcal{V}'$, with sources (U_1, U_2) and (V_1, V_2) , respectively, such that $\text{LCP}(U'_i, V'_i) = \text{LCP}_{k_i}(U_i, V_i)$. Our construction is based on a technique of [59] which gives an analogous family for two subsets of Σ^* (rather than $\Sigma^* \times \Sigma^*$) and a single threshold: We apply the approach of [59] to $\mathcal{U}_i = \{U_i : (U_1, U_2) \in \mathcal{U}\}$ and $\mathcal{V}_i = \{V_i : (V_1, V_2) \in \mathcal{V}\}$ with threshold k_i , and then combine the two resulting families \mathbf{P}_i to derive \mathbf{P} .

Strengthening the arguments of [59], we show that each string $F_i \in \mathcal{U}_i \cup \mathcal{V}_i$ is the source of $\mathcal{O}(1)$ modified strings $F'_i \in \mathcal{U}'_i \cup \mathcal{V}'_i$ for any single $(\mathcal{U}'_i, \mathcal{V}'_i) \in \mathbf{P}_i$ and $\mathcal{O}(\min(\ell, \log N)^{k_i})$ modified strings across all $(\mathcal{U}'_i, \mathcal{V}'_i) \in \mathbf{P}_i$. This allows bounding the size of individual sets

$(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$ by $\mathcal{O}(N)$ and the overall size by $\mathcal{O}(N \min(\ell, \log N)^{k_1+k_2})$. In order to efficiently build the compacted tries required at the input of the TWO STRING FAMILIES LCP PROBLEM, the modified strings $F'_i \in \mathcal{U}'_i \cup \mathcal{V}'_i$ are sorted lexicographically, and the two derived linear orders (for $i \in \{1, 2\}$) are maintained along with every pair $(\mathcal{U}', \mathcal{V}') \in \mathbf{P}$. Overall, the family \mathbf{P} is constructed in $\mathcal{O}(n + N \min(\ell, \log N)^{k_1+k_2})$ time and $\mathcal{O}(n + N)$ space.

The resulting instances of the TWO STRING FAMILIES LCP PROBLEM are solved using Lemma 3 (if $\ell > \log^{3/2} N$) or Lemma 4 otherwise; note that $\mathcal{U}', \mathcal{V}'$ are (ℓ, ℓ) -families. \blacktriangleleft

Recall that the algorithm of Theorem 24 is called $k + 1 = \mathcal{O}(1)$ times, always with $N = |A^S| + |A^T| = \mathcal{O}(n/\ell)$. Overall, the value $\max_{k'=0}^k \max \text{PairLCP}_{k', k-k'}(\mathcal{U}, \mathcal{V})$ is therefore computed in $\mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n)$ space in each of the following cases:

- in $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} N) = \mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n + \frac{n}{\ell}) = \mathcal{O}(n)$ space if $\ell > \log^{3/2} N$;
- in $\mathcal{O}(n + \frac{n}{\ell} \ell \log^{k-1/2} N) = \mathcal{O}(n \log^{k-1/2} n)$ time and $\mathcal{O}(n + \frac{n}{\ell} \ell / \log N) = \mathcal{O}(n)$ space if $\log N < \ell \leq \log^{3/2} N$;
- in $\mathcal{O}(n + \frac{n}{\ell} \ell^k \sqrt{\log N}) = \mathcal{O}(n \log^{k-1} n)$ time and $\mathcal{O}(n + \frac{n}{\ell}) = \mathcal{O}(n)$ space if $\ell \leq \log N$.

Accounting for $\mathcal{O}(n)$ time and space to determine the length d of an LCS between S and T , and the $\mathcal{O}(\log k)$ values ℓ that need to be tested so that the intervals $(\ell/2, \ell]$ cover $[d, (k+1)d + k]$, this concludes the proof of Theorem 2.

Moreover, the three cases yield the following complexities: $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} n)$ if $\ell > \log^{3/2} N$, $\mathcal{O}(n \log^{k-1/2} n) = \mathcal{O}(\frac{n}{\ell} \log^{k+1} n)$ if $\log N < \ell \leq \log^{3/2} N$, and $\mathcal{O}(n \log^{k-1} n) = \mathcal{O}(\frac{n}{\ell} \log^{k+1} n)$ if $\ell \leq \log N$, which gives an $\mathcal{O}(n + \frac{n}{\ell} \log^{k+1} n)$ -time solution for any ℓ , thus improving [24] for $k = \mathcal{O}(1)$.

References

- 1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *26th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.
- 2 Amihod Amir and Itai Boneh. Locally maximal common factors as a tool for efficient dynamic string algorithms. In Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 11:1–11:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.11.
- 3 Amihod Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/s00453-020-00744-0.
- 4 Lorraine A. K. Ayad, Carl Barton, Panagiotis Charalampopoulos, Costas S. Iliopoulos, and Solon P. Pissis. Longest common prefixes with k-errors and applications. In Travis Gagie, Alistair Moffat, Gonzalo Navarro, and Ernesto Cuadros-Vargas, editors, *String Processing and Information Retrieval - 25th International Symposium, SPIRE 2018, Lima, Peru, October 9-11, 2018, Proceedings*, volume 11147 of *Lecture Notes in Computer Science*, pages 27–41. Springer, 2018. doi:10.1007/978-3-030-00479-8_3.
- 5 Maxim A. Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. 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 572–591. SIAM, 2015. doi:10.1137/1.9781611973730.39.
- 6 Maxim A. 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 Ricardo A. Baeza-Yates. Improved string searching. *Software: Practice and Experience*, 19(3):257–271, 1989. doi:10.1002/spe.4380190305.
- 8 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM Journal on Computing*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.
- 9 Djamel Belazzougui. Worst case efficient single and multiple string matching in the RAM model. In Costas S. Iliopoulos and William F. Smyth, editors, *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers*, volume 6460 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2010. doi:10.1007/978-3-642-19222-7_10.
- 10 Djamel Belazzougui. Improved space-time tradeoffs for approximate full-text indexing with one edit error. *Algorithmica*, 72(3):791–817, 2015. doi:10.1007/s00453-014-9873-9.
- 11 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Optimal packed string matching. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPICs*, pages 423–432. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPICs.FSTTCS.2011.423.
- 12 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Towards optimal packed string matching. *Theoretical Computer Science*, 525:111–129, 2014. doi:10.1016/j.tcs.2013.06.013.
- 13 Stav Ben-Nun, Shay Golan, Tomasz Kociumaka, and Matan Kraus. Time-space tradeoffs for finding a long common substring. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPICs*, pages 5:1–5:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CPM.2020.5.
- 14 Philip Bille. Fast searching in packed strings. In Gregory Kucherov and Esko Ukkonen, editors, *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009, Lille, France, June 22-24, 2009, Proceedings*, volume 5577 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2009. doi:10.1007/978-3-642-02441-2_11.
- 15 Philip Bille. Fast searching in packed strings. *Journal of Discrete Algorithms*, 9(1):49–56, 2011. doi:10.1016/j.jda.2010.09.003.
- 16 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017, July 4-6, 2017, Warsaw, Poland*, volume 78 of *LIPICs*, pages 6:1–6:11. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.CPM.2017.6.
- 17 Dany Breslauer, Leszek Gasieniec, and Roberto Grossi. Constant-time word-size string matching. In Juha Kärkkäinen and Jens Stoye, editors, *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings*, volume 7354 of *Lecture Notes in Computer Science*, pages 83–96. Springer, 2012. doi:10.1007/978-3-642-31265-6_7.
- 18 Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69.
- 19 Gerth Stølting Brodal and Christian N. S. Pedersen. Finding maximal quasiperiodicities in strings. In Raffaele Giancarlo and David Sankoff, editors, *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2000. doi:10.1007/3-540-45123-4_33.

- 20 Mark R. Brown and Robert Endre Tarjan. A fast merging algorithm. *Journal of the ACM*, 26(2):211–226, 1979. doi:10.1145/322123.322127.
- 21 Domenico Cantone and Simone Faro. Pattern matching with swaps for short patterns in linear time. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings*, volume 5404 of *Lecture Notes in Computer Science*, pages 255–266. Springer, 2009. doi:10.1007/978-3-540-95891-8_25.
- 22 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. Compressed indexes for approximate string matching. *Algorithmica*, 58(2):263–281, 2010. doi:10.1007/s00453-008-9263-2.
- 23 Ho-Leung Chan, Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, and Swee-Seong Wong. A linear size index for approximate pattern matching. *Journal of Discrete Algorithms*, 9(4):358–364, 2011. doi:10.1016/j.jda.2011.04.004.
- 24 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 Gonzalo Navarro, David Sankoff, and Binhai Zhu, editors, *Annual Symposium on Combinatorial Pattern Matching, CPM 2018, July 2-4, 2018 - Qingdao, China*, volume 105 of *LIPICs*, pages 23:1–23:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CPM.2018.23.
- 25 Panagiotis Charalampopoulos, Paweł Gawrychowski, and Karol Pokorski. Dynamic longest common substring in polylogarithmic time. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPICs*, pages 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.ICALP.2020.27.
- 26 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Waleń, and Wiktor Zuba. Circular pattern matching with k mismatches. *Journal of Computer and System Sciences*, 115:73–85, 2021. doi:10.1016/j.jcss.2020.07.003.
- 27 Archie L. Cobbs. Fast approximate matching using suffix trees. In Zvi Galil and Esko Ukkonen, editors, *Combinatorial Pattern Matching, 6th Annual Symposium, CPM 95, Espoo, Finland, July 5-7, 1995, Proceedings*, volume 937 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 1995. doi:10.1007/3-540-60044-2_33.
- 28 Vincent Cohen-Addad, Laurent Feuilloley, and Tatiana Starikovskaya. Lower bounds for text indexing with mismatches and differences. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1146–1164. SIAM, 2019. doi:10.1137/1.9781611975482.70.
- 29 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.
- 30 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.
- 31 Maxime Crochemore, Costas S. Iliopoulos, Manal Mohamed, and Marie-France Sagot. Longest repeats with a block of k don't cares. *Theoretical Computer Science*, 362(1-3):248–254, 2006. doi:10.1016/j.tcs.2006.06.029.
- 32 Jonas Ellert and Johannes Fischer. Linear time runs over general ordered alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPICs*, pages 63:1–63:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.ICALP.2021.63.

- 33 Martin Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646102.
- 34 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. URL: <http://www.jstor.org/stable/2034009>.
- 35 Tomás Flouri, Emanuele Giaquinta, Kastian Kobert, and Esko Ukkonen. Longest common substrings with k mismatches. *Information Processing Letters*, 115(6-8):643–647, 2015. doi:10.1016/j.ipl.2015.03.006.
- 36 Kimmo Fredriksson. Faster string matching with super-alphabets. In Alberto H. F. Laender and Arlindo L. Oliveira, editors, *String Processing and Information Retrieval, 9th International Symposium, SPIRE 2002, Lisbon, Portugal, September 11-13, 2002, Proceedings*, volume 2476 of *Lecture Notes in Computer Science*, pages 44–57. Springer, 2002. doi:10.1007/3-540-45735-6_5.
- 37 Kimmo Fredriksson. Shift-or string matching with super-alphabets. *Information Processing Letters*, 87(4):201–204, 2003. doi:10.1016/S0020-0190(03)00296-5.
- 38 François Le Gall and Saeed Seddighin. Quantum meets fine-grained complexity: Sublinear time quantum algorithms for string problems. *CoRR*, abs/2010.12122, 2020. arXiv:2010.12122.
- 39 Emanuele Giaquinta, Szymon Grabowski, and Kimmo Fredriksson. Approximate pattern matching with k -mismatches in packed text. *Information Processing Letters*, 113(19-21):693–697, 2013. doi:10.1016/j.ipl.2013.07.002.
- 40 Szymon Grabowski. A note on the longest common substring with k -mismatches problem. *Information Processing Letters*, 115(6-8):640–642, 2015. doi:10.1016/j.ipl.2015.03.003.
- 41 Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 841–850. ACM/SIAM, 2003. URL: <http://dl.acm.org/citation.cfm?id=644108.644250>.
- 42 Yijie Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *Journal of Algorithms*, 50(1):96–105, 2004. doi:10.1016/j.jalgor.2003.09.001.
- 43 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.
- 44 Trinh N. D. Huynh, Wing-Kai Hon, Tak Wah Lam, and Wing-Kin Sung. Approximate string matching using compressed suffix arrays. *Theoretical Computer Science*, 352(1-3):240–249, 2006. doi:10.1016/j.tcs.2005.11.022.
- 45 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
- 46 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.
- 47 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.
- 48 Shmuel Tomi Klein and Miri Ben-Nissan. Accelerating Boyer Moore searches on binary texts. In Jan Holub and Jan Zdárek, editors, *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers*, volume 4783 of *Lecture Notes in Computer Science*, pages 130–143. Springer, 2007. doi:10.1007/978-3-540-76336-9_14.

- 49 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.
- 50 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 605–617, 2014. doi:10.1007/978-3-662-44777-2_50.
- 51 Roman M. Kolpakov and Gregory Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 596–604. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814634.
- 52 Gad M. Landau and Uzi Vishkin. Fast string matching with k differences. *Journal of Computer and System Sciences*, 37(1):63–78, 1988. doi:10.1016/0022-0000(88)90045-1.
- 53 J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, volume 161 of *LIPICs*, pages 24:1–24:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CPM.2020.24.
- 54 J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theoretical Computer Science*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.
- 55 Gonzalo Navarro and Mathieu Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In Martin Farach-Colton, editor, *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 20-22, 1998, Proceedings*, volume 1448 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1998. doi:10.1007/BFb0030778.
- 56 Tatiana 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.
- 57 Jorma Tarhio and Hannu Peltola. String matching in the DNA alphabet. *Software: Practice and Experience*, 27(7):851–861, 1997.
- 58 Sharma V. Thankachan, Chaitanya Aluru, Sriram P. Chockalingam, and Srinivas Aluru. Algorithmic framework for approximate matching under bounded edits with applications to sequence analysis. In Benjamin J. Raphael, editor, *Research in Computational Molecular Biology - 22nd Annual International Conference, RECOMB 2018, Paris, France, April 21-24, 2018, Proceedings*, volume 10812 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2018. doi:10.1007/978-3-319-89929-9_14.
- 59 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.
- 60 Dekel Tsur. Fast index for approximate string matching. *Journal of Discrete Algorithms*, 8(4):339–345, 2010. doi:10.1016/j.jda.2010.08.002.
- 61 Esko Ukkonen. Approximate string-matching over suffix trees. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *Combinatorial Pattern Matching, 4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993, Proceedings*, volume 684 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 1993. doi:10.1007/BFb0029808.
- 62 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11. IEEE Computer Society, 1973. doi:10.1109/SWAT.1973.13.