Time-Space Tradeoffs for Finding a Long Common Substring

Stav Ben-Nun

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
stav.bennun@live.biu.ac.il

Shay Golan

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
golansh1@cs.biu.ac.il

Tomasz Kociumaka

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
kociumaka@mimuw.edu.pl

Matan Kraus

Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
krausma@biu.ac.il

Abstract

We consider the problem of finding, given two documents of total length n, a longest string occurring as a substring of both documents. This problem, known as the Longest Common Substring (LCS) problem, has a classic $\mathcal{O}(n)$-time solution dating back to the discovery of suffix trees (Weiner, 1973) and their efficient construction for integer alphabets (Farach-Colton, 1997). However, these solutions require $\Theta(n)$ space, which is prohibitive in many applications. To address this issue, Starikovskaya and Vildhøj (CPM 2013) showed that for $n^{2/3} \le s \le n$, the LCS problem can be solved in $\mathcal{O}(s)$ space and $\tilde{\mathcal{O}}(\frac{n^2}{s})$ time.[1] Kociumaka et al. (ESA 2014) generalized this tradeoff to $1 \le s \le n$, thus providing a smooth time-space tradeoff from constant to linear space. In this paper, we obtain a significant speed-up for instances where the length L of the sought LCS is large. For $1 \le s \le n$, we show that the LCS problem can be solved in $\mathcal{O}(s)$ space and $\tilde{\mathcal{O}}(\frac{n^2}{L\cdot s} + n)$ time. The result is based on techniques originating from the LCS with Mismatches problem (Flouri et al., 2015; Charalampopoulos et al., CPM 2018), on space-efficient locally consistent parsing (Birenzwige et al., SODA 2020), and on the structure of maximal repetitions (runs) in the input documents.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases longest common substring, time-space tradeoff, local consistency, periodicity

Digital Object Identifier 10.4230/LIPIcs.CPM.2020.5

Funding Supported by ISF grants no. 1278/16 and 1926/19, a BSF grant no. 2018364, and an ERC grant MPM (no. 683064) under the EU's Horizon 2020 Research and Innovation Programme.

1 Introduction

The Longest Common Substring (LCS) problem is a fundamental text processing problem with numerous applications; see e.g. [1, 39, 22]. Given *two* strings (documents) S_1, S_2, the LCS problem asks for a longest string occurring in S_1 and S_2. We denote the length of the longest common substring by $\text{lcs}(S_1, S_2)$.

The classic text-book solution to the LCS problem is to build the (generalized) suffix tree of the documents and find the node that corresponds to an LCS [40, 26, 17]. While this can be achieved in linear time, it comes at the cost of using $\Omega(n)$ words (of $\Theta(\log n)$

[1] The $\tilde{\mathcal{O}}$ notation hides $\log^{\mathcal{O}(1)} n$ factors.

bits each) to store the suffix tree. In applications with large amounts of data or strict space constraints, this renders the classic solution impractical. To overcome the space challenge of suffix trees, succinct and compressed data structures have been subject to extensive research [25, 35]. Nevertheless, these data structures still use $\Omega(n)$ bits of space in the worst-case. Starikovskaya and Vildhøj [36] showed that for $n^{2/3} \leq s \leq n$, the LCS problem can be solved in $\mathcal{O}(\frac{n^2}{s} + s\log n)$ time using $\mathcal{O}(s)$ space. Kociumaka et al. [31] subsequently improved the running time to $\mathcal{O}(\frac{n^2}{s})$ and extended the parameter range to $1 \leq s \leq n$.

These previous works also considered a generalized version of the LCS problem, where the input consists of m documents $S_1, S_2, \ldots, S_m$ (still of total length n) and an integer $2 \leq d \leq m$. The task there is to compute a longest string occurring as a substring of at least d of the m input documents. In this setting, Starikovskaya and Vildhøj [36] achieve $\mathcal{O}(\frac{n^2\log^2 n}{s}(d\log^2 n + d^2))$ time and $\mathcal{O}(s)$ space for $n^{2/3} \leq s \leq n$, whereas Kociumaka et al. [31] showed a solution which takes $\mathcal{O}(\frac{n^2}{s})$ time and $\mathcal{O}(s)$ space for $1 \leq s \leq n$. The cost of this algorithm matches both a classic $\Theta(n)$-space algorithm [27] and the time-space tradeoff for $d = m = 2$. Nevertheless, in this paper we focus on the LCS problem for *two* strings only.

Kociumaka et al. [31] additionally provided a lower bound which states that any deterministic algorithm using $s \leq \frac{n}{\log n}$ space must cost $\Omega(n\sqrt{\log(n/(s\log n))/\log\log(n/(s\log n))})$ time. This lower bound is actually derived for the problem of distinguishing whether $\mathrm{lcs}(S_1, S_2) = 0$, i.e., deciding if the two input strings have any character in common. This state of affairs naturally leads to a question of whether distinguishing between $\mathrm{lcs}(S_1, S_2) < \ell$ and $\mathrm{lcs}(S_1, S_2) \geq \ell$ gets easier as ℓ increases, or equivalently, whether $L := \mathrm{lcs}(S_1, S_2)$ can be computed more efficiently when L is large. This case is relevant for applications since the existence of short common substrings is less meaningful for measuring string similarity.

1.1 Our Results

We provide new sublinear-space algorithms for the LCS problem optimized for inputs with a long common substring. The algorithms are designed for the word-RAM model with word size $w = \Theta(\log n)$, and they work for integer alphabets $\Sigma = \{1, 2, \ldots, n^{\mathcal{O}(1)}\}$. Throughout the paper, the input strings reside in a read-only memory and any space used by the algorithms is a working space; furthermore, we represent the output by witness occurrences in the input strings so that it fits in $\mathcal{O}(1)$ machine words. Our main result is as follows:

▶ **Theorem 1.** *Given s with $1 \leq s \leq n$, the LCS problem with $L = \mathrm{lcs}(S_1, S_2)$ can be solved deterministically in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2\log n\log^* n}{s\cdot L} + n\log n)$ time,*[2] *and in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2\log n}{s\cdot L} + n\log n)$ time with high probability using a Las-Vegas randomized algorithm.*

We remark that Theorem 1 improves upon the result of Kociumaka et al. [31] whenever $s < \frac{n}{\log n}$ and $L > \log n\log^* n$ (or $L > \log n$ if randomization is allowed).

We also show that the log factors can be removed from the running times in Theorem 1 if $s = \Theta(1)$. In fact, this yields an improvement upon Theorem 1 as long as $s < \log n\log^* n$.

▶ **Theorem 2.** *The LCS problem can be solved deterministically in $\mathcal{O}(1)$ space and $\mathcal{O}(\frac{n^2}{L})$ time, where $L = \mathrm{lcs}(S_1, S_2)$.*

As a step towards our main result, we solve the LCS_ℓ problem defined below.

[2] The *iterated logarithm* function $\log^*$ is formally defined with $\log^* x = 0$ for $x \leq 1$ and $\log^* x = 1 + \log^*(\log x)$ for $x > 1$. In other words $\log^* n$ is the smallest integer $k \geq 0$ such that $\log^{(k)} x \leq 1$.

LONGEST COMMON SUBSTRING WITH THRESHOLD ℓ (LCS_ℓ)

Input: Two strings S_1 and S_2 (of length at most n), an integer threshold $\ell \in [n]$

Output: A common substring G of S_1 and S_2 such that:

1. if $\ell \leq \mathrm{lcs}(S_1, S_2) < 2\ell$, then $|G| = \mathrm{lcs}(S_1, S_2)$;
2. if $\mathrm{lcs}(S_1, S_2) \geq 2\ell$, then $|G| \geq 2\ell$.

If $\mathrm{lcs}(S_1, S_2) < \ell$, then LCS_ℓ allows for an arbitrary common substring in the output.

▶ Remark 3. Note the following equivalent characterization of the output G of LCS_ℓ: for every common substring T with $\ell \leq |T| \leq 2\ell$, the common substring G is of length $|G| \geq |T|$.

▶ **Theorem 4.** *The LCS_ℓ problem can be solved deterministically in $\mathcal{O}(\frac{n \log^* n}{\ell})$ space and $\mathcal{O}(n \log n)$ time, and in $\mathcal{O}(\frac{n}{\ell})$ space and $\mathcal{O}(n \log n)$ time with high probability using a Las-Vegas randomized algorithm.*

1.2 Related work

The LCS problem has been studied in many other settings. Babenko and Starikovskaya [6], Flouri et al. [20], Thankachan et al. [38], and Kociumaka et al. [30] studied the LCS with k mismatches problem, where the occurrences in S_1 and S_2 can be at Hamming distance up to k. Charalampopoulos et al. [11] showed that this problem becomes easier when the strings have a long common substring with k mismatches (similarly to what we obtain for LCS with no mismatches). Thankachan et al. [37] and Ayad et al. [5] considered a related problem with edit distance instead of the Hamming distance. Alzamel et al. [2] proposed an $\tilde{\mathcal{O}}(n)$-time algorithm for the Longest Common Circular Substring problem, where occurrences in S_1 and S_2 can be cyclic rotations of each other.

Amir et al. [3] studied the problem of answering queries asking for the LCS after a single edit in either of the two original input strings. Subsequently, Amir et al. [4] and Charalampopoulos et al. [12] considered a fully dynamic version of the problem, in which the edit operations are applied sequentially, ultimately achieving $\tilde{\mathcal{O}}(1)$ time per operation.

1.3 Algorithmic Overview

We first give an overview of the algorithm of Theorem 4. Then, we derive Theorem 1 in two steps, with an $\mathcal{O}(s)$-space solution to the LCS_ℓ problem as an intermediate result.

An $\tilde{\mathcal{O}}(n/\ell)$-space algorithm for the LCS_ℓ problem. In Section 3, we define an *anchored* variant of the LONGEST COMMON SUBSTRING problem (LCAS). In the LCAS problem, we are given two strings S_1, S_2 and sets of positions A_1 and A_2, and we wish to find a longest common substring which can be obtained by extending (to the left and to the right) $S_1[p_1]$ and $S_2[p_2]$ for some $(p_1, p_2) \in A_1 \times A_2$. We then reduce the LCAS problem to the TWO STRING FAMILIES LCP problem, introduced by Charalampopoulos et al. [11] in the context of finding LCS with mismatches.

In Section 4, we show how to solve the LCS_ℓ problem by selecting positions in A_1 and A_2 so that *every* common substring T of S_1 and S_2 with $|T| \geq \ell$ can be obtained by extending $S_1[p_1]$ and $S_2[p_2]$ for some $(p_1, p_2) \in A_1 \times A_2$. To make this selection, we use *partitioning sets* by Birenzwige et al. [9], which consist of $\tilde{\mathcal{O}}(\frac{n}{\ell})$ positions chosen in a locally consistent manner. However, since partitioning sets do not select positions in long periodic regions, our algorithms use *maximal repetitions* (runs) [32, 7] and their *Lyndon roots* [13] to add $\mathcal{O}(\frac{n}{\ell})$ extra positions. Overall, we get an $\tilde{\mathcal{O}}(\frac{n}{\ell})$-space and $\tilde{\mathcal{O}}(n)$-time algorithm for the LCS_ℓ problem.

An $\mathcal{O}(s)$ space algorithm for the LCS_ℓ problem. In Section 5, we give a time-space tradeoff for the LCS_ℓ problem. The algorithm partitions the input strings into overlapping substrings, executes the algorithm of Section 4 for each pair of substrings, and returns the longest among the common substrings obtained from these calls. For a tradeoff parameter $1 \le s \le n$, the algorithm takes $\mathcal{O}(s)$ space and $\tilde{\mathcal{O}}(\frac{n^2}{s\cdot\ell} + n)$ time.

A solution to the LCS problem. In Section 6, we show how to search for LCS by repeatedly solving the LCS_ℓ problem with different choices of ℓ. We get an algorithm that takes $\mathcal{O}(s)$ space and $\tilde{\mathcal{O}}(\frac{n^2}{s\cdot L} + n)$ time, where $L = \operatorname{lcs}(S_1, S_2)$, as stated in Theorem 1.

2 Preliminaries

For $1 \le i < j \le n$, denote the integer *intervals* $[i\,..\,j] = \{i, i+1, \dots, j\}$ and $[k] = \{1, 2, \dots, k\}$.

A string S of length $n = |S|$ is a finite sequence of characters $S[1]S[2]\cdots S[n]$ over an alphabet Σ; in this paper, we consider polynomially-bounded integer alphabets $\Sigma = [1\,..\,n^{\mathcal{O}(1)}]$. The string $S^r = S[n]S[n-1]\cdots S[1]$ is called the *reverse* of the string S.

A string T is a *substring* of a string S if $T = S[x]S[x+1]\cdots S[y]$ for some $1 \le x \le y \le |S|$. We then say that T *occurs* in S at position x, and we denote the *occurrence* by $S[x\,..\,y]$. We call $S[x\,..\,y]$ a *fragment* of S. A fragment $S[x\,..\,y]$ is a *prefix* of S if $x = 1$ and a *suffix* of S if $y = |S|$. These special fragments are also denoted by $S[..\,y]$ and $S[x\,..]$, respectively. A *proper fragment* of S is any fragment other than $S[1\,..\,|S|]$. A common prefix (suffix) of two strings S_1, S_2 is a string that occurs as a prefix (resp. suffix) of both S_1 and S_2. The longest common prefix of S_1 and S_2 is denoted by $\mathrm{LCP}(S_1, S_2)$, and the longest common suffix is denoted by $\mathrm{LCP}^r(S_1, S_2)$. Note that $\mathrm{LCP}^r(S_1, S_2) = (LCP(S_1^r, S_2^r))^r$.

An integer $k \in [|S|]$, is a *period* of a string S if $S[i] = S[i+k]$ for $i \in [|S| - k]$. The shortest period of S is denoted by $\operatorname{per}(S)$. If $\operatorname{per}(S) \le \frac{1}{2}|S|$, we say that S is *periodic*. A periodic fragment $S[i\,..\,j]$ is called a *run* [32, 7] if it cannot be extended (to the left nor to the right) without increasing the shortest period. For a pair of parameters d and ρ, we say that a run $S[i\,..\,j]$ is a (d, ρ)*-run* if $|S[i\,..\,j]| \ge d$ and $\operatorname{per}(S[i\,..\,j]) \le \rho$. Note that every periodic fragment $S[i'\,..\,j']$ with $|S[i'\,..\,j']| \ge d$ and $\operatorname{per}(S[i'\,..\,j']) \le \rho$ can be uniquely extended to a (d, ρ)-run $S[i\,..\,j]$ while preserving the shortest period $\operatorname{per}(S[i\,..\,j]) = \operatorname{per}(S[i'\,..\,j'])$.

Tries and suffix trees. Given a set of strings $\mathcal{F}$, the compact trie [34] of these strings is the tree obtained by compressing each path of nodes of degree one in the trie [10, 21] of the strings in $\mathcal{F}$, which takes $\mathcal{O}(|\mathcal{F}|)$ space. Each edge in the compact trie has a label represented as a fragment of a string in $\mathcal{F}$. The suffix tree [40] of a string S is the compact trie of all the suffixes of S. The sparse suffix tree [29, 8, 28, 24] of a string S is the compact trie of selected suffixes $\{S[i\,..] : i \in B\}$ specified by a set of positions $B \subseteq [|S|]$.

3 Longest Common Anchored Substring problem

In this section, we consider an *anchored* variant of the LONGEST COMMON SUBSTRING problem. Let A_1 and A_2 be sets of distinguished positions, called *anchors*, in strings S_1 and S_2, respectively. We say that a string T is a *common anchored substring* of S_1 and S_2 with respect to A_1 and A_2 if it has occurrences $S_1[i_1\,..\,j_1] = T = S_2[i_2\,..\,j_2]$ with a *synchronized pair of anchors*, i.e., with some anchors $p_1 \in A_1$ and $p_2 \in A_2$ such that $p_1 - i_1 = p_2 - i_2 \in [0, |T|]$.[3]

[3] Note that the anchors could be at positions $p_1 = j_1 + 1$ and $p_2 = j_2 + 1$ (if $p_1 - i_1 = p_2 - i_2 = |T|$).

Longest Common Anchored Substring (LCAS)
Input: Two strings S_1, S_2 (of length at most n) and two sets of anchors $A_1 \subseteq [|S_1|]$, $A_2 \subseteq [|S_2|]$.
Output: A longest common anchored substring of S_1 and S_2 with respect to A_1, A_2.

We utilize the following characterization of the longest common anchored substring.

▶ **Fact 5.** *The length of a longest common anchored substring of two strings S_1 and S_2 (with respect to anchors at A_1 and A_2) is*

$$\max\{|\mathrm{LCP}(S_1[p_1\,..], S_2[p_2\,..])| + |\mathrm{LCP}^r(S_1[..\,p_1-1], S_2[..\,p_2-1])| : p_1 \in A_1, p_2 \in A_2\}.$$

Moreover, for any $(p_1, p_2) \in A_1 \times A_2$ maximizing this expression, a longest common anchored substring is the concatenation $\mathrm{LCP}^r(S_1[..\,p_1-1], S_2[..\,p_2-1]) \cdot \mathrm{LCP}(S_1[p_1\,..], S_2[p_2\,..])$.

The characterization of Fact 5 lets us use the following problem, originally defined in a context of computing LCS with mismatches.

Two String Families LCP (Charalampopoulos et al. [11])
Input: A compact trie $\mathcal{T}(\mathcal{F})$ of a family of strings $\mathcal{F}$ and two sets $\mathsf{P}, \mathsf{Q} \subseteq \mathcal{F}^2$.
Output: The value $\mathrm{maxPairLCP}(\mathsf{P}, \mathsf{Q})$, defined as

$$\mathrm{maxPairLCP}(\mathsf{P}, \mathsf{Q}) = \max\{|\mathrm{LCP}(P_1, Q_1)| + |\mathrm{LCP}(P_2, Q_2)| : (P_1, P_2) \in \mathsf{P},\ (Q_1, Q_2) \in \mathsf{Q}\},$$

along with pairs $(P_1, P_2) \in \mathsf{P}$ and $(Q_1, Q_2) \in \mathsf{Q}$ for which the maximum is attained.

Charalampopoulos et al. [11] observed that the Two String Families LCP problem can be solved using an approach by Crochemore et al. [14] and Flouri et al. [20].

▶ **Lemma 6** ([11, Lemma 3]). *The Two String Families LCP problem can be solved in $\mathcal{O}(|\mathcal{F}| + N \log N)$ time using $\mathcal{O}(|\mathcal{F}| + N)$ space*[4]*, where $N = |\mathsf{P}| + |\mathsf{Q}|$.*

By Fact 5, the LCAS problem reduces to the Two String Families LCP problem with:

$$\begin{aligned} \mathcal{F} &= \{S_1[p\,..] : p \in A_1\} \cup \{(S_1[..\,p-1])^r : p \in A_1\} \\ &\qquad \cup \{S_2[p\,..] : p \in A_2\} \cup \{(S_2[..\,p-1])^r : p \in A_2\}, && (1) \\ \mathsf{P} &= \{(S_1[p\,..], (S_1[..\,p-1])^r) : p \in A_1\}, && (2) \\ \mathsf{Q} &= \{(S_2[p\,..], (S_2[..\,p-1])^r) : p \in A_2\}. && (3) \end{aligned}$$

The following theorem provides an efficient implementation of this reduction. The most challenging step, to construct the compacted trie $\mathcal{T}(\mathcal{F})$, is delegated to the work of Birenzwige et al. [9], who show that a sparse suffix tree of a length-n string S with $B \subseteq [n]$ can be constructed deterministically in $\mathcal{O}(n \log \frac{n}{|B|})$ time and $\mathcal{O}(|B| + \log n)$ space.

▶ **Theorem 7.** *The Longest Common Anchored Substring problem can be solved in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(|A_1| + |A_2| + \log n)$ space.*

[4] The original formulation of [11, Lemma 3] does not discuss the space complexity. However, an inspection of the underlying algorithm, described in [14, 20], easily yields this additional claim.

Proof. We implicitly create a string $S = S_1\$S_1^r\$S_2\$S_2^r$ and construct a sparse suffix tree of S containing the following suffixes: $S_1[p\,..]\$S_1^r\$S_2\$S_2^r$ and $(S_1[..\,p-1])^r\$S_2\$S_2^r$ for $p \in A_1$, as well as $S_2[p\,..]\$S_2^r$ and $(S_2[..\,p-1])^r$ for $p \in A_2$. We then trim this tree, cutting edges immediately above any \$ on their labels, which results in the compacted trie $\mathcal{T}(\mathcal{F})$ for the family $\mathcal{F}$ defined in (1). We then build P and Q according to (2) and (3), respectively, and solve an instance of the Two String Families LCP problem specified by $\mathcal{T}(\mathcal{F}), \mathsf{P}, \mathsf{Q}$. This yields pairs in P and Q for which $\mathrm{maxPairLCP}(\mathsf{P}, \mathsf{Q})$ is attained. We retrieve the underlying indices $p_1 \in A_1$ and $p_2 \in A_2$ and derive a longest common anchored substring of S_1 and S_2 according to Fact 5: $\mathrm{LCP}^r(S_1[..\,p_1-1], S_2[..\,p_2-1]) \cdot \mathrm{LCP}(S_1[p_1\,..], S_2[p_2\,..])$.

With the sparse suffix tree construction of [9] and the algorithm of Lemma 6 that solves the Two String Families LCP problem, the overall running time is $\mathcal{O}(n \log \frac{n}{N} + N \log N) = \mathcal{O}(n \log n)$ and the space complexity is $\mathcal{O}(N + \log n)$, where $N = |A_1| + |A_2|$. ◀

4 Space-efficient $\tilde{\mathcal{O}}(n)$-time algorithm for the LCS_ℓ problem

Our approach to solve the LCS_ℓ problem is via a reduction to the LCAS problem. For this, we wish to select anchors $A_1 \subseteq [|S_1|]$ and $A_2 \subseteq [|S_2|]$ so that every common substring T of length at least ℓ is a common anchored substring. In other words, we need to make sure that T admits occurrences $S_1[i_1\,..\,j_1] = T = S_2[i_2\,..\,j_2]$ with a synchronized pair of anchors.

As a warm-up, we describe a simple selection of $\mathcal{O}(n/\sqrt{\ell})$ anchors based on *difference covers* [33], which have already been used by Starikovskaya and Vildhøj [36] in a time-space tradeoff for the LCS problem. For every two integers τ, m with $1 \le \tau \le m$, this technique yields a set $DC_\tau(m) \subseteq [m]$ of size $\mathcal{O}(m/\sqrt{\tau})$ such that for every two indices $i_1, i_2 \in [m-\tau+1]$, there is a shift $\Delta \in [0\,..\,\tau-1]$ such that $i_1 + \Delta$ and $i_2 + \Delta$ both belong to $DC_\tau(m)$. Hence, to make sure that every common substring of length at least ℓ is anchored, it suffices to select all $\mathcal{O}(n/\sqrt{\ell})$ positions in $DC_\ell(n)$ as anchors: $A_1 = [|S_1|] \cap DC_\ell(n)$ and $A_2 = [|S_2|] \cap DC_\ell(n)$.

We remark that such selection of anchors is *non-adaptive*: it does not depend on contents of the strings S_1 and S_2, but only on the lengths of these strings (and the parameter ℓ). In fact, any non-adaptive construction needs $\Omega(n/\sqrt{\ell})$ anchors in order to guarantee that every common substring T of length at least ℓ is a common anchored substring. In the following, we show how adaptivity allows us to achieve the same goal using only $\tilde{\mathcal{O}}(n/\ell)$ anchors.

4.1 Selection of Anchors: the non-periodic case

We first show how to accommodate common substrings T of length $|T| \ge \ell$ that do not contain a $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run. The idea is to use *partitioning sets* by Birenzwige et al. [9].

▶ **Definition 8** (Birenzwige et al. [9]). *A set of positions $P \subseteq [n]$ is called a (τ, δ)-*partitioning set *of a length-n string S, for some parameters $\tau, \delta \in [n]$, if it has the following properties:*

Local Consistency: *For every two indices $i, j \in [1+\delta\,..\,n-\delta]$ such that $S[i-\delta\,..\,i+\delta] = S[j-\delta\,..\,j+\delta]$, we have $i \in P$ if and only if $j \in P$.*

Compactness: *If $p_i < p_{i+1}$ are two consecutive positions in $P \cup \{1, n+1\}$ such that $p_{i+1} > p_i + \tau$, then $u = S[p_i\,..\,p_{i+1}-1]$ is periodic with period* $\mathrm{per}(u) \le \tau$.

Note that any (τ, δ)-partitioning set is also a (τ', δ')-partitioning set for any $\tau' \ge \tau$ and $\delta' \ge \delta$. The selection of anchors is based on an arbitrary $(\frac{1}{5}\ell, \frac{1}{5}\ell)$-partitioning set P of the string S_1S_2: for every position $p \in P$, p is added to A_1 (if $p \le |S_1|$) or $p - |S_1|$ is added to A_2 (otherwise). Below, we show that this selection satisfies the advertised property.

▶ **Lemma 9.** *Let T be a common substring of length $|T| \ge \ell$ which does not contain a $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run. Then, T is a common anchored substring with respect to A_1, A_2 defined above.*

Proof. Let $S_1[i_1\,..\,j_1]$ and $S_2[i_2\,..\,j_2]$ be arbitrary occurrences of T in S_1 and S_2, respectively. If there is a position $p_1 \in A_1$ with $p_1 \in [i_1+\delta\,..\,j_1-\delta]$, then the position $p_2 = i_2 + (p_1 - i_1)$ belongs to A_2 by the local consistency property of the underlying partitioning set, due to $S_1[p_1-\delta\,..\,p_1+\delta] = S_2[p_2-\delta\,..\,p_2+\delta]$. Hence, (p_1,p_2) is a synchronized pair of anchors and T is a common anchored substring with respect to A_1, A_2.

If there is no such position $p_1 \in A_1$, then $S_1[i_1+\delta\,..\,j_1-\delta]$ is contained within a block between two consecutive positions of the partitioning set. The length of this block is at least $|T| - 2\delta \geq \frac{3}{5}\ell > \tau$, so the block is periodic by the compactness property of the partitioning set. Hence, $\operatorname{per}(T[1+\delta\,..\,|T|-\delta]) = \operatorname{per}(S_1[i_1+\delta\,..\,j_1-\delta]) \leq \tau \leq \frac{1}{5}\ell$. A $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run in T can thus be obtained by maximally extending $T[1+\delta\,..\,|T|-\delta]$ without increasing the shortest period. Such a run in T is a contradiction to the assumption. ◀

Birenzwige et al. [9] gave a deterministic algorithm that constructs a $(\tau, \tau\log^* n)$-partitioning set of size $\mathcal{O}(\frac{n}{\tau})$ in $\mathcal{O}(n\log\tau)$ time using $\mathcal{O}(\frac{n}{\tau}+\log\tau)$ space. Setting appropriate $\tau = \Theta(\ell/\log^* n)$, we get an $(\frac{1}{5}\ell, \frac{1}{5}\ell)$-partitioning set of size $\mathcal{O}(\frac{n\log^* n}{\ell})$.

Furthermore, Birenzwige et al. [9] gave a Las-Vegas randomized algorithm that constructs a (τ,τ)-partitioning set of size $\mathcal{O}(\frac{n}{\tau})$ in $\mathcal{O}(n+\tau\log^2 n)$ time with high probability, using $\mathcal{O}(\frac{n}{\tau}+\log n)$ space. Setting $\tau = \frac{1}{5}\ell$, we get an $(\frac{1}{5}\ell, \frac{1}{5}\ell)$-partitioning set of size $\mathcal{O}(\frac{n}{\ell})$.

4.2 Selection of anchors: the periodic case

In this section, for any parameters $d, \rho \in [n]$ satisfying $d \geq 3\rho - 1$, we show how to accommodate all common substrings containing a (d,ρ)-run by selecting $\mathcal{O}(\frac{n}{d})$ anchors. This method is then used for $d = \frac{3}{5}\ell$ and $\rho = \frac{1}{5}\ell$ to complement the selection in Section 4.1.

Let T be a common substring of S_1 and S_2 containing a (d,ρ)-run. We consider two cases depending on whether the run is a proper fragment of T or the whole T. In the first case, it suffices to select as anchors the first and the last position of every (d,ρ)-run.

▶ **Lemma 10.** *Let A_1 and A_2 contain the boundary positions of every (d,ρ)-run in S_1 and S_2, respectively. If T is a common substring of S_1 and S_2 with a (d,ρ)-run r as a proper fragment, then T is a common anchored substring of S_1 and S_2 with respect to A_1, A_2.*

Proof. In the proof, we assume that $r = T[i\,..\,j]$ with $i \neq 1$. The case of $j \neq |T|$ is symmetric.

Suppose that an occurrence of T in S_1 starts at position i_1. The fragment matching r, i.e, $S_1[i_1+i-1\,..\,i_1+j-1]$, is periodic, has length at least d and period at most ρ, so it can be extended to a (d,ρ)-run in S_1. This run in S_1 starts at position i_1+i-1 due to $T[i-1] \neq T[i+\operatorname{per}(r)-1]$, so $p_1 := i_1+i-1 \in A_1$. The same argument shows that $p_2 := i_2+i-1 \in A_2$ if T occurs in S_2 at position i_2. Hence, (p_1,p_2) is a synchronized pair of anchors and T is a common anchored substring with respect to A_1, A_2. ◀

We are left with handling the case when the whole T is a (d,ρ)-run, i.e., when T is periodic with $|T| \geq d$ and $\operatorname{per}(T) \leq \rho$. In this case, we cannot guarantee that *every* pair of occurrences of T in A_1 and A_2 has a synchronized pair of anchors. For example, if $T = \mathtt{a}^d$ and $S_1 = S_2 = \mathtt{a}^n$ with $n \geq 2d$, this would require $\Omega(n/\sqrt{d})$ anchors. (There are $\Omega(n^2)$ pairs of occurrences, and each pair of anchors can accommodate at most $d+1$ out of these pairs.)

Hence, we focus on the leftmost occurrences of T and observe that they start within the first $\operatorname{per}(T)$ positions of (d,ρ)-runs. To achieve synchronization in these regions, we utilize the notion of the *Lyndon root* [13] $\operatorname{lyn}(X)$ of a periodic string X, defined as the lexicographically smallest rotation of $X[1\,..\,\operatorname{per}(X)]$. For each (d,ρ)-run x, we select as anchors the leftmost two positions where $\operatorname{lyn}(x)$ occurs within x (they must exist due to $d \geq 3\rho - 1$).

▶ **Lemma 11.** *Let A_1 and A_2 contain the first two positions where the Lyndon root occurs within each (d, ρ)-run of S_1 and S_2, respectively. If T is a common substring of S_1 and S_2 such that the whole T is a (d, ρ)-run, then T is a common anchored substring of S_1 and S_2.*

Proof. Let k be the leftmost position where $\mathrm{lyn}(T)$ occurs in T and $T = S_1[i_1 \mathinner{.\,.} j_1]$ be the leftmost occurrence of T in S_1. Since T is a (d, ρ)-run, $S_1[i_1 \mathinner{.\,.} j_1]$ can be extended to a (d, ρ)-run x in S_1. Note that $S_1[i_1 \mathinner{.\,.} j_1]$ starts within the first $\mathrm{per}(T)$ positions of x; otherwise, T would also occur at position $i_1 - \mathrm{per}(T)$. Consequently, position $i_1 + k - 1$ is among the first $2\mathrm{per}(T)$ positions of x, and it is a starting position of $\mathrm{lyn}(x) = \mathrm{lyn}(T)$. As the subsequent occurrences of $\mathrm{lyn}(x)$ within x start $\mathrm{per}(T)$ positions apart, we conclude that $i_1 + k - 1$ is one of the first two positions where $\mathrm{lyn}(x)$ occurs within x. Thus, $p_1 := i_1 + k - 1 \in A_1$. Symmetrically, $p_2 := i_2 + k - 1$ is added to A_2. Hence, (p_1, p_2) is a synchronized pair of anchors and T is a common anchored substring with respect to A_1, A_2. ◀

It remains to prove that Lemmas 10 and 11 yield $\mathcal{O}(\frac{n}{d})$ anchors and that this selection can be implemented efficiently. We use the following procedure as a subroutine:

▶ **Lemma 12** (Kociumaka et al. [19, Lemma 6])**.** *Given a string S, one can decide in $\mathcal{O}(|S|)$ time and $\mathcal{O}(1)$ space if S is periodic and, if so, compute $\mathrm{per}(S)$.*

First, we bound the number of (d, ρ)-runs and explain how to generate them efficiently.

▶ **Lemma 13.** *Consider a string S of length n and positive integers ρ, d with $3\rho - 1 \le d \le n$. The number of (d, ρ)-runs in S is $\mathcal{O}(\frac{n}{d})$. Moreover, there is an $\mathcal{O}(1)$-space $\mathcal{O}(n)$-time deterministic algorithm reporting them one by one along with their periods.*

Proof. Consider all fragments $u_k = S[k\rho \mathinner{.\,.} (k+2)\rho - 1]$ with boundaries within $[n]$. Observe that each (d, ρ)-run v contains at least one of the fragments u_k: if $v = S[i \mathinner{.\,.} j]$, then u_k with $k = \lceil i/\rho \rceil$ starts at $k\rho \ge i$ and ends at $(k+2)\rho - 1 \le i + \rho - 1 + 2\rho - 1 = i + 3\rho - 2 \le i + d - 1 \le j$. Moreover, if v contains u_k, then u_k is periodic with period $\mathrm{per}(u_k) = \mathrm{per}(v) \le \rho = \frac{1}{2}|u_k|$ (the first equality is due to $|u_k| = 2\rho \ge 2\mathrm{per}(v)$ and the periodicity lemma [18]), and v can be obtained by maximally extending u_k without increasing the shortest period.

This leads to a simple algorithm generating all (d, ρ)-runs, which processes subsequent integers k as follows: First, apply Lemma 12 to test if u_k is periodic and retrieve its period ρ_k. If this test is successful, then maximally extend u_k while preserving the period ρ_k, and denote the resulting fragment by v_k. If $|v_k| \ge d$, then report v_k as a (d, ρ)-run. We also introduce the following optimization: after processing k, skip all indices $k' > k$ for which $u_{k'}$ is still contained in v_k. (These indices k' are irrelevant due to $v_{k'} = v_k$ and they form an integer interval.)

The algorithm of Lemma 12 takes constant space and $\mathcal{O}(|u_k|)$ time, which sums up to $\mathcal{O}(n)$ across all indices k. The naive extension of u_k to v_k takes constant space and $\mathcal{O}(|v_k|)$ time. Due to the optimization, no two explicitly generated extensions v_k contain the same fragment $u_{k'}$. Hence, the total length of the fragments v_k (across indices k which were not skipped) is $\mathcal{O}(n)$. Thus, the overall running time is $\mathcal{O}(n)$ and the number of runs reported is $\mathcal{O}(\frac{n}{d})$. ◀

We conclude with a complete procedure generating anchors in the periodic case.

▶ **Proposition 14.** *There exists an $\mathcal{O}(1)$-space $\mathcal{O}(n)$-time algorithm that, given two strings S_1, S_2 of total length n, and parameters $d, \rho \in [n]$ with $d \ge 3\rho - 1$, outputs sets A_1, A_2 of size $\mathcal{O}(\frac{n}{d})$ satisfying the following property: If T is a common substring of S_1 and S_2 containing a (d, ρ)-run, then T is a common anchored substring of S_1 and S_2 with respect to A_1, A_2.*

Proof. The algorithm first uses the procedure of Lemma 13 to retrieve all (d, ρ)-runs in S_1 along with their periods. For each (d, ρ)-run $S_1[i \mathinner{.\,.} j]$, Duval's algorithm [16] is applied to find the minimum cyclic rotation of $S_1[i \mathinner{.\,.} i + \mathrm{per}(S_1[i \mathinner{.\,.} j]) - 1]$ in order to determine the Lyndon root $\mathrm{lyn}(S_1[i \mathinner{.\,.} j])$ represented by its occurrence at position $i + \Delta$ of S_1. Positions i, $i + \Delta$, $i + 2\Delta$, and j are reported as anchors in A_1. The same procedure is repeated for S_2 resulting in the elements of A_2 being reported one by one.

The space complexity of this algorithm is $\mathcal{O}(1)$, and the running time is $\mathcal{O}(n)$ (for Lemma 13) plus $\mathcal{O}(\rho) = \mathcal{O}(d)$ per (d, ρ)-run (for Duval's algorithm). This sums up to $\mathcal{O}(n)$ as the number of (d, ρ)-runs is $\mathcal{O}(\frac{n}{d})$. For the same reason, the number of anchors is $\mathcal{O}(\frac{n}{d})$.

For each T, the anchors satisfy the required property due to Lemma 10 or Lemma 11, depending on whether the (d, ρ) run contained in T is a proper fragment of T or not. ◀

4.3 $\tilde{\mathcal{O}}(n/\ell)$-space algorithm for arbitrary ℓ

The LCS_ℓ problem reduces to an instance of the LCAS problem with a combination of anchors for the non-periodic case and the periodic case. This yields the following result:

▶ **Theorem 15.** *The LCS_ℓ problem can be solved deterministically in $\mathcal{O}(\frac{n \log^* n}{\ell} + \log n)$ space and $\mathcal{O}(n \log n)$ time, and in $\mathcal{O}(\frac{n}{\ell} + \log n)$ space and $\mathcal{O}(n \log n + \ell \log^2 n)$ time with high probability using a Las-Vegas randomized algorithm.*

Proof. The algorithm first selects anchors A_1 and A_2 based on a $(\frac{1}{5}\ell, \frac{1}{5}\ell)$-partitioning set, as described in Section 4.1 (the partitioning set can be constructed using a deterministic or a randomized procedure). Additional anchors A'_1 and A'_2 are selected using Proposition 14 with $d = \frac{3}{5}\ell$ and $\rho = \frac{1}{5}\ell$. Finally, the algorithm runs the procedure of Theorem 7 with anchors $A_1 \cup A'_1$ and $A_2 \cup A'_2$ and forwards the obtained result to the output.

With this selection of anchors, every common substring T of length $|T| \geq \ell$ is a common anchored substring. Depending on whether T contains a $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run or not, this follows from Proposition 14 and Lemma 9, respectively. Consequently, the solution to the LCAS problem is a common substring of length at least $|T|$.

Proposition 14 yields $\mathcal{O}(\frac{n}{\ell})$ anchors whereas a partitioning set yields $\mathcal{O}(\frac{n \log^* n}{\ell})$ or $\mathcal{O}(\frac{n}{\ell})$ anchors, depending on whether a deterministic or a randomized construction is used. Consequently, the space and time complexity is as stated in the theorem, with the cost dominated by both the partitioning set construction and the algorithm of Theorem 7. ◀

▶ Remark 16. Note that the algorithms of Theorem 15 return a longest common substring as long as $\mathrm{lcs}(S_1, S_2) \geq \ell$ (and not just when $\ell \leq \mathrm{lcs}(S_1, S_2) \leq 2\ell$ as LCS_ℓ requires).

4.4 $\mathcal{O}(1)$-space algorithm for $\ell = \Omega(n)$

In Theorem 15, the space usage involves an $\mathcal{O}(\log n)$ term, which becomes dominant for very large ℓ. In this section, we design an alternative $\mathcal{O}(1)$-space algorithm for $\ell = \Omega(n)$. Later, in Theorem 19, we generalize this algorithm to arbitrary ℓ, which lets us obtain an analog of Theorem 15 with the $\mathcal{O}(\log n)$ term removed from the space complexity.

Our main tool is a constant-space pattern matching algorithm.

▶ **Lemma 17** (Galil-Seiferas [23], Crochemore-Perrin [15]). *There exists an $\mathcal{O}(1)$-space $\mathcal{O}(|P| + |T|)$-time algorithm that, given a read-only pattern P and a read-only text T, reports the occurrences of P in T in the left-to-right order.*

▶ **Lemma 18.** *The LCS_ℓ problem can be solved deterministically in $\mathcal{O}(1)$ space and $\mathcal{O}(n)$ time for $\ell = \Omega(n)$.*

Proof. We show how to find $\mathcal{O}(1)$ anchors such that if T is a common substring of S_1 and S_2 of length $|T| \geq \ell$, then T is a common anchored substring of S_1 and S_2.

We first use Proposition 14 with $d = \frac{3}{5}\ell$ and $\rho = \frac{1}{5}\ell$ to generate anchors A'_1 and A'_2 for the periodic case. The set of these anchors has a size of $\mathcal{O}(\frac{n}{d}) = \mathcal{O}(1)$, and, if T contains a $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run, then T is a common anchored substring of S_1 and S_2 with respect to A'_1, A'_2.

In order to accommodate the case where T does not contain any $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run, we construct sets A_1 and A_2 as follows. Consider all the fragments $u_k = S_1[k\frac{\ell}{5} \mathinner{.\,.} (k+3)\frac{\ell}{5} - 1]$ with boundaries within $[n]$. For each such fragment, add $k\frac{\ell}{5}$ into A_1. In addition, use Lemma 17 to find all occurrences of u_k in S_2, and add all the starting positions of the occurrences to A_2, unless the number of occurrences exceeds $\frac{n}{\ell/5}$ (then, $\mathrm{per}(u_k) \leq \frac{\ell}{5}$). The number of fragments u_k is $\mathcal{O}(\frac{n}{\ell/5}) = \mathcal{O}(1)$, so the sets A_1 and A_2 contain $\mathcal{O}(1)$ elements.

Let $T = S_1[i_1 \mathinner{.\,.} j_1] = S_2[i_2 \mathinner{.\,.} j_2]$ be arbitrary occurrences of T in S_1 and S_2, respectively. Then for $k = \lceil \frac{i_1}{\ell/5} \rceil$, the fragment u_k is contained within $S_1[i_1 \mathinner{.\,.} j_1]$. If $\mathrm{per}(u_k) \leq \frac{1}{5}\ell$, then the occurrence of u_k in T can be extended to a $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run in T (and that case has been accommodated using A'_1 and A'_2). Otherwise, $p_1 := k\frac{\ell}{5} \in A_1$ and all the positions where u_k occurs in S_2, including $p_2 := i_2 + (k\frac{\ell}{5} - i_1)$, are in A_2. Therefore, (p_1, p_2) is a synchronized pair of anchors and T is a common anchored substring with respect to A_1, A_2.

The number of pairs $(p_1, p_2) \in (A_1 \cup A'_1) \times (A_2 \cup A'_2)$ is $\mathcal{O}(1)$. For each such pair, the algorithm computes $|\mathrm{LCP}(S_1[p_1 \mathinner{.\,.}], S_2[p_2 \mathinner{.\,.}])| + |\mathrm{LCP}^r(S_1[\mathinner{.\,.} p_1 - 1], S_2[\mathinner{.\,.} p_2 - 1])|$ naively, and returns the common substring corresponding to a maximum among these values. The computation for each pair takes $\mathcal{O}(n)$ time. By the argument above, the algorithm finds a common substring of length at least $|T|$ for every common substring T with $|T| \geq \ell$. ◀

5 Time-space tradeoff for the LCS_ℓ problem

In this section, we show how to use the previous algorithms in order to solve the LCS_ℓ problem in space $\mathcal{O}(s)$, where s is a tradeoff paramater specified on the input. Our approach relies on the following algorithm which, given a paramter $m \geq \ell$, reduces a single arbitrary instance of LCS_ℓ to $\mathcal{O}(\lceil \frac{n}{m} \rceil^2)$ instances of LCS_ℓ with strings of length $\mathcal{O}(m)$.

Algorithm 1 Self-reduction of LCS_ℓ to many instances on strings of length $\mathcal{O}(m)$.

foreach $q_1 \in [|S_1|]$ *s.t.* $q_1 \equiv 1 \pmod m$ *and* $q_2 \in [|S_2|]$ *s.t.* $q_2 \equiv 1 \pmod m$ **do**
  Solve LCS_ℓ on $S_1[q_1 \mathinner{.\,.} \min(q_1 + 3m - 1, |S_1|)]$ and $S_2[q_2 \mathinner{.\,.} \min(q_2 + 3m - 1, |S_2|)]$;
return the longest among the common substrings reported;

Algorithm 1 clearly reports a common substring of S_1 and S_2. Moreover, if T is a common substring of S_1 and S_2 satisfying $\ell \leq |T| \leq 2\ell$, then T is contained in one of the considered pieces $S_1[q_1 \mathinner{.\,.} \min(q_1 + 3m - 1, |S_1|)]$ (the one with $q_1 = 1 + m\lfloor \frac{i_1}{m} \rfloor$ if $T = S_1[i_1 \mathinner{.\,.} j_1]$) and $T = S_2[i_2 \mathinner{.\,.} j_2]$ is contained in one of the considered pieces $S_2[q_2 \mathinner{.\,.} \min(q_2 + 3m - 1, |S_2|)]$ (the one with $q_2 = 1 + m\lfloor \frac{i_1}{m} \rfloor$ if $T = S_2[i_2 \mathinner{.\,.} j_2]$). Thus, the common substring reported by Algorithm 1 satisfies the characterization of the LCS_ℓ problem given in Remark 3.

▶ **Theorem 19.** *The LCS_ℓ problem can be solved deterministically in $\mathcal{O}(1)$ space and $\mathcal{O}(\frac{n^2}{\ell})$ time.*

Proof. We apply the self-reduction of Algorithm 1 with $m = \ell$ to the algorithm of Lemma 18. The running time is $\mathcal{O}((\frac{n}{m})^2 m) = \mathcal{O}(\frac{n^2}{m}) = \mathcal{O}(\frac{n^2}{\ell})$ and the space complexity is constant. ◀

This result allows for the aforementioned improvement upon the algorithms of Theorem 15.

▶ **Theorem 4.** *The LCS_ℓ problem can be solved deterministically in $\mathcal{O}(\frac{n\log^* n}{\ell})$ space and $\mathcal{O}(n\log n)$ time, and in $\mathcal{O}(\frac{n}{\ell})$ space and $\mathcal{O}(n\log n)$ time with high probability using a Las-Vegas randomized algorithm.*

Proof. If $\ell \ge \frac{n}{\log n}$, we use the algorithm of Theorem 19, which costs $\mathcal{O}(\frac{n^2}{\ell}) = \mathcal{O}(n\log n)$ time. Otherwise, we use the algorithm of Theorem 15. The running time is $\mathcal{O}(n\log n + \ell\log^2 n) = \mathcal{O}(n\log n)$, and the space complexity is $\mathcal{O}(\frac{n\log^* n}{\ell} + \log n) = \mathcal{O}(\frac{n\log^* n}{\ell})$ or $\mathcal{O}(\frac{n}{\ell} + \log n) = \mathcal{O}(\frac{n}{\ell})$, respectively. ◀

A time-space tradeoff is, in turn, obtained using Algorithm 1 on top of Theorem 4.

▶ **Theorem 20.** *Given a parameter $s \in [1, n]$, the LCS_ℓ problem can be solved deterministically in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2\log n\log^* n}{\ell\cdot s} + n\log n)$ time, and in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2\log n}{\ell\cdot s} + n\log n)$ time with high probability using a Las-Vegas randomized algorithm.*

Proof. For a randomized algorithm, we apply the self-reduction of Algorithm 1 with $m = \ell\cdot s$ to the algorithm of Theorem 4. The space complexity is $\mathcal{O}(\frac{m}{\ell}) = \mathcal{O}(s)$, whereas the running time is $\mathcal{O}(n\log n)$ if $m \ge n$ and $\mathcal{O}((\frac{n}{m})^2 \cdot m\log m) = \mathcal{O}(\frac{n^2\log m}{m}) = \mathcal{O}(\frac{n^2\log n}{\ell\cdot s})$ otherwise.

A deterministic version relies on the algorithm of Theorem 19 for $s < \log^* n$, which costs $\mathcal{O}(\frac{n^2}{\ell}) = \mathcal{O}(\frac{n^2\log^* n}{\ell\cdot s})$ time. For $s \ge \log^* n$, we apply the self-reduction of Algorithm 1 with $m = \frac{\ell\cdot s}{\log^* n}$ to the algorithm of Theorem 4. The space complexity is $\mathcal{O}(\frac{m\log^* n}{\ell}) = \mathcal{O}(s)$, whereas the running time is $\mathcal{O}(n\log n)$ if $m \ge n$ and $\mathcal{O}((\frac{n}{m})^2 \cdot m\log m) = \mathcal{O}(\frac{n^2\log m}{m}) = \mathcal{O}(\frac{n^2\log n\log^* n}{\ell\cdot s})$ otherwise. ◀

6 Time-space tradeoff for the LCS problem

In order to solve the LCS problem in time depending on $\mathrm{lcs}(S_1, S_2)$, we solve LCS_ℓ for exponentially decreasing thresholds ℓ.

Algorithm 2 A basic reduction from the LCS problem to the LCS_ℓ problem.

$\ell = n$;
do
  $\ell = \ell/2$;
  $T = LCS_\ell(S_1, S_2)$;
while $|T| < \ell$;
return T;

In Algorithm 2, as long as $\ell > \mathrm{lcs}(S_1, S_2)$, LCS_ℓ clearly returns a common substring shorter than ℓ. In the first iteration when this condition is not satisfied, we have $\ell \le \mathrm{lcs}(S_1, S_2) < 2\ell$, so LCS_ℓ must return a longest common substring.

If the algorithm of Theorem 19 is used for LCS_ℓ, then the space complexity is $\mathcal{O}(1)$, and the running time is $\mathcal{O}(\sum_{i=1}^{\lceil\log\frac{n}{L}\rceil} \frac{n^2}{n/2^i}) = \mathcal{O}(\sum_{i=1}^{\lceil\log\frac{n}{L}\rceil} n\cdot 2^i) = \mathcal{O}(\frac{n^2}{L})$,where $L = \mathrm{lcs}(S_1, S_2)$.

▶ **Theorem 2.** *The LCS problem can be solved deterministically in $\mathcal{O}(1)$ space and $\mathcal{O}(\frac{n^2}{L})$ time, where $L = \mathrm{lcs}(S_1, S_2)$.*

The $\mathcal{O}(1)$-space solution is still used if the input space restriction is $s = \mathcal{O}(\log n)$. Otherwise, we start with $\ell = \Theta(\frac{n}{s})$ (in the randomized version) or $\ell = \Theta(\frac{n\log^* n}{s})$ (in the deterministic version) and a single call to the algorithm of Theorem 15. This is correct due to

Remark 16, the space complexity is $\mathcal{O}(s)$, and the running time is $\mathcal{O}(n \log n)$. In subsequent iterations, the procedure of Theorem 20 is used, and its running time is dominated by the first term: $\mathcal{O}(\frac{n^2 \log n}{\ell \cdot s})$ or $\mathcal{O}(\frac{n^2 \log n \log^* n}{\ell \cdot s})$, respectively. These values form a geometric progression for exponentially decreasing ℓ, dominated by the running time of the last iteration: $\mathcal{O}(\frac{n^2 \log n}{L \cdot s})$ and $\mathcal{O}(\frac{n^2 \log n \log^* n}{L \cdot s})$, respectively. This analysis yields our main result:

▶ **Theorem 1.** *Given s with $1 \le s \le n$, the LCS problem with $L = \operatorname{lcs}(S_1, S_2)$ can be solved deterministically in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2 \log n \log^* n}{s \cdot L} + n \log n)$ time, and in $\mathcal{O}(s)$ space and $\mathcal{O}(\frac{n^2 \log n}{s \cdot L} + n \log n)$ time with high probability using a Las-Vegas randomized algorithm.*

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 Mai Alzamel, Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, Juliusz Straszynski, Tomasz Walen, and Wiktor Zuba. Quasi-linear-time algorithm for longest common circular factor. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, pages 25:1–25:14, 2019. doi:10.4230/LIPIcs.CPM.2019.25.

3 Amihood Amir, Panagiotis Charalampopoulos, Costas S. Iliopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common factor after one edit operation. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *24th International Symposium on String Processing and Information Retrieval, SPIRE 2017*, volume 10508 of *LNCS*, pages 14–26. Springer, 2017. doi:10.1007/978-3-319-67428-5_2.

4 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Longest common substring made fully dynamic. In *27th Annual European Symposium on Algorithms, ESA 2019*, pages 6:1–6:17, 2019. doi:10.4230/LIPIcs.ESA.2019.6.

5 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, *25th International Symposium on String Processing and Information Retrieval, SPIRE 2018*, volume 11147 of *LNCS*, pages 27–41. Springer, 2018. doi:10.1007/978-3-030-00479-8_3.

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 Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta. The "runs" theorem. *SIAM J. Comput.*, 46(5):1501–1514, 2017. doi:10.1137/15M1011032.

8 Philip Bille, Johannes Fischer, Inge Li Gørtz, Tsvi Kopelowitz, Benjamin Sach, and Hjalte Wedel Vildhøj. Sparse text indexing in small space. *ACM Transactions on Algorithms*, 12(3):39:1–39:19, 2016. doi:10.1145/2836166.

9 Or Birenzwige, Shay Golan, and Ely Porat. Locally consistent parsing for text indexing in small space. In *31st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 607–626. Society for Industrial and Applied Mathematics, 2020. doi:10.1137/1.9781611975994.37.

10 Rene De La Briandais. File searching using variable length keys. In *Western Joint Computer Conference*, pages 295–298. ACM Press, 1959. doi:10.1145/1457838.1457895.

11 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, *29th Annual Symposium on Combinatorial Pattern Matching, CPM 2018*, volume

105 of *LIPIcs*, pages 23:1–23:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CPM.2018.23.

12 Panagiotis Charalampopoulos, Paweł Gawrychowski, and Karol Pokorski. Dynamic longest common substring in polylogarithmic time. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

13 Maxime Crochemore, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. The maximal number of cubic runs in a word. *J. Comput. Syst. Sci.*, 78(6):1828–1836, 2012. doi:10.1016/j.jcss.2011.12.005.

14 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.

15 Maxime Crochemore and Dominique Perrin. Two-way string matching. *Journal of the ACM*, 38(3):651–675, 1991. doi:10.1145/116825.116845.

16 Jean-Pierre Duval. Factorizing words over an ordered alphabet. *Journal of Algorithms*, 4(4):363–381, 1983. doi:10.1016/0196-6774(83)90017-2.

17 Martin Farach-Colton. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102.

18 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. doi:10.1090/S0002-9939-1965-0174934-9.

19 Johannes Fischer, Travis Gagie, Paweł Gawrychowski, and Tomasz Kociumaka. Approximating LZ77 via small-space multiple-pattern matching. In Nikhil Bansal and Irene Finocchi, editors, *23rd Annual European Symposium on Algorithms, ESA 2015*, volume 9294 of *LNCS*, pages 533–544. Springer, 2015. doi:10.1007/978-3-662-48350-3_45.

20 Tomás Flouri, Emanuele Giaquinta, Kassian 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.

21 Edward Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960. doi:10.1145/367390.367400.

22 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Faster queries for longest substring palindrome after block edit. In *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019*, pages 27:1–27:13, 2019. doi:10.4230/LIPIcs.CPM.2019.27.

23 Zvi Galil and Joel I. Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983. doi:10.1016/0022-0000(83)90002-8.

24 Pawel Gawrychowski and Tomasz Kociumaka. Sparse suffix tree construction in optimal time and space. In Philip N. Klein, editor, *28th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 425–439. SIAM, 2017. doi:10.1137/1.9781611974782.27.

25 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

26 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

27 Lucas Chi Kwong Hui. Color set size problem with applications to string matching. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *3rd Annual Symposium on Combinatorial Pattern Matching, CPM 1992*, volume 644 of *LNCS*, pages 230–243. Springer, 1992. doi:10.1007/3-540-56024-6_19.

28 Tomohiro I, Juha Kärkkäinen, and Dominik Kempa. Faster sparse suffix sorting. In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects*

of Computer Science, STACS 2014, volume 25 of *LIPIcs*, pages 386–396. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.386.

29 Juha Kärkkäinen and Esko Ukkonen. Sparse suffix trees. In Jin-yi Cai and C. K. Wong, editors, *2nd Annual International Conference on Computing and Combinatorics, COCOON 1996*, volume 1090 of *LNCS*, pages 219–230. Springer, 1996. doi:10.1007/3-540-61332-3_155.

30 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.

31 Tomasz Kociumaka, Tatiana Starikovskaya, and Hjalte Wedel Vildhøj. Sublinear space algorithms for the longest common substring problem. In Andreas S. Schulz and Dorothea Wagner, editors, *22th Annual European Symposium on Algorithms, ESA 2014*, volume 8737 of *LNCS*, pages 605–617. Springer, 2014. doi:10.1007/978-3-662-44777-2_50.

32 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, 1999. doi:10.1109/SFFCS.1999.814634.

33 Mamoru Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985. doi:10.1145/214438.214445.

34 Donald R. Morrison. PATRICIA - practical algorithm to retrieve information coded in alphanumeric. *Journal of the ACM*, 15(4):514–534, 1968. doi:10.1145/321479.321481.

35 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

36 Tatiana Starikovskaya and Hjalte Wedel Vildhøj. Time-space trade-offs for the longest common substring problem. In Johannes Fischer and Peter Sanders, editors, *24th Annual Symposium on Combinatorial Pattern Matching, CPM 2013*, volume 7922 of *LNCS*, pages 223–234. Springer, 2013. doi:10.1007/978-3-642-38905-4_22.

37 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, *22nd Annual International Conference on Research in Computational Molecular Biology, RECOMB 2018*, volume 10812 of *LNCS*, pages 211–224. Springer, 2018. doi:10.1007/978-3-319-89929-9_14.

38 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.

39 Derrien Thomas, Estellé Jordi, Marco Sola Santiago, Knowles David G., Raineri Emanuele, Guigó Roderic, and Ribeca Paolo. Fast computation and applications of genome mappability. *PLOS ONE*, 7:1–16, January 2012. doi:10.1371/journal.pone.0030377.

40 Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, SWAT 1973*, pages 1–11, Washington, DC, USA, 1973. IEEE Computer Society. doi:10.1109/SWAT.1973.13.