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 \( O(n) \)-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 \( n^{2/3} \leq s \leq n \), the LCS problem can be solved in \( O(s) \) space and \( \tilde{O}(n^{2/3}) \) time.\(^1\) Kociumaka et al. (ESA 2014) generalized this tradeoff to \( 1 \leq s \leq n \), 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 \( 1 \leq s \leq n \), we show that the LCS problem can be solved in \( O(s) \) space and \( \tilde{O}(n^{2/3} + n) \) 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 \( lcs(S_1, S_2) \).

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 \( \Theta(n \log n) \) words.

\(^1\) The \( \tilde{O} \) notation hides \( \log^{O(1)} n \) factors.
Time-Space Tradeoffs for Finding a Long Common Substring

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 $O(n)$ 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 $O(n^2 + s \log n)$ time using $O(s)$ space. Kociumaka et al. [31] subsequently improved the running time to $O(n^2/s)$ 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 $O(n^2 \log^2 n (d \log^2 n + d^2))$ time and $O(s)$ space for $n^{2/3} \leq s \leq n$, whereas Kociumaka et al. [31] showed a solution which takes $O(n^2/s)$ time and $O(s)$ 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 $s \leq n \log n$ 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 $\text{lcs}(S_1, S_2) = 0$, 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 $\text{lcs}(S_1, S_2) < \ell$ and $\text{lcs}(S_1, S_2) \geq \ell$ gets easier as $\ell$ increases, or equivalently, whether $L := \text{lcs}(S_1, S_2)$ 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 $\Sigma = \{1, 2, \ldots, n^{O(1)}\}$. 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 $O(1)$ machine words. Our main result is as follows:

**Theorem 1.** Given $s$ with $1 \leq s \leq n$, the LCS problem with $L = \text{lcs}(S_1, S_2)$ can be solved deterministically in $O(s)$ space and $O(n^2 \log n \log^* n + n \log n)$ time, and in $O(s)$ space and $O(n^2 \log n / s + n \log n)$ 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 $s < n \log n$ 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 $O(1)$ space and $O(n^2/s)$ time, where $L = \text{lcs}(S_1, S_2)$.

As a step towards our main result, we solve the LCS$_L$ problem defined below.

---

2 The iterated logarithm function $\log^* x$ 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 $\log^{(k)} x \leq 1$.  

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 $O(s)$-space solution to the $LCS_\ell$ problem as an intermediate result.

An $\tilde{O}(n/\ell)$-space algorithm for the $LCS_\ell$ 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 $LCS_\ell$ 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 $O(\frac{n}{\ell})$ 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 $O(\frac{n}{\ell})$ extra positions. Overall, we get an $\tilde{O}(\frac{n}{\ell})$-space and $O(n)$-time algorithm for the $LCS_\ell$ problem.
An $O(s)$ space algorithm for the LCS$_t$ problem. In Section 5, we give a time-space tradeoff for the LCS$_t$ 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 $1 \leq s \leq n$, the algorithm takes $O(s)$ space and $O(n^2/n + n)$ time.

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

2 Preliminaries

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

A string $S$ of length $n = |S|$ is a finite sequence of characters $S[1]S[2] \cdots S[n]$ over an alphabet $\Sigma$; in this paper, we consider polynomially-bounded integer alphabets $\Sigma = [1..n^{O(1)}]$. 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 $1 \leq x \leq y \leq |S|$. 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 $S_1$ and $S_2$ 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 LCP$(S_1, S_2)$, and the longest common suffix is denoted by LCP$^r(S_1, S_2)$. Note that LCP$^r(S_1, S_2) = (\text{LCP}(S_1', S_2'))^r$.

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 per$(S)$. If $\text{per}(S) \leq \frac{1}{2}|S|$, 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 $\rho$, we say that a run $S[i..j]$ is a $(d, \rho)$-run if $|S[i..j]| \geq d$ and per$(S[i..j]) \leq \rho$. Note that every periodic fragment $S[i'..j']$ with $|S[i'..j']| \geq d$ and per$(S[i'..j']) \leq \rho$ can be uniquely extended to a $(d, \rho)$-run $S[i..j]$ while preserving the shortest period per$(S[i..j]) = \text{per}(S[i'..j'])$.

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 $O(|\mathcal{F}|)$ 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 $p_1 - i_1 = p_2 - i_2 \in [0, |T|]$.\footnote{Note that the anchors could be at positions $p_1 = j_1 + 1$ and $p_2 = j_2 + 1$ (if $p_1 - i_1 = p_2 - i_2 = |T|$).}
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.

\[ \text{Two String Families LCP (Charalampopoulos et al. [11])} \]

**Input:** A compact trie \( \mathcal{T}(\mathcal{F}) \) of a family of strings \( \mathcal{F} \) and two sets \( P, Q \subseteq \mathcal{F}^2 \).

**Output:** The value \( \text{maxPairLCP}(P, Q) \), defined as

\[
\text{maxPairLCP}(P, Q) = \max \{|\text{LCP}(P_1, Q_1)| + |\text{LCP}(P_2, Q_2)| : (P_1, P_2) \in P, (Q_1, Q_2) \in Q, |P_1| = |Q_1|, |P_2| = |Q_2| \}
\]

along with pairs \((P_1, P_2) \in P \) and \((Q_1, Q_2) \in Q \) 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 \( O(|\mathcal{F}| + N \log N) \) time using \( O(|\mathcal{F}| + N) \) space\(^4\), where \( N = |P| + |Q| \).

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

\[
\mathcal{F} = \{S_1[p..] : p \in A_1\} \cup \{(S_1[.p-1])^r : p \in A_1\}
\]
\[
\cup \{S_2[p..] : p \in A_2\} \cup \{(S_2[.p-1])^r : p \in A_2\},
\]

(1)

\[
P = \{(S_1[p..], (S_1[.p-1])^r) : p \in A_1\},
\]

(2)

\[
Q = \{(S_2[p..], (S_2[.p-1])^r) : p \in A_2\}.
\]

(3)

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 Birenzvige 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 \( O(n \log \frac{n}{|B|}) \) time and \( O(|B| + \log n) \) space.

**Theorem 7.** The Longest Common Anchored Substring problem can be solved in \( O(n \log n) \) time using \( O(|A_1| + |A_2| + \log n) \) 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.
5.6 Time-Space Tradeoffs for Finding a Long Common Substring

Proof. We implicitly create a string \( S = S_1S_1^\tau S_2^\tau S_2^\tau \) and construct a sparse suffix tree of \( S \) containing the following suffixes: \( S_1[p..]S_1^\tau S_2^\tau S_2^\tau \) and \( (S_1[p..]S_1^\tau S_2^\tau S_2^\tau) \) for \( p \in A_1 \), as well as \( S_2[p..]S_2^\tau \) and \( (S_2[p..]S_2^\tau) \) for \( p \in A_2 \). We then trim this tree, cutting edges immediately above any \( \$ \) on their labels, which results in the compacted trie \( T(F) \) for the family \( F \) 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 \( T(F), P, Q \). This yields pairs in \( P \) and \( Q \) for which maxPairLCP(\( P, Q \)) 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: \( \text{LCP}(S_1[p_1..], S_2[p_2..]) \).

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

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

Our approach to solve the \( LCS_k \) 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 \( \ell \) is a common anchored substring. In other words, we need to make sure that \( T \) admits occurrences \( S_1[i..j] = T = S_2[i..j] \) with a synchronized pair of anchors.

As a warm-up, we describe a simple selection of \( O(n/\sqrt{\tau}) \) 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 \( \tau, m \) with \( 1 \leq \tau \leq m \), this technique yields a set \( DC_\tau(m) \subseteq [m] \) of size \( O(m/\sqrt{\tau}) \) 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 \( \ell \) is anchored, it suffices to select all \( O(n/\sqrt{\tau}) \) positions in \( DC_\tau(n) \) as anchors: \( A_1 = [|S_1|] \cap DC_\tau(n) \) and \( A_2 = [|S_2|] \cap DC_\tau(n) \).

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 \( \ell \)). In fact, any non-adaptive construction needs \( \Omega(n/\sqrt{\tau}) \) anchors in order to guarantee that every common substring \( T \) of length at least \( \ell \) is a common anchored substring. In the following, we show how adaptivity allows us to achieve the same goal using only \( \tilde{O}(n/\ell) \) anchors.

4.1 Selection of Anchors: the non-periodic case

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

▷ Definition 8 (Birenzweig et al. [9]). A set of positions \( P \subseteq [n] \) is called a \((\tau, \delta)\)-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 \( \text{per}(u) \leq \tau \).

Note that any \((\tau, \delta)\)-partitioning set is also a \((\tau', \delta')\)-partitioning set for any \( \tau' \geq \tau \) and \( \delta' \geq \delta \). The selection of anchors is based on an arbitrary \((\frac{\ell}{k}, \frac{\ell}{k})\)-partitioning set \( P \) of the string \( S_1S_2 \); for every position \( p \in P \), \( p \) is added to \( A_1 \) (if \( p \leq |S_1| \) 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 \( |T| \geq \ell \) which does not contain a \((\frac{\ell}{k}, \frac{\ell}{k})\)-run. Then, \( T \) is a common anchored substring with respect to \( A_1, A_2 \) defined above.
Proof. Let $S_1[i_1 \ldots j_1]$ and $S_2[i_2 \ldots j_2]$ be arbitrary occurrences of $T$ in $S_1$ and $S_2$, respectively. If there is a position $p_1 \in A_1$ with $p_1 \in [i_1 + \delta \ldots j_1 - \delta]$, 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 $S_1[p_1 - \delta \ldots p_1 + \delta] = S_2[p_2 - \delta \ldots p_2 + \delta]$. 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 $S_1[i_1 + \delta \ldots j_1 - \delta]$ is contained within a block between two consecutive positions of the partitioning set. The length of this block is at least $|T| - 2\delta = \frac{3}{2}\ell > \tau$, so the block is periodic by the compactness property of the partitioning set. Hence, $\text{per}(T[1 + \delta \ldots |T| - \delta]) = \text{per}(S_1[i_1 + \delta \ldots j_1 - \delta]) \leq \tau = \frac{1}{5}\ell$. A $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-run in $T$ can thus be obtained by maximally extending $T[1 + \delta \ldots |T| - \delta]$ without increasing the shortest period. Such a run in $T$ is a contradiction to the assumption. $\blacksquare$

Birenzwige et al. [9] gave a deterministic algorithm that constructs a $(\tau, \tau \log^* n)$-partitioning set of size $O(\frac{n}{\tau})$ in $O(n \log \tau)$ time using $O(\frac{n}{\tau} + \log \tau)$ space. Setting appropriate $\tau = \Theta(\ell/\log^* n)$, we get an $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-partitioning set of size $O(n \log^* n)$.

Furthermore, Birenzwige et al. [9] gave a Las-Vegas randomized algorithm that constructs a $(\tau, \tau)$-partitioning set of size $O(\frac{n}{\tau})$ in $O(n + \tau \log^2 n)$ time with high probability, using $O(\frac{n}{\tau} + \log n)$ space. Setting $\tau = \frac{1}{5}\ell$, we get an $(\frac{3}{5}\ell, \frac{1}{5}\ell)$-partitioning set of size $O(\frac{n}{\tau})$.

### 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 $(d, \rho)$-run by selecting $O(\frac{n}{d})$ 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 $(d, \rho)$-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 $(d, \rho)$-run.

$\blacksquare$ **Lemma 10.** Let $A_1$ and $A_2$ contain the boundary positions of every $(d, \rho)$-run in $S_1$ and $S_2$, respectively. If $T$ is a common substring of $S_1$ and $S_2$ with a $(d, \rho)$-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 $r = T[i \ldots j]$ 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, $S_1[i_1 + i - 1 \ldots i_1 + j - 1]$, is periodic, has length at least $d$ and period at most $\rho$, so it can be extended to a $(d, \rho)$-run in $S_1$. This run in $S_1$ starts at position $i_1 + i - 1$ due to $T[i - 1] \neq T[i + \text{per}(r) - 1]$, 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 $A_1, A_2$. $\blacksquare$

We are left with handling the case when the whole $T$ is a $(d, \rho)$-run, i.e., when $T$ is periodic with $|T| \geq d$ and $\text{per}(T) \leq \rho$. 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 $T = a^d$ and $S_1 = S_2 = a^n$ with $n \geq 2d$, this would require $O(n/\sqrt{d})$ 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 $\text{per}(T)$ positions of $(d, \rho)$-runs. To achieve synchronization in these regions, we utilize the notion of the Lyndon root [13] $\text{lyn}(X)$ of a periodic string $X$, defined as the lexicographically smallest rotation of $X[1 \ldots \text{per}(X)]$. For each $(d, \rho)$-run $x$, we select as anchors the leftmost two positions where $\text{lyn}(x)$ 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 $(d,\rho)$-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 $(d,\rho)$-run, then $T$ is a common anchored substring of $S_1$ and $S_2$.

Proof. Let $k$ be the leftmost position where $\text{lyn}(T)$ occurs in $T$ and $T = S_1[i_1 \ldots j_1]$ be the leftmost occurrence of $T$ in $S_1$. Since $T$ is a $(d,\rho)$-run, $S_1[i_1 \ldots j_1]$ can be extended to a $(d,\rho)$-run $x$ in $S_1$. Note that $S_1[i_1 \ldots j_1]$ starts within the first $\text{per}(T)$ positions of $x$; otherwise, $T$ would also occur at position $i_1 + \text{per}(T)$. Consequently, position $i_1 + k - 1$ is among the first $2\text{per}(T)$ positions of $x$, and it is a starting position of $\text{lyn}(x) = \text{lyn}(T)$. As the subsequent occurrences of $\text{lyn}(x)$ within $x$ start per($T$) positions apart, we conclude that $i_1 + k - 1$ is one of the first two positions where $\text{lyn}(x)$ 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 $O(n^2)$ 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 $O(|S|)$ time and $O(1)$ space if $S$ is periodic and, if so, compute per($S$).

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

Lemma 13. Consider a string $S$ of length $n$ and positive integers $\rho, d$ with $3\rho - 1 \leq d \leq n$. The number of $(d,\rho)$-runs in $S$ is $O(n^2)$. Moreover, there is an $O(1)$-space $O(n)$-time deterministic algorithm reporting them one by one along with their periods.

Proof. Consider all fragments $u_k = S[kp \ldots (k+2)\rho - 1]$ with boundaries within $[n]$. Observe that each $(d,\rho)$-run $v$ contains at least one of the fragments $u_k$: if $v = S[i \ldots j]$, then $u_k$ with $k = [i/\rho]$ starts at $k\rho \geq i$ and ends at $(k+2)\rho - 1 \leq i + \rho - 1 + 2\rho - 1 = i + 3\rho - 2 \leq i + d - 1 \leq j$. Moreover, if $v$ contains $u_k$, then $u_k$ is periodic with period $\text{per}(u_k) = \text{per}(v) \leq \rho = \lfloor u_k \rfloor$ (the first equality is due to $|u_k| = 2\rho \geq 2\text{per}(v)$ 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 $(d,\rho)$-runs, which processes subsequent integers $k$ as follows: First, apply Lemma 12 to test if $u_k$ is periodic and retrieve its period $\rho_k$. If this test is successful, then maximally extend $u_k$ while preserving the period $\rho_k$ and denote the resulting fragment by $v_k$. If $|v_k| \geq d$, then report $v_k$ as a $(d,\rho)$-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 $O(|u_k|)$ time, which sums up to $O(n)$ across all indices $k$. The naive extension of $u_k$ to $v_k$ takes constant space and $O(|v_k|)$ 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 $O(n)$. Thus, the overall running time is $O(n)$ and the number of runs reported is $O(n^2)$.

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

Proposition 14. There exists an $O(1)$-space $O(n)$-time algorithm that, given two strings $S_1, S_2$ of total length $n$, and parameters $d, \rho \in [n]$ with $d \geq 3\rho - 1$, outputs sets $A_1, A_2$ of size $O(n^2)$ satisfying the following property: If $T$ is a common substring of $S_1$ and $S_2$ containing a $(d,\rho)$-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 \((d, \rho)-\)runs in \(S_1\) along with their periods. For each \((d, \rho)-\)run \(S_1[i..j]\), Duval's algorithm [16] is applied to find the minimum cyclic rotation of \(S_1[i..i + \text{per}(S_1[i..j]) - 1]\) in order to determine the Lyndon root \(\text{lyn}(S_1[i..j])\) 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 \(O(1)\), and the running time is \(O(n)\) (for Lemma 13) plus \(O(\rho) = O(d)\) per \((d, \rho)-\)run (for Duval’s algorithm). This sums up to \(O(n)\) as the number of \((d, \rho)-\)runs is \(O(n^2)\). For the same reason, the number of anchors is \(\tilde{O}(n^2)\).

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

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

The \(\text{LCS}_\ell\) 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:

\begin{itemize}
  \item \textbf{Theorem 15.} The \(\text{LCS}_\ell\) problem can be solved deterministically in \(O(n \log^* n + \log n)\) space and \(O(n \log n)\) time, and in \(O(\log^* n + \log n)\) space and \(O(n \log n + \ell \log^2 n)\) time with high probability using a Las-Vegas randomized algorithm.
\end{itemize}

Proof. The algorithm first selects anchors \(A_1\) and \(A_2\) based on a \((\frac{1}{2} \ell, \frac{1}{2} \ell)\)-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 \(d = \frac{3}{2} \ell\) and \(\rho = \frac{1}{2} \ell\). 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 \((\frac{3}{2} \ell, \frac{1}{2} \ell)\)-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 \(O(n^2)\) anchors whereas a partitioning set yields \(O(n \log^* n)\) or \(O(\frac{n}{\ell})\) 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. ▶

\begin{itemize}
  \item \textbf{Remark 16.} Note that the algorithms of Theorem 15 return a longest common substring as long as \(\text{lcs}(S_1, S_2) \geq \ell\) (and not just when \(\ell \leq \text{lcs}(S_1, S_2) \leq 2 \ell\) as \(\text{LCS}_\ell\) requires).
\end{itemize}

### 4.4 \(O(1)\)-space algorithm for \(\ell = \Omega(n)\)

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

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

\begin{itemize}
  \item \textbf{Lemma 17 (Galil-Seiferas [23], Crochemore-Perrin [15]).} There exists an \(O(1)\)-space \(O(|P| + |T|)\)-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.
  \item \textbf{Lemma 18.} The \(\text{LCS}_\ell\) problem can be solved deterministically in \(O(1)\) space and \(O(n)\) time for \(\ell = \Omega(n)\).
\end{itemize}
Time-Space Tradeoffs for Finding a Long Common Substring

Proof. We show how to find \(O(1)\) 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 \(d = \frac{\ell}{4}\) and \(p = \frac{\ell}{4}\) to generate anchors \(A'_1\) and \(A'_2\) for the periodic case. The set of these anchors has a size of \(O(\frac{n}{m}) = O(1)\), and, if \(T\) contains a \((\frac{3\ell}{4}, \frac{\ell}{4})\)-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 \((\frac{3\ell}{4}, \frac{\ell}{4})\)-run, we construct sets \(A_1\) and \(A_2\) as follows. Consider all the fragments \(u_k = S_1[k_2 \ldots (k + 3)\ell - 1]\) with boundaries within \([n]\). For each such fragment, add \(k\ell\) 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 \(\frac{n}{m}\) (then, \(\text{per}(u_k) \leq \frac{n}{m}\)). The number of fragments \(u_k\) is \(O(\frac{n}{m}) = O(1)\), so the sets \(A_1\) and \(A_2\) contain \(O(1)\) elements.

Let \(T = S_1[i_1 \ldots j_1] = S_2[i_2 \ldots j_2]\) be arbitrary occurrences of \(T\) in \(S_1\) and \(S_2\), respectively. Then for \(k = \lceil \frac{1}{\ell} \rceil\), the fragment \(u_k\) is contained within \(S_1[i_1 \ldots j_1]\). If \(\text{per}(u_k) \leq \frac{n}{m}\), then the occurrence of \(u_k\) in \(T\) can be extended to a \((\frac{3\ell}{4}, \frac{\ell}{4})\)-run in \(T\) (and that case has been accommodated using \(A_1\) and \(A_2\)). Otherwise, \(p_1 := k_2 \in A_1\) and all the positions where \(u_k\) occurs in \(S_2\), including \(p_2 := i_2 + (k_\ell - i_1)\), 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 \((p_1, p_2)\) \((A_1 \cup A'_1) \times \) \((A_2 \cup A'_2)\) is \(O(1)\). For each such pair, the algorithm computes \(\text{LCP}(i; S_1[p_1], S_2[p_2]) + \text{LCP}'(i; S_1[i - 1], S_2[p_2])\) naively, and returns the common substring corresponding to a maximum among these values. The computation for each pair takes \(O(n)\) 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\(\ell\) problem

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

Algorithm 1  Self-reduction of LCS\(\ell\) to many instances on strings of length \(O(m)\).

1 foreach \(q_1 \in [[S_1]]\) s.t. \(q_1 \equiv 1\) (mod \(m\)) and \(q_2 \in [[S_2]]\) s.t. \(q_2 \equiv 1\) (mod \(m\)) do
2     Solve LCS\(\ell\) on \(S_1[q_1 \ldots \min(q_1 + \ell - 1, |S_1|)]\) and \(S_2[q_2 \ldots \min(q_2 + \ell - 1, |S_2|)]\);
3 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 \(S_1[q_1 \ldots \min(q_1 + \ell - 1, |S_1|)]\) (the one with \(q_1 = 1 + m\lfloor \frac{\ell}{m}\rfloor\) if \(T = S_1[i_1 \ldots j_1]\)) and \(T = S_2[i_2 \ldots j_2]\) is contained in one of the considered pieces \(S_2[q_2 \ldots \min(q_2 + \ell - 1, |S_2|)]\) (the one with \(q_2 = 1 + m\lfloor \frac{\ell}{m}\rfloor\) if \(T = S_2[i_2 \ldots j_2]\)). Thus, the common substring reported by Algorithm 1 satisfies the characterization of the LCS\(\ell\) problem given in Remark 3.

\(\blacktriangleright\) Theorem 19. The LCS\(\ell\) problem can be solved deterministically in \(O(1)\) space and \(O(\frac{n^2}{m^2})\) time.

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

This result allows for the aforementioned improvement upon the algorithms of Theorem 15.
The LCSℓ problem can be solved deterministically in $O(n \log^{s} n)$ space and $O(n \log n)$ time, and in $O(l_{s})$ space and $O(n \log n)$ time with high probability using a Las-Vegas randomized algorithm.

Proof. If $\ell \geq \frac{n}{\log n}$, we use the algorithm of Theorem 19, which costs $O(l_{s}) = O(n \log n)$ time. Otherwise, we use the algorithm of Theorem 15. The running time is $O(n \log n + \ell \log^{2} n) = O(n \log n)$, and the space complexity is $O(n \log^{s} n + \log n) = O(n \log^{s} n)$ or $O(\frac{n}{\ell} + \log n) = O(\frac{n}{\ell})$, 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 LCSℓ problem can be solved deterministically in $O(s)$ space and $O(\frac{n^{2} \log^{s} n \cdot \log^{2} n}{s} + n \log n)$ time, and in $O(s)$ space and $O(\frac{n^{2} \log^{s} n}{s} + n \log n)$ 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 $O(l_{s}) = O(s)$, whereas the running time is $O(n \log n)$ if $m \geq n$ and $O((\frac{n}{m})^{2} \cdot m \log m) = O(\frac{n^{2} \log m}{m}) = O(\frac{n^{2} \log n}{s})$ otherwise.

A deterministic version relies on the algorithm of Theorem 19 for $s < \log^{*} n$, which costs $O(l_{s}) = O(\frac{n^{2} \log^{s} n}{s})$ time. For $s \geq \log^{*} n$, 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 $O(\frac{m \log^{s} n}{s}) = O(s)$, whereas the running time is $O(n \log n)$ if $m \geq n$ and $O((\frac{n}{m})^{2} \cdot m \log m) = O(\frac{n^{2} \log m}{m}) = O(\frac{n^{2} \log^{s} n}{s})$ otherwise.

6 Time-space tradeoff for the LCS problem

In order to solve the LCS problem in time depending on lcs(S1, S2), we solve LCSℓ for exponentially decreasing thresholds ℓ.

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

```
1 ℓ = n;
2 do
3   ℓ = ℓ/2;
4   T = LCSℓ(S1, S2);
5 while |T| < ℓ;
6 return T;
```

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

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

Theorem 21. The LCS problem can be solved deterministically in $O(1)$ space and $O(\frac{n^{2}}{2^{\ell}})$ time, where $L = \text{lcs}(S_1, S_2)$.

The $O(1)$-space solution is still used if the input space restriction is $s = O(\log n)$. Otherwise, we start with $\ell = \Theta(\frac{n}{\log n})$ (in the randomized version) or $\ell = \Theta(\frac{n \log n}{\log^{2} n})$ (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 $O(s)$, and the running time is $O(n \log n)$. In subsequent iterations, the procedure of Theorem 20 is used, and its running time is dominated by the first term: $O(\frac{n^2 \log n}{L_s})$ or $O(\frac{n^2 \log n \log^* n}{L_s})$, respectively. These values form a geometric progression for exponentially decreasing $\ell$, dominated by the running time of the last iteration: $O(\frac{n^2 \log n}{L_s})$ and $O(\frac{n^2 \log n \log^* n}{L_s})$, respectively. This analysis yields our main result:

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

### References


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
Time-Space Tradeoffs for Finding a Long Common Substring


