The Longest Filled Common Subsequence Problem

Mauro Castelli[1], Riccardo Dondi[2], Giancarlo Mauri[3], and Italo Zoppis[4]

1 **NOVA IMS, Universidade Nova de Lisboa, Lisbon, Portugal**
mcastelli@isegi.unl.pt
2 **Dipartimento di Lettere, Filosofia, Comunicazione, Università degli Studi di Bergamo, Bergamo, Italy**
riccardo.dondi@unibg.it
3 **Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano, Italy**
mauri@disco.unimib.it
4 **Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano, Italy**
zoppis@disco.unimib.it

Abstract

Inspired by a recent approach for genome reconstruction from incomplete data, we consider a variant of the longest common subsequence problem for the comparison of two sequences, one of which is incomplete, i.e. it has some missing elements. The new combinatorial problem, called Longest Filled Common Subsequence, given two sequences A and B, and a multiset $\mathcal{M}$ of symbols missing in B, asks for a sequence B^* obtained by inserting the symbols of $\mathcal{M}$ into B so that B^* induces a common subsequence with A of maximum length.

First, we investigate the computational and approximation complexity of the problem and we show that it is NP-hard and APX-hard when A contains at most two occurrences of each symbol. Then, we give a $\frac{3}{5}-$approximation algorithm for the problem. Finally, we present a fixed-parameter algorithm, when the problem is parameterized by the number of symbols inserted in B that "match" symbols of A.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases longest common subsequence, approximation algorithms, computational complexity, fixed-parameter algorithms

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.14

1 Introduction

The comparison of sequences via Longest Common Subsequence (LCS) has been applied in several contexts where we want to retrieve the maximum number of elements that appear in the same order in two or more sequences. There are well-known fields of application of LCS like scheduling and data compression, a notable example is the DIFF utility to compute the differences between two files.

The extraction of common subsequences has been widely applied to compare molecular sequences in bioinformatics [17, 14]. For example, the comparison of biological sequences provides a measure of their similarities and differences, aiming at understanding whether they encode similar/different functionalities. Different approaches for the comparison of two

genomes based on LCS have been considered in the last years, leading to variants of the longest common subsequence problem, like the constrained longest common subsequence [13, 8, 18, 11, 4] or the repetition-free longest common subsequence and variants thereof [7, 1, 6, 12].

The approaches based on LCS for genome comparison assume that the input sequences are complete, that is there are no missing data. However, while Next Generation Sequencing technologies are able to produce a huge amount of DNA/RNA fragments, the cost of reconstructing a complete genome is still high [10]. Hence, released genomes often contain errors or are incomplete. These incomplete genomes are called scaffolds. One approach to the reconstruction of genome is to fill scaffolds with missing genes, based on the comparison of an incomplete genome with a reference genome [16, 15, 9, 19]. Given an incomplete genome B, a set of missing genes (symbols) $\mathcal{M}$ and a reference genome A, the goal is to insert the missing symbols in B so that the number of common adjacencies between the resulting genome B^* and A is maximized. We have a common adjacency when two genes a, b are consecutive both in A and B^*, independently from the order. We mention briefly that there is also a variant of the scaffold filling approach that compares two incomplete genomes [15, 9].

Inspired by methods for genome comparison based on LCS and by the scaffold filling approach, we introduce a new variant of the LCS problem, called the *Longest Filled Common Subsequence* problem, for the comparison of a complete genome A and an incomplete genome B. The goal of the problem is to find the maximum number of genes that appear in the same order in both genomes. However, since some of the genes in B are missing (a multiset $\mathcal{M}$ of symbols), we have to compute a longest common subsequence of A and of a filling B^* of B, that is of a sequence obtained from B by inserting the symbols of $\mathcal{M}$ into B. Notice that while the scaffold filling problem aims to reconstruct a complete genome from an incomplete one by maximizing the number of common adjacencies, here we aim to infer only those elements (genes) that appear in the same order in the complete genome A and in the completed genome B^*.

In this paper, we investigate different algorithmic and complexity aspects of the Longest Filled Common Subsequence problem. First, in Section 3 we prove that it is NP-hard and APX-hard, even when genome A contains at most two occurrences of each symbol. Notice that bounding the maximum number of occurrences of symbols in a sequence is relevant in this case, as usually the number of copies of a gene inside a genome is bounded. Then, in Section 4 we present a polynomial-time approximation algorithm of factor $\frac{3}{5}$. In Section 5, we give a fixed-parameter algorithm, where the parameter is the number of inserted symbols that lead to a "match" with symbols of sequence A. Such a parameter can be of interest when the number of missing elements, and in particular those that lead to a "match" with symbols of A, is moderate, as the complexity of the algorithm depends exponentially only on this parameter.

Some of the proofs are omitted due to page limit.

2 Preliminaries

In this section we introduce some basic definitions that will be useful in the rest of the paper and we give the formal definition of the Longest Filled Common Subsequence problem. Let S be a sequence over an alphabet Σ, we denote by $|S|$ the length of S. Given a position i, with $1 \leq i \leq |S|$, we denote by $S[i]$ the symbol in position i of S. Given two positions i, j in S, with $1 \leq i \leq j \leq |S|$, we denote by $S[i, j]$ the substring of S that starts at position i and ends at position j. Given two sequences S and T, we denote by $S \cdot T$ the sequence that results by concatenating S and T.

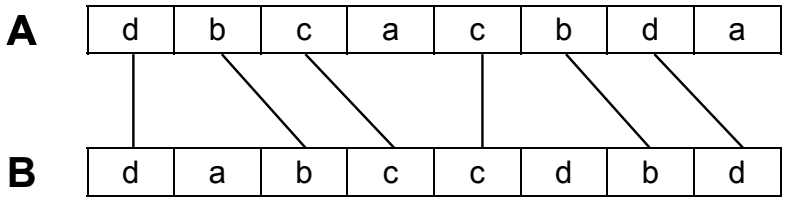

Figure 1 The threading schema of two sequences A and B: lines connect matched positions of A and B.

A subsequence of S is a sequence S' that is obtained from S by deleting some symbols (possibly none). A common subsequence S of two sequences A and B is a subsequence of both A and B. A longest common subsequence of A and B is a common subsequence of A and B having maximum length.

Given two sequences A and B, a common subsequence can be defined by aligning A and B and by connecting two positions of A and B containing an identical symbol with a line, such that there is no pair of crossing lines. This is called a *threading schema* (see Fig. 1). Given a threading schema for sequences A, B, a connection between two symbols in A and B, respectively, is called a *match* and the two positions incident in a line are said to be *matched*.

Given a sequence S and a multiset of symbols $\mathcal{M}$, we define a *filling* of S with $\mathcal{M}$ as a sequence S' obtained by inserting a subset $\mathcal{M}'$ of symbols of $\mathcal{M}$ into S. Notice that in a filling of S with $\mathcal{M}$ not all the symbols of $\mathcal{M}$ have to be inserted in S. Informally, we may not insert those symbols that do not induce matches, to simplify the algorithms we describe in Section 4 and in Section 5. Now, we are ready to present the formal definition of Longest Filled Common Subsequence.

▶ **Problem 1.** *Longest Filled Common Subsequence ($\mathcal{LFCS}$)*
Instance: *two sequences A and B over an alphabet Σ, and a multiset $\mathcal{M}$ over Σ.*
Solution: *a filling B^* of B with $\mathcal{M}$.*
Measure: *the length of a longest common subsequence of A and B^* (to be maximized).*

Given two sequences A, B and a multiset $\mathcal{M}$ over Σ, let B^* be a filling of B with $\mathcal{M}$. Consider a common subsequence of A and B^*, and their corresponding threading schema. A position of A can have two possible kinds of matches (see Fig. 2): a match with a position of B^* that contains a symbol of $\mathcal{M}$ inserted in B, called *match by insertion*, or a match with a position of B^* not involved in an insertion, called *match by alignment*. We can easily compute in polynomial-time two upper bounds on the number of positions of A that can be matched by alignment and by insertion, that will be useful in Section 4. The first upper bound is related to a longest common subsequence L of A and B, which can be computed in polynomial time. In fact, the maximum number of positions of A (and of a filling B^* of B with $\mathcal{M}$) that are matched by alignment is at most the length of L.

Next we show how to compute in polynomial-time an upper bound on the number of positions of a sequence A that can be matched by insertion. First, given a multiset $\mathcal{M}$ of symbols, we define an *ordering of* $\mathcal{M}$ as a sequence obtained by defining an order among each element of $\mathcal{M}$, that is each occurrence of a symbol of $\mathcal{M}$.

Consider the positions of A and of a filling B^* of B with $\mathcal{M}$ that are matched by insertion; the positions of A induce a subsequence A' of A, while the positions of B^* induce an ordering M of a subset $\mathcal{M}' \subseteq \mathcal{M}$. An upper bound on the length of M can be computed in polynomial time with the following greedy algorithm.

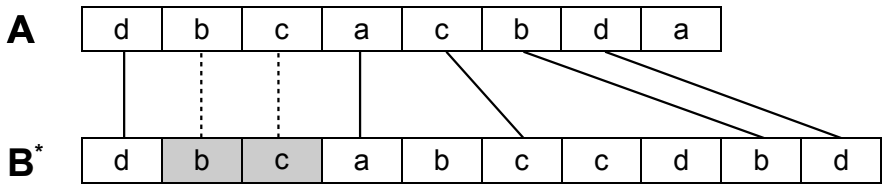

Figure 2 A filling B^* of sequence B in Fig. 1, computed by inserting a symbol in position 2 (symbol b) and a symbol in position 3 (symbol c), both in grey. A subsequence of A and B^* is induced by the threading schema of A and B^*, where straight lines represent matches by alignment, dashed lines represent matches by insertion.

Algorithm 1:

Data: A, $\mathcal{M}$
Result: a subsequence A' of A that matches the maximum number of symbols of a sequence M obtained by ordering $\mathcal{M}$

```
1 i := 1;
2 A' is an empty sequence;
3 while i ≤ |A| do
4     if α ∈ M with A[i] = α then
5         A' := A' · α;
6         M := M \ {α};
7     i := i + +;
```

Next, we prove the correctness of Algorithm 1.

▶ **Lemma 1.** *Given a sequence A, a multiset $\mathcal{M}$ on Σ, and a substring $A[1,i]$ of A, Algorithm 1 computes a subsequence of $A[1,i]$ that matches the maximum number of symbols of an ordering M of $\mathcal{M}$.*

3 Complexity of $\mathcal{LFCS}$

In this section, we investigate the computational (and approximation) complexity of the $\mathcal{LFCS}$ problem, and we prove that it is APX-hard when A contains at most two occurrences of each symbol in Σ (we denote this restriction of $\mathcal{LFCS}$ by 2-$\mathcal{LFCS}$). We prove the result by an L-reduction from the Maximum Independent Set problem on Cubic Graphs (Max-ISC), which is known to be APX-hard [2](see [5] for details on L-reduction). Max-ISC, given a cubic graph $G = (V, E)$[1], asks for a maximum cardinality subset $V' \subseteq V$ such that given $v_i, v_j \in V'$ it holds $\{v_i, v_j\} \notin E$.

Given a cubic graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots, v_n\}$ and $|E| = m$, in the following we show how to construct an instance $(A, B, \mathcal{M})$ of 2-$\mathcal{LFCS}$. Define an order on the edges incident on a vertex $v_i \in V$ assuming $\{v_i, v_j\} < \{v_i, v_h\}$ if $j < h$. Given a vertex v_i, and the edges $\{v_i, v_j\}, \{v_i, v_h\}, \{v_i, v_z\} \in E$, with $j < h < z$, we say that $\{v_i, v_j\}$ ($\{v_i, v_h\}$, $\{v_i, v_z\}$, respectively) is the first (second, third, respectively) edge incident on v_i.

First, we define the alphabet Σ:

$$\Sigma = \{x_{i,j} : v_i \in V, 1 \leq j \leq 3\} \cup \{y_{i,j} : v_i \in V, 1 \leq j \leq 2\} \cup \{z_{i,j} : 1 \leq i \leq n+m-1, 1 \leq j \leq 4\}.$$

[1] We recall that a cubic graph is an undirected graph where each vertex has degree exactly three.

The input sequences A and B are built by concatenating several substrings.

For each $v_i \in V$, we define the following substrings of the input sequences A, B:

$$A(v_i) = y_{i,1}y_{i,2}x_{i,1}x_{i,2}x_{i,3} \qquad B(v_i) = x_{i,1}x_{i,2}x_{i,3}y_{i,1}y_{i,2}\,.$$

For each $\{v_i, v_j\} \in E$, with $i < j$ (which is the p-th edge, $1 \leq p \leq 3$, incident on v_i and the q-th edge, $1 \leq q \leq 3$, incident on v_j), define the following substrings of A, B:

$$A(\{v_i, v_j\}) = x_{i,p}x_{j,q} \qquad B(\{v_i, v_j\}) = x_{j,q}x_{i,p}\,.$$

Finally, define $2(n + m - 1)$ additional substrings $S_{A,1}, S_{A,2}, \ldots, S_{A,m+n-1}$, $S_{B,1}, S_{B,2}, \ldots,$ $S_{B,m+n-1}$ where $S_{A,i}$, $S_{B,i}$, with $1 \leq i \leq m + n - 1$, are defined as follows:

$$S_{A,i} = S_{B,i} = z_{i,1}z_{i,2}z_{i,3}z_{i,4}.$$

Now, we are able to define the input sequences A and B, by concatenating the substrings previously defined, where substrings associated with edges of G are concatenated assuming some edge ordering (we assume that $\{v_1, v_w\}$ is the first edge, while $\{v_r, v_t\}$ is the last edge according to the ordering):

$$A = A(v_1) \cdot S_{A,1} \cdot A(v_2) \cdot \cdots \cdot S_{A,n-1} \cdot A(v_n) \cdot S_{A,n} \cdot A(\{v_1, v_w\}) \cdot \cdots \cdot S_{A,n+m-1} \cdot A(\{v_r, v_t\})\,,$$

$$B = B(v_1) \cdot S_{B,1} \cdot B(v_2) \cdot \cdots \cdot S_{B,n-1} \cdot B(v_n) \cdot S_{B,n}B(\{v_1, v_w\}) \cdot \cdots \cdot S_{B,n+m-1} \cdot B(\{v_r, v_t\})\,.$$

Notice that each substring associated with an edge $\{v_i, v_j\}$ appears exactly once in both A and B.

$\mathcal{M}$ (in this case is a set) is defined as follows: $\mathcal{M} = \{x_{i,t} : v_i \in V, 1 \leq t \leq 3\}$.

First, we prove that $(A, B, \mathcal{M})$ is an instance of 2-$\mathcal{LFCS}$, that is we prove that each symbol has at most two occurrences in A.

▶ **Lemma 2.** *Each symbol of Σ occurs at most twice in A.*

Proof. Notice that each symbol appearing in a substring $S_{A,i}$, $1 \leq i \leq m + n - 1$, does not appear in any other subsequence of A. Now, consider a symbol $y_{i,t}$, $1 \leq i \leq n$ and $1 \leq t \leq 2$, appearing in substring $A(v_i)$; $y_{i,t}$ does not appear in any other substring of A. Finally, consider a symbol $x_{i,t}$, $1 \leq i \leq n$ and $1 \leq t \leq 3$; $x_{i,t}$ has one occurrence in exactly two subsequences of A: subsequence $A(v_i)$ and subsequence $A(\{v_i, v_j\})$ (where $\{v_i, v_j\}$ is the t-th edges incident on v_i). ◀

Let B^* be a solution of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$. We denote by $S_{B^*,i}$ ($B^*(v_i)$, $B^*(\{v_i, v_j\})$, respectively), the substring of a solution B^* corresponding (after some insertion) to the substring $S_{B,i}$ ($B(v_i)$,$B(\{v_i, v_j\})$, respectively), of B.

Next, we show that we can assume that in a solution B^* of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$, a longest common subsequence of A and B^* matches by alignment a position of a subsequence $S_{A,i}$, $1 \leq i \leq m + n - 1$, only with a position of $S_{B^*,i}$, $1 \leq i \leq m + n - 1$.

▶ **Lemma 3.** *Given a cubic graph G, let $(A, B, \mathcal{M})$ be the corresponding instance of 2-$\mathcal{LFCS}$, and B^* a solution of 2-$\mathcal{LFCS}$ over $(A, B, \mathcal{M})$. Then a longest common subsequence of A and B^* contains each symbol $z_{t,q}$, with $1 \leq t \leq m + n - 1$ and $1 \leq q \leq 4$.*

Proof. Consider a solution B^* of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ and assume that it does not contain a symbol $z_{t,q}$, with $1 \leq t \leq m + n - 1$ and $1 \leq q \leq 4$. By construction a longest common subsequence of B^* and A matches by alignment a position of $A(v_i)$ either with a position of $B(v_i)$ or with a position of $B(\{v_i, v_j\})$.

First, we prove that a longest common subsequence between A and B^* matches by alignment a position of $A(v_i)$ only with a position of $B(v_i)$. Assume that i is the minimum value such that a longest common subsequence S of A and B^* matches by alignment a position of $A(v_i)$ and a position of $B^*(\{v_i, v_j\})$. Notice that, by construction of $(A, B, \mathcal{M})$, no position of $S_{A,i}$ can be matched. Now, starting from S we can compute a common subsequence S' of A and B^*, with $|S'| > |S|$, by modifying the alignment of S as follows: (i) match by alignment the positions of $A(v_i)$ and the positions of $B^*(v_i)$ containing symbols $y_{i,1}, y_{i,2}$; (ii) match by alignment the positions of subsequences $S_{A,i}$ containing symbol $z_{i,q}$ with position of subsequences $S_{B,i}$ containing symbol $z_{i,q}$; (iii) any other match is not modified. It follows that the number of positions in $A(v_i)$ matched by S' with respect to S is decreased by at most three, since eventually positions of $A(V_i)$ containing symbols $x_{i,1}$, $x_{i,2}$, $x_{i,3}$ will not be matched. The number of positions in $S_{A,i}$ matched by S' with respect to S is increased by at least 4, since each position of $S_{A,i}$ is not matched by S and it is matched by S'. By iterating this procedure, we eventually find a longest common subsequence S' of A and B^* where if each position of $A(V_i)$ is matched by alignment, then it is matched with a position of $B(v_i)$. By the maximality of S', this implies that each position of A containing a symbol $z_{t,q}$, with $1 \leq t \leq m + n - 1$ and $1 \leq q \leq 4$, matches a position of B^* containing symbol $z_{t,q}$. ◀

Consider a vertex $v_i \in V$ and the corresponding substrings $A(v_i)$, $B(v_i)$ of A and B. Moreover, let $\{v_i, v_j\}, \{v_i, v_h\}, \{v_i, v_z\} \in E$ be the three edges of G incident on v_i and consider the corresponding substrings $A(\{v_i, v_j\})$, $A(\{v_i, v_h\})$, $A(\{v_i, v_z\})$ ($B(\{v_i, v_j\})$, $B(\{v_i, v_h\})$, $B(\{v_i, v_z\})$, respectively), of A (of B, respectively). Informally, the reduction shows that there are essentially two possible configurations (called *I-configuration* and *C-configuration*) of the substring $B^*(v_i)$ (and possibly $B^*(\{v_i, v_j\})$, $B^*(\{v_i, v_h\})$ and $B^*(\{v_i, v_z\})$) of a filling B^* of B. A substring $B^*(v_i)$ having an I-configuration is related to the vertex v_i in an independent set of G, while a substring $B^*(v_i)$ having a C-configuration is related to the vertex v_i in a vertex cover of G.

We define now the two possible configurations, called *I-configuration* and *C-configuration*, for $B^*(v_i)$ and, possibly, for the substrings $B^*(\{v_i, v_j\})$, $B^*(\{v_i, v_h\})$ and $B^*(\{v_i, v_z\})$ of a filling B^* of B. An *I-configuration* for the substrings $B^*(v_i)$, $B^*(\{v_i, v_j\})$, $B^*(\{v_i, v_h\})$ and $B^*(\{v_i, v_z\})$ is defined as follows:

- $B^*(v_i) = B(v_i)$ (hence there is no insertion in $B(v_i)$).
- For each $\{v_i, v_t\}$, with $t \in \{j, h, z\}$, where $\{v_i, v_t\}$ is the p-th edge incident on v_i, $1 \leq p \leq 3$, and the q-th edge incident on v_t, $1 \leq q \leq 3$, $B^*(\{v_i, v_t\}) = x_{i,p}x_{j,q}x_{i,p}$ (hence $x_{i,p}$ is inserted in $B(\{v_i, v_t\})$).

If $B^*(v_i)$, $B^*(\{v_i, v_j\})$, $B^*(\{v_i, v_h\})$, $B^*(\{v_i, v_z\})$ have an *I-configuration*, a longest common subsequence of $B^*(v_i)$ and $A(v_i)$ has length three (it matches the positions containing $x_{i,1}$, $x_{i,2}$, $x_{i,3}$), and a longest common subsequence of $A(\{v_i, v_t\})$ and $B^*(\{v_i, v_t\})$, with $t \in \{j, h, z\}$, has length two (it matches the positions containing $x_{i,p}$, $x_{j,q}$).

A *C-configuration* for the substring $B^*(v_i)$ is defined as follows:

- $B^*(v_i) = x_{i,1}x_{i,2}x_{i,3}y_{i,1}y_{i,2}x_{i,1}x_{i,2}x_{i,3}$ (hence $B^*(v_i) = B(v_i) \cdot x_{i,1}x_{i,2}x_{i,3}$).

If $B^*(v_i)$ has a *C-configuration*, a longest common subsequence of $B^*(v_i)$ and $A(v_i)$ has length five, it matches the positions containing $y_{i,1}$, $y_{i,2}$, $x_{i,1}$, $x_{i,2}$, $x_{i,3}$.

Next, we present the main lemmata of this section.

▶ **Lemma 4.** *Let G be a cubic graph, instance of Max-ISC, and let $(A, B, \mathcal{M})$ be the corresponding instance of 2-$\mathcal{LFCS}$. Then, given an independent set I of G, we can compute*

in polynomial time a solution B^* of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ inducing a longest common subsequence with A of length $4(m+n-1)+6|I|+5(n-|I|)+m$.

Proof. Consider an independent set I and define a solution B^* of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ as follows. For each $v_i \in I$, with $\{v_i, v_j\}, \{v_i, v_h\}, \{v_i, v_z\} \in E$ the three edges of G incident on v_i, define an *I-configuration* for $B^*(v_i)$, $B^*(\{v_i, v_j\})$, $B^*(\{v_i, v_h\})$, $B^*(\{v_i, v_z\})$. For each $v_i \in V \setminus I$, define a *C-configuration* for $B^*(v_i)$. For each edge $\{v_i, v_j\} \in E$ if $v_i, v_j \in V \setminus I$, then $B^*(\{v_i, v_j\}) = B(\{v_i, v_j\})$; notice that in this case a longest common subsequence of $A(\{v_i, v_j\})$ and $B^*(\{v_i, v_j\})$ has length one, as it matches exactly one position containing either $x_{i,p}$ or $x_{j,q}$. Finally, each position of A in the substring $S_{A,i}$, with $1 \leq i \leq m+n-1$, is matched by alignment with the corresponding position of $S_{B^*,i}$.

Notice that the solution B^* is well-defined, as each $B^*(\{v_i, v_j\})$, with $\{v_i, v_j\} \in E$, can belong to an *I-configuration* of at most one of $B^*(v_i)$ and $B^*(v_j)$, since at most one of v_i, v_j belongs to I.

Now, consider a longest common subsequence S of A and B^*. S matches $4(m+n-1)$ positions in substrings $S_{A,1}, \ldots, S_{A,m+n-1}$, since all the positions of these substrings are matched and, by construction, the overall length of $S_{A,1}, \ldots, S_{A,m+n-1}$ is $4(m+n-1)$. Moreover, by definition of *I-configuration* and *C-configuration*, for each $v_i \in I$, S matches 3 positions of $A(v_i)$ and 2 positions of each $A(\{v_i, v_j\})$, with $\{v_i, v_j\} \in E$; for each $v_i \in V \setminus I$, S matches 5 positions of $A(v_i)$; for each $\{v_i, v_j\} \in E$, with $v_i, v_j \in V \setminus I$, S matches one position of $A(\{v_i, v_t\})$. Hence, S matches $4(m+n-1)+6|I|+5(n-|I|)+m$ positions of A and B^*. ◀

Based on Lemma 3, we can prove the following result.

▶ **Lemma 5.** *Let G be a cubic graph, instance of Max-ISC, and let $(A, B, \mathcal{M})$ be the corresponding instance of 2-$\mathcal{LFCS}$. Then, given a solution B^* of 2-$\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ of length $4(m+n-1)+6p+5(n-p)+m$, we can compute in polynomial time an independent set of G of size at least p.*

By Lemmata 4 and 5, and by the APX-hardness of Max-ISC [2] we can conclude that the 2-$\mathcal{LFCS}$ problem is APX-hard.

▶ **Theorem 6.** *2-$\mathcal{LFCS}$ is APX-hard.*

4 Approximating $\mathcal{LFCS}$

In this section we give a polynomial-time approximation algorithm for $\mathcal{LFCS}$ of factor $\frac{3}{5}$. The approximation algorithm picks the largest number of matched positions returned by two polynomial-time algorithms, Approx-Algorithm-1 and Approx-Algorithm-2. Notice that each algorithm does not return a filling of B with $\mathcal{M}$, but two disjoint subsets of positions of A that have to be matched by alignment and by insertion, respectively, by a subsequence of A and of a filling of B with $\mathcal{M}$. We can easily compute in polynomial time a filling B^* of B with $\mathcal{M}$ so that there exists a common subsequence of A and B^* that matches these two subsets of positions.

Both algorithms consist of two phases.

Approx-Algorithm-1. In the first phase, Approx-Algorithm-1 computes in polynomial time a longest common subsequence of A and B. Denote by $R_{1,a}$ the positions of A matched by alignment in the first phase and by A' the subsequence of A obtained by removing the positions of $R_{1,a}$. The second phase greedily computes in polynomial time a set $R_{1,i}$ of

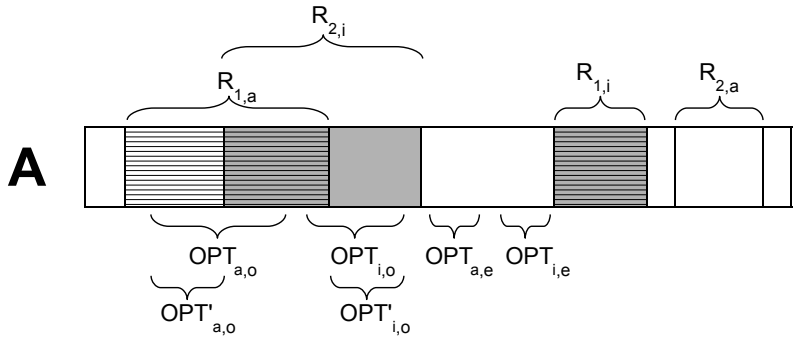

Figure 3 The input sequence A and the positions matched by solution R_1 (dashed) and by solution R_2 (in grey). In the upper part, brackets represent the subsets $R_{1,a}$ and $R_{1,i}$ of R_1, and $R_{2,a}$ and $R_{2,i}$ of R_2. In the lower part, the brackets represent the positions matched by OPT.

positions of A' of maximum size that matches $\mathcal{M}$ by insertion, applying Algorithm 1 on $(A', \mathcal{M})$. Denote by $R_1 = R_{1,a} \cup R_{1,i}$ the set of positions returned by Approx-Algorithm-1.

Approx-Algorithm-2. In the first phase, Approx-Algorithm-2 computes a subset $R_{2,i}$ of positions of A of maximum size that matches $\mathcal{M}$ by insertion applying Algorithm 1 on $(A, \mathcal{M})$. Denote by A'' the subsequence of A obtained by removing the positions of $R_{2,i}$. The second phase computes a longest common subsequence of B and A''; denote by $R_{2,a}$ the set of positions of A'' (and A) matched by this phase. Denote by $R_2 = R_{2,a} \cup R_{2,i}$ the set of positions returned by Approx-Algorithm-2.

Next, we show that the maximum number of positions matched by one of Approx-Algorithm-1 and Approx-Algorithm-2 gives a $\frac{3}{5}$-approximated solution. First, we introduce some notations (see Fig. 3). Let B^{opt} be an optimal solution of $\mathcal{LFCS}$ on instance $(A, B, \mathcal{M})$, and let OPT be a longest common subsequence of A and B^{opt}. We consider the following sets of positions of OPT. Denote by OPT_a the set of positions of A matched by alignment in OPT and by OPT_i the set of positions of A matched by insertion in OPT. Notice that by construction it holds $OPT_a \cap OPT_i = \emptyset$.

Define $OPT_{a,o} = OPT_a \cap (R_{1,a} \cup R_{2,i})$ and $OPT_{i,o} = OPT_i \cap (R_{1,a} \cup R_{2,i})$. Moreover, define $OPT_{a,e} = OPT_a \setminus OPT_{a,o}$ and $OPT_{i,e} = OPT_i \setminus OPT_{i,o}$. Informally, $OPT_{a,e}$ ($OPT_{i,e}$, respectively) is the set of positions of A matched by alignment (by insertion, respectively) in OPT that is not matched in the first phase by Approx-Algorithm-1 (in the second phase by Approx-Algorithm-2, respectively). Finally, define $OPT'_{i,o} = OPT_{i,o} \setminus R_{1,a}$ and $OPT'_{a,o} = OPT_{a,o} \setminus R_{2,i}$.

By definition of OPT, $OPT_{a,o}$, $OPT_{i,o}$, $OPT_{a,e}$ and $OPT_{i,e}$, it holds $|OPT| = |OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,o}| + |OPT_{i,e}|$.

We will show that the largest set between R_1 and R_2 gives a $\frac{3}{5}$-approximate solution, that is $\max(|R_1|, |R_2|) \geq \frac{3}{5}|OPT|$. We start by showing two bounds on OPT_i and OPT_a.

▶ **Lemma 7.** *$|R_{1,a}| \geq |OPT_a|$ and $|R_{2,i}| \geq |OPT_i|$.*

Proof. First, we prove that $|R_{1,a}| \geq |OPT_a|$. Consider the set of positions in OPT_a. Since each position in OPT_a is a position of A matched by alignment, it follows that the set OPT_a induces a common subsequence of A and B. Since the set $R_{1,a}$ of positions of A induces a longest common subsequence of A and B, it follows that $|R_{1,a}| \geq |OPT_a|$.

Now, we prove that $|R_{2,i}| \geq |OPT_i|$. Consider the set of positions in OPT_i. Each position in OPT_i is matched by insertion, hence it is matched with an inserted symbol of $\mathcal{M}$. By

Lemma 1, $R_{2,i}$ is a set of positions of A of maximum cardinality that can be matched by insertion with symbols of $\mathcal{M}$, hence $|R_{2,i}| \geq |OPT_i|$. ◀

As a consequence of Lemma 7, it follows that $|R_{1,a}| + |R_{2,i}| \geq |OPT_i| + |OPT_a| \geq |OPT|$. Hence the maximum of R_1, R_2 is (at least) $\frac{1}{2}|OPT|$. In the following, we show with a more refined analysis that the maximum of $|R_1|$, $|R_2|$ is at least $\frac{3}{5}|OPT|$.

We prove some bounds on $R_{1,i}$ and $R_{2,a}$, then we consider three cases depending on the values of $OPT_{a,o}$, $OPT_{i,o}$, $OPT_{a,e}$, $OPT_{i,e}$, $OPT'_{i,o}$ and $OPT'_{a,o}$. First, the following result holds.

▶ **Lemma 8.** *$|R_{1,i}| \geq |OPT'_{i,o}| + |OPT_{i,e}|$ and $|R_{2,a}| \geq |OPT'_{a,o}| + |OPT_{a,e}|$.*

Now, in the analysis of the approximation factor of Approx-Algorithm-1 and Approx-Algorithm-2, we consider three cases, depending on the values of $OPT_{i,e}$, $OPT_{i,o}$, $OPT'_{i,o}$.

Case 1

Assume that $|OPT_{i,e}| + |OPT'_{i,o}| \geq \frac{1}{2}|OPT_{i,o}|$, we show the following result.

▶ **Lemma 9.** *Assume that $|OPT_{i,e}| + |OPT'_{i,o}| \geq \frac{1}{2}|OPT_{i,o}|$, then $|R_1| \geq \frac{3}{5}|OPT|$.*

Proof. Since $|R_{1,i}| \geq |OPT'_{i,o}| + |OPT_{i,e}|$ by Lemma 8, it follows that

$$|R_{1,a}| + |R_{1,i}| \geq |R_{1,a}| + |OPT'_{i,o}| + |OPT_{i,e}| \geq$$

$$\frac{3}{5}(|R_{1,a}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT_{i,e}|) + |OPT'_{i,o}|.$$

By Lemma 7 it follows that $|R_{1,a}| \geq |OPT_a|$ and, since $|OPT_a| = |OPT_{a,o}| + |OPT_{a,e}|$, it follows that $|R_{1,a}| \geq |OPT_{a,o}| + |OPT_{a,e}|$, hence

$$\frac{3}{5}(|R_{1,a}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT_{i,e}|) + |OPT'_{i,o}| \geq$$

$$\frac{3}{5}(|OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT_{i,e}|) + |OPT'_{i,o}|.$$

Hence, it holds

$$|R_{1,a}| + |R_{1,i}| \geq \frac{3}{5}(|OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT_{i,e}|) + |OPT'_{i,o}|. \quad (1)$$

Notice that $|R_{1,a}| + |OPT'_{i,o}| \geq |OPT_{i,o}|$, since, by construction, each position in $OPT_{i,o}$ is either in $OPT'_{i,o}$ or in $R_{1,a}$. Then,

$$\frac{2}{5}(|R_{1,a}| + |OPT'_{i,o}|) \geq \frac{2}{5}|OPT_{i,o}|. \quad (2)$$

Since we are assuming that $|OPT_{i,e}| + |OPT'_{i,o}| \geq \frac{1}{2}|OPT_{i,o}|$, it holds

$$\frac{2}{5}(|OPT_{i,e}| + |OPT'_{i,o}|) \geq \frac{1}{5}|OPT_{i,o}|. \quad (3)$$

Combining Inequalities 2 and 3 with Inequality 1, we can conclude that, under the hypothesis $|OPT_{i,e}| + |OPT'_{i,o}| \geq \frac{1}{2}|OPT_{i,o}|$, it holds

$$|R_{1,a}| + |R_{1,i}| \geq \frac{3}{5}(|OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT_{i,e}|) + |OPT'_{i,o}| \geq$$

$$\frac{3}{5}(|OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,e}|) + \frac{2}{5}(|R_{1,a}| + |OPT'_{i,o}|) + \frac{2}{5}(|OPT_{i,e}| + |OPT'_{i,o}|) \geq$$

$$\frac{3}{5}(|OPT_{a,o}| + |OPT_{a,e}| + |OPT_{i,o}| + |OPT_{i,e}|).$$

It follows that, under the hypothesis $|OPT_{i,e}| + |OPT'_{i,o}| \geq \frac{1}{2}|OPT_{i,o}|$, it holds $|R_1| \geq \frac{3}{5}|OPT|$. ◀

Case 2

Assume that $|OPT_{a,e}| + |OPT'_{a,o}| \geq \frac{1}{2}|OPT_{a,o}|$. Similarly to Case 1, we can prove the following result.

▶ **Lemma 10.** *Assume that* $|OPT_{a,e}| + |OPT'_{a,o}| \geq \frac{1}{2}|OPT_{a,o}|$*, then* $|R_2| \geq \frac{3}{5}|OPT|$.

Case 3

Assume that both Case 1 and Case 2 do not hold. Then,

$$|OPT_{i,e}| + |OPT'_{i,o}| < \frac{1}{2}|OPT_{i,o}| \text{ and } |OPT_{a,e}| + |OPT'_{a,o}| < \frac{1}{2}|OPT_{a,o}|.$$

Since $|OPT_{i,e}| + |OPT'_{i,o}| < \frac{1}{2}|OPT_{i,o}|$, it follows that $|OPT_{i,e}| < \frac{1}{2}|OPT_{i,o}|$ and, since $|OPT_{a,e}| + |OPT'_{a,o}| < \frac{1}{2}|OPT_{a,o}|$, it follows that $|OPT_{a,e}| < \frac{1}{2}|OPT_{a,o}|$. But then, since $|OPT| = |OPT_{a,o}| + |OPT_{i,o}| + |OPT_{a,e}| + |OPT_{i,e}|$, it follows that

$$|OPT| \leq \frac{3}{2}(|OPT_{a,o}| + |OPT_{i,o}|)$$

We show that $|R_1| \geq |OPT_{a,o}| + |OPT_{i,o}|$, thus implying that $|R_1| \geq \frac{3}{5}|OPT|$.

▶ **Lemma 11.** $|R_{1,a} \cup R_{1,i}| \geq |OPT_{a,o}| + |OPT_{i,o}|$.

By Lemma 11, $|R_{1,a} \cup R_{1,i}| \geq |OPT_{a,o}| + |OPT_{i,o}|$. Since in this case we have shown that $|OPT| \leq \frac{3}{2}(|OPT_{a,o}| + |OPT_{i,o}|)$, it follows that $|R_1| = |R_{1,a} \cup R_{1,i}| \geq \frac{2}{3}|OPT| \geq \frac{3}{5}|OPT|$. From Lemma 9, Lemma 10 and Lemma 11, it follows the main result of this section.

▶ **Theorem 12.** *Given an instance* $(A, B, \mathcal{M})$ *of* $\mathcal{LFCS}$*, the largest solution returned by Approx-Algorithm-1 and Approx-Algorithm-2 is an approximate solution of factor* $\frac{3}{5}$.

Proof. From Lemma 9, Lemma 10 and Lemma 11, it follows that $\max(|R_1|, R_2|) \geq \frac{3}{5}|OPT|$.

We can compute a filling B_1 of B with $\mathcal{M}$ that matches at least $|R_1|$ positions with A as follows: we consider the positions in $R_{1,a}$ as matched by alignment, we insert symbols of $\mathcal{M}$ in B in order to match by insertion the positions in $R_{1,i}$. It follows that a longest common subsequence of A and B_1 matches at least $|R_1|$ positions.

Similarly, we can compute a filling B_2 of B with $\mathcal{M}$ that matches at least $|R_2|$ positions of A. We insert symbols of $\mathcal{M}$ in B so that the positions in $R_{1,i}$ are matched by insertion. Consider the subsequence A'' obtained after the removal of positions in $R_{1,i}$; a longest common subsequence of A'' and B matches at least $|R_{2,a}|$ positions. It follows that a longest common subsequence of A and B_2 matches at least $|R_2|$ positions. ◀

5 An FPT Algorithm

In this section, we present an FPT algorithm for $\mathcal{LFCS}$ parameterized by the number k of positions of A matched by insertions. Notice that $k < |\mathcal{M}|$. Here we assume that the input sequences A and B have been extended by adding two symbols $\$_A, \$_B \notin \Sigma$, respectively, in position 0 of A and B, respectively. Hence we assume that position 0 of A and of a filling B^* of B with $\mathcal{M}$ is not matched by alignment or by insertion by any solution of $\mathcal{LFCS}$ of length greater than zero.

The algorithm we present is based on the color-coding technique [3]. Next, we present the definition of perfect families of hash functions for a multiset of symbols, on which our color-coding approach is based.

▶ **Definition 13.** Let $\mathcal{M}$ be a multiset of positions and let F be a family of hash functions from $\mathcal{M}$ to a set $\{c_1, \ldots, c_k\}$ of colors. F is called *perfect* if for any subset $W \subseteq \mathcal{M}$, such that $|W| = k$, there exists a function $f \in F$ which is injective on W.

A perfect family F of hash functions from $\mathcal{M}$ to $\{c_1, \ldots, c_k\}$, having size $O(\log |\mathcal{M}| 2^{O(k)})$, can be constructed in time $O(2^{O(k)}|\mathcal{M}| \log |\mathcal{M}|)$ (see [3]).

Consider a perfect family of hash functions $F : \mathcal{M} \to \{c_1, \ldots, c_k\}$. Let $f \in F$ be an injective function, and define $L[i, j, C, l]$, with $C \subseteq \{c_1, \ldots, c_k\}$, $0 \leq i, l \leq |A|$ and $0 \leq j \leq |B|$, as follows:

- $L[i, j, C, l] = 1$ if and only if there exists a common subsequence of $A[0, i]$ and of a filling B^* of $B[0, j]$ with $\mathcal{M}$ having length l, such that there exist $|C|$ symbols of $\mathcal{M}$ inserted in $B[0, j]$, each one associated with a distinct color of C and matched by insertion with a position of A
- else $L[i, j, C, l] = 0$.

Next, we define the recurrence to compute $L[i, j, C, l]$, where $i \geq 1$ and $j \geq 0$.

$$L[i, j, C, l] = \max \begin{cases} L[i-1, j, C, l] & \\ L[i, j-1, C, l] & \text{if } j \geq 1 \\ L[i-1, j-1, C, l-1] & \text{if } A[i] = B[j] \text{ and } j \geq 1 \\ L[i-1, j, C \setminus \{c\}, l-1] & \text{if } A[i] = \alpha \text{ and there exists} \\ & \alpha \in \mathcal{M} \text{ with } f(\alpha) = c \in C \end{cases} \tag{4}$$

For the base case, since we have extended A and B so that position 0 in A and in the filling of B cannot be matched by insertions or by alignment, it holds $L[0, 0, C, l] = 1$, if $C = \emptyset$ and $l = 0$, else $L[0, 0, C, l] = 0$. Next, we prove the correctness of the recurrence.

▶ **Lemma 14.** *Let $F : \mathcal{M} \to \{c_1, \ldots, c_k\}$ be a perfect family of hash functions, let $f \in F$ be an injective function and let C be a subset of $\{c_1, \ldots, c_k\}$. Then there exists a common subsequence of length l, $l \geq 0$, of $A[0, i]$, $0 \leq i \leq |A|$, and of a filling of $B[0, j]$, $0 \leq j \leq |B|$, with $\mathcal{M}' \subseteq \mathcal{M}$ such that each symbol of $\mathcal{M}'$ matched by insertion is associated with a distinct color in C if and only if $L[i, j, C, l] = 1$.*

Now, we are able to prove the main result of this section.

▶ **Theorem 15.** *Let A, B be two sequences and $\mathcal{M}$ a mutliset of symbols. Then it is possible to compute in time $2^{O(k)} poly(|A| + |B| + |\mathcal{M}|)$ if there exists a solution B^* of $\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ such that a longest common subsequence S of A and B^* has length l and it matches by insertion k positions of A.*

Proof. The correctness of the algorithm follows from Lemma 14 and from the fact that entry $L[|A|, |B|, C, l] = 1$ if and only if there exists a solution of $\mathcal{LFCS}$ over instance $(A, B, \mathcal{M})$ having length l that matches by insertion k positions of A.

Next, we consider the time complexity of the algorithm. A perfect family of hash functions that color-codes the symbols of $\mathcal{M}$ can be computed in time $2^{O(k)}poly(|\mathcal{M}|)$. Then, the algorithm iterates through $2^{O(k)}poly(|\mathcal{M}|)$ color-codings. For each color-coding, the table $L[i, j, C, l]$ is computed in time $O(2^k|A|^2|B|k)$ (where $l \leq |A|$), since for each of the at most $O(2^k|A|^2|B|)$ entries we need to look for at most k possible entries. The overall complexity is then $2^{O(k)}poly(|A| + |B| + |\mathcal{M}|)$. ◀

6 Conclusion

We have introduced a variant of the LCS problem, called Longest Filled Common Subsequence ($\mathcal{LFCS}$), to compare a sequence A with an incomplete sequence B to be filled with a multiset $\mathcal{M}$ of symbols. We have shown that the problem is APX-hard (hence NP-hard), even when each symbol occurs at most twice in the input sequence A. Then, we have given an approximation algorithm of factor $\frac{3}{5}$ and a fixed-parameter algorithm, where the parameter is the number of symbols in $\mathcal{M}$ matched by insertion.

There are some interesting open problems related to $\mathcal{LFCS}$. It would be interesting to extend $\mathcal{LFCS}$ to the comparison of two incomplete sequences, similar to what has been done for Scaffold Filling [15]. Moreover, it would be interesting to design more efficient parameterized algorithms for $\mathcal{LFCS}$, for example by considering the algebraic technique used for the repetition-free longest common subsequence [6]. Another open problem is whether $\mathcal{LFCS}$ is NP-hard on a constant size alphabet.

References

1 Said Sadique Adi, Marília D. V. Braga, Cristina G. Fernandes, Carlos Eduardo Ferreira, Fábio Viduani Martinez, Marie-France Sagot, Marco A. Stefanes, Christian Tjandraatmadja, and Yoshiko Wakabayashi. Repetition-free longest common subsequence. *Discrete Appl. Math.*, 158(12):1315–1324, 2010. doi:10.1016/j.dam.2009.04.023.

2 Paola Alimonti and Viggo Kann. Some apx-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.

3 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

4 Abdullah N. Arslan and Ömer Egecioglu. Algorithms for the constrained longest common subsequence problems. *Int. J. Found. Comput. Sci.*, 16(6):1099–1109, 2005. doi:10.1142/S0129054105003674.

5 Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Heidelberg, 1999. doi:10.1007/978-3-642-58412-1.

6 Guillaume Blin, Paola Bonizzoni, Riccardo Dondi, and Florian Sikora. On the parameterized complexity of the repetition free longest common subsequence problem. *Inf. Process. Lett.*, 112(7):272–276, 2012. doi:10.1016/j.ipl.2011.12.009.

7 Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, Guillaume Fertin, Raffaella Rizzi, and Stéphane Vialette. Exemplar longest common subsequence. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 4(4):535–543, 2007. doi:10.1145/1322075.1322078.

8 Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Yuri Pirola. Variants of constrained longest common subsequence. *Inf. Process. Lett.*, 110(20):877–881, 2010. doi:10.1016/j.ipl.2010.07.015.

9 Laurent Bulteau, Anna Paola Carrieri, and Riccardo Dondi. Fixed-parameter algorithms for scaffold filling. *Theor. Comput. Sci.*, 568:72–83, 2015. doi:10.1016/j.tcs.2014.12.005.

10 P. G. S. Chain and et al. Genomics. Genome project standards in a new era of sequencing. *Science*, 326:236–237, 2009. doi:10.1126/SCIENCE.1180614.

11 Francis Y. L. Chin, Alfredo De Santis, Anna Lisa Ferrara, N. L. Ho, and S. K. Kim. A simple algorithm for the constrained sequence problems. *Inf. Process. Lett.*, 90(4):175–179, 2004. doi:10.1016/j.ipl.2004.02.008.

12 Carlos Eduardo Ferreira and Christian Tjandraatmadja. A branch-and-cut approach to the repetition-free longest common subsequence problem. *Electron. Notes Discrete Math.*, 36:527–534, 2010. doi:10.1016/j.endm.2010.05.067.

13 Zvi Gotthilf, Danny Hermelin, and Moshe Lewenstein. Constrained LCS: hardness and approximation. In Paolo Ferragina and Gad M. Landau, editors, *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, volume 5029 of *LNCS*, pages 255–262. Springer, 2008. doi:10.1007/978-3-540-69068-9_24.

14 Tao Jiang and Ming Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, 24(5):1122–1139, 1995. doi:10.1137/S009753979223842X.

15 Nan Liu, Haitao Jiang, Daming Zhu, and Binhai Zhu. An improved approximation algorithm for scaffold filling to maximize the common adjacencies. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 10(4):905–913, 2013. doi:10.1109/TCBB.2013.100.

16 Adriana Muñoz, Chunfang Zheng, Qian Zhu, Victor A. Albert, Steve Rounsley, and David Sankoff. Scaffold filling, contig fusion and comparative gene order inference. *BMC Bioinformatics*, 11:304, 2010. doi:10.1186/1471-2105-11-304.

17 Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147(1):195–197, 1981. doi:10.1016/0022-2836(81)90087-5.

18 Yin-Te Tsai. The constrained longest common subsequence problem. *Inf. Process. Lett.*, 88(4):173–176, 2003. doi:10.1016/j.ipl.2003.07.001.

19 Binhai Zhu. Genomic scaffold filling: A progress report. In Daming Zhu and Sergey Bereg, editors, *Proceedings of the 10th International Workshop on Frontiers in Algorithmics (FAW 2016)*, volume 9711 of *LNCS*, pages 8–16. Springer, 2016. doi:10.1007/978-3-319-39817-4_2.