Journal of Discrete Algorithms 21 (2013) 11–17

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

The constrained shortest common supersequence problem

Riccardo Dondi [1]

Dipartimento di Scienze Umane e Sociali, Università degli Studi di Bergamo, Via Donizetti 3, 24129 Bergamo, Italy

ARTICLE INFO

Article history:
Received 26 October 2011
Accepted 7 March 2013
Available online 21 March 2013

Keywords:
Shortest common supersequence
Approximation algorithms
Computational complexity
Fixed-parameter algorithms

ABSTRACT

Shortest common supersequence and longest common subsequence are two widely used measures to compare sequences in different fields, from AI planning to Bioinformatics. Inspired by recently proposed variants of these two measures, we introduce a new version of the shortest common supersequence problem, where the solution is required to satisfy a given constraint on the number of occurrences of each symbol. First, we investigate the computational and approximation complexity of the problem, then we give a $\frac{3}{2}$-approximation algorithm. Finally, we investigate the parameterized complexity of the problem, and we present a fixed-parameter algorithm.

1. Introduction

Longest Common Subsequence (LCS) and Shortest Common Supersequence (SCS) have been widely used as measures to compare strings in different fields, from AI planning to Bioinformatics. In AI planning [20,23], it is important to analyze how different parts of plans interact and to integrate such parts into a global efficient planning. Since the order in which the tasks of each plan are scheduled is relevant, and we want to minimize the overall length of the planning, the integration can be done by computing an SCS of a set of strings representing partial plans.

Another illustrative example is the comparison of genomes in Bioinformatics [14]. Genomes are usually viewed as strings, where each symbol represents a gene. The comparison of the strings associated with genomes provides a measure of their similarities and differences. Since the order in which genes appear in the genomes seems to be relevant when measuring their similarities, LCS and SCS are two promising approaches to compare genomes.

In the last years, different variants of longest common subsequence and shortest common supersequence have been proposed [5,1,6,11,22,7,3,13]. These versions of LCS and SCS usually impose a constraint on the common subsequence/supersequence to be computed. For example, the variants of longest common subsequence proposed in [5,1,6], impose a constraint on the number of occurrences of a symbol in the common subsequence of the input strings. The use of a constraint is motivated by some assumptions, such as the limitation of resources in AI planning [8,12] or such as the exemplar hypothesis [21], that aims to identify the original copy of a gene from which all other copies, through duplication, have been originated.

In this paper we focus on a variant of the SCS problem, where we impose some additional constraint on the common supersequence to be computed. The SCS problem is known to be solvable in polynomial time when the input consists of a constant number of strings [9], while it is NP-hard on an arbitrary number of strings [15], even when the strings are over a binary alphabet [19]. The SCS problem has been widely applied as a measure of similarity in Bioinformatics [17,18,8,11,12] and in AI planning [8,12]. Here, we introduce a new variant of the SCS problem, called *Constrained Shortest Common Supersequence* ($\mathcal{C}-\mathcal{SCS}$), where given two strings s_1, s_2 over alphabet Σ and a constraint T_L, which is a lower bound on the number of occurrences of any symbol in a feasible solution, we ask for a shortest common supersequence s that satisfies

E-mail address: riccardo.dondi@unibg.it.

[1] Partly supported by FAR 2011 grant "Algoritmi per il trattamento di sequenze".

1570-8667/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jda.2013.03.004

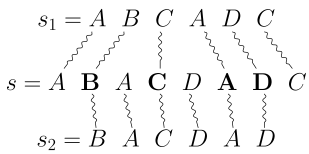

Fig. 1. A common supersequence $s = ABACDADC$ of the strings $s_1 = ABCADC$ and $s_2 = BACDAD$, and the corresponding threading schema. Notice that the matchings are in bold.

the constraint T_L. $\mathcal{C} - \mathcal{SCS}$ can be used to model situations in which we have to integrate some plans, and we have a set of specific demands on some tasks, that is we require that each task must be performed in the global planning at least a given number of times.

In this paper, we study the computational, approximation and parameterized complexity of the $\mathcal{C} - \mathcal{SCS}$ problem. In Section 2, we give the preliminary definition of $\mathcal{C} - \mathcal{SCS}$. In Section 3, we show that the problem is APX-hard, even when $T_L[i] \leqslant 3$, $1 \leqslant i \leqslant |\Sigma|$, and each symbol occurs at most twice in each input string, and we present a polynomial time algorithm when each $T_L[i] \geqslant 2$, $1 \leqslant i \leqslant |\Sigma|$. In Section 4, we give an approximation algorithm of factor $\frac{3}{2}$. Finally, in Section 5, we present a fixed-parameter algorithm for the $\mathcal{C} - \mathcal{SCS}$ problem of time complexity $O^*(1.7321)^k$, where the parameter k is the size of the solution (that is the length of the shortest common supersequence that satisfies the constraint).[2]

2. Preliminaries

In this section we introduce some basic definitions. Let s be a string over alphabet Σ, we denote by $|s|$ the length of s and by $s[i]$ the i-th symbol of s. Given two positions i, j in s, with $1 \leqslant i \leqslant j \leqslant |s|$, we denote by $s[i, j]$ the substring of s that starts at position i and ends at position j. Given a string over alphabet Σ and a symbol $a_j \in \Sigma$, we say that a position i, $1 \leqslant i \leqslant |s|$, is an occurrence of a_j if $s[i] = a_j$; we denote by $occ_s(a_j)$ the number of occurrences of symbol a_j in s. A subsequence of s is a string s' that can be computed by deleting some symbols (possibly none) in s. In this case s is a supersequence of s'.

Consider two strings s_1 and s_2. A common subsequence of s_1 and s_2 is a string s' that is a subsequence of both s_1 and s_2. A longest common subsequence of s_1, s_2 is a common subsequence of s_1 and s_2 having maximum length. A common supersequence of s_1, s_2 is a string s with the property that s_1 and s_2 are both subsequences of s. A shortest common supersequence of s_1, s_2 is a common supersequence of s_1, s_2 having minimum length.

Let s_1, s_2 be two strings, and let s be a common supersequence of s_1 and s_2, then we can define a *threading schema* (see Fig. 1) of s_1 and s_2 with respect to s as follows. The strings s_1, s_2, s are written on three parallel lines, and a set of lines are added, each one connecting an occurrence of an identical symbol in s_i, with $i \in \{1, 2\}$, and in s, such that no two lines cross. A position h in s connected by two lines is called a *matching* and in this case we say that the two positions of s_1 and s_2 connected to h *match* (see Fig. 1). A position of s_j, $1 \leqslant j \leqslant 2$, not connected to a matching, is called an *unmatched* position.

Aiming at the comparison of two strings with an additional constraint on the occurrences of symbols, we introduce the following problem:

Problem 1. $\mathcal{C} - \mathcal{SCS}$.
Input: two strings s_1 and s_2 over alphabet Σ, and a constraint T_L, an array of positive integer values having size $|\Sigma|$.
Output: a shortest common supersequence of s_1 and s_2, containing at least $T_L[i]$ occurrences of symbol $a_i \in \Sigma$, with $1 \leqslant i \leqslant |\Sigma|$.

In what follows we assume that, for each $a_i \in \Sigma$, $\max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} \leqslant T_L[i] \leqslant occ_{s_1}(a_i) + occ_{s_2}(a_i)$, otherwise the constraint on the symbol a_i is called *redundant*. Indeed any common supersequence of two strings s_1, s_2 contains at least $\max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\}$ occurrences of each symbol $a_i \in \Sigma$. Furthermore, we can assume that any common supersequence of s_1, s_2 contains at most $occ_{s_1}(a_i) + occ_{s_2}(a_i)$ occurrences of each symbol $a_i \in \Sigma$.

It is easy to see that a shortest common supersequence of two strings s_1 and s_2 is not necessarily a feasible solution of the $\mathcal{C} - \mathcal{SCS}$ problem. Consider a string $s_1 = a_1a_2a_3$, and a string $s_2 = a_3a_1a_2$ over alphabet $\Sigma = \{a_1, a_2, a_3\}$, and a constraint $T_L = [2, 2, 1]$. It is easy to see that the string $a_3a_1a_2a_3$ is a shortest common subsequence of s_1 and s_2, while it is not a feasible solution of $\mathcal{C} - \mathcal{SCS}$. The string $a_1a_2a_3a_1a_2$ is an optimal solution of $\mathcal{C} - \mathcal{SCS}$, but not a shortest common supersequence of s_1 and s_2.

[2] We recall that in the $O^*()$ notation, the polynomially bounded terms are suppressed.

3. Approximation and computational complexity of $\mathcal{C}-\mathcal{SCS}$

In this section, we investigate the approximation and computational complexity of the $\mathcal{C}-\mathcal{SCS}$ problem, and we establish the tractability of the problem depending on the constraint T_L. First, we prove in Theorem 1 the APX-hardness (and hence the NP-hardness) of the $\mathcal{C}-\mathcal{SCS}$ problem even in the restricted case when each $T_L[i] \leqslant 3$, $1 \leqslant i \leqslant |\Sigma|$, and each symbol occurs at most twice in each input string. Then, we prove in Theorem 4 that the $\mathcal{C}-\mathcal{SCS}$ is polynomially time solvable when each $T_L[i] \leqslant 2$, $1 \leqslant i \leqslant |\Sigma|$.

Theorem 1. *$\mathcal{C}-\mathcal{SCS}$ is APX-hard, even when each symbol occurs at most twice in each input string and $T_L[i] \leqslant 3$, for each i with $1 \leqslant i \leqslant |\Sigma|$.*

Proof. We prove the result by giving an L-reduction from Minimum Vertex Cover on Cubic graphs (MVCC), which is known to be APX-hard [2]. For details on L-reduction, see [4]. We recall that MVCC, given a cubic graph $G=(V,E)$, asks for a minimum cardinality set $V' \subseteq V$, such that for each $\{v_i, v_j\} \in E$, at least one of $v_i, v_j \in V'$. In what follows we denote by n the number of vertices in G, that is $n = |V|$.

Now, starting from a cubic graph $G=(V,E)$, we construct an instance (s_1, s_2, T_L) of $\mathcal{C}-\mathcal{SCS}$. Given a vertex $v_i \in V$, denote by $\{v_i, v_h\}$, $\{v_i, v_l\}$, $\{v_i, v_x\}$ the three edges of E incident on v_i. Define the alphabet Σ as follows: $\Sigma = \{v_{i,1}, v_{i,2}\colon 1 \leqslant i \leqslant n\} \cup \{e_{i,h}\colon \{v_i, v_h\} \in E\} \cup \{b_{i,j}\colon 1 \leqslant i \leqslant n \wedge 1 \leqslant j \leqslant 9\}$.

Each of the two strings s_1, s_2 is defined as the concatenation of n substrings, that is $s_1 = b_1(s_1)b_2(s_1)\dots b_n(s_1)$, and $s_2 = b_1(s_2)b_2(s_2)\dots b_n(s_2)$, where each substring $b_i(s_j)$, $1 \leqslant i \leqslant n$ and $j \in \{1,2\}$, is associated with vertex $v_i \in V$. The substrings $b_i(s_1)$ and $b_i(s_2)$ (associated with v_i) have both length 14 and are defined as follows:

$$b_1(s_i) = b_{i,1}\dots b_{i,9} v_{i,1} v_{i,2} e_{i,h} e_{i,l} e_{i,x},$$

$$b_2(s_i) = b_{i,1}\dots b_{i,9} e_{i,h} e_{i,l} e_{i,x} v_{i,1} v_{i,2}.$$

Notice that the only symbols having more than one occurrence in a string s_j, $1 \leqslant j \leqslant 2$, are those symbols $e_{i,l}$ associated with edges of G, that have one occurrence in the substrings $b_i(s_j)$ and $b_l(s_j)$.

Next, we define the input constraint T_L. For each symbol $b_{i,j}$, with $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant 9$, and for each symbol $v_{i,l}$, with $1 \leqslant i \leqslant n$ and $1 \leqslant l \leqslant 2$, T_L is equal to one. For each symbol $e_{i,j}$ associated with an edge $\{v_i, v_j\} \in E$, T_L is equal to 3.

First, we introduce a property of the two strings s_1, s_2. Let us consider the substrings $b_i(s_1)$ and $b_i(s_2)$, $1 \leqslant i \leqslant n$. Then, the (unique) shortest common supersequence of $b_i(s_1)$ and $b_i(s_2)$ is the following string: $s_i^* = b_{i,1}\dots b_{i,9} v_{i,1} v_{i,2} e_{i,h} e_{i,l} e_{i,v} v_{i,1} v_{i,2}$. Notice that $|s_i^*| = 16$. Define the string $s_i^+ = b_{i,1}\dots b_{i,9} e_{i,h} e_{i,l} e_{i,x} v_{i,1} v_{i,2} e_{i,h} e_{i,l} e_{i,x}$. Notice that s_i^+ is a supersequence (but not a shortest supersequence) of $b_i(s_1)$ and $b_i(s_2)$, that s_i^+ contains two occurrences of $e_{i,h}$, $e_{i,l}$, $e_{i,x}$, and that $|s_i^+| = 17$.

Now, in Claim 2 and in Claim 3, we prove the main properties of the reduction.

Claim 2. *Let $G=(V,E)$ be a cubic graph and let (s_1, s_2, T_L) be the corresponding instance of $\mathcal{C}-\mathcal{SCS}$. Then, given a vertex cover of G having size p, we can compute in polynomial time a solution of $\mathcal{C}-\mathcal{SCS}$ of length $9n + 8p + 7(n-p)$.*

Proof. Consider a vertex cover $V' \subseteq V$, with $|V'| = p$, and define a solution s of $\mathcal{C}-\mathcal{SCS}$ as follows. The string s is defined as the concatenation of the n strings $s_1 s_2 \dots s_n$, where each s_i, $1 \leqslant i \leqslant n$, is either equal to s_i^* or s_i^+. For each $v_i \in V'$, $s_i = s_i^+$, while for each $v_i \in V \setminus V'$, $s_i = s_i^*$. Since $|s_i^*| = 16$ and $|s_i^+| = 17$, it follows that s has length $9n + 8p + 7(n-p)$. Moreover, since V' is a vertex cover of G, for each symbol $e_{i,h}$ associated with the edge $\{v_i, v_h\} \in E$, s_i is equal to s_i^+ or s_h is equal to s_h^+, hence the constraint T_L for $e_{i,j}$ is satisfied. Notice that the constraint for symbols b_i and v_j is trivially satisfied. □

Claim 3. *Let $G=(V,E)$ be a cubic graph and let (s_1, s_2, T_L) be the corresponding instance of $\mathcal{C}-\mathcal{SCS}$. Then, given a solution of $\mathcal{C}-\mathcal{SCS}$ of length $9n + 8p + 7(n-p)$, we can compute in polynomial time a cover of the graph G of size p.*

Proof. Consider a solution s of $\mathcal{C}-\mathcal{SCS}$ of size $9n + 8p + 7(n-p)$. We can assume that s contains exactly one substring $b_{i,1}\dots b_{i,9}$, $1 \leqslant i \leqslant n$, and that each symbol in $b_{i,h}$, $1 \leqslant i \leqslant n$ and $1 \leqslant h \leqslant 9$, occurs exactly once in s. Indeed, by construction each of the symbols in $\{b_{i,1}, \dots, b_{i,9}\}$ occurs exactly once in s_j, $j \in \{1,2\}$, and more precisely, each of the symbols in $\{b_{i,1}, \dots, b_{i,9}\}$ belongs to the substring $b_{i,1}\dots b_{i,9}$ of s_j. Hence we can assume that s contains at most two occurrences of the substring $b_{i,1}\dots b_{i,9}$ and no other occurrence of a symbol in $\{b_{i,1}, \dots, b_{i,9}\}$ belongs to s.

Now, let i be the minimum index such that s contains two occurrences of $b_{i,1}\dots b_{i,9}$. If $i = 1$, then it is easy to see that by construction we can remove from s the rightmost occurrences of $b_{1,1}\dots b_{1,9}$. Assume that $i > 1$. The symbols $b_{i-1,1}\dots b_{i-1,9}$ occur exactly once in s in positions $h, \dots, h+8$ respectively, with $1 \leqslant h \leqslant |s| - 7$. It follows that the threading schema of s_j, $1 \leqslant j \leqslant 2$, with respect to s, maps the only occurrence of $b_{i-1,1}\dots b_{i-1,9}$ in s_j in positions $h, \dots, h+8$. Then (1) insert the string $s_a = e_{i-1,h} e_{i-1,l} e_{i-1,x} v_{i-1,1} v_{i-1,2} e_{i-1,h} e_{i-1,l} e_{i-1,x}$ in position $h+9$ of s; (2) remove from s the rightmost occurrence of $b_{i,1}\dots b_{i,9}$. It follows that s is a common supersequence of s_1 and s_2, as each symbol between $b_{i-1,9}$ and $b_{i,1}$ in s_j, $1 \leqslant i \leqslant 2$, is mapped by the threading schema in a symbol of the substring s_a of s. Notice that the size of s after the insertion of s_a and the deletion of $b_{i,1} \dots b_{i,9}$ is decreased by one, as $|s_a| = 8$. Hence we can assume that s contains exactly one occurrence of the substring $b_{i,1} \dots b_{i,9}$, for each i with $1 \leqslant i \leqslant n$, and exactly one occurrence of each symbol in $\{b_{i,1}, \dots, b_{i,9}\}$.

Define s_i, $1 \leqslant i \leqslant n$, as the substring of s bounded on the left by the occurrence of symbol $b_{i,1}$ (included) and on the right by the occurrence of $b_{i+1,1}$ (not included; notice that s_n is delimited on the right by the end of the string s). From the property of common supersequence, it follows that s_i, $1 \leqslant i \leqslant n$, must be a supersequence of $b_i(s_1)$ and $b_i(s_2)$. Now, we show that starting from s we can compute solution s' that consists of the concatenation of n substrings $s'_1, \dots, s'_n$ such that (1) for each i with $1 \leqslant i \leqslant n$, s'_i is either equal to s_i^+ or to s_i^*; (2) for each $e_{i,j} \in \Sigma$ associated with $\{v_i, v_j\} \in E$, it follows that s'_i is equal to s_i^+ or s'_j is equal to s_j^+; (3) $|s'| \leqslant |s|$.

Consider a symbol $e_{i,j} \in \Sigma$ associated with $\{v_i, v_j\} \in E$. Since s must contain at least three occurrences of $e_{i,j}$, it follows that either one of s_i, s_j contains at least two occurrences of $e_{i,j}$, or $e_{i,j}$ belongs to a substring s_h, with $h \neq i, j$. In the former case, if two occurrences of $e_{i,j}$ belong to s_i, define s'_i identical to s_i^+, else if two occurrences of $e_{i,j}$ belong to s_j, define s'_j identical to s_j^+. In the latter case, that is when $e_{i,j}$ belongs to a substring s_h, with $h \neq i, j$, define arbitrarily s'_i equal to s_i^+ or s'_j equal to s_j^+. For each other substring s'_i left, define s'_i equal to s_i^*.

Now, observe that s' is a solution of $\mathcal{C}-\mathcal{SCS}$ over instance (s_1, s_2, T_L), as by construction each s'_i, $1 \leqslant i \leqslant n$, is a short common supersequence of $b_i(s_1)$, $b_i(s_2)$ and since s' by construction contains at least three occurrences of each symbol $e_{i,j} \in \Sigma$, hence it satisfies the constraint T_L. Furthermore, $|s'| \leqslant |s|$. Indeed if $e_{i,j}$ belongs to s_i (or similarly to s_j), then $|s_i| \geqslant 17$ and $|s'_i| = 17$. If $e_{i,j}$ belongs to s_h, with $h \neq i, j$, then symbol $e_{i,j}$ does not belong to s'_h, while $|s'_i| \leqslant |s_i| + 1$ or $|s'_j| \leqslant |s_j| + 1$.

Now, starting from solution s', we can define a vertex cover V' of G as follows: $V' = \{v_i\colon\ s'_i$ is identical to $s_i^+\}$. Indeed, for each edge $\{v_i, v_j\}$, at least one of v_i, v_j belongs to V'. It follows that starting from a solution of $\mathcal{C}-\mathcal{SCS}$ of size $9n + 8p + 7(n-p)$, we can compute in polynomial time a solution of size p for MVCC. □

As a vertex cover of a cubic graph has size at least $\frac{|V|}{4}$, it follows by Claim 2 and by Claim 3 that we have described an L-reduction from MVCC to $\mathcal{C}-\mathcal{SCS}$. Hence $\mathcal{C}-\mathcal{SCS}$ is APX-hard, even when each symbol occurs at most twice in each input string and $T_L[i] \leqslant 3$, with $1 \leqslant i \leqslant |\Sigma|$. □

Next, we will consider the case when each $T_L[i] \leqslant 2$, $1 \leqslant i \leqslant |\Sigma|$.

Theorem 4. *$\mathcal{C}-\mathcal{SCS}$ problem is polynomially time solvable, when each $T_L[i] \leqslant 2$, $1 \leqslant i \leqslant |\Sigma|$.*

Proof. We will show that $\mathcal{C}-\mathcal{SCS}$ when each $T_L[i] \leqslant 2$, $1 \leqslant i \leqslant |\Sigma|$, can be reduced to the SCS problem. As the SCS problem is polynomially time solvable [9], if follows that also the restriction of the $\mathcal{C}-\mathcal{SCS}$ problem when each $T_L[i] \leqslant 2$, $1 \leqslant i \leqslant |\Sigma|$, is polynomially time solvable.

Let us consider different cases depending on the values of $occ_{s_1}(a_i)$, $occ_{s_2}(a_i)$. We assume that $\max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} \leqslant T_L[i] \leqslant occ_{s_1}(a_i) + occ_{s_2}(a_i)$ otherwise the constraint is redundant. Since $T_L[i] \leqslant 2$, it follows that $\max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} \leqslant 2$. Furthermore, we can assume that $\min\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} > 0$, otherwise two occurrences of a_i will never match.

Consider the case that $T_L[i] = 1$, for some i with $1 \leqslant i \leqslant |\Sigma|$. Since $\min\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} > 0$, the constraint $T_L[i]$ for a_i is satisfied by any common supersequence of s_1, s_2.

Consider the case that $T_L[i] = 2$ and $\max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\} = 2$. Again, the constraint is satisfied by any common supersequence of s_1, s_2, since any common supersequence of s_1 and s_2 contains at least two occurrences of a_i.

Consider the case that, for a symbol $a_i \in \Sigma$, $T_L[i] = 2$ and $occ_{s_1}(a_i) = occ_{s_2}(a_i) = 1$ and denote by $\Sigma' \subseteq \Sigma$ the set of such symbols. Since any solution of $\mathcal{C}-\mathcal{SCS}$ must contain two occurrences of $a_i \in \Sigma'$, we can assume that the two positions where a_i occurs in s_1 and s_2 will never match. It follows that we can reduce this restriction of $\mathcal{C}-\mathcal{SCS}$ to the problem of computing a shortest common subsequence s^* of two new strings $s_{1,r}$, $s_{2,r}$ obtained by removing the occurrences of symbols in Σ' from s_1, s_2 respectively. A solution s of $\mathcal{C}-\mathcal{SCS}$ can be easily computed by adding the symbols in Σ' to s^*, respecting the order in which they appear in s_1, s_2, so that s is an SCS of s_1, s_2. □

4. A $\frac{3}{2}$-approximation algorithm

In this section, we present an approximation algorithm for $\mathcal{C}-\mathcal{SCS}$ of factor $\frac{3}{2}$. A 2-approximation algorithm for the problem can be easily obtained by returning as a solution the concatenation of the two input strings s_1 and s_2.

Recall that, given a string s over alphabet Σ and a symbol $a_i \in \Sigma$, $occ_s(a_i)$ denote the number of occurrences of symbol a_i in s. The $\frac{3}{2}$-approximation algorithm first computes in polynomial time a shortest common supersequence s of s_1 and s_2. Then, it considers the set $D \subseteq \Sigma$ of symbols defined as follows: $D = \{a_i \in \Sigma\colon\ occ_s(a_i) < T_L[i]\}$. For each $a_i \in \Sigma \cap D$, define the string $s(a_i)$ as the string consisting of $T_L[i] - occ_s(a_i)$ occurrences of symbol a_i. The algorithm defines a solution s_{approx} as the concatenation of the string s with the strings $s(a_i)$, for each $a_i \in \Sigma \cap D$, arbitrarily ordered. The solution s_{approx} is, by construction, a common supersequence of s_1 and s_2. Furthermore, by construction, it satisfies the constraint T_L.

Before proving the approximation factor, we introduce some definitions. Define $s^l_{approx} = s_{approx}[1, |s|]$, and $s^r_{approx} = s_{approx}[|s| + 1, |s_{approx}|]$. Denote by s_{opt} an optimal solution of $\mathcal{C} - \mathcal{SCS}$, and observe that, since a solution of $\mathcal{C} - \mathcal{SCS}$ is a common supersequence of s_1 and s_2 and must satisfy T_L, $|s^l_{approx}| \leqslant |s_{opt}|$ and, by construction, $|s^r_{approx}| \leqslant |s_{opt}|$.

Consider a symbol $a_j \in \Sigma \cap D$. Solution s_{approx} contains exactly $T_L[j]$ occurrences of symbol a_j, hence $occ_{s_{approx}}(a_i) \leqslant occ_{s_{opt}}(a_i)$. Denote by $Bad = \{a_i \in \Sigma \setminus D\colon occ_{s_{approx}}(a_i) > T_L[i] \wedge occ_{s_{approx}}(a_i) > occ_{s_{opt}}(a_i)\}$. A symbol $a_i \in Bad$ is called a *bad symbol*. We define the set *Good* of symbols as $Good = \Sigma \setminus Bad$; a symbol $a_i \in Good$ is called a *good symbol*.

Starting from the string s_{approx}, it is possible to compute a subsequence s'_{approx} of s_{approx} such that $|s'_{approx}| = |s_{opt}|$ by removing $|s_{approx}| - |s_{opt}|$ occurrences of symbols in $\Sigma \cap Bad$. Define $ex(s_{approx}, a_i) = occ_{s_{approx}}(a_i) - occ_{s'_{approx}}(a_i)$ (called the *exceeding occurrences* of a_i in s_{approx}) and $ex(s_{approx}) = |s_{approx}| - |s_{opt}|$ (called the *exceeding occurrences* of s_{approx}). Notice that $ex(s_{approx}, a_i) \leqslant occ_{s_{approx}}(a_i) - occ_{s_{opt}}(a_i)$, for each $a_i \in Bad$.

Next, we show that the above approximation algorithm achieves an approximation factor of $\frac{3}{2}$.

Lemma 5. *For each symbol $a_i \in \Sigma$, s^l_{approx} contains at most $occ_{s_1}(a_i) + occ_{s_2}(a_i)$ occurrences of a_i.*

Proof. The proof follows from the properties of the shortest common supersequence. □

Lemma 6. $ex(s_{approx}) \leqslant \min\{|s^l_{approx}|, |s^r_{approx}|\}$.

Proof. First, assume that $|s^l_{approx}| > |s^r_{approx}|$. Observe that an exceeding occurrence of a bad symbol must be in s^l_{approx}, as all the symbols occurring in s^r_{approx} are good symbols. After the removal of $|s^r_{approx}|$ symbols from s_{approx}, the resulting string has size $|s^l_{approx}|$, and by construction $|s^l_{approx}| \leqslant |s_{opt}|$. It follows that in this case $ex(s_{approx}) \leqslant \min\{|s^l_{approx}|, |s^r_{approx}|\}$.

Assume that $|s^r_{approx}| \geqslant |s^l_{approx}|$. By removing s^l_{approx} symbols from s_{approx}, we obtain a string of size $|s^r_{approx}|$. Since $|s^r_{approx}| \leqslant |s_{opt}|$, it follows that also in this case $ex(s_{approx}) \leqslant \min\{|s^l_{approx}|, |s^r_{approx}|\}$. □

Now we are able to prove the main result of this section, that is that $|s_{approx}| \leqslant \frac{3}{2}|s_{opt}|$.

Theorem 7. $|s_{approx}| \leqslant \frac{3}{2}|s_{opt}|$.

Proof. Observe that, for each symbol $a_i \in Good$, s_{approx} contains exactly $T_L[i]$ occurrences of a_i and s_{opt} must contain at least $T_L[i]$ occurrences of a_i.

Consider the occurrences of bad symbols in s_{approx}. Observe that, for each bad symbol $a_i \in Bad$, it follows by Lemma 5 that s_{approx} contains at most $occ_{s_1}(a_i) + occ_{s_2}(a_i)$ occurrences of a_i, while $occ_{s_{opt}} \geqslant \max\{occ_{s_1}(a_i), occ_{s_2}(a_i)\}$. As the number of exceeding occurrences of a_i in s_{approx} is bounded by $\min\{occ_{s_1}(a_i), occ_{s_2}(a_i)\}$, it follows that, for each $a_i \in Bad$,

$$ex(s_{approx}, a_i) \leqslant occ_{s_{opt}}(a_i). \tag{1}$$

Furthermore, we claim that $ex(s_{approx}, a_i)$ is not greater than the number of occurrences of good symbols in s_{approx}. Indeed, an occurrence of a bad symbol is part of s^l_{approx}, as s^r_{approx} consists only of good symbols. Then, if $|s^l_{approx}| \leqslant |s^r_{approx}|$, the claim trivially holds. If $|s^l_{approx}| > |s^r_{approx}|$, then by Lemma 6 there exists at most $|s^r_{approx}|$ exceeding occurrences in s_{approx}. As a consequence, the following inequality holds:

$$\sum_{a_i \in Bad} ex(s_{approx}, a_i) \leqslant \sum_{a_j \in Good} occ_{s_{approx}}(a_j) \leqslant \sum_{a_j \in Good} occ_{s_{opt}}(a_j). \tag{2}$$

Combining Inequality (1) with Inequality (2), we get

$$\sum_{a_i \in Bad} ex(s_{approx}, a_i) \leqslant \frac{1}{2}\left(\sum_{a_i \in Bad} occ_{s_{opt}}(a_i) + \sum_{a_j \in Good} occ_{s_{opt}}(a_j)\right). \tag{3}$$

The length of s_{approx} can be bounded as follows:

$$|s_{approx}| \leqslant \sum_{a_i \in Bad} ex(s_{approx}, a_i) + \sum_{a_i \in Bad} occ_{s_{opt}}(a_i) + \sum_{a_j \in Good} occ_{s_{approx}}(a_j).$$

Similarly, the following inequality holds for the length of s_{opt}:

$$|s_{opt}| \geqslant \sum_{a_i \in Bad} occ_{s_{opt}}(a_i) + \sum_{a_j \in Good} occ_{s_{opt}}(a_j).$$

Since $occ_{s_{approx}}(a_j) \leqslant occ_{s_{opt}}(a_j)$, for each $a_j \in Good$, and from Inequality (3), it follows that

$$|s_{approx}| \leqslant \frac{3}{2}\left(\sum_{a_i \in Bad} occ_{s_{opt}}(a_i) + \sum_{a_j \in Good} occ_{s_{opt}}(a_j)\right) \leqslant \frac{3}{2}|s_{opt}|. \qquad \square$$

5. A fixed-parameter algorithm

In this section we investigate the Parameterized Complexity of the $\mathcal{C}-\mathcal{SCS}$ problem. For more details on Parameterized Complexity, we refer the reader to [10,16]. Let k be a positive integer, the parameterized version of the $\mathcal{C}-\mathcal{SCS}$ problem, denoted as $k-\mathcal{C}-\mathcal{SCS}$, given two input strings s_1, s_2 and a constraint T_L, asks if there exists a supersequence s of s_1, s_2 that satisfies the constraint T_L and such that $|s| \leqslant k$. In what follows we give a fixed-parameter algorithm for $k-\mathcal{C}-\mathcal{SCS}$ of time complexity $O^*(1.7321)^k$.

It is easy to see that the $k-\mathcal{C}-\mathcal{SCS}$ problem is fixed-parameter tractable. Indeed, for any shortest common supersequence s it holds $\max\{|s_1|, |s_2|\} \leqslant |s| \leqslant |s_1| + |s_2|$. Trying all the subsequences of s_1 (or equivalently of s_2) as the matching of a solution of $k-\mathcal{C}-\mathcal{SCS}$ leads to a fixed-parameter algorithm, as there exist at most $O(2^k)$ subsequences s' of s_1. Next, we propose a depth-bounded search tree algorithm of time complexity $O^*(1.7321)^k$. Informally, starting from the rightmost positions of s_1, s_2, the algorithm aims to reconstruct a solution s of $k-\mathcal{C}-\mathcal{SCS}$ by identifying which positions in s_1 and s_2 *match* and which positions *do not match*, checking that the constraint T_L is satisfied.

Consider the substrings $s_1[1,i]$, $s_2[1,j]$, and the constraint T_L. Let us assume that $s_1[i] = a_h$ (or similarly $s_2[j] = a_h$), for some $a_h \in \Sigma$. At each step the algorithm first checks if $T_L[h] \geqslant \max\{occ_{s_1}(a_h), occ_{s_2}(a_h)\}$. If this inequality holds, the algorithm assumes that each occurrence of a_h in $s_1[1,i]$, $s_2[1,j]$ does not match, as the constraint on a_h is redundant, hence defining a matching between an occurrence of a_h in s_1 and an occurrence of a_h in s_2 does not decrease the length of the solution.

Now, the algorithm considers the positions i and j in s_1, s_2 respectively, and checks if $s_1[i] = s_2[j] = a_h$. If this case holds (notice that we have assumed that $occ_{s_1}(a_h) + occ_{s_2}(a_h) > T_L[h]$), the algorithm defines a matching between positions i and j, it decreases $T_L[h]$ by 1 and does not branch. If $s_1[i] \neq s_2[j]$, then notice that at most one of the positions i and j can match. The algorithm branches in the following cases:

Case 1 Position i matches a position of $s_2[1,j]$; in particular, we assume that $s_1[i] = a_z$ and that i matches the rightmost position h of $s_2[1,j]$ such that $s_2[h] = a_z$. By hypothesis, $s_2[j] \neq s_1[i]$, hence $h < j$. Then all the positions in $s_2[h+1,j]$ are removed and considered unmatched positions. We decrease $T_L[z]$ by 1, and we decrease $T_L[w]$ by 1, for each $s_2[l] = a_w$, with $h < l \leqslant j$. Then the strings $s_1[1,i-1]$ and $s_2[1,h-1]$ are considered.

Case 2 Position j matches a position of $s_1[1,i]$; in particular, we assume that $s_2[j] = a_z$ and that j matches the rightmost position h of $s_1[1,i]$ such that $s_1[h] = a_z$. By hypothesis, $s_2[j] \neq s_1[i]$, hence $h < j$. Then all the positions in $s_1[h+1,i]$ are removed and considered unmatched positions of a solution s. We decrease $T_L[z]$ by 1, and we decrease $T_L[w]$ by 1, for each $s_1[l] = a_w$, with $h < l \leqslant i$. Then the strings $s_1[1,h-1]$ and $s_2[1,j-1]$ are considered.

Case 3 Positions i and j do not match; hence they are removed from $s_1[1,i]$ and $s_2[1,j]$ respectively. We decrease $T_L[h]$ by 1, and we decrease $T_L[w]$ by 1. Then the strings $s_1[1,i-1]$ and $s_2[1,j-1]$ are considered.

Notice that in each of the three branching cases, at least two positions of a solution s of $k-\mathcal{C}-\mathcal{SCS}$ are removed from s_1 and s_2.

The algorithm constructs a search tree by branching according to the three cases presented above. Notice that the branching procedure is terminated when a feasible solution of $k-\mathcal{C}-\mathcal{SCS}$ is computed or when a branch has reached length $k+1$. The algorithm returns a solution of $k-\mathcal{C}-\mathcal{SCS}$ if at least one of the branches of the search tree leads to a feasible solution. Let us now discuss the correctness of the algorithm.

Theorem 8. *The depth-bounded search tree algorithm returns a solution of $k-\mathcal{C}-\mathcal{SCS}$ if and only if the $k-\mathcal{C}-\mathcal{SCS}$ problem admits a solution, in time $O^*(1.7321)^k$.*

Proof. First, we prove the correctness of the algorithm. Consider a solution s returned by the algorithm. Notice that the solution satisfies each constraint. Indeed, if $T_L[h] \geqslant occ_{s_1}(a_h) + occ_{s_2}(a_h)$, then none of the occurrences of a_i will be defined as a matched position by the algorithm, and the solution returned by the algorithm will contain exactly $occ_{s_1}(a_h) + occ_{s_2}(a_h)$ occurrences of symbol a_h (other occurrences of a_h are eventually appended at the right end of s, in order to satisfy the constraint $T_L[h]$). Hence in this case s contains exactly $T_L[h]$ occurrences of a_h, and the same property holds for any solution of $k-\mathcal{C}-\mathcal{SCS}$.

Now, assume that $s_1[i] = s_2[j] = a_h$ and $T_L[h] < occ_{s_1}(a_h) + occ_{s_2}(a_h)$. We can assume that positions i, j match, as we can assume that in the rightmost position r of a solution s of $k-\mathcal{C}-\mathcal{SCS}$, $s[r] = a_h$.

Assume now that $s_1[i] = a_h \neq s_2[j]$, by the property of common supersequence at most one of the positions i, j can be a matching in a solution of $k-\mathcal{C}-\mathcal{SCS}$. In case both positions are unmatched in a solution s of $k-\mathcal{C}-\mathcal{SCS}$, then the correctness of the algorithm follows by Case 3 and by induction. Hence, assume that exactly one of i, j matches, w.l.o.g. i, matches position z, with $z < j$, of $s_2[1,j-1]$ in a solution s of $k-\mathcal{C}-\mathcal{SCS}$. We claim that there exists a solution of $k-\mathcal{C}-\mathcal{SCS}$ where $s_2[z]$ is the rightmost position of $s_2[1,j]$ containing a symbol a_h. Assume that in a solution s^* of

$k - \mathcal{C} - \mathcal{SCS}$, i matches a position w of s_2, such that $s_2[w] = a_h$ and $w < z$. By the property of common supersequence, if the position i_1 in s^* represents the matching between i and w, then there is a position i_2 in s^*, with $i_1 < i_2$, such that the unmatched position z, with $s_2[z] = a_h$, is mapped in i_2. Then we can assume that position i_2 represents a matching between i and z, as $s^*[i_2] = a_h$. A similar property holds when $s_2[j]$ matches some position of $s_1[1, i]$. In this case the correctness of the algorithm follows by Case 1 (or by Case 2) and by induction.

Next, let us discuss the time complexity of the algorithm. Observe that in Case 1 and Case 2 of the branching procedure, we remove one matched position of string s_i, with $1 \leqslant i \leqslant 2$, that belongs to solution s and at least one unmatched position that belongs to s. In the third case of branching, we remove two unmatched positions that belong to s. In each case we remove from the input strings s_1, s_2 at least two positions that belong to a solution s, thus we decrease the size of the solution we are computing by at least 2. Hence the time complexity of the algorithm is given by the following recurrence $T(k) = 3T(k-2) + O(n)$, as $O(n)$ time is required to search for the rightmost occurrence of a symbol a_h in a string and to update the constraint T_L. It follows that the algorithm has time complexity $O^*(1.7321)^k$. $\square$

6. Conclusion

In this paper we have introduced a new variant of the SCS problem, denoted as $\mathcal{C} - \mathcal{SCS}$, where given two input strings s_1, s_2 over alphabet Σ, and a constraint T_L, the problem asks for a shortest supersequence of s_1 and s_2 that satisfies the constraint T_L on the occurrences of each symbol in Σ. First we have investigated the computational and approximation complexity of the $\mathcal{C} - \mathcal{SCS}$ problem. While the $\mathcal{C} - \mathcal{SCS}$ problem is APX-hard when $T_L[i] \leqslant 3$ and each symbol occurs at most twice in each input string, the problem admits a polynomial time algorithm when $T_L[i] \geqslant 2$, for each $1 \leqslant i \leqslant |\Sigma|$. Furthermore, we have shown that $\mathcal{C} - \mathcal{SCS}$ admits an approximation algorithm of factor $\frac{3}{2}$. Finally, we have designed a fixed-parameter algorithm for the $\mathcal{C} - \mathcal{SCS}$ problem of time complexity $O^*(1.7321)^k$, where the parameter k is the size of the solution.

Interesting future directions include the investigation of the approximation complexity of the problem, in order to improve the approximation factor. Furthermore, it would be interesting to investigate if the $\mathcal{C} - \mathcal{SCS}$ problem can be further extended, defining new meaningful constraints.

References

[1] S. Adi, M. Braga, C. Fernandes, C. Ferreira, F. Martinez, M.-F. Sagot, M. Stefanes, C. Tjandraatmadja, Y. Wakabayashi, Repetition-free longest common subsequence, Discrete Appl. Math. 158 (12) (2010) 1315–1324.
[2] P. Alimonti, V. Kann, Some APX-completeness results for cubic graphs, Theor. Comput. Sci. 237 (1–2) (2000) 123–134.
[3] A.N. Arslan, Ö. Egecioglu, Algorithms for the constrained longest common subsequence problems, Int. J. Found. Comput. Sci. 16 (6) (2005) 1099–1109.
[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer, Heidelberg, 1999.
[5] P. Bonizzoni, G. Della Vedova, R. Dondi, G. Fertin, R. Rizzi, S. Vialette, Exemplar longest common subsequence, IEEE/ACM Trans. Comput. Biol. Bioinform. 4 (4) (2007) 535–543.
[6] P. Bonizzoni, G. Della Vedova, R. Dondi, Y. Pirola, Variants of constrained longest common subsequence, Inf. Process. Lett. 110 (20) (2010) 877–881.
[7] F.Y.L. Chin, A.D. Santis, A.L. Ferrara, N.L. Ho, S.K. Kim, A simple algorithm for the constrained sequence problems, Inf. Process. Lett. 90 (4) (2004) 175–179.
[8] R. Clifford, Z. Gotthilf, M. Lewenstein, A. Popa, Restricted common superstring and restrict common supersequence, in: Proc. of CPM, 2011, pp. 467–478.
[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, McGraw–Hill, 2002.
[10] R. Downey, M. Fellows, Parameterized Complexity, Springer-Verlag, 1999.
[11] C. Ferreira, C. Tjandraatmadja, A branch-and-cut approach to the repetition-free longest common subsequence problem, Electron. Notes Discrete Math. 36 (2010) 527–534.
[12] Z. Gotthilf, M. Lewenstein, A. Popa, On shortest common superstring and swap permutations, in: Proc. of SPIRE 2010, 2010, pp. 270–278.
[13] C.S. Iliopoulos, M.S. Rahman, New efficient algorithms for the LCS and constrained LCS problems, Inf. Process. Lett. 106 (1) (2008) 13–18.
[14] T. Jiang, M. Li, On the approximation of shortest common supersequences and longest common subsequences, SIAM J. Comput. 24 (5) (1995) 1122–1139.
[15] D. Maier, The complexity of some problems on subsequences and supersequences, J. ACM 25 (1978) 322–336.
[16] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.
[17] K. Ning, H. Kee Ng, H. Wai Leong, Finding patterns in biological sequences by longest common subsequences and shortest common supersequences, in: Proceedings of BIBE, 2006, pp. 53–60.
[18] S. Rahmann, The shortest common supersequence problem in a microarray production setting, in: Proceedings of European Conference on Computational Biology, 2003, pp. 156–161.
[19] K. Räihä, E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, Theor. Comput. Sci. 16 (1981) 187–198.
[20] E.D. Sacerdoti, A Structure for Plans and Behavior, Elsevier North-Holland, New York, 1977.
[21] D. Sankoff, Genome rearrangement with gene families, Bioinformatics 11 (1999) 909–917.
[22] Y. Tsai, The constrained longest common subsequence problems, Inf. Process. Lett. 88 (4) (2003) 173–176.
[23] D.E. Wilkins, Practical Planning: Extending the Classical AI Planning Paradigm, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1988.