Algorithmica (2011) 61:227–251
DOI 10.1007/s00453-010-9419-8

Reoptimization of the Shortest Common Superstring Problem

Davide Bilò · Hans-Joachim Böckenhauer · Dennis Komm · Richard Královič · Tobias Mömke · Sebastian Seibert · Anna Zych

Received: 4 August 2009 / Accepted: 31 May 2010 / Published online: 15 June 2010

Abstract A reoptimization problem describes the following scenario: given an instance of an optimization problem together with an optimal solution for it, we want to find a good solution for a locally modified instance.

In this paper, we deal with reoptimization variants of the shortest common superstring problem (SCS) where the local modifications consist of adding or removing a single string. We show the NP-hardness of these reoptimization problems and design

This work was partially supported by SNF grant 200021-121745/1 and SBF grant C 06.0108 as part of the COST 293 (GRAAL) project funded by the European Union. An extended abstract of this paper appeared at CPM 2009 [D. Bilò, H.-J. Böckenhauer, D. Komm, R. Královič, T. Mömke, S. Seibert, A. Zych, Reoptimization of the shortest common superstring problem. In: Proc. of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009). LNCS, vol. 5577, pp. 78–91. Springer, Berlin (2009) (extended abstract)].

D. Bilò
Department of Computer Science, University of L'Aquila, L'Aquila, Italy
e-mail: davide.bilo@univaq.it

H.-J. Böckenhauer · D. Komm (✉) · R. Královič · T. Mömke · A. Zych
Department of Computer Science, ETH Zurich, Zurich, Switzerland
e-mail: dennis.komm@inf.ethz.ch

H.-J. Böckenhauer
e-mail: hjb@inf.ethz.ch

R. Královič
e-mail: richard.kralovic@inf.ethz.ch

T. Mömke
e-mail: tobias.moemke@inf.ethz.ch

A. Zych
e-mail: anna.zych@inf.ethz.ch

S. Seibert
Department of Computer Science, RWTH Aachen University, Aachen, Germany
e-mail: seibert@cs.rwth-aachen.de

several approximation algorithms for them. First, we use a technique of iteratively using any SCS algorithm to design an approximation algorithm for the reoptimization variant of adding a string whose approximation ratio is arbitrarily close to $8/5$ and another algorithm for deleting a string with a ratio tending to $13/7$. Both algorithms significantly improve over the best currently known SCS approximation ratio of 2.5. Additionally, this iteration technique can be used to design an improved SCS approximation algorithm (without reoptimization) if the input instance contains a long string, which might be of independent interest. However, these iterative algorithms are relatively slow. Thus, we present another, faster approximation algorithm for inserting a string which is based on cutting the given optimal solution and achieves an approximation ratio of $11/6$. Moreover, we give some lower bounds on the approximation ratio which can be achieved by algorithms that use such cutting strategies.

Keywords Reoptimization · Shortest Common Superstring · Approximation algorithms

1 Introduction

In classical algorithmics, one is interested in finding good feasible solutions to input instances about which nothing is known in advance. Unfortunately, many practically relevant problems are computationally hard, and so different approaches such as approximation algorithms or heuristics are used for computing good approximations for optimal solutions. In the real world, however, some extra knowledge about the instance at hand might be already known. The concept of reoptimization employs a special kind of additional knowledge: under the assumption that we are given an instance of an optimization problem together with an optimal solution for it, we want to efficiently compute a good solution for a locally modified input instance.

This concept of reoptimization was mentioned for the first time in [15] in the context of postoptimality analysis for some scheduling problem. Postoptimality analysis deals with the related question of how much an instance may be altered without changing the set of optimal solutions, see, e.g., [19]. Since then, the concept of reoptimization has been successfully applied to various problems like the traveling salesman problem [1, 3, 7, 8], the Steiner tree problem [4, 10, 11], the knapsack problem [2], and various covering problems [5]. A survey of reoptimization problems can be found in [9].

In this paper, we investigate some reoptimization variants of the *shortest common superstring problem*, SCS for short. Given a substring-free set of strings, the SCS asks for a shortest common superstring of S, i.e., for a minimum-length string containing all strings from S as substrings. The SCS is one of the most prominent hard problems in stringology with many applications, e.g., in computational biology where it is used for modeling certain aspects of the DNA fragment assembly problem (see, for instance, [6, 16] for more details). The SCS is known to be NP-hard [12] and even APX-hard [20]. Many approximation algorithms have been devised for the SCS, the most popular being a greedy algorithm proposed by Tarhio and Ukkonen [18] which can be proven to achieve an approximation ratio of 3.5 [13], but is conjectured to be

2-approximative. The currently best known approximation algorithms achieve a ratio of 2.5 [14, 17].

In this paper, we deal with reoptimizing the SCS under the local modifications of adding or removing a single string. Our main results are the following. We show that both reoptimization versions of the SCS are NP-hard and propose some approximation algorithms for them. First, we devise an iteration technique for improving the approximation ratio of any SCS algorithm in the presence of a long string in the input which might be of independent interest. Then, we use this iteration technique to design an algorithm for SCS reoptimization which gives an approximation ratio arbitrarily close to 1.6 for adding a string and a ratio arbitrarily close to 13/7 for removing a string. This algorithm uses some known approximation algorithm for the original SCS (without reoptimization), and its approximation ratio depends on the ratio of this SCS algorithm. Thus, any improvement over the best known ratio of 2.5 for the SCS immediately yields also an improvement of these reoptimization results. Since the running time of this iterative algorithm is rather high, we also analyze a simple and fast reoptimization algorithm, called OneCut, for adding a string and prove an approximation ratio of 11/6 for it.

The paper is organized as follows. In Sect. 2, we formally define the reoptimization variants of the SCS and fix our notation. Section 3 is devoted to the hardness results, in Sect. 4, we present the iterative reoptimization algorithms, and Sect. 5 contains the analysis of the fast approximation algorithm for adding a string. Finally, in Sect. 6 we give lower bounds for generalizations of the algorithm OneCut and prove that 11/6 is a tight bound on the approximation ratio of OneCut, and conclude the paper with some open problems in Sect. 7.

2 Preliminaries

We start with defining some notations for dealing with strings that we will use throughout the paper. By λ we denote the empty string. The concatenation of two strings s and t will be written as $s \cdot t$, or as st for short. Let s, t, x, and y be some (possibly empty) strings such that $t = xsy$. Then s is a *substring* of t (we write $s \sqsubseteq t$) and t is a *superstring* of s. If x is empty, we say that s is a *prefix* of t, if y is empty, then s is a *suffix* of t. We say that a set S of strings is *substring-free* if $s \not\sqsubseteq t$, for all $s, t \in S$.

For two strings s_1 and s_2, the *overlap* $\mathrm{ov}(s_1, s_2)$ of s_1 and s_2 is the maximum-length *proper* suffix of s_1 that is also a *proper* prefix of s_2. A prefix [suffix] of s is proper if and only if it is not empty and not equal to s. If no such string exists, we define $\mathrm{ov}(s_1, s_2)$ to be the empty string λ. The corresponding prefix of s_1, i.e., the string p such that $s_1 = p \cdot \mathrm{ov}(s_1, s_2)$, is denoted by $\mathrm{pref}(s_1, s_2)$. The *merge* of s_1 and s_2 is defined as $\mathrm{merge}(s_1, s_2) := \mathrm{pref}(s_1, s_2) \cdot s_2$. We inductively extend this notion of merge to more than two strings by defining

$$\mathrm{merge}(s_1, \ldots, s_m) = \mathrm{merge}(\mathrm{merge}(s_1, \ldots, s_{m-1}), s_m).$$

We call a string s *periodic* with period π, if there exist a suffix $\overline{\pi}$ and a prefix $\overline{\pi}$ of the string π and some $k \in \mathbb{N}$ such that $s = \overline{\pi} \cdot \pi^k \cdot \overline{\pi}$. In this case, we also write $s \sqsubseteq \pi^\infty$.

The problem we are investigating in this paper is to find the shortest common superstring for a given set $S = \{s_1, \ldots, s_m\}$ of strings. If S is substring-free, then the shortest common superstring can be unambiguously described by the order in which the strings appear in it: if $s_{i_1}, \ldots, s_{i_m}$ is the order of appearance in a shortest superstring t, then $t = \text{merge}(s_{i_1}, \ldots, s_{i_m})$. This observation leads to the following formal definition of the problem.

Definition 1 The *shortest common superstring problem*, SCS for short, is the following optimization problem: Given a substring-free set of strings $S = \{s_1, \ldots, s_m\}$, the feasible solutions are all permutations $(s_{i_1}, \ldots, s_{i_m})$ of S. For any feasible solution $\text{Sol} = (s_{i_1}, \ldots, s_{i_m})$, the cost is $|\text{Sol}| = |\text{merge}(s_{i_1}, \ldots, s_{i_m})|$, i.e., the length of the shortest superstring for S containing the strings from S in the order as given by Sol. The goal is to find a permutation minimizing the length of the corresponding superstring.

In this paper, we deal with two reoptimization variants of the SCS. The local modifications we consider here are adding a string to our set of input strings or deleting one string from it. The corresponding reoptimization problems can be formally defined as follows.

Definition 2 The input for the *SCS reoptimization problem with adding a string*, SCS+ for short, consists of a substring-free set $S_O = \{s_1, \ldots, s_m\}$ of strings, an optimal SCS -solution Opt_O for it, and a string $s_{new} \notin S_O$ such that also $S_N = S_O \cup \{s_{new}\}$ is substring-free.

Analogously, the input for the *SCS reoptimization problem with removing a string*, SCS– for short, consists of a substring-free set of strings $S_O = \{s_1, \ldots, s_m\}$, an optimal SCS-solution Opt_O for it, and a string $s_{old} \in S_O$. In this case, $S_N = S_O \setminus \{s_{old}\}$.

For both problems, the goal is to find an optimal SCS-solution Opt_N for S_N.

In addition to the maximum overlap and merge as defined above, we also consider the overlap and merge inside a given solution. Let Sol be some solution for an SCS instance given by a set of strings S and let s and t be two strings from S which are not necessarily overlapping in Sol. Then $\text{ov}_{\text{Sol}}(s, t)$ denotes the overlap of s and t in Sol, and we use $\text{merge}_{\text{Sol}}(s, t) = \text{merge}(s, \ldots, t)$ as an abbreviation for the merge of s and t together with all input strings lying between them in Sol. By $\text{prefM}_{\text{Sol}}(s, t)$, we denote the prefix of $\text{merge}_{\text{Sol}}(s, t)$ such that $\text{prefM}_{\text{Sol}}(s, t) \cdot t = \text{merge}_{\text{Sol}}(s, t)$. Note that s may be a proper prefix of $\text{prefM}_{\text{Sol}}(s, t)$. For $\text{Sol} = \text{Opt}_O$, we use the notations ov_O, merge_O, and prefM_O for ov_{Opt_O}, $\text{merge}_{\text{Opt}_O}$, and $\text{prefM}_{\text{Opt}_O}$, respectively. Analogously, we use ov_N, merge_N, and prefM_N for $\text{Sol} = \text{Opt}_N$. Note that, for two consecutive strings s and t inside some solution Sol, $\text{merge}_{\text{Sol}}(s, t) = \text{merge}(s, t)$, but this equality does not necessarily hold for non-consecutive strings.

3 Hardness Results

In this section, we show that the considered reoptimization problems are NP-hard. Similarly to [9], we use a polynomial-time Turing reduction since we rely on repeatedly applying reoptimizations.

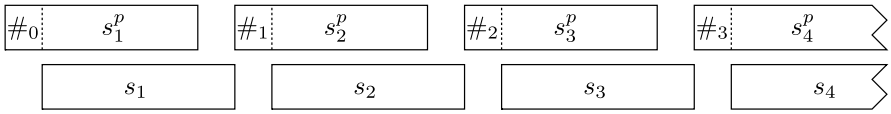


Fig. 1 An optimal solution for the easily solvable instance I'

Theorem 1 *The problems SCS+ and SCS– are NP-hard.*

Proof We split the reduction into several steps. Given an input instance I for SCS, we define a corresponding easily solvable instance I'. Then we show that I' is indeed solvable in polynomial time. Finally, we show how to use polynomially many reoptimization steps in order to transform the optimal solution for I' into an optimal solution for I.

At first, we consider the local modification of adding strings. For any SCS instance I, the easy instance I' consists of no strings. Obviously, the empty string is an optimal solution for I'. Now, I' can be transformed into any instance I by adding all strings from I one after the other. Thus, SCS+ is NP-hard.

Now, let us consider the local modification of removing strings. Let I be an instance for SCS that consists of m strings $s_1, \ldots, s_m$. For any i, let s_i^p be s_i without the last symbol.

We construct I' as follows. Let $\#_1, \ldots, \#_m$ be m different special symbols that do not appear in I. Then, we introduce the set of strings $S' := \{s'_1, \ldots, s'_m\}$, where $s'_i := \#_i s_i^p$, for each $i \in \{1, \ldots, m\}$. Let the instance I' be the set of the strings from I together with the strings from S'. It is clear that m local modifications, each removing one of the new strings, transform I' into I. Thus, it only remains to show that I' is efficiently solvable. To this end, we claim that no algorithm can do better than alternating the new and the old strings as depicted in Fig. 1.

We now formally prove the correctness of the construction above. First, observe that the constructed instance is substring-free. The solution obtained by alternating the new and old strings as in Fig. 1 has length $m + \sum_{i=1}^{m} |s_i|$. We need to show that this is optimal, i.e., no superstring of S' can be shorter.

Let us consider any common superstring t for I'. We decompose t into

$$w_0 w'_1 w_1 w'_2 w_2 \ldots w'_m w_m$$

such that each w'_i consists of exactly one special symbol. Hence, we can write that $w'_i = \#_{\phi_i}$ for some permutation ϕ of integers from 1 to m. Since no string from I contains any special symbols, it is contained in at least one of the strings w_i between the special symbols. Let k_i be the number of strings from I that are contained in w_i; it holds that $\sum_{i=0}^{m} k_i = m$. For any $i \geq 1$, $w'_i w_i$ is a superstring of some k_i words from I and the word from I' that contains w'_i, i.e., $s'_{\phi_i} = \#_{\phi_i} s_{\phi_i}^p$. Equivalently, w_i is a superstring of $s_{\phi_i}^p$ and some k_i words from I such that w_i starts with $s_{\phi_i}^p$.

Note that any common superstring t_1 of a substring-free set P of p strings has length at least $|w| + (p-1)$, where $w \in P$ is the first string in t_1 and therefore

$$|t_1| \geq |w| + p - 1. \tag{1}$$

Applying (1), we have a lower bound on the length of w_i for any $i \geq 1$:

$$|w_i| \geq |s_{\phi_i}^p| + k_i = |s_{\phi_i}| - 1 + k_i. \tag{2}$$

Obviously, the length of w_0 cannot be less than the number of strings it contains, i.e., $|w_0| \geq k_0$.

Hence, we have a lower bound on the length of t:

$$|t| = m + \sum_{i=0}^{m} |w_i| \geq m + \sum_{i=0}^{m} k_i + \sum_{i=1}^{m} (|s_{\phi_i}| - 1) \geq m + m - m + \sum_{i=1}^{m} |s_i|. \tag{3}$$

The lower bound of (3) matches exactly the upper bound of the solution in Fig. 1. Therefore, we conclude that SCS– is NP-hard. □

4 Iterative Algorithms for Adding or Removing a String

Consider any polynomial approximation algorithm A for SCS with approximation ratio γ. We show how to construct a polynomial reoptimization algorithm for SCS+ with approximation ratio arbitrarily close to $(2\gamma - 1)/\gamma$. Furthermore, we show a similar result for SCS– with approximation ratio $(3\gamma - 1)/(\gamma + 1)$. Since the best known polynomial approximation algorithm for SCS gives $\gamma = 2.5$, see [17], we obtain an approximation ratio arbitrarily close to $8/5 = 1.6$ for SCS+ and an approximation ratio arbitrarily close to $13/7 < 1.86$ for SCS–.

The core part of our reoptimization algorithms is an approximation algorithm for SCS that works well if the input instance contains at least one long string. More precisely, let $S = \{s_1, \ldots, s_m\}$ be an instance of SCS such that $\mu_0 \in S$ is a longest string in S, and let $|\mu_0| = \alpha_0|\mathrm{Opt}|$, for some $\alpha_0 > 0$, where Opt is an optimal solution of S.

Algorithm A_1 guesses the leftmost string l_1 and the rightmost string r_1 which overlap with μ_0 in the string corresponding to Opt, together with the respective overlap lengths. Afterwards, it computes a new instance S_1 by eliminating all substrings of $\mathrm{merge}_{\mathrm{Opt}}(l_1, \mu_0, r_1)$ from the instance S, calls the algorithm A on S_1 and appends $\mathrm{merge}(l_1, \mu_0, r_1)$ to the approximate solution returned by A.

Now we generalize A_1 by iterating this procedure k times. For an arbitrary constant k, we construct a polynomial-time approximation algorithm A_k for SCS that computes a solution of length at most

$$\left(1 + \frac{\gamma^k(\gamma - 1)}{\gamma^k - 1}(1 - \alpha_0)\right)|\mathrm{Opt}|.$$

For every $i \in \{1, \ldots, k\}$, we define strings l_i, r_i, and μ_i as follows: Let l_i be the leftmost string that overlaps with μ_{i-1} in Opt. If there is no such string, $l_i := \mu_{i-1}$. Similarly, let r_i be the rightmost string that overlaps with μ_{i-1} in Opt; if no such string exists, $r_i := \mu_{i-1}$. We define μ_i as $\mathrm{merge}_{\mathrm{Opt}}(l_i, \mu_{i-1}, r_i)$.

Algorithm A_k uses exhaustive search to find strings l_i, r_i and μ_i for every $i \in \{1, \ldots, k\}$. This can be done by assigning every possible string of S to l_i and r_i, and

trying every possible overlap between l_i, μ_{i-1} and r_i. For every feasible candidate set of strings and for every i, the algorithm computes the candidate solution Sol_i corresponding to the string $\mathrm{merge}(u_i, \mu_i)$, where u_i is the string corresponding to the result of algorithm A on the input instance S_i obtained by removing all substrings of μ_i from S. Algorithm A_k then outputs the best solution among all candidate solutions.

Theorem 2 *Let n be the total length of all strings in S, i.e., $n = \sum_{j=1}^{m} |s_j|$. Algorithm A_k works in time $\mathcal{O}(m^{2k}n^{2k}(kmn + kT(m,n)))$, where $T(m,n)$ is the time complexity of algorithm A on an input instance with at most m strings of total length at most n.*

Proof Algorithm A_k needs to test all $\mathcal{O}(m^{2k})$ possibilities for choosing $2k$ strings $l_1, r_1, \ldots, l_k, r_k$ from the m strings of S. For every such possibility, it must test all possible overlaps between the strings in order to obtain strings $\mu_1, \ldots, \mu_k$. Hence, the lengths of $2k$ overlaps must be tested. As the length of each overlap can be in the range from 0 to n, there are $\mathcal{O}(n^{2k})$ possibilities. For each of the $\mathcal{O}(m^{2k}n^{2k})$ possibilities, A_k tests if it is feasible (this can be done in time $\mathcal{O}(n)$) and computes the corresponding k candidate solutions. To compute one candidate solution Sol_i, the instance S_i is prepared in time $\mathcal{O}(mn)$ and algorithm A is executed in time $T(m,n)$. □

Theorem 3 *Algorithm A_k finds a solution of S of length at most*

$$\left(1 + \frac{\gamma^k(\gamma-1)}{\gamma^k - 1}(1-\alpha_0)\right)|\mathrm{Opt}|.$$

Proof Assume that A_k outputs a solution of length greater than $(1+\beta)|\mathrm{Opt}|$, for some $\beta > 0$. In the analysis, we focus on the part of the computation of A_k where the correct assignment of strings l_i, r_i, and μ_i is analyzed. By our assumption, every candidate solution Sol_i has length greater than $(1+\beta)|\mathrm{Opt}|$. The solution Sol_i corresponds to the string $\mathrm{merge}(u_i, \mu_i)$, where $|\mu_i| = \alpha_i|\mathrm{Opt}|$, for some $\alpha_i > 0$, and u_i is the result of algorithm A on the input instance S_i. Hence, $|\mathrm{Sol}_i| \le |u_i| + |\mu_i|$.

It is not difficult to check that, if we remove all substrings of μ_i from Opt, we obtain a feasible solution for S_i of length at most $|\mathrm{Opt}| - |\mu_{i-1}| = (1-\alpha_{i-1})|\mathrm{Opt}|$: by the definition of μ_i, we have removed every string that overlapped with μ_{i-1}. Hence, $|u_i| \le \gamma(1-\alpha_{i-1})|\mathrm{Opt}|$, and due to

$$(1+\beta)|\mathrm{Opt}| < |\mathrm{Sol}_i| \le (\gamma(1-\alpha_{i-1}) + \alpha_i)|\mathrm{Opt}|,$$

we conclude that

$$\alpha_i > 1 + \beta - \gamma + \gamma\alpha_{i-1}. \tag{4}$$

Solving the system of recurrent equations (4) yields

$$\alpha_k > (1+\beta-\gamma)\frac{\gamma^k - 1}{\gamma - 1} + \gamma^k\alpha_0. \tag{5}$$

Table 1 Ratios of A_k for different combinations of $|\mu_0|$, k, and γ

	Length of μ_0								
	$1/2 \cdot \|\text{Opt}\|$			$1/4 \cdot \|\text{Opt}\|$			$1/5 \cdot \|\text{Opt}\|$		
	Ratio γ			Ratio γ			Ratio γ		
k	2.0	2.5	3.5	2.0	2.5	3.5	2.0	2.5	3.5
1	2.0	2.25	2.75	2.5	2.86	≈3.63	2.6	3.0	3.8
2	≈1.67	≈1.89	≈2.36	2.0	≈2.34	≈3.04	≈2.07	≈2.43	≈3.18
3	≈1.57	≈1.80	≈2.28	≈1.86	≈2.2	≈2.92	≈1.91	≈2.28	≈3.05
5	≈1.52	≈1.76	≈2.26	≈1.77	≈2.14	≈2.89	≈1.83	≈2.21	≈3.00
10	≈1.5	≈1.75	≈2.25	≈1.75	≈2.13	≈2.88	≈1.80	≈2.20	≈3.00

Since μ_i is a substring of Opt for every i, it holds that $\alpha_k \leq 1$. Putting this together with (5) yields

$$\beta \leq \frac{\gamma^k(\gamma-1)}{\gamma^k-1}(1-\alpha_0). \qquad \square$$

In Table 1, we give some examplary ratios of A_k when using up to 10 iterations and the length of the longest string is either $1/2$, $1/4$, or $1/5$ of the length of the optimal solution. It is clear that the resulting approximation ratio highly depends on A's approximation ratio. As already mentioned, the best provable ratio is 2.5 using the algorithm of [17]. However, [18] introduces a much faster greedy algorithm which is conjectured to be a 2-approximation, although only a ratio of 3.5 is proven [13]. Due to this fact, we calculated the resulting ratios for both of them.

4.1 Reoptimization of SCS+

We now employ the iterative SCS algorithm described above for designing an approximation algorithm for SCS+. For every k, we define the algorithm A_k^+ for SCS+ as follows. Given an input instance S_O, its optimal solution Opt_O, and a new string s_{new}, the algorithm A_k^+ returns the solution Sol_1 corresponding to merge(Opt_O, s_{new}) or the solution Sol_2 computed by A_k for the input instance $S_N := S_O \cup \{s_{new}\}$, whichever is better.

Theorem 4 *Algorithm A_k^+ yields a solution of length at most*

$$\frac{2\gamma^{k+1}-\gamma^k-1}{\gamma^{k+1}-1}|\text{Opt}_N|.$$

Proof Let $|s_{new}| = \alpha|\text{Opt}_N|$. Then $|\text{Sol}_1| \leq (1+\alpha)|\text{Opt}_N|$. Since S_N contains a string of length at least $\alpha|\text{Opt}_N|$, Theorem 3 ensures that

$$|\text{Sol}_2| \leq \left(1+\frac{\gamma^k(\gamma-1)}{\gamma^k-1}(1-\alpha)\right)|\text{Opt}_N|.$$

Hence, the minimum of $|\mathrm{Sol}_1|$ and $|\mathrm{Sol}_2|$ is maximal if

$$(1+\alpha)|\mathrm{Opt}_N| = \left(1 + \frac{\gamma^k(\gamma-1)}{\gamma^k-1}(1-\alpha)\right)|\mathrm{Opt}_N|,$$

which happens if

$$\alpha = \frac{\gamma^{k+1}-\gamma^k}{\gamma^{k+1}-1}.$$

In this case, A_k^+ yields a solution of length at most

$$(1+\alpha)|\mathrm{Opt}_N| = \frac{2\gamma^{k+1}-\gamma^k-1}{\gamma^{k+1}-1}|\mathrm{Opt}_N|. \qquad \square$$

By choosing k sufficiently large, the approximation ratio of A_k^+ can be made arbitrarily close to $(2\gamma-1)/\gamma$. Algorithm A_k^+ is polynomial for every k, but the degree of the polynomial grows with k.

4.2 Reoptimization of SCS–

Similarly as for the case of SCS+, we define algorithm A_k^- for SCS– as follows. Given an input instance S_O, its optimal solution Opt_O and a string $s_{old} \in S_O$ to be removed, A_k^- returns the solution Sol_1 obtained from Opt_O by leaving out s_{old}, or the solution Sol_2 computed by A_k for input instance $S_N := S_O \setminus \{s_{old}\}$, whichever is better.

Theorem 5 *Algorithm A_k^- yields a solution of length at most*

$$\frac{3\gamma^{k+1}-\gamma^k-2}{\gamma^{k+1}+\gamma^k-2}|\mathrm{Opt}_N|.$$

Proof Let $l \in S_O$ ($r \in S_O$) be the string that immediately precedes [follows] s_{old} in Opt_O, respectively. We focus on the case where both l and r exist, the other cases are analogous. It is easy to see that

$$|\mathrm{Sol}_1| \le |\mathrm{Opt}_O| - |s_{old}| + |\mathrm{ov}(l, s_{old})| + |\mathrm{ov}(s_{old}, r)|.$$

Since augmenting Opt_N with s_{old} yields a feasible solution for S_O, we have $|\mathrm{Opt}_O| \le |\mathrm{Opt}_N| + |s_{old}|$.

Without loss of generality, assume that $|\mathrm{ov}(s_{old}, r)| \le |\mathrm{ov}(l, s_{old})| = \alpha|\mathrm{Opt}_N|$ for some $\alpha < 1$. Hence, $|\mathrm{Sol}_1| \le (1+2\alpha)|\mathrm{Opt}_N|$. Furthermore, S_N contains the string l of length at least $\alpha|\mathrm{Opt}_N|$, so Theorem 3 ensures that

$$|\mathrm{Sol}_2| \le \left(1 + \frac{\gamma^k(\gamma-1)}{\gamma^k-1}(1-\alpha)\right)|\mathrm{Opt}_N|.$$

The minimum of $|\mathrm{Sol}_1|$ and $|\mathrm{Sol}_2|$ is maximal if

$$(1+2\alpha)|\mathrm{Opt}_N| = \left(1+\frac{\gamma^k(\gamma-1)}{\gamma^k-1}(1-\alpha)\right)|\mathrm{Opt}_N|,$$

which happens if

$$\alpha = \frac{\gamma^{k+1}-\gamma^k}{\gamma^{k+1}+\gamma^k-2}.$$

In this case, A_k^- yields a solution of length at most

$$\frac{3\gamma^{k+1}-\gamma^k-2}{\gamma^{k+1}+\gamma^k-2}|\mathrm{Opt}_N|. \qquad \square$$

Similarly as in the case of SCS+, the approximation ratio of A_k^- can be made arbitrarily close to $(3\gamma-1)/(\gamma+1)$ by choosing k sufficiently large.

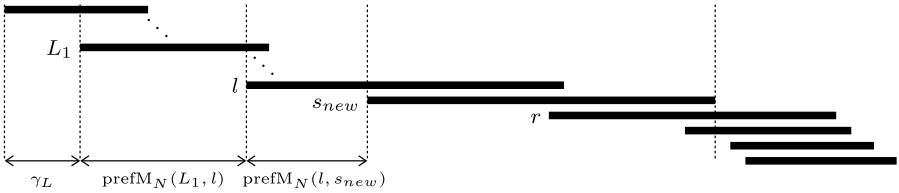
5 One-Cut Algorithm for Adding a String

In this section, we present a simple and fast algorithm OneCut for SCS+ which cuts the given solution at the best position possible and inserts the new string at this position. We prove that this algorithm achieves an 11/6-approximation ratio. As a first step, OneCut preprocesses the old optimal solution in such a way that it moves every string as much to the left as possible. After that, no string can be moved farther to the left; we call such a solution *maximally compressed*. The algorithm cuts Opt_O at all positions one by one. Recall that the given optimal solution Opt_O is represented by an ordering of the input strings, thus cutting Opt_O at some position yields a partition of the input strings into two sub-orderings. The two corresponding strings are then merged with s_{new} in between. The algorithm returns a shortest of the strings obtained in this manner, see Algorithm 1.

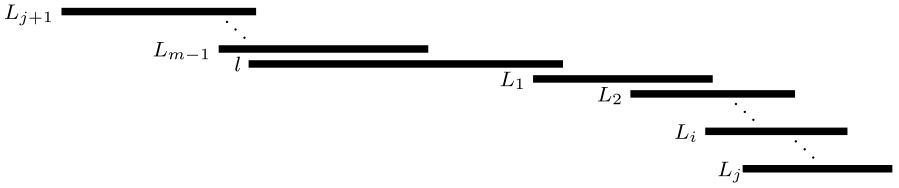
Algorithm 1: OneCut

Input: A set of strings $S = \{s_1, \ldots, s_m\}$, an optimal solution $\mathrm{Opt}_O = (s_1, \ldots, s_m)$ for S, and a string s_{new}
Preprocess(Opt_O);
for $i \in \{0, \ldots, m\}$ **do**
  Let $\mathrm{Solution}_i := (s_1, \ldots, s_i, s_{new}, s_{i+1}, \ldots, s_m)$;
end
Output: A best of the obtained solutions $\mathrm{Solution}_i$, for $0 \le i \le m$

Note that the preprocessing step of OneCut is necessary only for the analysis of the approximation ratio.



(a) The new optimal solution Opt_N (in the case that L_1 precedes l).



(b) The old optimal solution Opt_O (in the case that $L_i \neq \lambda$).

Fig. 2 The new and old optimal solution

Theorem 6 *The algorithm* OneCut *is an* 11/6*-approximation algorithm for SCS+ running in time* $\mathcal{O}(n \cdot m)$ *for inputs consisting of* m *strings of total length* n *over a constant-size alphabet.*

Proof We first analyze the running time of OneCut. The preprocessing can be done by successively finding the maximum overlap of $\text{merge}(s_1, \ldots, s_i)$ and s_{i+1}. This is possible in $\mathcal{O}(n \cdot m)$ time using standard pattern matching techniques. Then, using suffix trees, we can compute all pairwise overlaps of $\{s_{new}, s_1, \ldots, s_m\}$ in time $\mathcal{O}(n \cdot m)$, see e.g. [6]. Using these precomputed overlaps, each of the $m + 1$ iterations of OneCut can be performed in constant time. Thus, the overall running time of OneCut is also in $\mathcal{O}(n \cdot m)$.

We now show that OneCut provides an approximation ratio of 11/6 for SCS+. The proof is constructed in the following manner. One by one, we eliminate cases in which we can prove a ratio of 11/6 for OneCut, until all cases are covered. Each time we prove a ratio of 11/6 under some condition, we can deal in the following with the remaining cases under the assumption that this condition does not hold. In this way, we construct a list of assumptions which eventually lead to some final case.

Lemma 1 *If the added string* s_{new} *has a length of* $|s_{new}| \leq \frac{5}{6}|\text{Opt}_N|$*, then the algorithm* OneCut *provides an* 11/6*-approximation ratio.*

Proof Consider the trivial solution of appending s_{new} at the end of Opt_O. This solution is taken into account by OneCut. Note that if $|s_{new}| \leq (5/6) \cdot |\text{Opt}_N|$ then this trivial solution is already an 11/6-approximation. □

Lemma 1 shows that the desired approximation ratio can be reached whenever the string s_{new} is relatively short. This leads to the first assumption.

Assumption 1 *The length of the new string is* $|s_{new}| > \frac{5}{6}|\text{Opt}_N|$.

Under Assumption 1, we now look at the strings surrounding s_{new} in an arbitrary, but fixed optimal solution Opt_N of the modified instance. For this, let l be the string directly preceding s_{new} in Opt_N and let r be the direct successor of s_{new} in Opt_N (see Fig. 2(a), the additional strings $L_1, \ldots, L_{m-1}$ in Fig. 2 will be considered in a later stage of the analysis). If there is no predecessor [successor] of s_{new} in Opt_N, then l [r] is defined to be the empty string. Lemma 2 proves that we may assume, without loss of generality, that l and r almost completely cover the string s_{new}.

Lemma 2 *If* OneCut *returns an* $11/6$*-approximation for all instances where there is at most one letter from* s_{new} *not covered in* Opt_N *by either* l *or* r*, then it returns an* $11/6$*-approximation in general.*

Proof Assume that OneCut returns an $11/6$-approximation for any instance where there is at most one letter in s_{new} not covered in Opt_N by either l or r, and let us analyze the case when $s_{new} = \mathrm{ov}_N(l, s_{new}) \cdot \mu \cdot \mathrm{ov}_N(s_{new}, r)$ for some string μ such that $|\mu| > 1$. Consider an input instance for OneCut given by Opt_O and s'_{new}, where s'_{new} is s_{new} with μ replaced by a new symbol #. Let Opt'_N be a solution for Opt_O and s'_{new}, obtained by substituting s_{new} with s'_{new} in Opt_N. Note that Opt'_N is optimal: If there was a better solution, substitution of s'_{new} with s_{new} would give an improvement of Opt_N for the initial instance. Moreover, $|\mathrm{Opt}'_N| = |\mathrm{Opt}_N| - |\mu| + 1$. Let Sol' be a solution found by OneCut applied to Opt_O and s'_{new}. The solution Sol' is by assumption of the lemma an $11/6$-approximation of Opt'_N. We can obtain a feasible solution Sol for the initial instance $(\mathrm{Opt}_O, s_{new})$ by substituting s'_{new} with s_{new}. Then the following holds:

$$\begin{aligned}
|\mathrm{Sol}| &\leq |\mathrm{Sol}'| + |\mu| - 1 \leq \frac{11}{6}|\mathrm{Opt}'_N| + |\mu| - 1 \\
&\leq \frac{11}{6}|\mathrm{Opt}_N| - \frac{11}{6}(|\mu| - 1) + |\mu| - 1 \\
&\leq \frac{11}{6}|\mathrm{Opt}_N|.
\end{aligned}$$

Now it remains to observe that OneCut applied to $(\mathrm{Opt}_O, s_{new})$ considers Sol among other solutions. □

In what follows, we can therefore make a second assumption stating that the two strings l and r as defined above cover s_{new} almost completely.

Assumption 2 *In* Opt_N*, at most one letter of the string* s_{new} *is not covered by either* l *or* r*.*

Under this assumption, we show the following lemma which bounds the maximal length of the inserted string s_{new}.

Lemma 3 *Assumption 2 implies that either* $|s_{new}| \leq \frac{1}{2}|\mathrm{Opt}_N| + |\mathrm{ov}(l, s_{new})|$ *or* $|s_{new}| \leq \frac{1}{2}|\mathrm{Opt}_N| + |\mathrm{ov}(s_{new}, r)|$.

Proof Assume to the contrary that

$$|s_{new}| > \frac{1}{2}|\mathrm{Opt}_N| + |\mathrm{ov}(l, s_{new})| \text{ and } |s_{new}| > \frac{1}{2}|\mathrm{Opt}_N| + |\mathrm{ov}(s_{new}, r)|.$$

Summing up these two inequalities gives

$$2|s_{new}| > |\mathrm{Opt}_N| + |\mathrm{ov}(l, s_{new})| + |\mathrm{ov}(s_{new}, r)|.$$

According to Assumption 2, this implies $|s_{new}| > |\mathrm{Opt}_N| - 1$, contradicting the substring-freeness of the new instance. □

By Lemma 3 and Assumption 2, without loss of generality, we may assume the following for the rest of the proof.

Assumption 3 *The added string s_{new} has a length of $|s_{new}| \leq \frac{1}{2}|\mathrm{Opt}_N| + |\mathrm{ov}(l, s_{new})|$.*

From Assumption 3 and the fact that $|l| \geq |\mathrm{ov}(l, s_{new})|$, we obtain $|s_{new}| \leq \frac{1}{2}|\mathrm{Opt}_N| + |l|$. Together with Assumption 1 this implies the following:

Assumption 4 *The length of l can be bounded from below by $|l| \geq \frac{1}{3}|\mathrm{Opt}_N|$.*

We now enumerate the strings in Opt_O according to the position of l as shown in Fig. 2(b), i.e., Opt_O has the following composition

$$\mathrm{Opt}_O = (L_{j+1}, \ldots, L_{m-1}, l, L_1, \ldots, L_j)$$

for some $j \in \{0, \ldots, m-1\}$. In particular, let L_1 be the direct successor of l in Opt_O. If l has no successor in Opt_O, let $L_1 = \lambda$ be the empty string. In this case, the strings preceding l in Opt_O are $L_2, \ldots, L_m$, and L_1 is located at the end of Opt_O.[1]

In Lemma 4, we resolve the case where L_1 follows s_{new} in Opt_N.

Lemma 4 *Under Assumptions 1, 3, and 4, if L_1 is located after s_{new} in Opt_N, then* OneCut *returns an 11/6-approximation.*

Proof Consider the solution $\mathrm{Sol}_1 = \mathrm{merge}(L_{j+1}, \ldots, L_{m-1}, l, s_{new}, L_1, \ldots, L_j)$, where s_{new} is inserted between l and L_1, as presented in Fig. 3. Since L_1 is located after s_{new} in Opt_N, the size of $\mathrm{merge}(l, s_{new}, L_1)$ is bounded from above by the size of Opt_N. By Assumption 4 it follows that

$$\begin{aligned}
|\mathrm{Sol}_1| &\leq |\mathrm{Opt}_O| - |\mathrm{merge}_O(l, L_1)| + |\mathrm{merge}(l, s_{new}, L_1)| \\
&\leq |\mathrm{Opt}_O| - |\mathrm{merge}_O(l, L_1)| + |\mathrm{Opt}_N| \\
&\leq 2|\mathrm{Opt}_N| - |l| \leq \left(2 - \frac{1}{3}\right)|\mathrm{Opt}_N| \leq \frac{11}{6}|\mathrm{Opt}_N|
\end{aligned}$$

[1] Note that if $L_1 = \lambda$, there are $m-1$ strings preceding l in Opt_O, and we label them $L_2, \ldots, L_m$.

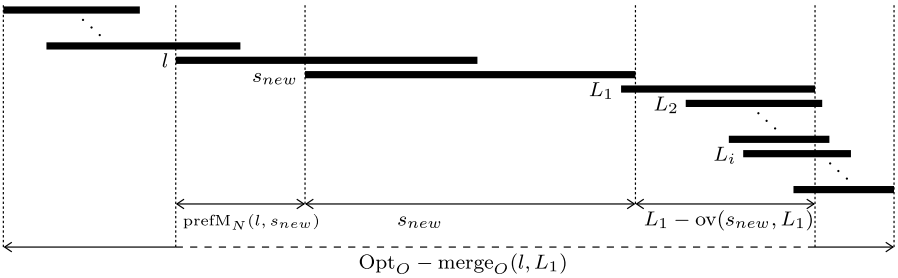
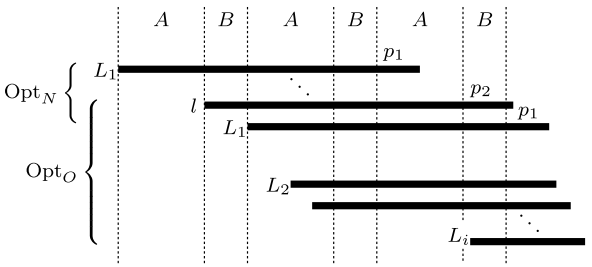


Fig. 3 The solution Sol_1

Fig. 4 Periodicity of l and L_1



which gives an 11/6-approximation ratio. □

If $L_1 = \lambda$, we may assume that it follows s_{new} in Opt_N. Thus, we can add the following assumption.

Assumption 5 *L_1 is non-empty and it precedes s_{new} in Opt_N.*

For the remainder of the proof, we need to analyze the periodic structure of the strings l and L_1. To this end, we introduce the following notation. We define $\pi_L = AB$, where $A = \mathrm{prefM}_N(L_1, l)$ and $B = \mathrm{prefM}_O(l, L_1) = \mathrm{pref}(l, L_1)$. Note that $L_1 = (AB)^g p_1$ and $l = (BA)^h p_2$ for some natural numbers g, h, where p_1 and p_2 denote some prefixes of AB and BA, respectively (see Fig. 4). Note that g and h might well be 0. Thus, $L_1, l \sqsubseteq \pi_L^\infty$ and

$$\mathrm{merge}_N(L_1, l) \sqsubseteq \pi_L^\infty. \tag{6}$$

Let γ_L denote the prefix of the superstring corresponding to Opt_N which precedes L_1 (see Fig. 2(a)).

We now distinguish two cases according to the length of π_L. The next lemma shows that we can guarantee our desired approximation ratio in case π_L is long.

Lemma 5 *Assumption 5 and $|\pi_L| \geq \frac{1}{6}|\mathrm{Opt}_N| - |\gamma_L|$ imply an approximation ratio of 11/6 for the algorithm* OneCut.

Proof Again, let us consider solution $\mathrm{Sol}_1 = \mathrm{merge}(L_{j+1}, \ldots, L_{m-1}, l, s_{new}, L_1, \ldots, L_j)$. The following three equalities (see Fig. 2)

1. $|\text{merge}_O(l, L_1)| = |l| + |L_1| - |\text{ov}_O(l, L_1)|$,
2. $|\text{pref}(l, s_{new})| = |\text{prefM}_N(l, s_{new})| \leq |\text{Opt}_N| - |\gamma_L| - |\text{prefM}_N(L_1, l)| - |s_{new}|$, and
3. $|l| - |\text{ov}_O(l, L_1)| = |\text{prefM}_O(l, L_1)|$

give the following bound on the cost of Sol_1:

$$\begin{aligned}|\text{Sol}_1| &\leq |\text{Opt}_O| - |\text{merge}_O(l, L_1)| + |s_{new}| + |\text{pref}(l, s_{new})| + |L_1| - |\text{ov}(s_{new}, L_1)| \\ &\leq 2|\text{Opt}_N| - |l| - |L_1| + |\text{ov}_O(l, L_1)| + |s_{new}| - |\gamma_L| \\ &\quad - |\text{prefM}_N(L_1, l)| - |s_{new}| + |L_1| \\ &\leq 2|\text{Opt}_N| - \underbrace{(|\text{prefM}_O(l, L_1)| + |\text{prefM}_N(L_1, l)|)}_{|\pi_L|} - |\gamma_L|.\end{aligned}$$

If $|\pi_L| \geq \frac{1}{6}|\text{Opt}_N| - |\gamma_L|$, then $|\text{Sol}_1| \leq \frac{11}{6}|\text{Opt}_N|$. □

In Lemma 5, we have handled the case that the period π_L is relatively long, yielding the following assumption for the rest of the proof.

Assumption 6 *The length of the period π_L is $|\pi_L| < \frac{1}{6}|\text{Opt}_N| - |\gamma_L|$.*

To proceed with the proof, we now need to look at the first string L_i after L_1 in Opt_O which is not periodic with period π_L, i.e., which satisfies $L_i \not\sqsubseteq \pi_L^\infty$. If there is no such string, let $L_i = \lambda$ be the empty string. Furthermore, let $L = \text{merge}_O(l, L_{i-1})$.

We now prove an approximation ratio of 11/6 for OneCut for the case in which L_i follows s_{new} in Opt_N. Note that this also holds if L_i is empty. To this end, we first need the following lemma (which does not depend on the position of L_i in Opt_N).

Lemma 6 *Under Assumptions 4, 5, and 6,*

$$|\text{merge}(L, s_{new}, L_i)| \leq |\text{merge}(l, s_{new}, L_i)| + |\pi_L|.$$

Proof Note that, due to Assumptions 4 and 6, we have $|l| > 2|\pi_L|$. We first prove that

$$|\text{merge}(L, q)| \leq |\text{merge}(l, q)| + |\pi_L|, \tag{7}$$

for an arbitrary string q.

Since both L and l are periodic with period π_L, to prove the claim it suffices to show that stretching $\text{merge}(l, q)$ by one period length yields a superstring of $\text{merge}(L, q)$. More precisely, since $L_{i-1} \sqsubseteq \pi_L^\infty = (BA)^\infty$, we can represent it as $L_{i-1} = s_{i-1}(BA)^f p_{i-1}$, for some $f \in \mathbb{N}$, s_{i-1} being a suffix and p_{i-1} being a prefix of BA. Since L is maximally compressed in Opt_O, s_{i-1} falls into the first period BA of l in Opt_O (see Fig. 5). Consider string l' obtained from string $\text{merge}(l, q)$ by shifting q to the right by one period BA, as shown in Fig. 5. If $|\text{ov}(l, q)| < |\pi_L|$, then l' is constructed as the concatenation of a string $l'' \sqsubseteq (\pi_L)^\infty$ and q, such that l is a prefix of l'' and $|l'| = |\text{merge}(l, q)| + |\pi_L|$.

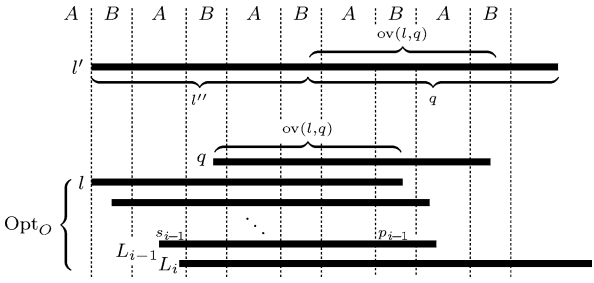


Fig. 5 The situation in the proof of Lemma 6

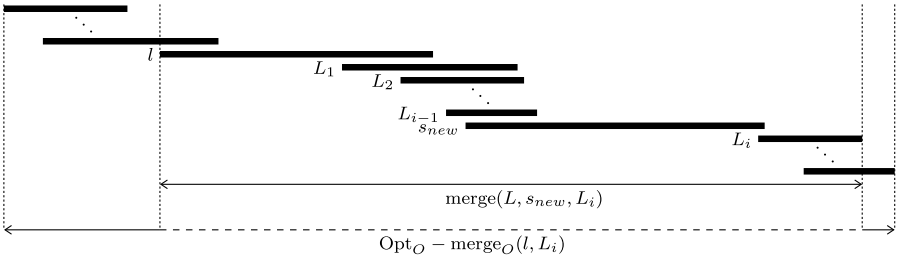


Fig. 6 The solution Sol_2

The string L must be a prefix of l'. Otherwise, in Opt_O, string L_{i-1} ends more than $|BA|$ away from the end of l, and this implies l being a substring of L_{i-1} (note that s_{i-1} falls into the first period BA of l in Opt_O). Therefore $|\mathrm{merge}(L, q)| \leq |l'| = |\mathrm{merge}(l, q)| + |\pi_L|$.

Choosing $q = \mathrm{merge}(s_{new}, L_i)$, the claim of the lemma follows immediately from (7). □

We are now ready to prove the claimed approximation ratio of 11/6 for the case when L_i follows s_{new} in Opt_N.

Lemma 7 *Under Assumptions* 1, 2, 3, 4, 5, *and* 6, *and if* L_i *follows* s_{new} *in* Opt_N, ONECUT *is an* 11/6*-approximation algorithm for SCS+.*

Proof We consider the solution obtained by inserting s_{new} before L_i in Opt_O, that is,

$$\mathrm{Sol}_2 = \mathrm{merge}(L_{j+1}, \ldots, L_{i-1}, s_{new}, L_i, \ldots, L_j)$$

(see Fig. 6). By Lemma 6, we can bound the length of the middle part of Sol_2 in the following way:

$$|\mathrm{merge}(L, s_{new}, L_i)| \leq |\mathrm{merge}(l, s_{new}, L_i)| + |\pi_L| \leq |\mathrm{Opt}_N| + |\pi_L|. \tag{8}$$

The bound for Sol_2 follows from Assumptions 4, 6, and (8):

$$|\mathrm{Sol}_2| \leq |\mathrm{Opt}_O| - |\mathrm{merge}_O(L, L_i)| + |\mathrm{merge}(L, s_{new}, L_i)|$$

$$\leq |\mathrm{Opt}_N| - |l| + |\mathrm{Opt}_N| + |\pi_L|$$
$$\leq 2|\mathrm{Opt}_N| + \frac{1}{6}|\mathrm{Opt}_N| - |\gamma_L| - \frac{1}{3}|\mathrm{Opt}_N| \leq \frac{11}{6}|\mathrm{Opt}_N|. \qquad \square$$

Thus, we can make the following assumption for our final case. (In the case where $L_i = \lambda$, we may assume that L_i follows s_{new} in Opt_N.)

Assumption 7 *L_i is non-empty and it precedes s_{new} in Opt_N.*

For dealing with the remaining case, we first need to bound the length of the overlap of L_{i-1} with L_i.

Lemma 8 *Under Assumptions 1, 2, 3, 5, 6, and 7,*

$$|\mathrm{ov}_O(L_{i-1}, L_i)| \leq |\pi_L| + |\gamma_L|.$$

Proof According to Assumptions 5 and 7, the case we are analyzing here is that both L_1 and L_i are placed before l in Opt_N. Recall that, by definition, $L_i \not\sqsubseteq \pi_L^\infty$. Since $\mathrm{merge}_N(L_1, l) \sqsubseteq \pi_L^\infty$ (see (6)), string L_i cannot be placed between L_1 and l in Opt_N. Hence, L_i is the first of these three strings to appear in Opt_N. Since $l, L_1, \ldots, L_{i-1} \sqsubseteq \pi_L^\infty$, also $L \sqsubseteq \pi_L^\infty$.

If $|\mathrm{ov}_O(L_{i-1}, L_i)| < |\mathrm{pref}_N(L_i, L_1)|$, the claim follows immediately since $\mathrm{pref}_N(L_i, L_1) \sqsubseteq \gamma_L$. Thus, we may assume in the following that $|\mathrm{ov}_O(L_{i-1}, L_i)| \geq |\mathrm{pref}_N(L_i, L_1)|$. Let $Q := \mathrm{pref}_N(L_i, L_1)$ and let P and R be such that $QP := \mathrm{ov}_O(L_{i-1}, L_i)$ and $QPR := L_i$ (see Fig. 7).

For the sake of contradiction, assume $|QP| > |\gamma_L| + |\pi_L|$. Note that $|Q| \leq |\gamma_L|$, and thus $|P| > |\pi_L|$. Since $QP \sqsubseteq L_{i-1} \sqsubseteq \pi_L^\infty$, it must hold that $QP \sqsubseteq \pi_L^\infty$. Let α be the suffix of π_L which is a prefix of P. Let β be a prefix of π_L which starts in P where α ends, such that $|\alpha\beta| = |\pi_L|$. This implies $\beta\alpha = \pi_L$, and $\alpha\beta$ is a prefix of P. It follows that Q ends with β or with a suffix $\overline{\beta}$ of β. Thus, $Q = \overline{\beta}$ or $Q = \overline{\beta\alpha}(\beta\alpha)^q\beta$ for some suffix $\overline{\beta\alpha}$ of $\beta\alpha$ and some $q \in \mathbb{N}$.

Now note that, since $Q = \mathrm{pref}_N(L_i, L_1)$ and $QPR = L_i$, PR has to be a prefix of L_1. Thus, because $\alpha\beta$ is a prefix of P, it is also a prefix of L_1. But π_L is a prefix of L_1, and $|\alpha\beta| = |\pi_L|$, which implies $\alpha\beta = \pi_L$.

Moreover, $L_1 = \pi_L^k\pi_1$ for some $k \in \mathbb{N}$ and some prefix π_1 of π_L. Since PR is a prefix of L_1, we can write $PR = \pi_L^p\pi_2$ for some $p \in \mathbb{N}$ and some prefix π_2 of π_L. Thus, $L_i = QPR$ can take one of the following two forms: either $L_i = \overline{\beta}\pi_L^p\pi_2$ or $L_i = \overline{\beta\alpha}(\beta\alpha)^q\beta\pi_L^p\pi_2 = \overline{\beta\alpha}\beta(\alpha\beta)^q\pi_L^p\pi_2$, where $\overline{\beta\alpha}$ is a suffix of $\beta\alpha$. In both cases, $L_i \sqsubseteq \pi_L^\infty$, contradicting the definition of L_i. $\square$

In the final case of the proof, as presented in Lemma 9, we use Assumptions 1 to 7 and Lemma 8 to prove our claim for all remaining situations not previously dealt with.

Lemma 9 *Under Assumptions 1, 2, 3, 5, 6, and 7,* OneCut *provides an 11/6-approximation ratio for SCS+.*

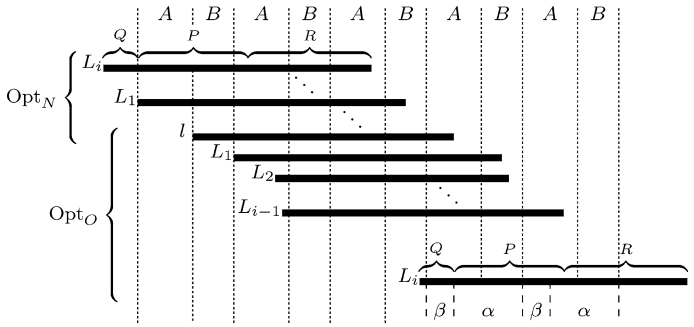


Fig. 7 Illustration of the proof of Lemma 8

Proof Again, consider solution Sol_2. Since $L = \mathrm{merge}_O(l, L_{i-1})$, it follows that

$$\begin{aligned}
|\mathrm{merge}_O(l, L_i)| &= |\mathrm{merge}_O(\mathrm{merge}_O(l, L_{i-1}), L_i)| \\
&= |\mathrm{merge}_O(L, L_i)| \\
&= |L| + |L_i| - |\mathrm{ov}_O(L, L_i)| \\
&= |L| + |L_i| - |\mathrm{ov}_O(L_{i-1}, L_i)| \quad \text{(by substring-freeness)}.
\end{aligned}$$

Due to Lemma 6, we obtain the following bound for Sol_2 (see Fig. 6):

$$\begin{aligned}
\mathrm{Sol}_2 &\leq |\mathrm{Opt}_O| - |\mathrm{merge}_O(l, L_i)| + |\mathrm{merge}(L, s_{new}, L_i)| \\
&\underset{\text{Lemma 6}}{\leq} |\mathrm{Opt}_N| - |\mathrm{merge}_O(l, L_i)| + |\mathrm{merge}(l, s_{new}, L_i)| + |\pi_L| \\
&\leq |\mathrm{Opt}_N| - |L| - |L_i| + |\mathrm{ov}_O(L_{i-1}, L_i)| + |l| + |s_{new}| \\
&\quad - |\mathrm{ov}(l, s_{new})| + |L_i| - |\mathrm{ov}(s_{new}, L_i)| + |\pi_L| \\
&\leq |\mathrm{Opt}_N| - |l| - |L_i| + |\mathrm{ov}_O(L_{i-1}, L_i)| + |l| + |s_{new}| \\
&\quad - |\mathrm{ov}(l, s_{new})| + |L_i| - |\mathrm{ov}(s_{new}, L_i)| + |\pi_L| \\
&\leq |\mathrm{Opt}_N| - |\mathrm{ov}(l, s_{new})| + |\pi_L| + |s_{new}| + |\mathrm{ov}_O(L_{i-1}, L_i)|.
\end{aligned}$$

By applying Lemma 8 and using Assumption 6, we obtain the following bound:

$$\begin{aligned}
|\mathrm{Sol}_2| &\leq |\mathrm{Opt}_N| + |s_{new}| - |\mathrm{ov}(l, s_{new})| + 2|\pi_L| + |\gamma_L| \\
&\leq \frac{4}{3}|\mathrm{Opt}_N| + |s_{new}| - |\gamma_L| - |\mathrm{ov}(l, s_{new})| \\
&\leq \frac{4}{3}|\mathrm{Opt}_N| + |s_{new}| - |\mathrm{ov}(l, s_{new})|.
\end{aligned}$$

Now, Assumption 3 gives the bound $|\mathrm{Sol}_2| \leq \frac{11}{6}|\mathrm{Opt}_N|$. □

The lemmata above directly imply that indeed, in any case, Algorithm 1 (OneCut) provides an 11/6 approximation. This completes the proof of Theorem 6. □

6 Lower Bounds for Cutting Algorithms

This section deals with lower bounds for Algorithm 1 and more general strategies which are obtained by increasing the number of cuts allowed.

6.1 Lower Bounds for Algorithm 1

First, we now show that the analysis in the proof of Theorem 6 is tight, i.e., there exist instances of SCS+ for which OneCut cannot achieve an approximation ratio strictly better than 11/6.

Theorem 7 *Algorithm* OneCut *cannot achieve an* $(\frac{11}{6} - \varepsilon)$*-approximation, for any* $\varepsilon > 0$.

Proof For any $n \in \mathbb{N}$, we construct an input instance that consists of the following strings:

$$S_O = \{\vdash, xa^{n+2}x, a^{n+1}xa^{n+1}, a^n xa^{n+1}xa^n, b^n yb^{n+1}yb^n, b^{n+1}yb^{n+1}, yb^{n+2}y, \dashv\}.$$

Obviously, arranging the strings in the order as presented forms an optimal solution Opt_O of length $6n + \mathcal{O}(1)$:

$$\begin{array}{cccccccccc}
 & a^n & & x\ a^{n+1} & x\ a^n & & b^n & y\ b^{n+1} & y\ b^n & \\
 & a^{n+1} & & x\ a^{n+1} & & & & b^{n+1} & y\ b^{n+1} & \\
x\ a^{n+2} & x & & & & & & & y\ b^{n+2} & y \\
\vdash & & & & & & & & & \dashv
\end{array}$$

The corresponding superstring is $\vdash xa^{n+2}xa^{n+1}xa^n b^n yb^{n+1}yb^{n+2}y\dashv$. Let

$$s_{new} := b^{n-1}yb^{n+1}yb^n \# a^n xa^{n+1}xa^{n-1}.$$

It is easy to see that there is a solution for $S_N = S_O \cup \{s_{new}\}$ which has asymptotically the same length as Opt_O:

$$\begin{array}{cccccccccc}
b^{n-1} & y\ b^{n+1} & y\ b^n & \# & a^n & x\ a^{n+1} & x\ a^{n-1} & & & \\
b^n & y\ b^{n+1} & y\ b^n & & a^n & x\ a^{n+1} & x\ a^n & & & \\
b^{n+1} & y\ b^{n+1} & & & & a^{n+1} & x\ a^{n+1} & & & \\
y\ b^{n+2} & y & & & & & x\ a^{n+2} & x & & \\
\vdash & & & & & & & & \dashv &
\end{array}$$

Applying algorithm OneCut for inserting s_{new} into the instance when Opt_O is given, however, does not find a common superstring that is shorter than $11n + \mathcal{O}(1)$ symbols.

Here, the crucial observation is that all strings in S_O need to be rearranged to construct Opt_N (which then means that no information is gained by the given additional knowledge). Therefore, 7 cuts are necessary to be optimal. Finally, we easily verify that $|\mathrm{Opt}_N| = 6n + \mathcal{O}(1)$. □

6.2 Lower Bounds for k-Cut Algorithms

It seems natural to consider an algorithm k-Cut that is allowed to cut the given instance Opt_O at most k times and, after the cutting, rearranges the $k+1$ parts together with s_{new} in an optimal way. In terms of running time, we make the following observations. Following the same strategy as OneCut, k-Cut computes all pairwise overlaps of the m strings and stores them in a suffix tree which can be done in time $\mathcal{O}(n \cdot m)$, where n is the total length of all strings of the input. Note that there are exactly $\binom{m-1}{k}$ possibilities to cut Opt_O at k places. The resulting $k+1$ strings and s_{new} can be arranged in $(k+2)!$ different ways. Measuring the length of each common superstring obtained in this way can be done in $\mathcal{O}(k)$ time. We conclude that the running time of k-Cut is

$$\mathcal{O}(n \cdot m) + \binom{m-1}{k} \cdot (k+2)! \cdot \mathcal{O}(k)$$

and therefore in

$$\mathcal{O}(n \cdot m + m^k \cdot (k+3)!).$$

Although the approximation ratio can be expected to improve with an increasing number of cuts, a formal analysis of the k-Cut algorithm appears to be technically very complex, thus we leave it as an open problem here.

We are, however, able to bound the approximation ratio of this k-Cut algorithm from below.

To begin with, note that the algorithm 1-Cut that (like OneCut) cuts exactly one place, but is allowed to rearrange the two resulting strings together with s_{new} arbitrarily, as well as the algorithm 2-Cut do not improve over OneCut, when dealing with an input instance as constructed in Sect. 6.1: a simple analysis shows that cutting the old instance at least three times is necessary to improve over $11n + \mathcal{O}(1)$. We easily verify that there are exactly 7 different ways to cut the given instance and thus $\binom{7}{2} = 21$ different cut possibilities all of which do not give something strictly better than $11n + \mathcal{O}(1)$.

As a next step, we now consider the general case of an algorithm k-Cut. The hard examples we are going to build all follow the same idea as the instances used in Sect. 6.1. The set S_O consists of $k+3$ strings. While s_{new} does not fit into Opt_O at any position, merging the given strings from S_O in reverse order compared to the given optimal solution Opt_O, gives another optimal solution for S_O that can easily be extended to the unique optimal solution for S_N. This complete rearrangement of the strings requires at least k cuts.

For $k = 3$, consider the following instance (again, every line contains one string of the input).

$$\begin{array}{l}
\vdash \\
\quad x\ a^n\ x\ a^2 \\
\qquad a^n\ x\ a^{n+1}\ x\ a^n \\
\qquad\quad a^{n+1}\ x\ a^{n+2}\ x\ a^{n+1} \\
\qquad\qquad a^{n+2}\ x\ a^{n+2}\ xa \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \dashv
\end{array}$$

The strings of this instance form the set S_O and the given order specifies a shortest common superstring Opt_O for S_O.

Let $s_{new} = \#a^{n+2}xa^{n+2}x$ be the added string. Then, a new optimal solution Opt_N for $S_N = S_O \cup \{s_{new}\}$ is

$$
\begin{array}{l}
\vdash \\
\quad \#\, a^{n+2}\; x\; a^{n+2}\; x \\
\quad\;\; a^{n+2}\; x\; a^{n+2}\; x\; a \\
\quad\;\; a^{n+1}\; x\; a^{n+2}\; x\; a^{n+1} \\
\qquad\quad a^{n} \quad x\; a^{n+1}\; x\; a^{n} \\
\qquad\qquad\qquad\quad x\; a^{n}\; x\; a^{2} \\
\qquad\qquad\qquad\qquad\quad \dashv .
\end{array}
$$

It is clear that any solution has to contain the substrings xa^nx and $xa^{n+1}x$. Furthermore, due to s_{new}, there have to be two disjoint substrings a^{n+2}. Therefore, all possible solutions have a length of more than $4n$. By distinguishing all cases, it is clear that the only possibility to achieve an optimal solution (which has length $4n + 14$) requires all five possible cuts in Opt_O. Four cuts are sufficient for getting a solution of length $5n + 15$ by omitting the cut between $\vdash$ and xa^nxa^2. All solutions with at most three cuts have a length of at least $6n + 17$.

For the general case, we show the following lower bound.

Theorem 8 *For any $k \geq 3$ and any arbitrarily small $\varepsilon > 0$, there exists an input instance of SCS+ for which the algorithm k-*Cut *is no better than $(1 + \frac{2}{k+1} - \varepsilon)$-approximative.*

Proof Let $s_{new} = \#a^{n+k-1}xa^{n+k-1}x$. Let $w_i = a^{n+i}xa^{n+i+1}xa^{n+i}$, for $0 \leq i \leq k-2$, and $w_{k-1} = s_{new}$. Then we define

$$S_O := \{w_i \mid 0 \leq i \leq k-2\} \cup \{xa^nxa^2, a^{n+k-1}xa^{n+k-1}xa, \vdash, \dashv\}$$

and

$$S'_N := \{w_i \mid 0 \leq i \leq k-2\} \cup \{s_{new}\}.$$

We denote the length of a shortest common superstring for S'_N by $|\mathrm{Opt}_{S'_N}|$.

Observe that the unique shortest common superstring for S'_N is merge$(w_{k-1}, w_{k-2}, \ldots, w_0)$. The following lemma shows that this order of strings is preserved even by all not too long suboptimal superstrings.

Lemma 10 *For $k \geq 3$, any common superstring for S'_N with length less than $|\mathrm{Opt}_{S'_N}| + 2n$ contains the strings from S'_N in the order $w_{k-1}, w_{k-2}, \ldots, w_0$.*

Proof We prove by induction on i that the strings w_i and w_{i-1} have to appear consecutively in any common superstring obeying the given length bound in the order (w_i, w_{i-1}). For this, we need the following auxiliary claim:

The partial substring s_i of the common superstring containing $w_{k-1}, w_{k-2}, \ldots, w_i$ is

$$s_i := \#a^{n+k-1}xa^{n+k-1}z_{k-1}xa^{n+k-2}z_{k-2}xa^{n+k-3}z_{k-3}x\ldots xa^{n+i+1}z_{i+1}xa^{n+i}, \quad (9)$$

where $z_l \in \{\lambda\} \cup \{xa^j \mid j \geq 0\}$, for $i < l \leq k-1$.

Intuitively speaking, the substring z_{l+1} models the possibility of having a non-maximal overlap between two consecutive strings w_{l+1} and w_l.

We are now ready to prove the claimed order of the strings and the validity of (9) by induction on i from $k-2$ downwards. For the induction basis, consider the case where $i = k-2$. We now distinguish the two cases whether the strings w_{k-2} and s_{new} are consecutive or are not. If they are consecutive, suppose on the contrary that s_{new} is on the right-hand side of w_{k-2}. But then, due to the special symbol # at the beginning of s_{new}, the left-hand side of s_{new} does not overlap with the right-hand side of w_{k-2}.

In any solution where w_{k-2} is on the left-hand side of s_{new}, there are at least 5 disjoint occurrences of the infix a^{n+k-2}, and thus each such solution is at least $2n$ symbols too long. Therefore, we can conclude that w_{k-2} is on the right-hand side of s_{new}, which satisfies the invariant.

If, however, s_{new} and w_{k-2} are not consecutive, then the infixes $xa^{n+l}x$ of the remaining strings prevent that s_{new} and w_{k-2} overlap. Therefore, any resulting common superstring contains at least five disjoint substrings a^{n+k-2}, two from s_{new} and three from w_{k-2}. Any common superstring has to contain all substrings $xa^{n+l}x$ for $l \in \{0, 1, \ldots, k-3\}$. Easily, these substrings are pairwise disjoint and none of them overlaps with any of the five substrings a^{n+k-2}. Hence, the minimal length of a common superstring containing these infixes is at least $|\mathrm{Opt}_{S'_N}| + 2n$.

We continue with the induction step. To this end, we show that, if the claimed invariant (9) holds for all values greater than i, it also holds for i.

The overlapping strings w_j for $j > i$ form the superstring

$$s_{i+1} := \#a^{n+k-1}xa^{n+k-1}z_{k-1}xa^{n+k-2}z_{k-2}xa^{n+k-3}z_{k-3}x\ldots xa^{n+i+2}z_{i+2}xa^{n+i+1}$$

according to the induction hypothesis. Similar as in the proof of the induction basis, we distinguish two cases according to whether w_i and s_{i+1} are consecutive or not.

In the first case, the same arguments as above show that w_i has to be on the right-hand side of s_{i+1}. If the two strings are not consecutive, again we can exclude that they overlap. Therefore, since s_{i+1} contains $1 + (n+k-1-(n+i+1-1)) = k-i$ disjoint substrings a^{n+i}, there are $k-i+3$ disjoint substrings a^{n+i} in any common superstring that is formed this way. Since the remaining i substrings $xa^{n+i-l}x$ for $i \geq l \geq 1$ also have to be in any superstring that is formed this way, the minimal length of a common superstring containing these infixes is more than $|\mathrm{Opt}_{S'_N}| + 2n$. □

We now consider the following given optimal solution for the SCS+ instance S_O as defined above:

$$
\begin{array}{l}
\vdash \\
\quad x\ a^n\ x\ a^2 \\
\qquad a^n\ x\ a^{n+1}\ x\ a^n \\
\qquad\quad a^{n+1}\ x\ a^{n+2}\ x\ a^{n+1} \\
\qquad\qquad a^{n+2}\ x\ a^{n+3}\ x\ a^{n+2} \\
\qquad\qquad\quad a^{n+3}\ x\ a^{n+4}\ x\ a^{n+3} \\
\qquad\qquad\qquad\qquad \ddots \\
\qquad\qquad\qquad\qquad\quad a^{n+k-3}\ x\ a^{n+k-2}\ x\ a^{n+k-3} \\
\qquad\qquad\qquad\qquad\qquad a^{n+k-2}\ x\ a^{n+k-1}\ x\ a^{n+k-2} \\
\qquad\qquad\qquad\qquad\qquad\quad a^{n+k-1}\ x\ a^{n+k-1}\ xa \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \dashv
\end{array}
$$

where, as above, each line presents one string from S_O and the corresponding shortest common superstring is

$$\mathrm{Opt}_O = \vdash xa^n xa^{n+1} xa^{n+2} x \dots xa^{n+k-1} xa^{n+k-1} xa \dashv .$$

Let $s_{new} = \#a^{n+k-1}xa^{n+k-1}x$ be the inserted string such that $S_N = S_O \cup \{s_{new}\}$. It is easy to see that

$$\mathrm{Opt}_N = \vdash \#a^{n+k-1}xa^{n+k-1}xa^{n+k-2}xa^{n+k-3}x \dots xa^{n+1}xa^n xa^2 \dashv$$

is a shortest common superstring for S_N. Note that, in Opt_N, the ordering of $w_0, w_1, \dots, w_{k-2}$ has been reversed compared to Opt_O.

Since S_N contains S'_N, because of Lemma 10, any solution that does not contain the strings $w_0, w_1, \dots, w_{k-2}$ in the order as in Opt_N has a length of at least $|\mathrm{Opt}_{S'_N}| + 2n$. The rearrangement cannot be done without separating the strings w_0 to w_{k-2} with $k-2$ cuts. Additionally, a cut between $xa^n xa^2$ and w_0 is necessary since otherwise there are at least $2n$ excessive symbols between w_1 and w_0.

Similarly, we need a cut between w_{k-1} and $a^{n+k-1}xa^{n+k-1}xa$. Moreover, without a cut between $a^{n+k-1}xa^{n+k-1}xa$ and $\dashv$, any solution contains at least 5 infixes a^{n+k-2}, whereas only 3 such infixes are necessary.

Thus, any solution obtained with at most k cuts has a length of at least $|\mathrm{Opt}_{S'_N}| + 2n \geq (k+3)n$, whereas Opt_N is composed of three special markers, $k+1$ symbols x and

$$(k+1)n + (k-1) + \sum_{i=0}^{k-1} i + 2$$

symbols a, which sums up to the length $(k+1)n + 5 + 3k/2 + k^2/2 < (k+1)n + (k+1)^2$ (remember that $k \geq 3$).

Therefore, we obtain

$$\frac{(k+3)n}{(k+1)n + (k+1)^2} = 1 + \frac{2}{k+1} - \frac{k+3}{n+k+1}$$

as a lower bound on the approximation ratio achieved by k-Cut, and thus, when choosing $n \geq \varepsilon^{-1}(k+3) - k - 1$, the lower bound satisfies the claim of the theorem. □

7 Conclusion

In this paper, we considered the *shortest common superstring* reoptimization problem, addressing the insertion and the deletion of strings as reoptimization variants. We showed both variants to be NP-hard and we presented an iterative polynomial-time algorithm that achieves an approximation ratio arbitrarily close to 1.6 for SCS+ and arbitrarily close to 13/7 for SCS–. The interest in the algorithm is twofold, because besides achieving a good approximation ratio for the two reoptimization problems, its core is to exploit the existence of a long string within the modified input instance. This concept is applicable universally, i.e., for any SCS instance that contains a long string, we are able to improve the ratio of any SCS approximation algorithm.

The drawback of the algorithm, however, is its runtime. Consequently, we presented a second strategy for SCS+, the ONECUT algorithm, which achieves an approximation ratio of 11/6 and runs in quadratic time. We showed that our analysis of the ONECUT algorithm is tight.

Furthermore, we introduced a straightforward generalization of ONECUT and gave lower bounds on its approximation ratio. It also seems worthwhile investigating different types of local modifications for SCS reoptimization.

References

1. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks **42**(3), 154–159 (2003)
2. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the 0-1 knapsack problem. Technical Report 267, University of Brescia (2006)
3. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R.V. (eds.) Proc. of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006). Lecture Notes in Computer Science, vol. 4059, pp. 196–207. Springer, Berlin (2006)
4. Bilò, D., Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Widmayer, P., Zych, A.: Reoptimization of Steiner trees. In: Gudmundsson, J. (ed.) Proc. of the 11th Scandinavian Workshop on Algorithm Theory (SWAT 2008). Lecture Notes in Computer Science, vol. 5124, pp. 258–269. Springer, Berlin (2008)
5. Bilò, D., Widmayer, P., Zych, A.: Reoptimization of weighted graph and covering problems. In: Bampis, E., Skutella, M. (eds.) Proc. of the 6th International Workshop on Approximation and Online Algorithms (WAOA 2008). Lecture Notes in Computer Science, vol. 5426, pp. 201–213. Springer, Berlin (2009)
6. Böckenhauer, H.-J., Bongartz, D.: Algorithmic Aspects of Bioinformatics. Natural Computing Series, Springer, Berlin (2007)
7. Böckenhauer, H.-J., Komm, D.: Reoptimization of the metric deadline TSP. In: Ochmanski, E., Tyszkiewicz, J. (eds.) Proc. of the 33th International Symposium on Mathematical Foundations of Computer Science (MFCS 2008). Lecture Notes in Computer Science, vol. 5162, pp. 156–167. Springer, Berlin (2008)
8. Böckenhauer, H.-J., Forlizzi, L., Hromkovič, J., Kneis, J., Kupke, J., Proietti, G., Widmayer, P.: Reusing optimal TSP solutions for locally modified input instances (extended abstract). In: Navarro, G., Bertossi, L.E., Kohayakawa, Y. (eds.) Proc. of the 4th IFIP International Conference on Theoretical Computer Science (TCS 2006). IFIP, vol. 209, pp. 251–270. Springer, New York (2006)
9. Böckenhauer, H.-J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008). Lecture Notes in Computer Science, vol. 4910, pp. 50–65. Springer, Berlin (2008)

10. Böckenhauer, H.-J., Hromkovič, J., Královič, R., Mömke, T., Rossmanith, P.: Reoptimization of Steiner trees: Changing the terminal set. Theor. Comput. Sci. **410**(36), 3428–3435 (2009)
11. Escoffier, B., Milanič, M., Paschos, V.T.: Simple and fast reoptimizations for the Steiner tree problem. Algorithmic Oper. Res. **4**(2), 86–94 (2009)
12. Gallant, J., Maier, D., Storer, J.A.: On finding minimal length superstrings. J. Comput. Syst. Sci. **20**(1), 50–58 (1980)
13. Kaplan, H., Shafrir, N.: The greedy algorithm for shortest superstrings. Inf. Process. Lett. **93**(1), 13–17 (2005)
14. Kaplan, H., Lewenstein, M., Shafrir, N., Sviridenko, M.: Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. J. ACM **52**(4), 602–626 (2005)
15. Schäffter, M.W.: Scheduling with forbidden sets. Discrete Appl. Math. **72**(1–2), 155–166 (1997)
16. Setubal, C., Meidanis, J.: Introduction to Computational Molecular Biology. Natural Computing Series, PWS Publishing Company, Boston (1997)
17. Sweedyk, Z.: A $2\frac{1}{2}$-approximation algorithm for shortest superstring. SIAM J. Comput. **29**(3), 954–986 (2000)
18. Tarhio, J., Ukkonen, E.: A greedy approximation algorithm for constructing shortest common superstrings. Theor. Comput. Sci. **57**(1), 131–145 (1988)
19. van Hoesel, S., Wagelmans, A.: On the complexity of postoptimality analysis of $0/1$ programs. Discrete Appl. Math. **91**(1–3), 251–263 (1999)
20. Vassilevska, V.: Explicit inapproximability bounds for the shortest superstring problem. In: Jedrzejowicz, J., Szepietowski, A. (eds.) Proc. of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS 2005). Lecture Notes in Computer Science, vol. 3618, pp. 793–800. Springer, Berlin (2005)