On the Greedy Algorithm for the Shortest Common Superstring Problem with Reversals

Gabriele Fici

Dipartimento di Matematica e Informatica, Università di Palermo, Italy

Tomasz Kociumaka

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Jakub Radoszewski

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland and Department of Informatics, King's College London, UK

Wojciech Rytter

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Tomasz Waleń

Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, Poland

Abstract

We study a variation of the classical Shortest Common Superstring (SCS) problem in which a shortest superstring of a finite set of strings S is sought containing as a factor every string of S *or* its reversal. We call this problem Shortest Common Superstring with Reversals (SCS-R). This problem has been introduced in *T. Jiang, M. Li, D.-Z. Du. A Note on Shortest Superstrings with Flipping. Inform. Process. Lett. 44: 195–199*, where the authors designed a greedy-like algorithm with length approximation ratio 4. In this paper, we show that a natural adaptation of the classical greedy algorithm for SCS has (optimal) *compression ratio* $\frac{1}{2}$, i.e., the sum of the overlaps in the output string is at least half the sum of the overlaps in an optimal solution. We also provide a linear-time implementation of our algorithm.

Keywords: Shortest Common Superstring, reversal, greedy algorithm.

1. Introduction

The Shortest Common Superstring (SCS) problem is a classical combinatorial problem on strings with applications in many domains, e.g. DNA fragment assembly, data compression, etc. (see [6] for a recent survey). It consists, given a finite set of strings S, in finding a shortest string containing

Email addresses: gabriele.fici@unipa.it (Gabriele Fici), kociumaka@mimuw.edu.pl (Tomasz Kociumaka), jrad@mimuw.edu.pl (Jakub Radoszewski), rytter@mimuw.edu.pl (Wojciech Rytter), walen@mimuw.edu.pl (Tomasz Waleń)

as factors (substrings) all the strings in S. The decision version of the problem is known to be NP-complete [13, 5, 4], even under several restrictions on the structure of S (see again [6]). However, a particularly simple greedy algorithm introduced by Gallant in his Ph.D. thesis [5] is widely used in applications since it has very good performance in practice (see for instance [12] and references therein). It consists in repetitively replacing a pair of strings with maximum overlap with the string obtained by overlapping the two strings, until one string remains. The greedy algorithm can be implemented using Aho-Corasick automaton in $\mathcal{O}(n)$ randomized time (with hashing on the symbols of the alphabet) or $\mathcal{O}(n \min(\log m, \log |\Sigma|))$ deterministic time (see [17]), where n is the sum of the lengths of the strings in S and m its cardinality.

The approximation of the greedy algorithm is usually measured in two different ways: one consists in taking into account the *approximation ratio* (also known as the *length ratio*) k_g/k_{min}, where k_g is the length of the output string of greedy and k_{min} the length of a shortest superstring, the other consists in taking into account the *compression ratio* $(n - k_g)/(n - k_{min})$.

For the approximation ratio, Turner [16] proved that there is no constant $c < 2$ such that $k_g/k_{min} \leq c$. The *greedy conjecture* states that this approximation ratio is in fact 2 [1]. The best bound currently known is 3.5 due to Kaplan and Shafrir [10]. Algorithms with better approximation ratio are known; the best one is due to Mucha, with an approximation ratio of $2\frac{11}{23}$ [14].

For the compression ratio, Tarhio and Ukkonen [15] proved that $(n - k_g)/(n - k_{min}) \geq 1/2$ and this bound is tight, since it is achieved for the set $S = \{ab^h, b^h a, b^{h+1}\}$ when greedy makes the first choice merging the first two strings together.

Let us formally state the SCS problem:

Shortest Common Superstring (*SCS*)
Input: strings $S = \{s_1, \ldots, s_m\}$ of total length n.
Output: a shortest string u that contains s_i for each $i = 1, \ldots, m$ as a factor.

Several variations of SCS have been considered in literature. For example, shortest common superstring problem with reverse complements was considered in [11]. In this setting the alphabet is $\Sigma = \{\mathtt{a}, \mathtt{t}, \mathtt{g}, \mathtt{c}\}$ and the complement of a string s is $\bar{s}^R$, where $\bar{}$ is defined by $\bar{\mathtt{a}} = \mathtt{t}$, $\bar{\mathtt{t}} = \mathtt{a}$, $\bar{\mathtt{g}} = \mathtt{c}$, $\bar{\mathtt{c}} = \mathtt{g}$, and t^R denotes the reversal of t, that is the string obtained reading t backwards.In particular, this problem was shown to be NP-complete.

Other variations of the SCS problem can be found in [8, 3, 7, 2].

In this paper, we address the problem of searching for a string u of minimal length such that for every $s_i \in S$, u contains as a factor s_i *or* its reversal s_i^R.

Shortest Common Superstring with Reversals (*SCS-R*)
Input: strings $S = \{s_1, \ldots, s_m\}$ of total length n.
Output: a shortest string u that contains for each $i = 1, \ldots, m$ at least one of the strings s_i or s_i^R as a factor.

For example, if $S = \{aabb, aaac, abbb\}$, then a solution of SCS-R for S is *caaabbb*. Notice that a shortest superstring with reversals can be much shorter than a classical shortest superstring. An extremal example is given by an input set of the form $S = \{ab^h, cb^h\}$.

The SCS-R problem was already considered by Jiang et al. [9], who observed that the problem is still NP-hard. We provide a proof of this fact for the sake of completeness (see Proposition 1).

In [9], the authors proposed a greedy 4-approximation algorithm. Here, we show that an adaptation of the classical greedy algorithm can be used for solving the SCS-R problem with an (optimal)

compression ratio $\frac{1}{2}$.

2. Basics and Notation

Let Σ be a finite alphabet. We assume that Σ is linearly sortable, i.e., $\Sigma = \{0, \ldots, n^{\mathcal{O}(1)}\}$. The *length* of a string s over Σ is denoted by $|s|$. The *empty string*, denoted by ε, is the unique string of length zero. A string t *occurs* in a string s if $s = vtz$ for some strings v, z. In this case we say that t is a *factor* of s. In particular, we say that t is a *prefix* of s when $v = \varepsilon$ and a *suffix* of s when $z = \varepsilon$. We say that a factor is *proper* if $|v| + |z| > 0$.

The string s^R obtained by reading s from right to left is called the *reversal* (or *mirror image*) of s. Given a set of strings $S = \{s_1, \ldots, s_m\}$, we define the set $S^R = \{s_1^R, \ldots, s_m^R\}$ and the set $\widetilde{S} = S \cup S^R$.

Given two strings u, v, we define the (maximum) overlap between u and v, denoted by $ov(u, v)$, as the length of the longest suffix of u that is also a prefix of v. Sometimes we abuse the notation and also say that the suffix of u of length $ov(u, v)$ is the overlap of u and v. The overlap operation is associative but not symmetric, because in general $ov(u, v)$ is not equal to $ov(v, u)$. However, it is readily verified that $ov(u, v) = ov(v^R, u^R)$. Additionally, we define $pr(u, v)$ as the prefix of u obtained by removing the suffix of length $ov(u, v)$ and denote $u \otimes v = pr(u, v)v$.

A set of strings S is called *factor-free* if no string in S is a factor of another string in S. We say that S is *reverse-factor-free* if there are no two different strings u, v in S such that u is a factor of v or v^R.

Given a set of strings $S = \{s_1, \ldots, s_m\}$, the SCS problem for S is equivalent to that of finding an optimal (i.e., with maximum total weight) Hamiltonian path π in the *overlap graph* G_S, which is the directed weighted graph (V, E, w) defined by $V = S$, $E = \{(s_i, s_j) \mid i \neq j\}$ and $w_{ij} = ov(s_i, s_j)$ (cf. [15]).

We consider the overlap graph $G_{\widetilde{S}}$. We say that a path π in $G_{\widetilde{S}}$ is *semi-Hamiltonian* if π contains, for every $i = 1, \ldots, m$, the vertex s_i or the vertex s_i^R but not both (if they are different). Hence, a solution to SCS-R problem for S corresponds to a longest semi-Hamiltonian path π in the overlap graph $G_{\widetilde{S}}$. Finally, by $ov(\pi)$ we denote the total weight of edges in the path π.

3. NP-completeness of the SCS-R Problem

Let $\Gamma = \Sigma \cup \{\$, \#\}$, where $\$, \# \notin \Sigma$. Let h be the morphism from Σ^* to Γ^* defined by $h(c) = \$\# c$, for every $c \in \Sigma$. Given $k \geq 1$, we also define the morphism $g_k(c) = c^k$, for every $c \in \Sigma$.

Observation 1. For every nonempty strings u, v, the strings $h(u)$ and $h(v)^R$ have an overlap of length at most one.

Observation 2. For every nonempty strings u, v, one has $|h(g_k(u))| = 3k|u|$, and $ov(h(g_k(u)), h(g_k(v))) = 3k \cdot ov(u, v)$.

Proposition 1. *The decision version of the SCS-R problem is NP-complete.*

Proof. Let $s_1, \ldots, s_m$ be an instance of the SCS problem. Set $y_i = h(g_m(s_i))$, for $i = 1, \ldots, m$. We have the following claim.

Claim 1. There exists a common superstring of $s_1, \ldots, s_m$ of length at most ℓ if and only if there exists a common superstring with reversal of $y_1, \ldots, y_m$ of length at most $3m\ell$.

Proof. ($\Rightarrow$) If u is a common superstring of $s_1, \ldots, s_m$, then $h(g_m(u))$ is a common superstring of $y_1, \ldots, y_m$. Hence, $h(g_m(u))$ is also a common superstring with reversals of $y_1, \ldots, y_m$. If $|u| = p$, then $|h(g_m(u))| = 3mp$.

($\Leftarrow$) Let u be a common superstring with reversals of $y_1, \ldots, y_m$. Let $\pi = z_{i_1}, \ldots, z_{i_m}$ be the sequence of nodes on the path in the overlap graph G that corresponds to u; here $\{i_1, \ldots, i_m\} = \{1, \ldots, m\}$ and each z_i is either y_i or y_i^R. Let us construct a new sequence of nodes π', that first contains all nodes from π of the form y_i in the same order as in π, and then all nodes from π of the form y_i^R, but given in the reverse order and taken without reversal.

Let u' be the common superstring corresponding to π'. By Observation 1, $|u'| \leq |u| + m - 1$.

Note that u' is also an ordinary common superstring (i.e., without reversal) of $y_1, \ldots, y_m$. By Observation 2, $|u'|$ is a multiple of $3m$ and u' corresponds to a common superstring v of $s_1, \ldots, s_m$ of length $|u'|/(3m)$. If $|u| \leq 3m\ell$, then

$$|v| = \frac{|u'|}{3m} \leq \left\lfloor \frac{|u| + m - 1}{3m} \right\rfloor \leq \ell.$$

□

The claim provides a reduction of the decision version of the SCS problem to the decision version of the SCS-R problem. This shows that the latter is NP-hard, hence NP-complete, as it is obviously in NP. □

4. Greedy Algorithm and its Linear Implementation

We define an auxiliary procedure Make-Reverse-Factor-Free(S) that iteratively removes from S every string u such that there is a different string v in S for which u is a factor of v or v^R. It runs until S is reverse-factor-free. Note that for any output S' of the procedure, the following holds: A string is a shortest common superstring with reversals for S' if and only if it is a shortest common superstring with reversals for S.

Example 1. Let $S = \{ab, aaa, aab, baa\}$. Then Make-Reverse-Factor-Free(S) produces $\{aaa, aab\}$ or $\{aaa, baa\}$.

Lemma 2. *Procedure* Make-Reverse-Factor-Free(S) *can be implemented in time linear in the total length of strings in* S.

Proof. Let us introduce two auxiliary procedures defined for a set of strings X. Procedure Remove-Reversals(X) for every pair of strings $u, v \in X$ such that $u = v^R$ leaves exactly one of them in X. Procedure Make-Factor-Free(X) iteratively removes from X every string u such that there is a string v in X for which u is a proper factor of v. We will show that Make-Reverse-Factor-Free(S) can be implemented using these two procedures. First we show that they can be implemented efficiently.

Claim 2. Remove-Reversals(X) can be implemented in time linear in the total length of the strings in X.

Proof. Let $X = \{x_1, \ldots, x_m\}$ and let n be the total length of strings in X. First we construct the set $\widetilde{X}$. For each element $v \in \widetilde{X}$ we store, as $ind(v)$, the index i of the element $x_i \in X$ from which v was constructed (as x_i or x_i^R). Then we sort all the strings in $\widetilde{X}$ using radix sort. This sorting requires only $\mathcal{O}(n)$-time due to the assumption of linearly-sortable alphabet. Then, we

iterate through the sorted set of strings and for each pair u, v of equal strings we store the pair of indices $ind(u)$, $ind(v)$ (assume that $ind(u) \leq ind(v)$). Note that each pair occurs exactly twice apart from the pairs of equal numbers that we can discard (then u is a palindrome). Finally, we sort all the pairs in $\mathcal{O}(m)$-time to remove duplicates and for each pair of indices remove exactly one string from X. In total the procedure takes $\mathcal{O}(n)$-time. □

Ukkonen [17] showed the following result about the preprocessing phase of the greedy algorithm for the ordinary SCS problem:

Claim 3 ([17]). MAKE-FACTOR-FREE(X) can be implemented in time linear in the total length of the strings in X.

We can now proceed to describe an efficient implementation of MAKE-REVERSE-FACTOR-FREE(S). In the first step we replace S with REMOVE-REVERSALS(S). From now on we only need to remove from S all strings u being *proper* factors of strings of the form v or v^R, for $v \in S$. For this, we compute $\widetilde{S}$, storing for each string the index of the string in S it corresponds to, and the set $S' =$ MAKE-FACTOR-FREE($\widetilde{S}$). Note that if we remove u from $\widetilde{S}$ in this procedure, then we also remove u^R. As the result we leave in S all strings that have their counterparts in S'. The whole procedure works in linear time in the total length of the strings in S. □

Now we can assume that S is reverse-factor-free. The Greedy-R algorithm works as follows: while $|S| > 1$, choose $u, v \in \widetilde{S}$ (excluding $u = v$ or $u = v^R$) with biggest overlap, insert in S the string $u \otimes v$ and remove from S the strings among u, v, u^R, v^R that are elements of S; see the pseudocode.

Algorithm *Greedy-R*(S)
Input: a non-empty set of strings S
Output: a superstring of S that approximates a solution of SCS-R problem for S
begin
  $S :=$ MAKE-REVERSE-FACTOR-FREE(S)
  while $|S| > 1$ **do**
    $P := \{(u, v) : u, v \in \widetilde{S}, u \notin \{v, v^R\}\}$
    { S is reverse-factor-free and $|S| > 1$, so $|P| \geq 1$ }
    take $(u, v) \in P$ with the maximal value of $ov(u, v)$
    $S := S \cup \{u \otimes v\}$
    $S := S \setminus \{u, v, u^R, v^R\}$
  return *the only element of* S
end

Observation 3. The length $ov(u, v)$ in the subsequent steps of the Greedy-R algorithm can only decrease. This is due to the greedy characteristic of the approach.

Ukkonen [17] showed that the Greedy algorithm for the original SCS problem can be implemented in linear time. A linear-time implementation of the Greedy-R algorithm is quite similar. We show the following result.

Theorem 3. *Greedy-R algorithm can be implemented in linear time.*

Proof. Let S be the input set of strings. By Lemma 2, we can make S reverse-factor-free in linear time.

Let $S = \{s_1, \ldots, s_m\}$ be the set of remaining strings and n the total length of strings in S. Let us introduce some useful notation. Denote by $\mathit{Pref}(\widetilde{S})$ the set of all different prefixes of strings in $\widetilde{S}$. Each element of this set can be represented as a state of the Aho-Corasick automaton constructed for $\widetilde{S}$, thus using $\mathcal{O}(1)$ space per element. Further, given a string w, denote by $\mathit{PrefSet}(w, \widetilde{S})$, $\mathit{SufSet}(w, \widetilde{S})$ the sets (represented as lists) of strings in $\widetilde{S}$ having w as a prefix, suffix, respectively. Ukkonen [17] used the Aho-Corasick automaton for $\widetilde{S}$ to show the following fact:

Claim 4 ([17]). $\mathit{PrefSet}(w, \widetilde{S})$, $\mathit{SufSet}(w, \widetilde{S})$ for all $w \in \mathit{Pref}(\widetilde{S})$ have total size $\mathcal{O}(n)$ and can be computed in $\mathcal{O}(n)$ time.

At each step of the Greedy-R algorithm we store a collection of disjoint paths in $G_{\widetilde{S}}$ representing all the $\otimes$-operations that were performed so far. In the end of the algorithm this collection becomes a semi-Hamiltonian path in $G_{\widetilde{S}}$. For each $v \in \widetilde{S}$ we remember if it is an endpoint of a path and, if so, what is the other endpoint of the path. In the course of the algorithm we will remove all strings $v \in \widetilde{S}$ that correspond to inner vertices of paths or their reversals (called *redundant* strings) from *PrefSet* and *SufSet* whenever encountered.

Now we show how to simulate the while-loop in the algorithm. By Observation 3, at each step of the loop the value $ov(u, v)$ is not greater than in the previous steps. Instead of simulating the while-loop directly, we will consider all $w \in \mathit{Pref}(\widetilde{S})$ in decreasing length order and for each of them check if there exist two strings $u, v \in \widetilde{S}$ such that:

(a) $u \notin \{v, v^R\}$, and

(b) u, v are endpoints of two different paths, and

(c) w is the overlap of u and v.

If so, by the greedy property we know that $|w|$ is the longest overlap, therefore we merge the two strings (consequently join the two paths).

Let us explain how this can be done efficiently. To iterate through all $w \in \mathit{Pref}(\widetilde{S})$ in decreasing length order we simply traverse all the states of the Aho-Corasick automaton in reverse-BFS order, breaking ties arbitrarily. To check the conditions (a)-(c), we iterate through all pairs of elements $v \in \mathit{PrefSet}(w, \widetilde{S})$ and $u \in \mathit{SufSet}(w, \widetilde{S})$ and verify if they satisfy the conditions. (We assume that the sets already do not contain redundant strings; if not, we simply remove the redundant strings whenever they are encountered, which accounts for only $\mathcal{O}(n)$ in the time complexity.) Note that for each $u \in \mathit{SufSet}(w, \widetilde{S})$ there are at most four elements $v \in \mathit{PrefSet}(w, \widetilde{S})$ that do not form a valid overlap w with u. Indeed, these are u, u^R, x and x^R, where x is the other endpoint of the path that starts with u. Hence, with constant-time overhead, either a pair of strings u, v satisfying (a)-(c) is found and can be merged, or it can be checked that no pair of strings u, v satisfies (a)-(c) for this particular string w and then we can continue iterating through strings in $\mathit{Pref}(\widetilde{S})$. If a pair of strings u, v is found, we start the next search with the same string w, since there could be many such pairs satisfying conditions (a)-(c) for the same string w.

To conclude: in every step of the simulation, in constant time we either find a pair of strings u, v to be merged or discard the given candidate $w \in \mathit{Pref}(\widetilde{S})$. The former situation takes place at most $m - 1$ times and the latter takes place at most n times. The whole algorithm thus works in $\mathcal{O}(n)$-time. □

5. Compression Ratio

In this section we prove that the compression ratio of the Greedy-R algorithm is always at least $\frac{1}{2}$, and that this value is effectively achieved, so the bound is tight.

Let $S = \{s_1, \ldots, s_m\}$ be the input set of strings. We assume that S is already reverse-factor-free. Let $\mathrm{opt}(S)$ be the length of a longest semi-Hamiltonian path in $G = G_{\widetilde{S}}$. Let *pgreedy*(S) denote the length of the semi-Hamiltonian path produced by the Greedy-R algorithm for S. We will show that *pgreedy*$(S) \geq \frac{1}{2}\,\mathrm{opt}(S)$.

In the proof we use as a tool the following fact from [15] (see Lemma 3.1 in [15]):

Lemma 4. *If x_1, x_2, x_3, x_4 are different vertices of G_X for some set of strings X such that*

$$ov(x_1, x_4), ov(x_2, x_3) \leq ov(x_1, x_3)$$

then

$$ov(x_1, x_4) + ov(x_2, x_3) \leq ov(x_1, x_3) + ov(x_2, x_4).$$

We proceed with the following crucial lemma.

Lemma 5. *Let S be a set of strings and let $u, v \in \widetilde{S}$ be two elements for which $ov(u, v)$ is maximal ($u \notin \{v, v^R\}$). Set $OV = ov(u, v)$. Let $\mathrm{opt}(S)$ be the length of a longest semi-Hamiltonian path in G and let $\mathrm{opt}'(S)$ be the length of a longest semi-Hamiltonian path in G that contains the edge (u, v). Then:*

$$\mathrm{opt}'(S) \geq \mathrm{opt}(S) - OV.$$

Proof. We consider the path π corresponding to $\mathrm{opt}(S)$ and show how it can be modified without losing its semi-Hamiltonicity so that the edge (u, v) occurs in the path and the length of the path decreases by at most OV.

Obviously, if π already contains the edge (u, v), nothing is to be done. If both u and v occur in π, we perform transformations as in the proof of a similar fact from [15] related to the ordinary SCS problem. If u occurs in π before v then we select π' as in Fig. 1 and:

$$ov(\pi') \geq ov(\pi) - ov(u, b) - ov(c, v) + ov(u, v) \geq ov(\pi) - ov(u, b) \geq ov(\pi) - OV.$$

Note that both nodes b, c exist (they could be the same node, though). If any of the remaining nodes does not exist, it is simply skipped on the path.

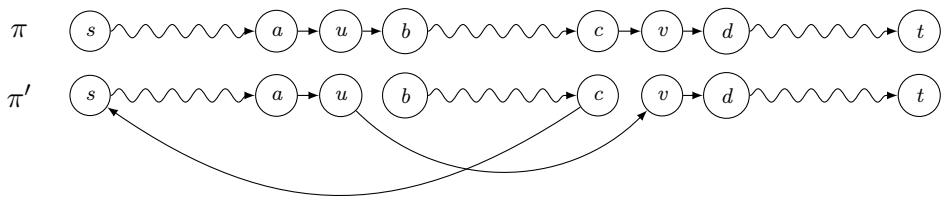

Figure 1: Proof of Lemma 5, first case: u occurs in π before v.

If v occurs before u, then by applying the inequality of Lemma 4 (with $x_1 = u$, $x_2 = a$, $x_3 = v$, $x_4 = d$) we have

$$ov(u, d) + ov(a, v) \leq ov(u, v) + ov(a, d).$$

By this inequality, for the path π' defined in Fig. 2 we have:

$$\begin{aligned} ov(\pi') &\geq ov(\pi) - ov(a,v) - ov(u,d) + ov(u,v) + ov(a,d) - ov(v,b) \\ &\geq ov(\pi) - ov(v,b) \geq ov(\pi) - OV. \end{aligned}$$

As before, if any of the depicted nodes does not exist, we simply skip the corresponding part of the path. In particular, if any of the nodes u, v is an endpoint of the path π, we do not need to use the aforementioned inequality to show that $ov(\pi') \geq ov(\pi) - OV$.

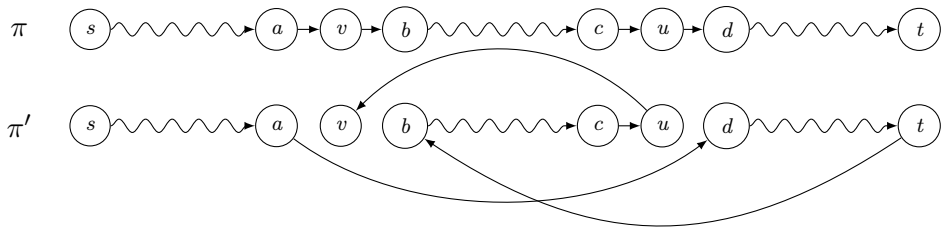

Figure 2: Proof of Lemma 5, second case: u occurs in π after v.

Differently from the original SCS problem considered in [15], it might not be the case that u and v are in π. If none of them is, then π contains both u^R and v^R and by reversing π (that is, reversing all the strings and the edges in the path) we obtain a semi-Hamiltonian path that contains both u and v, which was the case considered before. Thus, we can assume that u and v^R occur in π (the case of u^R and v is symmetric). Again, we have two cases, depending on which of the two strings comes first in π. If u occurs before v^R in π, then we have (note that $ov(c, v^R) = ov(v, c^R)$):

$$ov(\pi') \geq ov(\pi) - ov(u,b) - ov(v^R,d) + ov(u,v) \geq ov(\pi) - ov(u,b) \geq ov(\pi) - OV,$$

see also Fig. 3.

Finally, if v^R occurs before u, then (see Fig. 4):

$$ov(\pi') \geq ov(\pi) - ov(v^R,b) - ov(u,d) + ov(u,v) \geq ov(\pi) - ov(v^R,b) \geq ov(\pi) - OV.$$

This completes the proof of the lemma. □

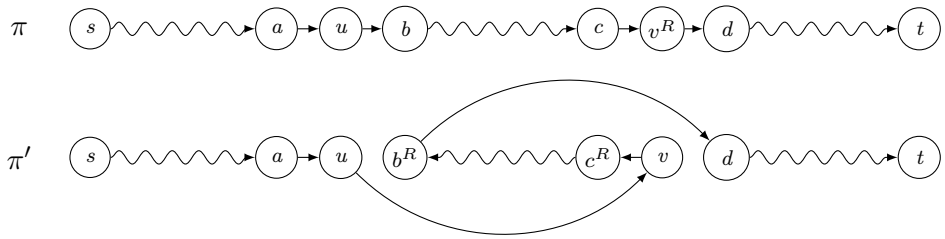

Figure 3: Proof of Lemma 5, third case: u occurs in π before v^R.

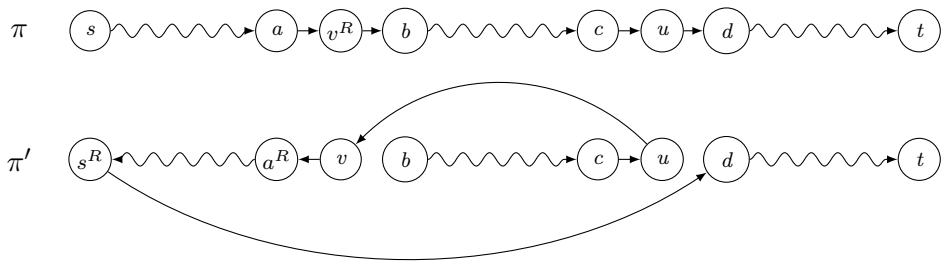

Figure 4: Proof of Lemma 5, third case: u occurs in π after v^R.

To conclude the proof of the compression ratio of the Greedy-R algorithm we use an inductive argument. Assume that $\mathit{pgreedy}(S') \geq \frac{1}{2}\operatorname{opt}(S')$ for all S' such that $|S'| < |S|$. By Lemma 5, for $S' = S \setminus \{u, v, u^R, v^R\} \cup \{u \otimes v\}$:

$$\operatorname{opt}(S) - OV \leq \operatorname{opt}'(S) \leq \operatorname{opt}(S') + OV,$$

so that $\operatorname{opt}(S') + 2OV \geq \operatorname{opt}(S)$. Moreover, $\mathit{pgreedy}(S) = \mathit{pgreedy}(S') + OV$ and, by the inductive hypothesis, $\mathit{pgreedy}(S') \geq \frac{1}{2}\operatorname{opt}(S')$. Consequently:

$$\mathit{pgreedy}(S) = \mathit{pgreedy}(S') + OV \geq \frac{1}{2}(\operatorname{opt}(S') + 2OV) \geq \frac{1}{2}\operatorname{opt}(S).$$

We arrive at the following theorem.

Theorem 6. *Greedy-R has compression ratio* $\frac{1}{2}$.

It turns out that the bound on the compression ratio of the Greedy-R algorithm is tight. It suffices to consider the set of strings $\{ab^h, b^hc, b^{h+1}\}$ (this is actually the same example as from the analysis of the Greedy algorithm [1]). The output of Greedy-R can be the string ab^hcb^{h+1} with total overlap h, whereas an optimal solution to SCS-R is $ab^{h+1}c$ of total overlap $2h$.

6. Acknowledgments

The authors thank anonymous referees for a number of helpful comments and remarks.

Gabriele Fici acknowledges the support of the PRIN 2010/2011 project "Automi e Linguaggi Formali: Aspetti Matematici e Applicativi" of the Italian Ministry of Education (MIUR). Tomasz Kociumaka is supported by Polish budget funds for science in 2013-2017 as a research project under the 'Diamond Grant' program. Jakub Radoszewski is a Newton International Fellow. Wojciech Rytter is supported by the Polish National Science Center, grant no 2014/13/B/ST6/00770.

References

[1] Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41(4):630–647, 1994.

[2] Raphael Clifford, Zvi Gotthilf, Moshe Lewenstein, and Alexandru Popa. Restricted common superstring and restricted common supersequence. In *Combinatorial Pattern Matching*, volume 6661 of *Lecture Notes in Computer Science*, pages 467–478. Springer Berlin Heidelberg, 2011.

[3] Maxime Crochemore, Marek Cygan, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Algorithms for three versions of the shortest common superstring problem. In *Combinatorial Pattern Matching*, volume 6129 of *Lecture Notes in Computer Science*, pages 299–309. Springer Berlin Heidelberg, 2010.

[4] John Gallant, David Maier, and James A. Storer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, 1980.

[5] John K. Gallant. *String compression algorithms*. PhD thesis, Princeton University, 1982.

[6] Theodoros P. Gevezes and Leonidas S. Pitsoulis. The shortest superstring problem. In *Optimization in Science and Engineering*, pages 189–227. Springer New York, 2014.

[7] Zvi Gotthilf, Moshe Lewenstein, and Alexandru Popa. On shortest common superstring and swap permutations. In *String Processing and Information Retrieval*, volume 6393 of *Lecture Notes in Computer Science*, pages 270–278. Springer Berlin Heidelberg, 2010.

[8] Tao Jiang and Ming Li. Approximating shortest superstrings with constraints. *Theoretical Computer Science*, 134(2):473 – 491, 1994.

[9] Tao Jiang, Ming Li, and Ding-Zhu Du. A note on shortest superstrings with flipping. *Information Processing Letters*, 44(4):195–199, 1992.

[10] Haim Kaplan and Nira Shafrir. The greedy algorithm for shortest superstrings. *Information Processing Letters*, 93(1):13–17, 2005.

[11] John D. Kececioglu and Eugene W. Myers. Combinatiorial algorithms for DNA sequence assembly. *Algorithmica*, 13(1/2):7–51, 1995.

[12] Bin Ma. Why greed works for shortest common superstring problem. *Theoretical Computer Science*, 410(51):5374–5381, 2009.

[13] David Maier and James A. Storer. A note on the complexity of the superstring problem. Technical Report 233, Princeton University, 1977.

[14] Marcin Mucha. Lyndon words and short superstrings. In *Proceedings of SODA 2013*, pages 958–972, 2013.

[15] Jorma Tarhio and Esko Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science*, 57:131–145, 1988.

[16] Jonathan S. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation*, 83(1):1–20, 1989.

[17] Esko Ukkonen. A linear-time algorithm for finding approximate shortest common superstrings. *Algorithmica*, 5(3):313–323, 1990.