Superstrings with multiplicities

Bastien Cazaux
Department of Computer Science, University of Helsinki, Helsinki, Finland;
L.I.R.M.M., Université Montpellier, Montpellier, France & Institute of Computational Biology, Montpellier, France
bastien.cazaux@cs.helsinki.fi

Eric Rivals
L.I.R.M.M., Université Montpellier, Montpellier, France & Institute of Computational Biology, Montpellier, France
rivals@lirmm.fr
https://orcid.org/0000-0003-3791-3973

Abstract

A superstring of a set of words $P = \{s_1, \dots, s_p\}$ is a string that contains each word of P as substring. Given P, the well known Shortest Linear Superstring problem (SLS), asks for a shortest superstring of P. In a variant of SLS, called Multi-SLS, each word s_i comes with an integer $m(i)$, its multiplicity, that sets a constraint on its number of occurrences, and the goal is to find a shortest superstring that contains at least $m(i)$ occurrences of s_i. Multi-SLS generalizes SLS and is obviously as hard to solve, but it has been studied only in special cases (with words of length 2 or with a fixed number of words). The approximability of Multi-SLS in the general case remains open. Here, we study the approximability of Multi-SLS and that of the companion problem Multi-SCCS, which asks for a shortest cyclic cover instead of shortest superstring. First, we investigate the approximation of a greedy algorithm for maximizing the compression offered by a superstring or by a cyclic cover: the approximation ratio is 1/2 for Multi-SLS and 1 for Multi-SCCS. Then, we exhibit a linear time approximation algorithm, Concat-Greedy, and show it achieves a ratio of 4 regarding the superstring length. This demonstrates that for both measures Multi-SLS belongs to the class of APX problems.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics, Theory of computation → Approximation algorithms analysis

Keywords and phrases greedy algorithm, approximation, overlap, cyclic cover, APX, subset system

Digital Object Identifier 10.4230/LIPIcs.CPM.2018.21

Funding This work is supported by the Institut de Biologie Computationnelle `http://www.ibc-montpellier.fr` (ANR-11-BINF-0002) and Défi MASTODONS C3G `http://www.lirmm.fr/~rivals/C3G/` from CNRS.

Acknowledgements We thank for the reviewers for their comments.

1 Introduction

Given a set of p words $P := \{s_1, s_2, \dots, s_p\}$ over a finite alphabet Σ, a superstring of P is a string containing each s_i for $1 \leq i \leq p$ as a substring. The **Shortest Linear Superstring** (`SLS`) problem is an optimization problem that asks for a superstring of P of minimal length. It is also known as the Shortet Common Superstring problem, which does not convey the fact that the output superstring is a linear rather than cyclic word. `SLS` has been studied in depth for its applications in data compression, where a superstring is an alternative representation of P, and in bioinformatics [11]. SLS is known to be hard to solve (**NP-hard** provided the input words are of length at least three) and to approximate (**MAX-SNP-hard**), and these difficulties remain even if one considers instances over a binary alphabet [10, 3, 17]. In bioinformatics, SLS models the initial step of genome assembly in a shotgun sequencing approach [1], whose input is a large and redundant set of *"reads"*. This first step consists in merging overlapping words to obtain partial substrings of the target genome. These output strings are called *contigs*. In practice, one never obtains a single superstring covering the genome, but a large set of contigs. A major difficulty that is inherent to biology comes from the presence of repeated regions in genomes. When assembled, the distinct copies of a repeat tend to collapse into a single occurrence, and the corresponding contig then exhibits a higher density of merged words [1]. By comparing the local density of a contig to the expected density, one can estimate the underlying number of copies for a repeat. The assembly process can then be rerun using these *multiplicities*, that is for each word, the number of times it must appear in the superstring. To take into account the issue of repeated regions in SLS, Crochemore et al. have proposed a variant of SLS called Multi-SLS[1]: the input consists in P with a function m giving the multiplicity of each word of P, and the output *multi superstring* must contain at least $m(s_i)$ occurrences of s_i, for any $1 \leq i \leq p$ [8]. They present two polynomial time algorithms to solve two special cases of Multi-SLS: First, the case where the number of input words is constant, and second the case where each input word has length 2. The latter generalizes SLS for words of length 2, which can also be solved in polynomial time [10].

Contributions To our knowledge, the approximability of Multi-SLS in the general case (*i.e.*, with an unbounded number of words of length ≥ 2) is wide open. As for SLS, two measures can be considered: the superstring length or its compression – the superstring length minus the sum of the lengths of all required occurrences of words of P. In general, for an optimization problem $\mathcal{P}$, we denote by $\mathcal{P}_{\text{comp}}$ the related problem that maximizes the compression measure.

▶ **Example 1.** Consider the instance (P, m) with $P := \{aab, abaa, baba\}$ with multiplicities $m(aab) = 2$, $m(abaa) = 1$, and $m(baba) = 2$. Then $w := aabaabababa$ is a multi superstring of (P, m), it has length 11 and achieves of compression of 9 symbols. Similarly, the string $y := aabaababaababababa$, which results from concatenating the required words, also is a multi superstring of (P, m) of length 18 and thus yields a compression of 0.

In Section 3, we study the greedy algorithm for $\texttt{Multi-SLS}_{\text{comp}}$ and show it has a compression ratio of 1/2 in Theorem 9. In Section 5, we propose for Multi-SLS the first polynomial time approximation algorithm, called **Concat-Greedy**, and prove in Theorem 15 that it admits an approximation ratio of 4 for the superstring length measure. Hence, we demonstrate:

▶ **Theorem 2.** *Both Multi-SLS and $\texttt{Multi-SLS}_{comp}$ belong to the class **APX**.*

In fact, the ratio of 4 follows from a stronger bound on the length of a solution of **Concat-Greedy** (see Proposition 14 p. 11). Note that the same ratio of 4 was proven for the classical SLS problem by [3] in 1994, using the **Concat-Cycles** algorithm. To achieve this bound, **Concat-Greedy** must solve a related problem called Multi-SCCS, where the

[1] According to the notation of [8], this variant was termed MULTI-SCS.

solution is a set of cyclic strings that collectively contain all the required occurrences of words of P. Such a set is called a *cyclic cover of strings*, or *cyclic cover* for short. First, we show in Section 3 that a greedy algorithm solves exactly `Multi-SCCS`, and then exhibit in Section 4 a graph based algorithm for it and bound its time complexity, which yields:

▶ **Theorem 3.** *The `Multi-Greedy` algorithm (Algo. 2) solves the `Multi-SCCS` problem in a time that is linear in the size of its output.*

2 Preliminaries

Here, we introduce basic notions on strings, permutations, superstrings and formally define the two problems `Multi-SLS` and `Multi-SCCS`. Then, we derive a logical, but important fact: all multi superstrings (resp. multi cyclic cover) we need to consider are induced by permutations. For any finite set U, $|U|$ denotes its cardinality.

About strings. Let u, v be two linear strings. We denote by $|u|$ the length of u, and by uv their concatenation. Given a linear string u, we obtain the *circular string* $\langle u \rangle$ by linking the last letter of the linear string u to its first letter. The length of the circular string $\langle u \rangle$ is the length of the linear string u. Given a set of linear or circular strings P, we call the *norm* of P, denoted by $||P||$, *i.e.* the sum of the lengths of the strings of P.

Let $x := x_1 \dots x_n$ and $y := y_1 \dots y_m$ be two linear strings (where for any $1 \leq j \leq m$, y_j is the j^{th} letter of y). We denote by $\texttt{Occ}(y, x)$ the set of the occurences of y in x, *i.e.*, the set of positions i between 1 and $n - m + 1$ such that $x_i \dots x_{i+m-1} = y_1 \dots y_m$. Whenever $\texttt{Occ}(y, x)$ is not empty, y is said to be a *substring* of x. We extend the notion of substring to circular strings by extending the set of occurences: we denote by $\texttt{Occ}(y, \langle x \rangle)$ the set $\texttt{Occ}(y, x^\infty) \cap \{1, \dots, |x|\}$ (where $x^\infty = xx\dots$). A *prefix* y (respectively a *suffix*) of a linear string x is a substring beginning (respectively ending) x, *i.e.*, $1 \in \texttt{Occ}(y, x)$ (resp. $|x| - |y| + 1 \in \texttt{Occ}(y, x)$). Furthermore, we say that y is a *proper* substring of x if $|y| < |x|$ (Definitions of a proper prefix/suffix are similar). Let M be a set of linear or circular strings; we denote by $\texttt{Occ}(x, M)$ the set of all occurences of x in all strings of M.

Problem definitions. Throughout the article, let $P := \{s_1, \dots, s_p\}$ be a set of linear strings P and a function m from P to $\mathbb{N}^*$ giving the multiplicity of each string. We assume that P is factor-free, *i.e.*, s_i is not a substring of s_j for any i, j in $\{1, \dots, p\}$. The pair (P, m) is the input of the problems `Multi-SLS` and `Multi-SCCS`. A *superstring* of P is a word w such that for any $1 \leq i \leq p$, $|\texttt{Occ}(s_i, w)| \geq 1$.

Let us define formally the two minimization problems `Multi-SLS` and `Multi-SCCS`: both seek to minimize their output length. Note that Definition 4 is equivalent to that `MULTI-SCS(k)` from [8].

▶ **Definition 4** (Multi Shortest Linear Superstring (`Multi-SLS`)). Let $P := \{s_1, \dots, s_p\}$ be a set of strings and m a function from P to $\mathbb{N}^*$. It seeks a linear string w of minimal length and such that for all $s_i \in P$, $|\texttt{Occ}(s_i, w)| \geq m(s_i)$.

▶ **Definition 5** (Multi Shortest Cyclic Cover of Strings (`Multi-SCCS`)). Let $P := \{s_1, \dots, s_p\}$ be a set of strings and m a function from P to $\mathbb{N}^*$. It seeks a set C of circular strings of minimal norm and such that for all $s_i \in P$, $|\texttt{Occ}(s_i, C)| \geq m(s_i)$.

In any solution of `Multi-SLS` or of `Multi-SCCS`, each string s of P must occur at least $m(s)$ times. Let us define $\widetilde{P}$ to be the set containing $m(s)$ copies of each word s of P; to

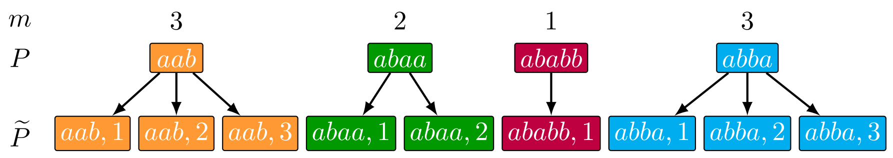


Figure 1 Example of $\widetilde{P}$ for the instance (P, m) of Example 6. Due to space constraints, for a pair of $\widetilde{P}$ we may write aab,2 or 2.

distinguish its copies we denote any element of $\widetilde{P}$ by a pair (s, i) for $1 \leq i \leq m(s)$ – see Example 6 and Figure 1. Formally, *i.e.*

$$\widetilde{P} = \bigcup_{s \in P} \left(\cup_{i=1}^{m(s)} \{(s, i)\} \right).$$

For an element (s, i) of $\widetilde{P}$, we denote by $\texttt{word}((s, i))$ the word s of P, *i.e.*, $\texttt{word}((s, i)) = s$.

Note that for some instances – when words of P do not overlap each other – an optimal solution for `Multi-SLS` is the concatenation of all strings in $\widetilde{P}$, and has length $||\widetilde{P}|| := \sum_{i=1}^{p} m(s_i)\,|s_i|$. This observation remains valid for `Multi-SCCS`. Any algorithm solving `Multi-SLS` or `Multi-SCCS` has its complexity bounded by the length of its output, *i.e.*, by $||\widetilde{P}||$, which we consider to be linear in the input size. In [8], the authors seek to find a compressed representation of the output; we dwell on this question on page 10.

▶ **Example 6** (see Figure 1). This same instance (P, m) is used as running example throughout the paper. Let $P = \{aab, abaa, ababb, abba\}$ be a set of strings and m be the function from P to $\mathbb{N}^*$ such that $m(aab) = 3$, $m(abaa) = 2$, $m(ababb) = 1$ and $m(abba) = 3$. We have that

$$\widetilde{P} = \{(aab, 1), (abb, 2), (abb, 3), (abaa, 1), (abaa, 2), (ababb, 1), (abba, 1), (abba, 2), (abba, 3)\}.$$

About permutations. Given a permutation σ of a set E, a *successor* y of an element x of E by σ, is an element of E such that $y = \sigma^k(x)$ where $\sigma^1(x) = \sigma(x)$ and $\sigma^k(x) = \sigma^{k-1}(\sigma(x))$. We denote by $\texttt{Part}(E, \sigma)$ the partition $\{E_1, \ldots, E_p\}$ of E where each element of E and its successors are in the same subset E_i. A permutation is said *circular* if all the elements of E are successors of any element of E, i.e. $\texttt{Part}(E, \sigma) = \{E\}$. Moreover, we denote by $\texttt{Decomp}(E, \sigma)$ the decomposition into circular permutations of the permutation σ, *i.e.*, the set of pairs (E_i, σ_i) where $E_i \in \texttt{Part}(E, \sigma)$ and where σ_i is the restriction of σ to the elements of E_i.

About linear and circular superstrings. Given two linear strings u and v, an *overlap* from u over v is a linear string that is a proper suffix of u and a proper prefix of v. We denote by $\texttt{ov}(u, v)$ the longest overlap from u to v (also termed maximal overlap). Overlaps are not symmetrical. The *prefix from u to v*, denoted by $\texttt{pr}(u, v)$ is the string satisfying $u = \texttt{pr}(u, v)\texttt{ov}(u, v)$. The *merge* from u to v is the linear string $\texttt{pr}(u, v)v$ if $u \neq v$, and the circular string $\langle \texttt{pr}(u, u) \rangle$ otherwise. Given a set of strings P, we denote by $\texttt{Ov}(P)$ the set of all the maximal overlaps between any two strings of P.

Let $P = \{s_1, \ldots, s_p\}$ be a set of linear strings. We denote by $\texttt{Linear}(s_1, \ldots, s_p)$ (resp. by $\texttt{Circular}(s_1, \ldots, s_p)$) the linear (resp. circular) string defined by the merge of $s_1, \ldots, s_p$ in this order:

$$\texttt{Linear}(s_1, \ldots, s_p) := \texttt{pr}(s_1, s_2)\texttt{pr}(s_2, s_3) \ldots \texttt{pr}(s_{p-1}, s_p)s_p$$

and

$$\texttt{Circular}(s_1, \ldots, s_p) := \langle \texttt{pr}(s_1, s_2)\texttt{pr}(s_2, s_3) \ldots \texttt{pr}(s_{p-1}, s_p)\texttt{pr}(s_p, s_1)\rangle.$$

▶ Remark. The starting point of the merge does not impact `Circular()`. Formally, for all $j \in \{1, \ldots, p\}$, $\texttt{Circular}(s_1, \ldots, s_p) = \texttt{Circular}(s_j, \ldots, s_p, s_1, \ldots, s_{j-1})$.

About multi superstrings and multi cyclic covers induced by a permutation. The number of possible superstrings or cyclic covers of $\widetilde{P}$ is infinite, which makes the search space for `Multi-SLS` / `Multi-SCCS` unpractical. Hence, a crucial issue is whether we can restrict this search space. For this sake, we introduce the notion of multi superstring/cyclic cover induced by a permutation.

Let τ be a permutation of $\widetilde{P}$. If τ is a circular permutation (meaning that all its elements are successors of each other), we can define the *multi superstring induced by* τ and by an element $\widetilde{s}$ of $\widetilde{P}$ as follows:

$$\texttt{Lin}(\widetilde{P}, \tau, \widetilde{s}) = \texttt{Linear}(next\_word(\widetilde{s}, 1), \ldots, next\_word(\widetilde{s}, |\widetilde{P}|))$$

where $next\_word(\widetilde{s}, k) = \texttt{word}(\tau^k(\widetilde{s}))$. Here, the term `Linear()` of this equation is the merge of the words of $\widetilde{P}$ in the order given by τ and ending with the chosen element $\widetilde{s}$ (indeed, $next\_word(\widetilde{s}, |\widetilde{P}|) = \texttt{word}(\widetilde{s})$).

In general, τ is not circular. It can be decomposed in several circular permutations (see Fig. 2a); we denote its decomposition by $\texttt{Decomp}(\widetilde{P}, \tau)$. We define, $\texttt{CC}(\widetilde{P}, \tau)$, the *multi cyclic cover of strings induced* by τ as follows:

$$\texttt{CC}(\widetilde{P}, \tau) = \bigcup_{(\widetilde{P}_i, \sigma_i) \in \texttt{Decomp}(\widetilde{P}, \tau)} \{\texttt{Circular}(next\_word(\widetilde{s}, 1), \ldots, next\_word(\widetilde{s}, |\widetilde{P}_i|))\}$$

where $\widetilde{s}$ is any element of $\widetilde{P}_i$ and $next\_word(\widetilde{s}, k) = \texttt{word}(\sigma_i^k(\widetilde{s}))$. $\texttt{CC}(\widetilde{P}, \tau)$ is a set of cyclic strings, each obtained by merging the words in the order given by a sub-permutation σ_i.

▶ **Example 7.** Let σ_1 and σ_2 be the permutations of $\widetilde{P}$ of Figure 2a and Figure 2b. Consider the pair $(abba, 3)$ in $\widetilde{P}$ (node $\boxed{3}$ in figures 2a and b); its direct successor with σ_1 is itself, *i.e.*, $\sigma_1((abba, 3)) = (abba, 3)$, and with σ_2, it is the node $\boxed{1}$ in Figure 2b, *i.e.*, $\sigma_2((abba, 3)) = (ababb, 1)$.

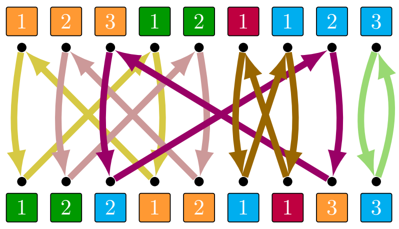
Some thoughts lead to the observation that any optimal multi superstring or multi cyclic cover is necessarily induced by a permutation on $\widetilde{P}$. This yields this proposition, which indeed restricts the search spaces of both problems. Due to space constraints, the proofs of some results (marked with a $\star$) are not included here; some proofs are given in the appendix.

▶ **Proposition 8** ($\star$). *Let (P, m) be an instance of `Multi-SLS` and of `Multi-SCCS`. Let w_{opt} be an optimal solution of `Multi-SLS` and let C_{opt} be an optimal solution of `Multi-SCCS`. Then, there exist*

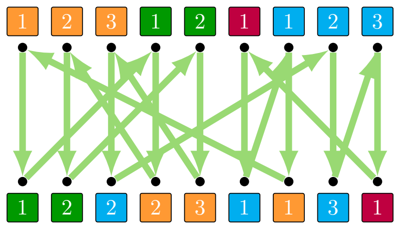
1. *a permutation τ of $\widetilde{P}$ such that $C_{opt} = \texttt{CC}(\widetilde{P}, \tau)$.*
2. *a circular permutation φ of $\widetilde{P}$ and an element $\widetilde{s}$ of $\widetilde{P}$ such that $w_{opt} = \texttt{Lin}(\widetilde{P}, \varphi, \widetilde{s})$.*

3 Approximation

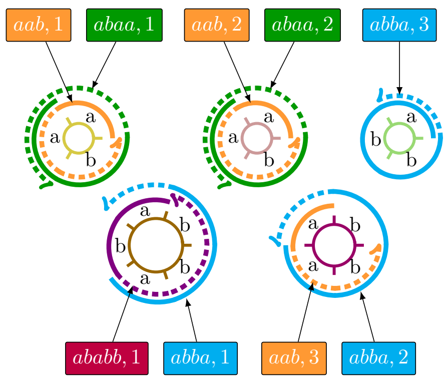
Here, let us define the greedy algorithms for `Multi-SLS` and `Multi-SCCS` problems and exhibit their approximation ratios for the measure of compression.



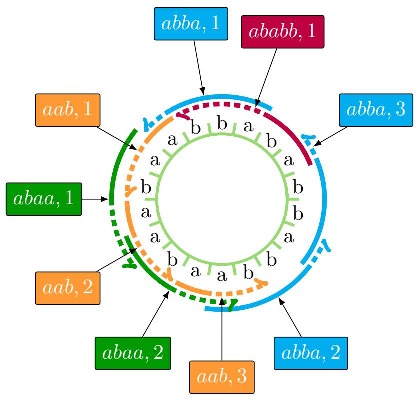
(a) Permutation σ_1 of $\widetilde{P}$



(b) Permutation σ_2 of $\widetilde{P}$



(c) Cyclic cover induced by σ_1



(d) Cyclic cover induced by σ_2

Figure 2 Running example: two possible permutations of $\widetilde{P}$ (Fig. a & b), and the cyclic covers induced by these permutations (Fig. c and d). Permutation σ_1 in (a) is decomposed in 5 circular permutations (five colors in a) and induces 5 cyclic strings (c), while permutation σ_2 cannot be decomposed and induces a single cyclic string (d). In (c, d) input words are drawn as arrows around the cyclic strings, and the dashed part represents the overlap with the successor.

Greedy algorithms. By Proposition 8, we have that each optimal solution of `Multi-SCCS` can be induced by a permutation on $\widetilde{P}$. We can generalize the greedy algorithm for `SCCS` [5] to `Multi-SCCS`.

The basic principle of the greedy algorithm for `SLS` or `SCCS` is 1/ to merge a pair of strings at each step until all merge possibilities have been exhausted, and 2/ to consider pairs of strings to be merged in order of decreasing overlap length, and 3/ to break ties randomly. It is greedy because it chooses merge operations that yield the best compression first, and never backtracks on these choices. In fact, the greedy algorithm determines a total ordering on the merge operations (it is the greedy algorithm of a precise subset system – see [6] for details). In stringology, the greedy algorithm is usually presented as in Algorithm 1: the initial set of words (set Q in Algorithm 1) is iteratively modified at each iteration of the main loop: a pair of strings is chosen, those strings removed from the set, and the string resulting from the merge is (re-)inserted in the set. The two formulations of the algorithm are equivalent [6], basically because the new string offers the same overlaps with remaining words as the strings that were merged.

Of course the algorithm differs between the linear and cyclic cases. For `SLS` or `Multi-SLS`, the loop merges pairs of words until getting a single linear string, which is the final result. For `SCCS` or `Multi-SCCS`, the result is a set of cyclic strings, which is iteratively built (solution set S). A merge of two linear string results in a linear string, but the merge of a single string that self-overlaps yields a cyclic string. A cyclic string has no overlap and cannot be merged.

Algorithm 1: The greedy algorithm for `Multi-SCCS`.

```
1  **Input**: a pair $(P, m)$; **Output**: $S$: a cyclic cover of strings covering $\widetilde{P}$;
2  $S := \emptyset$; // the solution set in construction
3  $Q := \widetilde{P}$;
4  newIndex := $|Q|$;
5  **while** $|\mathbf{Q}| > \mathbf{0}$ **do**
6      $(u, i)$ and $(v, j)$ two elements of $Q$ such that $u$ and $v$ have the longest overlap;
       // $u$ can be equal to $v$ and i equal to j
7      $w$ is the merge of $u$ and $v$;
8      $Q := Q \setminus \{(u, i), (v, j)\}$;
9      **if** $u = v$ *and* $i = j$ *(i.e., $w$ is a cyclic string)* **then** $S := S \cup \{w\}$;
10     **else** $Q := Q \cup \{(w, \text{newIndex}{+}{+})\}$;
11 **return** $S$
```

Hence, each cyclic string is directly inserted into the solution set (set S, line 9), while a linear string is re-inserted in the set of strings remaining to be merged (set Q, line 10). This explains why the loop condition is $|Q| > 0$ (line 5).

▶ Remark. Algorithm 1 is equivalent to iteratively merging the two elements u and v of $\widetilde{P}$ having the longest overlap, provided that u is not *merged on its right*[2] more than $m(u)$ times and v is not merged on its left more than $m(v)$ times. The word that results from the merge is inserted back into Q when it is linear, and inserted in the solution set S if it is cyclic. As elements of Q are pairs, we number each inserted word with a variable newIndex that is incremented on line 9.

We can also generalize the greedy algorithm for `Multi-SCCS` to `Multi-SLS`. To do so, we just need to change in Algorithm 1, the while condition "$|\mathbf{Q}| > \mathbf{0}$" by "$|\mathbf{Q}| > \mathbf{1}$" and, on line 6 "**and i equal to j**" by "**but i cannot be equal to j**".

Measure of compression. For the both problems `Multi-SLS` and `Multi-SCCS`, we want to minimize the length of the multi superstring or the norm of the multi cyclic cover of strings. If instead, we want to maximize the compression, that is the difference between the norm of $\widetilde{P}$ and the output size, we call the corresponding problems `Multi-SLS`$_{\text{comp}}$ and `Multi-SCCS`$_{\text{comp}}$.

As the size of the input is constant, all optimal solutions of `Multi-SCCS` are also optimal solutions of `Multi-SCCS`$_{\text{comp}}$, and vice versa. The set of optimal solutions of `Multi-SLS` is also equal to the set of optimal solutions of `Multi-SLS`$_{\text{comp}}$. By Proposition 8, as we can restrict to solutions induced by a permutation of $\widetilde{P}$, the compression can be seen as the sum of the lengths of the overlaps between two successive strings in the permutation. Indeed, for a permutation τ of $\widetilde{P}$,

$$||\widetilde{P}|| - |\mathtt{CC}(\widetilde{P}, \tau)| = \sum_{(\widetilde{P}_i, \sigma_i) \in \mathtt{Decomp}(\widetilde{P}, \tau)} \left(\sum_{j=1}^{|\widetilde{P}_i|} |\mathtt{ov}(next\_word(\widetilde{s}, j), next\_word(\widetilde{s}, j+1))| \right)$$

where $\widetilde{s} \in \widetilde{P}_i$ and $next\_word(\widetilde{s}, k) = \mathtt{word}(\sigma_i^k(\widetilde{s}))$. Similarly, we get that `Multi-SLS`$_{\text{comp}}$

[2] Merged on its right (resp. left) means using an overlap of its suffix (resp. prefix).

maximizes the sum of the lengths of the successive overlaps in a multi superstring induced by a permutation.

Approximation for compression. We can see the greedy algorithm for `SLS` (and $\texttt{SLS}_{\text{comp}}$) as the greedy algorithm for finding a maximum weighted Hamiltonian path (*Maximum Asymmetric Travelling Salesman Problem* – `Max-ATSP`) in the overlap graph [15]. The overlap graph is a complete digraph labelled on the arcs, where each input word is a node, and where the length of the maximal overlap between two words is a weight on the corresponding arc [3]. Theorem 9 generalizes the half compression of greedy algorithm for `SLS` from [15] to `Multi-SLS` (full proof in the Appendix).

▶ **Theorem 9.** *The greedy algorithm for* $\texttt{Multi-SLS}_{comp}$ *has a* $\frac{1}{2}$ *approximation ratio.*

Proof. (See details in Appendix.) In [6], we show that one can prove the approximation ratio of the greedy algorithm for $\texttt{SLS}_{\text{comp}}$ by combining the Monge inequality [14] with subset systems that simulate the greedy algorithm for `Max-ATSP` in graphs [13]. By building the overlap graph for $\widetilde{P}$ (see Figure 4a), we can use the same subset system on the maximal overlaps of $\widetilde{P}$ and obtain the same approximation ratio for the greedy algorithm of $\texttt{Multi-SLS}_{\text{comp}}$ as for that of $\texttt{SLS}_{\text{comp}}$. ◀

With the same arguments, we can show that the approximation ratio of the greedy algorithm for $\texttt{Multi-SCCS}_{\text{comp}}$ equals that of the greedy algorithm for $\texttt{SCCS}_{\text{comp}}$, which is 1 [6]. This yields Theorem 10.

▶ **Theorem 10.** *For both problems* $\texttt{Multi-SCCS}_{comp}$ *and* `Multi-SCCS`*, the greedy algorithm (Algorithm 1) yields an optimal solution.*

By Proposition 8 and by the fact that greedy solutions for `Multi-SCCS` are optimal, we can represent each greedy solution by a permutation of $\widetilde{P}$. For any instance (P, m), let $\texttt{GreedyPerm}(\widetilde{P})$ denote the set of permutations of $\widetilde{P}$ corresponding to greedy solutions for `Multi-SCCS`.

4 Linear construction of Multi-SCCS

In this section, we show how to compute a greedy solution for `Multi-SCCS` in linear time in the norm of the set of strings of the input and in the norm of an optimal solution of `Multi-SCCS`. To achieve this, we adapt the superstring graph [5] in order to model greedy solutions for `Multi-SCCS`. Now, assume that one stores $m(w)$, the multiplicity of a string w, in constant space ($O(1)$ bits); hence the input, (P, m), has size $O(||P||)$.

Red-Blue graphs. To begin with, we define the Red-Blue graphs, which are intermediate digraphs needed to define the multi superstring graph – see Figure 3 (or Figure 6 in appendix). Let τ be a permutation for $\widetilde{P}$ and $\widetilde{s}$ an element of $\widetilde{P}$. We define, $\texttt{RB-Graph}(\tau, \widetilde{s}) := (V, R, B)$, the Red-Blue graph of $\widetilde{s}$ for the permutation τ as

$$\begin{aligned}
V &= \{\texttt{word}(\widetilde{s}), \texttt{word}(\tau(\widetilde{s}))\} \cup \{y \in \texttt{Ov}(P) : |y| \geq |\texttt{ov}(\texttt{word}(\widetilde{s}), \texttt{word}(\tau(\widetilde{s})))|\,, \text{ and} \\
&\qquad (y \text{ suffix of } \texttt{word}(\widetilde{s}) \text{ or } y \text{ prefix of } \texttt{word}(\tau(\widetilde{s})))\}, \\
R &= \{(u, v) \in V \times V \mid v \text{ is the longest proper } \textbf{suffix} \text{ of } u \text{ in } V\}, \\
B &= \{(u, v) \in V \times V \mid u \text{ is the longest proper } \textbf{prefix} \text{ of } v \text{ in } V\}.
\end{aligned}$$

By the properties of prefixes/suffixes, Red-Blue graphs are path graphs, which we illustrate in Figure 3 (running example and permutation σ_1 from Fig. 2a). Note that a Red-Blue graph of $\widetilde{s}$ depends on $\texttt{Ov}(P)$: it may contain a suffix/prefix that is an overlap of another pair of

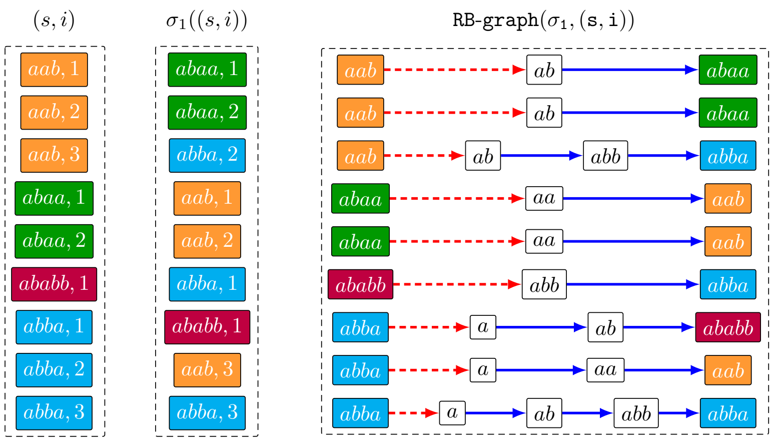


Figure 3 Running example: set of all the Red-Blue graphs of $(s, i) \in \widetilde{P}$ for the permutation σ_1 (see Figure 2a). A dashed arc (in red) links a string to its longest proper suffix, while a plain arc (in blue) links a longest proper prefix of a string to this string.

words $(\in \{(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s}))) \mid \widetilde{s} \in \widetilde{P}\})$. In Figure 3, it happens on the graph for the pair *aab* to *abba* since *abb* is not their maximal overlap.

Let u and v be in $P \cup \mathtt{Ov}(P)$. By the definition of Red-Blue graphs, the arc linking u to v occurs only once in a given Red-Blue graph, *i.e.*, $|\{(u, v)\} \cap (R \cup B)| \in \{0, 1\}$ (see Lemma 16 in Appendix). We define $\mathtt{NbOcc}(\tau, (u, v))$ as the number of occurrences of the arc (u, v) in all Red-Blue graphs for all $\widetilde{s}$ in $\widetilde{P}$. Thus, we get:

$$\mathtt{NbOcc}(\tau, (u, v)) := \sum_{\substack{\widetilde{s} \in \widetilde{P} \\ (V,R,B) = \mathtt{RB\text{-}Graph}(\tau, \widetilde{s})}} |\{(u, v)\} \cap (R \cup B)|.$$

Furthermore, we define $\mathtt{PrefixArc}(P)$ (resp. $\mathtt{SuffixArc}(P)$), as the set of arcs (u, v) (resp. (v, u)) of $(P \cup \mathtt{Ov}(P))^2$ such that u is the longest prefix (resp. suffix) of v in $P \cup \mathtt{Ov}(P)$.

For a permutation τ of $\widetilde{P}$ that corresponds to a greedy solution for $\mathtt{Multi\text{-}SCCS}$, we can count the $\mathtt{NbOcc}(\tau, (u, v))$ for all $(u, v) \in \mathtt{PrefixArc}(P) \cup \mathtt{SuffixArc}(P)$. For the sake of simplicity, we extend the function m to elements of $\mathtt{Ov}(P)$ and set: $m(w) = 0$ for any w in $\mathtt{Ov}(P)$.

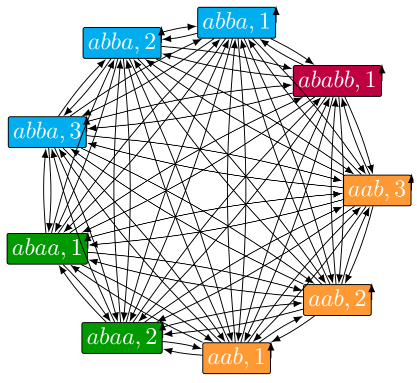
▶ **Proposition 11.** *Let be $\tau \in \mathtt{GreedyPerm}(\widetilde{P})$ and $(u, v) \in \mathtt{PrefixArc}(P) \cup \mathtt{SuffixArc}(P)$. We have that*

$$\mathtt{NbOcc}(\tau, (u, v)) = \begin{cases} \mathbf{Max}(m(v), -a(v)) & \text{if } |u| \leq |v| \\ \mathbf{Max}(m(u), a(u)) & \text{if } |u| > |v| \end{cases}$$

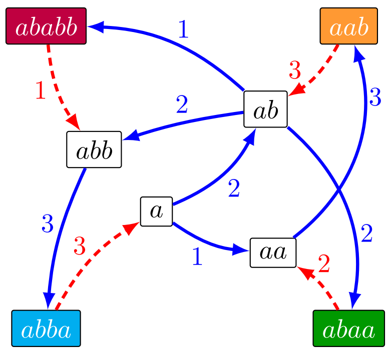
where $a(w) = \displaystyle\sum_{(w', w) \in \mathtt{SuffixArc}(P)} \mathtt{NbOcc}(\tau, (w', w)) - \sum_{(w, w') \in \mathtt{PrefixArc}(P)} \mathtt{NbOcc}(\tau, (w, w'))$.

Multi superstring graph. Let τ be a permutation of $\widetilde{P}$. We define $G_p(\tau) := (V, R, B, l)$ as the graph labelled on its arcs, which results from the merge of all Red-Blue graphs for all elements of $\widetilde{P}$ and for permutation τ. Formally:

$$\begin{array}{lcl} V & = & \mathtt{Ov}(P) \setminus U \\ R & = & \{(u, v) \in \mathtt{SuffixArc}(P) \mid \mathtt{NbOcc}(\tau, (u, v)) \neq 0\} \\ B & = & \{(u, v) \in \mathtt{PrefixArc}(P) \mid \mathtt{NbOcc}(\tau, (u, v)) \neq 0\} \\ l & : & (u, v) \mapsto \mathtt{NbOcc}(\tau, (u, v)) \end{array}$$



(a) Overlap graph of $\widetilde{P}$ (without weights)



(b) Multi superstring graph of (P, m)

Figure 4 Running example: overlap graph of $\widetilde{P}$ and multi superstring graph of (P, m).

where $U = \{v \in \texttt{Ov}(P) \mid v \text{ is not an extremity of an arc of } R \cup B\}$.

By Proposition 11, we have that for a permutation τ of $\texttt{GreedyPerm}(\widetilde{P})$ and $(u, v) \in \texttt{PrefixArc}(P) \cup \texttt{SuffixArc}(P)$, the number of occurrences of the arc (u, v), *i.e.* $\texttt{NbOcc}(\tau, (u, v))$, is independent of the permutation τ. From this observation and arguments from [6], we deduce Proposition 12.

▶ **Proposition 12** ($\star$). *Let τ_1, τ_2 be two permutations of* $\texttt{GreedyPerm}(\widetilde{P})$*. Then, $G_p(\tau_1) = G_p(\tau_2)$.*

By Proposition 12, all permutations inducing a greedy solution for an instance of `Multi-SCCS` yield the same graph, which we call the *multi superstring graph* and denote by $\texttt{SG}(P, m)$ (see Figure 4b). Using data structures like the (generalised) suffix tree to determine $\texttt{Ov}(P)$ [16], and with Proposition 11, we can build the multi superstring graph of (P, m) recursively and we obtain the following proposition.

▶ **Proposition 13** ($\star$). *The multi superstring graph can be built in linear time and space in $||P||$.*

Linear construction. By Proposition 11, we know that for $\texttt{SG}(P, m) = (V, R, B, l)$ the multi superstring graph of (P, m) the following equality holds:

$$\forall v \in V, \quad \sum_{(v,u) \in R} l((v, u)) - \sum_{(u,v) \in B} l((u, v)) = \sum_{(u,v) \in R} l((u, v)) - \sum_{(v,u) \in B} l((v, u)).$$

Hence, it follows that the multi superstring graph, in which the label of an arc is seen as a multi-arc, is Eulerian on each of its connected components. In Figure 4b, the arc from *abba* to *a* labelled by 3 means the Eulerian cycle must traverse this arc exactly thrice. Conversely, we can show that every set of cycles covering the multi superstring graph corresponds to a greedy solution for `Multi-SCCS`. As finding an Eulerian cycle cover of $\texttt{SG}(P, m)$ takes a time linear in $||P||$, we deduce Theorem 3 (p. 3).

Compressed output representation. At the beginning of this section, we have assumed that for each word of P, we can store the multiplicity in constant space. To improve the complexity, in this paragraph we assume that we can store the multiplicity of each string in $O(||P||)$ bits. In [8], the authors present a compact representation of a solution for `Multi-SLS`

Algorithm 2: The `Multi-Greedy` algorithm for `Multi-SCCS`

```
1 Input: a pair $(P, m)$. Output: $W$ a greedy solution for Multi-SCCS;
2 build SG$(P, m)$ the multi superstring graph of $(P, m)$;
3 compute an Eulerian multi-cycle $c = (c_1, \dots, c_n)$ of $G_P$;
4 for $j \in [1, n]$ do
5     traverse $c_j$: list the words of $P$ whose node is in $c_j$ and insert the cyclic string of
      the concatenation of the corresponding prefixes in $W$;
6 return $W$
```

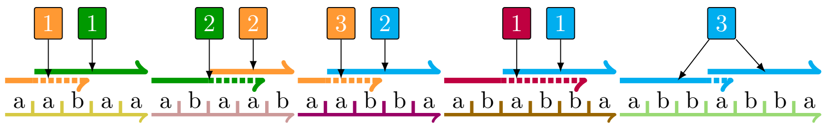


Figure 5 Running example: linearization $LinCC(\widetilde{P}, \sigma_1, W)$ of a cyclic cover of strings induced by permutation σ_1 (see Figure 2a) for $W :=$ (*abaa,1*, *aab,2*, *abba,2*, *abba,1*, *abba,3*).

with strings of length 2. They show that this compact representation has a size in $O(||P||^2)$ and can be computed in $O(||P||^2)$ time.

We can apply their technique to the multi superstring graph defined for `Multi-SCCS`. First, build the multi superstring graph of (P, m), and then using the algorithm *EulerianCycle* from [8] on SG(P, m), compute a compact representation of a multi cyclic cover of size $O(||P||^2)$ in $O(||P||^2)$ time. Now, as any connected component of SG(P, m) can be represented just by a permutation and its first element, one gets a compact representation of size $O(||P|| \times |P|)$, therefore improving on [8].

5 Approximation algorithm for Multi-SLS

Now, we propose an approximation algorithm for `Multi-SLS` and derive its approximation ratio with respect to the multi superstring length. By Theorem 9, we know that the greedy algorithm for `Multi-SLS`$_{\text{comp}}$ has an approximation ratio of $1/2$, and thus it belongs to **APX**. Here, we extend the `Concat-Cycles` algorithm from [3] and we show that this new algorithm, called `Concat-Greedy`, has an approximation ratio of 4 for `Multi-SLS`. The idea is to build an Eulerian multi-cycle of the multi superstring graph of (P, m), to break each cycle and merge its words to create linear strings, and to concatenate all these linear strings in an arbitrary order. Figure 5 displays an example of linearization.

To define formally the linearization of a cyclic cover of strings induced by permutation τ of $\widetilde{P}$, we denote $LinCC(\widetilde{P}, \tau, (w_1, \dots, w_p))$ the following linearization

$$LinCC(\widetilde{P}, \tau, (w_1, \dots, w_p)) = \texttt{Lin}(\widetilde{P_1}, \sigma_1, w_1) \dots \texttt{Lin}(\widetilde{P_p}, \sigma_p, w_p)$$

where $\texttt{Decomp}(\widetilde{P}, \tau) = \{(\widetilde{P_1}, \sigma_1), \dots, (\widetilde{P_p}, \sigma_p)\}$ and $(w_1, \dots, w_p) \in \widetilde{P_1} \times \dots \times \widetilde{P_p}$.

Now, let us define the algorithm `Concat-Greedy` by Algorithm 3.

Adapting the proof by Blum *et al.* of the approximation ratio of `Concat-Cycles` from [3], one gets the following bound on the length of a multi superstring computed by `Concat-Greedy`.

Algorithm 3: The algorithm `Concat-Greedy` for `Multi-SLS`

```
1 Input: a pair $(P, m)$. Output: a linear solution for Multi-SLS;
2 build SG$(P, m)$ the multi superstring graph of $(P, m)$;
3 compute an Eulerian multi-cycle of $G_P$ and take $\tau$ the permutation in
    GreedyPerm$(\widetilde{P})$ corresponding to this multi-cycle;
4 take a tuple $W$ of $E_1 \times \ldots \times E_p$ where Part$(\widetilde{P}, \tau) = \{E_1, \ldots, E_p\}$;
5 return $LinCC(\widetilde{P}, \tau, W)$
```

▶ **Proposition 14** (⋆). *Let w_{CG} be a solution of Algorithm 3, $w_{OPT(\texttt{Multi-SLS})}$ be an optimal solution of `Multi-SLS`, and $w_{OPT(\texttt{SLS})}$ be an optimal solution of `SLS`. We have:*

$$|w_{CG}| \quad \leq \quad |w_{OPT(\texttt{Multi-SLS})}| \quad + \quad 3 \; \times \; |w_{OPT(\texttt{SLS})}|.$$

As an optimal solution of `Multi-SLS` is longer than or equal to an optimal solution of `SLS`, one gets the following approximation ratio for **`Concat-Greedy`**, which is not tight.

▶ **Theorem 15.** *The approximation ratio of Algorithm **`Concat-Greedy`** for `Multi-SLS` is 4.*

▶ Remark. As we have made for `Multi-SCCS`, we can compute a compact representation of `Multi-SLS` of size $O(||P|| \times |P|)$ in time $O(||P||^2)$. Indeed, we linearize the compact representation of `Multi-SCCS` using **`Concat-Greedy`** to get a compact representation for `Multi-SLS`.

6 Conclusion

Here, we provide the first study of `Multi-SLS` in the general case, that is without restriction on the number of words, nor on the word length. `Multi-SLS` can be approximated for both the superstring length measure and for the compression measure. Finally, both `Multi-SLS` and $\texttt{Multi-SLS}_{\text{comp}}$ admit a constant approximation ratio, and thus belong to the class of **APX** problems. Proposition 14 shows that the difference in length between a multi-superstring returned by **`Concat-Greedy`** and an optimal multi-superstring is bounded by a term proportional to the length of an optimal superstring for `SLS`, on which the multiplicities have no impact. In practice, **`Concat-Greedy`** may produce solutions way below this bound. A future line of research is to implement this algorithm and evaluate its ratio experimentally, an approach of great interest for superstring problems. Indeed, for the classical `SLS` problem, a simple greedy like algorithm seems to yield superstrings very close to the optimum, achieving a ratio that is orders of magnitude smaller than the theoretical bound [4]. Indeed, experimental tests allow to compare approximation algorithms and may help pinpointing hard instances. Of course, the theoretical ratio of the greedy algorithm, and the best possible approximation ratio remain open questions for `Multi-SLS`.

Our main result regarding `Multi-SCCS` is its solvability in linear time. The **`Multi-Greedy`** algorithm paves the way to the design of new approximation algorithms for `Multi-SLS`, as was done for the classical `SLS` problem. Let us stress that even if our algorithm builds the multi superstring graph for (P, m), the multiplicities do not impact the numbers of nodes or of arcs, but only the weights on the arcs. As shown in Figure 4b, it is crucial that these numbers are independent of the multiplicities. Another issue is to understand what influences the number of cycles in a solution of `Multi-SCCS`; minimizing it may improve the output of **`Concat-Greedy`**, which "looses" some symbols each time it breaks a cycle.

Regarding future work, numerous variants of SLS (with reversals, with DNA strings [12, 9]) or restrictions of SLS (*e.g.* to strings of the same length [7]) can also be investigated with multiplicities. The question of updating a shortest superstring when the instance changes is challenging [2]. Here, a change of multiplicity can be considered as an alteration of the instance.

References

1 Eric L. Anson and Eugene W. Myers. Algorithms for whole genome shotgun sequencing. In Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology, RECOMB 1999, Lyon, France, April 11-14, 1999*, pages 1–9. ACM, 1999. doi:10.1145/299432.299442.

2 Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královic, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica*, 61(2):227–251, 2011. doi:10.1007/s00453-010-9419-8.

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

4 Bastien Cazaux, Samuel Juhel, and Eric Rivals. Practical lower and upper bounds for the shortest linear superstring. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms (SEA) 2018, June 27–29, 2018, L'Aquila, Italy*, volume 103 of *LIPIcs*, page in press, 2018. doi:10.4230/LIPIcs.SEA.2018.18.

5 Bastien Cazaux and Eric Rivals. A linear time algorithm for Shortest Cyclic Cover of Strings. *J. Discrete Algorithms*, 37:56–67, 2016.

6 Bastien Cazaux and Eric Rivals. The power of greedy algorithms for approximating Max-ATSP, Cyclic Cover, and superstrings. *Discrete Applied Mathematics*, 212:48–60, 2016.

7 Bastien Cazaux and Eric Rivals. Relationship between superstring and compression measures: New insights on the greedy conjecture. *Discrete Applied Mathematics*, 2017. doi:10.1016/j.dam.2017.04.017.

8 Maxime Crochemore, Marek Cygan, Costas S. Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. Algorithms for three versions of the shortest common superstring problem. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, volume 6129 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 2010. doi:10.1007/978-3-642-13509-5_27.

9 Gabriele Fici, Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Walen. On the greedy algorithm for the Shortest Common Superstring problem with reversals. *Inf. Proc. Letters*, 116(3):245–251, 2016.

10 John Gallant, David Maier, and James A. Storer. On finding minimal length superstrings. *Journal of Computer and System Sciences*, 20:50–58, 1980.

11 Theodoros P. Gevezes and Leonidas S. Pitsoulis. *Optimization in Science and Engineering: In Honor of the 60th Birthday of Panos M. Pardalos*, chapter The Shortest Superstring Problem, pages 189–227. Springer New York, New York, NY, 2014. doi:10.1007/978-1-4939-0808-0_10.

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

13 Julián Mestre. Greedy in Approximation Algorithms. In *Proceedings of 14th Annual European Symposium on Algorithms (ESA)*, volume 4168 of *Lecture Notes in Computer Science*, pages 528–539. Springer, 2006.

14 Gaspard Monge. Mémoire sur la théorie des déblais et des remblais. In *Mémoires de l'Académie Royale des Sciences*, pages 666–704, 1781.

15 Jorma Tarhio and Esko Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, 57:131–145, 1988. doi:10.1016/0304-3975(88)90167-3.

16 Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.

17 Virginia Vassilevska. Explicit inapproximability bounds for the shortest superstring problem. In *30th Int. Symp. on Mathematical Foundations of Computer Science (MFCS)*, volume 3618 of *Lecture Notes in Computer Science*, pages 793–800. Springer, 2005.

A Details on the proofs for Theorems 9 and 10

This section summarizes the main lines of the proofs for Theorems 9 and 10 – formal proofs are left for a full version of this article. The proof of Theorem 9 (resp. Theorem 10) follows that of Theorem 3 (resp. Theorem 4) in [6]. We refer the reader to [13] for details on subset systems and the notion of extendibility.

Both proofs rely on a subset system to analyze the greedy algorithm for solving `Max-ATSP` in general graphs, and on the proof of its approximation ratio on Overlap Graphs. The goal of `Max-ATSP` is to find a maximum weighted Hamiltonian path in a digraph $G = (V, A)$. The subset system enforces three conditions on the arcs incorporated in a greedy solution:

1. any two arcs must start from distinct nodes
2. any two arcs must end in distinct nodes (*i.e.*, the symmetrical of the first condition)
3. there exist no cycle of length smaller than the cardinality of V.

These conditions ensure that the greedy algorithm indeed builds a Hamiltonian path. Thanks to its 3-extendibility and to Theorem 1 from [13], one deduce that the greedy algorithm yields a $1/3$ approximation ratio for `Max-ATSP`, and similarly a $1/2$ ratio for the `Maximum Weighted Cycle Cover` problem. However, these are the ratios for general graphs. In the case of overlap graphs, which satisfy the Monge condition [14], the proof of Theorem 3 in [6] shows by analyzing finely the greedy approximation, that the greedy algorithm yields a $1/2$ approximation ratio for `Max-ATSP`. Since, it is known that an approximation ratio for `Max-ATSP` translates directly to an approximation ratio for `Maximum Compression` [11], which is the version of `Shortest Linear Superstring` that seeks to maximize compression measure, one gets a $1/2$ approximation ratio for `SLS`. By applying this result on the overlap graph of $\widetilde{P}$, one derives the $1/2$ ratio for $\texttt{Multi-SLS}_{\text{comp}}$. A similar proof ends up with an approximation ratio of 1 for the `Maximum Weighted Cycle Cover` problem on overlap graph. This yields the same ratio for $\texttt{Multi-SCCS}_{\text{comp}}$, thereby showing that the greedy algorithm solves this problem exactly.

B Proof of Lemma 16 and Proposition 11

▶ **Lemma 16.** *Let τ be a permutation of $\widetilde{P}$ and $\widetilde{s} \in \widetilde{P}$. Consider* $\texttt{RB-Graph}(\tau, \widetilde{s}) := (V, R, B)$ *be the Red-Blue graph of $\widetilde{s}$, and let u and v be two strings of V. Then, the arc (u, v) occurs only once in the Red-Blue graph, in other words*

$$|\{(u, v)\} \cap (R \cup B)| = 1.$$

Proof of Lemma 16. We face two alternatives: any arc belongs either to B or to R. By the definition of R, if (u, v) belongs to R, then v is the longest proper suffix of u in V. Thus, the length of u is strictly larger than that of v. By the definition of B, if $(u, v) \in R$, then u

is the longest proper prefix of v in V. Thus, $|u| < |v|$. Hence, any arc of $R \cup B$ is either in R or in B, *i.e.*, $R \cap B = \emptyset$. By the unicity of the longest proper prefix/suffix, (u,v) cannot appear more than once in R nor in B, which concludes the proof. ◀

Proof of Proposition 11. By definition,

$$\mathtt{NbOcc}(\tau,(u,v)) := \sum_{\substack{\widetilde{s} \in \widetilde{P} \\ (V,R,B)=\mathtt{RB\text{-}Graph}(\tau,\widetilde{s})}} |\{(u,v)\} \cap (R \cup B)|.$$

By Lemma 16, $\mathtt{NbOcc}(\tau,(u,v))$ is the number of times the arc (u,v) occurs in all Red-Blue graphs of all the elements of $\widetilde{P}$.

To simplify the proof, we consider four alternative cases.

The case where $\boldsymbol{u}$ is an element of $\boldsymbol{P}$. As P is factor-free, (u,v) is an arc of a Red-Blue graph (V,R,B), and (u,v) is an element of R (since $|u| > |v|$). Moreover, $a(u) = 0$ because the set $\{(w',w) \in \mathtt{SuffixArc}(P)\} \cup \{(w,w') \in \mathtt{PrefixArc}(P)\}$ is empty. And thus,

$$\begin{aligned} \mathtt{NbOcc}(\tau,(u,v)) &= |\{u \mid \exists k \in \mathbb{N}, (u,k) \in \widetilde{P}\}| \\ &= m(u) \\ &= \mathbf{Max}(m(u), a(u)). \end{aligned}$$

The case where $\boldsymbol{v}$ is an element of $\boldsymbol{P}$. As P is factor-free, we get that (u,v) is an element of B since $|u| < |v|$, and that $a(v) = 0$. Hence,

$$\begin{aligned} \mathtt{NbOcc}(\tau,(u,v)) &= |\{v \mid \exists k \in \mathbb{N}, (v,k) \in \widetilde{P}\}| \\ &= m(v) \\ &= \mathbf{Max}(m(v), -a(v)). \end{aligned}$$

The case where $\boldsymbol{u \notin P}$, $\boldsymbol{v \notin P}$ and $\boldsymbol{|u| < |v|}$. As $|u| < |v|$, the arc (u,v) is an element of B. As $u \notin P$ and $v \notin P$, $m(u) = m(v) = 0$.

$$\begin{aligned} \mathtt{NbOcc}(\tau,(u,v)) &= |\{\widetilde{s} \in \widetilde{P} \mid (u,v) \text{ is an arc of } \mathtt{RB\text{-}Graph}(\tau,\widetilde{s})\}| \\ &= |\{\widetilde{s} \in \widetilde{P} \mid \mathtt{ov}(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s}))) \text{ is a prefix of } u\}| \\ &= |\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| \leq |u|\}|. \end{aligned}$$

As τ is a permutation of $\mathtt{GreedyPerm}(\widetilde{\mathrm{P}})$, and assuming that the set

$$\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| \leq |u|\}$$

is not empty (otherwise, we would have $\mathtt{NbOcc}(\tau,(u,v)) = 0$), we deduce that

$$\begin{aligned} \mathtt{NbOcc}(\tau,(u,v)) = {} & |\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s}))\}| \\ & -|\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| = |v|\}| \\ & -|\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| > |v|\}| \\ = {} & \sum\nolimits_{(v,w) \in \mathtt{PrefixArc}(P)} \Big(|\{\widetilde{s} \in \widetilde{P} \mid u \text{ and } w \text{ are prefixes of } \mathtt{word}(\tau(\widetilde{s}))\}| \\ & -|\{\widetilde{s} \in \widetilde{P} \mid u \text{ and } w \text{ are prefixes of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| \geq |w|\}|\Big) \\ & -|\{\widetilde{s} \in \widetilde{P} \mid u \text{ is a proper prefix of } \mathtt{word}(\tau(\widetilde{s})), |ov(\mathtt{word}(\widetilde{s}), \mathtt{word}(\tau(\widetilde{s})))| = |v|\}| \\ = {} & \sum\nolimits_{(v,w) \in \mathtt{PrefixArc}(P)} \mathtt{NbOcc}(\tau,(v,w)) - \sum\nolimits_{(w',v) \in \mathtt{SuffixArc}(P)} \mathtt{NbOcc}(\tau,(w'v)). \end{aligned}$$

Hence,

$$\mathtt{NbOcc}(\tau,(u,v)) = \mathbf{Max}(m(v), -a(v)).$$

The case where $\boldsymbol{u \notin P}$, $\boldsymbol{v \notin P}$ and $\boldsymbol{|u| > |v|}$ is similar to the previous case, where $u \notin P$, $v \notin P$ and $|u| < |v|$.

All cases have been considered and this concludes the proof. ◀

C Example: set of all Red-Blue graphs

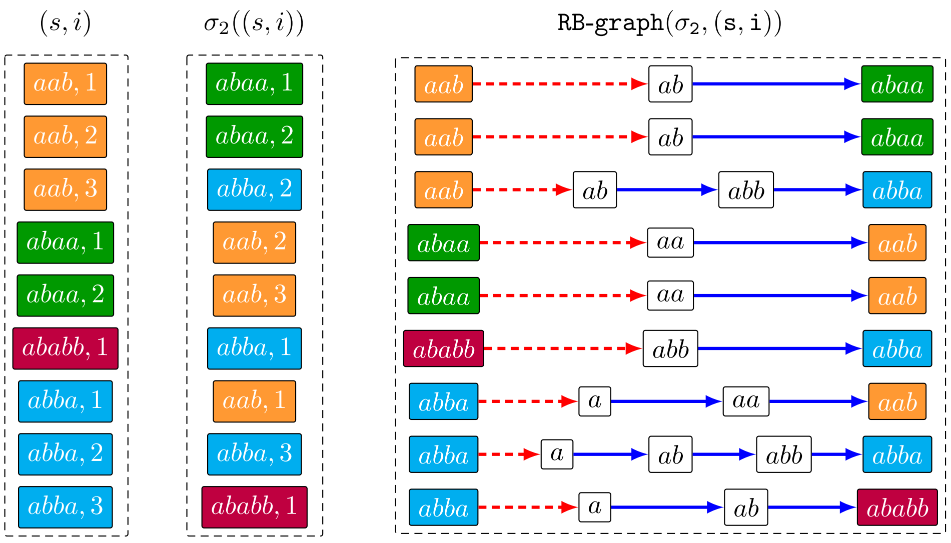


Figure 6 Running example: set of all the Red-Blue graphs of $(s, i) \in \widetilde{P}$ for the permutation σ_2 (see Figure 2b).