Superstrings with multiplicities
Eric Rivals, Bastien Cazaux

To cite this version:

HAL Id: lirmm-01843225
https://hal-lirmm.ccsd.cnrs.fr/lirmm-01843225
Submitted on 18 Jul 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Abstract

A superstring of a set of words $P = \{s_1, \ldots, s_p\}$ 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.

Introduction

Given a set of $p$ words $P := \{s_1, s_2, \ldots, s_p\}$ over a finite alphabet $\Sigma$, 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 Shortest 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 $\geq 2$) 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 $P$, we denote by $P_{\text{comp}}$ 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 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 Multi-SLS\(_{\text{comp}}\) 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. 12). 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 $(u)$ by linking the last letter of the linear string $u$ to its first letter. The length of the circular string $(u)$ 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 $x := x_1 \ldots x_n$ and $y := y_1 \ldots y_m$ be two linear strings (where for any $1 \leq j \leq m$, $y_j$ is the $j^{\text{th}}$ letter of $y$). We denote by $\text{Occ}(y, x)$ the set of the occurrences of $y$ in $x$, i.e., the set of positions $i$ between $1$ and $n - m + 1$ such that $x_{i} \ldots x_{i+m-1} = y_{1} \ldots y_{m}$. Whenever $\text{Occ}(y, x)$ 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 occurrences: we denote by $\text{Occ}(y, (x))$ the set $\text{Occ}(y, x^\infty) \cap \{1, \ldots, |x|\}$ (where $x^\infty = xx \ldots$). A prefix $y$ (respectively a suffix) of a linear string $x$ is a substring beginning (respectively ending) $x$, i.e., $1 \in \text{Occ}(y, x)$ (resp. $|x| - |y| + 1 \in \text{Occ}(y, x)$). 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 $\text{Occ}(x, M)$ the set of all occurrences of $x$ in all strings of $M$.

### Problem definitions

Throughout the article, let $P := \{s_1, \ldots, s_p\}$ 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 $i, j \in \{1, \ldots, p\}$. 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 $1 \leq i \leq p$, $|\text{Occ}(s_i, w)| \geq 1$.

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 $P := \{s_1, \ldots, s_p\}$ 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 $s_i \in P$, $|\text{Occ}(s_i, w)| \geq m(s_i)$.

**Definition 5** (Multi Shortest Cyclic Cover of Strings (Multi-SCCS)). Let $P := \{s_1, \ldots, s_p\}$ 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 $s_i \in P$, $|\text{Occ}(s_i, C)| \geq m(s_i)$.

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 $\hat{P}$ to be the set containing $m(s)$ copies of each word $s$ of $P$; to
distinguish its copies we denote any element of $\tilde{P}$ by a pair $(s, i)$ for $1 \leq i \leq m(s)$ – see Example 6 and Figure 1. Formally, i.e.

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

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

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 $\tilde{P}$, and has length $||P|| := \sum_{i=1}^{p} m(s_i) |s_i|$. 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 $||\tilde{P}||$, 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 11.

**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

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

**About permutations** Given a permutation $\sigma$ of a set $E$, a successor $y$ of an element $x$ of $E$ by $\sigma$, is an element of $E$ such that $y = \sigma^k(x)$ where $\sigma^1(x) = \sigma(x)$ and $\sigma^{k+1}(x) = \sigma^k(\sigma(x))$. We denote by $\text{Part}(E, \sigma)$ 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. $\text{Part}(E, \sigma) = \{E\}$. Moreover, we denote by $\text{Decomp}(E, \sigma)$ the decomposition into circular permutations of the permutation $\sigma$, i.e., the set of pairs $(E_i, \sigma_i)$ where $E_i \in \text{Part}(E, \sigma)$ and where $\sigma_i$ is the restriction of $\sigma$ 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 $\text{ov}(u, v)$ the longest overlap from $u$ to $v$ (also termed maximal overlap). Overlaps are not symmetrical. The prefix from $u$ to $v$, denoted by $\text{pr}(u, v)$ is the string satisfying $u = \text{pr}(u, v)ov(u, v)$. The merge from $u$ to $v$ is the linear string $\text{pr}(u, v)v$ if $u \neq v$, and the circular string $\text{pr}(u, v)$ otherwise. Given a set of strings $P$, we denote by $\text{ov}(P)$ 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 $\text{Linear}(s_1, \ldots, s_p)$ (resp. by $\text{Circular}(s_1, \ldots, s_p)$) the linear (resp. circular) string defined by the merge of $s_1, \ldots, s_p$ in this order:

$$\text{Linear}(s_1, \ldots, s_p) := \text{pr}(s_1, s_2)\text{pr}(s_2, s_3)\ldots\text{pr}(s_{p-1}, s_p)s_p$$
and
\[ \text{Circular}(s_1, \ldots, s_p) := \langle \text{pr}(s_1, s_2)\text{pr}(s_2, s_3) \ldots \text{pr}(s_{p-1}, s_p)\text{pr}(s_p, s_1) \rangle. \]

\begin{itemize}

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

\end{itemize}

### About multi superstrings and multi cyclic covers induced by a permutation

The number of possible superstrings or cyclic covers of \( \tilde{P} \) is infinite, which makes the search space for \text{Multi-SLS} / \text{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 \( \tau \) be a permutation of \( \tilde{P} \). If \( \tau \) is a circular permutation (meaning that all its elements are successors of each other), we can define the \textit{multi superstring induced by} \( \tau \) and by an element \( \tilde{s} \) of \( \tilde{P} \) as follows:

\[ \text{Lin}(\tilde{P}, \tau, \tilde{s}) = \text{Linear}(\text{next}_\omega(\tilde{s}, 1), \ldots, \text{next}_\omega(\tilde{s}, |\tilde{P}|)) \]

where \( \text{next}_\omega(\tilde{s}, k) = \text{word}(\tau^k(\tilde{s})) \). Here, the term \text{Linear}(\cdot) of this equation is the merge of the words of \( \tilde{P} \) in the order given by \( \tau \) and ending with the chosen element \( \tilde{s} \) (indeed, \( \text{next}_\omega(\tilde{s}, |\tilde{P}|) = \text{word}(\tilde{s}) \)).

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

\[ \text{CC}(\tilde{P}, \tau) = \bigcup_{(\tilde{P}, \sigma) \in \text{Decomp}(\tilde{P}, \tau)} \{ \text{Circular}(\text{next}_\omega(\tilde{s}, 1), \ldots, \text{next}_\omega(\tilde{s}, |\tilde{P}|)) \} \]

where \( \tilde{s} \) is any element of \( \tilde{P} \) and \( \text{next}_\omega(\tilde{s}, k) = \text{word}(\sigma^k(\tilde{s})) \). \( \text{CC}(\tilde{P}, \tau) \) is a set of cyclic strings, each obtained by merging the words in the order given by a sub-permutation \( \sigma \).

\begin{itemize}

\item \textbf{Example 7.} Let \( \sigma_1 \) and \( \sigma_2 \) be the permutations of \( \tilde{P} \) of Figure 2a and Figure 2b. Consider the pair \((\text{abba}, 3)\) in \( \tilde{P} \) (node 13 in figures 2a and b); its direct successor with \( \sigma_1 \) is itself, i.e., \( \sigma_1((\text{abba}, 3)) = (\text{abba}, 3) \), and with \( \sigma_2 \), it is the node 14 in Figure 2b, i.e., \( \sigma_2((\text{abba}, 3)) = (\text{ababb}, 1) \).

Some thoughts lead to the observation that any optimal multi superstring or multi cyclic cover is necessarily induced by a permutation on \( \tilde{P} \). 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 *) are not included here; some proofs are given in the appendix.

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

1. a permutation \( \tau \) of \( \tilde{P} \) such that \( C_{\text{opt}} = \text{CC}(\tilde{P}, \tau) \).
2. a circular permutation \( \varphi \) of \( \tilde{P} \) and an element \( \tilde{s} \) of \( \tilde{P} \) such that \( w_{\text{opt}} = \text{Lin}(\tilde{P}, \varphi, \tilde{s}) \).

\end{itemize}

### Approximation

Here, let us define the greedy algorithms for \text{Multi-SLS} and \text{Multi-SCCS} problems and exhibit their approximation ratios for the measure of compression.
Greedy algorithms

By Proposition 8, we have that each optimal solution of Multi-SCCS can be induced by a permutation on $\tilde{P}$. 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.
The greedy algorithm for Multi-SCCS.

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

   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 \(\tilde{P}\) 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 ’\(|Q| > 0\)’ by ’\(|Q| > 1\)’ 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 \(\tilde{P}\) 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 \(\tilde{P}\), 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 \(\tau\) of \(\tilde{P}\),

\[
||\tilde{P}|| - |\mathcal{CC}(\tilde{P}, \tau)| = \sum_{(\tilde{P}, \sigma) \in \text{Decomp}(\tilde{P}, \tau)} \left( \sum_{j=1}^{|	ilde{P}_i|} |\text{ov}(\text{next_word}(\tilde{s}, j), \text{next_word}(\tilde{s}, j + 1))| \right)
\]

where \(\tilde{s} \in \tilde{P}_i\) and \(\text{next_word}(\tilde{s}, k) = \text{word}(\sigma^k(\tilde{s}))\). Similarly, we get that Multi-SLS\(_{\text{comp}}\)
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 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).

\textbf{Theorem 9.} The greedy algorithm for Multi-SLS\text{comp} has a $\frac{1}{2}$ approximation ratio.

\textbf{Proof.} (See details in Appendix.) In [6], we show that one can prove the approximation ratio of the greedy algorithm for 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 $\tilde{P}$ (see Figure 4a), we can use the same subset system on the maximal overlaps of $\tilde{P}$ and obtain the same approximation ratio for the greedy algorithm of Multi-SLS\text{comp} as for that of SLS\text{comp}.

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

\textbf{Theorem 10.} For both problems Multi-SCCS\text{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 $\tilde{P}$ for the permutation $\sigma$. Therefore, the input, $(P,m)$, let GreedyPerm($\tilde{P}$) denote the set of permutations of $\tilde{P}$ 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 $O(||P||)$.

\textbf{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 $\tau$ be a permutation for $\tilde{P}$ and $\tilde{s}$ an element of $\tilde{P}$. We define, $\text{RB-Graph}(\tau, \tilde{s}) := (V,R,B)$, the Red-Blue graph of $\tilde{s}$ for the permutation $\tau$ as

\begin{align*}
V &= \{\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s}))\} \cup \{y \in \text{ov}(P) : |y| \geq \text{ov(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s})))}\}, \text{ and }
(y \text{ suffix of word}(\tilde{s}) \text{ or } y \text{ prefix of word}(\tau(\tilde{s})))
, \\
R &= \{(u,v) \in V \times V \mid v \text{ is the longest proper suffix of } u \text{ in } V\}, \\
B &= \{(u,v) \in V \times V \mid u \text{ is the longest proper prefix of } v \text{ in } V\}.
\end{align*}

By the properties of prefixes/suffixes, Red-Blue graphs are path graphs, which we illustrate in Figure 3 (running example and permutation $\sigma_1$ from Fig. 2a). Note that a Red-Blue graph
of $\tilde{s}$ depends on $\Omega(P)$: it may contain a suffix/prefix that is an overlap of another pair of words ($\in \{(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s}))) \mid \tilde{s} \in \tilde{P}\}$). In Figure 3, it happens on the graph for the pair $aab$ to $abba$ since $abbb$ is not their maximal overlap.

Let $u$ and $v$ be in $P \cup \Omega(P)$. 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 $NbOcc(\tau, (u, v))$ as the number of occurrences of the arc $(u, v)$ in all Red-Blue graphs for all $\tilde{s}$ in $\tilde{P}$. Thus, we get:

$$NbOcc(\tau, (u, v)) := \sum_{\tilde{s} \in \tilde{P}} |\{(u, v) \cap (R \cup B)\}|. $$

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

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

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

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

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

**Multi superstring graph** Let $\tau$ be a permutation of $\tilde{P}$. 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

\[
\begin{array}{|c|c|c|c|}
\hline
(s, i) & \sigma_1((s, i)) & \text{RB-graph}(\sigma_1, (s, i)) \\
\hline
aab, 1 & aaba, 1 & aaba \\
aab, 2 & aaba, 2 & aaba \\
aab, 3 & aaba, 2 & aaba \\
abaa, 1 & aaba, 1 & aaba \\
abaa, 2 & aaba, 1 & aaba \\
abaa, 3 & aaba, 1 & aaba \\
abbb, 2 & aaba, 1 & aaba \\
abbb, 1 & aaba, 1 & aaba \\
abbb, 3 & aaba, 1 & aaba \\
\hline
\end{array}
\]
21:10 Superstrings with multiplicities

![Diagram of an overlap graph and a multi superstring graph]

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

elements of \( \tilde{P} \) and for permutation \( \tau \). Formally:

\[
\begin{align*}
V &= \mathcal{Ov}(P) \setminus U \\
R &= \{(u,v) \in \text{SuffixArc}(P) \mid \text{NbOcc}(\tau, (u,v)) \neq 0\} \\
B &= \{(u,v) \in \text{PrefixArc}(P) \mid \text{NbOcc}(\tau, (u,v)) \neq 0\} \\
l : (u,v) &\mapsto \text{NbOcc}(\tau, (u,v))
\end{align*}
\]

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

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

&gt; Proposition 12 (*). Let \( \tau_1, \tau_2 \) be two permutations of \( \text{GreedyPerm}(\tilde{P}) \). Then, \( G_p(\tau_1) = G_p(\tau_2) \).

By Proposition 12, all permutations inducing a greedy solution for an instance of \text{Multi-SCCS} yield the same graph, which we call the multi superstring graph and denote by \( \text{SG}(P,m) \) (see Figure 4b).

Using data structures like the (generalised) suffix tree to determine \( \mathcal{Ov}(P) \) [16], and with Proposition 11, we can build the multi superstring graph of \((P,m)\) recursively and we obtain the following proposition.

&gt; Proposition 13 (*). The multi superstring graph can be built in linear time and space in \( ||P|| \).

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

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

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 \text{Multi-SCCS}. As finding an Eulerian cycle cover of \( \text{SG}(P,m) \) takes a time linear in \( ||P|| \), we deduce Theorem 3 (p. 3).
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 \(c = (c_1, \ldots, c_n)\) 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 \(W\)

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
\(O(|P|^2)\) bits. In [8], the authors present a compact representation of a solution for
Multi-SLS 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 \(\tau\)
of \(P\), we denote \(LinCC(\tilde{P}, \tau, (w_1, \ldots, w_p))\) the following linearization

\[
LinCC(\tilde{P}, \tau, (w_1, \ldots, w_p)) = Lin(\tilde{P}_1, \sigma_1, w_1) \ldots Lin(\tilde{P}_p, \sigma_p, w_p)
\]

where \(\text{Decomp}(\tilde{P}, \tau) = \{(\tilde{P}_1, \sigma_1), \ldots, (\tilde{P}_p, \sigma_p)\}\) and \((w_1, \ldots, w_p) \in \tilde{P}_1 \times \ldots \times \tilde{P}_p\).

\[
\text{Figure 5 Running example: linearization } LinCC(\tilde{P}, \sigma_1, W) \text{ of a cyclic cover of strings induced by permutation } \sigma_1 \text{ (see Figure 2a)} \text{ for } W := (aba, 1, aab, 2, abba, 2, abba, 1, abba, 3)\text{.}
\]

Now, let us define the algorithm Concat-Greedy by Algorithm 3.
Superstrings with multiplicities

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 \(\text{GreedyPerm}(\tilde{P})\) corresponding to this multi-cycle;
4 take a tuple \(W\) of \(E_1 \times \ldots \times E_p\) where \(\text{Part}(\tilde{P},\tau) = \{E_1,\ldots,E_p\}\);
5 return \(\text{LinCC}(\tilde{P},\tau,W)\)

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.

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

\[
|w_{CG}| \leq |w_{OPT(\text{Multi-SLS})}| + 3 \times |w_{OPT(\text{SLS})}|.
\]

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 Multi-SLScomp 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

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 \( \tilde{P} \), one derives the 1/2 ratio for Multi-SLScomp. 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 Multi-SCCScomp, thereby showing that the greedy algorithm solves this problem exactly.

Proof of Lemma 16 and Proposition 11

Lemma 16. Let \( \tau \) be a permutation of \( \tilde{P} \) and \( \tilde{s} \in \tilde{P} \). Consider \( \text{RB-Graph}(\tau, \tilde{s}) := (V, R, B) \) be the Red-Blue graph of \( \tilde{s} \), 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,

$$\text{NbOcc}(\tau, (u, v)) := \sum_{\tilde{s} \in \tilde{P}} |\{(u, v) \cap (R \cup B)\}|$$

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

To simplify the proof, we consider four alternative cases.

**The case where $u$ is an element of $P$.** 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 $\{(w', w) \in \text{SuffixArc}(P)\} \cup \{(w, w') \in \text{PrefixArc}(P)\}$ is empty. And thus,

$$\text{NbOcc}(\tau, (u, v)) = |\{u \mid \exists k \in \mathbb{N}, (u, k) \in \tilde{P}\}| = m(u) = \text{Max}(m(u), a(u)).$$

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

$$\text{NbOcc}(\tau, (u, v)) = |\{v \mid \exists k \in \mathbb{N}, (v, k) \in \tilde{P}\}| = m(v) = \text{Max}(m(v), -a(v)).$$

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

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

As $\tau$ is a permutation of $\text{GreedyPerm}(\tilde{P})$, and assuming that the set

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

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

$$\text{NbOcc}(\tau, (u, v)) = |\{\tilde{s} \in \tilde{P} \mid u \text{ is a proper prefix of word}(\tau(\tilde{s}))\}| - |\{\tilde{s} \in \tilde{P} \mid u \text{ is a proper prefix of word}(\tau(\tilde{s})), |\text{ov(word}(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s})))| = |v|\}| - |\{\tilde{s} \in \tilde{P} \mid u \text{ is a proper prefix of word}(\tau(\tilde{s})), |\text{ov(word}(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s})))| > |v|\}| = \sum_{(v, w) \in \text{PrefixArc}(P)} |\{\tilde{s} \in \tilde{P} \mid u \text{ and } w \text{ are prefixes of word}(\tau(\tilde{s}))\}| - |\{\tilde{s} \in \tilde{P} \mid u \text{ and } w \text{ are prefixes of word}(\tau(\tilde{s})), |\text{ov(word}(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s})))| \geq |v|\}| - |\{\tilde{s} \in \tilde{P} \mid u \text{ is a proper prefix of word}(\tau(\tilde{s})), |\text{ov(word}(\text{word}(\tilde{s}), \text{word}(\tau(\tilde{s})))| = |v|\}| = \sum_{(v, w) \in \text{PrefixArc}(P)} \text{NbOcc}(\tau, (v, w)) - \sum_{(w', v) \in \text{SuffixArc}(P)} \text{NbOcc}(\tau, (w', v)).$$

Hence,

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

**The case where $u \notin P$, $v \notin P$ and $|u| > |v|$ 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. △
### Example: set of all Red-Blue graphs

<table>
<thead>
<tr>
<th>$(s, i)$</th>
<th>$\sigma_2((s, i))$</th>
<th>RB-graph($\sigma_2, (s, i)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aab, 1</td>
<td>aab, 1</td>
<td>$\text{aab} \rightarrow \text{ab} \rightarrow \text{abb} \rightarrow \text{abaa}$</td>
</tr>
<tr>
<td>aab, 2</td>
<td>aab, 2</td>
<td>$\text{aab} \rightarrow \text{ab} \rightarrow \text{abb} \rightarrow \text{abaa}$</td>
</tr>
<tr>
<td>aab, 3</td>
<td>aab, 3</td>
<td>$\text{aab} \rightarrow \text{ab} \rightarrow \text{abb} \rightarrow \text{abaa}$</td>
</tr>
<tr>
<td>abaa, 1</td>
<td>abaa, 2</td>
<td>$\text{abaa} \rightarrow \text{aa} \rightarrow \text{aab}$</td>
</tr>
<tr>
<td>abaa, 2</td>
<td>abaa, 3</td>
<td>$\text{abaa} \rightarrow \text{aa} \rightarrow \text{aab}$</td>
</tr>
<tr>
<td>ababb, 1</td>
<td>abba, 1</td>
<td>$\text{ababb} \rightarrow \text{abb} \rightarrow \text{abaa}$</td>
</tr>
<tr>
<td>abba, 1</td>
<td>abba, 1</td>
<td>$\text{abba} \rightarrow \text{aa} \rightarrow \text{aab}$</td>
</tr>
<tr>
<td>abba, 2</td>
<td>abba, 3</td>
<td>$\text{abba} \rightarrow \text{aa} \rightarrow \text{aab}$</td>
</tr>
<tr>
<td>abba, 3</td>
<td>ababb, 1</td>
<td>$\text{abba} \rightarrow \text{aa} \rightarrow \text{aab}$</td>
</tr>
</tbody>
</table>

**Figure 6** Running example: set of all the Red-Blue graphs of $(s, i) \in \tilde{P}$ for the permutation $\sigma_2$ (see Figure 2b).