Fixed-Parameter Algorithms for Scaffold Filling

Laurent Bulteau, Anna Paola Carrieri, Riccardo Dondi

Fixed-Parameter Algorithms for Scaffold Filling

Laurent Bulteau[1], Anna Paola Carrieri[2], and Riccardo Dondi[3]

[1] Department of Software Engineering and Theoretical Computer Science, Technische Universität Berlin, Berlin - Germany
[2] Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano - Italy
[3] Dipartimento di Scienze Umane e Sociali, Università degli Studi di Bergamo, Bergamo - Italy
l.bulteau@gmail.com, annapaola.carrieri@disco.unimib.it, riccardo.dondi@unibg.it

Abstract. In this paper we consider two combinatorial problems related to genome comparison. The two problems, starting from possibly incomplete genomes produced from sequencing data, aim to reconstruct the complete genomes by inserting a collection of missing genes. More precisely, in the first problem, called *One-sided scaffold filling*, we are given an incomplete genome B and a complete genome A, and we look for the insertion of missing genes into B with the goal of maximizing the common adjacencies between the resulting genome B' and A. In the second problem, called *Two-sided scaffold filling*, we are given two incomplete genomes A, B, and we look for the insertion of missing genes into both genomes so that the resulting genomes A' and B' have the same multi-set of genes, with the goal of maximizing the common adjacencies between A' and B'. While both problems are known to be NP-hard, their parameterized complexity when parameterized by the number of common adjacencies of the resulting genomes is still open. In this paper, we settle this open problem and we present fixed-parameter algorithms for the *One-sided scaffold filling* problem and the *Two-sided scaffold filling* problem.

1 Introduction

Genome comparison is a fundamental problem in bioinformatics, and it aims to identify differences and similarities among genomes, with the goal of understanding their function and evolutionary history. In this context several interesting combinatorial problems have been introduced (see for example [10]).

The introduction of new sequencing techniques (Next Generation Sequencing technologies, NGS) has led to a huge increase of the amount of DNA/RNA and protein sequences available for genomic and trascriptomic analyses [4]. These high-throughput sequencing technologies produce millions of short DNA/RNA reads that are joined together into longer sequences by means of assembly algorithms.

However, due to limitations of the NGS technologies, the cost of finishing a genome is still high compared to the cost of sequencing, hence most of the released genomes are unfinished and incomplete [4].

The use of incomplete draft genomes (called *scaffolds*) in genomic analyses may introduce errors. Hence, a relevant combinatorial problem is to fill the scaffolds with missing genes in order to obtain complete genomes that are as similar as possible to a given reference genome. Recently in [14] it has been introduced the *One-sided scaffold filling problem* that consists of filling a scaffold B in order to obtain a complete genome B' such that the Double-Cut and Join (DCJ) distance [16] (the minimum number of allowed rearrangement operations transforming one genome into the other) between B' and the reference genome A is minimized. Moreover, Jiang et al. in [11] considered the *Two-sided scaffold filling problem*, where the second genome A (on which the comparison is based) is incomplete as well.

In this paper we consider a different similarity measure, that is *the maximum number of common adjacencies* between two genomes, which has been introduced for the One-sided/Two-sided scaffold filling problems in [5]. Both problems are NP-hard under this similarity measure [12]. However, it has been shown that both problems admit constant factor approximation algorithms. In [12] it has been given a factor $\frac{4}{3}$ approximation algorithm for the One-sided scaffold filling problem and a factor 2 approximation algorithm for the Two-sided scaffold filling problem. The former approximation factor has been recently improved in [13], where it has been presented an approximation algorithm of factor $\frac{5}{4}$ for the One-sided scaffold filling problem.

In this paper, we focus on the parameterized complexity of the two scaffold filling problems. Parameterized complexity aims to characterize the complexity of a problem with respect to interesting parameters, with the goal of understanding if the exponential explosion of an exact algorithm can be confined only to the considered parameters. For an introduction to parameterized complexity we refer the reader to [8,15].

A first step in the analysis of the parameterized complexity of the One-sided scaffold filling problem has started in [12]. The authors presented two Fixed Parameter Tractable (FPT) algorithms for two special cases of the One-sided scaffold filling problem. In the first case, the number k of common adjacencies between a filled genome B' and a reference genome A, and the maximal number d of occurrences of a gene are considered as parameters, and it is presented an FPT algorithm of time complexity $O((2d)^{2k}poly(|A||B|))$. In the second case, the authors consider, as parameters, the number k of common adjacencies between a filled genome B' and a reference genome A and the size c of the set of symbols (genes) and they give an FPT algorithm that runs in time $O(c^{2k}poly(|A||B|))$. However, the parameterized complexity of the One/Two-sided scaffold filling problems, when parameterized only by the maximum number of common adjacencies k, has been left open in [12].

Our contribution. In this paper we present two FTP-algorithms for both problems, thus answering an open question in [12]. More precisely, we present an algorithm of time complexity $2^{O(k)}poly(|A||B|)$ for One-sided scaffold filling and an algorithm of time complexity $2^{O(k\cdot\log k)}poly(|A||B|)$ for Two-sided scaffold filling.

The rest of the paper is organized as follows. First, in Section 2 we introduce some preliminary definitions and we formally define the two combinatorial problems we are interested in. Then, in Section 3, we describe the FPT algorithm for the One-sided case, while in Section 4 we present the FPT algorithm for the Two-sided case. We conclude the paper with some open problems. Some of the proofs are omitted due to page limit.

2 Preliminaries

Let Σ be a non-empty finite set of symbols. An (unsigned) unichromosomal genome A is represented as a string over an alphabet Σ, where the symbols in A (that are genes) form a multiset $[A]$ on Σ. For example $A = abcdabcdaa$ with $\Sigma = \{a, b, c, d\}$ and $[A] = \{a, a, a, a, b, b, c, c, d, d\}$. We write $A[i]$ for the symbol of A in i-th position, and we write $A[i \dots j]$ for the substring of A between positions i and j.

Given a string A, an *adjacency* is an unordered pair of consecutive elements of A ($A[i]A[i+1]$ or $A[i+1]A[i]$). A position i induces an adjacency ab, if either $A[i] = a$, $A[i+1] = b$ or $A[i] = b$ and $A[i+1] = a$. The endpoints of A, are the first and the last position of A.

We write $[\![A]\!]$ for the multi-set of adjacencies of A (i.e., if $A = abcdabcdaa$, then $[\![A]\!] = \{aa, ab, ab, ad, ad, bc, bc, cd, cd\}$).

In order to deal with endpoints of the two strings, we assume that given a string A, with $|A| = n$, $A[1] = A[n] = \sharp$, where $\sharp$ is a dummy symbol. The dummy symbols are not considered when defining the set of adjacencies $[\![A]\!]$, i.e. $\sharp A[1]$ and $A[n]\sharp$ are not in $[\![A]\!]$.

Comparing two strings A and B, we denote by $X = [A] \setminus [B]$ the multi-set of symbols of A missing in B, and by $Y = [B] \setminus [A]$ the multi-set of symbols of B missing in A. Given a multi-set of symbols $[G]$ over alphabet Σ, a *scaffold* is a string on $[G]$ with some missing elements.

The two scaffold filling problems we will deal with are based on the definition of *common adjacency* between two genomes (strings).

Definition 1. *Consider two strings A, B on alphabet Σ. The multi-set of* common adjacencies *between A, B is defined as $[\![A]\!] \cap [\![B]\!]$. A* matching *M of the adjacencies of A and the adjacencies of B is a relation between the positions of A and the positions of B such that:*

- *for each position i of A or B, there exists at most one pair in M containing i;*
- *for each position i of A and j of B, $(i, j) \in M$ if and only if position i and position j induces the same adjacency;*
- *each position that induces a common adjacency belongs to some pair of M.*

A position of A or B is *matched*, if it belongs to a pair of M. Informally, M relates the positions inducing common adjacencies.

Given a scaffold B and a multi-set of symbols X, a string B' is a *filling* of B with X if (1) $[B'] = [B] \cup X$, and (2) B is a subsequence of B' such that the first and last symbols of B' are respectively the first and last symbols of B.

In the following we give the definitions of the two Scaffold Filling problems (parameterized versions) investigated in this paper.

One-sided Scaffold Filling to Maximize the Number of common String Adjacencies (One-sided SF-MNSA)
Input: Two strings A and B, such that $[B] \subseteq [A]$.
Output: A filling B' of B with $X = [A] \setminus [B]$ such that A and B' have at least k common adjacencies.
Parameter: k.

Two-sided Scaffold Filling to Maximize the Number of common String Adjacencies (Two-sided SF-MNSA)
Input: Two strings A and B.
Output: A filling B' of B with $X = [A] \setminus [B]$ and a filling A' of A with $Y = [B] \setminus [A]$ such that A' and B' have at least k common adjacencies.
Parameter: k.

Notice that the restriction of Two-sided SF-MNSA with $Y = \emptyset$ is exactly the One-sided SF-MNSA problem.

Now, we discuss some properties that will be useful to design our FPT-algorithms. First, we present the following property for the parameter k, proved in [12].

Lemma 1 *[12] Let A and B two strings on an alphabet Σ, $X = [A] \setminus [B]$, and $Y = [B] \setminus [A]$. Let k be the optimal number of common adjacencies for Two-sided SF-MNSA between two fillings A' and B'. Then $|X|, |Y| \leq k$.*

Notice that Lemma 1 holds also for One-sided SF-MNSA, that is when $Y = \emptyset$, it holds $|X| \leq k$.

Let A and B be two strings of symbols over an alphabet Σ, which are input of One-sided SF-MNSA or Two-sided SF-MNSA. Consider now the set AD of common adjacencies between A and B. Notice that we can assume that $|AD| < k$, otherwise we already know that One-sided SF-MNSA/Two-sided SF-MNSA admits a solution consisting of at least k common adjacencies. Now, we can compute a partition of AD into two subsets as follows:

- the set $AD_{pr} \subseteq AD$ of common adjacencies that are preserved after the filling of B and/or A;
- the set $AD_{br} \subseteq AD$ of common adjacencies that are broken by inserting symbols of $X = [A] \setminus [B]$ (of $Y = [B] \setminus [A]$ respectively) into B (into A respectively).

Then the following easy property holds.

Property 1 *Let A and B be two strings of symbols over alphabet Σ and let AD be the set of common adjacencies between A and B. Then, if there exists a*

solution for the One-sided SF-MNSA/Two-sided SFMNSA that partitions the set AD into the sets AD_{pr}, AD_{br}, we can compute the partition of AD into the two subsets AD_{pr} and AD_{br} in time $O(2^k)$.

This property is implicitly used in the two fixed-parameter algorithms to guess which adjacencies of the set AD will be preserved, that is those adjacencies induced by positions where no insertion is possible when computing a filling of an input string. Hence, in what follows, we assume that when a string is inserted into A or B, then it is not inserted in a position associated with an adjacency in AD_{pr}.

Color Coding. The FPT-algorithms we present are mainly based on the *color-coding* technique and on the perfect family of hash functions [1]. Color-coding is a well-known technique for designing fixed-parameter algorithms, and it has been applied to several combinatorial problems, for example for the longest path problem [1], for the graph motif problem [9,6,2,7] and for problems on strings [3].

Informally, given a set U of size n, color-coding aims to identify a subset $S \subseteq U$ of size k by coloring the elements of U with k colors, so that each element in S is associated with a distinct color. While enumerating the subsets having size k of U takes time $O(n^k)$, by means of the coloring and using combinatorial properties of the problem, in some cases it is possible to compute whether a solution of size k exists in time $f(k)poly(n)$, thus leading to an FPT algorithm.

We now introduce the definition of a perfect family of hash functions, which are used to compute the coloring.

Definition 2. *Let I be a set, a family F of hash functions from I to $\{c_1, \ldots, c_k\}$ is called* perfect *if for any subset $I' \subseteq I$, with $|I'| = k$, there exists a function $f \in F$ which is injective on I'.*

A perfect family F of hash functions from I to $\{c_1, \ldots, c_k\}$, having size $O(\log |I| 2^{O(k)})$, can be constructed in time $O(2^{O(k)}|I| \log |I|)$ [1].

3 An FPT algorithm for One-sided SF-MNSA

In this section we present an FPT algorithm for One-sided SF-MNSA parameterized by k, the number of common adjacencies between the input string A and the filling B' of B with the multi-set X of symbols of A missing in B. Recall that, by Lemma 1, it holds $|X| \leq k$. Furthermore, we assume that we have already computed the subset AD_{pr} of AD (the common adjacencies of A, B) where no insertion is possible during the filling (see Prop. 1).

Let $C_A = \{c_1, \ldots, c_k\}$ be a set of *colors*. Consider a family F of perfect hash functions from the positions inducing the adjacencies of A in AD_{br} to colors in C_A. Informally, the coloring is used to identify a matching of the positions of A and the positions of B that induce new adjacencies due to the insertion of symbols in X.

In the following, we assume that the coloring of the positions of A is induced by some injective function $f \in F$. Given a string S, S is *colorful for* C_A if there exist $\{s_c \mid c \in C_A\} \subseteq [\![S]\!]$ such that for each $c \in C_A$ there is a position of A colored by c which induces the adjacency s_c. Our objective is thus to compute a filling of B colorful for C_A.

We first focus on inserting a set of elements at one given position of B. Given j a position in B, $X_j \subseteq X$ and $C_j \subseteq C_A$, define $\texttt{Ins}_j(X_j, C_j)$ as follows:

$$\texttt{Ins}_j(X_j, C_j) = \begin{cases} 1 & \text{if there exists a filling of } B[j-1, j] \text{ with } X_j \text{ which is} \\ & \text{colorful for } C_j, \\ 0 & \text{else.} \end{cases}$$

Using dynamic programming, $\texttt{Ins}$ can be computed in time $O(2^{2k}k^2)$ yielding the following lemma.

Lemma 2 *Let $X_j \subseteq X$, $C_j \subseteq C_A$, such that $|X_j|, |C_j| \leq k$ and j be an integer s.t. $j \leq |B|$. Then we can compute $\texttt{Ins}_j(X_j, C_j)$ in time $O(2^{2k}k^2)$.*

We now define a table $\texttt{Fill}_j(X', C'_A)$ computed by the following recurrence. The objective, as stated in Lemma 3, is to determine whether a prefix of B can be filled with any given subset of X so as to be colorful for any given subset of C_A.

Recurrence 1 *Let $X' \subseteq X$, $C'_A \subseteq C_A$.*

- *For $j = 1$, let $\texttt{Fill}_1(X', C'_A) = \texttt{Ins}_1(X', C'_A)$.*
- *For all $j \geq 2$, let:*

$$\texttt{Fill}_j(X', C'_A) = \max_{X_j \subseteq X', C_j \subseteq C'_A} \begin{cases} \texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) \\ \wedge\, \texttt{Ins}_j(X_j, C_j) \end{cases}$$

In the following, we prove that $\texttt{Fill}_{|B|}(X, C_A)$ allows us to determine whether B admits a filling with k common adjacencies.

Lemma 3 *Let (A, B) be an instance of One-sided Scaffold Filling, $X = [A] \setminus [B]$, k be an integer, C_A be a set of k colors, and F be a perfect family of hash functions from the positions of A to C_A. Then the following propositions are equivalent:*
(i) There exists a filling B' of B with X such that A and B' have k common adjacencies;
(ii) There exists a coloring $f \in F$ for which $\texttt{Fill}_{|B|}(X, C_A) = 1$.

Next, we show how the recurrence described in Recurrence 1 yields a dynamic programming algorithm to solve One-sided SF-MNSA.

Theorem 1 *Let A, B be two strings of symbols on an alphabet Σ and let $X = [A] \setminus [B]$ be the multiset of symbols missing in B. It is possible to compute a solution of One-sided SF-MNSA in time $O(2^{O(k)}poly(|A| + |B|))$.*

Proof. Recurrence 1 yields a dynamic programming algorithm: for each j from 1 to $n+1$, compute the entry $\texttt{Fill}_j\,(X', C'_A)$ for each set $X' \subseteq X$ and $C'_A \subseteq C_A$. Then, by Lemma 3, there exists a filling B' of B creating $|C_A| = k$ common adjaciencies if and only if $\texttt{Fill}_{|B|}\,(X, C_A) = 1$.

Now, we consider the time complexity of the algorithm. Write $n = |A| + |B|$. First, a perfect family of hash functions that color-codes the positions of A can be computed in time $2^{O(k)}poly(n)$. Once the family is computed, there are $2^{O(k)}\log(n)$ color codings to iterate through. For each color coding, the table $\texttt{Fill}_j\,(X', C'_A)$ is computed in time $O(2^{2k}k^2n)$ (see Lemma 2). Then the $O(2^{2k}n)$ entries of table $\texttt{Fill}_j\,(X', C'_A)$ are computed, where each entry requires $O(2^{2k})$ look-ups, depending on the choice of X_j and C_j. Thus Recurrence 2 requires $O(2^{4k}n)$ to compute table $\texttt{Fill}_j\,(X', C'_A)$. Finally, the overall complexity is indeed $2^{O(k)}poly(n)$. □

4 An FPT algorithm for Two-sided SF-MNSA

In this section, we consider the Two-sided SF-MNSA problem and we give a fixed-parameter tractable algorithm for it. As for the One-sided case, the algorithm is based on color-coding and dynamic programming. However, new challenges appear which make the problem more complicated. First, there exist a new kind of common adjacencies: with adjacencies that are created in the fillings although they never appear as such in the input strings. Also, unlike the One-sided case, it is not known a priori whether a given adjacency may be used in a common adjacency or should be split to insert a substring. We deal with the first issue by bounding (and enumerating) the possible arrangments of such rare adjacencies, and with the second by introducing "insertion" colors, where corresponding adjacencies can only be used to insert a substring, not to create a common adjacency.

Given two strings A and B over alphabet Σ, denote by k the number of common adjacencies between two fillings A' and B' of A, B respectively. Let $X = [A] \setminus [B]$ be the multi-set of symbols of A missing in B and let $Y = [B] \setminus [A]$ be the multi-set of symbols of B missing in A, where $X, Y \neq \emptyset$ (otherwise the problem is equivalent to One-sided SF-MNSA) and $X \bigcap Y = \emptyset$ (by the definition of sets X and Y for the Two-sided SF-MNSA).

Recall that, by Lemma 1, the following property holds: $|X|, |Y| \leq k$. Furthermore, as in the previous section, we assume that we have already computed the subset AD_{pr} of $[\![A]\!] \cap [\![B]\!]$, that is those common adjacencies of A, B, that must be preserved during the filling (see Prop. 1).

Before giving the details of the FPT-algorithm, we present an (informal) overview. A filling B' (A' respectively) of B (of A respectively) consists of inserting substrings over alphabet X (over alphabet Y respectively) into B (into A respectively). In the first step, the algorithm "guesses" how these insereted strings are formed from X and Y (since $|X|, |Y| \leq k$, the number of cases to try depends only on a function of k, see Prop. 2).

We now identify two kinds of common adjacencies for two fillings A', B'. In the first kind, one adjacency appears already in $[\![A]\!]$ or $[\![B]\!]$: this case can be dealt with as in the one-sided algorithm. In the second kind, both adjacencies have been created during the filling, using one element from X in B' and one from Y in A'. They are called (X, Y)-adjacencies. Since $X \cap Y = \emptyset$, such adjacencies use *exactly* one element of X and Y, hence they consist of an endpoint of an inserted string as well as a letter already present in the original strings A and B. The second step of the algorithm consists in identifying and matching the endpoints of strings which corespond to such (X, Y)-adjacencies (see Def. 4 and Prop. 3).

In Step 3 the algorithm opportunely color-codes the positions of A and B in order to (i) match non (X, Y)-adjacencies (like in the previous algorithm), and (ii) identify the positions of A and B where an insertion is possible (we will show that the number of these positions is bounded by k in Remark 2). This allows, in Step 4, to finally insert the strings into A and B by dynamic programming while creating the remaining adjacencies (see Recurrence. 2).

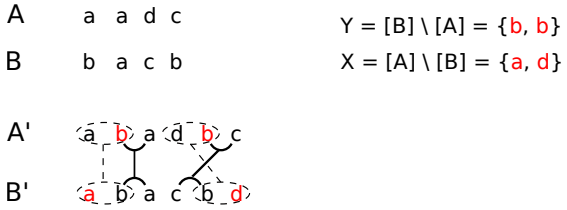

Fig. 1: An instance of the Two-sided SFMNA problem. Given two scaffolds A and B, we obtain the filled genomes A' and B' by inserting symbols X in B and Y in A (inserted symbols are in red). Lines connect common adjacencies, dotted lines connect (X,Y)-adjacencies.

We can now present the details of the algorithm.

Step 1: *Compute inserted strings.*

Let S_X and S_Y be the two multi-sets of strings over the multi-sets X and Y that have to be inserted in B and A respectively in an optimal solution. The algorithm simply iterates through all such pairs (S_X, S_Y) of multi-sets of strings over (X, Y): in some iteration, the correct pair (S_X, S_Y) is clearly considered. The following property bounds both the number of possible pairs (S_X, S_Y) and the number of positions where strings can be inserted in A and B.

Property 2 *Let X, Y be two multi-sets of symbols to be inserted into the strings B and A respectively. Then (1) the number of positions in each of A, B where a string of S_Y, S_X is inserted is bounded by k and (2) the number of possible multi-sets S_X and S_Y of strings over X, Y to be inserted into B and A respectively is bounded by $O(k^{2k})$.*

Step 2: *Identify (X, Y)-adjacencies.*

We first define formally the concept of (X, Y)-*adjacency* (see Fig. 1 for an example).

Definition 3. *Consider a filling B' of B with X and a filling A' of A with Y. A common adjacency $z \in [\![A']\!] \cup [\![B']\!]$ is an* (X, Y)-adjacency *if it is induced by positions i, j of A', B' respectively, and either $A'[i]$ or $A'[i+1]$ is the endpoint of an inserted string $s_x \in S_X$, and either $B'[j]$ or $B'[j+1]$ is the endpoint of an inserted string $s_y \in S_Y$.*

Notice that, since $X \cap Y = \emptyset$, it follows that any new common adjacency of A' (of B' respectively) is either not involved in an insertion (hence, in one string, it is induced by a position where no string is inserted), or it is an (X, Y)-adjacency.

Now, the algorithm defines which endpoints of the strings in S_X, S_Y induce a common (X, Y)-adjacency. Denote by E_X (E_Y respectively), the set of endpoints of the strings in set S_X (in set S_Y respectively). We consider a procedure, called *number assignment*, that associates with each endpoint in E_X and E_Y a number which identifies the (X, Y)-adjacency, if any, which uses this endpoint.

Definition 4. *A* number assignment *for the strings in $S_X \cup S_Y$ is the data of an integer k' and of a function from $E_X \cup E_Y$ to $\{0, 1, \ldots, k'\}$, where each number $\{1, \ldots, k'\}$ is assigned to exactly one endpoint in E_X and one endpoint in E_Y.*

Consider a solution, a corresponding number assignment is obtained as follows. Let k' be the number of (X, Y)-adjacencies. Consider an endpoint $e_z \in E_X \cup E_Y$, then:

- Endpoint e_z is associated with 0 iff it is not involved in an (X, Y)-adjacency;
- Endpoint e_z is associated with a number $i \in \{1, \ldots, k'\}$ iff it is involved in the i-th (X, Y)-adjacency.

The set $E'_X \subseteq E_X$ ($E'_Y \subseteq E_Y$) denotes the set of endpoints of E_X (of E_Y respectively) associated with a positive number.

The following property gives an easy upper bound on the number of such assignments.

Property 3 *There are at most $(2k)^{k+1}$ number assignments.*

Hence, in what follows assume that the algorithm guesses the correct number assignment to $E_X \cup E_Y$. Now, we show how we can bound the possible symbols that are adjacent to an endpoint in $E'_X \cup E'_Y$. First, we introduce the following definition.

Definition 5. *Consider a string $s_x \in S_X$ ($s_y \in S_Y$ respectively). Let $e_x \in E'_X$ ($e_y \in E'_Y$ respectively) be an endpoint of s_x (of s_y respectively). Then, $v(e_x)$ ($v(e_y)$ respectively) is the symbol of Y (of X respectively) adjacent to e_x in B' (to e_y in A' respectively).*

Notice that the number assignment immediately defines the values $v(e_x)$, $v(e_y)$, for each $e_x \in E'_X$, $e_y \in E'_Y$. Indeed, if $e_x \in E'_X$ and $e_y \in E'_Y$ are associated with the same number i, then $v(e_x)$ must be the symbol contained in $s_y[e_y]$, while $v(e_y)$ must be the symbol contained in $s_x[e_x]$.

Remark 1 *A number assignment uniquely determines the value $v(e_Z)$ for $e_z \in E'_X \cup E'_Y$.*

Using this value, the algorithm creates the following table which tells whether or not, according to (X, Y)-adjacencies, a string can be inserted at a certain position. Let Z be an input string among A, B, $s \in S_Z$, and $j \in \{1, \ldots, |Z|\}$. Write s_l and s_r for the left and right endpoints of s respectively:

$$\texttt{XY-Fits}_j\,(Z, s) = \begin{cases} 0 & \text{if } (s_l \in E'_Z \text{ and } Z[j-1] \neq v(s_l)) \\ & \text{or } (s_r \in E'_Z \text{ and } Z[j] \neq v(s_r)) \\ 1 & \text{otherwise.} \end{cases}$$

Step 3: *Color-code the positions in A and B.*

We are now able to define the color-coding of the positions of A and B. Consider a coloring f of the positions of A and B with a set C of z, $z \leq 2k$, colors. Moreover, we partition C into disjoint subsets $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{I,B}$ defined as follows:

- Let $C_{M,B}$ ($C_{M,A}$ respectively) be a set of colors associated with positions of B (of A respectively) that matches positions of A' (of B' respectively). Notice that in a position colored by $C_{M,A}$ ($C_{M,B}$ respectively) a string of S_X (of S_Y respectively) cannot be inserted.
- Let $C_{I,B}$ ($C_{I,A}$ respectively) be a set of colors assigned to positions in B (in A respectively) where insertions of strings of S_X (of S_Y respectively) are allowed.

Note that, since S_X, S_Y and the number assignment with k' (X, Y)-adjacencies are fixed, we only consider partitions where $|C_{I,A}| = |S_Y|$, $|C_{I,B}| = |S_X|$, and $|C_{M,A}| + |C_{M,B}| + k' = k$.

There are k values of z to test. For each z, there are $O(2^{O(z)} \log n)$ colorings [1], and for each coloring, 4^z ways of partitionning C into $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{I,B}$. Overall, there are thus $O(2^{O(k)} \log n)$ cases to consider.

Step 4: *Insert strings by dynamic programming.*

Now, we can define the dynamic programming recurrence. Similarly to the One-sided case, we define $\texttt{Ins}_{Z,j}\,(s, C_{M,j})$, where $C'_{M,j} \subseteq C_{M,W}$ and W, Z are different strings of $\{A, B\}$, as follows:

$$\texttt{Ins}_{Z,j}\,(s, C_{M,j}) = \begin{cases} 1 & \text{if } \texttt{XY-Fits}_j\,(Z, s) = 1 \text{ and the string } Z[j-1]sZ[j] \\ & \text{is colorful for } C_{M,j}, \\ 0 & \text{otherwise.} \end{cases}$$

Similary to Property 2, any entry $\texttt{Ins}_{Z,j}\left(s, C'_{M,W}\right)$ can be computed in time $O(2^{2k}n)$.

Lemma 4 *Let $C'_{M,W} \subseteq C_{M,W}$, and j be an integer s.t. $j \leq |W|$. Then we can compute $\texttt{Ins}_{Z,j}\left(s, C'_{M,W}\right)$ in time $O(2^{2k}n)$.*

We can now compute a filling of B satisfying all the above constraints. We define the following table $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}, C'_{I,B}\right)$ for each $S'_X \subseteq S_X$, $C'_{M,A} \subseteq C_{M,A}$, $C'_{I,B} \subseteq C_{I,B}$ and $0 \leq j \leq |B|$.

Recurrence 2 *Let $S'_X \subseteq S_X$, $C'_{M,A} \subseteq C_{M,A}$, $C'_{I,B} \subseteq C_{I,B}$*

- *For $j = 0$, $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}, C'_{I,B}\right) = 1$ iff $S'_X = C'_{M,A} = C'_{I,B} = \emptyset$.*
- *For all $j \geq 1$, $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}, C'_{I,B}\right) = 1$ iff one of the following is true:*
  - $\texttt{Fill-B}_{j-1}\left(S'_X, C'_{M,A}, C'_{I,B}\right) = 1$.
  - *$f(j) \in C'_{I,B}$ and $\exists s_x \in S'_X, C_{M,j} \subseteq C'_{M,A}$ such that*
    $$\begin{cases} \texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_x\}, C'_{M,A} \setminus C_{M,j}, C'_{I,B} \setminus \{f(j)\}\right) \\ \wedge\, \texttt{Ins}_{B,j}\left(s_x, C_{M,j}\right) \end{cases}$$

A filling of A is computed using a table $\texttt{Fill-A}_j\left(S'_Y, C_{M,B}, C_{I,A}\right)$ defined similarly.

Lemma 5 *Let (A, B) be an instance of One-sided Scaffold Filling, $X = [A] \setminus [B]$, $Y = [B] \setminus [A]$, k be an integer, C be a set of colors, and F be a perfect family of hash functions from the positions of A and B to C. Then the following propositions are equivalent:*
(i) There exists a filling A' of A with Y and a filling B' of B with X such that A' and B' have k common adjacencies;
(ii) There exist two multi-sets of strings S_X and S_Y over X, Y, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ such that $\texttt{Fill-A}_{|A|}\left(S_Y, C_{M,B}, C_{I,A}\right) = \texttt{Fill-B}_{|B|}\left(S_X, C_{M,A}, C_{I,B}\right) = 1$.

We present now the main result of this section.

Theorem 2 *Let A, B be two strings over alphabet Σ and let $X = [A] \setminus [B]$ be the multiset of symbols of A missing in B and $Y = [B] \setminus [A]$ the multiset of symbols of B missing in A. It is possible to compute a solution of Two-sided SFMNSA over instance (A, B) in time $2^{O(k \log k)} poly(n)$.*

Proof. The correctness of the algorithm is directly given by Lemma 5: once a perfect family of hash functions F is fixed and two multi-sets of strings S_X and S_Y over X, Y, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ are selected by exhaustive branching, it suffices to compute the entries $\texttt{Fill-A}_{|A|}\left(S_Y, C_{M,B}, C_{I,A}\right)$ and $\texttt{Fill-B}_{|B|}\left(S_X, C_{M,A}, C_{I,B}\right)$, and return the corresponding fillings of A and B if both entries are equal to 1.

The time complexity of the algorithm is dominated by the iteration over all possible pairs (S_X, S_Y) and of the number assignment. The number of possible sets S_X, S_Y is bounded by k^{2k} from Prop. 2. By Prop. 3 there are $O(2k^{k+1})$ number assignments to iterate through. The dynamic programming recurrence requires time $O(2^{4k}n)$. Since a family of perfect hash function of size $O(2^{O(k)} poly(n))$ can be computed in time $O(2^{O(k)} poly(n))$ [1], and the possible partitions of C into sets $C_{M,A}$, $C_{M,B}$, $C_{I,A}$, $C_{M,B}$ are less than 2^{4k} (including the constraint $|C_{M,A}| + |C_{M,B}| + k' = k$), it follows that the overall time complexity of the algorithm is bounded by $O((2k)^{2k+1} 2^{O(k)} poly(n)) = 2^{O(k \log k)} poly(n)$. □

5 Conclusion

In this paper we presented two FPT algorithms for the One-sided SF-MNSA problem and the Two-sided SF-MNSA problem. There are some interesting open problems from an algorithmic perspective. First, it would be interesting to improve upon the time complexity of the algorithms we presented. Moreover, the approximation complexity of the Scaffold Filling problems, in particular of the Two-sided case, should be further investigated. An interesting open problem in this direction is whether it is possible to design an approximation algorithm for Two-sided SF-MNSA with approximation factor better than 2.

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. Journal of the ACM 42(4), 844–856 (1995)
2. Betzler, N., van Bevern, R., Fellows, M.R., Komusiewicz, C., Niedermeier, R.: Parameterized algorithmics for finding connected motifs in biological networks. IEEE/ACM Trans. Comput. Biology Bioinform. 8(5), 1296–1308 (2011)
3. Bonizzoni, P., Della Vedova, G., Dondi, R., Pirola, Y.: Variants of constrained longest common subsequence. Inf. Process. Lett. 110(20), 877–881 (2010)
4. Chain, P., Grafham, D., Fulton, R., Fitzgerald, M., Hostetler, J., Muzny, D., Ali, J., et al: Genome project standards in a new era of sequencing. Science 326, 236–237 (2009)
5. Chen, Z., Fu, B., Yang, B., Xu, J., Zhao, Z., Zhu, B.: Non-breaking similarity of genomes with gene repetitions. Proc. of the 18th Symposium on Combinatorial Pattern Matching (CPM 2007) LNCS 4580, 119–130 (2007)
6. Dondi, R., Fertin, G., Vialette, S.: Complexity issues in vertex-colored graph pattern matching. J. Discrete Algorithms 9(1), 82–99 (2011)
7. Dondi, R., Fertin, G., Vialette, S.: Finding approximate and constrained motifs in graphs. Theor. Comput. Sci. 483, 10–21 (2013)
8. Downey, R., Fellows, M.: Parameterized Complexity. Springer (1999)
9. Fellows, M.R., Fertin, G., Hermelin, D., Vialette, S.: Upper and lower bounds for finding connected motifs in vertex-colored graphs. J. Comput. Syst. Sci. 77(4), 799–811 (2011)
10. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of genome rearrangements. The MIT Press, Cambridge (2009)
11. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint distance. In: Tannier, E. (ed.) RECOMB-CG 2010 LNCS, 6398, 83–92 (2010)
12. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint and related distances. IEEE/ACM Trans. Comput. Biology Bioinform 9(4), 1220–1229 (2012)
13. Liu, N., Jiang, H., Zhu, D., Zhu, B.: Approximation algorithm for scaffold filling to maximize the common adjacencies. COCOON 2013 pp. 397–408 (2013)
14. Muñoz, A., Zheng, C., Zhu, Q., Albert, V., Rounsley, S., Sankoff, D.: Scaffold filling, contig fusion and gene order comparison. BMC Bioinformatics 11, 304 (2010)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
16. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)