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. The new sequencing technologies, called *next-generation sequencing*, provide a huge amount of data that can be used to reconstruct genomes. However, the methods applied to reconstruct genomes often are not able to reconstruct a complete genome and provide only an incomplete information. Here we consider two combinatorial problems that aim to reconstruct complete genomes by inserting a collection of missing genes. The first problem we consider, called *One-sided scaffold filling*, given an incomplete genome B and a complete genome A, asks for the insertion of missing genes into an incomplete genome B with the goal of maximizing the common adjacencies between genomes B' (resulting from the insertion of missing genes in B) and A. The second problem, called *Two-sided scaffold filling*, given two incomplete genomes A, B, asks for the insertion of missing genes into both genomes so that the resulting genomes A' and B' have the same multiset of genes and the number of common adjacencies between A' and B' is maximized. Both problems were proved to be NP-hard, while their parameterized complexity, when the parameter is the number of common adjacencies of the resulting genomes, was left as an open problem. In this paper, we settle this open problem by presenting fixed-parameter algorithms for *One-sided scaffold filling* and *Two-sided scaffold filling*.

1 Introduction

Comparative genomics is a widely investigated field of bioinformatics in which the genomic features of different organisms are compared in order to identify biological differences and similarities. The genomic features include DNA sequences, genes, regulatory sequences and other genomic structural landmarks [22]. The ultimate goal of the approaches in this field is to understand genome functions, relationships between organisms and their evolutionary history. In this context, several interesting combinatorial problems have been introduced and studied by the computer science community (see for example [13]).

* A preliminary version of this paper appeared in the proceedings of ISCO 2014

The introduction of Next Generation Sequencing (NGS) technologies lead to a huge increase on the amount of DNA/RNA and protein sequences available for genomic and trascriptomic analyses [7]. NGS technologies produce millions of short DNA/RNA fragments, called reads, that are joined together to reconstruct longer sequences. While NGS technologies generate such a huge amount of data, the cost of obtaining a complete genome is still high, in particular if compared to the cost of sequencing. Due to this fact, often released genomes are unfinished and incomplete [7]. When used in genomic analyses, incomplete draft genomes (also called *scaffolds*) may introduce errors. Hence, a relevant problem for genome comparison is the filling of scaffolds with missing genes, by mean of combinatorial algorithms, in order to reconstruct complete genomes that share a high level of similarities with a known reference genome.

A combinatorial problem that has been introduced recently is the *One-sided scaffold filling problem* [20]. Such problem consists of filling a scaffold B so that the resulting complete genome B' minimizes the Double-Cut and Join (DCJ) distance [23] with respect to the reference genome A. Given two genomes, the DCJ distance is the minimum number of allowed rearrangement operations that transform one genome into the other. The authors presented a polynomial-time algorithm for the problem when the input genomes do not contain duplicated genes.

Later in [16], the scaffold filling problem has been investigated considering both the DCJ distance and the *breakpoint distance*. Given two related sequences A and B, two consecutive elements a_i and a_{i+1} in A form an *adjacency* if they are also consecutive in B independently from the order (i.e., as $a_i a_{i+1}$ or $a_{i+1} a_i$), otherwise they form a *breakpoint*. Therefore, the breakpoint distance between A and B is defined as the number of breakpoints in A, which is equal to that of B. In [16] Jiang et al. introduced a new related variant of the combinatorial problem, called *Two-sided scaffold filling problem*, where both genomes are incomplete. The authors show that when the input genomes do not contain gene repetitions the problem is polynomially solvable under both the DCJ distance and the breakpoint distance. However, when genomes contain duplicated genes, the scenario changes. Indeed, the authors showed that the One-sided problem is NP-complete even under the breakpoint distance.

In this paper we consider a different similarity measure to compare genomes, namely *the maximum number of common adjacencies* between two genomes. This measure has been introduced for the One-sided/Two-sided scaffold filling problems in [8]. The two problems were both proved to be NP-hard under this similarity measure [17]. The same paper has investigated the approximation complexity of the two problems, showing a 2-approximation algorithm for the Two-sided scaffold filling problem, and a $\frac{4}{3}$-approximation algorithm for the One-sided scaffold filling problem. This latter result has been recently improved in [18], where an approximation algorithm of factor $\frac{5}{4}$ for the One-sided scaffold filling problem has been presented. An approximation algorithm for a related variant of the problem has been given in [19].

In this paper, we focus on the parameterized complexity of One-sided scaffold filling and Two-sided scaffold filling. Parameterized complexity aims to investigate the computational complexity of a problem with respect to a set of interesting parameters, with the goal of understanding if the exponential explosion of an exact algorithm can eventually be confined only to the considered parameters (and not to the overall input). We refer the reader to [11,21] for an introduction to parameterized complexity.

A preliminary analysis of the parameterized complexity of the One-sided scaffold filling problem started in [17]. The authors presented two Fixed Parameter Tractable (FPT) algorithms for One-sided scaffold filling, under two different parameterizations. In the first case, they considered as parameters 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 inside a genome, and gave an FPT algorithm of time complexity $O((2d)^{2k}poly(|A||B|))$. In the second case, the authors considered as parameters the number k of common adjacencies between a filled genome B' and a reference genome A and the size c of the alphabet (that is the set of genes), and gave an FPT algorithm of time complexity $O(c^{2k}poly(|A||B|))$. A natural problem, left open in [17], is to consider the parameterized complexity of One-sided and Two-sided scaffold filling, when parameterized only by the maximum number of common adjacencies k.

Our contribution. In this paper we present two FTP-algorithms for both Scaffold Filling problems, thus answering the open question in [17]. More precisely, we give 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, where k is the number of common adjacencies between the resulting genomes (A and B' for the One-sided case, A' and B' for the Two-sided case). We point out that the contribution of the paper is mainly theoretical, since in practice the parameter k is often close to the length of the genomes.

The rest of the paper is organized as follows. First, in Section 2 we introduce some preliminary definitions that will be useful in the rest of the paper and we give the formal definition of the two Scaffold Filling problems. In Section 3, we present 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 possible future directions.

2 Preliminaries

Let Σ be an *alphabet*, that is a non-empty finite set of symbols. We represent an (unsigned) unichromosomal genome A as a string over alphabet Σ. It follows that the symbols in A (where each symbol represents a gene) form a multiset on Σ, denoted by $[A]$. Consider, for example, the string $A = abcdabcdaa$ on alphabet $\Sigma = \{a, b, c, d\}$, then $[A] = \{a, a, a, a, b, b, c, c, d, d\}$. Given a string A, we denote by $A[i]$ the symbol of A in i-th position, and by $A[i \dots j]$ the substring of A that starts at position i and ends in position j. Moreover, we denote the size of A

by $|A|$. Note that since we mostly work with multi-sets, operations $\cup$, $\cap$ and $\setminus$ are implicitely understood to be multi-set operations.

Given a string A, an *adjacency* of A is an unordered pair of consecutive elements of A, that is $A[i]A[i+1]$ or $A[i+1]A[i]$, with $1 \leq i \leq |A|-1$. We say that a position i, $1 \leq i \leq |A|$, *induces* an adjacency ab, if ($A[i] = a$ and $A[i+1] = b$) or ($A[i] = b$ and $A[i+1] = a$). We denote by $[\![A]\!]$ the multi-set of adjacencies of A. Following the previous example, where $A = abcdabcdaa$, we have that the multi-set of adjacencies of A is $[\![A]\!] = \{aa, ab, ab, ad, ad, bc, bc, cd, cd\}$).

The endpoints of A, are its first and last position, that is $A[1]$ and $A[|A|]$. In order to deal with endpoints of the two strings, for all the strings we consider, we assume that the first and the last positions contain a dummy symbol $\sharp$, that is not contained in any other position. Formally, for a string A, with $|A| = n$, it holds $A[1] = A[n] = \sharp$ and $A[i] \neq \sharp$ when $2 \leq i \leq n-1$. Notice that the dummy symbol is not considered when the set of adjacencies $[\![A]\!]$ is defined, i.e. $\sharp A[2]$ and $A[n-1]\sharp$ are not in $[\![A]\!]$.

When comparing two input 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 on an alphabet Σ, a *scaffold* is a string on Σ with some missing elements with respect to another string. For the One-sided scaffold filling problem, the multi-set Y is empty.

The two scaffold filling problems we consider in this paper are both based on the definition of *common adjacency* between two genomes (strings) (refer to Fig. 1 for an example).

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 induce the same adjacency;*
- *each position that induces a common adjacency belongs to some pair of M.*

We say that a position of A or B is *matched*, if it belongs to a pair of M. Informally, M relates the positions inducing common adjacencies of the two strings A and B. Notice that, unlike in permutations, where every position of a permutation inducing a common adjacency matches exactly one position in the other permutation, in strings a position of one string inducing a common adjacency may correspond to many positions of the second string (in Fig. 1, notice that position 1 of A, inducing adjacency ab, can match position 4 or position 5 of B').

Given a scaffold B and a multi-set X of symbols, a string B' is called a *filling* of B with X if

1. $[B'] = [B] \cup X$
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.

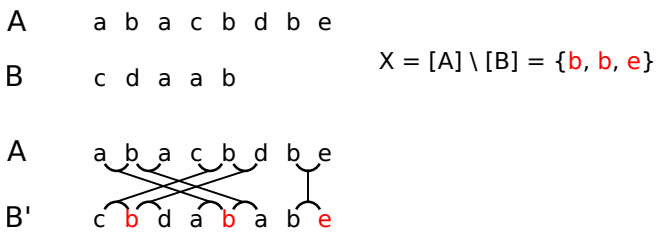

Fig. 1: An instance for the One-sided SF-MNSA problem. Given the complete genome A and the scaffold B, we compute the filled genome B' by inserting the symbols of X in B. The lines between A and B' connect positions inducing common adjacencies and represent a matching M between the adjacencies of A and B'. The number of common adjacencies between A and B' is 5.

Now, we are ready to present the formal definitions of the two Scaffold Filling problems investigated in this paper. Notice that, since we are interested in the parameterized complexity of the two problems, we give the definitions of the parameterized versions of the two problems.

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 One-sided SF-MNSA problem can be seen a restriction of Two-sided SF-MNSA with $Y = \emptyset$.

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 [17].

Lemma 1 *[17] Let A and B two strings, $X = [A] \setminus [B]$, 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, since One-sided SF-MNSA problem can be seen as a restriction of Two-sided SF-MNSA, 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, and consider the multiset AD of common adjacencies between A and B. We can assume that $|\mathsf{AD}| < k$, otherwise we already know that One-sided SF-MNSA/Two-sided SF-MNSA admits a solution consisting of at least k common adjacencies. Now, since $|\mathsf{AD}| < k$, we can compute a partition of AD into two subsets as follows:

- the multiset $\mathsf{AD}_{pr} \subseteq \mathsf{AD}$ of common adjacencies that are preserved after the filling of B and/or A;
- the multiset $\mathsf{AD}_{br} \subseteq \mathsf{AD}$ 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 on alphabet Σ and let AD be the multiset of common adjacencies between A and B. Consider a solution for One-sided SF-MNSA/Two-sided SF-MNSA that partitions AD into multisets AD_{pr}, AD_{br}, then we can compute the partition of AD into the two multisets AD_{pr} and AD_{br} in time $O(2^k)$.*

Proof. Recall that $|\mathsf{AD}| < k$, since otherwise we already know that the One-sided/Two-sided SF-MNSA problem admits a solution consisting of at least k common adjacencies. Then, it easy to see that there can be at most 2^k subsets AD_{pr} of AD, hence at most 2^k subsets AD_{br} of AD. □

This property is implicitly used in the two fixed-parameter algorithms, presented in the next sections, in order to guess which adjacencies of the set AD will be preserved. The latter adjacencies are induced by positions where no insertion is possible when a filling of an input string is computed. Hence, we assume in the following that when a string is inserted into A or B, it is not inserted in a position that induces an adjacency in AD_{pr}.

Color-Coding. The FPT algorithms we present are based on a well-known technique to design FPT algorithms, called color-coding [1]. Originally introduced to identify subgraphs such as simple paths inside a larger graph [1], color-coding has been applied to many graph problems, for example for the graph motif problem and variant thereof [12,9,2,10]. We apply this technique in a different context, that is for string comparison, following some recent examples [5,6].

Informally, given a set U of size n, the color-coding technique aims to find a solution $S \subseteq U$ of size k by coloring the elements of U with k colors, so that each element of 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 applying combinatorial properties of the problem, it is possible in some cases 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. *[1] 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 the One-sided SF-MNSA problem parameterized by the number k of common adjacencies between the input string A and a filling B' of B with the multi-set X. We recall that $X = [A] \setminus [B]$ and that by Lemma 1, it holds $|X| \leq k$. Furthermore, we assume that we have already computed the two subset AD_{pr} and AD_{br} of AD (the common adjacencies of A, B) and that no insertion is possible during the filling in a position associated with an adjacency of AD_{pr} by the matching M of the common adjacencies of A and B (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} (recall that $\mathsf{AD}_{br} = \mathsf{AD} \setminus \mathsf{AD}_{pr}$) to colors in C_A. Informally, the coloring is used to identify those positions of A that, due to the insertion of symbols in X, match positions of B, hence inducing new adjacencies.

In the following, we consider a function $f \in F$, for the positions of A, and we assume that such a function f is injective with reference to a filling B' of B, that is if there exists a filling B' that induces k common adjacencies, we assume that the positions of A inducing these common adjacencies are colored with k distinct color by f.

Given a string S, S is said *colorful for C_A* if there exist $\{s_c \mid c \in C_A\} \subseteq (\llbracket S \rrbracket \cap \llbracket A \rrbracket)$, 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 B' of B which is colorful for C_A.

The algorithm is based on two levels of dynamic programming recurrences. First, we present a dynamic programming recurrence to compute if there exists a string on X to be inserted at a given position j of A so that the resulting string is colorful for a given set $C_j \subseteq C_A$. Then, this result is used to define the dynamic programming recurrence for the insertion of a set of symbols in the string $B[1, j]$ so that the result is colorful for a set $C' \subseteq C_A$.

Inserting symbols at a specific position. We first focus on inserting a given set of symbols at one given position of B. Given a position j in B, two sets $X_j \subseteq X$ and $C_j \subseteq C_A$, define $\texttt{Ins}_j(X_j, C_j) \in \{0,1\}$ as follows:

$$\texttt{Ins}_j(X_j, C_j) = 1 \Leftrightarrow \text{there exists a filling of } B[j-1, j] \text{ with } X_j \text{ which is colorful for } C_j.$$

Note that $\texttt{Ins}_j(\emptyset, \emptyset) = 1$ for all j. In the following lemma, we show that $\texttt{Ins}$ can be computed in time $O(2^{2k}k^2)$ by dynamic programming.

Lemma 2 *We can compute the values of* $\texttt{Ins}_j(X_j, C_j)$ *— where* $X_j \subseteq X$, $C_j \subseteq C_A$ *with* $|X_j|, |C_j| \leq k$, *and* j *is an integer with* $1 < j \leq |B|$ *— in overall time* $O(2^{2k}k^2|B|)$.

We compute the table $\texttt{Ins}_j(X_j, C_j)$ by dynamic programming. More precisely, for any fixed j, we compute a table $\texttt{Add}_\alpha(X', C') \in \{0,1\}$ defined over all subsets $X' \subseteq X$, $C' \subseteq C$ and symbols $\alpha \in X'$ as follows:

$$\texttt{Add}_\alpha(X', C') = 1 \Leftrightarrow \text{there exists a string } s' \text{ with symbols } [s'] = X' \text{ so that the concatenation } s = B[j-1]s' \text{ is colorful for } C', \text{ and the rightmost symbol of } s \text{ is } \alpha$$

Intuitively, $\texttt{Add}$ gives, for any set of colors C', the strings formed over X which are colorful for C' if inserted after $B[j-1]$. In fact, rather than computing the precise ordering of each such string, the table only focuses on the set of symbols (X') and the last symbol (α) for each one. Moreover, for any given j, the table $\texttt{Add}$ contains enough information to deduce the values of $\texttt{Ins}_j(X_j, C_j)$ for all pairs (X_j, C_j).

We prove that table $\texttt{Add}$ can be computed using the following dynamic programming recurrence.

Recurrence 1 *Let* $X' \subseteq X, C' \subseteq C_A$, $\alpha \in \Sigma$.

- *if* $X' = \emptyset$, *then* $\texttt{Add}_\alpha(\emptyset, C') = 1$ *iff* $C' = \emptyset$ *and* $\alpha = B[j-1]$
- *if* $|X'| > 0$ *and* $\alpha \notin X'$, *then* $\texttt{Add}_\alpha(X', C') = 0$.
- *if* $|X'| > 0$ *and* $\alpha \in X'$, *then:*

$$\texttt{Add}_\alpha(X', C') = \max_{\beta \in X' \setminus \{\alpha\}} \begin{cases} \texttt{Add}_\beta(X' \setminus \{\alpha\}, C') \\ \vee\, \exists c \in C', \quad \texttt{Add}_\beta(X' \setminus \{\alpha\}, C' \setminus \{c\}) \text{ and there exists a position of } A \text{ colored by } c \text{ that induces an adjacency } \alpha\beta \end{cases}$$

Proof. Base case. Easily, any string s' with $[s'] = \emptyset$ yields $s = B[j-1]s' = B[j-1]$. Hence s is only colorful for the empty set of colors, and its last symbol is $\alpha = B[j-1]$.

Recurrence. The case where $\alpha \notin X'$ is equally trivial (any s' with $[s'] = X'$ would be non-empty and have a last letter in X', and so would $s = B[j-1]s'$). We now consider the case where $\alpha \in X'$. Assume that, for some $\beta \in X' \setminus \{\alpha\}$, we have $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C') = 1$ or $\exists c' \in C'$, $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C' \setminus \{c'\}) = 1$. In the former case there exists a string s' such that $B[j-1]s'$ is colorful for C', hence also $B[j-1]s'\alpha$ is colorful for C'. In the latter case there exists a string s' such that $B[j-1]s'$ is colorful for $C' \setminus \{c'\}$, where β is the last symbol of s'. Moreover, there exists a position of A colored by c' that induces an adjacency $\beta\alpha$, hence string $s = B[j-1]s'\alpha$ is colorful for C'. Note that in both cases, $[s'] = X' \setminus \{\alpha\}$ and $[s'\alpha] = X'$.

Reciprocally, assume that $\texttt{Add}_\alpha(X', C') = 1$, i.e. there exist strings s' and s ending with α such that $[s'] = X'$ and $s = B[j-1]s'$ is colorful for C'. Let $j = |s|$, and consider the substring $s^* = s[1, j-1]$. Then s^* is colorful for some $C^* \subseteq C'$ and write β for the last symbol of s^*. By definition, $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C^*) = 1$. Now, we have two possible cases. If $\beta\alpha$ does not induce a common adjacency (that is s^* is colorful for C'), then $\texttt{Add}_\beta(X' \setminus \{\alpha\}, C^*) = 1$. If $\beta\alpha$ induces a common adjacency (that is s^* is colorful for $C^* = C' \setminus \{c'\}$, for some $c' \in C'$), then $\texttt{Add}_\alpha(X' \setminus \{\alpha\}', C' \setminus \{c'\}) = 1$. In both cases, the recurrence correctly sets $\texttt{Add}_\alpha(X', C')$ to 1.

□

We now have all the tools to prove Lemma 2.

Proof (of Lemma 2). For any j, $1 < j \leq |B|$, the table $\texttt{Add}$ can be computed using Recurrence 1. It is easy to deduce the values of $\texttt{Ins}_j$:

$$\texttt{Ins}_j(X_j, C_j) = \max_{\alpha \in X_j} \begin{cases} \texttt{Add}_\alpha(X_j, C_j) \\ \vee\, \exists c \in C, \quad \texttt{Add}_\alpha(X_j, C_j \setminus \{c\}) \text{ and there exists} \\ \qquad\qquad \text{a position of } A \text{ colored by } c \text{ that induces} \\ \qquad\qquad \text{an adjacency } \alpha B[j] \end{cases}$$

Indeed, if $\texttt{Ins}_j(X_j, C_j) = 1$, then it follows that there is a substring s over alphabet X_j, such that $B[j-1]sB[j]$ is colorful for C_j. Write α for the last symbol of $B[j-1]s$. Then, either the substring $B[j-1]s$ is colorful for C_j, hence $\texttt{Add}_\alpha(X_j, C_j) = 1$, or $B[j-1]s$ is colorful for $C_j \setminus \{c'\}$, for some color $c' \in C_j$, hence $\texttt{Add}_\alpha(X_j, C_j \setminus \{c'\}) = 1$, and there exists a position of A colored by c' that induces an adjacency $\alpha B[j]$. The reciprocal is clear with a similar decomposition.

Now, we discuss the time complexity of computing $\texttt{Add}_\alpha(X', C')$. Table $\texttt{Add}_\alpha(X', C')$ consists of $O(2^{2k}k)$ entries, since $|X'|, |C'| \leq k$, hence we have 2^k possible subsets of each $X' \subseteq X$, $C' \subseteq C_A$. Each entry is computed in time $O(k^2)$, since the algorithm looks for at most k^2 entries in the table, depending on the chosen $\beta \in X'$ and $c \in C'$, and $|X'|, |C'| \leq k$. Given the table $\texttt{Add}_\alpha(X', C')$, the time complexity to compute each entry $\texttt{Ins}_j(X_j, C_j)$ is $O(k^2)$, since again in the worst case we have to look for k^2 entries, depending on the chosen $\alpha \in X_j$ and $c \in C_j$. This process has to be repeated $O(n)$ times, varying $j \leq |B|$. Hence the overall time complexity to compute the whole table $\texttt{Ins}_j(X_j, C_j)$ is $O(2^{2k}k^2n)$.

□

Inserting symbols in a prefix of B. Next we consider the general problem of inserting a multiset of symbol in different positions of a prefix of B.

Given a multiset $X' \subseteq X$ of symbols and a set $C'_A \subseteq C_A$ of colors, define $\texttt{Fill}_j(X', C'_A) \in \{0, 1\}$ as follows:

$$\texttt{Fill}_j(X', C'_A) = 1 \Leftrightarrow \text{there exists a filling of } B[1, \ldots, j] \text{ with } X' \text{ which is colorful for } C'_A.$$

We now prove that $\texttt{Fill}_j(X', C'_A)$ satisfies the following recurrence property. The objective, as stated in Lemma 3, is to determine whether a prefix of B can be filled with a given multiset of symbols X' contained in X, so that the resulting filling is colorful for a subset of C_A.

Recurrence 2 *Let $X' \subseteq X$, $C'_A \subseteq C_A$.*

- *For $j = 2$, $\texttt{Fill}_2(X', C'_A) = \texttt{Ins}_2(X', C'_A)$.*
- *For all $j > 2$,*

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

Proof. Base case. The case $j = 2$ is easily deduced by definition: $\texttt{Fill}_j(X', C'_A) = 1$ if and only if $\texttt{Ins}_j(X', C'_A) = 1$.

Consider $j > 2$, and assume that $\texttt{Fill}_j(X', C'_A) = 1$, that is there exists a filling B' of $B[1, \ldots, j]$ with X' colorful for C'_A. Consider the set of symbols $X_j \subseteq X$ inserted in position j of B and let C_j be a set of colors such that there exists a set of positions of A colored by C_j associated by the matching with adjacencies using elements of X_j. By definition, $\texttt{Ins}_j(X_j, C_j) = 1$ and $\texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) = 1$, hence the recurrence formula correctly sets $\texttt{Fill}_j(X, C_A)$ to 1.

Assume now that $\texttt{Ins}_j(X_j, C_j) = 1$ and $\texttt{Fill}_{j-1}(X' \setminus X_j, C'_A \setminus C_j) = 1$ for some $X_j \subseteq X'$, $C_j \subseteq C'_A$, that is the recurrence formula sets $\texttt{Fill}_j(X', C'_A)$ to 1. Then by definition it follows that there exists a filling of $B[1, \ldots, j-1]$ with $X' \setminus X_j$ which is colorful for $C'_A \setminus C_j$ and a filling of $B[j-1, j]$ with X_j which is colorful for C_j. Hence concatenating the two fillings (the filling of $B[1, \ldots, j-1]$ and the filling between $B[j-1, j]$), we obtain a filling B' of $B[1, \ldots, j]$ with X' which is colorful for C'_A, thus $\texttt{Fill}_j(X', C'_A) = 1$. □

In the following, we prove the correctness of Recurrence 2, that is 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 $\texttt{Fill}_{|B|}(X, C_A) = 1$.

Proof. (i)⇒(ii) Let B' be a filling of B with X such that A and B' have k common adjacencies; write $I' = [\![A]\!] \cap [\![B']\!]$. Since $|I'| = k$, there exists $f \in F$ such that f is injective on the positions of A that induce I'. For each color $c \in C_A$, there exists an adjacency $f(c)$ in B' such that $f(c)$ is induced by a position of A colored by c, hence B' is colorful for C_A. By definition of `Fill`, we thus have $\texttt{Fill}_{|B|}(X, C_A) = 1$.

(ii)⇒(i) Assume that for some $f \in F$, $\texttt{Fill}_{|B|}(X, C_A) = 1$, then there exists a filling of B with X colorful for C_A. For each $c \in C_A$, let x_c be the unique position of A colored by c inducing a common adjacencies with B'. Then $\{x_c \mid c \in C_A\}$ is a set of positions of size k yielding k common adjacencies between A and B'. □

Next, we show how the recurrence described in Recurrence 2 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 $2^{O(k)}poly(|A| + |B|)$.*

Proof. By Lemma 3, it suffices to compute a family F of hash functions from the positions of A to C_A. Then the problem admits a solution if and only if there exists a coloring $f \in F$ for which $\texttt{Fill}_{|B|}(X, C_A) = 1$. Recurrence 2 together with Lemma 2 yields a dynamic programming algorithm to compute $\texttt{Fill}_{|B|}(X, C_A)$ for each coloring.

Now, we consider the time complexity of the algorithm. Write $n = |A| + |B|$. First, a perfect family F 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, the algorithm iterates over the $2^{O(k)}\log(n)$ possible functions $f \in F$ and the respective color codings. For each function $f \in F$, the table $\texttt{Ins}_j(X', C'_A)$ is computed in time $O(2^{2k}k^2|B|)$ (see Lemma 2). Then the $O(2^{2k}|B|)$ entries of table $\texttt{Fill}_j(X', C'_A)$ are computed (Recurrence 2), where each entry requires $O(2^{2k})$ look-ups, depending on the choice of X_j and C_j. Thus the algorithm requires $O(2^{4k}n)$ time 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 present a fixed-parameter tractable algorithm for it. As for the One-sided case, the algorithm is based on color-coding and dynamic programming. However, the same approach described in the previous section cannot be applied directly and new challenges make Two-sided SF-MNSA more complicated than One-sided SF-MNSA. First, there exist a new kind of common adjacencies, namely adjacencies

that are created in the fillings although they never appear as such in any of the input strings. Also, unlike the One-sided case, it is not known a priori whether a given adjacency of the string A 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 arrangements of such adjacencies, and with the second by introducing "insertion" colors, so that the positions associated with such colors can only be used to insert a substring and not (directly) 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. 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, 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 and 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 on alphabet X (on alphabet Y respectively) into B (into A respectively). Now, the algorithm consists of four steps.

Step 1. In the first step, the algorithm "guesses" (that is iterates over all possible cases) how the letters from X and Y should be arranged into strings to be inserted into A and B. Such strings are called *filler strings*. Note that we do not guess the insertion point of those strings, and since $|X|, |Y| \leq k$, the number of cases to try depends only on a function of k (see Prop. 2).

Step 2. The second phase identifies 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 inducing a common adjacency of A' and B' 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 one element of Y. Therefore these adjacencies consist of an endpoint of an inserted string as well as a symbol already present in the original strings A and B. The second step of the algorithm identifies and matches the endpoints of inserted strings (computed in Step 1) which correspond to such (X, Y)-adjacencies (see Def. 4 and Prop. 3).

Step 3. In the third step, the algorithm opportunely color-codes the positions of A and B with two disjoint sets of colors, in order to:

1. match non (X, Y)-adjacencies (like in the previous algorithm)
2. 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)

Step 4. Step 4 finally inserts the strings into A and B by dynamic programming, while creating the remaining adjacencies (see Recurrence 3).

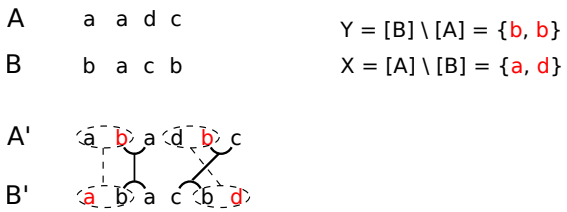

Fig. 2: An instance of the Two-sided SF-MNSA problem. Given two scaffolds A and B, we obtain the filled genomes A' and B' by inserting symbols X in B and Y in A (inserted symbols are in red). Straight Lines connect common adjacencies, dotted lines connect (X, Y)-adjacencies. Notice for example that the bd common adjacency is an (X, Y)-adjacency induced by the insertion of b into string A and of d into string B.

Now we are able to present the details of the different steps of the algorithm.

Step 1: *Compute filler strings.*

Consider a solution of Two-sided SF-MNSA (A', B') inducing k common adjacencies. We call *filler string* a non-empty string consisting of elements of X or of Y inserted into B or A to create B' or A'. We write S_X and S_Y for the two multi-sets of filler strings over the multi-sets X and Y that are inserted into B and A respectively. 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) will eventually be considered. The following property bounds both the number of possible pairs (S_X, S_Y) and the number of positions where filler strings can be inserted into A and B.

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

Proof. (1) The property follows easily from the fact that, since $|X|, |Y| \leq k$, $|S_X|, |S_Y| \leq k$.

(2) By Lemma 1, $|X|, |Y| \leq k$. Now, consider w.l.o.g. a multi-set S_X of filler strings inserted into B. This multi-set obviously consists of at most k strings, where each one has length bounded by $|X| \leq k$. Hence, the number of possible multi-sets of filler strings to be inserted into B is bounded by $O(k^k)$. We can conclude that the overall possible multi-sets of filler strings over X, Y inserted into B and A respectively, in order to obtain two fillings B', A', is bounded by $O(k^{2k})$. □

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

We first define formally the concept of (X,Y)-*adjacency* (see Fig. 2 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, such that*

- *one of $A'[i]$ or $A'[i+1]$ is the endpoint of a filler string $s_y \in S_Y$,*
- *and one of $B'[j]$ or $B'[j+1]$ is the endpoint of a filler string $s_x \in S_X$*

Notice that, since $X \cap Y = \emptyset$, it follows that any new common adjacency of A' (of B' respectively) is either involved in only one insertion (hence, in one input string, it is induced by a position where no insertion occurs), or it is an (X,Y)-adjacency.

Now, the algorithm considers the endpoints of the filler strings computed in the previous step, and defines which endpoints 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). Note that we consider that each string yields two endpoints, even length-one filler strings. In order to compute which endpoints of E_X and E_Y induce a common (X,Y)-adjacency, we use 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. The procedure assumes that k' is the number of induced (X,Y)-adjacencies.

Definition 4. *A* number assignment *for the strings in $S_X \cup S_Y$ is 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 of Two-sided SF-MNSA, a corresponding number assignment is obtained as follows (recall that k' denotes 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 if and only if it is not involved in an (X,Y)-adjacency;
- endpoint e_z is associated with a number $i \in \{1,\ldots,k'\}$ if and only if 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.

Next, we show how to compute a number assignment in time $O((4k)^{k+1})$. The following property gives an easy upper bound on the number of such assignments.

Property 3 *There exist at most $(4k)^{k+1}$ number assignments.*

Proof. Notice that $|E_X \cup E_Y| \leq 4k$, since $|S_X|, |S_Y| \leq k$. Moreover, each endpoint is assigned a number in $\{0,1,\ldots,k'\}$, with $k' \leq k$, hence there exist at most $(4k)^{k+1}$ number assignments. □

The algorithm iterates over each possible number assignment, hence, in what follows we assume that the algorithm guesses the correct number assignment to $E_X \cup E_Y$.

Once we have defined through the number assignment which endpoints in $E'_X \cup E'_Y$ induce a common adjacency, we have to bound the possible symbols adjacent to an endpoint in $E'_X \cup E'_Y$. Indeed, when we will insert into A a filler string s_y whose endpoint induces an (X, Y)-adjacency, we will not be able to define a matching between this adjacency and an adjacency of B, because we still have to insert the "companion" strings in B (that is the filler string s_x that induces an (X, Y)-adjacency with s_y). However, by restricting the possible symbols that must be made adjacent to the filler strings, we will be able to ensure that a common adjacency is eventually induced.

Now, we show how we can restrict 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 solution (A', B') to the scaffold-filling problem and a filler 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) yielding an (X, Y)-adjacency. Then we define $v(e_x)$ ($v(e_y)$ respectively) as the symbol of Y (of X respectively) adjacent to e_x in B' (to e_y in A' respectively).*

The symbols $v(e_x)$, $v(e_y)$, for each $e_x \in E'_X$, $e_y \in E'_Y$ are immediately deduced from the number assignment. Indeed, if $e_x \in E'_X$ and $e_y \in E'_Y$ are associated with the same number i (that is they induce an (X, Y)-adjacency), then $v(e_x)$ must be the symbol of the filler strings at endpoint e_y, while $v(e_y)$ must be the symbol at endpoint e_x.

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

Proof. Consider the case that $e_x \in E_X$ and $e_y \in E_Y$ are associated by the number assignment with the same number $i \in \{1, \ldots, k'\}$. Since $X \cap Y = \emptyset$, it follows that e_x must be inserted in a position of B containing the same symbol as e_y and e_y must be inserted in a position of A containing the same symbol as e_x. □

Using the number assignment and the values $v(e_z)$, with $e_z \in E'_X \cup E'_Y$, the algorithm creates the following table which tells whether, according to (X, Y)-adjacencies, a filler string can be inserted at a certain position. We define the table for filler strings of S_X, the definition for S_Y being similar. Let $j \in \{1, \ldots, |B|\}$, s be a filler string in S_X, and s_l, s_r for the left and right endpoints of s respectively:

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

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

Our next goal is to distinguish, for each input string, adjacencies that need to be broken to insert a filler string, from adjacencies that will yield a common adjacency when the other filling is created. Since there are at most k adjacencies of either kind, we use color-coding to achieve this goal. Consider a coloring $f \in F$ of the positions of A and B with a set C of z, $k \leq z \leq 2k$, colors. 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.

Since the two multisets of strings S_X, S_Y are fixed, and the number assignment of the k' (X,Y)-adjacencies is fixed, we only consider partitions where $|C_{I,A}| = |S_Y|$, $|C_{I,B}| = |S_X|$, and $|C_{M,A}| + |C_{M,B}| + k' = k$.

Step 4: *Insert strings by dynamic programming.*

Now we have all the tools to decide where each string must be inserted. Thanks to Steps 1–3, we can now deal with both sides independently, hence without loss of generality we describe the algorithm inserting filler strings of S_X in B. The constraints one needs to observe are the following.

- Filler strings are inserted at positions colored by $C_{I,B}$. Note that we do not require to insert a filler string in *every* colors of $C_{I,B}$: we only need to ensure that no adjacency having a color in $C_{M,B}$ is broken.
- (X,Y)-adjacencies are created as guessed during Step 3, that is each filler string s inserted at position j yields $\texttt{XY-Fits}_{B,j}(s) = 1$
- The remaining created adjacencies yield enough common adjacencies with A. More precisely, for each color $c \in C_{M,A}$, we create at least one adjacency which can be matched to an adjacency with color c in A.

The first two constraints can clearly be checked in constant time for any filler string s at any position j. The third constraint is dealt with as follows.

Let $s \in S_X$ be a filler string, and j be a position of B. Let H be the substring of $B[j-1]sB[j]$ from which $B[j-1]$ is removed if $s_l \in E'_B$ and from which $B[j]$ is removed if $s_r \in E'_B$. That is, if the string s is inserted at position j in B, then H covers the positions which may create a common adjacency which is not an (X,Y)-adjacency. In order to determine whether H induces enough new common adjacencies for a given set of colors, we define $\texttt{Col-Fits}_{B,j}(s, C_j) \in \{0,1\}$ for all $C_j \subseteq C_{M,A}$:

$$\texttt{Col-Fits}_{B,j}(s, C_j) = 1 \Leftrightarrow H \text{ is colorful for } C_j$$

Combining with the first two constraints, and similarly to the One-sided case, we define $\texttt{Ins}_{B,j}(s, C_j) \in \{0,1\}$, where $C_j \subseteq C_{M,A}$, as follows:

$$\begin{aligned}\texttt{Ins}_{B,j}(s, C_j) = 1 \Leftrightarrow \quad & B[j-1] \text{ is assigned a color in } C_{I,B} \\ & \wedge\, \texttt{XY-Fits}_{B,j}(s) = 1 \\ & \wedge\, \texttt{Col-Fits}_{B,j}(s, C_j) = 1\end{aligned}$$

We extend the definition to deal with the empty string ε:

$$\texttt{Ins}_{B,j}(\varepsilon, C_j) = 1 \Leftrightarrow\ C_j = \emptyset$$

Here table $\texttt{Ins}_{B,j}(s, C_j)$ is easier to compute than in the one-sided case, because the string to be insterted is known (and not only the set of its elements). Each entry can be computed in time $O(k^3)$ using a matching algorithm.

Lemma 4 *Let j be an integer s.t. $j \leq |B|$ and $C_j \subseteq C_{M,B}$. Then we can compute $\texttt{Ins}_{B,j}(s, C_j)$ in time $O(k^3)$.*

Proof. Recall that the color of $B[j-1]$ is known from Step 3 and that $\texttt{XY-Fits}_{B,j}(s)$ is directly deduced from the number assignment computed at Step 2. Hence computing $\texttt{Ins}_{B,j}(s, C_j)$ only requires to compute the value of $\texttt{Col-Fits}_{B,j}(s, C_j)$, which in turn goes down to deciding whether a given string H is colorful for a given set of colors C_j. This can be achieved as follows.

Let $\mathbb{H} = [\![H]\!]$ be the multiset of adjacencies of H (that is, the ones created by inserting string s which need to cover all colors in C_j). Create a bipartite graph G with vertex set $\mathbb{H} \cup C_j$. Add an edge (h, c) for each $h \in \mathbb{H}$ and $c \in C_j$ such that there exists a position with color c inducing an adjacency h. Then string H is colorful for C if and only if graph G admits a matching covering all vertices of C_j. The existence of such a matching can be determined in time $O(\sqrt{|\mathbb{H}| + |C_j|}|\mathbb{H}||C_j|) = O(k^3)$ [15]. We can then deduce the values of $\texttt{Col-Fits}_{B,j}(s, C_j)$ and $\texttt{Ins}_{B,j}(s, C_j)$. □

We can now compute a filling of B satisfying all the above constraints. We do this by dynamic programming, i.e. filling progessively B while keeping track of inserted substrings and covered colors.

Definition 6. *Let $S'_X \subseteq S_X$, $C'_{M,A} \subseteq C_{M,A}$, and $1 \leq j \leq |B|$. We define $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) \in \{0,1\}$ as follows. $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ if and only if there exists a filling B' of $B[1, \ldots, j]$ such that:*

1. *B' is obtained by inserting all strings in S'_X in different positions of $B[1, \ldots, j]$,*
2. *the inserted strings are inserted at positions colored by $C_{I,B}$,*
3. *for any inserted string $s \in S_X$ at position j, $\texttt{XY-Fits}_{B,j}(s) = 1$,*
4. *the filling is colorful for $C'_{M,A}$ after removing (X, Y)-adjacencies.*

We observe that the entries of `Fill-B` can be computed using the following dynamic programming recurrence.

Recurrence 3 *Let $S'_X \subseteq S_X$, $C'_A \subseteq C_A$.*

- *For $j = 1$, $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ iff $S'_X = \emptyset$ and $C'_{M,A} = \emptyset$.*
- *For all $j \geq 2$,*

$$\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = \max_{\substack{s_j \in S_X \cup \{\varepsilon\} \\ C_j \subseteq C'_{M,A}}} \begin{cases} \texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) \\ \wedge\, \texttt{Ins}_{B,j}\left(s_j, C_j\right) \end{cases}$$

Proof. In case $j = 1$, note that no substring can be inserted in the size-1 string $B[1 \ldots 1]$, hence $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ implies that $S'_X = \emptyset$. Note that in this case ($S'_X = \emptyset$), no common adjacencies can be created, hence $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ if and only if $C'_{M,A} = \emptyset$.

We now prove the recurrence formula.

Assume that $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$. Let B' be a filling of $B[1, \ldots, j]$ using S'_X and colorful for $C'_{M,A}$. If there is no string inserted at position $j-1$ (i.e. in $B[j-1, j]$), then we directly have $\texttt{Fill-B}_{j-1}\left(S'_X, C'_{M,A}\right) = 1$. Using $s_j = \varepsilon$, and since $\texttt{Ins}_{\emptyset}\left(B, \varepsilon\right) j = 1$, the formula is correct in this case. Otherwise, assume that a string $s_j \in S_X$ is included between $B[j-1]$ and $B[j]$, creating a string $B[j-1]s_jB[j]$ that is colorful for some $C_j \subseteq C'_{M,A}$ (after removing (X,Y)-adjacencies). Then, for this particular string s_j and this subset C_j, we have both $\texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) = 1$ and $\texttt{Ins}_{B,j}\left(s_j, C_j\right) = 1$. Thus, again, the recurrence formula correctly sets $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right)$ to 1.

Conversely, assume that

$$\max_{\substack{s_j \in S_X \cup \{\varepsilon\} \\ C_j \subseteq C'_{M,A}}} \begin{cases} \texttt{Fill-B}_{j-1}\left(S'_X \setminus \{s_j\}, C'_{M,A} \setminus C_j\right) \\ \wedge\, \texttt{Ins}_{C_j}\left(B, s_j\right) j \end{cases} = 1$$

Consider first the case where the maximum is obtained for $s_j = \varepsilon$ and some $C_j \subseteq C'_{M,A}$. Then $\texttt{Ins}_{C_j}\left(B, \varepsilon\right) j = 1$ yields $C_j = \emptyset$. Hence $\texttt{Fill-B}_{j-1}\left(S'_X, C'_{M,A}\right) = 1$, and there exists a filling of $B[1, \ldots, j-1]$ which is trivially extended to a filling B' of $B[1, \ldots, j]$ by adding element $B[j]$. Filling B' directly satisfies Definition 6. Otherwise, the max is obtained for some $s_j \in S'_X$ and some $C_j \subseteq C'_{M,A}$. We thus obtain a filling of $B[1, \ldots, j-1]$, and augment this filling by adding string $s_jB[j]$. It is easy to check that this new filling satisfies Definition 6. In both cases, we have $\texttt{Fill-B}_j\left(S'_X, C'_{M,A}\right) = 1$ □

The following Lemma sums up all the results from Steps 1–4. Note that since Step 4 must be run for both A and B independently, we define and compute a table `Fill-A` similarly to table `Fill-B`.

Lemma 5 *Let (A, B) be an instance of Two-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 $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B}) = \texttt{Fill-B}_{|B|}(S_X, C_{M,A}) = 1$.

Proof. (i)⇒(ii) Let A' be a filling of A with Y and B' be a filling of B with X such that A' and B' have k common adjacencies;

Define S_X, S_Y, a number assignment, a color-coding $f \in F$ and a partition $C = C_{M,A} \cup C_{M,B} \cup C_{I,A} \cup C_{I,B}$ according to these two fillings, so that Definition 6 is satisfied. Then it follows that $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B}) = \texttt{Fill-B}_{|B|}(S_X, C_{M,A}) = 1$.

(ii)⇒(i) Reciprocally, assume that $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B}) = 1$ and $\texttt{Fill-B}_{|B|}(S_X, C_{M,A}) = 1$. Then it follows that S_X, $C_{M,A}$, $C_{I,B}$ satisfy Definition 6 for $B[1, |B|]$. The same property holds for S_Y, $C_{M,B}$, $C_{I,A}$ satisfying Definition 6 for $A[1, |A|]$. This leads to a filling of A and B with k adjacencies as follows: k' (X, Y)-adjacencies obtained from the endpoints of the corresponding inserted strings (remember that $\texttt{XY-Fits}_{B,j}(s) = 1$ for any s inserted at a position j in string B), and at least $|C_{M,A}|$ (resp $|C_{M,B}|$) other common adjacencies between the filling of B (resp. the filling of A) and A (resp. of B). Note that these last common adjacencies appear also in the filling of A (resp., of B) since no substring may break them (all substring are inserted in positions colored by $C_{I,B}$ and $C_{I,A}$). Thus there are, overall, $k' + |C_{M,A}| + |C_{M,B}| = k$ common adjacencies between the fillings of A and B. □

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 SF-MNSA 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 $\texttt{Fill-A}_{|A|}(S_Y, C_{M,B})$ and $\texttt{Fill-B}_{|B|}(S_X, C_{M,A})$, 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 $O(4k^{k+1})$ number assignments to iterate through. The dynamic programming recurrence requires time $O(2^{4k} n)$.

Consider now, the color-coding. There exists 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.

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 investigated two variants of the Scaffold Filling problem (One-sided SF-MNSA and the Two-sided SF-MNSA), a problem related to the reconstruction of complete genomes from incomplete draft genomes. We presented two FPT algorithms for the two variants of Scaffold Filling, where the parameter is the number of common adjacencies in the resulting genomes.

There are some interesting open problems from an algorithmic perspective. First, it would be interesting to improve upon the time (and space) complexity of the two algorithms. In this direction, it would be interesting to investigate whether the algebraic technique applied to Graph Motif [3,14] and Repetition Free Longest Common Subsequence [4] can be useful in this context. Then, it would be interesting to study the kernelization complexity of the two problems. 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 (see [19]) 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. Björklund, A., Kaski, P., Kowalik, L.: Probably optimal graph motifs. In: Portier, N., Wilke, T. (eds.) STACS. LIPIcs, vol. 20, pp. 20–31. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2013)
4. Blin, G., Bonizzoni, P., Dondi, R., Sikora, F.: On the parameterized complexity of the repetition free longest common subsequence problem. Inf. Process. Lett. 112(7), 272–276 (2012)
5. Bonizzoni, P., Della Vedova, G., Dondi, R., Pirola, Y.: Variants of constrained longest common subsequence. Inf. Process. Lett. 110(20), 877–881 (2010)
6. Bonizzoni, P., Dondi, R., Mauri, G., Zoppis, I.: Restricted and swap common superstring: a multivariate algorithmic perspective. Algorithmica (to appear)
7. Chain, P., Grafham, D., Fulton, R., Fitzgerald, M., Hostetler, J., Muzny, D., Ali, J., et al.: Genomics. genome project standards in a new era of sequencing. Science 326, 236–237 (2009)
8. Chen, Z., Fu, B., Goebel, R., Lin, G., Tong, W., Xu, J., Yang, B., Zhao, Z., Zhu, B.: On the approximability of the exemplar adjacency number problem for genomes with gene repetitions. Theoretical Computer Science 550, 59–65 (2014)

9. Dondi, R., Fertin, G., Vialette, S.: Complexity issues in vertex-colored graph pattern matching. J. Discrete Algorithms 9(1), 82–99 (2011)
10. Dondi, R., Fertin, G., Vialette, S.: Finding approximate and constrained motifs in graphs. Theor. Comput. Sci. 483, 10–21 (2013)
11. Downey, R., Fellows, M.: Parameterized Complexity. Springer (1999)
12. 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)
13. Fertin, G., Labarre, A., Rusu, I., Tannier, E., Vialette, S.: Combinatorics of genome rearrangements. The MIT Press, Cambridge (2009)
14. Guillemot, S., Sikora, F.: Finding and counting vertex-colored subtrees. Algorithmica 65(4), 828–844 (2013)
15. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. 2(4), 225–231 (1973)
16. 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)
17. 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)
18. Liu, N., Jiang, H., Zhu, D., Zhu, B.: An improved approximation algorithm for scaffold filling to maximize the common adjacencies. IEEE/ACM Trans. Comput. Biology Bioinform. 10(4), 905–913 (2013)
19. Liu, N., Zhu, D.: The algorithm for the two-sided scaffold filling problem. Algorithmica (to appear)
20. 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)
21. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
22. Xia, X.: Comparative Genomics. SpringerBriefs in Genetics (2013)
23. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics 21, 3340–3346 (2005)