Beyond Adjacency Maximization: Scaffold Filling for New String Distances

Laurent Bulteau, Guillaume Fertin, Christian J Komusiewicz

Beyond Adjacency Maximization: Scaffold Filling for New String Distances*

Laurent Bulteau[1], Guillaume Fertin[2], and Christian Komusiewicz[3]

1 **LIGM UMR 8049, Université Paris-Est, Marne-la-Vallée, France**
laurent.bulteau@u-pem.fr
2 **LS2N UMR CNRS 6004, Université de Nantes, Nantes, France**
guillaume.fertin@univ-nantes.fr
3 **Friedrich-Schiller-Universität Jena, Jena, Germany**
christian.komusiewicz@uni-jena.de

Abstract

In Genomic Scaffold Filling, one aims at polishing *in silico* a draft genome, called scaffold. The scaffold is given in the form of an ordered set of gene sequences, called contigs. This is done by confronting the scaffold to an already complete reference genome from a close species. More precisely, given a scaffold $\mathcal{S}$, a reference genome G and a score function $f()$ between two genomes, the aim is to complete $\mathcal{S}$ by adding the missing genes from G so that the obtained complete genome $\mathcal{S}^*$ optimizes $f(\mathcal{S}^*, G)$. In this paper, we extend a model of Jiang et al. [CPM 2016] (i) by allowing the insertions of strings instead of single characters (i.e., some groups of genes may be forced to be inserted together) and (ii) by considering two alternative score functions: the first generalizes the notion of common adjacencies by maximizing the number of common k-mers between $\mathcal{S}^*$ and G (k-Mer Scaffold Filling), the second aims at minimizing the number of breakpoints between $\mathcal{S}^*$ and G (Min-Breakpoint Scaffold Filling). We study these problems from the parameterized complexity point of view, providing fixed-parameter (FPT) algorithms for both problems. In particular, we show that k-Mer Scaffold Filling is FPT wrt. parameter ℓ, the number of additional k-mers realized by the completion of $\mathcal{S}$—this answers an open question of Jiang et al. [CPM 2016]. We also show that Min-Breakpoint Scaffold Filling is FPT wrt. a parameter combining the number of missing genes, the number of gene repetitions and the target distance.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases computational biology, strings, FPT algorithms, kernelization

Digital Object Identifier 10.4230/LIPIcs.CPM.2017.27

1 Introduction

The recent development and continuous improvement of NGS technologies has increased our ability to produce, rapidly and inexpensively, a first draft of any genome. However, the cost of polishing such drafts to obtain a complete genome has not decreased at the same rate, thus many species are left with a genome in its *scaffold* form: a scaffold may be represented as a sequence of *contigs* (each being a contiguous sequence of genes), separated by unknown

* Christian Komusiewicz gratefully acknowledges support from the DFG, project MAGZ (KO 3669/4-1). A visit from Laurent Bulteau and Christian Komusiewicz which initiated this study was funded by PEPS INS2I from CNRS and Internationalisation des Laboratoires – Action 1 from Université de Nantes.

gaps, sometimes with an indication on the length of the gap. It is thus natural to ask for methods that reconstruct the complete original genome starting from its scaffold form. This is usually done with the help of a reference genome G, that is, the complete genome of a close-enough species, in the following way: turn the scaffold $\mathcal{S}$ into a complete genome $\mathcal{S}^*$ by adding, in between the contigs of $\mathcal{S}$, genes that are present in G but not in $\mathcal{S}$, in such a way that some predefined score function between $\mathcal{S}^*$ and G is optimized. The score function is usually defined so as to follow the parsimony principle: when $\mathcal{S}^*$ and G are as close as possible, the score is optimized.

Formally, a genome G is a string built on some alphabet Σ (each character in the alphabet representing a gene or gene family), and a scaffold $\mathcal{S}$ is defined as sequence $(C_1, \ldots, C_m)$ of contigs, where each C_i, $1 \leq i \leq m$, is itself a string over Σ. For a string S of length n, we let $c(S)$ be the multiset of characters it contains, and $a(S) := \{S[i, i+1] \mid i \in [n-1]\}$ be the *multiset of adjacencies* in S. By extension, if $\mathcal{S}$ is a scaffold, $c(\mathcal{S})$ (resp. $a(\mathcal{S})$) denotes the multiset of characters (resp. adjacencies) contained in the contigs of $\mathcal{S}$. For two strings S and T, let $a(S,T) := a(S) \cap a(T)$ denote the *multiset of common adjacencies* in S and T. For a scaffold $\mathcal{S}$ and a multiset $\mathcal{T}$ of strings, we use $\mathcal{S} + \mathcal{T}$ to denote the set of strings that can be obtained from $\mathcal{S}$ by inserting the strings of $\mathcal{T}$ in between the contigs of $\mathcal{S}$. The One-Sided-Scaffold-Filling problem, introduced in [13], was the first serious attempt at modeling scaffolds as a sequence of contigs with repeats.

One-Sided-Scaffold-Filling
Input: A complete genome G and a scaffold $\mathcal{S}=(C_1, \ldots, C_m)$ over alphabet Σ, and a multiset $\mathcal{T} = c(G) - c(\mathcal{S})$ of characters.
Task: Find $\mathcal{S}^* \in \mathcal{S} + \mathcal{T}$ s.t. $|a(\mathcal{S}^*, G)|$ is maximized.

Note that Jiang et al. [13] also considered the variant in which only a subset $\mathcal{T}' \subseteq \mathcal{T}$ of the letters of $\mathcal{T}$ needs to be inserted. In this paper, we study two alternative problems.

***k*-Mer Scaffold Filling.** The first one generalizes both of the problems considered by Jiang et al. [13] in several ways. First, we do not constrain the multiset of letters to insert to be $c(G) - c(\mathcal{S})$. Instead, the set $\mathcal{T}$ could contain letters in higher or lower multiplicity than in $c(G) - c(\mathcal{S})$. This is helpful if it is known, for example, that some genes occur in higher multiplicity in the desired genome than in G. Second, we allow that $\mathcal{T}$ contains strings instead of only letters. This allows to incorporate knowledge about the gene order that is not present in the tuple of scaffold contigs. For example, one may now deal with contigs whose position relative to the other contigs is not known. Third, we allow that the number of strings to insert can be prespecified as an input constraint. More precisely, in our variant the input contains two numbers t_1, t_2 and we search for a solution that inserts at least t_1 and at most t_2 strings from $\mathcal{T}$. This way, one can guarantee for example that the size of the resulting genome lies within some predetermined range. If we want all of $\mathcal{T}$ to be inserted in $\mathcal{S}$, it suffices to set $t_1 := |\mathcal{T}| =: t_2$. The second variant of Jiang et al. [13] in which an arbitrary subset of $\mathcal{T}$ may be inserted is obtained by setting $t_1 := 1$ and $t_2 := |\mathcal{T}|$.

Finally, as similarity measurement we do not restrict ourselves to maximizing the number of common adjacencies. Instead, we maximize, for a predetermined parameter k, the number of common k-mers (the term *k-mer* is usually used for DNA strings; we thus extend its use here in the context of gene sequences): indeed, as illustrated in Figure 1, a higher value of k tends to increase the accuracy of the result.

For a string S of length n and a positive integer k, let $a_k(S) := \{S[i, i+k] \mid i \in [n-k]\}$ denote the *multiset of k-mers* in S. For two strings S and T, $a_k(S,T) := a_k(S) \cap a_k(T)$

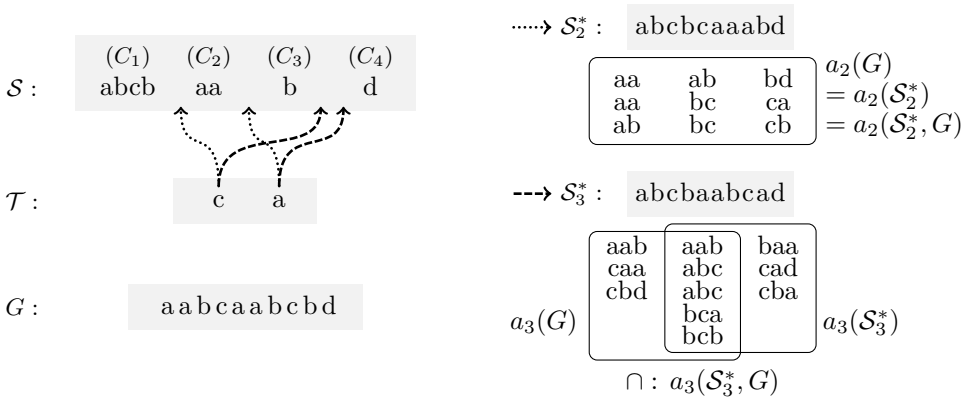

Figure 1 Left: an example instance of k-Mer-SF, for $k = 2$ and $k = 3$, with scaffold $\mathcal{S}$ containing 4 contigs, $\mathcal{T}$ containing 2 length-1 strings to be inserted, $t_1 = t_2 = 2$, and a reference genome G. An optimal solution for $k = 2$ (resp. $k = 3$) inserts the strings from $\mathcal{T}$ as indicated by dotted (resp. dashed) arcs to create $\mathcal{S}_2^*$ (resp. $\mathcal{S}_3^*$). Top-right: the 2-mers of $\mathcal{S}_2^*$ and G: note that the maximum number of common 2-mers is reached, although the strings $\mathcal{S}_2^*$ and G are quite different. Bottom-right: the 3-mers of $\mathcal{S}_3^*$ and G (there are 5 common 3-mers). Note that neither solution is optimal for both values of k, since $|a_2(\mathcal{S}_3^*, G)| = 7 < 9$, and $|a_3(\mathcal{S}_2^*, G)| = 4 < 5$. In this example, $\mathcal{S}_3^*$ should be more relevant than $\mathcal{S}_2^*$, since the former can be obtained from G with a single transposition event (by swapping factors aabca and abcb).

denotes the *multiset of common k-mers* in S and T. Note that counting common adjacencies is the special case $k = 2$. The problem we are interested in is thus defined as follows (see Figure 1 for an illustration with $k = 2$ and $k = 3$).

k-Mer Scaffold Filling (k-Mer-SF)
Input: A complete genome G and a scaffold $\mathcal{S}$ of contigs $(C_1, \ldots, C_m)$ over alphabet Σ, a multiset $\mathcal{T}$ of strings over Σ and integers, t_1, t_2 s.t. $t_1 \leq t_2 \leq |\mathcal{T}|$.
Task: Find $\mathcal{T}' \subseteq \mathcal{T}$, $t_1 \leq |\mathcal{T}'| \leq t_2$, and $\mathcal{S}^* \in \mathcal{S} + \mathcal{T}'$ s.t. $|a_k(\mathcal{S}^*, G)|$ is maximized.

Min-Breakpoint Scaffold Filling. The second problem we study here differs from One-Sided-Scaffold-Filling in two ways: first, just as k-Mer-SF, we allow that $\mathcal{T}$ contains strings instead of only letters. Second, instead of maximizing the number of common adjacencies, we aim at minimizing the number of breakpoints. We need additional definitions: given two strings S and T such that $c(S) = c(T)$, thus of same length n, and a bijection $\pi : [n] \to [n]$ such that $S[i] = T[\pi(i)]$, the *breakpoint distance wrt. π* between S and T is $|\{i \mid \pi(i+1) \neq \pi(i), 1 \leq i < n\}|$. The *string breakpoint distance* between S and T, denoted $b(S, T)$, is the minimum over all bijections π of the breakpoint distance of S and T wrt. π. We are now ready to define our second problem, which is also illustrated in Figure 2.

Min-Breakpoint Scaffold Filling (Min-Bkpt-SF)
Input: A complete genome G and a scaffold $\mathcal{S}$ over alphabet Σ and a multiset $\mathcal{T}$ of strings over Σ.
Task: Find $\mathcal{S}^* \in \mathcal{S} + \mathcal{T}$ s.t. $b(\mathcal{S}^*, G)$ is minimized.

Note that there exists attempts at defining breakpoints in strings in the context of scaffold filling [15, 16]. In that case, just as in permutations, breakpoints are defined as the dual of common adjacencies. This differs from the present definition: in our case, there exist strings

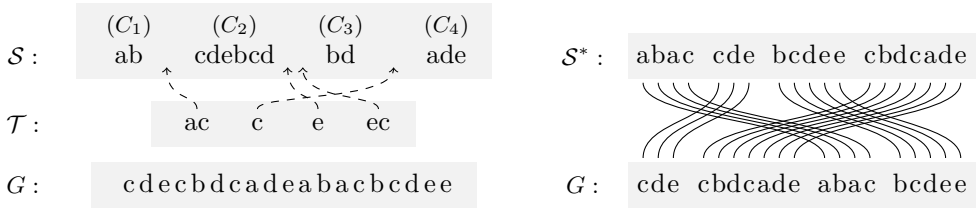

Figure 2 Left: an example instance of Min-Bkpt-SF, with scaffold $\mathcal{S}$ containing 4 contigs, $\mathcal{T}$ containing 4 strings of length 1 and 2 to be inserted, and a reference genome G. An optimal solution inserts the strings from $\mathcal{T}$ as indicated by dashed arcs to create $\mathcal{S}^*$. Right: the resulting string $\mathcal{S}^*$ is at breakpoint distance 3 from G: the letters of $\mathcal{S}$ and G can be matched together to form 4 common blocks (hence $4 - 1 = 3$ breakpoints).

S and T of length n such that $a(S,T) + b(S,T) = n + k$ with $k = \Omega(n)$. Moreover, our definition of breakpoints is of particular interest when S and T are very close: for instance, the number of common adjacencies cannot discriminate the case $S = T$ from $S \neq T$ (take e.g. S =aabbab and T =abaabb), whereas our definition of breakpoints does, and even allows to estimate how S differs from T.

Finally, note that following the motivation in genomic scaffolding, we demand that no insertions are made before C_1 or after C_m. This variant is more general than the one where insertions are allowed everywhere since we can simply add two additional contigs which have letters not occurring in G, one in the beginning and one in the end. Then, there is always an optimal solution that does not insert before the first or after the last contig.

k-Mer-SF and Min-Bkpt-SF are studied here from the parameterized (or multivariate) complexity point of view [5, 8, 10]. The main parameters that will be used in the following are: k, the length of the k-mers; $\ell := a_k(\mathcal{S}^*, G) - a_k(\mathcal{S}, G)$, the number of additional common k-mers brought by completion; d, the duplication number, that is, the maximum number of times a letter appears in G; m, the number of contigs in $\mathcal{S}$; t_2, the upper bound on number of letters to insert; λ, an upper bound on the length of the strings in $\mathcal{T}$; $b = b(\mathcal{S}^*, G)$, the sought breakpoint distance.

Related work and our contribution. Genomic Scaffold Filling (GSF) has been introduced by Muñoz et al. [20] in 2010. The problem has initially been defined for permutations, i.e. genomes are modeled as duplication-free sequences. Under this setting, GSF is polynomial-time solvable for the DCJ distance [20] and for maximizing the number of adjacencies [16, 19] (or, equivalently, minimizing the breakpoint distance). This method has been validated through simulations and the comparison of two plant genomes [20].

When scaffolds are modeled as strings (thus allowing gene repetitions), it becomes harder to compute relevant parsimony measures, hence almost all works are concerned with maximizing the number of common adjacencies. In many cases with this model, the "contig" constraint has been lifted, so that a scaffold has been modeled as a simple string, and insertion can be done between any pair of consecutive letters. For GSF, Jiang et al. [17, 16] showed the problem to be NP-hard, and from then on several approximation algorithms have been given – the best to date achieves a ratio of 1.2 [14]. Bulteau et al. [3] showed that GSF is FPT in the sought number of adjacencies ℓ.

More recently, as in this paper, scaffolds were considered to be a sequence $(C_1, \ldots, C_m)$ of contigs. Jiang et al. [13] considered GSF under this model, again with the maximum adjacency measure. They proved the problem to be NP-hard even if only two contigs are given, gave a 2-approximation for the problem, and showed the problem to be FPT

wrt. the combined parameter ℓ, the number of sought common adjacencies, and d, the duplication number. A short survey of the most recent results concerning GSF under the maximum adjacency setting can be found in [21]. In particular, the following question was raised [13, 21]: what is the FPT status of the problem when the parameter is the number of adjacencies?

In this paper, we study k-Mer-SF and Min-Bkpt-SF from a parameterized complexity point of view. In particular, we show that k-Mer-SF is W[1]-hard wrt. parameter t_2, and FPT wrt. parameter ℓ, thereby answering the above open question positively. We also provide a polynomial kernel for the parameter $\ell + m$ for the case where $t_2 = |\mathcal{T}|$, $\lambda = 1$ and $k = 2$, which corresponds to earlier definitions of the GSF problem. Concerning Min-Bkpt-SF, we provide hardness results even in some restricted cases (e.g. when $b = 0$ and $m = 2$), and we provide several FPT results wrt. combinations of some of the input parameters.

Preliminaries. For a string S, we use $S[i]$ to denote the letter at position i and $S[i, j]$ to denote the substring starting at position i and ending at position j; if $i > j$, then $S[i, j]$ is defined as the empty string. We use $S_k[i] := S[i, i + k - 1]$ to denote the length-k substring of S starting at position i. For two strings S and T we denote the concatenation of S and T by $S \circ T$. For a multiset or tuple of strings S, we use $||S||$ to denote the sum of the lengths of the strings contained in S. We use $[n] := \{1, \ldots, n\}$ to denote the numbers from 1 through n. For a multiset X over a universe U and an element u of U, let $m(X, u)$ denote the multiplicity of u in X. If $m(X, u) \geq 1$, then we write $x \in X$, if $m(X, u) = 0$, then we write $u \notin X$. We extend the definition of functions in a natural way to work with multiset domains. That is, a function $f : X \to Y$ with X a multiset is defined as a function $f_S : S_X \to Y$ where $S_X := \{(x, i) : x \in X, i \in [m(X, x)]\}$ is a set containing, for each $x \in X$, $m(X, x)$ many different elements corresponding to x. We write $f(x) := \{f_S((x, i)) : i \in [m(X, x)]\}$ to denote the set of images of x. Throughout the paper, $n := |G|$ will denote the length of the input genome G both in k-Mer-SF and Min-Bkpt-SF.

For the relevant definitions of parameterized complexity theory, refer to [8, 10].

2 The Relation between k-Mer Scaffold Filling and Partial Set Cover

In the following, we describe a reduction from the following variant of Set Cover to k-Mer-SF.

Partial Set Cover
Input: A family $\mathcal{F} = \{F_1, \ldots, F_m\}$ of subsets of a universe $U = \{u_1, \ldots, u_n\}$ and integers κ and τ.
Task: Find a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ of size at most κ such that $|\bigcup_{F_i \in \mathcal{F}'} F_i| \geq \tau$.

On the positive side, Partial Set Cover can be solved in $(2e)^\tau \cdot |U| \cdot |F|$ time [2]. For the parameter κ, however, Partial Set Cover is W[1]-hard [12].

▶ **Lemma 1.** *For each $k \geq 2$, there is a polynomial-time reduction from* Partial Set Cover *to* k-Mer-SF *such that $t_2 = \kappa$ and $|G| = \mathcal{O}(n)$.*

Proof. Given an instance of Partial Set Cover, construct an instance of k-Mer-SF for $k = 2$ as follows. For each u_i, introduce two letters a_i and b_i and introduce a further letter x. Now let G be the string $a_1 b_1 x a_2 b_2 x \cdots x a_n b_n$. Observe that each element in U corresponds to exactly one adjacency in G.

Now, for each $F_i \in \mathcal{F}$, construct a string T_i and add it to $\mathcal{T}$. The string T_i contains the substring $a_j b_j$ for each $u_j \in F_i$. More precisely, if $F_i = \{u_i^1, \ldots, u_i^q\}$, then $T_i = a_i^1 b_i^1 \cdots a_i^q b_i^q$. Now add two contigs $C_1 = C_2 = y$ to the scaffold $\mathcal{S}$ and set $t_1 = 0$ and $t_2 = \kappa$. This concludes the construction of the k-Mer-SF instance. It remains to show the correctness of the reduction, that is,

$(\mathcal{F}, \kappa, \tau)$ is a yes-instance of Partial Set Cover $\Leftrightarrow$ there is a solution $\mathcal{S}^*$ of $(G, \mathcal{S}, \mathcal{T}, 0, \kappa)$ for k-Mer-SF such that $|a_k(\mathcal{S}^*, G)| \geq \tau$.

($\Rightarrow$) Let $\mathcal{F}'$ be a solution of Partial Set Cover. Then, for each $F_i \in \mathcal{F}'$, insert the string T_i in $\mathcal{S}$ (in arbitrary order) and denote the resulting string by $\mathcal{S}^*$. Since the scaffold $\mathcal{S}$ contains no letters from G, all adjacencies of G are missing in $\mathcal{S}$. Let U' denote the elements that are covered by $\mathcal{F}'$. For each $u_j \in U'$, there is some F_i containing u_j and thus some T_i containing the adjacency $a_j b_j$ which is an adjacency of G. Thus, $\mathcal{S}^*$ contains at least $|U'|$ many adjacencies.

($\Leftarrow$) Let $\mathcal{S}^*$ be a solution such that $|a_k(\mathcal{S}^*, G)| \geq \tau$. Observe that every common adjacency of $\mathcal{S}^*$ and G is of the form $a_j b_j$, since all other adjacencies of G contain the letter x. Since each adjacency is of the form $a_j b_j$, there are thus at least τ distinct indices j such that $\mathcal{S}^*$ contains the adjacency $a_j b_j$. Moreover, every such adjacency is contained in some $T_i \in \mathcal{T}'$. By construction of $\mathcal{T}$, u_j is contained in the set F_i. Therefore, the set $\mathcal{F}' := \{F_i \mid T_i \in \mathcal{T}'\}$ covers at least τ elements of U. Since $\mathcal{T}' \leq \kappa$, there are thus at most κ sets in $\mathcal{F}$ that cover at least τ elements of U.

To obtain the reduction for arbitrary $k > 2$, one may adapt the construction by representing each u_i by a string of length k. ◀

Lemma 1 directly implies the following hardness result for the parameter t_2 that bounds the number of strings to insert.

▶ **Corollary 2.** *For each $k \geq 2$, k-Mer-SF is W[1]-hard with respect to the parameter t_2.*

Next, observe that Partial Set Cover is a special case of Set Cover. This implies that Partial Set Cover does not admit a polynomial kernel with respect to $|U| + \kappa$ unless coNP $\subseteq$ NP/poly [9]. Together with Lemma 1 and the facts that the decision version of k-Mer-SF is contained in NP and that Set Cover is NP-complete, we thus obtain the following.

▶ **Corollary 3.** *For each $k \geq 2$, k-Mer-SF does not admit a kernel with respect to $|G| + \lambda + t_2$ unless coNP $\subseteq$ NP/poly.*

3 A Fixed-Parameter Algorithm for k-Mer Scaffold Filling

We now show how to solve k-Mer-SF in $2^{\mathcal{O}(\ell)} \cdot n^{\mathcal{O}(1)}$ time. Let $p_k(\mathcal{S}, G) := a_k(G) \setminus a_k(\mathcal{S})$ denote the multiset of k-mers that is in G but not in the scaffold $\mathcal{S}$. We call these the *potential* common k-mers. Also, for a solution $\mathcal{S}^*$ we will call the common k-mers of $\mathcal{S}^*$ and G that are not k-mers of $\mathcal{S}$ the *realized k-mers*. The algorithm that we describe is based on a combination of dynamic programming and color-coding [1]. It has running time $\mathcal{O}(n^2 \cdot m \cdot k^3 \cdot \ell \cdot |\mathcal{T}| \cdot 8.16^{\ell} \cdot 5.44^{t_2})$. Thus, it is a fixed-parameter algorithm for the combined parameter $\ell + t_2$. As the following reduction rules show, this also gives a fixed-parameter algorithm for the parameter $k + \ell$.

▶ **Reduction Rule 1.** *If $t_1 > k \cdot \ell + 1$, then set $t_1 = k \cdot \ell + 1$. If $t_2 > k \cdot \ell + 1$, then set $t_2 = k \cdot \ell + 1$.*

Proof of Correctness. First, consider the change of t_1. The reduction rule decreases the value of the lower bound t_1 for $|\mathcal{T}'|$. Thus, every feasible solution for the original instance is a feasible solution for the reduced instance that realizes the same number of common k-mers. To show correctness, we must thus only show that for every feasible solution of the reduced instance, there is a feasible solution of the original instance that realizes the same number of common k-mers. To this end, let $\mathcal{T}^*$ be an optimal feasible solution of the new instance and let $\mathcal{S}^*$ denote the resulting string. Let $K := a_k(\mathcal{S}^*, G) \setminus (a_k(\mathcal{S}) \cup a_k(G))$ denote the multiset of potential common k-mers that are realized by $\mathcal{S}^*$. By definition of ℓ, we have $\ell = |K|$. Consider an injective mapping from K to the k-mers in $\mathcal{S}^*$ that contain at least one position from an inserted string $T \in \mathcal{T}^*$. Observe that the total number of positions in these k-mers of $\mathcal{S}^*$ is at most $k \cdot \ell$. By pigeonhole principle, there is thus at least one string $T \in \mathcal{T}^*$ such that none of the k-mers containing T is an image of the mapping. Now obtain a solution for the original instance by adding $t_2 - (k \cdot \ell + 1)$ strings from $\mathcal{T} \setminus \mathcal{T}^*$ directly after T. This does not affect any k-mers that are images of the mapping. Thus, the number of realized k-mers does not decrease, and the decrease of t_1 is correct.

Next, consider the case that the rule increases the value of the upper bound t_2 for $|\mathcal{T}'|$. Thus, every feasible solution for the reduced instance is a feasible solution for the original instance that realizes the same number of common k-mers. To show correctness, we must thus show that for every feasible solution of the original instance, there is a feasible solution of the reduced instance that realizes the same number of common k-mers. To this end, let $\mathcal{T}'$ be an optimal feasible solution of the original instance and assume that $\mathcal{T}'$ is the smallest among all optimal solutions. If $|\mathcal{T}'| \leq k \cdot \ell + 1$, then $\mathcal{T}'$ gives also a feasible solution for the reduced instance. Otherwise, as in the proof for the reduction of t_1, there is at least one string $T \in \mathcal{T}'$ such that none of the k-mers containing T is an image of the mapping from the realized k-mers. Now obtain a solution for the original instance by removing T. This does not affect any k-mers that are images of the mapping. Thus, the number of realized k-mers does not decrease. Moreover, since $t_1 \leq k \cdot \ell + 1$, $|\mathcal{T}' \setminus \{T\}| \geq t_1$. Thus, $|\mathcal{T}' \setminus \{T\}|$ is a feasible solution for the original instance which contradicts the assumption that $\mathcal{T}'$ is a smallest such solution. ◀

Color Coding. Somewhat deviating from the standard color-coding, we use two random colorings α and β. Here, $\alpha : p_k(\mathcal{S}, G) \to [\ell]$ is a coloring of the potential common k-mers and $\beta : \mathcal{T} \to [t_2]$ is a coloring of the strings that may be inserted. The idea is that there is a significant chance for the two random colorings that all of the realized common k-mers and inserted strings have different colors. Under this assumption, we can use dynamic programming on $\mathcal{S}$ to reconstruct a solution that realizes ℓ of the potential common k-mers.

Dynamic Programming. In the dynamic programming routine, we gradually find partial solutions of increasing size, inserting strings from $\mathcal{T}$ into the scaffold in a left-to-right manner. That is, we first insert between the first and second contig, then between the second and third and so on. We use the coloring to avoid inserting some string of $\mathcal{T}$ twice. We fill a five-dimensional table $Q[i, j, \kappa, A, B]$ with 0/1-entries corresponding to partial solutions. In this table:

- the index $i \in [m]$ corresponds to the set of contigs that precede the last character that was inserted,
- the indices $j \in [|G|]$ and $\kappa \in \{0, \ldots, k\}$ are used to identify the longest common suffix between the partial solution and G,

- the color sets $A \subseteq [\ell]$ and $B \subseteq [t_2]$ denote the colors of the k-mers that were realized and the colors of the strings that were inserted.

We define the meaning of table Q as follows. A table entry $Q[i, j, \kappa, A, B] = 1$ if and only if there is a set $p' \subseteq p_k(\mathcal{S}, G)$ and a set $\mathcal{T}' \subseteq \mathcal{T}$ such that

1. $\alpha(p') = A$ and $|p'| = |A|$,
2. $\beta(\mathcal{T}') = B$ and $|\mathcal{T}'| = |B|$,
3. there is a string $\mathcal{S}^* \in \mathcal{S}_i + \mathcal{T}'$ such that $a(\mathcal{S}^*, G) \setminus a(\mathcal{S}, G) \subseteq p'$ and $G_\kappa[j]$ is the longest common substring of G that is a suffix of $\mathcal{S}^*$, among all substrings of length at most k of G.

Here, $\mathcal{S}_i := (C_1, \ldots, C_i)$ denotes the scaffold consisting of the first i contigs in the same order as in $\mathcal{S}$ and $G_\kappa[j]$ denotes the length-κ substring of G starting at position j. Before we describe the recurrence in detail, consider the following. When extending a partial solution $\mathcal{S}^*$, we have two choices: either add a string from $\mathcal{T}$ at the end of $\mathcal{S}^*$ or add the next contig, that is, add C_i if the last contig in $\mathcal{S}^*$ is C_{i-1}. The resulting string contains additional k-mers as substrings and some of these k-mers may be potential common k-mers with G. Clearly, to determine the set of additional k-mers it suffices to know the length-k suffix of $\mathcal{S}^*$ and the string that we add. The number of different length-k suffixes, however, is $|\Sigma|^k$. Therefore, storing these in a dynamic programming table would incur a substantial overhead for both running time and space consumption. To be more efficient, we make use of the following fact.

▶ **Fact 1.** *Let S, T and G be strings such that the longest substring of S that is a suffix of G has length at most κ, and let $S' = (S \circ T)[i, j]$ be a substring of $S \circ T$ such that $i \leq |S|$ and $j > |S|$. If S' is a substring of G, then $i \geq |S| - \kappa$.*

Hence, the set of additional common k-mers of $\mathcal{S}^* \circ T$ and G are completely determined by the combination of T and the longest suffix of $\mathcal{S}^*$ that is also a substring of G. Finally, the additional realized k-mers are those that are not yet realized by $\mathcal{S}^*$. For our dynamic programming table, we are thus interested in the k-mers that have a certain color set. To determine the possible contribution of adding a string T we use a table $P[j, \kappa, A, T]$. An entry $P[j, \kappa, A, T]$ of P has value 1 if there is a surjective mapping

$$\psi : a_k(G_\kappa[j] \circ T, G) \to A$$

from the multiset of common k-mers of $G_\kappa[j] \circ T$ and G to the color set $A \subseteq [\ell]$ such that $\psi(x) \subseteq \alpha(x)$ for all $x \in a_k(G_\kappa[j] \circ T, G)$. Otherwise, the entry has value 0. Informally, the table P tells us whether adding T to a partial solution helps to realize potential k-mers with the colors of A. If we add a contig C_i, then we may count only those common k-mers that are not completely contained in C_i. Accordingly, the entries $P[j, \kappa, A, C_i]$ have value 1 if there is a surjective mapping

$$\psi : (a_k(G_\kappa[j] \circ C_{i+1}, G) \setminus a_k(C_i)) \to A$$

such that $\psi(x) \subseteq \alpha(x)$ for all $x \in a_k(G_\kappa[j] \circ T, G) \setminus a_k(C_i)$.

▶ **Lemma 4.** *The value of each entry of P can be computed in $\mathcal{O}(k \cdot |T| + |T| \cdot |A|^2)$ time if the multiset of k-mers of G is stored in a trie.*

Proof. First, in $\mathcal{O}(k + |T|)$ time, compute the string $T' = G_\kappa[j] \circ T$. Then compute the multiset $a_k(T', G)$ of common k-mers of T' and G. This can be done in $\mathcal{O}(k \cdot |T|)$ time: the number of k-mers in T' is at most $|T|$ and for each k-mer, we may use the trie to

check whether it is in G and to count the number of k-mers of T' that are equivalent. The multiset of common k-mers is then given by determining for each k-mer, the minimum of the multiplicities in T' and in G.

Now, we can determine whether there is a mapping ψ by computing a maximum matching in the graph that is defined by the restriction of α to $a_k(T', G)$, that is, the bipartite graph constructed as follows: For each k-mer K of $a_k(T', G)$ we introduce $m(a_k(T', G), K)$ vertices corresponding to K; this gives one part $V_{T'}$ of the bipartition. The other part of the bipartition is given by A. We draw an edge between $v \in V_{T'}$ and $u \in A$ if $v \in \alpha(K_v)$, where K_v is the k-mer corresponding to v. Then we compute a maximum matching in this graph. Since this matching has at most $|A|$ edges and since the graph has size $\mathcal{O}(|T| \cdot |A|)$, this can be done in time $\mathcal{O}(|T| \cdot |A|^2)$. If every vertex of A is contained in a matching edge, then the table entry is set to 1, otherwise it is 0. ◀

With the table P at hand, we use the following recurrence to fill Q. Informally, the table entry has value 1 if some string T or the next contig C_i can be used, together with a previous partial solution, to realize common k-mers of the desired colors and if the resulting partial solution has a suffix as specified by the values of j and κ.

$$Q[i, j, \kappa, A, B] = \begin{cases} 1 & \exists j', \kappa', A' \subseteq A, T \in \mathcal{T} : \\ & \quad Q[i, j', \kappa', A', B \setminus \beta(T)] = 1 \wedge \\ & \quad P[j', \kappa', A \setminus A', T] = 1 \wedge \\ & \quad G_\kappa[j] = \mathrm{len}(G, G_{\kappa'}[j'] \circ T) \\ 1 & \exists j', \kappa', A' : \\ & \quad Q[i-1, j', \kappa', A', B] = 1 \wedge \\ & \quad P[j', \kappa', A \setminus A', C_i] = 1 \wedge \\ & \quad G_\kappa[j] = \mathrm{len}(G, G_{\kappa'}[j'] \circ C_i) \\ 0 & \text{otherwise.} \end{cases}$$

Here, for two strings S and T we use $\mathrm{len}(S, T)$ to denote the longest substring of S that is a suffix of length at most k of T.

▶ **Theorem 5.** k-Mer-SF *can be solved in* $\mathcal{O}(n^2 \cdot m \cdot k^2 \cdot |\mathcal{T}| \cdot 8.16^\ell \cdot 5.44^{t_2})$ *time.*

Proof. The overall number of entries in P is $\mathcal{O}(n \cdot k \cdot |\mathcal{T}| \cdot 2^{t_2})$. Each entry of P can be computed in $\mathcal{O}(k \cdot |T| + |T| \cdot |A|^2)$ time by Lemma 4. This gives an overall running time of $\mathcal{O}(n \cdot k^2 \cdot ||\mathcal{T}|| \cdot 2^{t_2} \cdot (t_2)^2)$ for filling P. The overall number of entries in Q is $\mathcal{O}(m \cdot n \cdot k \cdot 2^\ell \cdot 2^{t_2})$. For each entry $Q[i, j, \kappa, A, B]$, we consider $\mathcal{O}(n \cdot k \cdot 2^{|A|} \cdot |\mathcal{T}|)$ cases in the recurrence. The first two conditions of each case can be determined in $\mathcal{O}(1)$ time, the third condition can be computed in $\mathcal{O}(1)$ time after a preprocessing in which we compute $\mathrm{len}(G, G_\kappa[j] \circ T)$ for each combination of T, κ and j. This can be done in $\mathcal{O}(n^2 \cdot k \cdot |\mathcal{T}|)$ time overall. Thus, the total running time for filling Q including preprocessing is $\mathcal{O}(n^2 \cdot m \cdot k^2 \cdot |\mathcal{T}| \cdot 3^\ell \cdot 2^{t_2})$. After Q is filled, we can determine whether there is a solution realizing ℓ potential common k-mers by considering $Q[m, j, \kappa, [\ell], B]$ for all B with $t_1 \leq |B| \leq t_2$. The overall running time of the algorithm now follows from the number of trials that are necessary to obtain a constant false-negative error probability, as shown by Alon et al. [1], these are exactly $e^{\ell + t_2}$ many. ◀

▶ **Corollary 6.** k-Mer-SF *can be solved in* $\mathcal{O}(n^2 \cdot m \cdot k^2 \cdot |\mathcal{T}| \cdot 8.16^\ell \cdot 5.44^{k\ell})$ *time.*

4 A Polynomial Kernel for a Special Case

As an additional result, we obtain a polynomial problem kernel for the special case when $t_2 \geq 2\ell + 1$, $\lambda = 1$ and $k = 2$ and the parameter is the combination of ℓ and m. Observe that $t_2 \geq 2\ell + 1$ essentially means that there are no upper-bound constraints on the solution size. Thus, our kernel also works for the natural case that $t_2 = |\mathcal{T}|$. The details are given in Appendix. Moreover, observe that even though the problem setting is very restricted compared to the general k-Mer Scaffold Filling, it contains the GSF problem of Jiang et al. [13] as a special case.

▶ **Theorem 7.** *For $k = 2$, $\lambda = 1$ and $t_2 \geq 2\ell + 1$, k-Mer-SF admits a problem kernel of size $\mathcal{O}(\ell^3 \cdot (\ell + m)^2)$ that can be computed in polynomial time.*

5 Minimizing the Number of Breakpoints

In this section, we consider the Min-Bkpt-SF problem. Another formulation of the string breakpoint distance between S and T is via minimum common string partitions [7]. Intuitively, the breakpoint distance is b if the strings S and T can each be partitioned into $b + 1$ factors, so that both partitions use the same multiset of factors. For example, strings *aabcda* and *bcaada* have a breakpoint distance of 2 since they can both be partitioned into the size-three factor set $\{aa, bc, da\}$ (see also Figure 2 for another example). This distance is NP-hard to compute [11], however, several FPT algorithms can be used. We will make use of the following two:

- An FPT algorithm for the parameter combining b, the breakpoint distance, and d, the number of duplications of any letter [4];
- An FPT algorithm for parameter b alone [6] (which is mainly of theoretical interest, as the exponential running time on b is rather prohibitive).

We first give two NP-hardness results, each one using a different approach giving different constraints on the values of the parameters. We then introduce two FPT algorithms using $|\mathcal{T}|$ as a parameter (as well as the breakpoint distance b, and either the number of contigs m or the duplication number d). Without parameter $|\mathcal{T}|$, we show that the problem is in XP for parameter b when all strings in $\mathcal{T}$ have length 1.

NP-hardness for $|\mathcal{T}| = 0$, $m = 1$, and either $d = 2$ or $|\Sigma| = 2$. The first hardness result below directly follows from the fact that the string breakpoint distance is hard to compute. Hence, any parameterized algorithm needs to put some restriction on the target distance b.

▶ **Theorem 8.** Min-Bkpt-SF *is NP-hard with $|\mathcal{T}| = 0$, $m = 1$ even when either $d = 2$ or $|\Sigma| = 2$.*

Proof. With an empty set $\mathcal{T}$ and a single contig C_1, Min-Bkpt-SF comes down to computing the breakpoint distance between two strings. It is NP-hard even in special cases of binary alphabet [18], as well as when any letter occurs at most twice [11]. ◀

NP-hardness for $m = 2$ and $b = 0$. When $b = 0$, we look for a way of inserting strings of $\mathcal{T}$ in $\mathcal{S}$ in order to obtain exactly G. This problem turns out to be NP-hard, hence, again, any parameterized algorithm needs not only to put a bound on the the target distance b, but also some restriction on the set of missing strings.

▶ **Theorem 9.** Min-Bkpt-SF *is NP-hard even when* $m = 2$ *and* $b = 0$.

Proof. We propose a reduction from Unary Bin Packing:

> **Input:** A list of n integers $(x_1, x_2, \ldots, x_n)$, given in unary, integers B and k such that, $kB = \sum x_i$.
> **Task:** Find a partition $(P_1, \ldots, P_k)$ of $[n]$ such that, for all $j \in [k]$, $\sum_{i \in P_j} x_i = B$.

We reduce to Min-Bkpt-SF as follows: let $G = (10^B)^k 1$, $\mathcal{S}$ consist of $m = 2$ contigs C_1 and C_2 with $C_1 = C_2 = 1$, let $\mathcal{T}$ contain $k-1$ strings 1 and strings 0^{x_i} for all $i \in [n]$. Finally, set $b = 0$. Consider $\mathcal{S}^* \in \mathcal{S} + \mathcal{T}$; $\mathcal{S}^*$ yields a partition of $[n]$ into k subsets P_j, where $i \in P_j$ if string 0^{x_i} from $\mathcal{T}$ is inserted between the $(j-1)$th and jth string 1 of $\mathcal{T}$ (or before the first/after the last for $j = 1$ and $j = k$ respectively). Then $b(\mathcal{S}^*, G) = 0$ if and only if each P_j, $j \in [k]$, is such that $\sum_{i \in P_j} x_i = B$. Conversely, any such partition of $[n]$ yields a string in $\mathcal{S} + \mathcal{T}$ at distance 0 from G. Hence, the instance $(\mathcal{S}, \mathcal{T}, G, b = 0)$ of Min-Bkpt-SF is a yes-instance if and only if the original Unary Bin Packing instance is a yes-instance. ◀

An FPT Algorithm for the parameter $(|\mathcal{T}|, m, b)$. We now present an algorithm for the case that the parameter combines the number τ of strings in $\mathcal{T}$, the number of contigs m, and the breakpoint distance b.

▶ **Theorem 10.** Min-Bkpt-SF *is FPT for parameters* $|\mathcal{T}|$, b *and* m.

Proof. If we consider parameters $|\mathcal{T}|$ and m, then together they allow the exhaustive enumeration of all possible strings in $\mathcal{S} + \mathcal{T}$: First, compute the $|\mathcal{T}|!$ possible arrangements of strings in $\mathcal{T}$, then split the resulting string into at most m blocks without breaking substrings corresponding to the strings in $\mathcal{T}$ (this creates at most $\binom{|\mathcal{T}|}{m} \leq 2^{|\mathcal{T}|}$ branches), then consider all choices to insert them between the contigs of $\mathcal{S}$ (this creates at most 2^m branches).

Once a candidate $\mathcal{S}^*$ is known, it remains to compute the breakpoint distance with G in time $f(b)\mathrm{n}^{\mathcal{O}(1)}$ [6]. Overall, this gives an FPT algorithm for parameters $|\mathcal{T}|$, b and m. ◀

An FPT Algorithm for the parameter $(|\mathcal{T}|, d, b)$. The next algorithm is more efficient if the number of duplications d is small and avoids the dependency on the contig number m.

▶ **Theorem 11.** Min-Bkpt-SF *can be solved in time* $\mathcal{O}((4|\mathcal{T}|d^2)^{|\mathcal{T}|} d^{2b} b n^2)$.

Proof. Given an optimal solution $\mathcal{S}^*$, we call T-factor a maximal factor of $\mathcal{S}^*$ containing strings from $\mathcal{T}$ (a T-factor is the concatenation of all substrings inserted between two contigs). A T-factor is *left-joined* (resp. *right-joined*) if there is no breakpoint to its left, i.e., between the last letter of the previous contig and its own first letter (resp., to its right). A T-factor is *stand-alone* if it is neither left- nor right-joined. We can assume wlog. that there is a single stand-alone T-factor, as any two such factors can be inserted in the same gap between two contigs, so that they are merged into one without increasing the distance.

The first step of our algorithm is to guess the T-factors (using $|\mathcal{T}|^{|\mathcal{T}|}$ branches). They will be denoted $f_1, \ldots, f_h$. For each T-factor, we guess whether it is left-joined, and whether it is right-joined (using $4^{|\mathcal{T}|}$ branches). We first deal with the single stand-alone T-factor, if any: guess the correct gap where it should be inserted (among $m \leq n$ choices), insert it there and merge it with the surrounding two contigs. Consider now a T-factor f_j, assume that it is left-joined (otherwise it is necessarily right-joined, then processed symmetrically). Let u be the first letter of f_j. Guess the position i such that $G[i]$ is matched to u (among d options). Let u' be the letter at position $i-1$ in G. Since there is no breakpoint before u, f_j

must be inserted after a contig ending with u'. There are at most d such contigs, so we can enumerate all options. There are at most d^2 branches for each T-factor, and $d^{2|\mathcal{T}|}$ branches overall. The branching above allows to enumerate all candidate strings for a solution, it remains to check whether any of them has breakpoint distance b to G. We compute this distance for each candidate, using an FPT algorithm [4] with running time $\mathcal{O}(d^{2b}bn)$. The overall running time is $\mathcal{O}((4|\mathcal{T}|d^2)^{|\mathcal{T}|}d^{2b}bn^2)$. ◀

An XP Algorithm for the parameter b when $\lambda = 1$

▶ **Theorem 12.** Min-Bkpt-SF *can be solved in time* $\mathcal{O}(n^{b+1}(b+1)!)$ *when all strings in* $\mathcal{T}$ *have length 1.*

Proof. The algorithm runs as follows: first enumerate all possible positions of the breakpoints in G, using $|G|^b$ branches. The optimal string $\mathcal{S}^*$ can be guessed by trying all possible rearrangements of the $b+1$ factors separated by breakpoints, using $(b+1)!$ branches.

It remains to check that $\mathcal{S}^* \in \mathcal{S} + \mathcal{T}$, i.e., some filling of $\mathcal{S}$ gives $\mathcal{S}^*$. This task is NP-hard in the general case (see Theorem 9) however, it is straightforwardly achieved in linear time when all strings in T have length 1: it suffices to check that all contigs in $\mathcal{S}$ are factors of $\mathcal{S}^*$ in the correct order, and that $\mathcal{T}$ contains exactly the multi-set of missing letters. ◀

6 Conclusion

For k-Mer-SF, the most interesting direction seems to be to extend the problem kernelization to more general cases by allowing either $k > 2$ (that is, considering k-mer distance for general k), $\lambda > 1$ (that is, allowing the insertion of strings or letters), or considering the case where the number of scaffold contigs is unbounded. For Min-Bkpt-SF it remains open whether the problem is FPT for other parameters than $|\mathcal{T}|$, for example m, b or d. We conjecture that the FPT algorithm for MCSP with parameters b and d [4] can be extended for our problem with this combination of parameters. Hopefully some new techniques might reduce the complexity to achieve fixed-parameter tractability for $b+d$ or $b+\ell$ only.

Finally, from a broader point of view, the problems that we consider here are fundamental on strings. Indeed, they belong to a larger family of problems that can be described as follows: Given a string G and a partial string S, complete the partial string S such that it is as close as possible to G. Investigating this type of problems more systematically could be a rewarding topic.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995. doi:10.1145/210332.210337.

2 Markus Bläser. Computing small partial coverings. *Inf. Process. Lett.*, 85(6):327–331, 2003. doi:10.1016/S0020-0190(02)00434-9.

3 Laurent Bulteau, Anna Paola Carrieri, and Riccardo Dondi. Fixed-parameter algorithms for scaffold filling. *Theor. Comput. Sci.*, 568:72–83, 2015. doi:10.1016/j.tcs.2014.12.005.

4 Laurent Bulteau, Guillaume Fertin, Christian Komusiewicz, and Irena Rusu. A fixed-parameter algorithm for minimum common string partition with few duplications. In Aaron E. Darling and Jens Stoye, editors, *Proceedings of the 13th International Workshop on Algorithms in Bioinformatics (WABI 2013)*, volume 8126 of *LNCS*, pages 244–258. Springer, 2013. doi:10.1007/978-3-642-40453-5_19.

5 Laurent Bulteau, Falk Hüffner, Christian Komusiewicz, and Rolf Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bull. EATCS*, 114, 2014. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/310`.

6 Laurent Bulteau and Christian Komusiewicz. Minimum common string partition parameterized by partition size is fixed-parameter tractable. In Chandra Chekuri, editor, *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 102–121. SIAM, 2014. `doi:10.1137/1.9781611973402.8`.

7 Xin Chen, Jie Zheng, Zheng Fu, Peng Nan, Yang Zhong, Stefano Lonardi, and Tao Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 2(4):302–315, October 2005. `doi:10.1109/TCBB.2005.48`.

8 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. `doi:10.1007/978-3-319-21275-3`.

9 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and IDs. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. `doi:10.1145/2650261`.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. `doi:10.1007/978-1-4471-5559-1`.

11 Avraham Goldstein, Petr Kolman, and Jie Zheng. Minimum common string partition problem: Hardness and approximations. *Electron. J. Comb.*, 12, 2005. URL: `http://www.combinatorics.org/Volume_12/Abstracts/v12i1r50.html`.

12 Jiong Guo, Rolf Niedermeier, and Sebastian Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501–520, 2007. `doi:10.1007/s00224-007-1309-3`.

13 Haitao Jiang, Chenglin Fan, Boting Yang, Farong Zhong, Daming Zhu, and Binhai Zhu. Genomic scaffold filling revisited. In Roberto Grossi and Moshe Lewenstein, editors, *Proceedings of the 27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *LIPIcs*, pages 15:1–15:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.CPM.2016.15`.

14 Haitao Jiang, Jingjing Ma, Junfeng Luan, and Daming Zhu. Approximation and nonapproximability for the one-sided scaffold filling problem. In Dachuan Xu, Donglei Du, and Ding-Zhu Du, editors, *Proceedings of the 21st International Conference on Computing and Combinatorics (COCOON 2015)*, volume 9198 of *LNCS*, pages 251–263. Springer, 2015. `doi:10.1007/978-3-319-21398-9_20`.

15 Haitao Jiang, Chunfang Zheng, David Sankoff, and Binhai Zhu. Scaffold filling under the breakpoint distance. In Eric Tannier, editor, *Proceedings of the International Workshop on Comparative Genomics (RECOMB-CG 2010)*, volume 6398 of *LNCS*, pages 83–92. Springer, 2010. `doi:10.1007/978-3-642-16181-0_8`.

16 Haitao Jiang, Chunfang Zheng, David Sankoff, and Binhai Zhu. Scaffold filling under the breakpoint and related distances. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 9(4):1220–1229, 2012. `doi:10.1109/TCBB.2012.57`.

17 Haitao Jiang, Farong Zhong, and Binhai Zhu. Filling scaffolds with gene repetitions: Maximizing the number of adjacencies. In Raffaele Giancarlo and Giovanni Manzini, editors, *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching (CPM 2011)*, volume 6661 of *LNCS*, pages 55–64. Springer, 2011. `doi:10.1007/978-3-642-21458-5_7`.

18 Haitao Jiang, Binhai Zhu, Daming Zhu, and Hong Zhu. Minimum common string partition revisited. *J. Comb. Optim.*, 23(4):519–527, 2012. `doi:10.1007/s10878-010-9370-2`.

19 Nan Liu, Peng Zou, and Binhai Zhu. A polynomial time solution for permutation scaffold filling. In T.-H. Hubert Chan, Minming Li, and Lusheng Wang, editors, *Proceedings of the 10th International Conference on Combinatorial Optimization and Applica-*

tions (COCOA 2016), volume 10043 of *LNCS*, pages 782–789. Springer, 2016. doi:10.1007/978-3-319-48749-6_60.

20 Adriana Muñoz, Chunfang Zheng, Qian Zhu, Victor A. Albert, Steve Rounsley, and David Sankoff. Scaffold filling, contig fusion and comparative gene order inference. *BMC Bioinformatics*, 11:304, 2010. doi:10.1186/1471-2105-11-304.

21 Binhai Zhu. Genomic scaffold filling: A progress report. In Daming Zhu and Sergey Bereg, editors, *Proceedings of the 10th International Workshop on Frontiers in Algorithmics (FAW 2016)*, volume 9711 of *LNCS*, pages 8–16. Springer, 2016. doi:10.1007/978-3-319-39817-4_2.

A Kernelization Algorithm from Section 4

In this section, we present an kernelization algorithm for the parameter ℓ and the special case when $t_2 =\geq 2\ell + 1$, $\lambda = 1$ and $k = 2$ and m is a constant.

To obtain a problem kernel, we need to reduce the size of three objects: the genome G, the scaffold $\mathcal{S}$, and the multiset $\mathcal{T}$ of letters that we may add. We will achieve this in two steps: First, we will reduce the number of copies of letters in $\mathcal{T}$ and of copies of adjacencies and the size of $\mathcal{S}$. After this reduction step, we observe that it only remains to reduce the number of different letters in the instance. Consequently, this is what we reduce in the remainder of the algorithm.

Throughout the description of the algorithm, let x denote a letter that does not occur in $\mathcal{S}$ and not in $\mathcal{T}$ and let y denote a letter that does not occur in G.

The first rule for the kernel removes superfluous copies of letters from $\mathcal{T}$. It is obviously correct, since no solution can add such letters.

▶ **Reduction Rule 2.** *If $\mathcal{T}$ contains a letter b more than t_2 times, remove a copy of b from $\mathcal{T}$.*

The next rule, Rule 3 aims at separating the adjacencies in G and keeping only the potential adjacencies in G. This will be useful to reduce the size of G but also to reduce the size of $\mathcal{S}$, which is done by Rule 4. Recall that the potential common k-mers of an instance are the k-mers which are contained in G not in $\mathcal{S}$. For $k = 2$, we may speak of *potential common adjacencies*. Also, for a solution $\mathcal{S}^*$ we will call the common adjacencies of $\mathcal{S}^*$ and G that are not adjacencies of $\mathcal{S}$ the *realized adjacencies*.

▶ **Reduction Rule 3.** *Let x be a letter that is not contained in any contig of $\mathcal{S}$ and not contained in any string of $\mathcal{T}$, and let $p(\mathcal{S}, G) := \{P_1, \ldots, P_q\} = a(G) \setminus a(\mathcal{S})$ denote the potential common adjacencies of G and $\mathcal{S}$. Replace G by $P_1 \circ x \circ P_2 \circ \cdots \circ P_{q-1} \circ x \circ P_q$.*

▶ **Reduction Rule 4.** *Let y be a letter not contained in G. Replace every contig C_i of length at least 2 by $C_i[1] \circ y \circ C_i[|C_i|]$.*

▶ **Lemma 13.** *Let $I = (G, \mathcal{S}, \mathcal{T}, t_1, t_2)$ be an instance of k-Mer-SF, and let $I' = (G', \mathcal{S}', \mathcal{T}', t_1', t_2')$ be the instance obtained from I by applying Rules 3 and 4 to I. Then, I and I' are equivalent.*

Proof. A solution for I realizing at least ℓ adjacencies, directly gives a solution for I' realizing at least ℓ adjacencies: Every realized adjacency for I is from $p(\mathcal{S}, G) = a(G) \setminus a(\mathcal{S})$ and thus contained in $a(G')$. Moreover, it is not contained in $a(\mathcal{S}')$ and thus it is a potential common adjacency in I'. Now since $\mathcal{T} = \mathcal{T}'$ and since, for any solution of I, all realized adjacencies contain either only letters from $\mathcal{T}$ or only the first or the last letter from some contig C_i, they can be realized in I' as well by inserting the letters in the same order and between the same contigs as in I. The converse direction follows from the same arguments. ◀

Observe that after application of Rule 4, the scaffold has size $\mathcal{O}(m)$.

After separating the potential adjacencies in G with the help of Rule 3, we may now speak of *removing a copy of a potential adjacency bc from* G, which means to replace

$$G = P_1 \circ y \circ \cdots \circ P_{i-1} \circ y \circ bc \circ y \circ P_{i+1} \cdots \circ P_q$$

by

$$P_1 \circ y \circ \cdots \circ P_{i-1} \circ y \circ P_{i+1} \cdots \circ P_q$$

for some arbitrary $P_i = bc$.

▶ **Reduction Rule 5.** *If there is a potential adjacency bc that occurs more than ℓ times in G, then remove a copy of this adjacency from G.*

Proof of Correctness. Let I denote the original instance and let I' denote the instance obtained by the application of the rule. We need to show only that if I has a solution, then so does I', as the other direction is trivial. Thus, assume that I has a solution realizing at least ℓ adjacencies. Choose an arbitrary multiset P of ℓ realized adjacencies and observe that there is at least one copy of the adjacency bc in G that is not in P. Thus, removing this adjacency from G gives a new genome G^* such that $a(G^*) \setminus a(\mathcal{S}) \geq |P| \geq \ell$ since the adjacencies of P are contained in $a(G^*)$. Since the multiset of potential common adjacencies of G^* and of the genome in I' are the same and since the scaffold $\mathcal{S}$ and the set $\mathcal{T}$ are not changed by the rule, I' has a solution realizing at least ℓ common adjacencies. ◀

As the following lemma shows, we have already achieved the goal of the first step, that is, we have reduced the number of copies of all letters in I.

▶ **Lemma 14.** *Let I be an instance that is reduced with respect to Rules 1–5, and let c denote the number of different letters occurring in G, $\mathcal{S}$, and $\mathcal{T}$. Then, $|G| \leq 3\ell c^2$ and $||\mathcal{T}|| \leq c \cdot (2\ell + 1)$.*

Proof. First, we bound the size of G. If the overall number of letters is c, then there are at most c^2 different adjacencies in G. Moreover, by the construction of G, every third adjacency does not contain x. Thus, if $|G| > 3\ell \cdot c^2$, then there is some adjacency that does not contain the letter x and that occurs more than ℓ times. This contradicts the assumption that the instance is reduced with respect to Rule 5.

The bound on the total length of $\mathcal{T}$ follows from the fact that $\mathcal{T}$ contains at most c many different letters, each of which occurs at most $2\ell + 1$ times since the instance is reduced with respect to Rule 1 and 2. ◀

According to Lemma 14, to obtain a polynomial problem kernel it is sufficient to reduce the number of letters in the instance to $\ell^{\mathcal{O}(1)}$. Consequently, this is our aim in the second step of the kernelization algorithm.

First, we remove those letters from G and $\mathcal{T}$ which are useless in the sense that they occur in no adjacencies which can become common adjacencies of a solution $\mathcal{S}^*$ and G.

▶ **Definition 15.** We call an adjacency bc *realizable* if bc occurs in G, and

- $b \in \mathcal{T}$ and $c \in \mathcal{T}$, or
- $b = C_i[|C_i|]$ and $c \in \mathcal{T}$ for some contig C_i, or
- $b = C_i[|C_i|]$ and $c = C_{i+1}[1]$ for some contig C_i, or
- $b \in \mathcal{T}$ and $c = C_i[1]$ for some contig C_i.

We now remove those letters that do not occur in realizable adjacencies.

▶ **Reduction Rule 6.** *If $|\mathcal{T}| > t_1$ and $\mathcal{T}$ contains a letter b that occurs in no realizable adjacency, then remove a copy of b from $\mathcal{T}$.*

If G contains an adjacency bc that does not contain x and that cannot be realized, then remove bc from G.

The correctness of the rule above follows in a straightforward manner from the fact that we will never insert a letter that is removed by the rule or realize an adjacency that is removed by the rule.

For the final step of the kernelization, we build an auxiliary letter-adjacency graph $H = (V, E)$ as follows. For each letter in $\mathcal{T}$, G, and $\mathcal{S}$, add one vertex to H. Make two vertices b and c adjacent in this graph if the adjacency bc or the adjacency cb is realizable. Observe that after application of Rule 6, every vertex in H except x,y, and possibly the $2m-2$ vertices corresponding to letters of contigs, has at least one neighbor. Thus, our aim in the following is to reduce the number of vertices in H that have at least one neighbor.

▶ **Reduction Rule 7.** *Let M be a maximum matching in G. If $|M| \geq \ell + 1$, then answer "yes".*

Proof of Correctness. We show how to construct a solution for the k-Mer-SF instance. Let $\{\{b_1, c_1\}, \{b_2, c_2\}, \ldots, \{b_{\ell+1}, c_{\ell+1}\}\}$ be a set of $\ell + 1$ edges contained in M and assume without loss of generality that $b_i c_i$ is a realizable adjacency for each i.

First, assume that for all $i \in [\ell + 1]$ either $b_i \in \mathcal{T}$ or $c_i \in \mathcal{T}$. For each i, do the following. If $b_i \in \mathcal{T}$ and $c_i \in \mathcal{T}$, add $b_i c_i$ between C_1 and C_2. If $b_i \in \mathcal{T}$ and $c_i = C_j[1]$ for some C_j, then add b_i directly in front of C_j. If $b_i = C_j[|C_j|]$ and $c_i \in \mathcal{T}$, then add c_i directly after C_j. The set of inserted letters realizes at least $\ell+1$ adjacencies, since the $\ell+1$ pairs $\{b_i, c_i\}$ are disjoint and since for each we realize one adjacency. To obtain a feasible solution, insert $t_1 - (\ell + 1)$ further letters at an arbitrarily chosen position. This breaks at most one adjacency thus giving a solution that realizes at least ℓ adjacencies.

If for some i, we have $b_i = C_j[|C_j|]$ and $c_i = C_{j+1}[1]$, then choose an arbitrary such i and add all adjacencies $b_q c_q$ with $b_q \in \mathcal{T}$ and $c_q \in \mathcal{T}$ between C_j and C_{j+1}. Add $t_1 - (\ell + 1)$ further letters right before C_{j+1}. Insert all other letters as described above. The number of realized adjacencies is at least ℓ. All adjacencies of M except the adjacency $b_i c_i$ are realized: if at least one letter in the adjacency is from $\mathcal{T}$, then they are realized because this letter is inserted in the right position. If neither b_j nor c_j are from T, then the adjacency is realized because no letters are inserted between the consecutive contigs that contain b_j and c_j. ◀

Now let $V(M)$ denote the endpoints of the matching. We show that if $|V \setminus V(M)| > (2\ell + m) \cdot |V(M)|$, then we can safely remove some adjacency from G.

To apply the next rule, we build two bipartite graphs H^1 and H^2. In both graphs, the vertex parts are $B := V(M)$ and $C := (V \setminus V(M))$. In H^1, we add an edge between $b \in B$ and $c \in C$ when bc is a realizable adjacency. In H^2, we add an edge between $b \in B$ and $c \in C$ when cb is a realizable adjacency.

▶ **Reduction Rule 8.**

- *If there is a vertex $b \in B$ of degree at least $2\ell + m + 1$ in H^1, then remove the adjacency bc from G, where c is an arbitrary neighbor of b.*
- *If there is a vertex $b \in B$ of degree at least $2\ell + m + 1$ in H^2, then remove the adjacency cb from G, where c is an arbitrary neighbor of b.*

Proof of Correctness. We show the correctness for the first part of the rule, the second part is symmetric. Consider an instance before application of the rule and assume it has a solution realizing at least ℓ adjacencies. If none of these adjacencies is bc, the adjacency removed from G by the rule, then this solution directly implies a solution for the reduced instance. Otherwise, fix an arbitrary minimal set P of positions containing a letter of the ℓ many realized adjacencies. Observe that $|P| \leq 2\ell$ and there are at most ℓ pairs of consecutive contigs that have nonempty intersection with these positions. Now, consider the adjacency bc that is contained in the solution but not contained in G. There are at least $m+1$ letters d that are adjacent to b in H^1 such that not all copies of d are contained in P. Of these, at most m are letters from contings. Thus, there is a d such that $d \in \mathcal{T}$ and not all copies of d are contained in P. Therefore, inserting d behind b destroys the adjacency bc, but instead creates the adjacency bd. This adjacency is also contained in G and not realized by any position of P. This restores the number of realized adjacencies to ℓ. ◀

▶ **Theorem 7.** *For $k = 2$, $\lambda = 1$ and $t_2 \geq 2\ell + 1$, k-Mer-SF admits a problem kernel of size $\mathcal{O}(\ell^3 \cdot (\ell + m)^2)$ that can be computed in polynomial time.*

Proof. Consider an instance that is reduced with respect to all presented reduction rules. By Lemma 14, our claim follows if we show that the number of letters in I is $\mathcal{O}(\ell \cdot (\ell + m))$. This can be seen by considering the graph H: the graph H has $\mathcal{O}(m)$ vertices that have no neighbors. The number of further vertices is $\mathcal{O}(\ell \cdot (\ell + m))$: After applying Rule 7, the size of the matching M is $\mathcal{O}(\ell)$. Any vertex in H that is incident with at least one edge and not an endpoint of M is adjacent to a vertex of M either in H^1 or in H^2. After applying Rule 8, the number of these vertices is at most $|V(M)| \cdot 2 \cdot (2\ell + m + 1) = \mathcal{O}(\ell \cdot (\ell + m))$. This gives the bound on the number of vertices in G and thus on the instance size.

The running time follows from the fact that all reduction rules can be clearly performed in polynomial time. ◀