Genomic Scaffold Filling Revisited

Haitao Jiang[1], Chenglin Fan[2], Boting Yang[3], Farong Zhong[4], Daming Zhu[5], and Binhai Zhu[6]

1 **School of Computer Science and Technology, Shandong University, Jinan, Shandong, China**
htjiang@sdu.edu.cn

2 **Department of Computer Science, Montana State University, Bozeman, MT 59717, USA**
chenglin.fan@msu.montana.edu

3 **Department of Computer Science, University of Regina, Regina, Saskatchewan S4S 0A2, Canada**
boting.yang@uregina.ca

4 **College of Math, Physics and Information Technology, Zhejiang Normal University, Jinhua, Zhejiang, China**
zfr@zjnu.cn

5 **School of Computer Science and Technology, Shandong University, Jinan, Shandong, China**
dmzhu@sdu.edu.cn

6 **Department of Computer Science, Montana State University, Bozeman, MT 59717, USA**
bhz@montana.edu

Abstract

The genomic scaffold filling problem has attracted a lot of attention recently. The problem is on filling an incomplete sequence (scaffold) I into I', with respect to a complete reference genome G, such that the number of adjacencies between G and I' is maximized. The problem is NP-complete and APX-hard, and admits a 1.2-approximation. However, the sequence input I is not quite practical and does not fit most of the real datasets (where a scaffold is more often given as a list of contigs). In this paper, we revisit the genomic scaffold filling problem by considering this important case when, (1) a scaffold S is given, the missing genes $X = c(G) - c(S)$ can only be inserted in between the contigs, and the objective is to maximize the number of adjacencies between G and the filled S', and (2) a scaffold S is given, a subset of the missing genes $X' \subset X = c(G) - c(S)$ can only be inserted in between the contigs, and the objective is still to maximize the number of adjacencies between G and the filled S''. For problem (1), we present a simple NP-completeness proof, we then present a factor-2 greedy approximation algorithm, and finally we show that the problem is FPT when each gene appears at most d times in G. For problem (2), we prove that the problem is W[1]-hard and then we present a factor-2 FPT-approximation for the case when each gene appears at most d times in G.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity

Keywords and phrases Computational biology, Approximation algorithms, FPT algorithms, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.CPM.2016.15

27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016).
Editors: Roberto Grossi and Moshe Lewenstein; Article No. 15; pp. 15:1–15:13

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The cost of sequencing a genome has been reduced significantly in the last decade, with the current cost being around $1k. This results in a lot of genomes being sequenced, usually not completely finished (we call them *draft* genomes). On the other hand, the cost to finish these genomes completely has not been decreased as much compared with a decade ago [9]. The result is that we are having more and more draft genomes. On the other hand, for many tools to analyze the genomic data we do need complete genomes. For instance, to compute the reversal distance between two genomes we do need two complete genomes. Hence, there is a need to turn a draft genome into a complete one.

To make the result biologically interesting, Munoz *et al.* first proposed the following *scaffold filling* problem (on multichromosomal genomes with no gene repetition) as follows [28]. Given a complete (permutation) genome R and an incomplete scaffold S, fill the missing genes in $R - S$ into S to have S' such that the genomic distance (or DCJ distance [30]) between R and S' is minimized. It was shown that this problem can be solved in polynomial time. In [22], Jiang *et al.* considered the case for singleton genomes without gene repetition (i.e., permutations), using the simplest *breakpoint* distance as the similarity measure. It was shown that this problem is solvable in polynomial time; in fact, even for the two-sided case when both the input scaffolds, being a reference to each other, are incomplete permutations.

When the genomes and scaffolds contain gene repetitions, the problem becomes harder. (That should not be considered as a surprise as even computing certain similarity measure between two complete genomes is NP-complete, for instance, with the exemplar breakpoint distance [11, 13, 2, 5, 24], exemplar adjacency number [12, 14], or the minimum common string partition [15].) The similarity measure adopted for the scaffold filling problem is the *number of common (string) adjacencies*, which can be computed in polynomial time [2, 21, 22]. In [21, 22], it was shown by Jiang *et al.* that scaffold filling to maximize the number of common string adjacencies (SF-MNSA) is NP-hard. (Formally, the problem is to fill an incomplete sequence scaffold I into I', with respect to a complete reference genome G, such that the missing letters in $G - I$ are inserted back to I and the number of common adjacencies between G and I' is maximized.) A factor-1.33 approximation was designed in [21, 22], and this bound has been improved to 1.25 [25], and to 1.20 [23]. For the corresponding two-sided case, i.e., when two scaffolds are references to each other, the problem admits a factor-1.5 approximation with the number of common adjacencies between the filled scaffolds being maximized [26]. Using the number of common adjacencies as a parameter, it was shown that this problem is also fixed-parameter tractable (FPT) – this only handles that case when G and I' are not very similar so it is only of a theoretical meaning [7].

The motivation of this paper is two-fold. Firstly, the 'scaffold' used in most of these papers is an incomplete sequence, i.e., a missing gene can be inserted anywhere in such a 'scaffold'. In practice, most of the real datasets are not in this format; in fact, a scaffold in a real dataset is usually composed of a sequence of contigs, where a contig is usually computed with mature tools like BLAST [1], hence should not be arbitrarily altered. This case was considered briefly in [28, 22], all other research on scaffold filling used an incomplete sequence as a scaffold. Secondly, take a complete reference genome G and a scaffold S, there is no guarantee that the filled scaffold S' is of the same length as that of G; in fact, sometimes we could know roughly the length of the target genome S^* (S' should be as close to S^* as possible). Then, we might only need to insert a subset of letters in $G - S$ into S (to obtain S').

The main contribution of this paper is to present some research results along these two lines. We formally call the two problems as One-sided Scaffold Filling (One-sided-SF-max), and One-sided Subset Scaffold Filling (One-sided-SF-max($\subset$)). (For the important practical case when a gene can only appear at most d times in G, we call the corresponding problems One-sided-SF-max(d) and One-sided-SF-max($\subset, d$) respectively.) The objective function in both cases are to maximize the number of common adjacencies between the reference and the filled scaffold. For One-sided-SF-max, we present a simple reduction from the Hamiltonian Path problem hence showing it to be NP-hard, we then present a factor-2 approximation. Then we show that One-sided-SF-max(d) is FPT. For One-sided-SF-max($\subset$), we prove a stronger negative result by showing that, parameterized by the number of missing genes inserted, the problem is W[1]-hard. We then present a factor-2 FPT-approximation for the special case One-sided-SF-max($\subset, d$). As far as we know, this is the first W[1]-hardness result on the research of scaffold filling.

The paper is organized as follows. In Section 2, we give the preliminaries. In Section 3, we present the approximation results for One-sided-SF-max. In Section 4, we present the FPT algorithm for One-sided-SF-max(d). In Section 5, we present the results for One-sided-SF-max($\subset$). We conclude the paper in Section 6.

2 Preliminaries

Throughout this paper we focus only on singleton genomes (i.e., each is a sequence). But the results can be easily generalized to multichromosomal or circular genomes, with minor changes.

At first, we review some necessary definitions, which are also defined in [22, 31]. We assume that all genes and genomes are unsigned, and it is straightforward to generalize the result to signed genomes. Given a gene set Σ, a string P is called *permutation* if each element in Σ appears exactly once in P. We use *c(P)* to denote the set of elements in permutation P. A string A is called *sequence* if some genes appear more than once in A, and $c(A)$ denotes genes of A, which is a multi-set of elements in Σ. For example, $\Sigma = \{a, b, c, d\}$, $A = abcdacd$, $c(A) = \{a, a, b, c, c, d, d\}$. A *sequence scaffold* is an incomplete sequence, typically obtained by some sequencing and assembling process. A substring with m genes (in a sequence) is called an *m-substring*, and a 2-substring is also called a *pair*; as the genes are unsigned, the relative order of the two genes of a pair does not matter, i.e., the pair xy is equal to the pair yx. Given an incomplete sequence (or sequence scaffold) $A{=}a_1a_2a_3\cdots a_n$, let $P_A = \{a_1a_2, a_2a_3, \ldots, a_{n-1}a_n\}$ be the set of pairs in A.

▶ **Definition 1.** Given two sequence scaffolds $A{=}a_1a_2\cdots a_n$ and $B{=}b_1b_2\cdots b_m$, if $a_ia_{i+1} = b_jb_{j+1}$ (or $a_ia_{i+1}{=}b_{j+1}b_j$), where $a_ia_{i+1} \in P_A$ and $b_jb_{j+1} \in P_B$, we say that a_ia_{i+1} and b_jb_{j+1} are matched to each other. In a maximum matching of pairs in P_A and P_B, a matched pair is called an **adjacency**, and an unmatched pair is called a **breakpoint** in A and B respectively.

It follows from the definition that sequence scaffolds A and B contain the same set of adjacencies but distinct breakpoints. The maximum matched pairs in B (or equally, in A) form the *(common) adjacency set* between A and B, denoted as $a(A, B)$. We use $b_A(A, B)$ and $b_B(A, B)$ to denote the set of breakpoints in A and B respectively. We illustrate the above definitions in Fig. 1.

For a sequence A and a multi-set of elements X, let $A + X$ be the set of all possible resulting sequences after filling all the elements in X into A. We define a contig as a string

$$
\begin{aligned}
sequence\ scaffold\ A &= \langle c\ b\ c\ e\ d\ a\ b\ a\ \rangle \\
sequence\ scaffold\ B &= \langle a\ b\ a\ b\ d\ c\rangle \\
P_A &= \{cb, bc, ce, ed, da, ab, ba\} \\
P_B &= \{ab, ba, ab, bd, dc\} \\
matched\ \ pairs &: (ab \leftrightarrow ba), (ba \leftrightarrow ab) \\
a(A,B) &= \{ab, ba\} \\
b_A(A,B) &= \{cb, bc, ce, ed, da\} \\
b_B(A,B) &= \{ab, bd, dc\}
\end{aligned}
$$

Figure 1 An example for adjacency and breakpoint definitions.

over a gene set Σ whose contents should not be altered. A *scaffold* S is simply a sequence of contigs $\langle C_1, ..., C_m\rangle$. We define $c(S) = c(C_1) \cup \cdots \cup c(C_m)$. Now, we define the problems on scaffolds formally.

▶ **Definition 2.** One-Sided-SF-max.
Input: a complete genome G and a scaffold $S = \langle C_1, C_2, ..., C_m\rangle$ where G and the contig C_i's are over a gene set Σ, a multiset $X = c(G) - c(S) \neq \emptyset$.
Question: Find $S^* \in S + X$ such that $|a(S^*, G)|$ is maximized.

One-Sided-SF-max($\subset$) is exactly the same as One-Sided-SF-max except that only a subset $X' \subset X$ need to be inserted into S. When a gene can appear at most d times in G, the two versions of problems are abbreviated as One-Sided-SF-max(d) and One-Sided-SF-max($\subset, d$) respectively.

We first present a simple reduction from Hamiltonian Path to One-Sided-SF-max.

▶ **Theorem 3.** *The decision version of One-Sided-SF-*max *is NP-complete.*

Proof. It is obvious that the decision version of One-Sided-SF-max is in NP, so we just focus on the reduction from Hamiltonian Path. Given a connected graph $H = (V, E)$, with $V = \{v_1, v_2, \cdots, v_n\}$ and $e_i = (v_{i,1}, v_{i,2})$, for $e_i \in E$, let $e'_i = v_{i,1}v_{i,2}$, for $i = 1..m$. Let $deg(v)$ be the degree of vertex v (assuming $deg(v) > 1$ for all v). G and S are constructed as follows.

$$G = \#e'_1\#e'_2\#\cdots\#e'_m\# \circ \#_2\#_3\#_1^n,$$

and

$$S = \langle C_1, C_2\rangle,$$

with $C_1 = \langle \#_2 v_1^{deg(v_1)-1}\#_1 \cdots v_n^{deg(v_n)-1}\#_1\#\rangle$ and $C_2 = \langle \#^m\#_3\rangle$. Here $\circ$ is a connector and $X = c(G) - c(S) = V$. As there are only three places to insert elements in X back to S, moreover, the only possible adjacencies are between two vertices forming an edge in H and between a vertex and a $\#$, it is obvious that to maximize the number of adjacencies we need to insert the sequence of vertices forming a Hamiltonian Path in between C_1, C_2.

We make the following claim: H has a Hamiltonian path iff n missing genes can be inserted into S to obtain $n+1$ adjacencies. We only show the "only if" part here as the other direction is trivial. If n missing genes can be inserted into S to obtain $n+1$ adjacencies, say they are inserted between C_1 and C_2 as $v'_1v'_2\cdots v'_n$ (where $v'_j = v_i$), then $n-1$ adjacencies must be $v'_jv'_{j+1}$ and the other two are $\#v'_1$ and $v'_n\#$. Then each $v'_jv'_{j+1}$ corresponds to an

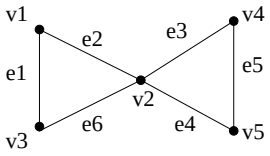

Figure 2 A simple graph H for the reduction.

edge in H and $v'_1v'_2\cdots v'_n$ corresponds to a Hamiltonian path in H. It is obvious that this reduction take $O(n^2)$ time. ◀

We show a simple example for the reduction. The graph H is given in Fig. 2. We have

$$G = \#v_1v_3\#v_1v_2\#v_2v_4\#v_2v_5\#v_4v_5\#v_2v_3\#\#_2\#_3\#_1\#_1\#_1\#_1\#_1,$$

$$S = \langle \boxed{\#_2v_1\#_1v_2v_2v_2\#_1v_3\#_1v_4\#_1v_5\#_1\#} \rangle, \langle \boxed{\#\#\#\#\#\#\#_3} \rangle.$$

After inserting genes in V into S, we obtain

$$S^* = \boxed{\#_2v_1\#_1v_2v_2v_2\#_1v_3\#_1v_4\#_1v_5\#_1\#} v_1v_3v_2v_4v_5 \boxed{\#\#\#\#\#\#\#_3}.$$

It is easy to verify that we have $n+1 = 6$ common adjacencies between G and S^*: $\#v_1$, v_1v_3, v_3v_2, v_2v_4, v_4v_5 and $v_5\#$.

We note that the reduction for the unbounded case SF-MNSA (from X3C in [21, 22]) in fact also works for One-Sided-SF-max – just making each letter in I a contig. (Of course, this would make the contigs too artificial.) But it is obvious that the above proof is simpler and more straightforward. We next present an approximation algorithm for One-sided-SF-max.

3 An Approximation Algorithm for One-Sided-SF-max

Before presenting our algorithm, we make the following definitions.

Let α_i, β_i be the first and last letter of $C_i, i = 1..m$, respectively. Then $\langle \beta_i, \alpha_{i+1} \rangle$ constitutes a region where missing genes can inserted between β_i and α_{i+1}, for $i = 1..m$. Here, we also have two open regions on the two ends of S. We denote them as $\langle -\infty, \alpha_1 \rangle$ and $\langle \beta_m, +\infty \rangle$ respectively.

We define a type-1 substring s of length $\ell \geq 1$, over X, as one which can be inserted in $\langle \beta_i, \alpha_{i+1} \rangle$, for $1 \leq i \leq m-1$, to generate $\ell + 1$ new common adjacencies. We call $\langle \beta_i, \alpha_{i+1} \rangle$ a type-1 *slot* for s. (Throughout this paper, once a type-1 slot is inserted with a corresponding substring we do not allow the insertion of any other letter.) It is easy to see that we could have at most $m-1$ type-1 slots.

Then, we define a type-2 substring s of length $\ell \geq 1$, over X, as one which can be inserted in $\langle \beta_i, \alpha_{i+1} \rangle$, for $0 \leq i \leq m$, to generate ℓ common adjacencies. (We write $\beta_0 = -\infty$ and $\alpha_{m+1} = +\infty$. Clearly the two open slots can be type-2 or type-3.) Note that in this case, in $\langle \beta_i, \alpha_{i+1} \rangle$, we could have two type-2 slots, i.e., right after β_i (written as $\beta_i\circ$) or right before α_{i+1} (written as $\circ\alpha_{i+1}$). By definition, for a fixed $\langle \beta_i, \alpha_{i+1} \rangle$, it cannot be type-1 and type-2 at the same time. It is easy to see that we could have at most $2(m-1)+2 = 2m$ type-2 slots.

Note that if $\beta_i\alpha_{i+1}$ is already a common adjacency with respect to G, then it is possible that s is inserted in the slot to generate $|s|+1$ common adjacencies (while destroying the common adjacency $\beta_i\alpha_{i+1}$). In this case, s really increases the total number of common adjacencies by $|s|$. Hence, s is considered as type-2. For convenience, we simply say that

in this case s generates $|s|$ new common adjacencies. In fact, with a simple example we could show that such an existing adjacency in a slot must be destroyed to obtain an optimal solution. Example: $G = \langle 1, 1, 5, 4, 3, 5, 3, 7, 7\rangle$, $S = \langle \boxed{1,7,3,5}, \boxed{3,1,5,7}\rangle$, the missing gene 4 must be inserted between $\boxed{1,7,3,5}, \boxed{3,1,5,7}$ to obtain the optimal solution.

Finally, we define a type-3 substring s of length $\ell \geq 1$, over X, as one which can be inserted in the slot $\langle \beta_i, \alpha_{i+1}\rangle$, for some i, to generate $\ell - 1$ common adjacencies. Note that a type-3 substring can only form adjacencies internally, hence it does not matter where we insert s – provided that it does not destroy any existing adjacencies.

We show an example as follows:

$G = \langle 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6\rangle$,

$S = \langle \boxed{1,5}, \boxed{3,6}, \boxed{2,4}\rangle$.

We have $\alpha_1 = 1, \beta_1 = 5$, $\alpha_2 = 3, \beta_2 = 6$, $\alpha_3 = 2, \beta_3 = 4$. Then, $X = \{1, 2, 3, 4, 5, 6\}$ are missing from S. One of the optimal solution is

$S' = \langle 1, 2, \boxed{1,5}, 6, \boxed{3,6}, 5, 4, 3, \boxed{2,4}\rangle$.

In this case, $\langle 5, 4, 3\rangle$ is type-1, 6 and $\langle 1, 2\rangle$ are type-2.

We comment that in general a type-j substring, $j = 1, 2, 3$, does not have to be a substring of G. If a type-j substring is composed of i letters, we call it an i-type-j substring.

Let the number of common adjacencies between G and S be k_0, and the number of newly increased common adjacencies be k_1 (after all genes in X have been inserted into S). To approximate $k_0 + k_1$, it suffices to approximate k_1. This is because if we have an approximation solution A_1 for k_1, i.e., $|A_1| \geq k_1/\rho$, then $k_0 + |A_1| \geq (k_0 + k_1)/\rho$ (for $\rho > 1$). From now on, we will only discuss the approximation for the newly increased common adjacencies.

Our **Algorithm 1** is a simple greedy one:

1. Scan through all slots, if an 1-string (i.e., a letter) x or a 2-string xy in X can be inserted in such a slot t to obtain two adjacencies or three adjacencies, insert x or xy into t, lock t. Update $X \leftarrow X - \{x\}$ or $X \leftarrow X - \{x, y\}$ accordingly.
2. For all the remaining (type-2) slots, if a letter $x \in X$ could be inserted to obtain one adjacency, then insert x into the slot and update the the slot as follows. If x is inserted at the slot $y\circ$ (resp. $\circ y$) then update the slot as $x\circ$ (resp. $\circ x$).
3. For all the letters in X after Step 1 (including those already inserted at Step 2), compute a multigraph Q with the vertices being these letters in X (after Step 1), and if xy is a potential adjacency in G (ignoring those already matched with the ones computed at Step 1 and 2), then there is an edge between all $x \in X$ and all $y \in X$. Compute a maximum matching M in Q. For all the pairs xy in M with one end x being a letter inserted at Step 3, insert y before or after x accordingly. For the remaining pairs in M, insert them arbitrarily in any unlocked slot in S, provided no existing adjacency is destroyed.
4. Insert the remaining letters in X arbitrarily in any unlocked slot in S, provided no existing adjacency is destroyed.

Let b_{ij} denote the number of j-type-i substrings in some optimal solution. Then the optimal solution value

$$Opt = \sum_{j=1..p} (j+1)b_{1j} + \sum_{j=1..q} jb_{2j} + \sum_{j=2..r} (j-1)b_{3j},$$

for some p, q, r. Let b'_{ij} denote the number of j-type-i substrings in the approximation solution. We show the properties of the greedy algorithm as follows.

▶ **Lemma 4.** *After Step 1,* $2b'_{11} + 3b'_{12} \geq \frac{1}{2}(2b_{11} + 3b_{12})$.

Proof. By the greedy choice, we have $b'_{11} + b'_{12} \geq b_{11} + b_{12}$. Then,

$$\begin{aligned} 2b'_{11} + 3b'_{12} &\geq 2b'_{11} + 2b'_{12} \\ &\geq 2b_{11} + 2b_{12} \\ &= \frac{1}{2}(4b_{11} + 4b_{12}) \\ &\geq \frac{1}{2}(2b_{11} + 3b_{12}). \end{aligned}$$

◀

▶ **Lemma 5.** *After Step 2,* $b'_{21} \geq b_{21}$.

Proof. If a slot t could be either inserted with an i-type-1 substring s_i for $i = 1, 2$, then a 1-type-2 substring (letter) x could not be inserted at the slot t in an optimal solution. The reason is as follows. (1) Suppose that t can be inserted with an 1-type-1 substring s_1. If t in the optimal solution is inserted with x to generate one adjacency, then we could swap x with s_1 to generate at least two adjacencies. This contradicts with the optimality of the assumed optimal solution. (2) Suppose that t can be inserted with an 2-type-1 substring s_2. If t in the optimal solution is inserted with x to generate one adjacency, then, again, we could swap x with s_2 to generate at least three adjacencies. This implies that there is an optimal solution where all 2-type-1 substrings are always inserted before any 1-type-2 substring is processed.

Then following the greedy choice at Step 2, we have $b'_{21} \geq b_{21}$. ◀

Hence, we could have the following theorem.

▶ **Theorem 6.** *One-Sided-SF-*max *can be approximated within a factor of 2.*

Proof. By definition, the optimal solution value OPT satisfies

$$Opt = \sum_{j=1..p} (j+1)b_{1j} + \sum_{j=1..q} jb_{2j} + \sum_{j=2..r} (j-1)b_{3j},$$

for some p, q, r. At Step 3, the size of the maximum matching, $|M|$, satisfies

$$|M| \geq \frac{1}{2}\left(\sum_{j=3..p} (j+1)b_{1j} + \sum_{j=2..q} jb_{2j} + \sum_{j=2..r} (j-1)b_{3j}\right).$$

The right-hand side of the above inequality represents the optimal internal adjacencies among the corresponding type-1, type-2, and type-3 substrings in the optimal solution. The approximation solution value, *App*, satisfies

$$\begin{aligned} App &= (2b'_{11} + 3b'_{12}) + b'_{21} + |M| \\ &\geq \frac{1}{2}(2b_{11} + 3b_{12}) + b'_{21} + |M| \quad \text{(by Lemma 4)} \\ &\geq \frac{1}{2}(2b_{11} + 3b_{12}) + b_{21} + |M| \quad \text{(by Lemma 5)} \\ &\geq \frac{1}{2}Opt. \end{aligned}$$

◀

4 An FPT Algorithm for One-Sided-SF-max(d)

In this section, we present an FPT algorithm for One-Sided-SF-max(d), parameterized by the optimal number of common adjacencies k. Whether One-Sided-SF-max is FPT is still open, but One-Sided-SF-max(d) represents the important practical version where each gene appears in a genome at most d times. We first review *Fixed-Parameter Tractable* (FPT) algorithms.

4.1 Definition of FPT Algorithms

Let Σ be the alphabet, and $Q \subseteq \Sigma^*$ be a classic decision problem. A parameterized problem is a pair (Q, κ) where $\kappa : \Sigma^* \to \mathbb{N}$ is a polynomial computable function. An instance of (Q, κ) is a pair $(x, \kappa(x))$ consisting of a string $x \in \Sigma^*$ and an integer $\kappa(x)$.

▶ **Definition 7.** Let (Q, κ) be a parameterized problem. We say (Q, κ) is Fixed-Parameter Tractable (FPT) if for each instance $(x, \kappa(x))$, there is an algorithm A which decides whether $x \in Q$ in $f(\kappa(x)) \cdot |x|^c$ time, where f is an arbitrary computable function and c is a constant.

As a convention now, we write $O(f(\kappa(x))n^c) = O^*(f(k))$. FPT algorithms are efficient tools for handling some NP-complete problems, especially when $k = \kappa(x)$ is small in some practical datasets [16, 18, 29].

4.2 The FPT Algorithm

We now present an FPT algorithm for One-Sided-SF-max(d), parameterized by the optimal number of common adjacencies k. (Here k includes the existing number of common adjacencies between S and G, though it is obvious that our algorithm also works by looking at newly created common adjacencies.) As the running time of the algorithm is high and the result is mostly for theoretical purpose.

Our idea is as follows. We use the color-coding method to find a potential ℓ-type-i substring for $i = 1, 2$. Then we use the property that each gene appears at most d times to search for a slot to put this string in a right slot. After this process are repeated for all potential type-1 and type-2 substrings, type-3 substrings can then be inserted arbitrarily, as long as they do not destroy the existing adjacencies.

Note that a 1-type-3 substring cannot contribute any common adjacency with respect to G, so it is *useless*. All other inserted letters are *useful*. We first show the following lemma regarding the number of useful letters in an optimal solution.

▶ **Lemma 8.** *Let $X^* \subseteq X$ be the set of genes in X that contribute in generating some new common adjacencies. If the optimal number of common adjacencies between G and S^* is k, then $|X^*| \leq 2k$.*

Proof. From the previous discussions, a ℓ-type-1 substring creates $\ell + 1$ common adjacencies, a ℓ-type-2 substring creates ℓ common adjacencies, and a ℓ-type-3 substring creates $\ell - 1$ common adjacencies. Hence, in the worst case, the k common adjacencies are created by $2k$ type-3 substrings, each of length 2 (creating one common adjacency). In this case, these genes form the set of optimal active genes X^*, with $|X^*| \leq 2k$. ◀

We then make use of the color-coding method [3, 4], summarized as the following lemma. For a positive integer n, let $[n] = \{1, 2, ..., n\}$.

▶ **Lemma 9** ([3, 4]). *Let $1 \leq \ell \leq k$. For every n, ℓ there is a family $\Delta_{n,\ell}$ of polynomial time computable functions from $[n]$ to $[k]$ such that for every ℓ-element subset Y of $[n]$, there is an $h \in \Delta_{n,\ell}$ such that h is injective on Y. Moreover, $\Delta_{n,\ell}$ can be computed in time $2^{O(k)} \cdot n^{O(1)}$.*

The following lemma is similar to that for solving the k-path problem using color-coding [3, 4].

▶ **Lemma 10.** *Given a fixed slot, a p-type-j substring, $j = 1, 2$, can be computed in FPT time.*

Proof. A p-type-j substring is formed by the 2-substrings (or, at most $n-1$ possible adjacencies) in G. We use the color-coding technique. For the ease of description, we focus on $j = 1$. We give each 2-substring in G one of the $p+1$ random colors. A p-type-1 substring for a given slot is determined by $p+1$ 2-substrings in G. The probability that we could find such a colorful p-type-1 substring is at least

$$\frac{(p+1)!}{(p+1)^{p+1}} = \frac{\sqrt{2\pi(p+1)}}{e^{p+1}} > (\frac{1}{e})^{p+1},$$

where $p! \sim \sqrt{2\pi p}(\frac{p}{e})^p$, following Stirling's formula. To guarantee that we could obtain a valid solution, we simply run this algorithm e^{p+1} times. This process can be derandomized with standard techniques [16, 18, 4]. The total running time of this algorithm is then bounded by $O^*(e^{p+1})$. For constructing the corresponding p-type-2 and p-type-3 substrings (over the unused/unmatched 2-substrings in G), the running times are $O^*(e^p)$ and $O^*(e^{p-1})$ respectively. Note that type-3 substrings are not relevant to any specific slot. ◀

▶ **Theorem 11.** *One-Sided-SF-*max*(d) is FPT.*

Proof. The general idea is a combination of bounded-degree search and color-coding. Following Lemma 9 and 10, the algorithm generates a proper p-type-j substring s, where $p \leq k-1, j = 1$ or $p \leq k, j = 2$, for a potential slot $\langle \beta_i, \alpha_{i+1} \rangle$. As β_i and α_{i+1} can each appear d times, we could have $2d$ possible slots to put s. We then delete the letters in s from X and repeat the process until no type-1 or type-2 substring can be inserted in S. If the number of common adjacencies is at least k, we stop and insert the remaining letters in X arbitrarily, not to destroy any existing adjacency. If the number of common adjacencies is still less than k, we use Lemma 10 to generate some p-type-3 substring and insert it arbitrarily into S (not to destroy any existing adjacency). By Lemma 8, the search stops when a total of at most $2k$ useful letters have been inserted. (The remaining letters can be inserted arbitrarily, provided that they do not destroy any existing common adjacency). We can then check and report a solution with at least k common adjacencies, or report that such a solution does not exist.

The total running time of this algorithm is

$$O^*((2d \cdot e^k)^k) = O^*(2^k d^k e^{2k}).$$

Hence we have the theorem. ◀

In the next section, we discuss the One-sided Subset Scaffold Filling (One-sided-SF-max($\subset$)) problem.

5 Results for One-Sided-SF-max($\subset$)

In this section, we present some results for One-Sided-SF-max($\subset$). We prove that if the parameter is the number of genes inserted, then the problem is W[1]-hard. This implies that the problem cannot be solved with an FPT algorithm, unless FPT=W[1] [16, 18, 29]. We then present a simple FPT-approximation for the problem, with a factor of 2, for One-Sided-SF-max($\subset, d$).

5.1 W[1]-Hardness Result

The main theorem is stated as follows.

▶ **Theorem 12.** *One-Sided-SF-*max($\subset$) *parameterized by the number of genes inserted is W[1]-hard.*

Proof. Throughout this proof, assume that $k \leq (n-1)/2$. We show that Independent Set can be reduced to One-Sided-SF-max($\subset$) via a linear FPT reduction. Given a graph $Q = (V, E)$, if the maximum vertex degree is Δ, then for each vertex $u_i \in V$ with degree $deg(u_i) < \Delta$, we create $\Delta - deg(u_i)$ new nodes and connect them only to u_i. In the resulting graph $Q' = (V', E')$, all the original vertices in V have degree Δ. It can be easily seen that Q has an independent set of size k iff k vertices in Q' can be selected to cover exactly $k\Delta$ edges. This part of the proof is adapted from [20].

Now we arrange the graph $Q' = (V', E')$ as a genome G as follows. WLOG, still assume that $|V'| = n, |E'| = m$ throughout this proof. For each $v_i \in V'$, construct E_i as the list of edges incident to v_i (ordered by their indices). Then we use separators #'s and $\#_j$, for $j = 1..5$. The set of genes are $\{e_i | i = 1..m\} \cup \{\#_j | j = 1..5\} \cup \{\#\}$. Finally we arrange G as follows.

$$G = \#^{m+1} \circ \#_1\#_2\#_3\#_4 \circ \#_4\#_3\#_2\#_1 \circ \#_5 E_1 \#_5 E_2 \#_5 \cdots \#_5 E_n \#_5.$$

Note that $\circ$ is used as a connector, each e_i $(i = 1..m)$, $\#_j$ $(j = 1..4)$ appears twice in G, # appears $m+1$ times and $\#_5$ appears $n+1$ times in G. S is constructed such that it is composed of exactly $k+1$ contigs $C_1, ..., C_{k+1}$, each C_i starts and ends with $\#_5$. For C_1, between the two $\#_5$'s, we arrange all the genes #'s and e_i's such that $C_1 = \#_5\#e_1\#e_2\#\cdots\#e_m\#\#_5$. We construct $C_2 = \#_5\#_3\#_1\#_4\#_2 \circ \#_5^{n-2k-1} \circ \#_2\#_4\#_1\#_3\#_5$. The remaining contigs are constructed as $C_i = \#_5\#_5$ for $i = 3, ..., k+1$.

It is clear that in S we have missed a copy of e_i for each $i = 1..m$. Due to the construction of G, e_i cannot form any common adjacency with # or $\#_j$ for $j = 1..4$, the only possible common adjacencies are from e_i and e_ℓ's (i.e., in some sequences of E_p's, each of length Δ) and between e_i and $\#_5$'s. To maximize the common adjacencies obtained, these missing genes can only be inserted in k slots, after C_i and before C_{i+1} for $i = 1..k$. Then, it is safe for us to claim, with some easy details omitted, that Q has an independent set of size k iff $k\Delta$ missing genes can be inserted into the k slots in S to obtain a maximum of $k(\Delta+1)$ adjacencies with respect to the reference genome G. This is obviously an FPT-reduction. ◀

With the above W[1]-hardness result, it is easy to obtain the following corollary (part of it is similar to the corollary in [27]).

▶ **Corollary 13.** *The optimization version of One-Sided-SF-*max($\subset$) *does not admit an EPTAS (resp. FPTAS) unless FPT=W[1].*

Proof. Assume that there is an EPTAS (resp. FPTAS) which runs in time $O((\frac{1}{\epsilon})^{O(\frac{1}{\epsilon})}n^c)$ (resp. $O((\frac{1}{\epsilon})^{c_1}n^{c_2})$), for some constant c (resp. c_1 and c_2); moreover, it achieves an approximation factor of $1+\epsilon$, for any $\epsilon > 0$. Then, if k^* is the optimal solution value and APP is the approximation solution value, we have

$$APP \geq \frac{k^*}{1+\epsilon}.$$

Setting $\epsilon = \frac{1}{2k^*-1}$, we have $APP \geq \frac{k^*}{1+\epsilon} = k^* - \frac{1}{2}$, which further implies $APP = k^*$. In this case, the running time of the algorithm becomes $O((k^*)^{O(k^*)}n^c)$ (resp. $O((k^*)^{O(c_1)}n^{c_2})$); i.e., the problem would admit an FPT algorithm. A contradiction to Theorem 12, unless FPT=W[1]. ◀

5.2 FPT-Approximation for One-Sided-SF-max($\subset$,d)

For W[1]-hard problems, a natural way to handle them is to use FPT-approximations. Here we briefly review the Fixed-Parameter Tractable Approximation Algorithm (FPT-approximation for short), which was first proposed in 2006 [10, 17, 8] (but the development has been slow.)

▶ **Definition 14.** A Fixed-Parameter Tractable ρ-approximation for a minimization (resp. maximization) parameterized problem (Q, κ) is an FPT algorithm which, given any instance $(x, k) \in (Q, \kappa)$, returns a solution of cost at most $\rho(k) \cdot k$ (resp. at least $k/\rho(k)$) if a solution of cost at most (resp. at least) k exists.

Our FPT-approximation algorithm for One-Sided-SF-max$(\subset, d)$, parameterized by the number of inserted genes, is as follows.

1. As in Theorem 11, use bounded-degree search and color-coding to insert ℓ $(0 \leq \ell \leq k)$ type-1 and type-2 substrings into the ℓ slots, which can be done in FPT time.
2. If these ℓ substrings have a total length at least k, then the problem can be solved optimally in FPT time.
3. If these ℓ substrings have a total length k_1 with $k_1 < k$, then we insert enough type-3 substrings (of a total length $k - k_1$) as follows.
4. We use a maximum matching method to insert $k - k_1$ letters. For all the remaining genes to be inserted into G, form a graph D such that there is an edge connecting two such genes if they could potentially form a common adjacency with respect to G. Then simply compute a maximum matching in D and insert all the pairs in the matching arbitrarily into D (provided that they do not destroy any existing common adjacency).

▶ **Theorem 15.** *One-Sided-SF-*max$(\subset, d)$ *parameterized by the number of genes inserted admits a factor-2 FPT-approximation.*

Proof. The analysis of the first two steps of the FPT algorithm is the same as in Theorem 11, hence omitted.

Let k_1 letters inserted at Step 1 generate k_1^* common adjacencies. The $k - k_1$ genes forming type-3 substrings could generate at most $k_2 \leq k - k_1 - 1$ common adjacencies. By the maximum matching algorithm at step 4, we could generate at least $k_2/2$ common adjacencies. (For any connected component in D, if it contains a path of length $k_3 \leq k_2$ then the maximum matching algorithm could return at least $k_3/2$ common adjacencies.) Then

$$OPT = k_1^* + k_2,$$

and

$$APP \geq k_1^* + k_2/2 \geq \frac{OPT}{2}.$$

The whole algorithm obviously takes FPT time. ◀

6 Concluding Remarks

In this paper, we revisit the genomic scaffold filling problem by considering each scaffold as a sequence of contigs (instead of as an incomplete sequence as in most of the previous research). We obtain a list of algorithmic results, some of which could eventually lead to the practical processing of genomic datasets. However, as in [7], the parameter k (i.e., number of common adjacencies) in reality should be relatively large, so the FPT algorithms we obtained here are only theoretically meaningful. Further research is needed along this line. On the other hand, theoretically, it is interesting to decide whether One-Sided-SF-max is FPT and whether One-Sided-SF-max($\subset$) admits an FPT-approximation.

Acknowledgments This research is partially supported by the Open Fund of Top Key Discipline of Computer Software and Theory in Zhejiang Provincial Colleges at Zhejiang Normal University. We also thank anonymous reviewers for several useful comments.

References

1 S. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman. Basic local alignment search tool. *J. Molecular Biology*, 215(3):403-410, 1990.

2 S. Angibaud, G. Fertin, I. Rusu, A. Thevenin and S. Vialette. On the approximability of comparing genomes with duplicates. *J. Graph Algorithms and Applications*, 13(1):19-53, 2009.

3 N. Alon, R. Yuster and U. Zwick. Color-coding. *J. ACM*, 42(4):844-856, 1995.

4 N. Alon, R. Yuster and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209-218, 1997.

5 G. Blin, G. Fertin, F. Sikora and S. Vialette. The exemplar breakpoint distance for non-trivial genomes cannot be approximated. *Proc. 3nd Workshop on Algorithm and Computation (WALCOM'2009)*, LNCS 5431, pp. 357-368, 2009.

6 H. Bodlaender, R. Downey, M. Fellows and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423-434, 2009.

7 L. Bulteau, A.P. Carrieri and R. Dondi. Fixed-parameter algorithms for scaffold filling. *Theoretical Computer Science*, 568: 72–83, 2015.

8 L. Cai and X. Huang. Fixed-parameter approximation: conceptual framework and approximability results. *Algorithmica*, 57(2):398-412, 2010.

9 P.S. Chain, D.V. Grafham, R.S. Fulton, *et al.* Genome project standards in a new era of sequencing. *Science*, 326:236-237, 2009.

10 Y. Chen, M. Grohe and M. Grueber. On parameterized approximability. *Proc. 2nd Intl. Workshop on Parameterized and Exact Computation (IWPEC'06)*, LNCS 4169, pp. 109-120, 2006.

11 Z. Chen, B. Fu and B. Zhu. The approximability of the exemplar breakpoint distance problem. *Proc. 2nd Intl. Conf. on Algorithmic Aspects in Information and Management (AAIM'06)*, LNCS 4041, pp. 291-302, 2006.

12 Z. Chen, B. Fu, B. Yang, J. Xu, Z. Zhao, and B. Zhu. Non-breaking similarity of genomes with gene repetitions. In *Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, LNCS 4580, pp. 119–130, 2007.

13 Z. Chen, B. Fu, R. Fowler and B. Zhu. On the inapproximability of the exemplar conserved interval distance problem of genomes. *J. Combinatorial Optimization*, **15**(2):201-221, 2008.

14 Z. Chen, B. Fu, R. Goebel, G. Lin, W. Tong, J. Xu, B. Yang, Z. Zhao and B. Zhu. On the approximability of the exemplar adjacency number problem of genomes with gene repetitions. *Theoretical Computer Science*, **550**:59-65, 2014.

15 G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA'02)*, pp. 667-676, 2002.

16 R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag. 1999.

17 R. Downey, M. Fellows, C. McCartin and F. Rosamond. Parameterized approximation of dominating set problems. *Info. Process. Lett.*, 109(1): 68–70, 2008.

18 J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer-Verlag. 2006.

19 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman. 1979.

20 J. Guo, R. Niedermeier and S. Wernicke. Parameterized complexity of vertex cover variants. *Theory Comput. Syst.*, 41(3):501-520. 2007.

21 H. Jiang, F. Zhong and B. Zhu. Filling scaffolds with gene repetitions: maximizing the number of adjacencies. *Proc. 22nd Annual Combinatorial Pattern Matching Symposium (CPM'11)*, LNCS 6661, pp. 55-64, Palermo, Italy, June 27-29, 2011.

22 H. Jiang, C. Zheng, D. Sankoff, and B. Zhu. Scaffold filling under the breakpoint and related distances. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1220-1229, July/August, 2012.

23 H. Jiang, J. Ma, J. Luan and D. Zhu. *Approximation and nonapproximability for the one-sided scaffold filling problem. Proc. 21st Intl. Ann. Comput. and Combinatorics (COCOON'15),* LNCS 9198, pp. 251-263, 2015,

24 M. Jiang. The zero exemplar distance problem. *Proc. of the 2010 International RECOMB-CG Workshop (RECOMB-CG'10)*, LNBI 6398, pp. 74-82, 2010.

25 N. Liu, H. Jiang, D. Zhu, and B. Zhu. An improved approximation algorithm for scaffold filling to maximize the common adjacencies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(4):905-913, July/August, 2013.

26 N. Liu, D. Zhu, H. Jiang and B. Zhu. A 1.5-approximation algorithm for two-sided scaffold filling. *Algorithmica*, 74(1):91-116, 2016.

27 D. Marx. Parameterized complexity and approximation algorithms. *Computer Journal*, 51(1):60-78, 2008.

28 A. Muñoz, C. Zheng, Q. Zhu, V. Albert, S. Rounsley and D. Sankoff. Scaffold filling, contig fusion and gene order comparison. *BMC Bioinformatics*, 11:304, 2010.

29 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford Univ. Press. 2006.

30 S. Yancopoulos, O. Attie and R. Friedberg. *Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics*, **21**:3340-3346, 2005.

31 B. Zhu. A retrospective on genomic preprocessing for comparative genomics. In Chauve et al., eds., *Models and Algorithms for Genome Evolution*, pages 183-206. Springer, 2013.