Journal of Bioinformatics and Computational Biology
© Imperial College Press

A 2-APPROXIMATION ALGORITHM FOR THE CONTIG-BASED GENOMIC SCAFFOLD FILLING PROBLEM

Haitao Jiang
School of Computer Science and Technology, Shandong University
Jinan, Shandong, China
Email: htjiang@sdu.edu.cn.

Letu Qingge
Gianforte School of Computing, Montana State University
Bozeman, MT 59717, USA
Email: letu.qingge@msu.montana.edu

Daming Zhu
School of Computer Science and Technology, Shandong University
Jinan, Shandong, China
Email: dmzhu@sdu.edu.cn.

Binhai Zhu
Gianforte School of Computing, Montana State University
Bozeman, MT 59717, USA
Email: bhz@montana.edu

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

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 common/shared adjacencies between G and I' is maximized. The problem is NP-complete, and admits a constant-factor 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 a scaffold S is given, the missing genes can only be inserted in between the contigs, and the objective is to maximize the number of common adjacencies between G and the filled genome S'. For this problem, we present a simple NP-completeness proof, we then present a factor-2 approximation algorithm.

Keywords: Comparative genomics; Scaffold filling; Contigs; Adjacencies; NP-completeness; Approximation algorithms

1. Introduction

Due to the advancement in genome sequencing technology, the cost for sequencing a genome has been reduced significantly in the last decade. Currently, the cost for sequencing a (human) genome is only around \$1k. With this low cost, a lot of genomes have been sequenced, although usually not completely finished. (We typically call these incomplete genomes *draft* genomes). There has been tools to fill the gaps for draft genomes [3,25], but it is not always guaranteed that one could fill all the gaps for a given draft genome. In fact, the cost for making draft genomes complete has not been reduced a lot compared with more than a decade ago [5]. The consequence is that there are more and more draft genomes. On the other hand, to analyze the genomic data a lot of the corresponding tools do need complete genomes as input. For example, we need two complete genomes to compute their corresponding minimum reversal distance. Therefore, it is necessary to make a draft genome complete.

To make the resulting genome biologically meaningful, Munoz *et al.* first studied the *scaffold filling* problem (on multichromosomal genomes with no gene repetition) which we define, following [24], as follows. Given a complete genome R (represented as a permutation) and an incomplete scaffold S (represented as a list of contigs), fill the missing genes in $R - S$ into S to obtain S' such that the genomic (or DCJ) distance [26] between R and S' is minimized. They showed that this problem is polynomially solvable. Later, Jiang *et al.* showed that the two-sided case, where each scaffold serves as a reference for the other, is also polynomially solvable [18]. In [18], Jiang *et al.* also studied the case for singleton permutation genomes (i.e., with no gene repetition in the genomes), using the simplest *breakpoint* distance as the distance measure. It was shown that that this problem is also polynomially solvable; in fact, even for the two-sided case when both of the input scaffolds are incomplete permutations [18]. (Again, in this case, each of the two scaffolds is a reference to the other.)

When gene repetitions are allowed in the genomes and scaffolds, and when the missing genes can be inserted freely in the scaffolds (usually implying the poor quality of scaffolds), the problem becomes much harder. (That should not surprise the readers as even computing several similarity measures between two complete genomes with the same gene content is NP-complete, for example, under the exemplar breakpoint distance [6,8,1,2,19], under the exemplar adjacency number [7,9], or under the minimum common string partition [10].) For this case, the favorable similarity measure for the genomic scaffold filling problem has been the *number of common (string) adjacencies*, which can be easily computed in polynomial time [1,17,18]. Jiang *et al.* showed that genomic scaffold filling with the objective being maximizing the number of common string adjacencies (SF-MNSA) is NP-hard [17,18]. (The problem can be formally defined as: filling an incomplete sequence scaffold I into I', using a complete genome G as a reference, such that the missing letters in $G - I$ can be freely inserted into I and the resulting number of common adjacencies between G

and I' is maximized.) A 1.33-approximation algorithm was designed in [17,18], and this bound has recently been improved to 1.25 [20]. For the corresponding two-sided case, i.e., when each of the two scaffolds serves as a reference to the other, the problem can be approximated with a factor of 1.5 (using the same similarity measure) [21] and 1.4 [23]. When the number of common adjacencies is used as a parameter, this problem has been shown to be fixed-parameter tractable (FPT) [4]. Of course, this FPT algorithm can only handle the case when the number of common adjacencies between the two genomes G and I' is a small parameter, i.e., when the input genomes are not quite similar, hence the result is only of theoretical meaning.

We now depict the motivation of this paper as follow. When gene repetitions are allowed, a 'scaffold' in consideration is simply an incomplete sequence, i.e., a missing gene can be inserted freely into such a 'scaffold'. In comparative genomics, most of the real datasets are not under this format. Most of the time, a scaffold in a real genomic dataset is composed of a list of contigs, which have been researched a lot in the past [15,14]. Nowadays, a contig is usually computed with some mature tools like Celera (`http://www.celera.com`) and Spades (`http://cab.spbu.ru/software/spades/`), etc, hence should be of a decent quality and should not be arbitrarily changed. This case was considered in [24,18,22], but never in a systematic way. (In fact, all other research on scaffold filling simply used an incomplete sequence as a scaffold.) For a recent survey on the genomic scaffold filling research, the readers are referred to [28].

The main contribution of this paper is on filling a scaffold composed of a set of contigs, with a reference genome. We formally define the problem as One-sided Scaffold Filling (One-sided-SF-max), where the objective is to maximize the number of common adjacencies between the (complete) reference genome 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, and we then present a factor-2 approximation. We comment that the analysis of the 2-approximation covered in [16] is incomplete, so this paper can be considered as a fix-up for that part. (In [16], there are more FPT results which we will not cover here.)

The paper is organized as follows. In Section 2, we give the basic definitions. In Section 3, we present the approximation results for One-sided-SF-max. We conclude the paper in Section 4.

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 [18,27]. We assume that all genes and genomes are unsigned. 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 we use $c(A)$ to denote the set of genes in A, which is a multi-set of elements in Σ. For example, $\Sigma = \{a,\ b,\ c,\ d\}$, A = *abcdacd*, $c(A) = \{a,\ a,\ b,\ c,\ c,\ d,\ d\}$. Let G be a (complete) reference geonome with the corresponding gene set $c(G)$, which is a multiset over Σ. With respect to G, a *sequence scaffold* A is an incomplete sequence such that $c(A) \subset c(G)$. A is typically obtained by some sequencing and assembling process. A substring with m genes (in a sequence A) 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 (more formally, an edge is created between these two pairs, each representing a vertex, in a bipartite graph $\mathcal{B}$). In a maximum matching of $\mathcal{B}$, the vertex of a matching edge or a matched pair in P_A and P_B, is called an **adjacency**, and an unmatched vertex (or 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.

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

Fig. 1. An example for adjacency and breakpoint definitions. Note that there are three pairs ab (or equivalently, ba) in P_B, but there are only two in P_A, hence one of them in P_B is not matched in the maximum matching for the corresponding bipartite graph $\mathcal{B}$.

For any two sequences Z_1, Z_2, the longest common subsequence between them is represented as $lcs(Z_1, Z_2)$. 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, i.e., for any $Z \in A + X$, we have $lcs(Z, A) = A$ and moreover, $c(Z) - c(A) = X$. We define a contig as a string 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 $c(S) = c(C_1) \cup \cdots \cup c(C_m)$. Let $\tilde{S}$ be the sequence $\langle c_1, ..., c_m\rangle$ where each contig C_i is compressed into a new letter $c_i \notin \Sigma \cup X$ for $i = 1..m$. Then, $S + X$ is defined as the set which is obtained by taking all sequences in $\tilde{S} + X$ and uncompressing each c_i in $\tilde{S}$ back to C_i for $i = 1..m$. 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.

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

Theorem 1. *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 $i = 1..m$, and 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. WLOG, we assume that $deg(v) > 1$ for all v, as otherwise in the reduction we might have $deg(v) - 1 = 0$, which will introduce some messy degenerate cases. (This assumption will not change the complexity of the Hamiltonian Path problem — a graph admitting a Hamiltonian path contains at most two degree-1 nodes and we could easily replace a degree-1 node v by a triangle $\triangle(vv'v'')$ where v' and v'' are newly introduced nodes.) 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 $\#$ and $\#_j, j = 1..3$ are new letters and $\circ$ 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 $v'_j = v_i$ for some i), then the $n-1$ adjacencies $v'_jv'_{j+1}$ must correspond to an edge in H and the other two are $\#v'_1$ and $v'_n\#$. Then, by definition, the substring $v'_1v'_2\cdots v'_n$ corresponds to a Hamiltonian path in H. It is obvious that this reduction takes $O(n^2)$ time. □

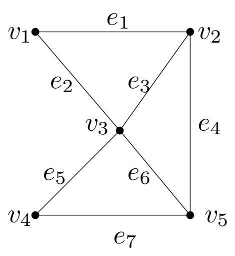

Fig. 2. A simple graph H for the reduction.

We show a simple example for the reduction. The graph H is given in Fig. 2. We have $G = \#v_1v_2\#v_1v_3\#v_2v_3\#v_2v_5\#v_3v_4\#v_3v_5\#v_4v_5\#\#_2\#_3\#_1\#_1\#_1\#_1\#_1$,

$$S = \langle \boxed{\#_2v_1\#_1v_2v_2\#_1v_3v_3v_3\#_1v_4\#_1v_5v_5\#_1\#}, \boxed{\#\#\#\#\#\#\#\#_3} \rangle.$$

After inserting all the genes in V into S, we eventually obtain
$S^* = \boxed{\#_2v_1\#_1v_2v_2\#_1v_3v_3v_3\#_1v_4\#_1v_5v_5\#_1\#}\,v_1v_3v_4v_5v_2\boxed{\#\#\#\#\#\#\#\#_3}$.

It is easy to verify that we have $n+1 = 6$ common adjacencies between G and S^*: $\#v_1$, v_1v_3, v_3v_4, v_4v_5, v_5v_2 and $v_2\#$.

We note that the reduction for the unbounded case SF-MNSA (from X3C in [17,18]) 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 approximation algorithm for One-sided-SF-max, we make the following definitions.

Define α_i, β_i as the first and last letter of contig $C_i, i = 1..m$, respectively. Then $\langle \beta_i, \alpha_{i+1}\rangle$ forms a region (or slot) where missing genes can be inserted between β_i and α_{i+1}, for $i = 1..m$. Note that we also have two open regions at the two ends of S. For convenience, 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 a string which can be inserted in slot $\langle \beta_i, \alpha_{i+1}\rangle$, for some $1 \leq i \leq m-1$, 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 and subsequently we lock it right away. (Here, 'locking' means resetting a variable associated with a slot to ON, where initially all such variables are set as OFF.) It is easy to see that we could have at most $m-1$ type-1 slots as there are m contigs and a type-1 slot can only be between two contigs.

Then, we define a type-2 substring s of length $\ell \geq 1$, over X, as a string which can be inserted in slot $\langle \beta_i, \alpha_{i+1} \rangle$, for some $0 \leq i \leq m$, 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, which will be formally defined next.) 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 slot $\langle \beta_i, \alpha_{i+1} \rangle$, 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.

It should be noted that even if $\beta_i\alpha_{i+1}$ is already a common adjacency with respect to the reference genome G, it is still possible that s can be inserted in the slot $\langle \beta_i, \alpha_{i+1} \rangle$ to obtain $|s|+1$ common adjacencies (while the existing common adjacency $\beta_i\alpha_{i+1}$ is destroyed). In this case, s in fact increases the total number of common adjacencies only by $|s|$. Hence, s should be 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: $S^* = \langle \boxed{1,7,3,5}, 4, \boxed{3,1,5,7} \rangle$.

For convenience, for a letter $x \in X$, we say a common adjacency xy or (x,y) is *external* if $y = \alpha_i$ or $y = \beta_i$ for some $i \in [1..m]$; otherwise, xy is *internal*. Finally, we define a type-3 substring s of length $\ell \geq 1$, over X, as a string which can be inserted in slot $\langle \beta_i, \alpha_{i+1} \rangle$, for some i, to generate $\ell - 1$ common adjacencies. Clearly, a type-3 substring s can form nothing but internal adjacencies; hence, as long as s does not destroy any existing adjacencies it does not matter where we insert it.

We show an example as follows:

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

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

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

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

In this case, $\langle 4,3,4 \rangle$ and $\langle 1 \rangle$ are type-1, and $\langle 3,2 \rangle$ is type-2. Among the eight common adjacencies obtained, the two $(2,1)$'s are external, both of $(5,4)$ and $(4,5)$

are external, both of $(4, 3)$ and $(3, 4)$ are internal, $(3, 2)$ is internal, and $(1, 6)$ is external.

We comment that in general a type-j substring, $j = 1, 2, 3$, does not have to be a substring of G. (For example, $\langle 4, 3, 4\rangle$ in the previous example is not a substring of G.) If a type-j substring is composed of i letters, we say that it is a (length-i,type-j) substring.

Let the number of common adjacencies between G and S be k_0. After all genes in X have been inserted into S, let the number of newly increased common adjacencies be k_1. 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 based on greedy search and two levels of maximum matchings, which is given separately as follows. To help readers understand the algorithm better, we first give a rough sketch of the algorithm.

Step 1.	Insert (length-1,type-1) letter greedily.
Step 2.	Insert (length-1,type-2) letters for external adjacencies using bipartite maximum matching.
Step 3.	Insert the remaining letters involved in a maximum matching of a general multigraph for internal adjacencies.
Step 4.	Insert the remaining letters arbitrarily provided that no existing adjacency is destroyed.
Step 5.	Return the filled scaffold.

Fig. 3. A sketch of Algorithm 1.

Then we run the approximation algorithm on the previous example as follows.

(1) We insert 4 between $\boxed{1,5}$ and $\boxed{5,2}$, and insert 1 between $\boxed{5,2}$ and $\boxed{6,6}$. Four common adjacencies are obtained due to the insertion of these type-1 substrings.
(2) With the bipartite maximum matching, we insert 2 before $\boxed{1,5}$ to obtain an external common adjacency (2,1). Here, $\langle 2\rangle$ is type-2.
(3) The multigraph Q is a cycle of edges: (2,3), (3,4), (4,3), and (3,2). With the maximum matching, we insert 3 before 2 (inserted at Step 2).
(4) Finally we insert $\langle 3, 4\rangle$ after $\boxed{6,6}$, as a type-3 substring.
(5) We obtain the final filled scaffold: $S' = \langle 3, 2, \boxed{1,5}, 4, \boxed{5,2}, 1, \boxed{6,6}, 3, 4\rangle$.

There are a total of seven common adjacencies obtained, while the optimal solution value is eight.

Let b_{ij} denote the number of (length-i,type-j) substrings in some optimal solution and let B_{ij} be the corresponding set of (length-i,type-j) substrings. Then the optimal solution value is

$$Opt = \sum_{i=1..p} (i+1)b_{i1} + \sum_{i=1..q} ib_{i2} + \sum_{i=2..r} (i-1)b_{i3},$$

for some p, q, r. Let b'_{ij} denote the number of (length-i,type-j) substrings in the approximation solution and let B'_{ij} be the corresponding set of (length-i,type-j) substrings. We show the properties of the approximation algorithm as follows.

```
Algorithm 1: On input (G, S)
1  Use the greedy method to scan from left to right
   in $S$ for a (length-1,type-1) slot for a letter $x \in X$ to be
   inserted (to generate two new external common adjacencies),
   and insert it accordingly in the respective slot. Lock this slot.
2  Use the bipartite maximum matching to identify all (length-1,type-2)
   substrings (or, external common adjacencies) $xz$ such that
   the letter $x \in X$ and $z = \alpha_j$ or $\beta_j$ for some $j = 1..m$.
   Insert $x$ into the slot and update the the slot as follows.
   If $x$ is inserted at the slot $z\circ$ (resp. $\circ z$) then insert $x$ after $z$
   (resp. insert $x$ before $z$) and update the slot as $x\circ$ (resp. $\circ x$).
3  For all the remaining letters in $X$ after Step 1 (including those
   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
   internal 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 2, insert $y$ before or after $x$ accordingly.
   For the remaining pairs in $M$, insert them arbitrarily in any
   unlocked slot in $S$, provided that no existing adjacency is destroyed.
4  Insert the remaining letters in $X$ arbitrarily in any unlocked
   slot in $S$, provided that no existing adjacency is destroyed.
5  Return the filled scaffold $S'$.
```

Lemma 1. *After Step 1, a misplaced (length-1,type-1) letter c in B'_{11} (compared to B_{11}) could at most change a (length-1,type-1) substring in the optimal solution into type-3 and two type-1 substrings ((length-v,type-1) and (length-w,type-1) respectively) into type-2, with $v, w \geq 1$.*

Proof. All we need to prove is that if a (length-1,type-1) letter was inserted at a wrong slot, then at most three type-1 substrings s_u, s_v and s_w are not type-1 anymore. To see this, assume that in some optimal solution a (length-1,type-1) letter c is inserted in the slot v_1v_2 and three type-1 substrings s_u, s_v, s_w are inserted in the slots z_iz_{i+1} respectively, for $i = 1, 2, 3$. Suppose that c is inserted in one of the slots z_iz_{i+1}, say z_1z_2, as type-1. Then, in the worst case, all s_u, s_v, s_w cannot be type-1 anymore: s_u could be type-3 and s_v, s_w could be type-2.

We could verify this with a generic example. First, assume that in G the letter c is only involved in four adjacent pairs a_1c, a_2c, b_1c and b_2c, and d_1 is only involved in two adjacent pairs b_1d_1 and b_2d_1 (in this example all letters are different unless otherwise specified). On top of this assumption, let $G = \langle \cdots, a_1, c, a_2, \cdots, b_1, d_1, b_2, \cdots, b_3, d_3, b_4, \cdots, b_5, d_5, b_6, \cdots \rangle$, $S = \langle \boxed{\cdots a_1}, \boxed{a_2 \cdots}, \ldots, \boxed{\cdots b_1}, \boxed{b_2 \cdots}, \boxed{\cdots b_3}, \boxed{b_4 \cdots b_5}, \boxed{b_6 \cdots} \rangle$. Let $X = \{c, d_1, d_3, d_5\}$. The optimal solution is to insert c between a_1 and a_2, and d_i's between b_i, b_{i+1}, for $i = 1, 3, 5$. If c is inserted between b_1 and b_2, then all d_i's cannot be type-1 anymore. This happens when, for instance, $b_4 = b_5 = c$, $d_3 = b_1$ and $d_5 = b_2$. Then, d_3 and d_5 are made into type-2, while d_1 is made into type-3. □

Naturally, this lemma could force the approximation to be 2. Let $|s_u| = |s_v| = |s_w| = 1$. The insertion of c and s_u, s_v, s_w optimally will generate $4 \times 2 = 8$ adjacencies. If c is misplaced with the greedy search step, a total of 4 adjacencies are generated (2 from c; 1 each from s_v and s_w, which are type-2 now, and 0 from s_u, which is type-3 now).

Let $Y_{i,j}$ be the set of (length-i,type-1) substrings in some optimal solution converted into type-j by the approximation algorithm — due to the misplacement of some (length-1,type-1) substrings at Step 1, for $i \geq 1, 2 \leq j \leq 3$. We have the following lemma.

Lemma 2. *At Step 2, a total of* $b'_{12} + \sum_{i \geq 1} |Y_{i,2}|$ *new type-2 common (external) adjacencies are generated; moreover,* $b'_{12} \geq b_{12}$.

Proof. If a slot $t = \langle u, v \rangle$ could be inserted with a (length-1,type-1) substring s_i, then there is an optimal solution in which a (length-1,type-2) substring (letter) x is not inserted at the slot t. The reason is as follows. Suppose that the slot t can be inserted with a (length-1,type-1) substring s_i to obtain two new common adjacencies us_i and s_iv. Suppose that in some optimal solution the slot t is inserted with an x to generate one common adjacency and s_i is inserted at some slot t' instead. Then, if s_i generates no common adjacency in t' we could swap x with s_i to generate at least two common adjacencies. This contradicts with the optimality of the assumed optimal solution. If s_i generates one common adjacency in t', swapping x with s_i gives us a solution at least as good as in the optimal solution. If x is type-2 at both the slots t and t', and s_i is type-1 at both slots the slots t and t', then we could in fact obtain an optimal solution with the greedy search.

If a slot $t = \langle u, v \rangle$ could be inserted with a (length-1,type-1) substring s_i but was inserted with two (length-1,type-2) substrings s_j and s_k instead, a similar argument follows. In this case we could swap s_i and s_js_k to have another optimal solution with the same number of adjacencies; moreover, the (length-1,type-1) substring is always inserted before (length-2,type-1) substrings.

Then, following the maximality of the bipartite maximum matching at Step 2, we have $b'_{12} \geq b_{12}$. By the previous lemma, each misplaced (length-1,type-1) substring could result in putting two type-1 substrings into $Y_{i,2}$ and $Y_{j,2}$, for some $i, j \geq 1$. With the bipartite maximum matching, one external common adjacency is generated for each substring in $Y_{i,2}$. Hence the lemma is proved. □

Lemma 3. *At Step 3, the size of the maximum matching, $|M|$, satisfies*
$|M| \geq \frac{1}{2}\left(\sum_{i=2..p}(i+1)b_{i1} + \sum_{i=2..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3}\right)$
$+ \left(\sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,2}| + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}|\right)$.

Proof. On the right hand side of the above inequality, the first summation represents the optimal internal adjacencies among the corresponding type-1, type-2, and type-3 substrings in the optimal solution (with length at least 2). The second summation represents the number of internal adjacencies of the substrings in $Y_{i,2}$ and $Y_{i,3}$, converted from type-1 due to the greedy search at Step 1. □

Lemma 4. *Let b'_{11} be the number of (length-1,type-1) substrings obtained at Step 1, then by Step 3 the number of adjacencies generated by the algorithm which are due to the selection of B'_{11} (instead of B_{11}) satisfies that*

$$2b'_{11} + \sum_{i\geq 1}(1 + \lfloor\frac{i}{2}\rfloor|Y_{i,2}|) + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}| \geq b_{11}.$$

Proof. As discussed earlier (Lemma 1), each letter in B'_{11} could make two type-1 substrings, s_v and s_w, into type-2; and make one type-1 substring, s_u, into type-3. With Step 2 (and Lemma 2), we could guarantee that the algorithm can generate one external adjacency for all type-2 substrings (including those type-2 ones in some optimal solution and those type-1 ones in some optimal solution but converted to type-2 due to a misplacement of some (length-1,type-1) substring at Step 1). With Step 3, we could guarantee that for each type-2 and type-3 substring of length ℓ one could generate a number of internal adjacencies at least $\lfloor\ell/2\rfloor$. Then following the approximation solution, with a misplaced (length-1,type-1) substring x, we generate $2 + (1 + \lfloor|s_v|/2\rfloor) + (1 + \lfloor|s_w|/2\rfloor) + \lfloor|s_u|/2\rfloor$ adjacencies, while with the optimal solution we could generate $2 + (|s_v| + 1) + (|s_w| + 1) + (|s_u| + 1)$ adjacencies. This gives us

$$\frac{2 + (|s_v| + 1) + (|s_w| + 1) + (|s_u| + 1)}{2 + (1 + \lfloor|s_v|/2\rfloor) + (1 + \lfloor|s_w|/2\rfloor) + \lfloor|s_u|/2\rfloor}$$

$$= \frac{2 + (|s_v| + 1) + (|s_w| + 1) + (|s_u| + 1)}{1 + (1 + \lfloor|s_v|/2\rfloor) + (1 + \lfloor|s_w|/2\rfloor) + (1 + \lfloor|s_u|/2)\rfloor},$$

which is obviously bounded above by 2, due to that $|s_u|, |s_v|, |s_w| \geq 1$.

Therefore, by Step 3 the number of adjacencies generated by the algorithm due to the forming of B'_{11} (instead of B_{11}) satisfies that

$$2b'_{11} + \sum_{i\geq 1}(1 + \lfloor\frac{i}{2}\rfloor|Y_{i,2}|) + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}| \geq \frac{1}{2}(2b_{11}) = b_{11}.$$

□

Hence, we could have the following theorem.

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

Proof. By definition, the optimal solution value OPT satisfies

$$Opt = \sum_{i=1..p}(i+1)b_{i1} + \sum_{i=1..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3},$$

for some p, q, r. At Step 3, the size of the maximum matching, $|M|$, satisfies $|M| \geq \frac{1}{2}\left(\sum_{i=2..p}(i+1)b_{i1} + \sum_{i=2..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3}\right)$ $+ \left(\sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,2}| + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}|\right)$.

The approximation solution value, *App*, satisfies

$$\begin{aligned}
App &= 2b'_{11} + (b'_{12} + \sum_{i\geq 1}|Y_{i,2}|) + |M| \\
&\geq 2b'_{11} + (b'_{12} + \sum_{i\geq 1}|Y_{i,2}|) \\
&\quad + \frac{1}{2}\{\sum_{i=2..p}(i+1)b_{i1} + \sum_{i=2..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3}\} \\
&\quad + (\sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,2}| + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}|) \\
&\quad (by\ Lemma\ 3) \\
&\geq \{2b'_{11} + \sum_{i\geq 1}(1 + \lfloor\frac{i}{2}\rfloor|Y_{i,2}|) + \sum_{i\geq 2}\lfloor\frac{i}{2}\rfloor|Y_{i,3}|\} + b_{12} \\
&\quad + \frac{1}{2}\{\sum_{i=2..p}(i+1)b_{i1} + \sum_{i=2..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3}\} \\
&\quad (by\ Lemma\ 2) \\
&\geq b_{11} + b_{12} \\
&\quad + \frac{1}{2}\{\sum_{i=2..p}(i+1)b_{i1} + \sum_{i=2..q} ib_{i2} + \sum_{i=2..r}(i-1)b_{i3}\} \\
&\quad (by\ Lemma\ 4) \\
&\geq \frac{1}{2}Opt.
\end{aligned}$$

□

The running time of the algorithm is dominated by the computation of maximum matching in a bipartite graph with $O(n)$ vertices at Step 2, and also by the computation of maximum matching in a general graph with $O(n)$ vertices at Step 3, both taking $O(n^{2.5})$ time [13,11].

Regarding the lower bound of the running time, in fact, we could show that the algorithm must take at least $\Omega(n^{2.5})$ time (or, the current best running time to compute the maximum matching in a general graph) [11]. The proof is by modifying that of Theorem 1. We could use two additional genes $\#_4, \#_5$ (after $\#_2\#_3$ in G). Then, we keep $S = \langle C_1, C_2\rangle$, but we make minor changes on C_1 and C_2; i.e., $C_1 = \langle \#_2 v_1^{deg(v_1)-1}\#_1 \cdots v_n^{deg(v_n)-1}\#_1\#_5\rangle$ and $C_2 = \langle \#_3\#^{m+1}\#_4\rangle$. Obviously, running the approximation algorithm on this instance, Step 1 and Step 2 does nothing as no external adjacencies could be formed. Then Step 3 must run a maximum matching algorithm on the original graph H, on which G and S are constructed, to return the approximate solution. Hence, this algorithm must take $\Omega(n^{2.5})$ time, which is the current best running time for computing the maximum matching in a general graph H [11].

4. 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). Here a contig cannot be altered, hence missing genes (relative to a reference gonome) can only be inserted in between contigs. We obtain a factor-2 approximation algorithm for this NP-complete problem. We hope this could attract more algorithmic results which could eventually lead to the practical processing of genomic datasets.

Acknowledgments

This research is partially supported by National Natural Science Foundation of China under projects 61628207, 61472222, 61761136017 and 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 suggestions which greatly improve the presentation of this paper.

References

1. 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.
2. 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.
3. M. Boetzer and W. Pirovano. Toward almost closed genomes with GapFiller. *Genome Biology*, 13(6):R56, 2012.
4. L. Bulteau, A.P. Carrieri and R. Dondi. Fixed-parameter algorithms for scaffold filling. *Theoretical Computer Science*, 568: 72–83, 2015.
5. 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.
6. 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.
7. 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.

8. 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.
9. 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.
10. 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.
11. H. Gabow. The weighted matching approach to maximum cardinality matching. *Fundam. Inform.*, 154(1-4):109-130, 2017.
12. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman. 1979.
13. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225-231, 1973.
14. M. Hunt, C. Newbold, M. Berriman, and T. Otto. A comprehensive evaluation of assembly scaffolding tools. *Genome Biology*, 15(3):R42, 2014.
15. D. Huson, K. Reinert, and E. Myers. The greedy path-merging algorithm for contig scaffolding. *J. ACM*, 49(5):603-615, 2002.
16. H. Jiang, C. Fan, B. Yang, F. Zhong, D. Zhu, and B. Zhu. Genomic scaffold filling revisited. *Proc. 27th Annual Combinatorial Pattern Matching Symposium (CPM'16)*, LIPIcs 54, pp. 15:1-15:13, 2016.
17. 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, 2011.
18. 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.
19. M. Jiang. The zero exemplar distance problem. *Proc. of the 2010 International RECOMB-CG Workshop (RECOMB-CG'10)*, LNBI 6398, pp. 74-82, 2010.
20. 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.
21. 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.
22. N. Liu, P. Zou, and B. Zhu. A polynomial time solution for permutation scaffold filling. *Proc. 10th Annual International Conference on Combinatorial Optimization and Applications (COCOA'16)*, LNCS 10043, pp. 782-789, 2016.
23. J. Ma, H. Jiang, D. Zhu and S. Zhang. A 1.4-approximation algorithm for two-sided scaffold filling. *Proc. 11th Intl. Frontiers in Algorithmics Workshop (FAW'17)*, LNCS 10336, pp. 196-208, 2017.
24. 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.
25. D. Paulino, R. Warren, B. Vandervalk, A. Raymond, S. Jackman, and I. Birol. Sealer: a scalable gap-closing application for finishing draft genomes. *BMC Bioinformatics*, 16(1):230, 2015.
26. S. Yancopoulos, O. Attie and R. Friedberg. *Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics*, **21**:3340-3346, 2005.

27. 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.
28. B. Zhu. Genomic scaffold filling: A progress report. *Proc. 10th Intl. Frontiers in Algorithmics Workshop (FAW'16)*, LNCS 9711, pp. 8-16, 2016.

Haitao Jiang is currently an associate professor in School of Computer Science and Technology, Shandong University, China. He obtained his Ph.D. in Computer Science from Shandong University in 2011. During August 2009 and January 2011, he visited Montana State University where he finished most of his thesis research. Dr. Jiang's research interests are bioinformatics, computational biology and algorithms. He has published over 40 papers in these areas.

Letu Qingge is currently a PhD student at Gianforte School of Computing, Montana State University, USA. His research interests are computational biology and algorithms.

Daming Zhu is currently a professor in School of Computer Science and Technology, Shandong University, China. He obtained his Ph.D. in Computer Science from Institute of Computing Technology, Chinese Academy of Sciences, China in 1999. Dr. Zhu's research interests are algorithms, computational biology and machine learning. He has published over 60 papers in these areas.

Binhai Zhu is currently a professor in computer science at Gianforte School of Computing, Montana State University, USA. He obtained his Ph.D. in Computer Science from McGill University, Canada, in 1994. He was a post-doctoral research associate at Los Alamos National Laboratory, USA from 1994 to 1996. From 1996 to 2000, he was an assistant professor at City University of Hong Kong.
He has been at Montana State University since 2000 (associate professor until 2006, professor since 2006). Professor Zhu's research interests are geometric computing, biological/geometric modeling, bioinformatics and combinatorial optimization. He has published over 180 papers in these areas.