A 2-Approximation Algorithm for Contig-based Genomic Scaffold Filling

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

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

Farong Zhong
College of Math, Physics and Information Engineering
Zhejiang Normal University
Jinhua, Zhejiang
China
Email: zfr@zjnu.cn

Binhai Zhu
Corresponding author
Gianforte School of Computing
Montana State University
Bozeman, MT 59717, USA
Email: 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 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 $X = c(G) - c(S)$ can only be inserted in between the contigs, and the objective is to maximize the number of common adjacencies between $G$ and the filled $S'$. 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.

I. 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 [5]. The consequence 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 [18]. 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 [19]) between $R$ and $S'$ is minimized. It was shown that this problem can be solved in polynomial time. In [13], 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 [6], [8], [2], [3], [14], exemplar adjacency number [7], [9], or the minimum common string partition [10].) The similarity measure adopted for the scaffold filling problem is the number of common (string) adjacencies, which can be computed in polynomial time [2], [12], [13]. In [12], [13], 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 proposed in [12], [13], and this bound has been improved to 1.25 [15]. 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 [16]. Using the number of common adjacencies as a parameter, it was also shown that this problem is 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 [4].

The motivation of this paper is as follows. 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 Celera Assembler [1], hence should not be arbitrarily altered. This case was considered briefly in [18], [13], [17], all other research on scaffold filling used an incomplete sequence as a scaffold.

The main contribution of this paper is to present some research result along this line. We formally call the problem as One-sided Scaffold Filling (One-sided-SF-max). The objective function in this case is to maximize the number of common adjacencies between the 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.

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. We conclude the paper in Section 4.

II. 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 [13], [20]. We assume that all genes and genomes are unsigned, and it is straightforward to generalize the result to signed genomes. Given a gene set \( X \), a string \( P \) is called permutation if each element in \( X \) 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 \( X \). For example, \( X = \{a, b, c, d\} \), \( A = abcdacd \), \( c(A) = \{a, a, b, c, c, d, d\} \). 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 \( a_{i+1} = b_{j+1} \) (or \( a_{i+1} = b_{j+1} \)), where \( a_i, a_{i+1} \in P_A \) and \( b_j, b_{j+1} \in P_B \), we say that \( a_i, a_{i+1} \) and \( b_j, b_{j+1} \) 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.

![Fig. 1. An example for adjacency and breakpoint definitions.](image-url)

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 over a gene set \( X \) whose contents should not be altered. A scaffold \( S \) is simply a sequence of contigs \( \langle C_1, \ldots, C_m \rangle \). 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 \( S = \langle C_1, C_2, \ldots, C_m \rangle \) where \( G \) and the contig \( C_i \)'s are over a gene set \( \Sigma \), 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 \( V = \{v_1, v_2, \ldots, v_n\} \) and \( e_i = (v_i, v_{i+1}) \) for \( i = 1 \ldots m \), and for \( e_i \in E \), let \( \ell_i = v_i v_{i+1} \) for \( i = 1 \ldots m \). Let \( deg(v) \) be the degree of vertex \( v \) (assuming \( deg(v) > 1 \) for all \( v \)). \( G \) and \( S \) are constructed as follows.

\[
G = \#e'_1 \#e'_2 \# \cdots \#e'_m \# o \#2 #\#1, \]

and

\[
S = \langle C_1, C_2 \rangle, \]

with \( C_1 = \langle #2 v_1^{deg(v_1)-1} #1 \cdots v_{m}^{deg(v_m)-1} #1 \rangle \) and \( C_2 = \langle #m #3 \rangle \). Here \( o \) 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 if and only if \( n \) 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 \( v'_i v'_2 \cdots v'_n \) (where \( v'_i = v_i \) for some \( i \)), then the \( n - 1 \) adjacencies \( v'_i v'_{i+1} \) must correspond to an edge in \( H \) and the other two are \#\( v'_{i} \) and \( v'_{n} \# \). Then, by definition, the substring \( v'_i v'_2 \cdots v'_n \) corresponds to a Hamiltonian path in \( H \). It is obvious that this reduction take \( O(n^2) \) 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
\[
G = \#v_1 v_3 \#v_1 v_2 \#v_2 v_4 \#v_2 v_5 \#v_4 v_5 \#v_2 v_3 \\
\#\#\#\#\#\#\#
\]
\[
S = \#2v_1 \#1v_2 v_2 \#1v_3 \#1v_4 \#1v_5 \#1\#
\]
After inserting genes in \( V \) into \( S \), we obtain
\[
S^* = \#2v_1 \#1v_2 v_2 \#1v_3 \#1v_4 \#1v_5 \#1\#
\]
It is easy to verify that we have \( n + 1 = 6 \) common adjacencies between \( G \) and \( S^* \): \#1v_1, \#1v_3, \#1v_2, \#1v_4, \#1v_5, \#1v_3.

We note that the reduction for the unbounded case SF-MNSA (from X3C in [12], [13]) 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.

III. AN APPROXIMATION ALGORITHM FOR ONE-SIDED-SF-MAX

Before presenting our algorithm, we make the following definitions.

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

We define a type-1 substring \( s \) of length \( \ell \geq 1 \), over \( X \), as one which can be inserted in \( (\beta_i, \alpha_{i+1}) \), for \( 1 \leq i \leq m - 1 \), to generate \( \ell + 1 \) new common adjacencies. We call \( (\beta_i, \alpha_{i+1}) \) 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 lock it right away.) 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 \( (\beta_i, \alpha_{i+1}) \), for \( 0 \leq i \leq m \), to generate \( \ell \) 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 \( (\beta_i, \alpha_{i+1}) \), we could have two type-2 slots, i.e., right after \( \beta_i \) (written as \( \beta_i \alpha_i \)) or right before \( \alpha_{i+1} \) (written as \( \alpha_{i+1} \beta_i \)). By definition, for a fixed slot \( (\beta_i, \alpha_{i+1}) \), 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 even if \( \beta_i \alpha_{i+1} \) is already a common adjacency with respect to \( G \), it is still 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: \( G = (1, 1, 5, 4, 3, 5, 3, 7, 7), S = (1, 1, 7, 3, 5, 3, 1, 5, 7) \), the missing gene 4 must be inserted between \( 1, 7, 3, 5, 3, 1, 5, 7 \) to obtain the optimal solution.

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 one which can be inserted in the slot \( (\beta_i, \alpha_{i+1}) \), for some \( i \), to generate \( \ell - 1 \) common adjacencies. Note that a type-3 substring can only form internal adjacencies, hence it does not matter where we insert \( s \) — provided that it does not destroy any existing adjacencies.

We show an example as follows:
\[
G = (1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6), \\
S = (1, 5, 3, 6, 2, 4). \\
\]
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
\[
S' = (1, 2, 1, 5, 6, 3, 6, 5, 4, 3, 2, 4). \\
\]
In this case, \( (5, 4, 3) \) is type-1, \( (6) \) and \( (1, 2) \) are type-2. Among the common adjacencies obtained, one of \( (1, 2) \) and \( (2, 1) \) is internal (and the other external), both of \( (5, 6) \) and \( (6, 5) \) are external, both of \( (5, 4) \) and \( (4, 3) \) are internal, and \( (3, 2) \) is external.

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 based on greedy search and two levels of maximum matchings:
Let $b_{ij}$ denote the number of $j$-type-1 substrings in some optimal solution and let $B_{ij}$ be the corresponding set of $j$-type-1 substrings. Then the optimal solution value is

$$\text{Opt} = \sum_{j=1}^{p} \sum_{i=1}^{q} b_{ij} + \sum_{j=1}^{r} \sum_{i=1}^{s} j b_{ij} + \sum_{j=2}^{t} \sum_{i=1}^{u} \sum_{j=2}^{t} (j-1) b_{ij},$$

for some $p,q,r$. Let $b'_{ij}$ denote the number of $j$-type-1 substrings in the approximation solution and let $B'_{ij}$ be the corresponding set of $j$-type-1 substrings. We show the properties of the approximation algorithm as follows.

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

**Proof:** All we need to prove is that if an 1-type-1 letter was inserted at a wrong slot, then at most three type-1 substrings $s_u, s_t,$ and $s_w$ are not type-1 anymore. To see this, assume that in some optimal solution an 1-type-1 letter $c$ is inserted in the slot $v_1$ and three type-1 substrings $s_u, s_t,$ and $s_w$ are inserted in the slots $z_1, z_t, z_w$ respectively, for $i = 1, 2, 3$. Suppose that $c$ is inserted in one of the slots $z_{i+1}$, say $z_{i+1}$, as type-1. Then, in the worst case, all $s_u, s_t,$ and $s_w$ cannot be type-1 anymore: $s_u$ could be type-3 and $s_t$ and $s_w$ could be type-1.

We could verify this with a simple example: $S = \{a_2, a_3, a_4, \ldots, b_1, b_2, \ldots, b_3, b_4, \ldots, b_5, b_6, \ldots\}$. Let $X = \{c, a_1, a_2, d_1, d_2, d_3\}$. 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, $b_3 = a_1 = c$, $d_3 = b_1$ and $d_5 = b_2$; and $d_1$ is made into type-3.

Naturally, this lemma could force the approximation to be 2. Let $|s_u| = |s_t| = |s_w| = 1$. The insertion of $c$ and $s_u, s_t, s_w$ 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 $c$: each from $s_u$ 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 $i$-type-1 substrings in some optimal solution converted into type-$j$ by the approximation algorithm — due to the misplacement of some 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 $b'_{21} + \sum_{j=1}^{3} \sum_{i=1}^{j} y_{i,j}$ new type-2 common (external) adjacencies are generated; moreover, $b'_{21} \geq b_{21}$.

**Proof:** If a slot $t = \langle u,v \rangle$ could be inserted with an 1-type-1 substring $s_i$, then there is an optimal solution in which an 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 an 1-type-1 substring $s_i$ to obtain two new common adjacencies $u_s$ and $w_s$. 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 an 1-type-1 substring $s_i$ but was inserted with two 1-type-2 substrings $s_j$ and $s_k$ instead, a similar argument follows. In this case we could swap $s_i$ and $s_j$ to have another optimal solution with the same number of adjacencies; moreover, the 1-type-1 substring is always inserted between 1-type-2 substrings.

Then, following the maximality of the bipartite maximum matching at Step 2, we have $b'_{21} \geq b_{21}$. By the previous lemma, each misplaced 1-type-1 substring could put two type-1 substrings in $Y_{i,2}$ 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

$$|M| \geq \sum_{i=2}^{p} \sum_{j=1}^{q} (j+1) b'_{ij} + \sum_{j=2}^{r} j b_{ij} + \sum_{j=2}^{t} \sum_{i=1}^{u} \sum_{j=2}^{t} (j-1) b_{ij} + \sum_{i=2}^{p} \sum_{j=2}^{r} \sum_{i=1}^{s} \sum_{j=2}^{t} (j-1) b_{ij} - \sum_{i=2}^{p} \sum_{j=2}^{r} \sum_{i=1}^{s} \sum_{j=2}^{t} (j-1) b_{ij}.$$

**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 $Y_{j,3}$, converted from type-1 due to the greedy search at Step 1.

**Lemma 4:** Let $b'_{11}$ be the number of 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

$$2b'_{11} + \sum_{i=2}^{p} \sum_{j=1}^{q} \sum_{i=1}^{s} \sum_{j=2}^{t} 2b_{ij} + \sum_{j=2}^{t} \sum_{i=1}^{s} \sum_{j=2}^{t} \sum_{i=1}^{s} \sum_{j=2}^{t} (j-1) b_{ij} \geq 0.$$
and make one type-1 substring, $s_v$, 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 misplaced of some 1-type-1 substring at Step 1). With Step 3, we could guarantee that for each type-2 and type-3 substring of length $\ell$ one could generate a number of internal adjacencies at least $\lfloor \ell / 2 \rfloor$. Then following the approximation solution, with a misplaced 1-type-1 substring $x$, we generate $2 + (1 + \lfloor s_u \rfloor / 2) + (1 + \lfloor s_w \rfloor / 2) + \lfloor s_u \rfloor / 2$ adjacencies, while with the optimal solution we could generate $2 + (|s_v| + 1) + (|s_u| + 1) + (|s_u| + 1)$ adjacencies. This gives us

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

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

\[
2b_{11}' + \sum_{i \geq 1} (1 + \lfloor i/2 \rfloor |Y_i, 2|) + \sum_{j \geq 2} \lfloor j/2 \rfloor |Y_j, 3| \geq \frac{1}{2} (2b_{11}) = b_{11}.
\]

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

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

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

\[
|M| \geq \frac{1}{2} \left( \sum_{j = 2..p} (j + 1)b_{1j} + \sum_{j = 2..q} jb_{2j} + \sum_{j = 2..r} (j - 1)b_{3j} \right) + \left( \sum_{i \geq 2} \lfloor i/2 \rfloor |Y_i, 2| + \sum_{j \geq 2} \lfloor j/2 \rfloor |Y_j, 3| \right).
\]

The approximation solution value, $App$, satisfies

\[
App = 2b_{11}' + (b_{21}' + \sum_{i \geq 1} |Y_i, 2|) + |M| \geq 2b_{11}' + (b_{21}' + \sum_{i \geq 1} |Y_i, 2|)
\]

\[
+ \frac{1}{2} \left( \sum_{j = 2..p} (j + 1)b_{1j} + \sum_{j = 2..q} jb_{2j} + \sum_{j = 2..r} (j - 1)b_{3j} \right) + \left( \sum_{i \geq 2} \lfloor i/2 \rfloor |Y_i, 2| + \sum_{j \geq 2} \lfloor j/2 \rfloor |Y_j, 3| \right)
\]

(by Lemma 3)

\[
\geq \left( 2b_{11}' + \sum_{i \geq 1} (1 + \lfloor i/2 \rfloor |Y_i, 2|) + \sum_{j \geq 2} \lfloor j/2 \rfloor |Y_j, 3| \right) + b_{21}
\]

\[
+ \frac{1}{2} \left( \sum_{j = 2..p} (j + 1)b_{1j} + \sum_{j = 2..q} jb_{2j} + \sum_{j = 2..r} (j - 1)b_{3j} \right)
\]

(by Lemma 2)

\[
\geq b_{11} + b_{21}
\]

\[
+ \frac{1}{2} \left( \sum_{j = 2..p} (j + 1)b_{1j} + \sum_{j = 2..q} jb_{2j} + \sum_{j = 2..r} (j - 1)b_{3j} \right)
\]

(by Lemma 4)

\[
\geq \frac{1}{2} OPT.
\]

The running time of the algorithm is dominated by the computation of maximum matching in a general graph at Step 3, which takes $O(n^3)$ time.

**IV. 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 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. On the other hand, theoretically, it is interesting to decide whether One-Sided-SF-max is FPT.

**ACKNOWLEDGMENTS**

This research is partially supported by NSF of China under project 61628207 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 comments.

**REFERENCES**


Haitao Jiang

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 2010, 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.

Farong Zhong

Farong Zhong is currently a professor in College of Mathematics, Physics and Information Engineering, Zhejiang Normal University, China. He obtained his Ph.D. in Computer Science from Shanghai Jiao Tong University, China in 2005. Dr. Zhong’s research interests are formal methods, algorithms and theory of computation. He has published over 40 papers in these areas.

Binhai Zhu

Binhai Zhu is currently a professor in computer science at 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.