Scaffold Filling Under the Breakpoint and Related Distances

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

Chunfang Zheng
Départment d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, Quebec H3C 3J7
Canada
Email: chunfang313@gmail.com

David Sankoff
Department of Mathematics and Statistics
University of Ottawa
Ottawa, Ontario
Canada KIN 6N5
Email: sankoff@uottawa.ca

Binhai Zhu
Corresponding author
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA
Email: bhz@cs.montana.edu

***Abstract*—Motivated by the trend of genome sequencing without completing the sequence of the whole genomes, a problem on filling an incomplete multichromosomal genome (or *scaffold*) I with respect to a complete target genome G was studied. The objective is to minimize the resulting genomic distance between I' and G, where I' is the corresponding filled scaffold. We call this problem the *one-sided* scaffold filling problem. In this paper, we conduct a systematic study for the scaffold filling problem under the breakpoint distance and its variants, for both unichromosomal and multichromosomal genomes (with and without gene repetitions). When the input genome contains no gene repetition (i.e., is a fragment of a permutation), we show that the *two-sided* scaffold filling problem (i.e., G is also incomplete) is polynomially solvable for unichromosomal genomes under the breakpoint distance and for multichromosomal genomes under the genomic (or DCJ — Double-Cut-and-Join) distance. However, when the input genome contains some repeated genes, even the one-sided scaffold filling problem becomes NP-complete when the similarity measure is the maximum number of adjacencies between two sequences. For this problem, we also present efficient constant-factor approximation algorithms: factor-2 for the general case and factor 1.33 for the one-sided case.**

Keywords: comparative genomics, scaffold filling, breakpoint distance, genomic distance, DCJ, NP-completeness, algorithms.

I. Introduction

Due to the advancement of genome sequencing technology, it is possible to sequence more organisms for genomic analysis. (Throughout this paper, a multichromosomal genome is represented as sequences of genes, while a unichromosomal genome is just represented as a sequence of genes.) Interestingly and somehow contradictorily, the cost of genome sequence finishing has not decreased at the same rate as the cost of random sequencing [2]. This means that many released genomes are not completely finished. In some situations, it would be unsuitable to use these incomplete genomes (scaffolds) for genomic analysis, simply due to the errors they could introduce.

Therefore, a natural problem is to fill the missing genes into scaffolds, with combinatorial algorithms. As one must find a biologically meaningful way to fill scaffolds, it makes sense to use a complete genome from a close species. Muñoz *et al*. recently carried out this idea on filling an incomplete multichromosomal scaffold I to have I', such that the genomic distance between I' and a given (complete) genome G is minimized [10]. The genomic distance is also called *rearrangement* distance or DCJ — Double-Cut-and-Join distance [15], which is the minimum number of allowed rearrangement operations transforming one genome into the other. We call this the *one-sided* scaffold filling problem. Basically, the one-sided scaffold filling can be solved in polynomial time; in fact, linear time when the breakpoint graph on I and G is constructed [10].

In a recent paper [10], much effort has been put on several practical issues. For instance, what if the missing genes can only be inserted in certain locations? What if some missing genes in I are not really missing (i.e., they should not appear in G)? However, the corresponding two-sided problem, i.e., when G also contains missing genes, is not tackled in [10].

In this paper, we first follow Muñoz *et al*. to investigate the scaffold filling problem under the breakpoint distance for the simplest unichromosomal genomes. When the input genome contains no gene repetition (i.e., is a fragment of a permutation), we show that the *two-sided* scaffold filling problem under the breakpoint distance is polynomially solvable. Consequently, we generalize the idea to solve the two-sided scaffold filling problem in polynomial time for multichromosomal genomes under the DCJ distance.

Next, we extend this research to filling scaffolds with gene repetitions. There is some interesting point regarding this work. First, when there are gene repetitions, some practical way is missing to define the genomic (say breakpoint) distance between two genomes G_1, G_2. (The breakpoint distance for permutations was defined in [14].) The *exemplar genomic*

distance between G_1 and G_2 (loosely speaking, the 'true' distance when the redundant genes are deleted), while biologically interesting, is too hard to compute [12]. (In fact, unless P=NP, there does not exist any polynomial time approximation for such a distance even when each gene is allowed to repeat three times [5], [3] or even two times [1], [9].) We try to apply a different similarity measure here. We use the number of (common) adjacencies between G_1 and G_2, which has been used before in genomic analysis [4]. (Note that this is not a distance measure.) Not surprisingly, we prove that even the corresponding one-sided problem using this similarity measure is NP-complete, implying the two-sided problem to be NP-complete as well. Then, we design a factor-1.33 approximation for the one-sided problem — filling scaffolds with gene repetitions to maximize the number of adjacencies, improving upon a trivial factor-2 approximation for the general (two-sided) case. For genomic problems with gene repetitions, as far as we know, this is the first one which admits such a small approximation factor.

Although this paper is a theoretical one, it is necessary to point out that some practical issues need to be further investigated. For instance, our first algorithm does not consider the fact that in some situations one cannot insert a missing gene into a contig. Intuitively, this makes the problem harder. In fact, in practice even ordering contigs into a draft genome needs some novel tool (see [11] and the references therein). For more information, the interested readers are referred to public Web pages like Sequence Similarity Searching (`http://www.ebi.ac.uk/Tools/sss/`), etc.

This paper is organized as follows. In Section 2, we give the necessary definitions. In Section 3, we present a polynomial time algorithm for the scaffold filling problem under the breakpoint distance. In Section 4, we show the NP-completeness proof for the one-sided scaffold filling problem when gene duplications are allowed and when the similarity measure is the maximum number of (common) adjacencies, and we also present efficient constant-factor approximation algorithms for the problem. In Section 5, we show how to adapt our ideas from Section 3 to solve the two-sided scaffold filling problem under the rearrangement distance (i.e., for multichromosomal genomes) in polynomial time. In Section 6, we conclude the paper.

II. Preliminaries

We first present some necessary definitions.

Given an alphabet Σ, a string S is called a *permutation* if each element in Σ appears exactly once in S. We also use $c(S) \subseteq \Sigma$ to denote the set of elements in permutation S. An (unsigned) unichromosomal genome is just a permutation over Σ.

A *scaffold* is an incomplete permutation, i.e., with some missing elements. We use $+$ to denote permutation scaffold filling, e.g., for a permutation A and an element set X such that $c(A) \cap X = \emptyset$, $A + X$ is the set of the resulting permutations after filling all the elements in X into A. So if $A^* \in A + X$, then A is a subsequence of A^*. Similarly, we use $-$ to denote element elimination from the permutation when $c(A) \cap X = X$, i.e., $A - X$ is the resulting permutation after elements in X are removed from A. In other words, if $A' = A - X$ then A is a supersequence of A'. Given two permutations A and B, if $c(A) = c(B)$, then A and B are *related*. Given two related permutations A and B, two consecutive elements a_i and a_{i+1} (i.e., a 2-substring $a_i a_{i+1}$) in A form an *adjacency* if they are also consecutive in B (i.e., as a 2-substring $a_i a_{i+1}$ or $a_{i+1} a_i$), otherwise they form a *breakpoint*. The number of breakpoints in A, which is equal to that of B, is the breakpoint distance between A and B, denoted as $bd(A, B)$. Note that our breakpoint definition and the corresponding results all adapt to the case when the letters (or genes) are possibly *signed*, with a simple twist (i.e., a substring $a_i a_{i+1}$ of A is an adjacency if either $a_i a_{i+1}$ or $-a_{i+1} - a_i$ appears in B).

Given two related permutations P and Q of length n, let the number of breakpoints in P and Q be $bd(P, Q)$ and let the number of adjacencies in P and Q be $a(P, Q)$, then we have

$$a(P, Q) + bd(P, Q) = n - 1,$$

moreover, when $a(P, Q) = n - 1$ then we have $P = Q$ or P is the reversal of Q. However, when we are given two sequences G_1 and G_2, over the same multiset of letters (i.e., possibly with duplications of some or all letters), then the latter relation is not true any more. Example: Let $G_1 = bcidabeb$ and $G_2 = bebcidab$, then the number of adjacencies between G_1 and G_2 is 7 (which is the maximum), but G_1 is not equal to G_2 or its reversal.

We define two (*two-sided*) scaffold filling problems on unichromosomal genomes as follows. The case on multichromosomal genomes will be covered at the end of this paper.

Scaffold Filling under the Minimum Permutation Breakpoint Distance (SF-PBD)
Input: two incomplete permutations A and B, and two sets of elements X and Y, where $X = c(B) - c(A)$ and $Y = c(A) - c(B)$.
Question: minimize $bd(A^*, B^*)$, with $A^* \in A + X, B^* \in B + Y$.

scaffold A: a b c d e f g h i k l m o p (j and n missing)

scaffold B: a h f d j i b c k o n l p (e,g and m missing)

filled
scaffold A: a#b c#d e f g h#i j#k#l m n o#p

filled
scaffold B: a#h g f e d#j i#b c#k#o n m l#p

Fig. 1. **An example for SF-PBD. Each # indicates a breakpoint.**

In Figure 1, we show an easy example for SF-PBD. Note that the optimally filled solutions might not necessarily be unique. In the above definition, when either X or Y is empty, we have the *one-sided* scaffold filling problem. Note that if A and B were related (i.e., $c(A) = c(B)$), then we would have $X = Y = \emptyset$.

In practice, sometimes we need to deal with genomes with orthologous (duplicated) genes. Then, our definition on

scaffold and the related $+$ operation can be generalized to sequences. Let $C(S)$ be a multiset denoting all the appearances of all the elements in a sequence S. For example, given $S = abacb$, $C(S) = \{a, a, b, b, c\}$.

We comment that for general *related* sequences G, H (with $C(G) = C(H)$ and possibly with duplicated letters), the number of (common) adjacencies is counted in an one-to-one matching fashion, i.e., a 2-substring ab in G is only counted once if it matches to another (uncounted) 2-substring ab or ba in H, and vice versa. For example, $G = abab, H = aabb$, then the number of common adjacencies between G and H is one. The reason is that while the 2-substring ab in H can be matched to three 2-substrings in G, once counted, we have no 'matching' (hence no common adjacency) anymore.

Scaffold Filling to Maximize the Number of (Sequence) Adjacencies (SF-MNSA)
Input: two incomplete sequences G_1 and G_2 and two multisets of elements X and Y, where $X = C(G_2) - C(G_1)$ and $Y = C(G_1) - C(G_2)$.
Question: maximize the number of adjacencies $a(g_1, g_2)$, with $g_1 \in G_1 + X, g_2 \in G_2 + Y$.

Again, note that if G_1 and G_2 were related (i.e., $C(G_1) = C(G_2)$), then we would have $X = Y = \emptyset$. In the above definition, when either X or Y is empty, we have the corresponding *one-sided* scaffold filling problem. To save space, we only give a definition for one-sided SF-MNSA, which is formally presented as follows.

One-sided SF-MNSA
Input: an incomplete sequence I and a complete sequence G, with $X = C(G) - C(I) \neq \emptyset$ and $Y = C(I) - C(G) = \emptyset$.
Question: maximize the number of adjacencies $a(z, G)$, with $z \in I + X$.

In the next sections we show that SF-PBD is polynomially solvable, and that the One-Sided SF-MNSA problem is NP-hard, which implies that SF-MNSA is also NP-hard. For SF-MNSA we obtain an easy factor-2 approximation which can be improved to 4/3 for the special case, One-Sided SF-MNSA.

III. A Polynomial Time Algorithm for SF-PBD

In this section we present a polynomial time algorithm for scaffold filling under the permutation breakpoint distance. While the details are a bit long, the general idea is in fact simple. We first identify the maximal sequences from A and B respectively such that they (i.e., $A-Y$ and $B-X$) have the same gene content. Then we fill the missing genes between the common adjacencies in $A-Y$ and $B-X$ (the correctness follows from Lemma 1). Thirdly, we fill the missing genes between the breakpoints in $A-Y$ and $B-X$ (the correctness follows Lemmas 2-3). Finally, we insert the remaining missing genes into the current solution, always maintaining the number of breakpoints and their order in the original input. The proofs and the detailed algorithm are presented as follows. Note that at the end of each of these steps the intermediate permutations obtained are related.

Lemma 1: Given two incomplete permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is an adjacency $a_i a_{i+1}$ in the two related permutations $A - Y$ and $B - X$, then there exists an optimal scaffold filling such that every two consecutive elements between a_i and a_{i+1} (in the optimal A^* and B^*) form an adjacency.

Proof: To obtain the optimal $A^* \in A + X$ and $B^* \in B + Y$, we need to insert the elements in X into A and insert the elements in Y into B respectively. In other words, we need to insert $X \cup Y$ to both $A - Y$ and $B - X$. Let $y = \langle y_1, y_2, \ldots, y_r \rangle$ be the subpermutation (i.e., substring) between a_i and a_{i+1} in A, and let $x = \langle x_1, x_2, \ldots, x_l \rangle$ be the subpermutation between a_i and a_{i+1} in B. Obviously, $c(y) \subseteq Y$ and $c(x) \subseteq X$. We can construct two new subpermutations $p = \langle y_1, \ldots, y_r, x_1, \ldots, x_l \rangle$ and $\bar{p} = \langle x_l, \ldots, x_1, y_r, \ldots, y_1 \rangle$. Then we insert p between a_i and a_{i+1} in A, and insert p between a_i and a_{i+1} in B if $a_i a_{i+1}$ appears in B or insert $\bar{p}$ between a_{i+1} and a_i in B if $a_{i+1} a_i$ appears in B. As $a_i a_{i+1}$ is an adjacency in $A-Y$ and $B-X$, all the elements between a_i and a_{i+1} (in both $A+c(x)$ and $B+c(y)$) form adjacencies. For all adjacencies, their corresponding new subpermutations (constructed as above) are mutually disjoint. So there exists an optimal scaffold filling such that there is no breakpoint between the two elements of any adjacency in $A - Y$ and $B - X$. In other words, as $a_i a_{i+1}$ is an adjacency in $A - Y$ and $B - X$, we have the full freedom to insert the respective elements in $X \cup Y$ between a_i and a_{i+1} such that they all form adjacencies in A^* and B^*. ■

Lemma 2: Given two permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is a breakpoint $b_i b_{i+1}$ in $A - Y$, then in any scaffold filling, there is at least one breakpoint between b_i and b_{i+1} in $A' \in A + X$.

Proof: $A' \in A + X$ and $B' \in B + Y$ can be obtained by inserting the elements in $X \cup Y$ into $A - Y$ and $B - X$ respectively. When we insert some subpermutation $p = \langle y_1, y_2, \ldots, y_r \rangle$ between b_i and b_{i+1} in $A - Y$, where $c(p) \subseteq (X \cup Y)$, if p contains a breakpoint then the lemma is proven. If p does not contain any breakpoint, that means p also appears as a subpermutation in B'. Since b_i and b_{i+1} are not adjacent in $B - X$, $b_i y_1$ and $y_r b_{i+1}$ cannot be adjacencies at the same time. Then at least one of $b_i y_1$ and $y_r b_{i+1}$ is a breakpoint. ■

Lemma 3: Given two permutations A and B, let $X = c(B) - c(A)$ and $Y = c(A) - c(B)$ be the sets of elements to be filled into A and B respectively. If there is a breakpoint $b_i b_{i+1}$ in $A - Y$, then there exists an optimal scaffold filling such that there is only one breakpoint between b_i and b_{i+1} in A^*.

Proof: As shown in the previous lemma, we insert the elements in $X \cup Y$ into $A-Y$ and $B-X$ to obtain the optimal A^* and B^* respectively. If there is no element between b_i and b_{i+1} in A, then we will not insert any element between b_i and b_{i+1}, hence the lemma follows. If there is a subpermutation $p = \langle y_1, y_2, \ldots, y_r \rangle$ between b_i and b_{i+1} in A, where $c(p) \subseteq (X \cup Y)$, then we insert p to $A - Y$ at its original position. Recall that $\bar{p} = \langle y_r, y_{r-1}, \ldots, y_1 \rangle$. We have four ways to insert elements of p into $B - X$:

1) Insert p right after b_i in $B - X$, which means $y_r b_{i+1}$ is a breakpoint in A^*,
2) Insert $\bar{p}$ right before b_i in $B - X$, which means $y_r b_{i+1}$

is a breakpoint in A^*,

3) Insert p right before b_{i+1} in $B-X$, which means b_iy_1 is a breakpoint in A^*,
4) Insert $\bar{p}$ right after b_{i+1} in $B-X$, which means b_iy_1 is a breakpoint in A^*.

WLOG, let $B-X=\langle\cdots,b_k,b_i,b_l,\cdots,b_s,b_{i+1},b_t,\cdots\rangle$. Then, following the above four procedures, we generate the new breakpoints in B^* accordingly: (1) y_rb_l, (2) b_ky_r, (3) b_sy_1, and (4) y_1b_t. ■

Algorithm *Minimum Breakpoint Scaffold Filling*
Input: *Two incomplete permutations A and B, with X=c(B)-c(A) and Y=c(A)-c(B)*
Output: *Two permutations* $A^*\in A+X$ *and* $B^*\in B+Y$

1 Delete the elements in $X\cup Y$ from A and B.
2 Compute the breakpoints and adjacencies of $A-Y$ and $B-X$.
3 For each adjacency a_ia_{i+1} in $A-Y$,
 3.1 Let $y=\langle y_1,y_2,\ldots,y_r\rangle$ be the subpermutation between a_i and a_{i+1} in A, let $x=\langle x_1,x_2,\ldots,x_l\rangle$ be the subpermutation between a_i and a_{i+1} in B.
 3.2 Construct new subpermutations $p=\langle y_1,\ldots,y_r,x_1,\ldots,x_l\rangle$, $\bar{p}=\langle x_l,\ldots,x_1,y_r,\ldots,y_1\rangle$.
 3.3 Insert p into $A-Y$ between a_i and a_{i+1};
 3.4 Insert p into $B-X$ between a_i and a_{i+1} if a_ia_{i+1} is a 2-substring in $B-X$ or insert $\bar{p}$ into $B-X$ between a_{i+1} and a_i.
4 Let the filled $A-Y$ and $B-X$ after Step 3 be A_1 and B_1 respectively.
5 For each breakpoint b_ib_{i+1} in A_1,
 5.1 Let $q=\langle y_1,y_2,\ldots,y_s\rangle$ be the subpermutation between b_i and b_{i+1} in A, let $\bar{q}=\langle y_s,y_{s-1},\ldots,y_1\rangle$.
 5.2 Insert q into A_1 between b_i and b_{i+1}.
 5.3 Assume that B_1 has the form $\langle\cdots,b_k,b_i,b_l,\cdots\rangle$ (resp. $\langle\cdots,b_k,b_{i+1},b_l,\cdots\rangle$),
 5.4 Insert q right after b_i (resp. right before b_{i+1}) into B_1, if b_ib_l (resp. b_kb_{i+1}) forms a breakpoint;
 5.5 Or insert $\bar{q}$ right before b_i (resp. right after b_{i+1}) in B_1, if b_kb_i (resp. $b_{i+1}b_l$) forms a breakpoint.
6 Let the filled A_1 and B_1 after Step 5 be A_2 and B_2 respectively.
7 For each 2-substring c_ic_{i+1} in A (resp. B) with exactly one of $\{c_i,c_{i+1}\}$ in the current B_2 (resp. A_2),
 7.1 If $c_i\in B_2$ (resp. $\in A_2$),
  7.1.1 Insert c_{i+1} after c_i in A_2 (resp. B_2);
  7.1.2 Insert c_{i+1} after c_i in B_2 (resp. A_2) if this does not increase bd(A_2,B_2);
  7.1.3 Or insert c_{i+1} before c_i in B_2 (resp. A_2) if this does not increase bd(A_2,B_2).
 7.2 If $c_{i+1}\in B_2$ (resp. $\in A_2$),
  7.2.1 Insert c_i before c_{i+1} in A_2 (resp. B_2);
  7.2.2 Insert c_i before c_{i+1} in B_2 (resp. A_2) if this does not increase bd(A_2,B_2);
  7.2.3 Or insert c_i after c_{i+1} in B_2 (resp. A_2) if this does not increase bd(A_2,B_2).
8 Return the filled A_2 and B_2 as A^* and B^*.

Theorem 1: Given two permutations A and B, let $X=c(B)-c(A)$ and $Y=c(A)-c(B)$ be the sets of elements to be filled into A and B respectively. Then the optimal permutations $A^*\in A+X, B^*\in B+Y$ with $bd(A-Y,B-X)=bd(A^*,B^*)$ can be computed in $O(n^2)$ time.

Proof: Following Lemmas 1-3 and the algorithm, it is easy to see that $bd(A-Y,B-X)=bd(A^*,B^*)$ as at Step 2, 4 and 6 the related incomplete permutations obtained all maintain the number of breakpoints. Finally, we work on the permutations A_2,B_2 thus obtained at Step 7. The idea is to insert the remaining missing letters, which does not belong to $A-Y$ and $B-X$, one by one into A_2 and B_2. Moreover, the newly filled A_2 and B_2 keep to be related, $bd(A_2,B_2)$ does not change and the inserted letters maintain their order as in the input A and B. At the end we have A^* and B^*. Therefore, in this whole process of inserting elements we always maintain the number of breakpoints. The quadratic running time is dominated by maintaining the correspondence between the identical characters in A and B, i.e., after each insertion, a letter and its location in A and B can be retrieved in $O(1)$ time, and the content of a location in A and B can be accessed in $O(1)$ time. ■

An example of the above algorithm is as follows.

$$A=acefh, B=adbhge$$
$$X=\{d,b,g\}, Y=\{c,f\}$$
$$A-Y=aeh, B-X=ahe, bd(A-Y,B-X)=1$$
$$A_1=aefgh, B_1=ahgfe\ (Step\ 4)$$
$$A_2=acefgh, B_2=achgfe\ (Step\ 6)$$
$$A^*=adbcefgh, B^*=adbchgfe, bd(A^*,B^*)=1.$$

We comment that our algorithm also works when in A and B there are predefined blocks one could not break, as long as these blocks have no conflict. For example, we have $A=\cdots\boxed{\text{ac}}\cdots\boxed{\text{gh}}\cdots$ and $B=\cdots\boxed{\text{ea}}\cdots\boxed{\text{fg}}\cdots$, with predefined blocks in boxes. When we fill e,f into A and c,h into B, the algorithm still works as the predefined blocks are not in conflict. However, if $A=\cdots\boxed{\text{bac}}\cdots\boxed{\text{ghf}}\cdots$ and $B=\cdots\boxed{\text{bae}}\cdots\boxed{\text{gfh}}\cdots$, then our result does not hold anymore. In [10], this is related to inserting missing genes only in between contigs (i.e., not anywhere), which is a problem needing further study.

IV. Hardness and Approximation for SF-MNSA

A. Hardness of SF-MNSA

In this section, we first prove that SF-MNSA is NP-hard; in fact, even the One-sided SF-MNSA problem is NP-hard.

It is easy to see that the decision version of One-sided SF-MNSA is in NP. We try to make a reduction from the NP-complete problem Exact Cover by 3-Sets (X3C) [8]. Recall that the input for X3C is a set of 3-sets $S=\{S_1,S_2,...,S_m\}$. Each set S_i contains exactly 3 elements from a base set $X=\{x_1,x_2,...,x_n\}$, where $n=3q$ for some integer q. The problem is to decide whether there are q 3-sets in S which cover each element in X exactly once.

The main idea of our proof is to first show that a special case of X3C, X3C-1, is NP-complete. X3C-1 has the property that each 3-set S_u can share at most one element with any other 3-set S_v. Then we reduce X3C-1 to One-sided SF-MNSA by constructing a complete genome G and an incomplete genome I; moreover, before the missing genes are inserted into I, there is no adjacency between G and I.

Lemma 4: X3C-1 is NP-complete.

Proof: Again, it is easy to show that X3C-1 is in NP, so we focus on reducing X3C to X3C-1. Given an X3C instance with $3q$ elements and m 3-sets, we construct an instance of X3C-1 with $3q+6m$ elements and $5m$ 3-sets. Assume that

we are given a 3-set $S_u = \{x_i, x_j, x_k\}$, we construct five 3-sets as follows: $S'_u = \{ \{x_i, y_{u,1}, y_{u,2}\}, \{x_j, y_{u,3}, y_{u,4}\}, \{x_k, y_{u,5}, y_{u,6}\}, \{y_{u,1}, y_{u,3}, y_{u,5}\}, \{y_{u,2}, y_{u,4}, y_{u,6}\} \}$. Now in the new set of 3-sets $S' = \cup_u S'_u$ for all u, we have $5m$ 3-sets. Obviously, any pair of 3-sets in S' share at most one element. If $S_u = \{x_i, x_j, x_k\}$ is selected for a solution for S then we select $\{x_i, y_{u,1}, y_{u,2}\}$, $\{x_j, y_{u,3}, y_{u,4}\}$, and $\{x_k, y_{u,5}, y_{u,6}\}$ as a part of solution for S'. If $S_u = \{x_i, x_j, x_k\}$ is not selected for a solution for S then we select $\{y_{u,1}, y_{u,3}, y_{u,5}\}$ and $\{y_{u,2}, y_{u,4}, y_{u,6}\}$.

If the input X3C instance has a solution, one can easily use the selected q 3-sets (from S) to obtain $q+2m$ 3-sets to cover the $3q + 6m$ elements contained in S'. Now assume that S' has an exact cover of size $q + 2m$. First, the base set contains $3q + 6m$ elements. By the construction of S'_u, if a solution other than $\{\{x_i, y_{u,1}, y_{u,2}\}, \{x_j, y_{u,3}, y_{u,4}\}, \{x_k, y_{u,5}, y_{u,6}\}\}$ or $\{\{y_{u,1}, y_{u,3}, y_{u,5}\}, \{y_{u,2}, y_{u,4}, y_{u,6}\}\}$ is selected to cover elements $y_{u,j}, j = 1..6$, then either we have to cover some elements more than once or we fail to cover all of the $3q + 6m$ elements. Then, clearly the $q + 2m$ 3-sets selected for S' implies an exact cover of size q for S.

Therefore, we can conclude that the given X3C instance has a solution of size q iff the constructed X3C-1 instance has a solution of size $q + 2m$. It is easy to see that this reduction takes polynomial time. So the lemma is proven. ■

Theorem 2: SF-MNSA is NP-hard and its decision version is NP-complete.

Proof: Following Lemma 4, we reduce X3C-1 to One-sided SF-MNSA. Let the instance of X3C-1 be $S = \{S_1, S_2, ..., S_m\}$. Each set S_i contains exactly 3 elements from a base set $Y = \{y_1, y_2, ..., y_n\}$, where $n = 3q$ for some integer q and $|S_u \cap S_v| \leq 1$ for all u, v. The problem is to decide whether there are q 3-sets in S which cover each element in Y exactly once.

Without loss of generality, we assume that both m and $3m - 3q$ are even, and the elements in each 3-set $S_u = \{y_{u1}, y_{u2}, y_{u3}\}$ are ordered by the indices of the elements, i.e., $u1 \leq u2 \leq u3$. (When one or both of m and $3m-3q$ are odd, one just need to modify the construction by reordering some substrings in G and I, which should be straightforward.) Let

$$T_u = g_u f_u y_{u1} y_{u2} y_{u3} f'_u g'_u$$

and

$$M_u = g'_u f_u f'_u g_u.$$

Let y_i appear in all 3-sets of S for a total of n_i times (so $\sum_i n_i = 3m$). For each i, we rename each of the $n_i - 1$ copies of y_i as a new letter z_j (for the ease of description), we hence have a total of $3m-3q$ newly named z_j letters which are really the original elements in Y. (Note that z_j could be empty — if the corresponding y_i appears exactly once.) We also use $3m-3q$ triples of separators $p_i, p'_i, r_i, i = 1..3m-3q$, with

$$V_i = p_i p'_i.$$

We construct two sequences G, I as follows.
$G = r_1 r_2 \cdots r_{3m-3q} T_1 T_2 \cdots T_{m-1} T_m$
$V_1 V_2 \cdots V_{3m-3q-1} V_{3m-3q}$.
$I = z_1 r_1 z_2 r_2 \cdots z_{3m-3q} r_{3m-3q} p_1 p_2 \cdots p_{3m-3q-1} p_{3m-3q}$
$p'_1 p'_2 \cdots p'_{3m-3q-1} p'_{3m-3q}$
$M_1 M_3 \cdots M_{m-1} M_2 M_4 \cdots M_m$.

Note that in G and I, except for y_i's (including the renamed z_j's), all other letters are used as separators. Also, there is no adjacency between G and I. Moreover, in the incomplete sequence I, each y_i is missed exactly once as $|G| = 16m-9q$ and $|I| = 16m - 12q$. We claim that there is a solution for the X3C-1 instance iff the maximum number of adjacencies generated by inserting these missing y_i's back into I is $4q$. Among all separators, only some f_j or f'_j can form some adjacencies with some y_i in the filled I.

'$\rightarrow$' It is easy to see that if X3C-1 has a solution then we can obtain $4q$ adjacencies between G and I'. To obtain $4q$ of adjacencies between G and I', we simply insert a triple of elements (y_i, y_j, y_k) such that they are inserted back between f_u and f'_u in M_u, with $S_u = \{y_i, y_j, y_k\}$ being in the X3C-1 solution.

'$\leftarrow$' Suppose that one can insert the missing y_i's back into I to obtain I' such that there are $4q$ adjacencies between G and I', first notice that due to the construction of G and I, y_i's have to be inserted in M_j's, as inserting three missing y_i's anywhere before M_1 can only generate at most 3 adjacencies. Second, due to that any pair of 3-sets have at most one common element, we cannot insert two y_i's in some M_j and one y_i in some M_k, as that can only generate at most 3 adjacencies. Then, to insert three missing y_i's (say (y_i, y_j, y_k), ordered by their indices) between f_l and f'_l in M_l to generate 4 adjacencies, we must have $T_l = g_l f_l y_i y_j y_k f'_l g'_l$, or, in other words, $S_l = \{y_i, y_j, y_k\}$. This results in a solution for X3C-1.

It is easy to see that this construction takes polynomial time. As One-sided SF-MNSA is a special case of SF-MNFA, the theorem is proven. ■

We show a simple example for the reduction. $Y = \{1, 2, 3, 4, 5, 6, a, b, c, d, e, f\}$. $S_1 = \{1, 2, 3\}$, $S_2 = \{1, 5, a\}$, $S_3 = \{4, 5, 6\}$, $S_4 = \{a, b, c\}$, $S_5 = \{4, b, e\}$, $S_6 = \{d, e, f\}$. So $m = 6, q = 4$. The six elements $1, 4, 5, a, b, e$ appear twice in S, so we can rename them as $z_1 = 1$, $z_2 = 4$, $z_3 = 5$, $z_4 = a$, $z_5 = b$, and $z_6 = e$. We now have
$G = r_1 r_2 r_3 r_4 r_5 r_6$
$g_1 f_1 123 f'_1 g'_1 \cdot g_2 f_2 15a f'_2 g'_2 \cdot g_3 f_3 456 f'_3 g'_3$
$g_4 f_4 abc f'_4 g'_4 \cdot g_5 f_5 4be f'_5 g'_5 \cdot g_6 f_6 def f'_6 g'_6$
$p_1 p'_1 p_2 p'_2 p_3 p'_3 p_4 p'_4 p_5 p'_5 p_6 p'_6$.

$I = 1 r_1 4 r_2 5 r_3 a r_4 b r_5 e r_6$
$p_1 p_2 p_3 p_4 p_5 p_6 \cdot p'_1 p'_2 p'_3 p'_4 p'_5 p'_6$
$g'_1 f_1 f'_1 g_1 \cdot g'_2 f_2 f'_2 g_2 \cdot g'_3 f_3 f'_3 g_3$
$g'_4 f_4 f'_4 g_4 \cdot g'_5 f_5 f'_5 g_5 \cdot g'_6 f_6 f'_6 g_6$.

The optimal solution is to insert the 12 elements in S_1, S_3, S_4, S_6 into I to obtain $4q = 16$ adjacencies: $f_1 123 f'_1$, $f_3 456 f'_3$, $f_4 abc f'_4$, and $f_6 def f'_6$.

In the next subsection, we present some approximation algorithms for SF-MNSA.

B. Approximation Algorithms for SF-MNSA

We first show some properties for SF-MNSA, which implies an easy 2-approximation for SF-MNSA. Then we try to

improve the approximation factor to 1.33 for the One-Sided SF-MNSA problem. For both of the problems, there might be some existing adjacencies before any filling. It is in fact easy to show that for both SF-MNSA and its one-sided version there exists an optimal solution which does not insert missing genes in existing adjacencies — unless between the filled genomes G'_1 and G'_2 (with length n) there are $n-1$ adjacencies (i.e., no breakpoint), which can be easily checked in polynomial time. WLOG, assume that in the unfilled genomes G_1, G_2 we have an adjacency ab, and in the optimally filled genomes G'_1, G'_2 we have some breakpoint, say xy and xz (in G'_1, G'_2) respectively. If some string $s_1 s_2 \cdots s_m$ is inserted between a and b in G'_1, G'_2 (to have $m+1$ adjacencies), one can move $s_1 s_2 \cdots s_m$ after x's to have $xs_1 s_2 \cdots s_m y$ and $xs_1 s_2 \cdots s_m z$ in G'_1, G'_2 respectively. Apparently we have kept the original adjacency ab as well as the total number of adjacencies in the (new) optimal solution.

Henceforth, we only consider the case when there is some breakpoint in the filled genomes G'_1 and G'_2. Following the above discussion, our algorithms will not alter these existing adjacencies and will only focus on generating the maximum number of *new* adjacencies. This method is further supported by the following folklore lemma.

Lemma 5: Let Π be a maximization problem with an optimal solution value OPT. If $OPT = O_1^* + O_2^*$ with O_1^* being polynomially computable, then an algorithm $\mathcal{A}$ which does not alter O_1^* and which approximates O_2^* with an approximation ratio of c (> 1) is also a factor-c approximation for Π.

Proof: By the assumption, algorithm $\mathcal{A}$ approximates O_2^* with a solution value of O_2, with

$$O_2 \geq O_2^*/c.$$

As O_1^* is polynomially computable and algorithm $\mathcal{A}$ does not alter O_1^*, $\mathcal{A}$ can return a solution of value $O_1^* + O_2$ for Π. Apparently, this is a factor-c approximation for Π, as we have

$$O_1^* + O_2 \geq O_1^* + O_2^*/c \geq (O_1^* + O_2^*)/c = OPT/c.$$

■

In the remaining part of this section, we can fix the existing adjacencies between two sequences G_1 and G_2 by first computing a bipartite graph H. The vertices of H are the 2-substrings in G_1 and G_2, there is an edge between a pair of 2-substrings (U, V), with U in G_1 and V in G_2 respectively, if U and V form a common adjacency. Then, the existing common adjacencies can be fixed by computing a maximal matching in H. In other words, if (U, V) is in the computed maximal matching, then label the 2-substrings (2-blocks) U and V. We will never insert any letter or gene into a labelled 2-block, i.e., once labelled, a 2-block will not be altered. We call this polynomial-time procedure *block-labeling*.

1) A 2-Approximation Algorithm for SF-MNSA:

Lemma 6: Suppose that O^* is the optimal number of newly created adjacencies for SF-MNSA (after the block-labeling procedure), then we need to insert at least $\lceil O^*/2 \rceil$ and at most O^* genes into G_1 and G_2 when $C(G_1) \cap C(G_2) \neq \emptyset$.

Proof: First, notice that when a missing gene x is inserted into G_1 or G_2, at most two adjacencies can be formed. To complete the proof of the lemma, all we need to show is to insert one missing gene into G_1 or G_2 to generate, on average, at least one (new) adjacency. This can be easily shown in a constructive and greedy fashion: After the block-labeling process, for any unlabelled 2-substring $b_i b_{i+1}$ (say in G_1), if one can insert m missing genes to obtain at least m adjacencies, say $b_i z_1 z_2 ... z_m$ or $z_1 z_2 ... z_m b_{i+1}$ or $b_i z_1 z_2 ... z_m b_{i+1}$, then insert $z_1 z_2 ... z_m$ (which is an unlabelled substring in G_2) in between b_i and b_{i+1} (in G_1) and label the corresponding m or $m+1$ new adjacencies (2-blocks) in the current G_1 and G_2. Notice that b_i (resp. b_{i+1}) could be empty when b_{i+1} (resp. b_i) is the leftmost (resp. rightmost) letter in G_1 or G_2. ■

From the above lemma, it is easily seen that the optimal solutions are obtained in a way, (1) either a sequence of m missing genes $z_1, z_2, ..., z_m$ are inserted to obtain $m+1$ adjacencies (say $y_1 z_1 z_2 \cdots z_m y_2$), (2) or a sequence of l missing genes $x_1, x_2, ..., x_l$ are inserted to form l adjacencies (say $x_1 x_2 \cdots x_l y$). Note that $m, l \geq 1$. We call the corresponding missing genes (and the inserted substrings) type-1 and type-2 respectively. If a type-1 inserted substring contains m genes, then it is also called a *type-1 m-substring*.

The above lemma also implies a factor-2 approximation for SF-MNSA, as the greedy algorithm in the proof of Lemma 6 could make all the inserted missing genes type-2 in the worst case. We next show how to improve this factor to 1.33 for the One-sided SF-MNSA problem.

2) A 1.33-Approximation Algorithm for One-sided SF-MNSA: Recall that in the One-sided SF-MNSA problem, we have input I and G, and we need to insert the missing genes into I. We first run the block-labeling procedure on G, I. Then, let k_1 be the number of missing genes inserted. (By Lemma 6, $\lceil O^*/2 \rceil \leq k_1 \leq O^*$.) Let b_i be the number of type-1 i-substrings in the optimal solution. (Example, $I = abcd$, $G = acbacd$, and we need to insert two type-1 1-substrings: a and c. Four new adjacencies are formed, and $b_1 = 2$.) Then we have,

Lemma 7: Suppose that O^* is the optimal number of newly created adjacencies for One-sided SF-MNSA (after the block-labeling procedure), then

$$O^* = k_1 + b_1 + b_2 + \cdots + b_q \leq \frac{4}{3}(k_1 + b_1/2 + b_2/4).$$

Proof: By definition, $O^* = k_1 + b_1 + b_2 + \cdots + b_q$. As each inserted substring counted in b_j has j genes for $j \geq 3$ and following Lemma 6, $b_3 + b_4 + \cdots b_q \leq \frac{1}{3}(k_1 - b_1 - 2b_2)$. Hence,

$$O^* \leq \frac{4}{3}(k_1 + b_1/2 + b_2/4).$$

■

From Lemma 7, it is easy to see that our algorithm hinges on approximating $b_1/2 + b_2/4$, which uses a greedy idea. The algorithm goes as follows. At Step 1, for each missing gene a to be inserted into I, we use a greedy method to scan from left to right to find an (unlabelled) substring cd in I such that the insertion of a results in a substring cad which contains two adjacencies $\langle ca \rangle$ and $\langle ad \rangle$. At Step 2, for each pair of missing genes x, y to be inserted into I, we scan in I, again left to right, to find whether there is an (unlabelled) substring wz such that x, y can be inserted into it to obtain three adjacencies, i.e.,

either $\langle wxyz\rangle$ or $\langle wyxz\rangle$. We insert all such pairs of missing genes, in a greedy fashion, into I. At Step 3, we insert the remaining missing genes in greedy fashion into the unlabelled parts of I, provided that each inserted missing gene generates one adjacency — similar to Lemma 6.

We have the following lemma regarding this greedy algorithm.

Lemma 8: Let b'_1, b'_2 be the number of new type-1 1-substrings and 2-substrings inserted at Step 1 and Step 2 of our greedy algorithm respectively. Then $b'_1 + b'_2 \geq \frac{b_1}{2} + \frac{b_2}{4}$.

Proof: Let k'_1, k'_2 be the number of missing genes inserted at Step 1 and Step 2 respectively. (So $b'_1 = k'_1$ and $b'_2 = k'_2/2$.) First, note that each of the k'_1 inserted missing genes can destroy at most two type-1 1-substrings in some optimal solution. (This is simply due to the fact that such an inserted gene x, which should be inserted optimally to have axb, is inserted at a wrong place to obtain cxd while ab is possibly filled with some other letter, say z. The optimal adjacencies could have been axb and cyd.) Also, each of the k'_1 inserted missing genes can destroy at most two type-1 2-substrings in some optimal solution, this will be illustrated with an example at the end of this paragraph. Let b'_{10} be the number of missing genes inserted at Step 1 which destroy exactly one type-1 1-substring (and some type-1 m-substring, with $m \geq 3$) in some optimal solution. Let b'_{11} be the number of missing genes inserted at Step 1 which destroy exactly two type-1 1-substrings in some optimal solution. Let b'_{12} be the number of missing genes inserted at Step 1 which destroy one type-1 1-substring and one type-1 2-substring in some optimal solution. Let b'_{13} be the number of missing genes inserted at Step 1 which destroy exactly two type-1 2-substrings in some optimal solution. Obviously,

$$k'_1 = b'_1 = b'_{10} + b'_{11} + b'_{12} + b'_{13}.$$

Then, we show an example for a, one of the b'_{13} inserted missing genes that destroy two type-1 2-substrings in the optimal solution (i.e., counted into b_2). Let $G = \cdots\alpha a\beta\cdots\gamma ab\delta\cdots\alpha uv\beta\cdots$ and let $I = \alpha\cdots\alpha\beta\cdots\gamma\delta\cdots\beta\cdots a\cdots$. We need to insert a, b, u, v into I. Due to the greedy fashion of the algorithm, a is inserted between α, β in I to have $\alpha a\beta$ (destroying the possibility of inserting uv at the same location). On the other hand, due to the insertion of a (instead of ab), ab cannot be inserted in between γ and δ. Therefore, we destroy the optimal adjacencies $\langle\alpha uv\beta\rangle$ and $\langle\gamma ab\delta\rangle$ (with the corresponding two type-1 2-substrings: uv and ab).

Next, we need to show that at Step 2, each of the inserted type-1 2-substrings can destroy at most three type-1 2-substrings in some optimal solution. This is again easy to see. Suppose that we need to insert xy into I to obtain three adjacencies $\langle\alpha xy\beta\rangle$. Due to the greedy fashion something else (like uv) are inserted between α, β instead. Then xw and yz could be destroying two other locations for the optimal insertion of type-2 2-substrings.

Now, putting all together,

$$b_1 \leq b'_{10} + 2b'_{11} + b'_{12},$$

and

$$b_2 \leq 3b'_2 + b'_{12} + 2b'_{13}.$$

Then

$$\begin{aligned}\frac{b_1}{2} + \frac{b_2}{4} &\leq \frac{b'_{10} + 2b'_{11} + b'_{12}}{2} + \frac{3b'_2 + b'_{12} + 2b'_{13}}{4} \\ &= (\frac{b'_{10}}{2} + b'_{11} + \frac{3b'_{12}}{4} + \frac{b'_{13}}{2}) + \frac{3b'_2}{4} \\ &\leq b'_1 + b'_2\end{aligned}$$

■

Following Lemma 8 and Lemma 5, we have the following theorem.

Theorem 3: There is a greedy algorithm which approximates One-sided SF-MNSA with a factor of 1.33.

Proof: Following the greedy algorithm and Lemma 8, we have the approximation solution value APP (for O^*), which satisfies the following inequalities.

$$\begin{aligned}APP &= k_1 + b'_1 + b'_2 \\ &\geq k_1 + \frac{b_1}{2} + \frac{b_2}{4} \\ &\geq \frac{3}{4}O^* \\ &= O^*/1.33\end{aligned}$$

Following Lemma 5, the theorem is proven. ■

C. FPT Algorithms for Two Special Cases of One-Sided SF-MNSA

In many biological applications, it makes sense to design FPT algorithms for the corresponding problem. Basically, an FPT algorithm for a decision problem of solution value (parameter) k is one which runs in $O(f(k)n^{O(1)}) = O^*(f(k))$ time, where $f(-)$ is a function only on k and n is the size of the input. More details can be found in [6], [7].

It is unknown whether the One-Sided SF-MNSA problem admits an FPT algorithm, with k being the corresponding optimal solution value. We show below that some restricted cases of One-Sided SF-MNSA are in FPT.

Firstly, in *d-SF-MNSA* each gene in G_1 or G_2 only appears at most d times. So for One-sided d-SF-MNSA each gene in I, G appears at most d times. Then, let us assume that we need to insert an x into I. Certainly, we hope to insert x into I to obtain at least one adjacency xy (or yx). Now let us look at G, as x appears in G at most d times, x has at most $2d$ neighbors (at least one of them should be y). As we have no information on what the right y should be, we have to try over all such (at most) $2d$ possibilities in I for each inserted x. Since we might have to insert a total of k letters, the running time is

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

In *SF-MNSA^c*, each gene is selected from a set of c letters Σ. For One-sided SF-MNSA^c, the algorithm is similar to that of One-sided d-SF-MNSA, with a simple twist. Assume that one needs to insert a missing x into I to obtain some adjacency xy (or yx), one just needs to find the neighbors of x in G. It is easy to see that if we have two substrings xy's in

G then we can match either one of them with the intended adjacency xy (introduced due to the insertion of x in I). As Σ contains only c letters, we could have c choices for the intended adjacencies xy (resp. yx). When we insert x into I, we also have c possibilities to obtain xy. Therefore, we need to try over all such (at most) c^2 possibilities for each inserted x. By Lemma 6, we need to insert at most k genes into I, so the running time of the algorithm is

$$O^*(c^{2k}).$$

Theorem 4: One-sided d-SF-MNSA can be solved in $O^*((2d)^{2k})$ time and One-sided SF-MNSA^c can be solved in $O^*(c^{2k})$ time.

V. Two-sided Scaffold Filling Under the Rearrangement Distance

In this section, we show how to adapt the ideas in Section 3 to solve the two-sided scaffold filling problem under the rearrangement distance, when the genomes are multichromosomal and the genes are signed.

Rearrangement distance

The *rearrangement distance* or *genomic distance* $D(G_1, G_2)$ is a metric counting the number of genomic rearrangement operations necessary to transform one signed multichromosomal genome G_1 containing n distinct genes into another, G_2. The $+$ or $-$ sign indicates the "reading direction" of a gene, left-to-right or right-to-left.

For a comprehensive repertoire of operations, which we need not elaborate here, Yancopoulos *et al.* [15] showed that D could be calculated efficiently using the *breakpoint graph* [13] of G_1 and G_2 as follows:

1) Replace each positively-signed gene g on a chromosome by the vertex pair g_t, g_h; replace a negative $-g$ by g_h, g_t.
2) Each pair of successive genes in the genome defines one edge connecting the pair of vertices that are adjacent in the vertex order. E.g., if $i\ j\ -k$ are neighboring genes on a chromosome then the two edges they define are $\{i_h, j_t\}$ and $\{j_h, k_h\}$. This leaves two unconnected vertices at the ends of each chromosome. Define an edge incident to each such vertex in genome G_1 and G_2 connecting it to a new vertex, all labelled T_1 in G_1 and T_2 in G_2.
3) Color the edges of G_1 and G_2 blue and red, respectively.
4) Identify (i.e., superimpose) each vertex in G_1 with the identically labelled vertex in G_2.
5) Make a cycle of any path ending in two T_1 or two T_2 vertices, connecting them by a red or blue edge, respectively, while for a path ending in a T_1 and a T_2, collapse them to form one T vertex.
6) Each vertex is now incident to one blue and one red edge. This bicolored graph, the breakpoint graph, decomposes uniquely into κ alternating cycles. If n' is the number of blue edges, $D(G_1, G_2) = n' - \kappa$ [15] and the optimizing rearrangements are rapidly recovered by operations on the graph.

Bundles

Now consider the case where the genes in G_2 are a subset of the genes in G_1. We say some genes are *missing* from G_2. The one-sided scaffold filling problem is to insert the missing genes in G_2, thus forming $\bar{G}_2$, in such a way as to minimize $D(G_1, \bar{G}_2)$. In [10], it was shown that the one-sided scaffold filling problem under the rearrangement distance can be solved in polynomial time, in fact, in linear time once the breakpoint graph is constructed.

We can still construct the breakpoint graph, except that some vertices, called *free ends*, will only be incident to a blue edge and thus paths in the graph can end not only in T vertices but also in free ends. When this happens, step 5 in the breakpoint graph construction cannot be completed, and the decomposition and calculation in step 6 are blocked.

A *bundle* is a subset of the paths in this partial breakpoint graph of G_1 and G_2. (Partial because some paths are not cycles nor do they end in a T.) Each bundle is associated with one or more of the missing genes. The vertices corresponding to each missing gene, its free ends, must be in the same bundle and must be endpoints of one or two paths. To simplify the exposition, we assume that no bundle consists entirely of blue edges, i.e., no chromosome in G_1 has all its genes absent from G_2. This case is easily handled separately, and does not affect the distance calculations.

To construct a bundle, we initiate it with any path not already in any bundle and ending with a free end. Then if a path containing free end g_t is in a bundle B, then we also include the path with g_h as a free end, and vice versa. There can be zero or two T vertices in a bundle. We now present the details on the two-sided scaffold filling problem under the rearrangement distance. It is not hard to show the following lemma.

Lemma 9: If genomes G_1 and G_2 both contain the same n genes, and if $m \geq 1$ genes are inserted anywhere into both G_1 and G_2 to get $\bar{G}_1$ and $\bar{G}_2$, then $D(\bar{G}_1, \bar{G}_2) \geq D(G_1, G_2)$.

In a one-sided scaffold filling problem for G_1 and G_2, suppose there are r cycles in the partial breakpoint graph and suppose this graph determines β bundles. Let $\underline{G}_1$ be the genome formed by deleting from G_1 the genes already missing from G_2.

Lemma 10: Let κ be the number of cycles in the breakpoint graph of $\underline{G}_1$ and G_2. Then $\kappa = \beta + r$.

Proof: If a_t and a_h are a pair of free ends in a bundle, incident to blue edges (a_t, x) and (a_h, y), remove a_t and a_h and replace (a_t, x) and (a_h, y) by (x, y). Repeat until there are no more free ends in the bundle. This process converts a bundle into a cycle. Repeated across all bundles it also removes all the missing genes. Therefore the number of cycles in the partial breakpoint graph plus the number of bundles determined by this graph equals the number of cycles in the breakpoint graph of $\underline{G}_1$ and G_2. ■

It was shown in [10] how the one-sided scaffold filling problem can be solved by *completing* each bundle separately, i.e., by inserting the missing genes or drawing the red edges between free ends within each bundle. It turns out that we have the following theorem [10].

Theorem 5: For a bundle with ν paths, there are $\nu+1$ cycles produced by completing the bundle, while ν genes inserted in G_2.

The following corollary follows immediately.

Corollary 1: After completing all the bundles with $(\nu_1, \nu_2, \cdots, \nu_\beta)$ paths, there are $m = \nu_1 + \nu_2 + \cdots + \nu_\beta$ genes inserted and the number of cycles increases by $m + \beta$.

Therefore, we have the following main result.

Theorem 6: $D(G_1, \bar{G}_2) = D(\underline{G}_1, G_2)$.

Proof: The breakpoint graph of G_1 and $\bar{G}_2$ has m more blue edges than the breakpoint graph of $\underline{G}_1$ and G_2, corresponding to the insertion of the m missing genes. (No chromosome in G_1 has all its genes absent from G_2, this is why we have m more blue edges.) But since it had r cycles before bundle completion, the breakpoint graph of G_1 and $\bar{G}_2$ has $r + m + \beta$ cycles, from Corollary 1. This is m more than the $r + \beta$ cycles in the breakpoint graph of $\underline{G}_1$ and G_2 (Lemma 9). By the definition of distance D, we have $D(G_1, \bar{G}_2) = D(\underline{G}_1, G_2)$. ■

For two-sided scaffold filling, we thus have the following exact algorithm for the case when G_1 contains genes $G + X$ and G_2 contains genes $G + Y$, where G, X and Y are disjoint sets of genes.

1) Remove all the genes in set X from G_1 to get G_1', containing only the genes in G.
2) Apply one-sided scaffold filling to G_2 and G_1', inserting all the genes in Y into G_1', producing genome G_1''.
3) Restore genes in set X to G_1''. The insertion points are only constrained by the gene order in G_1. The new genome, G_1''' contains all the genes in $G, X,$ and Y.
4) Apply one-sided scaffold filling to G_1''' and G_2, inserting all the genes in X into G_2.

Because the distance between the two genomes reduced to the genes in G is a lower bound for the distance between any two genomes enlarged through gene insertion, by Lemma 8, and because steps 2 and 4 do not increase this distance, by Theorem 6, and because step 3 is as general as possible while respecting the gene order of G_1, the correctness of the algorithm is verified.

As for the one-sided scaffold filling, once the breakpoint graph is constructed, the remaining steps run in linear time.

VI. Concluding Remarks

In this paper, we investigate the scaffold filling problems under the breakpoint distance (and related similarity measures). A very interesting open problem is when the missing genes can be only inserted in between contigs (i.e., in some predefined locations), as was solved for the one-sided problem by Muñoz *et al.* [10]; our current method cannot generate any result with some performance guarantee. The second interesting open problem is whether one can improve the factor-2 approximation for SF-MNSA. The final open problem is whether an FPT algorithm only parameterized on k exists for the (One-sided) SF-MNSA problem.

VII. Acknowledgments

This research is partially supported by NSF under grant DMS-0918034, by NSF of China under grant 60928006, by NSERC of Canada, by China Postdoctoral Science Foundation under grant 2011M501133 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

[1] G. Blin, G. Fertin, F. Sikora, and S. Vialette. The exemplar breakpoint distance for non-trivial genomes cannot be approximated. *Proc. of the 3rd Workshop on Algorithm and Computation (WALCOM'2009)*, LNCS 5431, pages 357–368, 2009.

[2] P. Chain, D. Grafham, R. Fulton, M. Fitzgerald, J. Hostetler, D. Muzny, and J. Ali, et al. Genome project standards in a new era of sequencing. *Science*, 326:236–237, 2009.

[3] 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.

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

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

[6] R. Downey and M. Fellows. *Parameterized Complexity*, Springer-Verlag. 1999.

[7] J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer-Verlag. 2006.

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

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

[10] 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.

[11] A. Rissman, B. Mau, B. Biehl, A. Darling, J. Glasner, and N. Perna. Reordering contigs of draft genomes using the Mauve Aligner. *Bioinformatics*, **25**(16):2071–2073, 2009.

[12] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, **16**(11):909-917, 1999.

[13] G. Tesler. Efficient algorithms for multichromosomal genome rearrangements. *J. Computer and System Sciences*, 65:587–609, 2002.

[14] G. Watterson, W. Ewens, T. Hall, and A. Morgan. The chromosome inversion problem. *J. Theoretical Biology*, **99**:1-7, 1982.

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