A 1.5-approximation Algorithm For Two-Sided Scaffold Filling

Nan Liu[1,2], Daming Zhu[1], Haitao Jiang[1] and Binhai Zhu[3]

[1] School of Computer Science and Technology, Shandong University, Jinan, China.
[2] School of Computer Science and Technology, Shandong Jianzhu University, Jinan, China.
[3] Department of Computer Science, Montana State University, Bozeman, USA.

ABSTRACT The scaffold filling problem aims to set up the whole genomes by filling those missing genes into the scaffolds to optimize a similarity measure of genomes. A typical and frequently used measure for the similarity of two genomes is the number of common adjacencies. One-sided scaffold filling is given by a scaffold and a whole genome, and asks to fill the missing genes into that scaffold to result in such a genome that the number of common adjacencies between it and the given genome is maximized. Two-sided scaffold filling is given by two scaffolds, and asks to fill the missing genes into those two scaffolds respectively to result in such two genomes that the number of common adjacencies between them is maximized. One-sided scaffold filling can be approximated to $\frac{5}{4}$ by now. However, the algorithmic progress for two-sided scaffold filling seems rare. What we know for two-sided scaffold filling is a 2-approximation algorithm by now. In this paper, we propose a new algorithm for two-sided scaffold filling which can achieve a performance ratio of $\frac{3}{2}$ in $O(N^3)$ time, where N is the number of genes in an output genome. An example can be given to show that the performance ratio $\frac{3}{2}$ for this algorithm is actually tight.

1 Introduction

There has been a trend in increasing the phylogenetic scope of genome sequencing without finishing the sequence of the genome. Increasing numbers of genomes are being published in a scaffold or contig form [6]. The use of draft genomes makes many analysis and interpretations tentative and prone to error, and leads to particular problems in the comparative study of gene order. Many algorithms for studying genome rearrangement require the whole genome data on gene order or syntenic block order. The scaffold filling problem arises out of this situation [11]. It aims to fill the missing gene fragments into the scaffolds in a combinatorial way, so that the 'augmented' genomes can be obtained. An 'augmented' genome can then be applied as the referenced whole genome to genome analysis and interpretations.

Muñoz *et al.* first investigated the one-sided scaffold filling problem for permutations, and proposed an exact algorithm for this problem to minimize the

rearrangement (DCJ) distance [11]. Subsequently, Jiang *et al.* proposed a polynomial time algorithm for the two-sided scaffold filling problem for permutations under the DCJ distance [10].

For genomes with gene repetitions, there are three general criteria to measure the similarity of them: the exemplar genomic distance [12], the minimum common string partition (MCSP) distance [3] and the maximum number of common adjacencies [1, 9]. Unfortunately, unless P=NP, there does not exist any polynomial time approximation algorithm (regardless of the factor) for computing the exemplar genomic distance even when each gene is allowed to repeat three times [5, 4] or even two times [2, 8]. The minimum common string partition problem is NP-complete even if each gene repeats at most twice [7]. By now, the best known performance ratio for the general version of this problem is $O(\log n \log^* n)$ [3].

The number of common adjacencies has been introduced to compare genomes by Angibaud *et al.* [1]. They proposed a combinatorial problem which asks to find a matching between two genomes in order to maximize the number of common adjacencies of them, and designed a 4-approximation algorithm to deal with the balance genomes, where a balance genome has all its gene families with the same number of genes. To aim for the maximum number of common adjacencies, the problem of one-sided scaffold filling was proven to be NP-hard by Jiang, Zhong and Zhu [10]. With a greedy strategy, they also proposed a 1.33-approximation algorithm for this problem [9]. Recently, Liu *et al.* improved the performance ratio to solve this problem to 1.25 by the so-called maximum matching and local improvement technique [13].

Aiming for the maximum number of common adjacencies, the two-sided scaffold filling problem for genomes with gene repetitions is also NP-hard, as one-sided scaffold filling is a special version of it. In [9], the authors provided a 2-approximation algorithm for two-sided scaffold filling, which is the unique algorithmic approach as we know by now.

In this paper, we propose a 1.5-approximation algorithm for two-sided scaffold filling to maximize the number of common adjacencies. Primarily, we can devise a polynomial time algorithm for filling a special type of scaffolds. This algorithm makes it possible to design a simple algorithm, which can always use an insertion of one gene to increase one or more common adjacencies averagely. Since an insertion of one gene can increase at most two common adjacencies, it suffices to achieve the performance ratio 1.5 by greedily finding those genes along with their insertions each of which can increase two common adjacencies. We can give an example for which our algorithm can just achieve the performance ratio 1.5 tightly. An extended abstract of this paper has been presented in the 10th annual international conference on Theory and Applications of Models of Computation (TAMC 2013) [14].

This paper is organized as follows. The necessary definitions and notations are introduced in Section 2. A polynomial time algorithm is presented for a special version of the two-sided scaffold filling problem in Section 3. The approximation algorithm for the two-sided scaffold filling problem with performance ratio 1.5 is presented in Section 4, where a bound has to be derived to evaluate the number of

common adjacencies for those two scaffolds as an optimal solution in Section 4.1; and the 1.5-approximation algorithm is given in Section 4.2. Finally in Section 5, the paper is concluded by considering how to improve the performance of the algorithm.

2 Preliminaries

Let Σ be a set of symbols. Each symbol in Σ represents a *gene family*. A *scaffold* on Σ is a sequence of occurrences of symbols in Σ. A symbol can occur more than once in a scaffold. An occurrence of a symbol in a scaffold is a *gene*. All genes in a scaffold form a multiset. Let $A = a[1]\ ...\ a[n]$ be a scaffold on Σ, then the multiset of A is $\{a[1],\ ...\ ,a[n]\}$ and denoted as $c(A)$. For example, let $\Sigma = \{a,\ b,\ c,\ d\}$, $S = abcdacd$, then the set of genes in S is $c(S) = \{a,\ a,\ b,\ c,\ c,\ d,\ d\}$. We usually do not distinguish a symbol and a gene of that symbol if their meanings in the context are explicit. A subsequence of k genes in a scaffold is referred to as a *k-string* in the scaffold, $k \geq 1$, or a *string* briefly. The number of genes in a string is referred to as the *length* of the string. Two genes are *equal*, if they are both the occurrences of a symbol. We write it $a_1 = a_2$, if a_1 is equal to a_2. Let $I = x_1\ ...\ x_k$ be a string. Then the *reversal* of I is $x_k\ ...\ x_1$ and denoted as $-I$. Two k-strings, say $I_1 = x_1\ ...\ x_k$ and $I_2 = y_1\ ...\ y_k$, are *equal*, if $x_i = y_i$ for $1 \leq i \leq k$. We write it $I_1 = I_2$ if I_1 is equal to I_2. Two strings, say I_1, I_2, are *identical* if $I_1 = I_2$ or $I_1 = -I_2$. Two genes are identical, if and only if they are equal. We denote by $\tau(A)$ the set of all 2-strings in A. Let $A = a[1]\ ...\ a[n]$. Then $\tau(A) = \{\ a[1]a[2],\ ...,\ a[n-1]a[n]\ \}$.

Let A and B be two scaffolds on Σ. If A and B are both permutations over Σ, i.e. $c(A) = c(B) = \Sigma$, then two genes can form a 2-string in A as well as in B for at most once. Thus a 2-string in A is a *common adjacency* relative to B, if it is identical to a 2-string in B, otherwise, a *breakpoint* relative to B. Symmetrically, a 2-string in B is a *common adjacency* relative to A, if it is identical to a 2-string in A, otherwise, a *breakpoint* relative to A. For A and B as generic scaffolds, we have to base a matching between $\tau(A)$ and $\tau(B)$ to identify a 2-string in A or B as a common adjacency or a breakpoint.

Let $\tau(A)$ and $\tau(B)$ be the sets of all 2-strings in A and B respectively. For $\tau_1 \subseteq \tau(A)$, $\tau_2 \subseteq \tau(B)$, we denote by $R(\tau_1, \tau_2) \subseteq \tau_1 \times \tau_2$ the set of pairs of 2-strings in which a 2-string $a[i]a[i+1] \in \tau_1$ pairs with $b[j]b[j+1] \in \tau_2$, if and only if $a[i]a[i+1]$ is identical to $b[j]b[j+1]$. We usually call a pair of 2-strings a *pair* for short. A *maximum match* between τ_1 and τ_2 is a subset of $R(\tau_1, \tau_2)$ which subjects to: (1) any 2-string in τ_1 or τ_2 cannot occur in two or more pairs in the subset; (2) the cardinality of the subset is maximized. Thus the *common adjacency* and *breakpoint* can be formally defined as follows.

Definition 1. *Let M be a maximum matching between $\tau(A)$ and $\tau(B)$. A 2-string in A is a common adjacency relative to B with respect to M, if and only if in M it pairs with a 2-string in B. A 2-string in B is a common adjacency relative to A with respect to M, if and only if in M it pairs with a 2-string in*

A. A 2-string in A (resp. B) is a breakpoint relative to B (resp. A) with respect to M if it is not a common adjacency relative to B (resp. A).

In what follows, for a, b as two arbitrary symbols in Σ, a *string of* ab is a 2-string which contains a gene of a and a gene of b.

Lemma 1. *Let M be a maximum matching between $\tau(A)$ and $\tau(B)$. Then for any two symbols a and b in Σ, there are as many breakpoints (common adjacencies) of ab in A (resp. B) relative to B (resp. A) with respect to M as those in A (resp. B) relative to B (resp. A) with respect to an arbitrary maximum matching between $\tau(A)$ and $\tau(B)$.*

Proof. Let M' be a maximum matching between $\tau(A)$ and $\tau(B)$ other than M. If with respect to M, there is no breakpoint of ab in A relative to B, there must be no more strings of ab in A than in B. Thus with respect to M', there cannot be any breakpoint of ab in A relative to B. That is, the number of common adjacencies of ab in A relative to B with respect to M must be equal to the number of common adjacencies of ab in A relative to B with respect to M'.

If there are k (≥ 1) breakpoints of ab in A relative to B with respect to M, then there are k more strings of ab in A than in B. Thus with respect to M', there must be k breakpoints of ab in A relative to B. □

In what follows, if there is no special need, we usually ignore the maximum matching, and call a 2-string in A a common adjacency or breakpoint relative to B. When there is no confusion, we also call a 2-string a common adjacency or breakpoint in a scaffold without mentioning that relative scaffold. We denote by $\alpha(A, B)$ ($\beta(A, B)$) the set of common adjacencies (breakpoints) in A relative to B. Since $|\alpha(A, B)| = |\alpha(B, A)|$, we usually refer to the number of common adjacencies in A or B as the number of common adjacencies between A and B. Let $A = a[1]\ ...\ a[n]$ be a scaffold on Σ and x be a gene of a symbol in Σ. An insertion of x into A to result in A' refers to the operation to insert x into the gap between $a[i]$ and $a[i+1]$ to form $A' = a[1]...a[i]xa[i+1]\ ...\ a[n]$, $1 \leq i \leq n-1$, the left side of $a[1]$ to form $A' = xa[1]\ ...\ a[n]$, or the right side of $a[n]$ to form $A' = a[1]\ ...\ a[n]x$. The number of common adjacencies between A and B can be used to measure the similarity of them. Generally, the larger the number of common adjacencies between two scaffolds is, the more similar the two scaffolds are. Thus the two-sided scaffold filling problem is given by two scaffolds A and B on Σ, and asks to find a group of insertions of the genes in $c(A) - c(B)$ into B to result in B^*, and a group of insertions of the genes in $c(B)$ - $c(A)$ into A to result in A^*, such that in number, those common adjacencies between A^* and B^* are maximized. That is,

Instance: Scaffolds A and B on Σ.

Question: Find a group of insertions of the genes in $c(A) - c(B)$ into B to result in B^*, and of the genes in $c(B) - c(A)$ into A to result in A^*, such that $|\alpha(A^*, B^*)| - |\alpha(A, B)|$ is maximized.

This problem is referred to as TSSF-MNCA for short. In what follows, let $X = c(B) - c(A)$, $Y = c(A) - c(B)$. We denote by $A + X$ (resp. $B + Y$) the set

of all scaffolds which can be resulted by inserting the genes in X (resp. Y) into A (resp. B). For ease of description, we will treat the scaffolds $A' \in A + X$ and $B' \in B + Y$ rather than those insertions for A and B as a feasible solution of TSSF-MNCA. For avoiding to insert a gene into the leftmost or the rightmost side of a scaffold, we add a special gene # to both ends of A and B, and accept that any gene other than the first and the last in A or B must not be #. Thus, we will treat $A = a[0]a[1]$ ... $a[n+1]$, $B = b[0]b[1]...b[m+1]$ as the instance of TSSF-MNCA, where $a[0] = a[n+1] = b[0] = b[m+1] = \#$.

3 A Polynomial Time Algorithm for a Special Version

In this section, we show that a breakpoint can always fall into three types, no matter it is in A or B. If either of A and B does not have any breakpoints of the first type, we can design a polynomial time algorithm to solve TSSF-MNCA. This algorithm will be the basis of our final algorithm to achieve the performance ratio $\frac{3}{2}$.

3.1 Classification of the breakpoints

Let $A = a[0]a[1]$ ... $a[n]a[n+1]$, $B = b[0]b[1]...b[m]b[m+1]$, where $a[0] = a[n+1] = b[0] = b[m+1] = \#$. We call $a[i+1]$ (resp. $b[j+1]$) the *right neighbour* of $a[i]$ (resp. $b[j]$) in A (resp. B), if $a[i+1]$ (resp. $b[j+1]$) exists in A (resp. B); and $a[i-1]$ (resp. $b[j-1]$) the *left neighbour* of $a[i]$ (resp. $b[j]$) in A (resp. B), if $a[i-1]$ (resp. $b[j-1]$) exists in A (resp. B). A left or right neighbour of a gene in A (resp. B) is a *neighbour* of that gene. For a string I in A (resp. B), the *left neighbour of* I is the the left neighbour of the first gene of I, while the *right neighbour of* I is the right neighbour of the last gene of I. Both the left and the right neighbour of a string are *neighbours* of that string.

A string, say I, is a *maximal string of successive* a in A (resp. B), if every gene in I is an occurrence of 'a', while either neighbour of I is not. If a k-string is a maximal string of successive a, then it contributes $k-1$ 2-strings in which a occurs twice, and two 2-strings in which a occurs once. Thus we have,

Lemma 2. *If A (resp. B) has k maximal strings of successive a and a occurs for l times in A (resp. B), then A (resp. B) has $l-k$ 2-strings in which a occurs twice, and $2k$ 2-strings in which a occurs once.*

Proof. Let I_1, ..., I_k be the maximal strings of successive a in A (B), where the length of I_i is $l(I_i)$, $1 \leq i \leq k$. Then A (B) must have $l(I_1) - 1 + ... + l(I_k) - 1 = l - k$ 2-strings in which a occurs twice. Moreover, I_1, ..., I_k must contribute $2k$ 2-strings in which a occurs once. □

Lemma 3. *If a symbol always occurs in the common adjacencies in A (resp. B), then this symbol occurs in A (resp. B) for no more times than in B (resp. A).*

Proof. Let a be a symbol in Σ. If $a = \#$, the proof is trivial. Thus let $a \neq \#$ later. Let A, B have k_1, k_2 maximal strings of successive a, while a occur in A, B for l_1, l_2 times respectively. If a always occurs in the common adjacencies in A, then by Lemma 2 and Definition 1, $k_2 \geq k_1$ and $l_2 - k_2 \geq l_1 - k_1$. Thus $l_2 \geq l_1 - k_1 + k_2 \geq l_1$. By the same argument, $k_1 \geq k_2$ and $l_1 \geq l_2$ if a always occurs in those common adjacencies in B. □

Lemma 4. *If a symbol occurs in A (resp. B) for the same number of times as it occurs in B (resp. A) and always occurs in the common adjacencies in A (resp. B), then this symbol must occur in the common adjacencies in B (resp. A).*

Proof. Let A, B have k_1, k_2 maximal strings of successive a, while a occur in A, B for l_1, l_2 times respectively. If $a \in \Sigma$ always occurs in the common adjacencies in A, then $l_1 - k_1 \leq l_2 - k_2$ and $k_1 \leq k_2$ by Lemma 2 and Definition 1. If a also occurs in a breakpoint in B, then by Lemma 2 and Definition 1 again, either $l_1 - k_1 < l_2 - k_2$ or $k_1 < k_2$ holds true. Since $l_1 = l_2$, $l_1 - k_1 < l_2 - k_2$ must contradict with $k_1 \leq k_2$; $k_1 < k_2$ must contradict with $l_1 - k_1 \leq l_2 - k_2$. These contradictions imply that a cannot occur in a breakpoint in B. That is, it must occur in the common adjacencies in B. The other case is symmetric, hence omitted. □

We use the following lemma to recognize those genes in the breakpoints in A (resp. B).

Lemma 5. *A gene in a breakpoint in A (resp. B) must be identical to a member in Y (resp. X), if it is not identical to any gene in a breakpoint in B (resp. A); a gene in a breakpoint in A (resp. B) must be identical to a gene in a breakpoint in B (resp. A), if it is not identical to any member in Y (resp. X).*

Proof. Let $a \in \Sigma$ be a symbol which occurs in a breakpoint in A. Let A, B have k_1, k_2 maximal strings of successive a, while a occur in A, B for l_1, l_2 times respectively. (1) If a does not occur in any breakpoint in B, then $l_1 > l_2$ by Lemma 3 and Lemma 4. Thus there must exist a gene of a in Y. (2) If no gene of a is in Y, then $l_1 \leq l_2$. We use proof by contradiction to show that a must occur in a breakpoint in B. If not so, then by Lemma 3, $l_1 \geq l_2$. Thus $l_1 = l_2$. By Lemma 4, a must occur in the common adjacencies in A, which contradicts with the assumption of the lemma. For the case that a occurs in a breakpoint in B, the proof is the same. □

By Lemma 5, we can distinguish the breakpoints in A or B according to what kinds of genes they contain. A breakpoint in A (resp. B) is of type 1, if it has one gene identical to a member in Y (resp. X), and the other gene identical to a gene in a breakpoint in B (resp. A). A breakpoint in A (resp. B) is of type 2, if those two genes of it are identical to two members in Y (resp. X) respectively, and neither identical to a gene in a breakpoint in B (resp. A). A breakpoint in A (resp. B) is of type 3 if those two genes of it are identical to two genes in the breakpoints in B (resp. A) respectively, and neither identical to a member in Y (resp. X).

3.2 A polynomial time algorithm when there is no breakpoint of type 1

In this subsection, A and B are supposed to have no breakpoint of type 1. We design a polynomial time algorithm for this special case of TSSF-MNCA. A string, say I in A (resp. B), is a *breakpoint string* if every 2-string in I is a breakpoint in A (resp. B) relative to B (resp. A). A breakpoint string, say I is *maximal* if, (1) the first gene of I is #, or the first gene of I and its left neighbour form a common adjacency in A (resp. B) relative to B (resp. A), (2) the last gene is #, or the last gene and its right neighbour form a common adjacency in A (resp. B) relative to B (resp. A). The first and the last gene of a maximal breakpoint string each is an *end* of that maximal breakpoint string. In the following of this subsection, we only argue the detail for how to fill genes in Y into B. The algorithm starts with the observation about the maximal breakpoint strings. That is,

Lemma 6. *Those breakpoints in a maximal breakpoint string in A are either all of type 2, or all of type 3.*

Proof. Let $I = a[i]a[i+1]$ ... $a[j-1]a[j]$ be a maximal breakpoint string in A. Firstly, $a[i]$ cannot be both identical to a member in Y and identical to a gene in a breakpoint in B. This is because if so, $a[i]a[i+1]$ must be a breakpoint of type 1, a contradiction. If Y has an identical gene to $a[i]$, but no breakpoint in B has an identical gene to $a[i]$, then so must happen to $a[i+1]$. Otherwise the breakpoint $a[i]a[i+1]$ will be of type 1, a contradiction. For the same reason, each of $a[i+2]$, ..., $a[j]$ must play the same role as $a[i]$ and $a[i+1]$. On the other hand, if B has a breakpoint with an identical gene to $a[i]$, but Y has no identical gene to $a[i]$, then so must happen to $a[i+1]$, ..., $a[j]$, for the same reason as before. □

By Lemma 6, a maximal breakpoint string is of *type* 2 (*type* 3) if it has a breakpoint of type 2 (type 3). Only those genes in the maximal breakpoint strings of type 2 in A have identical genes in Y for filling of B. The following two corollaries will be used for counting the identical genes which act as the ends of all maximal breakpoint strings of type 2 in A.

Corollary 1. *If a symbol occurs in a maximal breakpoint string of type 2 in A, then every maximal breakpoint string which contains a gene of that symbol in A is of type 2.*

Proof. Let I be a maximal breakpoint string of type 2 in A and a be a symbol which occurs in I. If a occurs in another maximal breakpoint string, say I' in A, then I' contains an identical gene to a member in Y rather than in any breakpoint in B. By Lemma 6, I' is a maximal breakpoint string of type 2. □

Corollary 2. *If a symbol occurs in a maximal breakpoint string of type 2 in A, then it either dose not occur in B or occurs in a common adjacency in B.*

Proof. Let a be a symbol which occurs in a breakpoint, say $a[i]a[i+1]$, of type 2 in A. By Lemma 6, an identical gene to $a[i]$ and an identical gene to $a[i+1]$ must be contained in Y. Consequently, a gene of a is in Y. If a occurs in a breakpoint in B, then $a[i]a[i+1]$ must be of type 1 in A, a contradiction. □

By the following three lemmas, we show that any symbol occurs in A as ends of all maximal breakpoint strings for even times.

Lemma 7. *Let $I = a[i]...a[j]$ be a maximal string of successive a in A, $a \neq \#$. If $a[i-1]a[i]$ and $a[j]a[j+1]$ are both common adjacencies or breakpoints, then in I, there are even number of genes which act as ends of the maximal breakpoint strings.*

Proof. We decide if in I, the number of ends of the maximal breakpoint strings is even.

(1) If $a[i-1]a[i]$ and $a[j]a[j+1]$ are both common adjacencies in A relative to B, then every maximal breakpoint string must be a substring of I if it shares genes with I. A maximal breakpoint string in I contributes two ends. Thus in I, there must be even number of ends in all the maximal breakpoint strings.

(2) On the other hand, let $a[i-1]a[i]$ and $a[j]a[j+1]$ both be breakpoints in A relative to B. If there is no common adjacency in I, then there is no end of maximal breakpoint string in I. Otherwise, let $a[l]a[l+1]$ be a common adjacency in I such that l is the minimum integer with $l \geq i$, while $a[r-1]a[r]$ be a common adjacency in I such that r is the maximum integer with $r \leq j$. Then $a[l]$ is the end of that maximal breakpoint string which contains $a[i]$, and $a[r]$ is the end of that maximal breakpoint string which contains $a[j]$. Moreover, if $l = r-1$, there are two ends of maximal breakpoint strings in I; if $l < r-1$, then by the arguments of (1), there are even number of ends of maximal breakpoint strings in the substring $a[l+1]$ ... $a[r-1]$. Thus in I, there are even number of ends in all the maximal breakpoint strings. □

Lemma 8. *Let $I = a[i]$ ... $a[j]$ be a maximal string of successive a in A, $a \neq \#$. If one of $a[i-1]a[i]$ and $a[j]a[j+1]$ is a common adjacency, while the other is a breakpoint in A, then in I, there are odd number of genes which act as ends of the maximal breakpoint strings.*

Proof. Without loss of generality, let $a[i-1]a[i]$ be a common adjacency in A relative to B, while $a[j]a[j+1]$ be a breakpoint in A relative to B. Let $a[r-1]a[r]$ be the common adjacency in I such that r is the maximum integer with $r \leq j$. Then $a[r]$ is an end of the maximal breakpoint string which contains $a[j]$. If $r = i$, then $a[r]$ is the unique end of the maximal breakpoint strings. Otherwise, by Lemma 7, there are even number of genes which act as ends of the maximal breakpoint strings in the substring $a[i]$ ... $a[r-1]$. Thus in I, there are odd number of genes which act as ends of the maximal breakpoint strings. □

Lemma 9. *For an arbitrary $a \in \Sigma$, there are even number of genes of a, which act as ends of the maximal breakpoint strings of type 2 in A.*

Proof. If in A, no end of a maximal breakpoint string is a gene of a, the proof is trivial. If a occurs in a breakpoint of type 3, then by corollary 1, a cannot occur in any maximal breakpoint string of type 2 in A, the proof is also trivial. Later, a is supposed to occur in A as an end of a maximal breakpoint string of type 2. Thus by corollary 1, every maximal breakpoint string in A which contains a gene of a must be of type 2. That is, we only need to decide if the genes of a act as ends of the maximal breakpoint strings for even or odd times, without regarding the type of the maximal breakpoint strings in A.

For ease of description, we denote by ax a 2-string in which one gene is an occurrence of a while the other gene is not. A maximal string of successive a always contributes two strings of ax. Let B have p maximal strings of successive a. Then by corollary 2, the $2p$ strings of ax in B are all common adjacencies. Thus A also has just $2p$ strings of ax which are common adjacencies.

In A, a maximal string of successive a always *contribute* two strings of ax, each of which can be a common adjacency or a breakpoint. Let in A, S_0, S_1, S_2 be sets of maximal strings of successive a, which contribute zero, one, and two strings of ax as breakpoints respectively. Then there are $2|S_0| + |S_1|$ strings of ax as common adjacencies in A. Following $2|S_0| + |S_1| = 2p$, $|S_1|$ is even. By Lemma 7 and 8, there are even number of genes of a which act as ends of the maximal breakpoint strings in $S_0 \cup S_1 \cup S_2$, and in A as well. □

The following lemma is used to find a gap in B for an insertion of some genes in Y.

Lemma 10. *A symbol in Σ which occurs in A as an end of a maximal breakpoint string must occur in B.*

Proof. Let a be a symbol which occurs in A as an end of a maximal breakpoint string I. If I is of type 3, a must occur in B due to the definition of the breakpoint of type 3. If I is of type 2, since the gene of a as an end of I forms a common adjacency in A relative to B with one neighbour of I, thus a must occur in B. □

In what follows, an insertion *to insert a string into B to increase k common adjacencies* for B relative to A refers to that this insertion inserts a string into the gap between two adjacent genes in B to result in B', such that $|\alpha(B', A)| - |\alpha(B, A)| = k$. We aim to find insertions of the genes in Y into B to increase $|c(A) - c(B)|$ common adjacencies for B totally. To hit this aim, we try to make a k-string which can be inserted into B to increase just k common adjacencies for B relative to A.

We denote by $\beta_2(A, B)$, $\beta_3(A, B)$ the sets of breakpoints of type 2 and type 3 in A relative to B respectively. Let $I = x[1] \ldots x[t]$ be an arbitrary string. We set a relation between $\beta_2(A, B)$ and $\tau(I)$ and name it as $R(\beta_2(A, B), \tau(I))$. A pair, say $(a[j]a[j+1], x[i]x[i+1])$, belongs to $R(\beta_2(A, B), \tau(I))$, if and only if $a[j]a[j+1] \in \beta_2(A, B)$ is identical to $x[i]x[i+1] \in \tau(I)$, $1 \leq i \leq t-1$, $0 \leq j \leq n$. A matching between $\beta_2(A, B)$ and $\tau(I)$ is a subset of $R(\beta_2(A, B), \tau(I))$, such that any 2-string in $\beta_2(A, B)$ or $\tau(I)$ cannot occur in the pairs in the subset for two or more times. A matching between $\beta_2(A, B)$ and $\tau(I)$ is maximum if its

cardinality is maximized. For convenience, we represent a string, say I, as $I = l(I)M(I)r(I)$, where $l(I)$, $r(I)$ are the first and the last gene of I, and $M(I)$ is the substring of I except $l(I)$ and $r(I)$. The following lemma states the nature of those strings which consist of the genes in Y and can be provided for insertions into B.

Lemma 11. *If a string I subjects to:* (1) $l(I) = r(I)$*;* (2) *a common adjacency in B has an identical gene to $l(I)$;* (3) *there exists a matching of cardinality $|I|-1$ between $\beta_2(A, B)$ and $\tau(I)$, then either $M(I)r(I)$ or $l(I)M(I)$ can be inserted into B to increase $|I| - 1$ common adjacencies for B relative to A, where $|I|$ is the length of I.*

Proof. Let $I = x[0]x[1]...x[t]x[t+1]$ with $x[0] = x[t+1]$. Let the $|I|-1$ 2-strings $x[0]x[1]$, ..., $x[t-1]x[t]$, $x[t]x[t+1]$ in I be identical to the breakpoints $a[i_1]a[i_1+1]$, ..., $a[i_t]a[i_t+1]$, $a[i_{t+1}]a[i_{t+1}+1]$ in A as well as $\beta_2(A, B)$ respectively. If $b[j]b[j+1]$ is a common adjacency in B and $b[j] = x[0]$, then we can insert $M(I)r(I)$ between $b[j]$ and $b[j+1]$. This insertion transforms B into $B' = \#$ ... $b[j]x[1]$ ... $x[t]x[t+1]b[j+1]$ ... $\#$, where $M(\beta_2(A, B), \tau(I)) = \{$ $(a[i_1]a[i_1+1], b[j]x[1])$, ..., $(a[i_{t+1}]a[i_{t+1}+1], x[t]x[t+1])$ is a maximum matching between $\beta_2(A, B)$ and $\tau(I)$. If M is a maximum matching between $\tau(A)$ and $\tau(B)$, with respect to which the common adjacencies in A and B are identified, then $M' = M \bigcup M(\beta_2(A, B), \tau(I))$ is a matching between $\tau(A)$ and $\tau(B')$. Here, that pair ($\bullet$, $b[j]b[j+1]$) in M turns into ($\bullet$, $x[t+1]b[j+1]$), which can be thought of as nothing changed for M. Thus $|\alpha(A, B')| - |\alpha(A, B)| = |M'| - |M| = t + 1 = |I|-1$. If $b[j]b[j+1]$ is a common adjacency in B and $b[j+1] = x[0]$, then we can insert $l(I)M(I)$ between $b[j]$ and $b[j+1]$ to result in B'. Similar as the argument for the insertion of $M(I)r(I)$, we can set a matching between $\tau(A)$ and $\tau(B')$, such that $|\alpha(A, B')| - |\alpha(A, B)| = |M'| - |M| = t + 1 = |I| - 1$. □

A string is *good*, if it subjects to the three properties in Lemma 11. Actually,

Corollary 3. *If I is good, then $c(l(I)M(I)) \subseteq Y$ and $c(M(I)r(I)) \subseteq Y$.*

Proof. By Corollary 2, a symbol which occurs in $l(I)M(I)$ must occur in a common adjacency in B, if it occurs in B. Let B' be the scaffold resulted by the method in Lemma 11 to insert $l(I)M(I)$ ($M(I)r(I)$) into B. By Lemma 11, each symbol which occurs in $l(I)M(I)$ must occur in a common adjacency in B'. By Lemma 3, each symbol must occur no more times in $c(l(I)M(I))$ ($c(M(I)r(I))$) than in Y. That is, $c(l(I)M(I)) = c(M(I)r(I)) \subseteq Y$. □

Let I_1 and I_2 be two strings with $r(I_1) = l(I_2)$, then we refer to the action to remove $r(I_1)$ from I_1 and then concatenate $l(I_1)M(I_1)$ with I_2 as the operation to *tie* I_1 and I_2 and write it by the operator $\bowtie$. Formally, $I_1 \bowtie I_2 = l(I_1)M(I_1)l(I_2)M(I_2)r(I_2)$. Fortunately, we can make a good string by tying some maximal breakpoint strings of type 2.

Lemma 12. *If $Y \neq \emptyset$, then A has a set of maximal breakpoint strings of type 2, which can be tied into a good string.*

Proof. Let Γ be the set of all maximal breakpoint strings of type 2 in A. By Lemma 9, each symbol occurs in A as ends of the maximal breakpoint strings in Γ for even times . We make a set Θ of maximal breakpoint strings by the following algorithm:

1. Select a maximal breakpoint string I in Γ randomly.
2. $t \leftarrow 1$; $I[1] \leftarrow I$; $\Theta \leftarrow \{I[1]\}$; $\Gamma \leftarrow \Gamma - \{I\}$.
3. If $l(I[1]) \neq r(I[t])$, select $I \in \Gamma$ such that, $l(I) = r(I[t])$ or $r(I) = r(I[t])$.
   (a) If $(l(I) = r(I[t]))$ then $I[t+1] \leftarrow I$; else $I[t+1] \leftarrow -I$.
   (b) $\Theta \leftarrow \Theta \cup \{I[t+1]\}$; $\Gamma \leftarrow \Gamma - \{I\}$; $t \leftarrow t+1$.
4. Repeat step 3 until $l(I[1]) = r(I[t])$.

In this algorithm, step 3 is feasible because the identical genes to $r(I[t])$ must act as ends of the maximal breakpoint strings in Θ for odd times if $l(I[1]) \neq r(I[t])$. This algorithm must end up with $\Theta \neq \emptyset$ if $\Gamma \neq \emptyset$. This is because if not so, the identical genes to $r(I[t])$ will act as ends of the maximal breakpoint strings in Γ for odd times, which contradicts with Lemma 9.

Let $I[1]$, ..., $I[t]$ be the maximal breakpoint strings in Θ in the order they are appended into Θ. Then $I[1] \bowtie ... \bowtie I[t]$ is good, because (1) $l(I[1]) = r(I[t])$; (2) a common adjacency in B must have an identical gene to $l(I[1])$ by Lemma 10; (3) each 2-string in $\tau(I[1] \bowtie ... \bowtie I[t])$ is also in $\tau(I[k])$, where $I[k] \in \Theta \subseteq \Gamma$, $1 \leq k \leq t$. □

For evaluating the time complexity of making a good string, let $N = |A| + |X| = |B| + |Y|$. It takes $O(N^2)$ time to set a maximum matching between $\tau(A)$ and $\tau(B)$. It has to take $O(N^2)$ time to identify a 2-string to be a common adjacency or a breakpoint of type 2 or type 3. Thus the time complexity of getting all breakpoints along with their types in A and B is $O(N^3)$. It takes $O(N)$ time to identify those maximal breakpoint strings in A and B, if all breakpoints in A and B are known. Thus the time complexity of getting those maximal breakpoint strings in A and B is also $O(N^3)$. Two maximal breakpoint strings are concatenated because they have the identical genes as their ends. It takes $O(N)$ time to find in Γ a maximal breakpoint string with a gene of a certain symbol as its end. Thus the time complexity of making a good string is $O(N^2)$, if we do not count in the time to get those maximal breakpoint strings in A and B.

For a string I, we refer to $l(I)M(I)$ and $M(I)r(I)$ as the *brothers* of I. We can always make a good string if $Y \neq \emptyset$ by the approaches in Lemma 12. By Lemma 11 and Corollary 3, a brother of a good string has all its genes in Y and can be inserted into B to increase as many common adjacencies for B as those genes in it. One may feel afraid if the scaffold A can keep without having breakpoint of type 1, after B has been filled by that brother of a good string. We show that this is true.

Lemma 13. *If an insertion of a brother of a good string into B transforms B into B', then A has no breakpoint of type* 1 *relative to B', no matter with respect to what maximum matching the common adjacencies in A and B' are identified.*

Proof. Let M be that maximum matching with respect to which the common adjacencies in A and B are identified. Let $I = x[0]x[1]$ ... $x[t]x[t+1]$ be a good string. Without loss of generality, let $b[j]b[j+1]$ be a common adjacency in B, where $b[j] = x[0]$. If B' is the scaffold resulted by inserting $M(I)r(I)$ into the gap between $b[j]$ and $b[j+1]$, then the 2-strings $b[j]x[1]$, ..., $x[t-1]x[t]$, $x[t]x[t+1]$ must all become common adjacencies in B' relative to A with respect to M', where $M' = M \cup M(\beta_2(A, B), \tau(I))$ as in Lemma 11. Thus a breakpoint in B' with respect to M' must be a breakpoint in B with respect to M. Namely, if A has a breakpoint of type 1 relative to B' with respect to M', it must be a breakpoint of type 1 relative to B with respect to M, a contradiction. Even if a maximum matching, say M" other than M' between $\tau(A)$ and $\tau(B')$ are set to identify the common adjacencies, by Lemma 1 and the definition of the breakpoint of type 1, A still has no breakpoint of type 1 relative to B' with respect to M". □

By Lemma 13, if $Y \neq \emptyset$ after a brother of a good string is inserted into B, we can use the approaches in Lemma 12 again to make a good string for another insertion. Such insertions can be repeated until all genes in Y are inserted into B. The algorithm is specialized as SpecialInsert(A, B). In the description of the algorithm, the subroutine Insert$(\bullet, b[j], b[j+1], B)$ inserts a string into the gap between $b[j]$ and $b[j+1]$ in B.

Algorithm *SpecialInsert*(A, B)
1. While$(Y \neq \emptyset)$
2. Make a good string I;(Lemma 12)
3. Find a common adjacency $b[j]b[j+1]$ in B for I; (Lemma 10,11)
4. If $(b[j] = l(I))$ $B \leftarrow$ Insert$(M(I)r(I), b[j], b[j+1], B)$;
5. Else $B \leftarrow$ Insert$(l(I)M(I), b[j], b[j+1], B)$;
6. $Y \leftarrow Y - c(l(I)M(I))$;
7. End while
8. Return B.

One time for getting all maximal breakpoint strings in A takes $O(N^3)$ time. Making a good string from the set of maximal breakpoint strings takes $O(N^2)$ time. Finding a common adjacency for an insertion of a good string takes $O(N)$ time. Thus, the time complexity of SpecialInsert(A, B) is $O(N^3)$, where $N = |A| + |X| = |B| + |Y|$.

By Lemma 12, 13, if A does not have any breakpoint of type 1, we can use Special-Insert(A, B) to insert all genes in Y into B. Let SpecialInsert(A, B) return B^* as its solution. Following Lemma 11, we have $|\alpha(A, B^*)|$ - $|\alpha(A, B)|$ $= |Y|$. If B does not have any breakpoint of type 1 on the other hand, we can also use SpecialInsert(B, A) to insert all genes in X into A to result in A^* with $|\alpha(A^*, B)| - |\alpha(A, B)| = |X|$. Actually we have,

Lemma 14. *Let* $B^* \leftarrow$ *SpecialInsert*(A, B), $A^* \leftarrow$ *SpecialInsert*(B, A). *Then* $|\alpha(A^*, B^*)| - |\alpha(A, B)| \geq |X| + |Y|$.

Proof. Let I_1^Y, ..., I_k^Y be the good strings made by SpecialInsert(A, B), and I_1^X, ..., I_l^X be the good strings made by SpecialInsert(B, A). Let M be the maximum matching with respect to which the common adjacencies in A and B are identified. Let $M(\beta_2(A,B), \tau(I_i^Y))$ $(1 \leq i \leq k)$ be the maximum matching between $\beta_2(A,B)$ and $\tau(I_i^Y)$, while $M(\tau(I_j^X), \beta_2(B,A))$ $(1 \leq j \leq l)$ be the maximum matching between $\tau(I_j^X)$ and $\beta_2(B,A)$. Let $M_1 = \bigcup_{j=1}^{l} M(\tau(I_j^X), \beta_2(B,A))$, $M_2 = \bigcup_{i=1}^{k} M(\beta_2(A,B), \tau(I_i^Y))$. Then $|M_1| = |X|$, $|M_2| = |Y|$ by Lemma 11. Each pair in $M \cup M_1 \cup M_2$ must happen between $\tau(A^*)$ and $\tau(B^*)$. By Lemma 11 again, a pair in M cannot share any 2-string with a pair in $M_1 \cup M_2$. Since a breakpoint of type 2 in A cannot be identical to any breakpoint of type 2 in B, a pair in M_1 cannot share a 2-string with any pair in M_2. Thus $M \cup M_1 \cup M_2$ must be a matching between $\tau(A^*)$ and $\tau(B^*)$ with cardinality $|M|+|M_1|+|M_2|$. Thus, $|\alpha(A^*, B^*)| \geq |\alpha(A,B)| + |X| + |Y|$. □

We show that A^* and B^* as a solution of TSSF-MNCA is optimal. That is, $|\alpha(A^*, B^*)| = max\{|\alpha(A', B')| : A' \in A + X, B' \in B + Y\}$.

Lemma 15. *If A and B do not have any breakpoint of type 1, then $\beta_3(A^*, B^*) = \beta_3(A,B)$, $\beta_3(B^*, A^*) = \beta_3(B,A)$.*

Proof. No gene can be inserted into a breakpoint of type 3 in B and A by SpecialInsert(A, B) as well as SpecialInsert(B, A). Moreover, each breakpoint of type 2 in A relative to B turns into a common adjacency in A^* relative to B^* by Lemma 11, and so does each breakpoint of type 2 in B relative to A. Thus $\beta_3(A^*, B^*) = \beta_3(A,B)$, $\beta_3(B^*, A^*) = \beta_3(B,A)$. □

Since $c(A^*) = c(B^*)$, $|\alpha(A^*, B^*)| = |\alpha(B^*, A^*)|$, thus $|\beta_3(A^*, B^*)| = |\beta_3(B^*, A^*)|$. By Lemma 15, we have,

Corollary 4. *If A and B do not have any breakpoint of type 1, then $|\beta_3(A,B)| = |\beta_3(B,A)|$.*

The number of common adjacencies between a scaffold in $A+X$ and a scaffold in $B+Y$ can be bounded by,

Lemma 16. *Let $A' \in A+X$, $B' \in B+Y$. Then $|\alpha(A', B')| \leq |\alpha(A,B)| + |X| + |Y|$.*

Proof. We argue that $|\alpha(A', B')| \leq |c(B')| - |\beta_3(B,A)| - 1$. If B has no breakpoint of type 3, the proof is trivial. Otherwise, let $b[j]b[j+1]$ be an arbitrary breakpoint of type 3 in B.

If $b[j]b[j+1]$ remains to be a 2-string in B', it must be a breakpoint in B' relative to A'. This is because, (1) A does not have any 2-string which contains two genes identical to $b[j]$ and $b[j+1]$ respectively by the definition of breakpoint of type 3; (2) since no identical gene to $b[j]$ or $b[j+1]$ exists in X, any insertion of the genes in X into A cannot make a 2-string which contains two genes identical to $b[j]$ and $b[j+1]$ respectively.

If in B', some genes, say $y[1]$, ..., $y[t]$, are located between $b[j]$ and $b[j+1]$ in the order from left to right, then $b[j]y[1]$ and $y[t]b[j+1]$ must be breakpoints in B' relative to A'. This is because, (1) if A has a 2-string which contains two genes identical to $b[j]$ and $y[1]$, or a 2-string which contains two genes identical to $y[t]$ and $b[j+1]$ respectively, it must be a breakpoint of type 1 in A; (2) since $X \cap Y = \emptyset$, any insertion of the genes in X into A can not make a 2-string which contains two genes identical to $b[j]$ and $y[1]$, or a 2-string which contains two genes identical to $y[t]$ and $b[j+1]$.

Thus, there must exist no fewer breakpoints in B' relative to A' than those in B relative to A. A 2-string in B' is either a breakpoint or a common adjacency relative to A'. Thus $|\alpha(A', B')| = |\alpha(B', A')| \leq |c(B')| - |\beta_3(B, A)| - 1$. Since $|c(B')| = |c(B)| + |Y|$, $|\alpha(B, A)| + |\beta_2(B, A)| + |\beta_3(B, A)| = |c(B)| - 1$, and $|\beta_2(B, A)| = |X|$, thus $|c(B')| - |\beta_3(B, A)| - 1 = |c(B)| + |Y| - |c(B)| + 1 + |\alpha(B, A)| + |X| - 1 = |\alpha(A, B)| + |X| + |Y|$. □

The undermentioned theorem follows from Lemma 14 and Lemma 16.

Theorem 1. *Let* $A^* \leftarrow$ *SpecialInsert*(B, A), $B^* \leftarrow$ *SpecialInsert*(A, B). *Then* $|\alpha(A^*, B^*)| = max\{|\alpha(A', B')| : A' \in A + X,\ B' \in B + Y\}$.

4 The 1.5-Approximation Algorithm

In this section, $A^* \in A + X$ and $B^* \in B + Y$ as a whole is treated as an optimal solution of the TSSF-MNCA instance A, B.

4.1 A bound for the number of common adjacencies

We propose a simple algorithm for TSSF-MNCA at the beginning of this subsection. This simple algorithm will be used to derive a bound of the number of common adjacencies between a scaffold in $A + X$ and a scaffold in $B + Y$, and in the next subsection will act as a subroutine in our approximation algorithm.

Lemma 17. *If* A *has a breakpoint of type* 1*, then* Y *has a gene which can be inserted into* B *to increase at least one common adjacency for* B.

Proof. Let M be the maximum matching with respect to which the common adjacencies in A and B are identified. Without loss of generality, let $a[i]a[i+1]$ be a breakpoint of type 1 in A, where $a[i]$ is identical to a gene, say $y \in Y$, while $a[i+1]$ is identical to a gene in a breakpoint, say $b[j]b[j+1]$ in B. The insertion of y into the gap between $b[j]$ and $b[j+1]$ will transform B into $B' =$ # ... $b[j]yb[j+1]$ ... #. If $b[j] = a[i+1]$, then $a[i]a[i+1]$ is identical to $b[j]y$, $(a[i]a[i+1], b[j]y)$ can be appended to M to form a matching between $\tau(A)$ and $\tau(B')$. If $b[j+1] = a[i+1]$, then $a[i]a[i+1]$ is identical to $yb[j+1]$, $(a[i]a[i+1]$, $yb[j+1])$ can be appended to M to form a matching between $\tau(A)$ and $\tau(B')$. Thus, $|\alpha(A, B')| - |\alpha(A, B)| \geq 1$. □

If A has a breakpoint of type 1, we can insert a gene in Y into B by the method in Lemma 17. Such an insertion can increase at least one common adjacency for B, and can be repeated until A has no breakpoint of type 1. If $Y \neq \emptyset$ while A has no breakpoint of type 1, SpecialInsert(A, B) can be used to insert the genes in Y into B. Thus we can design an algorithm for inserting those genes in Y into B to increase at least $|Y|$ common adjacencies for B. This algorithm is named as SimpleInsert(A, B). The subroutine Type1(A, B) in the algorithm returns two integers i and j, where $a[i] \in c(A)$ is identical to a gene in Y, and can be inserted into the breakpoint $b[j]b[j+1]$ in B to increase at least 1 common adjacencies for B. If A has no breakpoint of type 1, then Type1(A, B) returns (-1,-1).

Algorithm *SimpleInsert*(A, B)
1. While $(Y \neq \emptyset)$
2. $\quad (i, j) \leftarrow$ Type1(A, B);
3. $\quad$ If $(i \geq 0)$
4. $\quad\quad \{\ B \leftarrow$ Insert$(a[i], b[j], b[j+1], B);\ Y \leftarrow Y - \{a[i]\}\ \}$
5. $\quad$ Else $\{\ B \leftarrow$ SpecialInsert$(A, B)\ \}$
6. End while
7. Return B.

As in Section 3.2, let $N = |A| + |X| = |B| + |Y|$. It takes $O(N^2)$ time to find a breakpoint of type 1 in A and a breakpoint in B which provides a gap for an insertion. That is, once to call Type1(A, B) takes $O(N^2)$ time. The while loop of this algorithm can repeat at most N times, where the subroutine SpecialInsert(A, B) can be called for at most once. In summary, the time complexity of SimpleInsert(A, B) is $O(N^3)$.

Let B' be the scaffold returned by SimpleInsert(A, B). Then, SimpleInsert(B', A) can be used to insert those genes in X into A to increase at least $|X|$ common adjacencies for A relative to B'. Thus,

Lemma 18. *There exists a scaffold $A' \in A + X$ and a scaffold $B' \in B + Y$, such that $|\alpha(A', B')| - |\alpha(A, B)| \geq |X| + |Y|$.*

Proof. Let B' be the scaffold returned by SimpleInsert(A, B). Then by Lemma 17 and Lemma 14, $|\alpha(A, B')| - |\alpha(A, B)| \geq |Y|$.

Let A' be the scaffold returned by SimpleInsert(B', A). Since $c(B') - c(A) = c(B) - c(A) = X$, by Lemma 17 and Lemma 14 again, $|\alpha(A', B')| - |\alpha(A, B')| \geq |X|$. Finally, $|\alpha(A', B')| - |\alpha(A, B)| = |\alpha(A', B')| - |\alpha(A, B')| + |\alpha(A, B')| - |\alpha(A, B)| \geq |X| + |Y|$. □

Let I be a substring in A (B). A 2-string is *involved by* I, if it shares at least one gene with I. A k-string can involve at most $k + 1$ 2-strings. Thus, an insertion of a k-string into B (A) can increase at most $k+1$ common adjacencies for B (A). Actually, we can always use one insertion to insert a k-string into A or B to increase k or $k + 1$ common adjacencies for A or B. Thus, we only pay attention to those k-strings for $k \geq 1$, which can be inserted into A or B by one insertion, thus to increase k or $k + 1$ common adjacencies. A k-string in $A' \in$

$A + X$ ($B' \in B + Y$) is *k-type*-1, if every 2-string involved by the k-string is a common adjacency in A' (B'). A k-string is *k-type*-2 in A' (B'), if k of those $k+1$ 2-strings involved by the k-string are common adjacencies in A' (B').

Although a symbol can both occur in A (B) and in X (Y), a gene in a scaffold in $A + X$ ($B + Y$) is either in $c(A)$ ($c(B)$) or in X (Y). A substring in a scaffold in $A + X$ ($B + Y$) is *super*, if each gene in it is in X (Y), while the neighbours of it are in $c(A)$ ($c(B)$). For a 2-string $a[i]a[i+1]$ in A, a super string in A^* (B^*) *stands in* $a[i]a[i+1]$ in A (B), if in A^* (B^*), it is located in the gap between $a[i]$ and $a[i+1]$. Let M be a maximum matching between $\tau(A)$ and $\tau(B)$. Then a super string in A^* (B^*) *stands in* a breakpoint (common adjacency) in A (B) relative to B (A) with respect to M, if in A^* (B^*), it stands in $a[i]a[i+1]$ as a breakpoint (common adjacency) in A (B) relative to B (A) with respect to M.

Lemma 19. *A super k-string in A^* is either k-type-1 or k-type-2. And so is a super k-string in B^*.*

Proof. Let $I = x[1]x[2] \ldots x[k]$ be a super k-string in A^*, which stands in $a[i]a[i+1]$. We argue that I involves at most one breakpoint in A^* relative to B^*. If not so, let $I' = x[s]$ ... $x[t]$ be a substring of I such that both $x[s-1]x[s]$ and $x[t]x[t+1]$ are breakpoints in A^* relative to B^*, where $x[s-1]$ and $x[t+1]$ are the left neighbour of $x[s]$ and the right neighbour of $x[t]$. Then we can remove all genes in I' from A^* to result in a scaffold A_1^*. Thus $|\alpha(A^*, B^*)| - |\alpha(A_1^*, B^*)| \leq t - s$. By Lemma 18, we can insert those genes in I' into A_1^* again to result in a scaffold A_2^* so that $|\alpha(A_2^*, B^*)| - |\alpha(A_1^*, B^*)| \geq t - s + 1$. Thus $|\alpha(A_2^*, B^*)| - |\alpha(A^*, B^*)| \geq 1$. This implies A^* and B^* as a solution of TSSF-MNCA is not optimal, a contradiction. By the same argument, a super k-string in B^* must be k-type-1 or k-type-2. □

Let $M(A^*, B^*)$ be the maximum matching with respect to which the common adjacencies in A^* and B^* are identified. A pair in $M(A^*, B^*)$ is *original*, if either of its common adjacencies is not involved by any super string. A pair in $M(A^*, B^*)$ is *new* if at least one common adjacency of it is involved by a super string. Especially, a pair in $M(A^*, B^*)$ is *extra*, if both of its common adjacencies are involved by super strings. Since $X \cap Y = \emptyset$, an extra pair in $M(A^*, B^*)$ must have one common adjacency with a gene in X and a gene in $c(A)$ and the other common adjacency with a gene in Y and a gene in $c(B)$. To formulate the number of common adjacencies between A^* and B^*, we use the following notations to represent those specialized pairs in $M(A^*, B^*)$, and those super strings in A^* and B^*.

(1) $M_0(A^*, B^*)$: the subset of $M(A^*, B^*)$, in which each pair is original.
(2) $M_1(A^*, B^*)$: the subset of $M(A^*, B^*)$, in which each pair is extra.
(3) $M_2(A^*, B^*)$: the subset of $M(A^*, B^*)$, in which each pair is new.
(4) $A^*[k, 1]$ ($B^*[k, 1]$): the set of k-type-1 super strings in A^* (B^*), $k \geq 1$.
(5) $A^*[k, 2]$ ($B^*[k, 2]$): the set of k-type-2 super strings in A^* (B^*), $k \geq 1$.

Lemma 20. *The number of common adjacencies between A^* and B^* is,*

$$|\alpha(A^*, B^*)| = |M_0(A^*, B^*)| + \sum_{k=1}^{|X|}(k+1)|A^*[k,1]| + \sum_{k=1}^{|X|} k|A^*[k,2]| + \sum_{k=1}^{|Y|}(k+1)|B^*[k,1]| + \sum_{k=1}^{|Y|} k|B^*[k,2]| - |M_1(A^*, B^*)|. \quad (1)$$

Proof. A new pair in $M(A^*, B^*)$ must have one common adjacency which is involved by a super string in A^* or B^*. By Lemma 19, the common adjacencies involved by the super strings in A^* can be summed up to be $\sum_{k=1}^{|X|}(k+1)|A^*[k,1]| + \sum_{k=1}^{|X|} k|A^*[k,2]|$, where a common adjacency in each extra pair in $M(A^*, B^*)$ is counted once. On the other hand, the common adjacencies involved by the super strings in B^* can be summed up to be $\sum_{k=1}^{|Y|}(k+1)|B^*[k,1]| + \sum_{k=1}^{|Y|} k|B^*[k,2]|$, where a common adjacency in each extra pair in $M(A^*, B^*)$ is counted once too. Since $X \cap Y = \emptyset$, a new pair in $M(A^*, B^*)$ which has one common adjacency involved by a super string in A^* cannot have the other common adjacency involved by a super string in B^*, if it is not extra. Thus $|\alpha(A^*, B^*)| = |M(A^*, B^*)| = |M_0(A^*, B^*)| + \sum_{k=1}^{|X|}(k+1)|A^*[k,1]| + \sum_{k=1}^{|X|} k|A^*[k,2]| + \sum_{k=1}^{|Y|}(k+1)|B^*[k,1]| + \sum_{k=1}^{|Y|} k|B^*[k,2]| - |M_1(A^*, B^*)|$. □

Let M be a maximum matching between $\tau(A)$ and $\tau(B)$ with respect to which the common adjacencies in A and B are identified. We denote by $B\_A^*[1,1,M]$ ($B\_B^*[1,1,M]$) the set of 1-type-1 super strings in A^* (B^*), each of which stands in a breakpoint in A (B) relative to B (A) with respect to M, and involves no common adjacency of an extra pair. Moreover, we denote by $C\_A^*[1,1,M]$ ($C\_B^*[1,1,M]$) the set of 1-type-1 super strings in A^* (B^*), each of which stands in a common adjacency in A (B) relative to B (A) with respect to M. To bound $|\alpha(A^*, B^*)|$, we start with bounding the number of 1-type-1 super strings which stand in the common adjacencies in A and B respectively.

Lemma 21. *There exists a maximum matching, say M between $\tau(A)$ and $\tau(B)$, with respect to which the common adjacencies in A and B are identified such that, $|M \cap M(A^*, B^*)| = |M_0(A^*, B^*)|$; $|M| - |M_0(A^*, B^*)| \geq |C\_A^*[1,1,M]|$; $|M| - |M_0(A^*, B^*)| \geq |C\_B^*[1,1,M]|$.*

Proof. For any two symbols a, b in Σ, a pair of (ab, ab) refers to a pair with two strings of ab. Let ab occur in A and B for $N(a,b,A)$ and $N(a,b,B)$ times respectively. Then any maximum matching between $\tau(A)$ and $\tau(B)$ must contain $\min\{N(a,b,A),\ N(a,b,B)\}$ pairs of (ab, ab). If in A^* and B^*, there are $n(a,b,A)$ and $n(a,b,B)$ super strings which stand in the strings of ab, then we can set a maximum matching, M namely, between $\tau(A)$ and $\tau(B)$, in which just $\min\{N(a,b,A) - n(a,b,A),\ N(a,b,B) - n(a,b,B)\}$ pairs of (ab, ab) act as original pairs in $M(A^*, B^*)$. This implies we have at most $\min\{N(a,b,A),\ N(a,b,B)\} - \min\{N(a,b,A) - n(a,b,A),\ N(a,b,B) - n(a,b,B)\}$ k-type-1 ($k \geq 1$) super strings

in A^*, each of which stands in a common adjacency of ab in A with respect to M. Since,

$$\sum_{a,b} min\{N(a,b,A) - n(a,b,A), N(a,b,B) - n(a,b,B)\} = |M_0(A^*, B^*)|, \quad (2)$$

M must have $|M_0(A^*, B^*)|$ pairs each of which acts as an original pair in $M(A^*, B^*)$. Moreover,

$$\sum_{a,b} min\{N(a,b,A), N(a,b,B)\} = |M|. \quad (3)$$

Thus, there cannot exist more than $|M| - |M_0(A^*, B^*)|$ 1-type-1 super strings in A^* which stand in the common adjacencies in A with respect to M. That is,

$$|M| - |M_0(A^*, B^*)| \geq |C_- A^*[1, 1, M]|. \quad (4)$$

By the same reason,

$$|M| - |M_0(A^*, B^*)| \geq |C_- B^*[1, 1, M]|. \quad (5)$$

□

To show that Lemma 21 can work for an arbitrary maximum matching between $\tau(A)$ and $\tau(B)$, we set a mapping between two scaffolds in $A + X$ as well as $B + Y$. Let A_1^* and A_2^* be two scaffolds in $A + X$; B_1^* and B_2^* be two scaffolds in $B + Y$. Let f be a 1-1 mapping from the set of super strings in A_1^* (B_1^*) to the set of super strings in A_2^* (B_2^*). Then for two maximum matchings M and M' between $\tau(A)$ and $\tau(B)$, f is *identical*, if for each super string I in A_1^* (B_1^*) and its image $f(I)$ in A_2^* (B_2^*), it subjects to,

(1) I is identical to $f(I)$;
(2) if I stands in $a[i]a[i+1]$ in A_1^* (B_1^*), $f(I)$ stands in $a[j]a[j+1]$ in A_2^* (B_2^*), then $a[i]Ia[i+1]$ is identical to $a[j]f(I)a[j+1]$;
(3) if I stands in a breakpoint (common adjacency) in A (B) with respect to M, then $f(I)$ stands in a breakpoint (common adjacency) in A (B) with respect to M'.

Lemma 22. *Let M be a maximum matching between $\tau(A)$ and $\tau(B)$ which subjects to Lemma 21, M' be a maximum matching between $\tau(A)$ and $\tau(B)$ other than M. Then there exist two scaffolds $A_1^* \in A + X$ and $B_1^* \in B + Y$ such that,*

(1) *for M and M', an identical mapping exists from the set of super strings in A^* (B^*) to the set of super strings in A_1^* (B_1^*);*

(2) *there exists a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ which shares $|M_0(A^*, B^*)|$ pairs with M' exactly.*

Proof. For two arbitrary symbols a, b in Σ, M and M' must have the same number of pairs of (ab, ab) by lemma 1. Since by Lemma 21, $|M \cap M(A^*, B^*)|$ $= |M_0(A^*, B^*)|$, let M'_0 be a subset of M' which has as many pairs of (ab, ab) as $M_0(A^*, B^*)$ has. We make two scaffolds A_1^* and B_1^* by re-inserting those super strings in A^* and B^* into A and B respectively, and set a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ which shares $|M'_0|$ pairs with M'.

Let in A^*, $I[1]$, ..., $I[t]$ be the super strings which stand in the breakpoints $a[i_1]a[i_1+1]$, ..., $a[i_t]a[i_t+1]$ in A with respect to M, respectively. Then by Lemma 1, there must exist t breakpoints in A, say $a[j_1]a[j_1+1]$, ..., $a[j_t]a[j_t+1]$ with respect to M', such that $a[j_k]a[j_k+1]$ is identical to $a[i_k]a[i_k+1]$ for $1 \leq k \leq t$. Thus we re-insert $I[k]$ or its reversal into $a[j_k]a[j_k+1]$ for $1 \leq k \leq t$ as follows. (1) Set the image of $I[k]$ as $f(I[k]) = I[k]$ if $a[i_k]a[i_k+1] = a[j_k]a[j_k+1]$, and $f(I[k]) = -I[k]$ otherwise. (2) Insert $f(I[k])$ into $a[j_k]a[j_k+1]$. This leads to $a[j_k]f(I[k])a[j_k+1]$ which is identical to $a[i_k]I[k]a[i_k+1]$.

Let in A^*, $J[1]$, ..., $J[s]$ be the super strings which stand in the common adjacencies $a[u_1]a[u_1+1]$, ..., $a[u_s]a[u_s+1]$ in A with respect to M respectively. By Lemma 1 again, there must exist s common adjacencies, say $a[v_1]a[v_1+1]$, ..., $a[v_s]a[v_s+1]$ in A with respect to M', such that $a[v_k]a[v_k+1]$ does not belong to any pair in M'_0 and is identical to $a[u_k]a[u_k+1]$, $1 \leq k \leq s$. Thus following the method as for those which stand in the breakpoints in A, we can set an image of $J[k]$ as $f(J[k]) = J[k]$ or $-J[k]$ and insert it into $a[v_k]a[v_k+1]$. This leads to $a[v_k]f(J[k])a[v_k+1]$ which is identical to $a[u_k]J[k]a[u_k+1]$, $1 \leq k \leq s$. Let A_1^* be the scaffold resulted by these re-insertions of the super strings. Then f must be identical from the set of super strings in A^* to the set of super string in A_1^* for M and M'.

By the same method, we can re-insert those super strings in B^* into B to form a new scaffold B_1^* and set the identical mapping from the set of super strings in B^* to the set of super strings in B_1^*, where any super string in B_1^* does not stand in a common adjacency which belongs to a pair in M'_0.

Since f is identical, there are as many strings of ab in A^* as those in A_1^* for any two symbols a, b in Σ. So many are there in B^* as in B_1^*. Thus there exists a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$, which has $|M(A^*, B^*)|$ pairs, and moreover, as many pairs of (ab, ab) as $M(A^*, B^*)$ has, for any two symbols a, b in Σ. By the method to re-insert those super strings, a 2-string, if it belongs to a pair in M'_0, must occur in A_1^* (B_1^*) and not be involved by a super string in A_1^* (B_1^*). In addition, there are as many strings of ab in $\tau(A_1^*)$ ($\tau(B_1^*)$) without belonging to any pair in M'_0 as those in $\tau(A^*)$ ($\tau(B^*)$) without belonging to any pair in $M_0(A^*, B^*)$ for arbitrary a, b in Σ. Since $M(A^*, B^*)$ has $|M_0(A^*, B^*)|$ original and $|M(A^*, B^*)| - |M_0(A^*, B^*)|$ new pairs, we can set $|M(A^*, B^*)| - |M_0(A^*, B^*)|$ distinct pairs by the 2-strings involved by the super strings in A_1^* and the 2-strings in B_1^* which do not belong to any pair in M'_0, plus the 2-strings involved by the super strings in B_1^* and the 2-strings in A_1^* which do not belong to any pair in M'_0. A maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ can be made by merging M'_0 with these pairs. □

Although in Lemma 22, a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ was confirmed to contain $|M_0(A^*, B^*)|$ original and $|M(A^*, B^*)| - |M_0(A^*, B^*)|$ new pairs, those new pairs of this maximum matching has not been made certain. Actually, to bound $|\alpha(A^*, B^*)|$ for arriving at the algorithm we cry for, it suffices to set the correspondence between the number of 1-type-1 strings in A^* (B^*) and the number of 1-type-1 super strings in A_1^* (B_1^*). This asks us to set those new pairs for that maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ first.

In the following lemma, M, M', A_1^*, B_1^* are the same used as in Lemma 22.

Lemma 23. *There exists a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ with respect to which the common adjacencies in A_1^* and B_1^* are identified such that,*

$$|B\_A_1^*[1,1,M']| = |B\_A^*[1,1,M]|, |C\_A_1^*[1,1,M']| = |C\_A^*[1,1,M]|,$$
$$|B\_B_1^*[1,1,M']| = |B\_B^*[1,1,M]|, |C\_B_1^*[1,1,M']| = |C\_B^*[1,1,M]|.$$

Proof. Let $M(A_1^*, B_1^*)$ be that maximum matching whose original pairs are confirmed by Lemma 22. Let $M_0' = M(A_1^*, B_1^*) \cap M'$, then $|M_0'| = |M_0(A^*, B^*)|$. The remainder is to set $|M(A^*, B^*)| - |M_0(A^*, B^*)|$ new pairs each of which shares no 2-string with any pair in M_0'. We replace the 2-strings of the new pairs in $M(A^*, B^*)$ to meet this aim.

(1) Replace the 2-strings involved by the super strings

By Lemma 22, let in A^*, the super string $I[k]$ stand in the breakpoint $a[i_k]a[i_k+1]$ in A with respect to M, while in A_1^*, $f(I[k])$ stand in the breakpoint $a[j_k]a[j_k+1]$ in A with respect to M', $1 \le k \le t$. If $f(I[k]) = I[k]$, then $a[j_k]f(I[k])a[j_k+1]$ in A_1^* is equal to $a[i_k]I[k]a[i_k+1]$ in A^*. Thus if a new pair in $M(A^*, B^*)$ has a 2-string in $\tau(a[i_k]I[k]a[i_k+1])$, we replace it with the 2-string in $\tau(a[j_k]f(I[k])a[j_k+1])$ at the same position of $a[j_k]f(I[k])a[j_k+1]$ as that of $a[i_k]I[k]a[i_k+1]$, $1 \le k \le t$. If $f(I[k]) = -I[k]$, $a[j_k+1] - f(I[k])a[j_k]$ is equal to $a[i_k]I[k]a[i_k+1]$. Thus if a new pair in $M(A^*, B^*)$ has a 2-string in $\tau(a[i_k]I[k]a[i_k+1])$, we replace it with the 2-string in $\tau(a[j_k]f(I[k])a[j_k+1])$ at the same position of $a[j_k+1] - f(I[k])a[j_k]$ as that of $a[i_k]I[k]a[i_k+1]$, $1 \le k \le t$. This can be done for replacing all 2-strings involved by the super strings in B^* which stand in the breakpoints in B.

By the same method as for replacing the 2-strings involved by the super strings which stand in the breakpoints in A or B, we replace each 2-string which belongs to a new pair in $M(A^*, B^*)$, if it is involved by a super string which stands in a common adjacency in A or B with respect to M. After the afore mentioned replacement, each extra pair in $M(A^*, B^*)$ must turn into a pair with a 2-string in A_1^* and a 2-string in B_1^*, while a new other than extra pair in $M(A^*, B^*)$ only has one 2-string in A_1^* or B_1^* and the other in A^* or B^*. Since an extra pair has two 2-strings replaced by two 2-strings involved by the super strings, it must be extra in a maximum matching between $\tau(A_1^*)$ and $\tau(B_1^*)$ which contains it.

(2) Replace the 2-strings as breakpoints and common adjacencies

Let in B^*, $b[p_1]b[p_1+1]$, ..., $b[p_x]b[p_x+1]$ be the 2-strings each of which belongs to a new pair in $M(A^*, B^*)$, and plays a role of breakpoint in B with

respect to M. Then by Lemma 1, we can find x 2-strings $b[q_1]b[q_1+1]$, ..., $b[q_x]b[q_x+1]$ in B_1^* each of which plays a role of breakpoint in B with respect to M', such that $b[q_k]b[q_k+1]$ is identical to $b[p_k]b[p_k+1]$ for $1 \leq k \leq x$. Thus we replace $b[p_k]b[p_k+1]$ with $b[q_k]b[q_k+1]$ for $1 \leq k \leq x$ for those new pairs in $M(A^*, B^*)$. Symmetrically, we can replace those 2-strings which belong to new pairs in $M(A^*, B^*)$ as breakpoints in A with respect to M with the same number of 2-strings in A_1^* which play the roles of breakpoints in A with respect to M'.

Let in B^*, $b[p'_1]b[p'_1+1]$, ..., $b[p'_y]b[p'_y+1]$ be the 2-strings each of which belongs to a new pair in $M(A^*, B^*)$ and plays a role of common adjacency in B with respect to M. By Lemma 1 and Lemma 22, we can find y 2-strings, say $b[q'_1]b[q'_1+1]$, ..., $b[q'_y]b[q'_y+1]$ in B_1^* such that, (1) $b[q'_k]b[q'_k+1]$ is a common adjacency in B with respect to M' and identical to $b[p'_k]b[p'_k+1]$; (2) $b[q'_k]b[q'_k+1]$ does not belong to any pair in M'_0, $1 \leq k \leq y$. Thus we can replace $b[p'_k]b[p'_k+1]$ with $b[q'_k]b[q'_k+1]$ for the new pairs in $M(A^*, B^*)$, $1 \leq k \leq y$. Symmetrically, for all new pairs in $M(A^*, B^*)$, we can replace their common adjacencies in A with respect to M by the same number of common adjacencies in A with respect to M', which do not belong to any pair in M'_0.

After the replacements of (1) and (2), each new pair in $M(A^*, B^*)$ turns into a pair with a 2-string in A_1^* and a 2-string in B_1^*. Except having all pairs in M'_0, let $M(A_1^*, B_1^*)$ also have those pairs resulted by replacing the 2-strings in the new pairs in $M(A^*, B^*)$. Thus $|M(A_1^*, B_1^*)| = |M(A^*, B^*)|$.

If a super string $I[k]$ in A^* stands in a breakpoint (common adjacency) in A with respect M, then $f(I[k])$ in A_1^* must stand in a breakpoint (common adjacency) in A with respect to M'. This is because by Lemma 22, we always choose a breakpoint or common adjacency in A with respect to M' for the reinsertion of $f(I[k])$, according to whether in A^*, it stands in a breakpoint or common adjacency in A with respect to M. If a super string $I[k]$ in A^* is 1-type-1, $f(I[k])$ must be 1-type-1 in A_1^*. This is because if $I[k]$ in A^* involves two common adjacencies relative to B^*, then by the rule to replace the 2-strings in the new pairs, $f(I[k])$ must involve two common adjacencies in A_1^* relative to B_1^*. Thus $|C\_A_1^*[1,1,M']| = |C\_A^*[1,1,M]|$. For the same reason, $|C\_B_1^*[1,1,M']| = |C\_B^*[1,1,M]|$.

Moreover, if $I[k]$ as a super string in A^* does not involve any common adjacency of an extra pair in $M(A^*, B^*)$, then $f(I[k])$ in A_1^* must not involve any common adjacency of an extra pair in $M(A_1^*, B_1^*)$. This is because the replacement of the 2-strings involved by the super strings can only make as many extra pairs for $M(A_1^*, B_1^*)$ as those in $M(A^*, B^*)$, while the replacement of the other 2-strings in B^* and A^* does not make any extra pair for $M(A_1^*, B_1^*)$. Thus $|B\_A_1^*[1,1,M']| = |B\_A^*[1,1,M]|$. For the same reason, $|B\_B_1^*[1,1,M']| = |B\_B^*[1,1,M]|$. □

By Lemma 22 and 23, Lemma 21 can be further stated as,

Corollary 5. *Let M' be an arbitrary maximum matching with respect to which the common adjacencies in A and B are identified. Then there exist two scaffolds $A_1^* \in A + X$ and $B_1^* \in B + Y$ such that, $|M(A_1^*, B_1^*)| = |\alpha(A^*, B^*)|$, $|M' \cap$*

$M(A_1^*, B_1^*)| = |M_0(A_1^*, B_1^*)|$; $|M'| - |M_0(A_1^*, B_1^*)| \geq |C_-A_1^*[1,1,M']|$, $|M'| - |M_0(A_1^*, B_1^*)| \geq |C_-B_1^*[1,1,M']|$.

Lemma 24. *Let M' be an arbitrary maximum matching between $\tau(A)$ and $\tau(B)$. Then there exist $A_1^* \in A + X$ and $B_1^* \in B + Y$ such that,*

$$|\alpha(A^*, B^*)| \leq |\alpha(A, B)| + \frac{3}{2}(|X| + |Y|) + \frac{1}{2}(|B_-A_1^*[1,1,M']| + |B_-B_1^*[1,1,M']|).$$

Proof. Let A_1^* and B_1^* be those two scaffolds which subjects to Corollary 5. Then,

$$2|C_-A_1^*[1,1,M']| + 2|C_-B_1^*[1,1,M']| - |M'| + |M_0(A_1^*, B_1^*)| \leq \frac{3}{2}(|C_-A_1^*[1,1,M']| + |C_-B_1^*[1,1,M']|) \quad (6)$$

A 1-type-1 super string in $A_1^*[1,1]$ ($B_1^*[1,1]$) other than $B_-A_1^*[1,1,M'] \cup C_-A_1^*[1,1,M']$ ($B_-B_1^*[1,1,M'] \cup C_-B_1^*[1,1,M']$) can involve at least one common adjacency of an extra pair. Thus,

$$|M_1(A_1^*, B_1^*)| \geq |A_1^*[1,1]| - |B_-A_1^*[1,1,M']| - |C_-A_1^*[1,1,M']|, \quad (7)$$
$$|M_1(A_1^*, B_1^*)| \geq |B_1^*[1,1]| - |B_-B_1^*[1,1,M']| - |C_-B_1^*[1,1,M']|. \quad (8)$$

Thus we have,

$$\begin{aligned} &2(|A_1^*[1,1]| - |B_-A_1^*[1,1,M']| - |C_-A_1^*[1,1,M']|) \\ &+2(|B_1^*[1,1]| - |B_-B_1^*[1,1,M']| - |C_-B_1^*[1,1,M']|) - |M_1(A_1^*, B_1^*)| \\ &\leq \frac{3}{2}(|A_1^*[1,1]| - |B_-A_1^*[1,1,M']| - |C_-A_1^*[1,1,M']|) \\ &\quad +\frac{3}{2}(|B_1^*[1,1]| - |B_-B_1^*[1,1,M']| - |C_-B_1^*[1,1,M']|). \end{aligned} \quad (9)$$

Note that $|M'| = |\alpha(A, B)|$. By Lemma 20, the number of common adjacencies between A_1^* and B_1^* can be bounded by,

$$\begin{aligned} |\alpha(A_1^*, B_1^*)| \leq\ &|\alpha(A, B)| + 2|B_-A_1^*[1,1,M']| + 2|B_-B_1^*[1,1,M']| \\ &+\frac{3}{2}(|C_-A_1^*[1,1,M']| + |C_-B_1^*[1,1,M']|) \\ &+\frac{3}{2}(|A_1^*[1,1]| - |B_-A_1^*[1,1,M']| - |C_-A_1^*[1,1,M']|) \\ &+\frac{3}{2}(|B_1^*[1,1]| - |B_-B_1^*[1,1,M']| - |C_-B_1^*[1,1,M']|) \\ &+|A_1^*[1,2]| + |B_1^*[1,2]| \\ &+\sum_{k=2}^{|X|}(k+1)|A_1^*[k,1]| + \sum_{k=2}^{|X|}k|A_1^*[k,2]| \\ &+\sum_{k=2}^{|Y|}(k+1)|B_1^*[k,1]| + \sum_{k=2}^{|Y|}k|B_1^*[k,2]|. \end{aligned} \quad (10)$$

Since $|X| = \sum_{k=1}^{|X|} k|A_1^*[k,1]| + \sum_{k=1}^{|X|} k|A_1^*[k,2]|$, $|Y| = \sum_{k=1}^{|Y|} k|B_1^*[k,1]| + \sum_{k=1}^{|Y|} k|B_1^*[k,2]|$,

$$\begin{aligned}|\alpha(A_1^*, B_1^*)| &\leq |\alpha(A,B)| + 2|B\_A_1^*[1,1,M']| + 2|B\_B_1^*[1,1,M']| \\ &\quad + \frac{3}{2}(|A_1^*[1,1]| - |B\_A_1^*[1,1,M']|) + \frac{3}{2}(|B_1^*[1,1]| - |B\_B_1^*[1,1,M']|) \\ &\quad + \frac{3}{2}(|X| - |A_1^*[1,1]|) + \frac{3}{2}(|Y| - |B_1^*[1,1]|) \\ &= |\alpha(A,B)| + \frac{3}{2}(|X| + |Y|) + \frac{1}{2}(|B\_A_1^*[1,1,M']| + |B\_B_1^*[1,1,M']|) \quad (1)\end{aligned}$$

Since by Corollary 5, $|\alpha(A^*, B^*)| = |\alpha(A_1^*, B_1^*)|$, the proof is done. □

4.2 The algorithm for two-sided scaffold filling

In this subsection, let the maximum matching between $\tau(A)$ and $\tau(B)$ be set as M', with respect to which the common adjacencies in A and B are identified. One may want to get enough k-strings, whose insertions into A or B can increase $k+1$ common adjacencies for A or B. This subsection aims to show that the greedy method of finding insertions for those 1-type-1 super strings can approximate TSSF-MNCA to a performance ratio $\frac{3}{2}$. The following lemma specializes a kind of insertions which can insert one gene into B to increase 2 common adjacencies for B relative to A.

Lemma 25. *If $y \in \Sigma$ occurs in two breakpoints of type 1 in A, and a breakpoint in B has two genes identical to the other two genes in those two breakpoints in A respectively than those two genes of y, then a gene of y in Y can be inserted into B to increase 2 common adjacencies for B.*

Proof. Without loss of generality, let $a[i]a[i+1]$ and $a[j]a[j+1]$ be two breakpoints in A with $a[i]$, $a[j]$ both as genes of y, and $b[k]b[k+1]$ be a breakpoint in B with $b[k] = a[i+1]$, $b[k+1] = a[j+1]$. If a gene of y exists in Y, then the insertion of that gene into $b[k]b[k+1]$ transforms B into B' in which a new 3-string $b[k]yb[k+1]$ arises. Here we also use y to represent a gene of y. We can add $(a[i]a[i+1], b[k]y)$, $(a[j]a[j+1], yb[k+1])$ to M', thus to result in a matching between $\tau(A)$ and $\tau(B')$ with 2 more pairs than those in M'. This implies $|\alpha(A,B')| - |\alpha(A,B)| = 2$. □

Lemma 25 holds too, if in its description, we replace A and B each other and replace Y with X. By Lemma 25, we can find a gene and insert it into B to increase 2 common adjacencies for B. This can be done repeatedly until no such gene can be found. The same method can be used to insert a number of genes in X into A, where each insertion can increase 2 common adjacencies for A. This is presented as GreedyInsert(A,B). In the description of GreedyInsert(A,B), the subroutine OneTypeOne(A,B) returns a gene, say $y \in Y$ and an integer i, such that inserting y between $b[i]$ and $b[i+1]$ can increase 2 common adjacencies for B relative to A. If OneTypeOne(A,B) returns a $y = NULL$, then no gene in Y can be inserted into B to increase 2 common adjacencies for B.

Algorithm *GreedyInsert*(A, B)
1. $(y, i) \leftarrow$ OneTypeOne(A, B);
2. While $(y \neq NULL)$
3. $\{B \leftarrow$ Insert$(y, b[i], b[i+1], B)$; $(y, i) \leftarrow$ OneTypeOne(A, B);$\}$
4. End while
5. $(x, j) \leftarrow$ OneTypeOne(B, A);
6. While $(x \neq NULL)$
7. $\{A \leftarrow$ Insert$(x, a[j], a[j+1], A)$; $(x, j) \leftarrow$ OneTypeOne(B, A);$\}$
8. End while
9. Return A, B.

It takes $O(N^2)$ time to find a gene for an insertion into B to increase 2 common adjacencies by OneTypeOne(A, B). Thus the time complexity of this algorithm is $O(N^3)$, where $N = |A| + |X| = |B| + |Y|$. In general terms, GreedyInsert(A, B) cannot insert all genes in Y (X) into B (A). Let $A_1^* \in A + X$, $B_1^* \in B + Y$ be those two scaffolds which subjects to Lemma 24. Then the number of common adjacencies contributed by GreedyInsert(A, B) can be evaluated by,

Lemma 26. *Let A', B' be the scaffolds returned by GreedyInsert(A, B). Then $|\alpha(A', B')| \geq |\alpha(A, B)| + |B\_A_1^*[1, 1, M']| + |B\_B_1^*[1, 1, M']|$.*

Proof. In GreedyInsert(A, B), a gene is always inserted into the breakpoint in A or B. Let in B_1^*, $B\_B_1^*[1, 1, M'] = \{y[1], ..., y[t]\}$, where $y[k]$ stands in the breakpoint $b[j_k]b[j_k + 1]$ in B $(1 \leq k \leq t)$. Symmetrically, let in A_1^*, $B\_A_1^*[1, 1, M'] = \{x[1], ..., x[s]\}$, where $x[k]$ stands in the breakpoint $a[i_k]a[i_k+1]$ in A $(1 \leq k \leq s)$. Let $B_0(B) = \{b[j_k]b[j_k+1] | 1 \leq k \leq t\}$, $B_0(A) = \{a[i_k]a[i_k+1] | 1 \leq k \leq s\}$. Note that $y[k] \in B\_B_1^*[1, 1, M']$ is also a gene, and so is $x[k] \in B\_A_1^*[1, 1, M']$.

Firstly, we evaluate the number of the genes in $B\_B_1^*[1, 1, M']$ which can be inserted into the breakpoints in B by GreedyInsert(A, B).

(1) If a gene, say $y[k]$, is just inserted into $b[j_k]b[j_k+1]$ by GreedyInsert(A, B), then this insertion increases 2 common adjacencies for B and leaves the other breakpoints in $B_0(B)$ than $b[j_k]b[j_k + 1]$ for the rest insertions.

(2) If a gene, say $y[k]$, is inserted into $b[j_l]b[j_l + 1]$ by GreedyInsert(A, B), $k \neq l$, this insertion must increase 2 common adjacencies for B. This make it impossible for GreedyInsert$(\bullet)$ to insert $y[k]$ into $b[j_k]b[j_k + 1]$ and $y[l]$ into $b[j_l]b[j_l + 1]$. In this case, the breakpoints in $B_0(B)$ except $b[j_k]b[j_k + 1]$ and $b[j_l]b[j_l + 1]$ can all be left for the rest insertions.

(3) An insertion of a gene in $B\_B_1^*[1, 1, M']$ into B can also make it impossible to insert a gene in $B\_A_1^*[1, 1, M']$ into A. If a gene, say $y[k]$, is inserted into a breakpoint, say $b[i_l]x[l]$ or $x[l]b[i_l + 1]$ in B to increase 2 common adjacencies for B, then this insertion make it impossible for GreedyInsert(A, B) to insert $y[k]$ into $b[j_k]b[j_k + 1]$ and $x[l]$ into $a[i_l]a[i_l + 1]$. In this case, the breakpoints in $B_0(B)$ except $b[j_k]b[j_k + 1]$ and the breakpoints in $B_0(A)$ except $a[i_l]a[i_l + 1]$ can all be left for the rest insertions.

Namely, a greedy insertion of a gene in $B\_B_1^*[1, 1, M']$ may reject at most two other genes in $B\_B_1^*[1, 1, M']$, or one gene in $B\_B_1^*[1, 1, M']$ and one gene in

$B\_A_1^*[1,1,M']$ for their right insertions. Thus in GreedyInsert(A,B), if the insertions of the genes in $B\_B_1^*[1,1,M']$ reject K genes in $B\_A_1^*[1,1,M']$ to be inserted into their right breakpoints, then at least $\frac{1}{2}(t-K)$ genes in $B\_B_1^*[1,1,M']$ can be inserted into the breakpoints in $B_0(B)$, along with K genes in $B\_B_1^*[1,1,M']$ inserted into the other K breakpoints in B rather than $B_0(B)$. Each insertion of these genes can increase 2 common adjacencies for B relative to A.

Secondly, by the same argument, at least $\frac{1}{2}(s-K)$ genes in $B\_A_1^*[1,1,M']$ can be inserted into the breakpoints in A, to increase $s-K$ common adjacencies for A relative to B.

Finally, $|\alpha(A',B')| - |\alpha(A,B)| \geq (t-K)+2K+s-K = s+t = |B\_A_1^*[1,1,M']| + |B\_B_1^*[1,1,M']|$. □

Corollary 6. *Let A', B' be the scaffolds returned by GreedyInsert(A,B). Then*

$$|\alpha(A',B')| - |\alpha(A,B)| \geq |c(A') - c(A)| + |c(B') - c(B)| + \frac{1}{2}|B\_A_1^*[1,1,M']| + \frac{1}{2}|B\_B_1^*[1,1,M']|. \quad (12)$$

Proof. Since an insertion of a gene in GreedyInsert(A,B) always increase 2 common adjacencies, thus $|\alpha(A',B')| - |\alpha(A,B)| = 2|c(A') - c(A)| + 2|c(B') - c(B)|$. Moreover, since $|\alpha(A',B')| - |\alpha(A,B)| \geq |B\_A_1^*[1,1,M']| + B\_B_1^*[1,1,M']$ by Lemma 26, $|\alpha(A',B')| - |\alpha(A,B)| \geq |c(A') - c(A)| + |c(B') - c(B)| + \frac{1}{2}$ $|B\_A_1^*[1,1,M']| + \frac{1}{2}$ $|B\_B_1^*$ $[1,1,M']|$. □

If GreedyInsert(A,B) returns A', B' with $c(A') - c(B') \neq \emptyset$ or $c(B') - c(A') \neq \emptyset$, we can use SimpleInsert(A',B') to insert those genes in $c(A') - c(B')$ into B', then use SimpleInsert$(B",A')$ to insert those genes in $c(B') - c(A')$ into A'. Finally, the algorithm for solving TSSF-MNCA is given briefly as TwoSideInsert(A,B).

Algorithm *TwoSideInsert(A,B)*
Input: Scaffolds A,B, where $X = c(B) - c(A)$, $Y = c(A) - c(B)$.
Output: $A" \in A + X$, $B" \in B + Y$.
1. $(A',B') \leftarrow$ GreedyInsert(A,B);
2. $B" \leftarrow$ SimpleInsert(A',B');
3. $A" \leftarrow$ SimpleInsert$(B",A')$;
4. Return $A"$, $B"$.

Since the time complexity of GreedyInsert(A,B) is $O(N^3)$ and so is that of SimpleInsert (A,B), the time complexity of this algorithm is $O(N^3)$. We evaluate the number of common adjacencies between those two scaffolds returned by TwoSideInsert(A,B).

Lemma 27. *Let $A"$, $B"$ be the scaffolds returned by TwoSideInsert(A,B). Then* $|\alpha(A",B")| - |\alpha(A,B)| \geq (|X|+|Y|) + \frac{1}{2}$ $(|B\_A_1^*[1,1,M']| + |B\_B_1^*[1,1,M']|)$.

Proof. Let A', B' be the scaffolds returned by GreedyInsert(A,B) in step 1. By Corollary 6, $|\alpha(A',B')| - |\alpha(A,B)| \geq |c(A')| - |c(A)| + |c(B')| - |c(B)| +$

$\frac{1}{2}|B\_A_1^*[1,1,M']| + \frac{1}{2}|B\_B_1^*[1,1,M']|$. By Lemma 18, $|\alpha(A", B")| - |\alpha(A', B')| \geq |c(A")| - |c(A')| + |c(B")| - |c(B')|$. Thus, $|\alpha(A", B")| - |\alpha(A, B)| \geq |c(A")| - |c(A)| + |c(B")| - |c(B)| + \frac{1}{2}|B\_A_1^* [1,1,M']| + \frac{1}{2} |B\_B_1^*[1,1,M']| = |X| + |Y| + \frac{1}{2} |B\_A_1^* [1,1,M']| + \frac{1}{2} |B\_B_1^*[1,1,M']|$. □

Theorem 2. $\frac{|\alpha(A^*,B^*)|-|\alpha(A,B)|}{|\alpha(A",B")|-|\alpha(A,B)|} \leq \frac{3}{2}$.

Proof. By Lemma 24 and Lemma 27, and the fact $|B\_A_1^*[1,1,M']| + |B\_B_1^*[1,1,M']| \geq 0$, the proof is done. □

This theorem indicates that the algorithm can approximate TSSF-MNCA to a performance ration $\frac{3}{2}$. For example, let $A = \#ax1234\#$, $B = \#ay1342\#$. Thus $X = \{y\}$, $Y = \{x\}$. Since either insertion of x and y cannot increase two common adjacencies, GreedyInsert(A, B) must return $A' = A$ and $B' = B$. Then x may be inserted between a and y in B by SimpleInsert(A', B'), and y may be inserted between 1 and 2 in A by SimpleInsert$(B", A')$. These two insertions result in $A" = \#ax1y234\#$ and $B" = \#axy1342\#$. Thus, $|\alpha(A", B")| - |\alpha(A, B)| = 2$. However, the optimal solution of this instance is $A^* = \#axy1234\#$, $B^* = \#axy1342\#$, with $|\alpha(A^*, B^*)| - |\alpha(A, B)| = 3$.

5 Conclusion

In this paper, we have presented an approximation algorithm for the two-sided scaffold filling problem aiming for the maximum number of common adjacencies. This algorithm can always return a solution with an approximation ratio of $\frac{3}{2}$ and has a time complexity of $O(N^3)$, where N is the number of genes in the genomes. We believe that the polynomial time algorithm for the scaffolds without any breakpoints of type 1 can also be used in the future for some improved algorithms. This approximation algorithm can be improved for its performance ratio by finding more 1-strings to increase 2 common adjacencies and 2-strings to increase 3 common adjacencies for filling of the scaffolds. It would be interesting how to find more such kind of strings.

Acknowledgments

This research is supported by the National Nature Science Foundation of China (61070019, 61202014), the Natural Science Foundation of Shandong Province (ZR-2012-Z002), and the Chinese Postdoctoral Science Foundation (2011-M-501133, 2012-T-50614).

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 nontrivial genomes cannot be approximated. *Proc. 3nd Workshop on Algorithm and Computation*, LNCS 5431, pp. 357-368, 2009.
3. 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.
4. Z. Chen, R. Fowler, B. Fu and B. Zhu. On the inapproximability of the exemplar conserved interval distance problem of genomes. *J. Combinatorial Optimization*, 15(2):201-221, 2008.
5. Z. Chen, B. Fu and B. Zhu. The approximability of the exemplar breakpoint distance problem. *Proc. 2nd International Conf. on Algorithmic Aspects in Information and Management* ($AAIM'06$), LNCS 4041, pp. 291-302, 2006.
6. D. H. Huson, K. Reinert and E. W. Myers. The Greedy Path-Merging Algorithm for Contig Scaffolding. *Journal of the ACM*, 49(5): 603-615, September 2002.
7. A. Goldstein, P. Kolman and J. Zheng. Minimum common string partitioning problem: Hardness and approximations. *Proc.15th International Symposium on Algorithms and Computation* ($ISAAC'04$), LNCS 3341, pp. 473-484, 2004.
8. M. Jiang. The zero exemplar distance problem. *Proc. of the 2010 International RECOMB-CG Workshop (RECOMB-CG'10)*, LNBI 6398, pages 74-82, 2010.
9. H. Jiang, C. Zheng, D. Sankoff and B. Zhu. Scaffold Filling under the Breakpoint and Related Distances. *IEEE/ACM Trans. Comput. Biology Bioinform*,9(4): 1220-1229, 2012.
10. H. Jiang, F. Zhong and B. Zhu. Filling scaffolds with gene repetitions: maximizing the number of adjacencies. *Proc. 22nd Annual Symposium on Combinatorial Pattern Matching* ($CPM'11$), LNCS 6661, pp. 55-64, 2011.
11. 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.
12. D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11): 909-917, 1999.
13. N. Liu, H. Jiang, D. Zhu and B. Zhu. An Improved Approximation Algorithm for Scaffold Filling to Maximize the Common Adjacencies. *The 19th Annual International Computing and Combinatorics Conference* ($COCOON'13$), LNCS 7936, pp. 397-408, 2013. *IEEE/ACM Trans. Comput. Biology Bioinform*, 10(4):905-913, 2013.
14. N. Liu and D. Zhu. The algorithm for two sided scaffold filling problem. *The 10th Annual International Conference on Theory and Applications of Models of Computation* ($TAMC2013$), LNCS 7876, pp. 236-247, 2013.