Chapter 9
A Retrospective on Genomic Preprocessing for Comparative Genomics

Binhai Zhu

Abstract In this paper, we present a survey of research on genomic preprocessing for comparative genomics, i.e., handling genomes with gene repetitions, missing or redundant genes, initiated by David Sankoff in 1999. The development of this research ends with several interesting results within and beyond computational biology and bioinformatics, with possible new contributions in the future. We will describe the history of development of this research and review the current status of the corresponding problems. For the problem of handling missing genes (scaffold filling), we also present some technical details which are not given in the previous papers. Some open problems will be listed at the end for further research.

9.1 Introduction

In computational biology, we constantly need to process various biological data to extract meaningful biological relation, like building a phylogenetic tree. Such a process sometimes involves computing the genomic distance between two genomes, which was first investigated as early as in 1926 [61, 62]. The problem was more formally studied in 1990s and is in general polynomially solvable for signed genomes, e.g., under the signed translocation distance [7, 37, 49, 56], under the signed reversal distance [3, 38, 48, 63, 64], and under the DCJ distance [69]. For unsigned genomes, the problems are typically NP-hard, e.g., sorting by reversals [17], sorting by translocations [71], sorting by DCJ operations [19], and sorting by transpositions [16]. But these problems on sort unsigned genomes do admit small-factor ($\leq$1.5) polynomial-time approximations, e.g., sorting by reversals [9, 28], sorting by translocations [31, 46], sorting by DCJ operations [19, 20, 42], and sorting by transpositions [33].

The above results are all under the assumption that each genome is given in a form where there is no loss and duplication of genes and a genome is represented as a permutation of genes. For many genomes, due to the fast evolution/self-reproduction process, duplicated (paralogous) genes are common. So it is useful to

B. Zhu (✉)
Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA
e-mail: bhz@cs.montana.edu

C. Chauve et al. (eds.), *Models and Algorithms for Genome Evolution*,
Computational Biology 19, DOI 10.1007/978-1-4471-5298-9_9,

select the ancestral ortholog of a gene family on an evolutionary basis. In 1999, David Sankoff first formulated this problem as an algorithmic problem, now known as the *Exemplar Breakpoint/Genomic Distance* problem [59]. In Sect. 9.2, we will survey the development of the follow-up research since 1999, mostly with negative complexity results. Some of these methods and results have already been applied in other (biological and non-biological) problems [6, 58, 67].

In some eukaryotic genomes, under many situations, like sequencing error or errors due to an inappropriate design of the biological experiments, we might have noise and redundant genes. Before eliminating these redundancies, using the given genomes for many biological studies might introduce further errors. While this problem was known to the biologists long time ago, in 2007 David Sankoff again first formulated this as an algorithmic problem, now known as the *Maximal Strip Recovery* and the *Complementary Maximal Strip Recovery* problems [27, 70]. This again led to a series of research on fixed-parameter tractable and approximation algorithms, performed by several groups in US, Canada, Europe and China. In Sect. 9.3, we will survey the most recent development of these researches.

Genome sequencing has been a hot research area for the last 20 years. Behind the huge success a commonly ignored fact is that most genomes sequenced are not really 'sequences'; in fact, most of them are made of scaffolds, i.e., composed of incomplete gene markers. David Sankoff and his group initiated this problem of scaffold filling in 2010 [54]. My group and a group led by Prof. Daming Zhu at Shandong University (China) have been following up this research. While initially the work was done on filling scaffolds with no gene repetitions, which is a problem polynomially solvable, recently a lot of effort has been put on filling scaffolds with gene repetitions (which is in general NP-hard). In Sect. 9.4, we will survey the current status of this research.

In the area of bioinformatics and computational biology, for a lot of NP-complete problems one would typically apply three methods to handle them. One is to find an approximation solution, with the requirement being that the approximation factor is small (better close to one). The other is to look for an exact solution (FPT algorithm) when some parameter (say, the solution size) of the problem is small. The vast majority of practical solutions for bioinformatics and computational biology are heuristic ones, which are possibly based on some formal methods like integer linear programming, branch-and-bound, etc.

In this survey, we focus on the approximability and fixed-parameter tractability results for the above three general problems related to computing genomic distance with some preprocessing. In these problems, we are given some genomes or genetic maps and we try to optimize some solution values by deleting some genes or gene markers. So these problem fit naturally for approximation and/or FPT solutions. Unfortunately, as we will review a bit later, some of these problems are very hard in both aspects. In other words, it might be impossible to design good approximation and/or FPT algorithms for them, unless $P = NP$, $NP = ZPP$ or $FPT = W[1]$. On the other hand, many problems are still open along these lines.

The paper is organized as follows. In Sect. 9.2, we first review the approximability and fixed-parameter tractability for the Exemplar Breakpoint Distance (EBD)

problem. We then review the approximability and fixed-parameter tractability for the Exemplar Non-breaking Similarity (ENbS) problem (which is the dual of EBD). In Sect. 9.3, we review the approximability and fixed-parameter tractability for the Maximal Strip Recovery (MSR) problem and its complement, the Complementary Maximal Strip Recover (CMSR) problem. In Sect. 9.4, we review the approximation results for the Scaffold Filling Problem, focusing on the One-sided Scaffold Filling Problem with Gene Repetitions. In Sect. 9.5, we list a set of open problems to conclude this paper.

9.2 The Exemplar Breakpoint Distance and Related Problems

As we have covered in the introduction, in the genome comparison and rearrangement area, a standard problem is to compute the number (i.e., genetic distance) and the actual sequence of genetic operations which converts a source genome to a target genome. This problem is important in evolutionary molecular biology as it gives some useful information on genome evolution. Typical genetic distances include edit [53], signed reversal [4, 38, 52, 57] and breakpoint [66], etc. In fact, the idea of signed reversal and, implicitly, breakpoint, was initiated as early as in 1926 by Sturtevant [61]. In the past years, conserved interval distance was also proposed to measure the similarity of multiple sequences of genes [8]. Interested readers are referred to [35] for a summary of the research performed in this area.

In genome rearrangement research, it is usually assumed that each gene appears in a genome exactly once. Under this assumption, the genome rearrangement problem is in essence the problem of comparing and sorting signed permutations [35, 38]. However, this assumption is very restrictive and is only justified in several small virus genomes. For example, this assumption does not hold on eukaryotic genomes where paralogous genes exist [55, 59]. So we have to handle this gene duplication problem.

David Sankoff first considered the problem of computing the breakpoint distance with duplicated genes. In [59], Sankoff proposed a way to select, from the duplicated copies of genes, the common ancestor gene such that the breakpoint distance between the reduced genomes (*exemplar genomes*) is minimized. The distance is called the *exemplar breakpoint distance* henceforth. A general branch-and-bound algorithm was also implemented in [59]. In [55], Nguyen, Tay and Zhang proposed to use a divide-and-conquer method to compute the exemplar breakpoint distance empirically.

For the theoretical part of research, it was shown that both of the problems of computing the signed reversal and breakpoint distances between exemplar genomes are NP-complete [14]. A few years ago, Blin and Rizzi further proved that computing the conserved interval distance between exemplar genomes is NP-complete [11]; moreover, it is NP-complete to compute the minimum conserved interval matching (i.e., without deleting the duplicated copies of genes). Starting in 2005, we showed much stronger inapproximability results for the exemplar breakpoint and conserved

interval distance problems (even under a weaker model of approximation) [21, 24]. (In fact, a series of workshops were organized at University of Texas—Pan American between 2005 and 2008, focusing on this topic.) While various exemplar genomic distances have been researched before, in this survey we will focus on the exemplar breakpoint distance. In fact, all the inapproximability result for exemplar breakpoint distance holds for any other genomic distance $d(-,-)$ satisfying $d(G,H)=0$ implies $G=H$ or $G=-H$.

9.2.1 Problem Definitions

In the genome comparison and rearrangement problem, we are given a set of genomes, each of which is a signed sequence of genes where the order of the genes corresponds to the position of them on the linear chromosome and the signs correspond to which of the two DNA strands the genes are located. Here we interpret a genome as a set of such sequences (chromosomes), though we focus mostly on *singleton* genomes, i.e., a single sequence, in this paper. When the input genomes contain gene repetitions, Sankoff proposed a method to select an *exemplar genome*, by deleting redundant copies of a gene, such that in an exemplar genome any gene appears exactly once; moreover, the resulting exemplar genomes should have a property that a given genetic distance between them is minimized [59].

The following definitions are very much following those in [11]. Given n *gene families* (alphabet) $\mathcal{F}$, a genome $\mathcal{G}$ is a sequence of elements of $\mathcal{F}$ such that each element is with a sign ($+$ or $-$). In general, we allow the repetition of a gene family in any genome. Each occurrence of a gene family is called a *gene*, though we will not try to distinguish a gene and a gene family if the context is clear. Given a genome with no repetition of any gene $G=g_1g_2\ldots g_m$, we say that gene g_i *immediately precedes* g_j if $j=i+1$. Given genomes G, H (with no gene repetition), if gene a immediately precedes b in G and neither a immediately precedes b nor $-b$ immediately precedes $-a$ in H, then they constitute a *breakpoint* in G. The *breakpoint distance* is the number of breakpoints in G (symmetrically, it is the number of breakpoints in H), denoted as $\mathrm{bd}(G,H)$.

The number of a gene g appearing in a genome $\mathcal{G}$ is called the cardinality of g in $\mathcal{G}$, written as $card(g,\mathcal{G})$. A gene in $\mathcal{G}$ is called *trivial* if g has cardinality exactly 1; otherwise, it is called *non-trivial*. A genome $\mathcal{G}$ is called r*-repetitive*, if all the genes from the same gene family appear at most r times in $\mathcal{G}$. For example, $\mathcal{G}=c-adc-bdeb$ is 2-repetitive.

Given a genome $\mathcal{G}$ over $\mathcal{F}$, an *exemplar genome* of $\mathcal{G}$ is a genome G' obtained from $\mathcal{G}$ by deleting duplicating genes such that each gene family in $\mathcal{G}$ appears exactly once in G'. For example, let $\mathcal{G}=-bcaadag-e$, there are two exemplar genomes: $-bcadg-e$ and $-bcdag-e$.

The Exemplar Breakpoint Distance (EBD) problem is defined as follows:

Instance: Genomes $\mathcal{G}$ and $\mathcal{H}$, each is of length $O(m)$ and each covers n gene families (i.e., at least one gene from each of the n gene families appears in both $\mathcal{G}$ and $\mathcal{H}$); integer K.

Question: Are there two respective exemplar genomes of $\mathcal{G}$ and $\mathcal{H}$, G and H, such that $\text{bd}(G, H) \leq K$?

9.2.2 Algorithmic Foundations

In the next subsection, we present some hardness results on the approximability and fixed-parameter tractability for EBD, namely, the hardness to compute or approximate the minimum value K in the above formulation. Here we give some standard definitions regarding approximation and FPT algorithms. Given a minimization (maximization) problem Π, let the optimal solution value of Π be OPT. We say that an approximation algorithm $\mathcal{A}$ provides a *performance guarantee* of α for Π if for every instance I of Π, the solution value returned by $\mathcal{A}$ is at most $\alpha \times \text{OPT}$ (at least OPT/α). Usually we say that $\mathcal{A}$ is a factor-α approximation for Π. For the obvious reason, we are only interested in polynomial-time approximation algorithms. Readers are referred to [29, 34] for more details regarding the definitions related to approximation algorithms and NP-completeness.

As a well-known subject as well, an FPT algorithm for a decision problem with parameter k is an algorithm which solves the problem in $O(f(k)n^c)$ time, where f is any function only on k and c is some fixed constant not related to k. More details on FPT algorithms can be found in [32].

9.2.3 Hardness Results

In [21], we presented the first set of inapproximability results for the Exemplar Breakpoint Distance problem, given two genomes each containing only one sequence of genes drawn from n gene families. We showed that even if a gene appears at most three times, deciding whether the optimal exemplar breakpoint distance is zero, i.e, whether $G = H$, is NP-complete. It was left as an open problem whether the result holds when each gene appears at most twice in each of the input genomes [2, 21]. Recently, this open question was finally answered, i.e., it remains NP-complete even when each gene appears at most two times [13, 47]. Combining these results, we have the following inapproximability result.

Theorem 1 *If both $\mathcal{G}$ and $\mathcal{H}$ are 2-repetitive genomes, then the Exemplar Breakpoint Distance problem does not admit any polynomial-time approximation (regardless of its approximation factor), unless $P = NP$.*

Proof If we view the Exemplar Breakpoint Distance problem as a minimization problem, then the result in [13], with an example presented at the end of this subsection, implies that deciding whether $\text{OPT} = 0$ is NP-complete (even if the input

genomes are 2-repetitive). Let $\mathcal{A}$ be any approximation algorithm for EBD with factor α. By definition, $\mathcal{A}$ returns an approximation solution value APP, with

$$\text{APP} \leq \alpha \times \text{OPT}.$$

When $\text{OPT} = 0$, clearly APP must also satisfy $\text{APP} = 0$. In other words, $\mathcal{A}$ would be able to solve the instance in [13] in polynomial time. This, however, contradicts with the corresponding NP-completeness result (unless $\text{P} = \text{NP}$). □

Regarding the fixed-parameter intractability for EBD, we have the following theorem.

Theorem 2 *If both $\mathcal{G}$ and $\mathcal{H}$ are 2-repetitive genomes, then the Exemplar Breakpoint Distance problem does not admit any FPT algorithm, unless $P = NP$.*

Proof Again, if we view the Exemplar Breakpoint Distance problem as a minimization problem, then the result in [13, 47] implies that deciding whether $\text{OPT} = 0$ is NP-complete (even if the input genomes are 2-repetitive). Let $\mathcal{B}$ be any FPT algorithm for EBD which runs in $O(f(k)n^c)$ time. When $\text{OPT} = k = 0$, $\mathcal{B}$ solves EBD in $O(f(0)n^c) = O(n^c)$ time. In other words, $\mathcal{B}$ would be able to solve the instance in [13] in polynomial time. This, again, contradicts with the corresponding NP-completeness result, unless $\text{P} = \text{NP}$. □

On the other hand, it is necessary to point out that the reduction in [21, 24] is much simpler than in [13, 47]. As a matter of fact, it has been applied to show the NP-hardness of other problems in computational geometry [6], computational biology [67] and program download [58]. We show a simple example on this reduction.

Given a 3SAT formula $\phi = F_1 \wedge F_2 \wedge F_3 \wedge F_4$, where $F_1 = (x_1 \vee \overline{x_2} \vee x_3)$, $F_2 = (\overline{x_1} \vee x_2 \vee \overline{x_4})$, $F_3 = (\overline{x_2} \vee \overline{x_3} \vee x_4)$, and $F_4 = (x_1 \vee \overline{x_3} \vee \overline{x_4})$, we want to find a truth assignment for ϕ. For each variable x_i, define S_i (resp. S_i') as the list of clauses containing x_i (resp. $\overline{x_i}$) followed by clauses containing $\overline{x_i}$ (resp. x_i). So $S_1 = F_1F_4F_2$ and $S_1' = F_2F_1F_4$, etc.

Then we construct two sequence $\mathcal{G} = S_1g_1S_2g_2S_3g_3S_4$, $\mathcal{H} = S_1'g_1S_2'g_2S_3'g_3S_4'$, where g_j's are peg genes only appearing once. Each gene appears at most three times as each clause contains three literals. The truth assignment can be set as follows: if $x_i = \text{TRUE}$, then keep the clauses in S_i and S_i' which contain x_i; if $x_i = \text{FALSE}$, then keep the clauses in S_i and S_i' which contain $\overline{x_i}$. If there are still duplicated clauses after this, then keep one such clause and delete the remaining ones arbitrarily. Regarding the above example, we can have $x_1 = x_3 = \text{TRUE}$, $x_2 = x_4 = \text{FALSE}$. So the corresponding exemplar genomes obtained are $G = H = F_4g_1F_3g_2F_1g_3F_2$, whose breakpoint distance is zero.

In different applications, F_i's and g_j's can be constructed to fit the corresponding problems, for instance as geometric points [6, 67] or programs to be downloaded [58].

9.2.4 The Complement Problem—ENbS

We comment that the negative results in Sect. 9.2.3 hold for any genomic distance $d(-,-)$ satisfying that $d(G,H)=0$ implies $G=H$ or $G=-H$. This, of course, implies that all the exemplar genomic distance problems (like exemplar reversal, exemplar transposition, and exemplar conserved interval distances) do not admit any polynomial-time approximation algorithms or any FPT algorithm, unless P = NP.

There have been two ways to handle this problem. One is to use a weak model of approximation, which will be covered as related to open problems in Sect. 9.5. The other, on the other hand, is to use a different similarity measure. In this case, one would try to maximize certain similarity measure. The most notable of such measures include non-breaking similarity (or number of adjacencies) [23] and the number of common intervals [12]. (A common interval is a pair of substrings appearing in the two genomes with the same genes, but possibly different orders. Example. $G=abced$, $H=deacb$. (abc, acb) is a length-3 common interval.) We will focus on the non-breaking similarity, which is really the complement of the breakpoint distance.

For two exemplar genomes G and H over the same alphabet of size n, recall that a breakpoint in G is a two-gene substring g_ig_{i+1} such that neither g_ig_{i+1} nor $-g_{i+1}-g_i$ is a substring in H. A *non-breaking point* (or an *adjacency*) is a common two-gene substring g_ig_{i+1} that appears either as g_ig_{i+1} or as $-g_{i+1}-g_i$ in G and H. The number of non-breaking points between G and H is also called the *non-breaking similarity* between G and H, denoted as $\mathrm{nbs}(G,H)$. Clearly, we have $\mathrm{nbs}(G,H)+\mathrm{bd}(G,H)=n-1$. For two genomes $\mathcal{G}$ and $\mathcal{H}$, their *exemplar non-breaking similarity* $\mathrm{enbs}(\mathcal{G},\mathcal{H})$ is the maximum $\mathrm{nbs}(G,H)$, where G and H are exemplar genomes derived from $\mathcal{G}$ and $\mathcal{H}$. Again we have $\mathrm{enbs}(\mathcal{G},\mathcal{H})+\mathrm{ebd}(\mathcal{G},\mathcal{H})=n-1$.

The Exemplar Non-breaking Similarity (ENbS) problem is formally defined as follows:

Instance: Genomes $\mathcal{G}$ and $\mathcal{H}$, each is of length $O(m)$ and each covers n gene families (i.e., at least one gene from each of the n gene families appears in both $\mathcal{G}$ and $\mathcal{H}$); integer K.

Question: Are there two respective exemplar genomes of $\mathcal{G}$ and $\mathcal{H}$, G and H, such that the non-breaking similarity between them is at least K?

We have the following negative results which have been proved in [23, 26].

Theorem 3 *If one of $\mathcal{G}$ and $\mathcal{H}$ is exemplar and the other is 2-repetitive, then the Exemplar Non-breaking Similarity problem does not admit any factor-$n^{0.5-\epsilon}$ polynomial-time approximation unless NP = ZPP.*

Proof We give a sketch of proof from [23, 26]. In [23, 26], it was shown that Independent Set can be linearly reduced to ENbS; i.e., the input graph has an independent set of size k iff the constructed ENbS instance has a non-breaking similarity (or number of adjacencies) equal to k. As Independent Set cannot be approximated

within a factor of $|V|^{1-\epsilon}$ unless NP = ZPP [39] and as in the reduction we use $\Theta(|V|^2)$ genes (where $|V|$ is the number of vertices in the input graph), the theorem follows. □

In [26], a factor-$O(\sqrt{n})$ approximation was presented for ENbS, show that the above inapproximability result is tight.

Theorem 4 *If one of $\mathcal{G}$ and $\mathcal{H}$ is exemplar and the other is 2-repetitive, the Exemplar Non-breaking Similarity problem does not admit an FPT algorithm unless FPT = W[1].*

Proof It is noted that the reduction from Independent Set to ENbS in [23, 26] is in fact an FPT reduction. As Independent Set is W[1]-complete [32], the theorem simply follows. □

In fact, with the lower bound results proved in [18], Independent Set (hence ENbS) cannot be solved in $O(f(k)n^{o(k)})$ time even if k is bounded by an arbitrarily small function of n, unless ETH fails. (ETH—Exponential Time Hypothesis: 3SAT cannot be solved in subexponential time.)

In the next section, we will survey another problem initiated by David Sankoff on computing syntenic blocks from genetic maps.

9.3 Maximal Strip Recovery and Its Complement

In a genome or physical map, the distance between two genes is exact. This is different in a genetic map, where only the relative positions between gene markers along chromosomes are indicated. A genetic map is usually constructed from DAGs (Directed Acyclic Graphs) which represent the partial order of gene markers. We omit the construction of genetic maps and interested readers are referred to [10, 68]. It should be noted that in a genetic map all the gene markers are distinct.

Given two genetic maps G and H represented by a sequence of n gene markers, a *strip* (syntenic block) is a sequence of distinct markers of length at least two which appear as subsequences in both of the input maps, either directly or in reversed and negated form. The problem *Maximal Strip Recovery* (MSR) is to find two subsequences G' and H' of G and H, respectively, such that the total length of disjoint strips in G' and H' is maximized An example is as follows: $G = abcdefgh$, $H = h - g - fcbdae$ and the optimal solution is $G' = cdefg$ and $H' = -g - fcde$, each containing two syntenic blocks cde and fg.

The MSR problem was proposed to handle the elimination of noise and ambiguities in genetic maps. This is related to the well-known problem in comparative genomics—to decompose two given genomes into syntenic blocks, i.e., segments of chromosomes which are deemed to be homologous in the two input genomes. In 2007, a heuristic method was proposed to handle the MSR problem [27, 70]. In [25], a factor-4 polynomial-time approximation algorithm was proposed for the problem. This was done by applying the Maximum Weight Independent Set on 2-interval

graphs, which admit a factor-4 approximation [5]. We also proved that several close variants of MSR, MSR-d (with $d > 2$ input maps), MSR-DU (with marker duplications), and MSR-WT (with markers weighted) are all NP-complete. It was left as an open problem whether the problem can be solved in polynomial time or is NP-complete [25].

Recently, in [65] we showed that MSR is in fact NP-complete, via a polynomial-time reduction from One-in-Three 3SAT (which was shown to be NP-complete in [34, 60]). We summarize the results in [25, 65] as follows.

Theorem 5 *MSR is NP-complete, and it admits a factor-*4 *polynomial-time approximation.*

As an effort to solve the MSR problem practically, we tried to handle MSR by solving its complement (CMSR) with FPT algorithms, i.e., showing that CMSR is fixed-parameter tractable [65]. Note that CMSR is a minimization problem where one deletes some markers such that the remaining ones in the genetic maps all belong to some syntenic blocks. With the previous example $G = abcdefgh$ and $H = h - g - fcbdae$, the optimal CMSR solution is to delete markers a, b, h.

Let k be the minimum number of markers deleted in some optimal solution of CMSR, the running time of known algorithms are $O(3^k n + n^2)$ [43], and $O(2.36^k n + n^2)$ [15]. In [45], we proved a $18k$ parameterized search space for CMSR and subsequently obtained a linear kernel of size (the actual size should be $78k$, slight better than in the conference version). Combining all these results, we have the following theorem.

Theorem 6 *Let k be the optimal number of gene markers deleted from the input genetic maps. CMSR can be solved in $O(2.36^k k + n^2)$ time; i.e., CMSR is fixed-parameter tractable.*

Note that as k is typically greater than 50 in real datasets, our FPT algorithms are not yet practical.

At the same time, approximation algorithms are presented for CMSR in the last couple of years. In [43], a factor-3 approximation was presented. The current best approximation factor is 2.33 [50]. Further improvement of approximation and FPT algorithms for CMSR remains open.

In the next section, we will survey the scaffold filling problem, again initiated by David Sankoff. Due to the technical difficulty of handling breakpoints and adjacencies in sequences (which was not completely given in [44]), this time we focus more on the details.

9.4 Approximation for Scaffold Filling with Gene Duplications

With respect to a target singleton genome, possibly with gene repetitions, a *scaffold* is simply an incomplete sequence. It was found that most of the sequenced

genomes are in fact in the form of scaffolds. Muñoz et al. first formulate the problem of filling an incomplete scaffold H into H', using a reference genome G, such that certain genomic distance between H' and G is minimized [54]. More specifically, they showed for multichromosomal genomes, this (one-sided) scaffold filling problem under the DCJ distance is polynomially solvable. David Sankoff visited Montana State University in early 2010 and gave a talk on this topic. We then started to collaborate by showing that for singleton genomes without gene repetitions, under the breakpoint distance, even the two-sided scaffold filling problem (i.e., both G, H are incomplete scaffolds or permutations) is polynomially solvable [40]. Then this result is generalized to multichromosomal genomes under the DCJ distance [44].

When genomes contain some duplicated genes, the scenario is completely different. There are three general criteria (or distance) to measure the similarity of genomes: the exemplar genomic distance [59], the minimum common string partition (MCSP) distance [30] and the maximum number of common string adjacencies [2, 41, 44]. Unfortunately, as covered in Sect. 9.2, unless P = NP, there does not exist any polynomial-time approximation (regardless of the factor) for computing the exemplar genomic distance even when each gene is allowed to repeat three times [21, 24] or even two times [13, 47]. The MCSP problem is NP-complete even if each gene repeats at most two times [36] and the best known approximation factor for the general problem is $O(\log n \log^* n)$ [30]. Based on the maximum number of common string adjacencies, Jiang et al. proved that the one-sided scaffold filling problem is also NP-complete, and designed a 1.33-approximation algorithm with a greedy strategy [41, 44]. As some of the details on handling breakpoints/adjacencies for sequences are missing in [44], we try to present the complete solution here. We comment that handling breakpoints/adjacencies for permutations is much easier.

9.4.1 Preliminaries

At first, we revise some necessary definitions, which are also defined in [44], but not in a perfect way. (Also, note that the breakpoint and adjacency definitions are more general than in Sect. 9.2 which only handle permutations.) We assume that all genes and genomes are unsigned, and it is straightforward to generalize the result to signed genomes. Given a gene set Σ, a string P is called *permutation* if each element in Σ appears exactly once in P. We use $c(P)$ to denote the set of elements in permutation P. A string A is called *sequence* if some genes appear more than once in A, and $c(A)$ denotes genes of A, which is a multi-set of elements in Σ. For example, $\Sigma = \{a, b, c, d\}$, $A = abcdacd$, $c(A) = \{a, a, b, c, c, d, d\}$. A *scaffold* is an incomplete *sequence*, typically obtained by some sequencing and assembling process. A substring with m genes (in a sequence) is called an *m-substring*, and a 2-substring is also called a *pair*, as the genes are unsigned, the relative order of the two genes of a pair does not matter, i.e., the pair xy is equal to the pair yx. Given a scaffold $A = a_1a_2a_3 \ldots a_n$, let $P_A = \{a_1a_2, a_2a_3, \ldots, a_{n-1}a_n\}$ be the set of pairs in A.

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

Fig. 9.1 An example for adjacency, breakpoint and the related definitions

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

It follows from the definition that scaffolds A and B contain the same set of adjacencies but distinct breakpoints. The maximum matched pairs in B (or equally, in A) form the *adjacency set* between A and B, denoted as $a(A, B)$. We use $b_A(A, B)$ and $b_B(A, B)$ to denote the set of breakpoints in A and B, respectively. A gene is called a *bp-gene*, if it appears in a breakpoint. A maximal substring T of A (or B) is call a *bp-string*, if each pair in it is a breakpoint. The leftmost and rightmost genes of a bp-string T are call the *end-genes* of T, the other genes in T are called the *mid-genes* of T. We illustrate the above definitions in Fig. 9.1.

Given two scaffolds $A = a_1a_2 \ldots a_n$ and $B = b_1b_2 \ldots b_m$, as we can see, each gene except the four ending ones is involved in two adjacencies or two breakpoints or one adjacency and one breakpoint. To get rid of this imbalance, we add "#" to both ends of A and B, which fixes a small bug in [41, 44]. From now on, we assume that $A = a_0a_1 \ldots a_na_{n+1}$ and $B = b_0b_1 \ldots b_mb_{m+1}$, where $a_0 = a_{n+1} = b_0 = b_{m+1} = \#$.

For a sequence A and a multi-set of elements X, let $A + X$ be the set of all possible resulting sequences after filling all the elements in X into A. Now, we define the problems we study in this paper formally.

Definition 2 Scaffold Filling to Maximize the Number of (String) Adjacencies (SF-MNSA).

Input: Two scaffolds A and B over a gene set Σ and two multi-sets of elements X and Y, where $X = c(B) - c(A)$ and $Y = c(A) - c(B)$.
Question: Find $A^* \in A + X$ and $B^* \in B + Y$ such that $|a(A^*, B^*)|$ is maximized.

The one-sided SF-MNSA problem is a special instance of the SF-MNSA problem where one of X and Y is empty.

Definition 3 One-sided SF-MNSA.

Input: A complete sequence G and an incomplete scaffold I over a gene set Σ, a multi-set $X = c(G) - c(I) \neq \emptyset$ with $c(I) - c(G) = \emptyset$.
Question: Find $I^* \in I + X$ such that $|a(I^*, G)|$ is maximized.

Note that while the two-sided SF-MNSA problem is more general and more difficult, the One-Sided SF-MNSA problem is more practical as a lot of genome analysis are based on some reference genome [54].

We now list a few basic properties of this problem.

Lemma 1 *Let G and I be the input of an instance of the One-sided SF-MNSA problem, and x be any gene which appears the same times in G and I. If x does not constitute breakpoint in G (resp. I), then it also does not constitute any breakpoint in I (resp. G).*

Proof W.L.O.G, assume that x appears q times in I and G, respectively. Also, assume that there are q_1 adjacencies in the form "xx", and q_2 adjacencies in the form "xy" ($y \neq x$) in G. In G, since each copy of x is involved in two adjacencies: one adjacency on its left and one adjacency on its right, but the two x's share the adjacency "xx", so the total number of adjacencies containing x is $2q - q_1$, then we have $2q - q_1 = q_1 + q_2$, which implies $2q - 2q_1 = q_2$.

In the scaffold I, there must be at least q_1 "xx" adjacencies. As x appears only q times, x has $2q$ neighbors where there are at least $2q_1$ x's. So x has at most $2q - 2q_1$ neighbors which are not x, which means that there are at most $2q - 2q_1$ ($=q_2$) pairs in the form "xy" ($y \neq x$) in I. Since there are q_2 "xy" ($y \neq x$) adjacencies in G, there must be q_2 "xy" ($y \neq x$) adjacencies in I. Therefore, there are exactly q_1 adjacencies in the form "xx", and all the q_2 pairs in the form "xy" ($y \neq x$) are adjacencies in I, and none of them is a breakpoint. □

Lemma 2 *Let G and I be the input of an instance of the One-sided SF-MNSA problem, let $bp(I)$ and $bp(G)$ be the multi-set of bp-genes in I and G, respectively. Then any gene in $bp(G)$ appears in $bp(I) \cup X$, and $bp(I) \subseteq bp(G)$.*

Proof Assume to the contrary that there exists a gene x, $x \in bp(G)$, but $x \notin bp(I) \cup X$. Since $x \notin X$, x appears the same number of times in G and I; moreover, $x \notin bp(I)$, then all the pairs in I containing x are adjacencies. From Lemma 1, all the pairs involving x in G are adjacencies, contradicting the assumption that $x \in bp(G)$. So any gene in $bp(G)$ appears in $bp(I) \cup X$. By a similar argument, we can prove $bp(I) \subseteq bp(G)$. □

Each breakpoint contains two genes, from what we discussed in Lemma 2, every breakpoint in the complete sequence G belongs to one of the three multi-sets according to the affiliation of its two bp-genes.

$$\begin{aligned}
\textit{scaffold}\ \ G &= \langle \#1\ 2\ 3\ 4\ a\ x\ y\ b\ \# \rangle \\
\textit{scaffold}\ \ I &= \langle \#1\ 3\ 2\ 4\ a\ b\ \# \rangle \\
X &= \{x, y\} \\
a(I, G) &= \{\#1, 23, 4a, b\#\} \\
b_G(I, G) &= \{12, 34, ax, xy, yb\} \\
BP_1(G) &= \{ax, yb\} \\
BP_2(G) &= \{xy\} \\
BP_3(G) &= \{12, 34\} \\
b_I(I, G) &= \{13, 24, ab\}
\end{aligned}$$

Fig. 9.2 Classification of the breakpoints

$BP_1(G)$: breakpoints with one bp-gene in X and the other bp-gene not in X.
$BP_2(G)$: breakpoints with both of the bp-genes in X.
$BP_3(G)$: breakpoints with both of the bp-genes not in X.

An example is shown in Fig. 9.2.

9.4.2 Approximation Algorithm for One-Sided SF-MNSA

In this subsection, we present a 1.33-Approximation algorithm for the one-sided SF-MNSA problem. The goal of solving this problem is, while inserting the genes of X into the scaffold I, to obtain as many adjacencies as possible. No matter in what order the genes are inserted, they appears in groups in the final $I' \in I + X$, so we can consider that I' is obtained by inserting strings (composed of genes of X) into I.

Obviously, inserting a string of length one (i.e., a single gene) will generate at most two adjacencies, and inserting a string of length m will generate at most $m + 1$ adjacencies. Therefore, we will have two types of inserted strings.

1. Type-1: a string of k missing genes $x_1, x_2, \ldots, x_k$ are inserted in between $y_i y_{i+1}$ in the scaffold I to obtain $k+1$ adjacencies (i.e., $y_i x_1, x_1 x_2, \ldots, x_{k-1} x_k, x_k y_{i+1}$), where $y_i y_{i+1}$ is a breakpoint.

   In this case, $x_1 x_2 \ldots x_k$ is called a k-Type-1 string, $y_i y_{i+1}$ is called a *dock*, and we also say that $y_i y_{i+1}$ *docks* the corresponding k-Type-1 string $x_1 x_2 \ldots x_k$.
2. Type-2: a sequence of l missing genes $z_1, z_2, \ldots, z_l$ are inserted in between $y_j y_{j+1}$ in the scaffold I to obtain l adjacencies (i.e., $y_j z_1$ or $z_l y_{j+1}$, $z_1 z_2$, $\ldots$, $z_{l-1} z_l$), where $y_j y_{j+1}$ is a breakpoint; or a sequence of l missing genes $z_1, z_2, \ldots, z_l$ are inserted in between $y_j y_{j+1}$ in the scaffold I to obtain $l + 1$ adjacencies (i.e., $y_j z_1$, $z_1 z_2$, $\ldots$, $z_{l-1} z_l$, $z_l y_{j+1}$), where $y_j y_{j+1}$ is an adjacency.

This is the basic observation for devising our algorithm. Most of our work is devoted to searching the Type-1 strings.

Searching the 1-Type-1 Strings To identify the 1-Type-1 strings, we use a greedy method. For each gene x_i of X and each breakpoint $y_j y_{j+1}$ of $b_I(I, G)$, if we can obtain two adjacencies by inserting x_i in between $y_j y_{j+1}$, then insert x_i to $y_j y_{j+1}$.

Algorithm 1: *Greedy*1(G, I)
1 Insert x_i in between $y_j y_{j+1}$ whenever two new adjacencies are generated, where $x_i \in X$ and $y_j y_{j+1} \in b_I(I, G)$.

Searching the 2-Type-1 Strings To identify the 2-Type-1 strings, we again use a greedy method. For each pair of missing genes $x_i x_k$ if we can obtain three adjacencies by inserting $x_i x_k$ in between $y_j y_{j+1}$, where $y_j y_{j+1} \in b_I(I, G)$, then insert $x_i x_k$ in between $y_j y_{j+1}$.

Algorithm 2: *Greedy*2(G, I)
1 Insert $x_i x_k$ in between $y_j y_{j+1}$ whenever three new adjacencies are generated, where $x_i, x_k \in X$ and $y_j y_{j+1} \in b_I(I, G)$.

Inserting the Remaining Genes In this subsection, we present a polynomial-time algorithm guaranteeing that the number of adjacencies increases by the same number of the genes inserted. A general idea of this algorithm was mentioned in [44], with many details missing, and we will present the details here.

Given the complete sequence G and the scaffold I, as we discussed in Sect. 9.4.1, the breakpoints in G can be divided into three sets: $BP_1(G)$, $BP_2(G)$, and $BP_3(G)$. In any case, the breakpoints in $BP_3(G)$ cannot be converted into adjacencies; so we try to convert the breakpoints in $BP_1(G)$ and $BP_2(G)$ into adjacencies.

Lemma 3 *If $BP_1(G) \neq \emptyset$, then there exists a breakpoint in I where after some gene of X is inserted, the number of adjacencies increases by one.*

Proof Let $t_i t_{i+1}$ be a breakpoint in G, satisfying that $t_i t_{i+1} \in BP_1(G)$, $t_i \in X$, and, from Lemma 2, $t_{i+1} \in bp(I)$. Then, there exists a breakpoint $t_{i+1} s_j$ or $s_k t_{i+1}$ in I. Hence, if we insert t_i in between that breakpoint, we will obtain a new adjacency $t_i t_{i+1}$ without affecting any other adjacency. □

Thus, it is trivial to obtain one more adjacency whenever $BP_1(G) \neq \emptyset$.

Lemma 4 *For any $x \in X \cap c(I)$, if there is an "xx" breakpoint in G then after inserting x in between some "xy" pair in I, the number of adjacencies increases by one.*

Proof If "xy" is a breakpoint, then after inserting an 'x' in between it, we obtain a new adjacency "xx". If "xy" is an adjacency, then after inserting an 'x' in between it, we have "xxy". The adjacency "xy" still exists, and we obtain a new adjacency "xx". □

Lemma 5 *If there is a breakpoint "xy" in $BP_2(G)$ and a breakpoint "xz" (resp. "yz") in I, then after inserting y (resp. x) in between "xz" (resp. "yz") in I, the number of adjacencies increases by one.*

Proof From the definition of $BP_2(G)$, we know that $x, y \in X$. Since "xy" is a breakpoint in G and "xz" is a breakpoint in I, we obtain a new adjacency "xy" by inserting y in between "xz", without affecting any other adjacency. A similar argument for inserting x in between "yz" also holds. □

Next, we show that the following case is polynomially solvable. This case satisfies the following conditions.

1. $BP_1(G) = \emptyset$;
2. It does not contain a breakpoint like "xx" in G unless $x \notin X \cap c(I)$;
3. For any breakpoint of the form "xy" in $BP_2(G)$, all the pairs in I involving x or y are adjacencies.

Let $BS_2(G)$ be the set of bp-strings in G with all breakpoints belonging to $BP_2(G)$.

Lemma 6 *In the case satisfying* (1), (2) *and* (3), *the number of times a gene appears as an end-gene of some bp-string of $BS_2(G)$ is even.*

Proof Let gene $x = t_i$ be an end-gene of some bp-string $t_i t_{i+1} \ldots t_j$ of $BS_2(G)$. Since $BP_1(G) = \emptyset$ and x will not be involved in any breakpoint of $BP_3(G)$, $t_{i-1}t_i$ must be an adjacency. Assume that x appears q times in G and q' ($<q$) times in I. As there is no breakpoint in the form "xx" in G and I, we could assume that there are q_1 adjacencies in the form "xx" in G and I. Then, the total number of pairs (adjacencies and breakpoints) involving x in G is $2(q - q_1)$, and of which, $2(q' - q_1)$ are adjacencies. So the number of breakpoints involving x in G is $2(q - q')$, which is even. An end-gene only constitutes one breakpoint and other mid-genes each constitutes two breakpoints. Therefore, any gene should appear at the end of some bp-string of $BS_2(G)$ for an even number of times. □

From Lemma 6, if we denote each bp-string of $BS_2(G)$ by a vertex, and there is an edge between two vertices iff their corresponding bp-strings have a common end-gene, the resulting graph contains a cycle of distinct vertices. Traveling this cycle, concatenating the bp-strings corresponding to the vertices, and deleting one copy of the common end-gene, eventually we can obtain a string composed of genes of X. The following lemma and corollary shows that this string can be inserted into I entirely, generating no breakpoint at all.

Lemma 7 *In the case satisfying* (1), (2) *and* (3), *for a gene x, let q_1 be the number that it appears as an end-gene, let q_2 be the number that it appears in some bp-string of $BS_2(G)$ as a mid-gene, and let r be the number that it appears in X. Then, we have $r = q_1/2 + q_2$.*

Proof Assume that x appears q times in G, q' ($< q$) times in I, and there are p adjacencies in the form "xx" in G and I. Then, the total number of pairs (adjacencies and breakpoints) involving x in G is $2(q - p)$, and of which, $2(q' - p)$ are adjacencies. So the number of breakpoints involving x in G is $2(q - q')$. Each x of q_1 end-genes contributes to one breakpoint, and each x of q_2 mid-genes contributes to two breakpoints, thus, $2(q - q') = q_1 + 2q_2$. Note that $(q - q')$ is exactly r; and following Lemma 6, q_1 is even. Then, $r = q_1/2 + q_2$. □

We summarize the above ideas as the following algorithm, which ensures us to obtain as many adjacencies as the number of missing genes inserted.

For two strings s_1 and s_2, if the right end-gene $r(s_1)$ of s_1 is the same as the left end-gene $\ell(s_2)$ of s_2, we use $s_1 \bowtie s_2$ to represent the string obtained by first concatenating s_1 with s_2 and then delete one copy of $r(s_1)$ and $\ell(s_2)$. For example, $s_1 = acbd$, $s_2 = decb$, then $s_1 \bowtie s_2 = abcdecb$.

Theorem 7 *The algorithm Insert-Whole-Strings*(•) *guarantees that the number of adjacencies increased is not smaller than the number of genes inserted.*

Proof At step 2, 3, 4 of the algorithm, one gene is inserted into I and each time one more adjacency is obtained. At each round of step 6, a string of length l is inserted in between an adjacency in I, then we obtain $l+1$ new adjacencies with one destroyed. So the number of adjacencies increased is not smaller than the number of genes inserted. □

Algorithm 3: *Insert-Whole-Strings*(G, I)
1 Identify the adjacencies and breakpoints in G and I.
2 If $BP_1(G) \neq \emptyset$,
  Insert a gene of X into I according to Lemma 3.
3 If there is an "xx" breakpoint in G, $x \in X$
  Insert x into I according to Lemma 4.
4 If there is an "xy" breakpoint in G and an "xz" breakpoint in I, $x, y \in X$,
  Insert y into Y according to Lemma 5.
5 Compute the set of bp-strings $BS_2(G) = \{s_1, \ldots, s_p$, where $s_j = x_{j,1} \ldots x_{j,u_j}$ and all $x_{j,k} \in X\}$.
6 WHILE ($BS_2(G) \neq \emptyset$)
  \\ Compute a string L composed of some bp-strings of $BS_2(G)$ whose two end-genes are the same.
  {
  (6.1) Choose any bp-string of $BS_2(G)$, say s_j. Let $L = s_j = x_{j,1} \ldots x_{j,u_j}$.
  (6.2) WHILE ($\ell(L) \neq r(L)$)
    Find a bp-string $s_i = x_{i,1} \ldots x_{i,u_i}$ (or its reversal $\overline{s_i} = x_{i,u_i} \ldots x_{i,1}$) of $BS_2(G)$, such that $r(L) = \ell(s_i)$ or $r(L) = \ell(\overline{s_i})$.
    Update $L \leftarrow L \bowtie s_i$ or $L \leftarrow L \bowtie \overline{s_i}$.
  (6.3) Replace some gene identical to $\ell(L)$ in I by the string L.
  (6.4) Update the set $BS_2(G)$.
  }
7 Return the resulting I.

We run the above algorithm on the following example.

$$G = \#daebxceafceb1234\#, \qquad I = \#dafcxb1324\#, \qquad X = \{a, b, c, e, e, e\},$$

$$BP_1(G) = \emptyset, \qquad BP_2(G) = \{ae, eb, ce, ea, ce, eb\}, \qquad BP_3(G) = \{12, 34\},$$

then the set of breakpoint strings $BS_2(G) = \{aeb, cea, ceb\}$. According to the algorithm, we have $L = aebecea$. Gene a in I is replaced with string L to obtain sequence $I^* = \#daebeceafcxb1324\#$. The number of adjacencies is added to by 6 and no new breakpoint is generated.

9.4.3 Analysis of the Approximation Algorithm

In this subsection, we will prove that the approximation factor of our algorithm is $4/3$. Firstly, we present a lower bound of the optimal solution.

A Lower Bound Given an instance of One-sided SF-MNSA, let $I^* \in I + X$ be the final scaffold in the optimal solution after inserting all genes of X into I. Compared to I, all genes belonging to X appear as substrings in I^*. Let $x_1x_2 \ldots x_l$ be a string inserted in between $y_i y_{i+1}$ in I^*, then either $y_i x_1$ or $x_l y_{i+1}$ or both are adjacencies. Since otherwise, we could delete this string from I^* (number of adjacencies decreases by at most $l - 1$), re-insert it following the algorithm *Insert-Whole-Strings*($\bullet$) (number of adjacencies increases by at least l), and obtain one more adjacency. Thus, we have the following corollary of Theorem 7,

Corollary 1 *Each substring in I^* composed of genes of X is either Type-1 or Type-2.*

Now, we present a lower bound for the optimal number of adjacencies.

Lemma 8 *Let OPT be the number of adjacencies between G and I^*, k_0 be the number of adjacencies between G and I, and $k_1 = |X|$. Let b_i be the number of i-Type-1 substrings and q be the maximum length of Type-1 substrings in the optimal solution between G and I^*. Then*

$$OPT - k_0 = k_1 + b_1 + b_2 + \cdots + b_q \leq \frac{4}{3}\left(k_1 + \frac{1}{2}b_1 + \frac{1}{4}b_2\right) \tag{9.1}$$

Proof Define C as the total number of genes in Type-2 substrings in I^*. Since inserting an l-Type-1 string will generate $l + 1$ more adjacencies, and inserting a l-Type-2 string will generate l more adjacencies, we have,

$$OPT = k_0 + \sum_{i=1}^{q}(i + 1) \times b_i + C.$$

By the definition of Type-1 and Type-2 substrings, we have

$$k_1 = \sum_{i=1}^{q} (i \times b_i) + C \geq b_1 + 2b_2 + 3(b_3 + b_4 + \cdots + b_q) + C.$$

Thus,

$$\sum_{i=3}^{q} b_i \leq (k_1 - C - b_1 - 2b_2)/3.$$

Hence, we have

$$\begin{aligned} OPT - k_0 &= C + \sum_{i=1}^{q} i \times b_i + b_1 + b_2 + \cdots + b_q \\ &= k_1 + b_1 + b_2 + \cdots + b_q \\ &\leq k_1 + b_1 + b_2 + (k_1 - C - b_1 - 2b_2)/3 \\ &\leq \frac{4}{3}\left(k_1 + \frac{1}{2}b_1 + \frac{1}{4}b_2\right). \end{aligned}$$

□

Lemma 8 shows that if the number of Type-1 substrings computed in the approximation algorithm is not smaller than $(2b_1 + b_2)/4$, then the approximation factor is $4/3$.

Description of the Main Algorithm There are three main steps in our algorithm. Firstly, we try to search the 1-Type-1 strings; secondly, we try to search the 2-Type-1 strings; finally, we insert the remaining genes in X, guaranteeing that on average we will obtain at least one adjacency for each inserted missing gene.

Main Algorithm	
Input: Complete sequence G and incomplete scaffold I, $X = c(G) - c(I)$.	
Output: $I' \in I + X$	
1	Call *Greedy*1(G, I), let the resulting incomplete scaffold be I_1.
2	Call *Greedy*2(G, I_1), let the resulting incomplete scaffold be I_2.
3	Call *Insert-Whole-Strings*(G, I_2). Let the resulting complete scaffold be I'.
4	Return I'.

9.4.4 Proof of the Approximation Factor

In our algorithm, we make effort to insert Type-1 substrings as much as possible. But a Type-1 substring (say I_s) inserted by our algorithm may make other Type-1 substrings in some optimal solution infeasible, we say I_s *destroys* them. The following lemma shows the number of Type-1 substrings that could be destroyed by a given Type-1 substring.

Lemma 9 *A i-Type-1 substring can destroy at most $i+1$ Type-1 substrings in some optimal solution.*

Proof Assume that an i-Type-1 substring I_s is inserted in between some breakpoint $y_j y_{j+1}$ in I. Then each of the genes in I_s, if not use by I_s, could form a distinct Type-1 substring in some optimal solution. Also, there may exist another Type-1 substring that could be inserted in between the breakpoint $y_j y_{j+1}$ in the optimal solution. Totally, at most $i+1$ Type-1 substrings in the optimal solution could be destroyed by I_s. □

We have the following lemma regarding this greedy algorithm.

Lemma 10 *Let b'_1, b'_2 be the number of 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, by Lemma 9, each of the k'_1 inserted missing genes can destroy at most two type-1 1-substrings in some optimal solution. Moreover, 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 $G = \ldots\alpha a\beta\ldots\gamma ab\delta\ldots\alpha uv\beta\ldots$ and let $I = \alpha\ldots\alpha\beta\ldots\gamma\delta\ldots\beta\ldots a\ldots$. 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).

Again, by Lemma 9, at Step 2, each of the inserted 2-type-1 substrings can destroy at most three 2-type-1 substrings in some optimal solution.

Now, putting all together,

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

and

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

Then

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

□

Theorem 8 *There is a greedy algorithm which approximates One-sided SF-MNSA with a factor of* 1.33.

Proof Following the greedy algorithm, Theorem 7, Lemma 8, and Lemma 10, we have the approximation solution value *APP*, which satisfies the following inequalities:

$$APP - k_0 = k_1 + b'_1 + b'_2 \geq k_1 + \frac{1}{2}b_1 + \frac{1}{4}b_2 \geq \frac{3}{4}(OPT - k_0).$$

So, we have $APP \geq \frac{3}{4}OPT + \frac{1}{4}k_0 \geq \frac{3}{4}OPT$. Hence $\frac{OPT}{APP} \leq 1.33$, and the theorem is proven. □

In [51], a better factor-1.25 approximation was proposed. While the overall framework is similar, the details are quite different. The new approximation is achieved by a combination of maximum matching, local improvement and greedy search.

9.5 Concluding Remarks and Open Problems

The negative results on EBD and ENbS do not mean that we have absolutely no way to tackle these problems. For instance, in [1], with integer linear programming, very nice empirical results are obtained. Here, we try to present a different way to handle these problems formally.

In many biological problems, the optimal solution value OPT could be zero. (Besides EBD, in some minimum recombination haplotype reconstruction problems the optimal solution value could be zero.) As implied by Theorem 1, if computing such an optimal solution with zero solution value is NP-complete then the problem does not admit *any* polynomial-time approximation (unless P = NP). However, in reality one would be satisfied to obtain a solution with value one or two. Due to this reason, we can relax the traditional definition of approximation to a *weak approximation*. Given a minimization problem Π, let the optimal solution of Π be OPT. We say that a weak approximation algorithm $\mathcal{W}$ provides a *performance guarantee* of α

for Π if for every instance I of Π, the solution value returned by $\mathcal{W}$ is at most $\alpha \times (OPT + 1)$.

In [21, 22, 24] we showed that EBD and the exemplar conserved interval distance problems are both hard to approximate even under the weak approximation model. But for the exemplar reversal distance problem, no such result is known yet.

For the exemplar common interval number problem [12], the only negative result is its NP-hardness. It would also be interesting to know whether it admits an efficient polynomial-time approximation. We conclude this paper with a list of open problems.

1. For the One-sided Exemplar Breakpoint Distance problem, does there exist a factor-$o(n)$ approximation? The only known negative result is the APX-hardness of the problem.
2. For the exemplar common interval number problem, does there exist a good approximation?
3. For the CMSR problem, does there exist faster FPT algorithm and/or a smaller linear kernel?
4. For the One-side SF-MNSA problem, does there exist an FPT algorithm?

Acknowledgements I would like to thank my collaborators for this series of research: Zhixiang Chen, Richard Fowler, Bin Fu, Haitao Jiang, Minghui Jiang, Zhong Li, Guohui Lin, Nan Liu, David Sankoff, Weitian Tong, Lusheng Wang, Boting Yang, Zhiyu Zhao, Chunfang Zheng and Daming Zhu.

References

1. Angibaud, S., Fertin, G., Rusu, I., Thévenin, A., Vialette, S.: Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. J. Comput. Biol. **15**, 1093–1115 (2008)
2. Angibaud, S., Fertin, G., Rusu, I., Thevenin, A., Vialette, S.: On the approximability of comparing genomes with duplicates. J. Graph Algorithms Appl. **13**(1), 19–53 (2009)
3. Bader, D., Moret, B., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comput. Biol. **8**(5), 483–491 (2001)
4. Bafna, V., Pevzner, P.: Sorting by reversals: genome rearrangements in plant organelles and evolutionary history of X chromosome. Mol. Biol. Evol. **12**, 239–246 (1995)
5. Bar-Yehuda, R., Halldórsson, M.M., Naor, J.(S.), Shachnai, H., Shapira, I.: Scheduling split intervals. SIAM J. Comput. **36**, 1–15 (2006)
6. Bereg, S., Jiang, M., Wang, W., Yang, B., Zhu, B.: Simplifying 3D polygonal chains under the discrete Fréchet distance. In: Proc. 8th Latin American Theoretical Informatics Symposium (LATIN'08), April 7–11, 2008. LNCS, vol. 4957, pp. 630–641 (2008)
7. Bergeron, A., Mixtacki, J., Stoye, J.: On sorting by translocations. J. Comput. Biol. **13**(2), 567–578 (2006)
8. Bergeron, A., Stoye, J.: On the similarity of sets of permutations and its applications to genome comparison. In: Proc. 9th Intl. Ann. Comput. and Combinatorics (COCOON'03). LNCS, vol. 2697, pp. 68–79 (2003)
9. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-approximation algorithm for sorting by reversals. In: Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02), pp. 200–210 (2002)

10. Bertrand, D., Blanchette, M., El-Mabrouk, N.: Genetic map refinement using a comparative genomic approach. J. Comput. Biol. **16**(10), 1475–1486 (2009)
11. Blin, G., Rizzi, R.: Conserved interval distance computation between non-trivial genomes. In: Proc. 11th Intl. Ann. Comput. and Combinatorics (COCOON'05). LNCS, vol. 3595, pp. 22–31 (2005)
12. Blin, G., Chauve, C., Fertin, G., Rizzi, R., Vialette, S.: Comparing genomes with duplicates: a computational complexity point of view. IEEE/ACM Trans. Comput. Biol. Bioinform. **4**, 523–534 (2007)
13. Blin, G., Fertin, G., Sikora, F., Vialette, S.: The exemplar breakpoint distance for non-trivial genomes cannot be approximated. In: Proc. 3nd Workshop on Algorithm and Computation (WALCOM'09). LNCS, vol. 5431, pp. 357–368 (2009)
14. Bryant, D.: The complexity of calculating exemplar distances. In: Sankoff, D., Nadeau, J. (eds.) Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families, pp. 207–212. Kluwer Academic, Dordrecht (2000)
15. Bulteau, L., Fertin, G., Jiang, M., Rusu, I.: Tractability and approximability of maximal strip recovery. Theor. Comput. Sci. **440–441**, 14–28 (2012)
16. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Discrete Math. **26**(3), 1148–1180 (2012)
17. Caprara, A.: Sorting permutations by reversals and Eulerian cycle decompositions. SIAM J. Discrete Math. **12**, 91–110 (1999)
18. Chen, J., Huang, X., Kanj, I., Xia, G.: Linear FPT reductions and computational lower bounds. In: Proceedings of the 36th ACM Symposium on Theory of Computing (STOC'04), pp. 212–221 (2004)
19. Chen, X.: On sorting permutations by double-cut-and-joins. In: Proc. of the 16th International Conf. on Computing and Combinatorics (COCOON'10), pp. 439–448 (2010)
20. Chen, X., Sun, R., Yu, J.: Approximating the double-cut-and-join distance between unsigned genomes. BMC Bioinform. **12**(Suppl. 9), S17 (2011)
21. Chen, Z., Fu, B., Zhu, B.: The approximability of the exemplar breakpoint distance problem. In: Proc. 2nd Intl. Conf. on Algorithmic Aspects in Information and Management (AAIM'06). LNCS, vol. 4041, pp. 291–302 (2006)
22. Chen, Z., Fu, B., Fowler, R., Zhu, B.: Lower bounds on the approximation of the exemplar conserved interval distance problem of genomes. In: Proc. 12th Intl. Ann. Comput. and Combinatorics (COCOON'06). LNCS, vol. 4112, pp. 245–254 (2006)
23. Chen, Z., Fu, B., Yang, B., Xu, J., Zhao, Z., Zhu, B.: Non-breaking similarity of genomes with gene repetitions. In: Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM'07). LNCS, vol. 4580, pp. 119–130 (2007)
24. Chen, Z., Fu, B., Fowler, R., Zhu, B.: On the inapproximability of the exemplar conserved interval distance problem of genomes. J. Comb. Optim. **15**(2), 201–221 (2008)
25. Chen, Z., Fu, B., Jiang, M., Zhu, B.: On recovering syntenic blocks from comparative maps. J. Comb. Optim. **18**, 307–318 (2009)
26. Chen, Z., Fu, B., Goebel, R., Lin, G., Tong, W., Xu, J., Yang, B., Zhao, Z., Zhu, B.: On the approximability of the exemplar non-breakpoint similarity problem of genomes with gene repetitions. Theor. Comput. Sci. (2013, to appear)
27. Choi, V., Zheng, C., Zhu, Q., Sankoff, D.: Algorithms for the extraction of synteny blocks from comparative maps. In: Proc. of the 7th International Workshop on Algorithms in Bioinformatics (WABI'07), pp. 277–288 (2007)
28. Christie, D.: A 3/2-approximation algorithm for sorting by reversals. In: Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'98), pp. 244–252 (1998)
29. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
30. Cormode, G., Muthukrishnan, S.: The string edit distance matching problem with moves. In: Proc. 13th ACM-SIAM Symp. on Discrete Algorithms (SODA'02), pp. 667–676 (2002)

31. Cui, Y., Wang, L., Zhu, D., Liu, X.: A $(1.5+\epsilon)$-approximation algorithm for unsigned translocation distance. IEEE/ACM Trans. Comput. Biol. Bioinform. **5**(1), 56–66 (2008)
32. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Berlin (1999)
33. Elias, I., Hartman, T.: A 1.375-approximation algorithm for sorting by transpositions. IEEE/ACM Trans. Comput. Biol. Bioinform. **3**, 369–379 (2006)
34. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
35. Gascuel, O. (ed.): Mathematics of Evolution and Phylogeny. Oxford University Press, Oxford (2004)
36. Goldstein, A., Kolman, P., Zheng, J.: Minimum common string partitioning problem: hardness and approximations. In: Proc.15th Intl. Symposium on Algorithms and Computation (ISAAC'04). LNCS, vol. 3341, pp. 473–484 (2011). Also in: Electron. J. Comb. **12**, paper R50 (2005)
37. Hannenhalli, S.: Polynomial-time algorithm for computing translocation distance between genomes. Discrete Appl. Math. **71**(1–3), 137–151 (1996)
38. Hannenhalli, S., Pevzner, P.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. ACM **46**(1), 1–27 (1999)
39. Hästad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Math. **182**, 105–142 (1999)
40. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint distance. In: Proc. of the 2010 International RECOMB-CG Workshop (RECOMB-CG'10). LNBI, vol. 6398, pp. 83–92 (2010)
41. Jiang, H., Zhong, F., Zhu, B.: Filling scaffolds with gene repetitions: maximizing the number of adjacencies. In: Proc. 22nd Annual Symposium on Combinatorial Pattern Matching (CPM'11). LNCS, vol. 6661, pp. 55–64 (2011)
42. Jiang, H., Zhu, B., Zhu, D.: Algorithms for sorting unsigned linear genomes by the DCJ operations. Bioinformatics **27**(3), 311–316 (2011)
43. Jiang, H., Li, Z., Lin, G., Wang, L., Zhu, B.: Exact and approximation algorithms for the complementary maximal strip recovery problem. J. Comb. Optim. **23**(4), 493–506 (2012)
44. Jiang, H., Zheng, C., Sankoff, D., Zhu, B.: Scaffold filling under the breakpoint and related distances. IEEE/ACM Trans. Bioinform. Comput. Biol. **9**(4), 1220–1229 (2012)
45. Jiang, H., Zhu, B.: A linear kernel for the complementary maximal strip recovery problem. In: Proc. 23rd Annual Combinatorial Pattern Matching Symposium (CPM'12). LNCS, vol. 7354, pp. 349–359 (2012)
46. Jiang, H., Wang, L., Zhu, B., Zhu, D.: A $(1.408+\epsilon)$-approximation algorithm for sorting unsigned genomes by reciprocal translocations. In: RECOMB'13, poster (2013)
47. Jiang, M.: The zero exemplar distance problem. In: Proc. of the 2010 International RECOMB-CG Workshop (RECOMB-CG'10). LNBI, vol. 6398, pp. 74–82 (2010)
48. Kaplan, H., Shamir, R., Tarjan, R.: A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29**, 880–892 (1999)
49. Li, G., Qin, X., Wang, X., Zhu, B.: A linear-time algorithm for computing translocation distance between signed genomes. In: Proc. of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM'04), pp. 323–332 (2004)
50. Lin, G., Goebel, R., Li, Z., Wang, L.: An improved approximation algorithm for the complementary maximal strip recovery problem. J. Comput. Syst. Sci. **78**(3), 720–730 (2012)
51. Liu, N., Jiang, H., Zhu, D., Zhu, B.: An improved approximation algorithm for scaffold filling to maximize the common adjacencies. In: Proc. of the 19th Intl. Conf. on Computing and Combinatorics (COCOON'13). LNCS, vol. 7936, pp. 397–408 (2013)
52. Makaroff, C., Palmer, J.: Mitochondrial DNA rearrangements and transcriptional alternatives in the male sterile cytoplasm of Ogura radish. Mol. Cell. Biol. **8**, 1474–1480 (1988)
53. Marron, M., Swenson, K., Moret, B.: Genomic distances under deletions and insertions. Theor. Comput. Sci. **325**(3), 347–360 (2004)
54. Muñoz, A., Zheng, C., Zhu, Q., Albert, V., Rounsley, S., Sankoff, D.: Scaffold filling, contig fusion and gene order comparison. BMC Bioinform. **11**, 304 (2010)

55. Nguyen, C.T., Tay, Y.C., Zhang, L.: Divide-and-conquer approach for the exemplar breakpoint distance. Bioinformatics **21**(10), 2171–2176 (2005)
56. Ozery-Flato, M., Shamir, R.: An $O(n^{\frac{3}{2}}\sqrt{\log n})$ algorithm for sorting by reciprocal translocations. J. Discrete Algorithms **9**(4), 344–357 (2011)
57. Palmer, J., Herbon, L.: Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. J. Mol. Evol. **27**, 87–97 (1988)
58. Peng, C., Zhou, J., Zhu, B., Zhu, H.: The program download problem: complexity and algorithms. In: Proc. of the 19th Intl. Conf. on Computing and Combinatorics (COCOON'13). LNCS, vol. 7936, pp. 688–695 (2013)
59. Sankoff, D.: Genome rearrangement with gene families. Bioinformatics **16**(11), 909–917 (1999)
60. Schaefer, T.: The complexity of satisfiability problem. In: Proceedings of the 10th ACM Symposium on Theory of Computing (STOC'78), pp. 216–226 (1978)
61. Sturtevant, A.: A crossover reducer in Drosophila melanogaster due to inversion of a section of the third chromosome. Biol. Zent.bl. **46**, 697–702 (1926)
62. Sturtevant, A., Dobzhansky, T.: Inversions in the third chromosome of wild races of drosophila pseudoobscura, and their use in the study of the history of the species. Proc. Natl. Acad. Sci. USA **22**, 448–450 (1936)
63. Swenson, K., Rajan, V., Lin, Y., Moret, B.: Sorting signed permutations by inversions in $O(n \log n)$ time. J. Comput. Biol. **17**(3), 489–501 (2010)
64. Tannier, E., Sagot, M.-F.: Sorting by reversals in subquadratic time. In: Proc. of 15th Symp. Combinatorial Pattern Matching (CPM'04), pp. 1–13 (2004)
65. Wang, L., Zhu, B.: On the tractability of maximal strip recovery. J. Comput. Biol. **17**(7), 907–914 (2010). (Correction, **18**(1) (Jan. 2011))
66. Watterson, G., Ewens, W., Hall, T., Morgan, A.: The chromosome inversion problem. J. Theor. Biol. **99**, 1–7 (1982)
67. Wylie, T., Zhu, B.: Protein chain pair simplification under the discrete Frechet distance. IEEE/ACM Trans. Comput. Biol. Bioinform. 2013). doi:167B699B-E22D-471A-8EE7-01F51E8230D4. Special Issue of ISBRA'12
68. Yap, I., Schneider, D., Kleinberg, J., et al.: A graph-theoretic approach to comparing and integrating genetic, physical and sequence-based maps. Genetics **165**, 2235–2247 (2003)
69. Yancopoulos, S., Attie, O., Friedberg, R.: Efficient sorting of genomic permutations by translocation, inversion and block interchange. Bioinformatics **21**, 3340–3346 (2005)
70. Zheng, C., Zhu, Q., Sankoff, D.: Removing noise and ambiguities from comparative maps in rearrangement analysis. IEEE/ACM Trans. Comput. Biol. Bioinform. **4**, 515–522 (2007)
71. Zhu, D., Wang, L.: On the complexity of unsigned translocation distance. Theor. Comput. Sci. **352**(1–3), 322–328 (2006)