Models and Algorithms for Comparative Genomics

THÈSE N^{O} 6706 (2015)

PRÉSENTÉE LE 17 JUILLET 2015

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE BIOLOGIE COMPUTATIONNELLE ET BIOINFORMATIQUE
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Mingfu SHAO

acceptée sur proposition du jury:

Prof. N. Vishnoi, président du jury
Prof. B. Moret, directeur de thèse
Prof. R. Shamir, rapporteur
Prof. D. Sankoff, rapporteur
Dr Ph. Bucher, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2015

To my parents and my sister

献给我的父母和我的姐姐

Acknowledgements

First of all, I would like to express my deepest gratitude to Prof. Bernard Moret, my advisor, for his continuous guidance and patient support over the course of my Ph.D. study. I have been benefiting a lot from his great insight and taste in research, immense depth and width in techniques, and incredible proficiency in academic skills. I am very grateful to the trust and freedom he has always granted me, which has given me the chance to explore and to develop independence. Beyond that, Bernard is a close friend to his students. I very much appreciate the immediate and tremendous help he has offered in my difficult moments, and his sharing and encouragement for my progress. His attitude of always being kind to students has been carved in my mind.

I gratefully acknowledge Prof. Nisheeth Vishnoi, Dr. Philipp Bucher, Prof. David Sankoff, and Prof. Ron Shamir, for their willingness to be in my thesis committee and their precious time. I appreciate very much that Prof. Nisheeth Vishnoi has agreed presiding the jury. I would like to give my sincere appreciation to Dr. Philipp Bucher with whom I have had many inspiring discussions at various occasions. I am very grateful to Prof. David Sankoff for his enlightening and constructive suggestions and comments during our encounters at conferences. I would like to express my sincere gratitude to Prof. Ron Shamir, for his valuable suggestions for my future research, and enthusiastic support in my applications for postdocs and fellowships.

I would like to thank very much Prof. Dongbo Bu, my master advisor, for this encouragement and valuable opinions for my academic path. I appreciate very much Prof. Wei-Mou Zheng, from whom I have gained many insightful thoughts through our discussions. I am also very grateful to Prof. Louxin Zhang, who provides us softwares for comparison, as well as offers me opportunities to review conference papers.

I count myself very fortunate to have so many splendid colleagues in the LCBB family. My special gratitude goes to Yu Lin, who has introduced me to this research area and provided me guidance. We have kept a pleasant and fruitful collaboration since then. I am grateful to Xiuwei Zhang, Yann Christinat, and Vaibhav Rajan, for their helpful advices in the beginning of my Ph.D. study. I sincerely give my warmest appreciation to Nishanth Nair, Cristina Ghiurcuta, and Min Ye for their companion, with whom I have experienced a fantastic and unforgettable four years' time in LCBB. I would like to thank two previous lab members, Jijun Tang and Krister Swenson, for bringing me insightful discussions as well as many interesting topics during their visits. I sincerely appreciate Sylvie Fiaux, our secretary, for all the help and support throughout my stay here.

I am also very fortunate to have many talented, open, interesting, and warmhearted friends around me, who have made my life here exciting and colorful. I sincerely thank Yu Lin, Xiuwei Zhang, and Le Chen, who have helped me with almost everything over the years. My special thank goes to Min Ye, Ping Xi and Fei Pu, who have always been there for me. I would like to thank Tian Guo for introducing me to jogging, Yun Bai for organizing hiking tours, Zhou Xue and Xiaolu Sun for improving my cooking skills, and many others—Han Wu, Meiyue Shao, Lu Qiao and Peng Chen, Liang Qiao, and Yuxuan Ji, etc, with whom I have spent many pleasant moments.

Finally, I am deeply indebted to my beloved family—my grandmother, my parents, my sister and brother-in-law, and my little nephew, for their unconditional love, care and support.

Lausanne, 11 May 2015 Mingfu Shao

Abstract

The deluge of sequenced whole-genome data has motivated the study of comparative genomics, which provides global views on genome evolution, and also offers practical solutions in deciphering the functional roles of components of genomes. A fundamental computational problem in whole-genome comparison is to infer the most likely large-scale events (rearrangements and content-modifying events) of given genomes during their history of evolution. Based on the principle of parsimony, such inference is usually formulated as the so called edit distance problems (for two genomes) or median problems (for multiple genomes), i.e., to compute the minimum number of certain types of large-scale events that can explain the differences of the given genomes. In this dissertation, we develop novel algorithms for edit distance problems and median problems and also apply them to analyze and annotate biological datasets.

For pairwise whole-genome comparison, we study the most challenging cases of edit distance problems—the given genomes contain duplicate genes. We proposed several exact algorithms and approximation algorithms under various combinations of large-scale events. Specifically, we designed the first exact algorithm to compute the edit distance under the DCJ (double-cut-and-join) model, and the first exact algorithm to compute the edit distance under a model including DCJ operations and segmental duplications. We devised a $(1.5+\epsilon)$-approximation algorithm to compute the edit distance under a model including DCJ operations, insertions, and deletions. We also proposed a very fast and exact algorithm to compute the exemplar breakpoint distance. For multiple whole-genome comparison, we study the median problem under the DCJ model. We designed a polynomial-time algorithm using a network flow formulation to compute the so called adequate subgraphs—a central phase in computing the median. We also proved that an existing upper bound of the median distance is tight.

These above algorithms determine the correspondence between functional elements (for instance, genes) across genomes, and thus can be used to systematically infer functional relationships and annotate genomes. For example, we applied our methods to infer orthologs and in-paralogs between a pair of genomes—a key step in analyzing the functions of protein-coding genes. On biological whole-genome datasets, our methods run very fast, scale up to whole genomes, and also achieve very high accuracy.

Key words: rearrangement, inversion, double-cut-and-join, edit distance, median, gene duplication, ortholog, paralog, integer linear program, network flow

Contents

List of Figures

List of Tables

1 Introduction

The genome of an organism is the set of its genetic material, which encodes all the hereditary instructions for building and maintaining that organism. During the history of evolution, genomes have been constantly shaped by mutation and selected by natural environment, which eventually leads to the myriad species with which we share the planet. As more and more genomes of different species are sequenced facilitated by the rapid development of high-throughput sequencing technologies, comparative genomics has found its indispensable place in deciphering the mystery of genome and evolution. In this chapter, we first introduce the biological background of comparative genomics, mainly focusing on the two types of evolutionary events and their underlying molecular mechanisms. Then we state the fundamental computational problems in comparative genomics as well as the current challenges and state-of-the-art solutions. Finally, we summarize the contributions of this dissertation.

1.1 Genome Evolution

In biology, a mutation is a permanent change of the nucleotide sequence of genomes. Mutations are raw materials and driving force of evolution—every genetic feature in every organism was, initially, the result of a mutation. According to the range of affected genome, genomic mutations can be divided into *small-scale* events and *large-scale* events. These two types of events have different underlying molecular mechanisms and also convey distinct genetic consequences.

Small-scale events include nucleotides substitutions and indels. Substitutions, which exchange a single nucleotide for another, are often caused by chemicals or malfunction of DNA replication. The effect of a substitution that occurs within the protein coding regions of a gene can be *silent*, if the erroneous codon codes for the same (or a sufficiently similar) amino acid, or *missense*, if it codes for a different amino acid, or *nonsense*, if it codes for a stop codon and thus truncates the protein. Indels, which add or remove one or several nucleotides, are usually caused by transposable elements, or errors during the replication of repeating elements. In protein coding regions, an indel may result in a *frameshift*, if the length of an indel is not a multiple of 3. Frameshift will alter all the following amino acids, which usually cause to encounter a premature stop codon and thus produces a truncated protein. If the length of an indel is divisible by three, usually the produced protein will gain or loss a few amino acids.

In contrast to the small-scale events that only affect a short and local region of genomes, large-scale events (also called structural variations, genomic disorders) are referred to DNA changes ranging from thousands to millions of base pairs. These events are further partitioned into *rearrangements*—inversions, translocations, chromosome fusions and fissions—and *content-modifying events*—tandem duplications, segmental duplications, whole-genome duplications, gene insertions (including lateral transfer) and losses. Rearrangements may disrupt genes, create fusion genes, shuffle orders and switch transcriptional orientations of genes (see Figure 1.1). Content-modifying events can also affect copy number of genes.

Although still quite debatable, several molecular mechanisms have been proposed for large-scale events [1]. Non-allelic homologous recombination (NAHR) between two low-copy repeats (LCRs) is a major mechanism for recurrent large-scale events [2]. LCRs usually share very high degree of sequence identity, which act as the mediators of NAHR. In meiosis or mitosis, instead of the copies at the usual allelic positions, non-allelic copies of LCRs can be aligned. Consequently, the subsequent crossover can result in various large-scale events (see Figure 1.2). For example, when two LCRs are located on the same chromosome and in direct orientation, NAHR between them causes reciprocal duplication and deletion; when they are on the same chromosome but in opposite orientation, NAHR results in inversion.

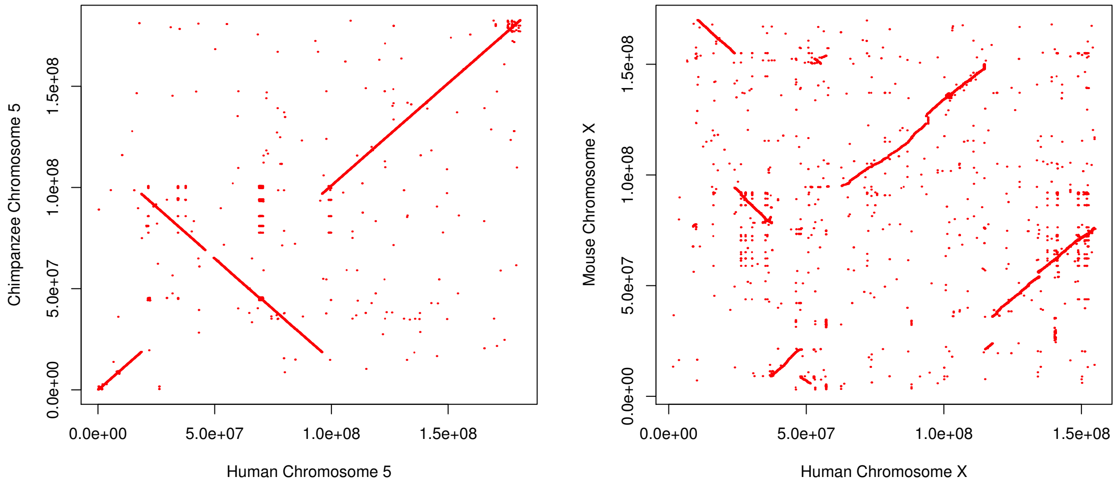


Figure 1.1 – The left part shows the dot-plot between chromosome 5 of human and chromosome 5 of chimpanzee. There is a clear inversion between these two chromosomes. The right part shows the dot-plot between chromosome X of human and chromosome X of mouse. There are several rearrangements between these two chromosomes.

Non-homologous end-joining (NHEJ) is a major mechanism for nonrecurrent large-scale events [3]. NHEJ is a pathway that repairs double-strand breaks (DSBs) in DNA. Inappropriate NHEJ may lead to translocations and telomere fusion. For example, two DSBs followed by two NHEJs that join the four DNA ends in a wrong combination can result in a translocation (see Figure 1.3).

Fork stalling and template switching (FoSTeS) is regarded as a replication-based mechanism for explaining complex large-scale events [4] (see Figure 1.4). FoSTeS has also been generalized into another model, called microhomology-mediated break-induced replication (MMBIR) [5]. In this model, the replication fork may switch template in virtue of microhomology multiple times, and thus causes complicated large-scale events.

1.2 Comparative Genomics

Due to the accumulation of these mutation events in the course of evolution, extant genomes of different species exhibit certain degree of divergence. However, the functional elements (genes,

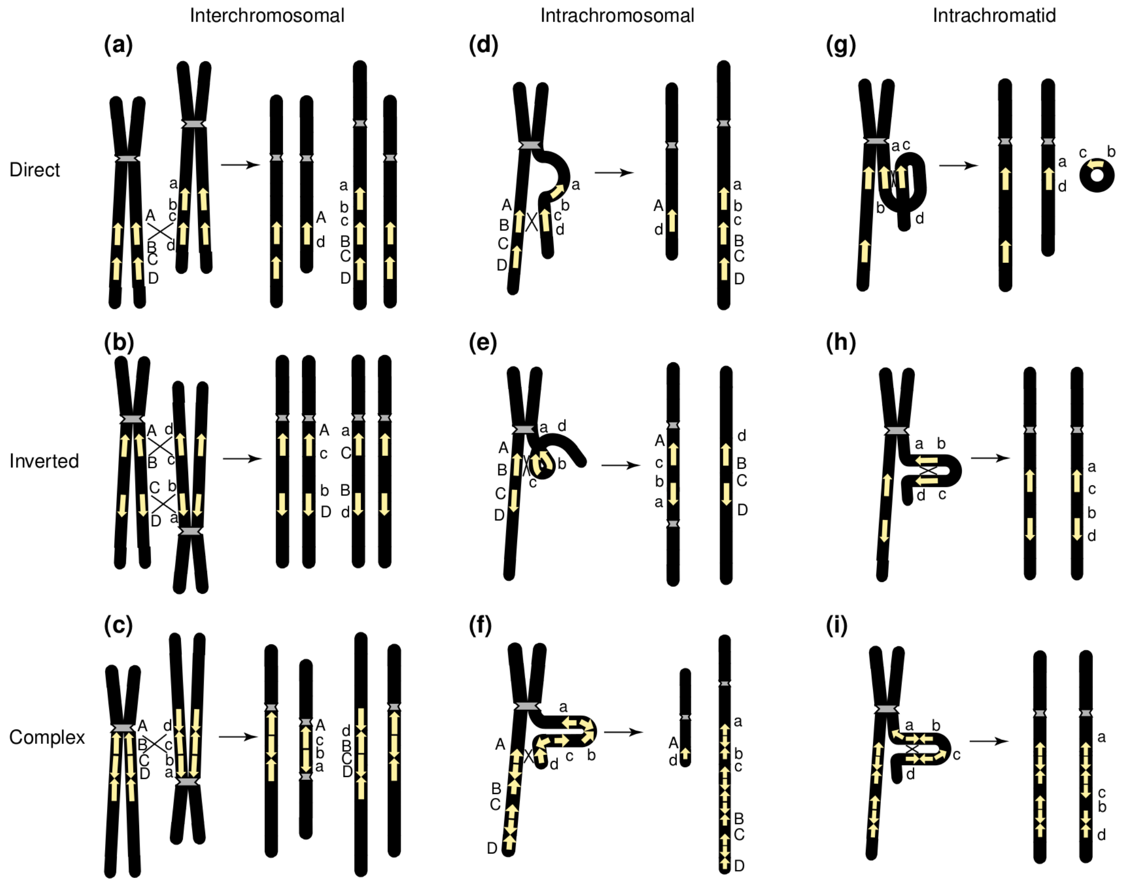


Figure 1.2 – Various non-allelic homologous recombinations (NAHR) mediated by two or more LCRs (marked by yellow arrows) [6].

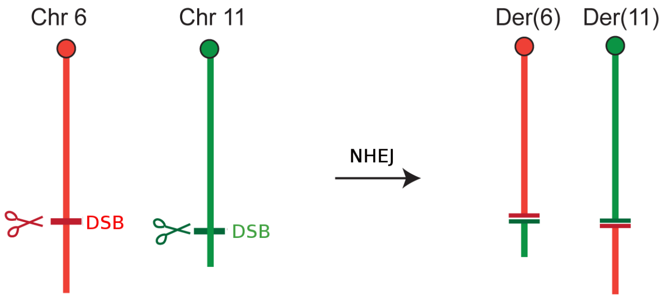


Figure 1.3 – An example illustrating that non-homologous end-joining (NHEJ) results in a translocation [7].

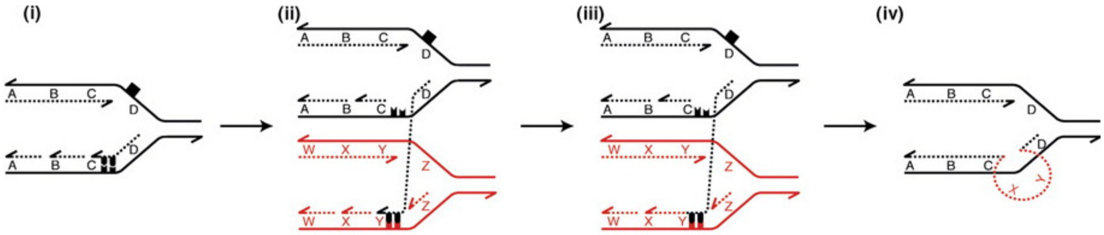


Figure 1.4 – Illustration of fork stalling and template switching (FoSTeS). (i) A replication fork stalls after encountering a DNA lesion (black square) on the template strand. (ii) The lagging strand from the stalled replication fork anneals to the template strand of another replication fork using microhomology. (iii) The invading strand is extended. (iv) The lagging strand returns to the original template, allowing normal DNA replication to resume [8].

regulatory elements, etc) and biological pathways across related species have a strong tendency of conservation by virtue of natural selection. Comparative genome analysis thus allows us to establish the correspondence between functional elements across compared genomes, as well as to understand their functions by translating knowledge gained about some genomes to the object of study.

To achieve this, a basic and fundamental task is to correctly infer the evolutionary events took place during the history of evolution. This task is usually formulated as an optimization problem under the principle of parsimony, called *edit distance* problem (for two genomes) or *median* problem (for multiple genomes), which is to seek a set of minimum number of mutation events that can explain the difference of the compared genomes. For different genomic datasets (sequence data or whole-genome data), different mutation events should be considered, and different edit distance problems and median problems are formulated and studied.

1.2.1 Sequence Comparison

When we are interested in comparing nucleotide sequences (usually a single transcript or a protein sequence), small-scale evolutionary events are considered. In this case, the edit distance between two sequences, i.e., the minimum number of substitutions and indels that can transform one sequence into the other, can be computed by efficient dynamic programming algorithm. Further, counting events is replaced by more sophisticated substitution matrices and gap models, and under this generalized scoring function, dynamic programming algorithm can still be used to compute the optimal solution. In addition to pairwise sequence comparison, multiple sequence alignment is also widely studied. Many tools and softwares have been proposed for both pairwise and multi-way sequence comparison, which are now playing a very important role in the study of biology.

1.2.2 Whole-genome Comparison

When we are interested in comparing whole-genomes, large-scale events are considered. In this case, it is more interesting and convenient to only consider genes (or syntenic blocks, conserved sequence markers) in the genomes, rather than modeling each genome as a long DNA sequence [9]. That is, we model each genome as a set of chromosomes while model each chromosome as a linear or circular list of genes (see Figure 1.5). In this model, each gene has a sign, indicating its transcriptional direction. All genes in the given genomes are grouped into *gene families*, which are usually built through clustering based on their sequence similarity. We say genes in the same gene family are *homologous*. Throughout this dissertation, homologous genes are represented with the same symbol and distinguished by different superscripts.

With this model of genomes, these large-scale events can be formally defined. For rearrangement events, an *inversion* reverses a segment (several continuous genes on one chromosome) and also switches all the signs of the genes on this segment (see Figure 1.6(**a**)); an *translocation* reciprocally exchanges segments (with telomeres) between two linear chromosomes (see Figure 1.6(**b**)); an *chromosome fusion* merges two chromosomes into one while an *chromosome fission* splits one chromosome into two (see Figure 1.6(**c**)). For content-modifying events, a *segmental duplication* copies a segment and then inserts the copy to another position (see Figure 1.7).

Yancopoulos *et al.* proposed a universal operation, called double-cut-and-join (DCJ), which

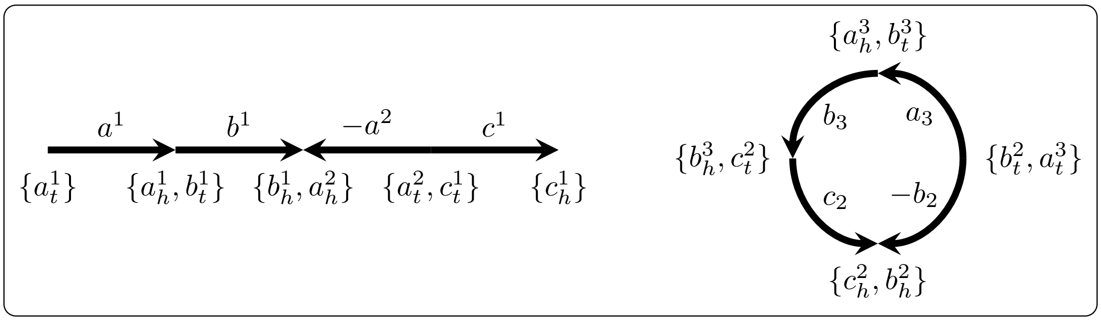


Figure 1.5 – Example of a genome consisting of a linear chromosome with 4 genes and a circular chromosome with 4 genes. All adjacencies and telomeres are also illustrated.

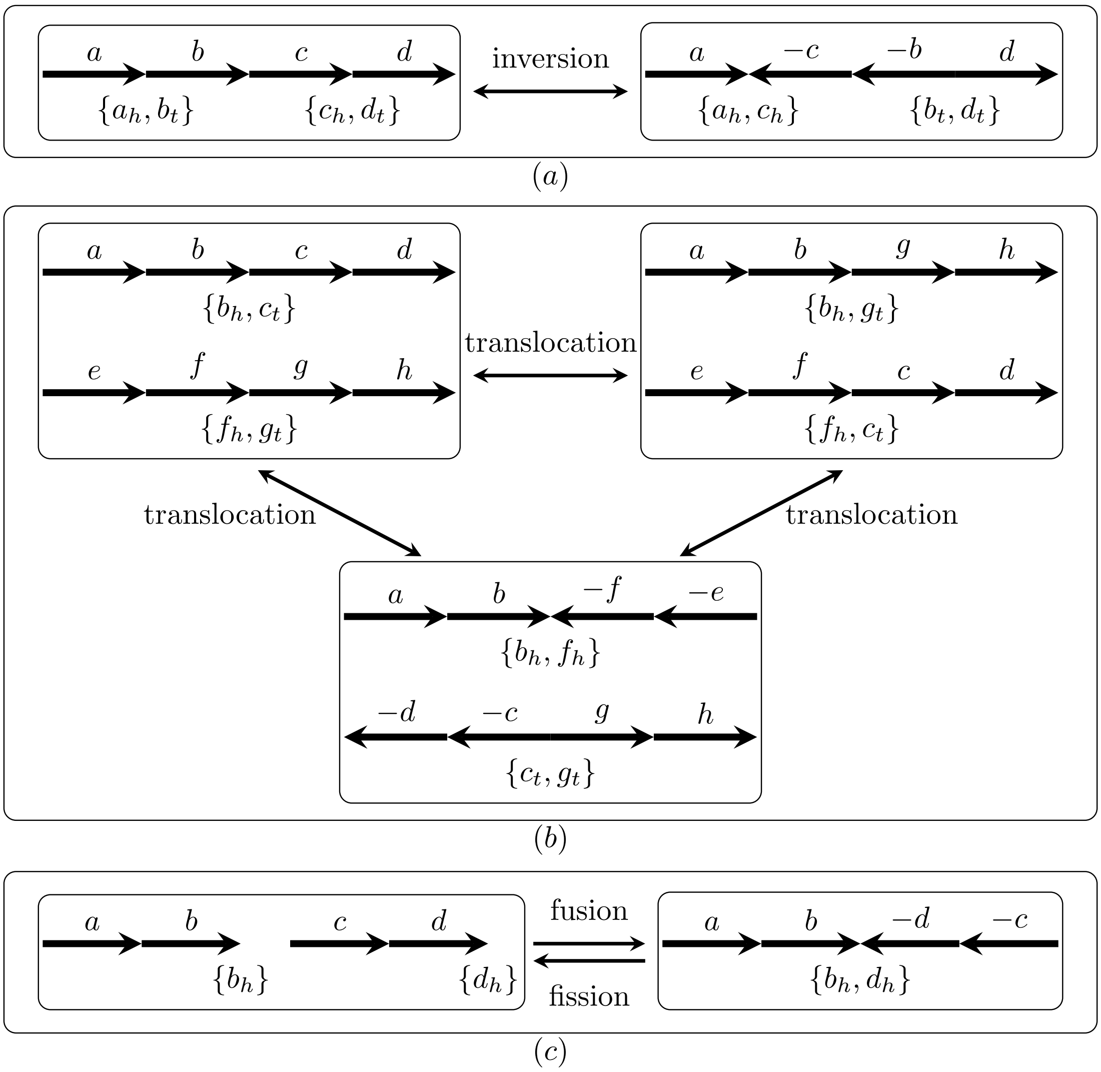


Figure 1.6 – Examples of rearrangements.

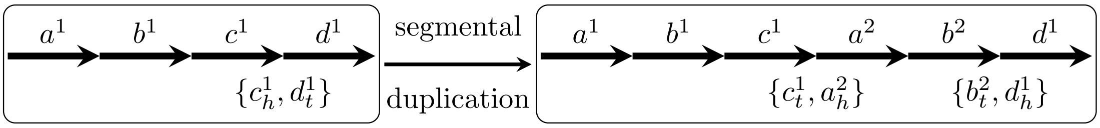


Figure 1.7 – Example of a segmental duplication, in which segment of (a^2, b^2) is duplicated using segment of (a^1, b^1) as template.

can be used to represent almost all the existing rearrangement events [10]. To define the DCJ operation, we further introduce some notations [11]. The two ends of a gene g (head and tail, represented by g_h and g_t respectively) are called *extremities*. Two consecutive genes form one *adjacency*, which is represented as a set of two extremities. The two ends of each linear chromosome are called *telomeres*, each of which is represented by a set of a single extremity (see Figure 1.5). With these notations, a DCJ operation can be defined as follows: it takes one or two adjacencies (or telomeres) in the genome as input, then produces one or two new adjacencies (or telomeres) through recombining all the involved extremities. Formally, a DCJ operation falls in one of the following four cases:

- $\{p, q\} + \{r, s\} \rightarrow \{p, r\} + \{q, s\}$, or $\{p, s\} + \{q, r\}$;
- $\{p, q\} + \{r\} \rightarrow \{p, r\} + \{q\}$, or $\{p\} + \{q, r\}$;
- $\{p, q\} \rightarrow \{p\} + \{q\}$;
- $\{p\} + \{q\} \rightarrow \{p, q\}$.

Clearly, inversion, translocation, chromosome fusion and fission are all special cases of DCJ operation (see Figure 1.6). Because of its generality and universality, DCJ operation has formed the basis for most algorithmic research on rearrangements over the last few years [12, 13, 14, 15, 16, 17].

Feijão and Meidanis proposed another general operation, called single-cut-or-join (SCJ) [18]. A SCJ operation either performs a *cut* in the genome, which breaks one adjacency into two telomeres, or performs a *join*, which merges two telomeres into one adjacency. SCJ model is a formalization of the commonly used term of *breakpoint* in studying structural variations.

Similar to sequence comparison, a basic and fundamental computational problem for pairwise whole-genome comparison is to compute the *edit distance* between two given genomes, which is defined as the minimum number of certain types of large-scale events that can transform one genome into the other. There are many different versions of edit distance problems, depending on different assumptions on the given genomes and various combinations of large-scale events.

When the two given genomes do not contain duplicated genes, i.e., each gene family in each genome consists of exactly one gene (see Figure 1.8**(a)**), most of the edit distances are well-defined and can be computed in polynomial-time. Under the inversion model, Hannenhalli and Pevzner pioneeringly gave the first polynomial-time algorithm to compute the edit distance for two unichromosomal genomes [19], which was later improved to linear time [20]. Under the model including inversions and translocations, the edit distance problem has been studied through a series of papers [19, 21, 22, 23], culminating in a fairly complex linear-time algorithm [14]. Under the DCJ model, the edit distance can be computed in linear time in a elegant way [11]. Under the SCJ model, the edit distance (also called breakpoint distance) can also be computed very simply in linear time. Under the model including inversions and insertions (or deletions), El-Mabrouk proposed an efficient algorithm to compute the edit distance [24]. Finally, under the model including DCJ operations, insertions and deletions, Braga *et al.* gave a linear time algorithm to compute the edit distance [25, 26], which was then simplified in [27].

However, gene duplications are widespread events and have long been recognized as a major driving force of evolution [28, 29]. Thus, it is much more interesting and important to compare whole-genomes with duplicate genes (see Figure 1.8(**b**)). In this case, we need to infer the one-to-one correspondence between the homologous genes across the two given genomes. Formally, the edit distance problem in the presence of duplicate genes can be defined as to compute a matching between homologous genes, such that the edit distance (under a certain model) between the two genomes induced by this matching (we can assign each pair of genes in it a new gene family, thus the two genomes can be regarded without duplicate genes and the edit distance is well-defined) is minimized. According to the cardinality of the matching for each gene family, we have three different strategies:

- *exemplar* strategy [30], which is for each gene family to select exactly one gene and remove other copies in each genome (see Figure 1.8(**c**));
- *intermediate* strategy [31, 32], which is for each gene family to select the same number (at least one) of genes and remove other copies in each genome;
- *maximum matching* strategy [33], which is for each gene family to select as many genes as possible (the smaller size of this gene family between the two genomes) and remove other copies in each genome (see Figure 1.8(**d**)).

Unfortunately, with either strategy almost all edit distance formulations are NP-hard. For example, with the exemplar strategy, the breakpoint distance problem and the inversion distance problem have been proved NP-hard [34]. Moreover, the breakpoint distance cannot

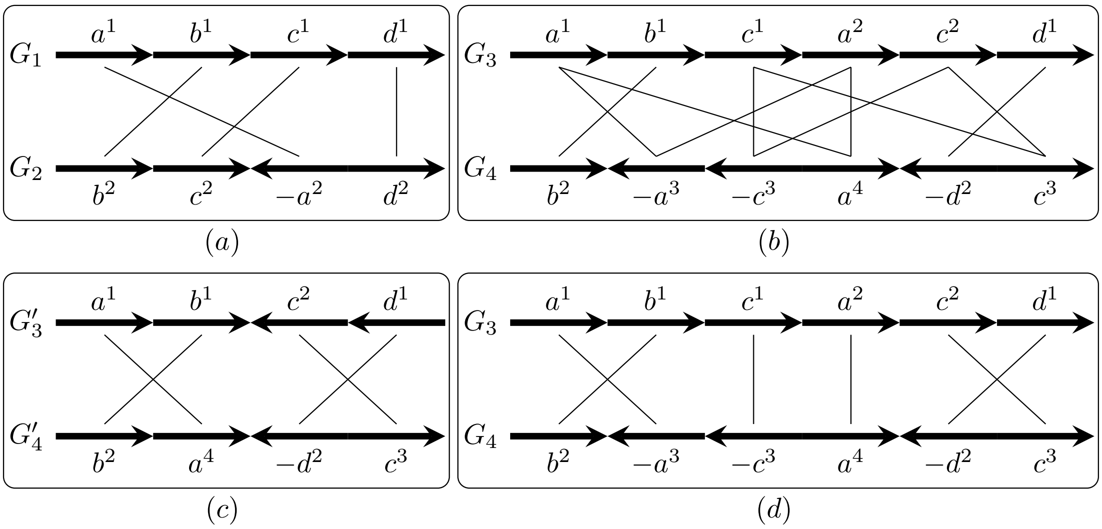


Figure 1.8 – (a) Two genomes without duplicate genes. The one-to-one correspondence (represented by thin lines) between homologous genes is unique. **(b)** Two genomes with duplicate genes, in which the correspondence is not unique. **(c)** One possibility under exemplar strategy for G_3 and G_4. **(d)** One possibility under the maximum matching strategy for G_3 and G_4.

be approximated in polynomial time unless P = NP [35, 36]. With the maximum matching strategy, both the inversion distance and the DCJ distance have been proved NP-hard [37].

Many exact algorithms and heuristics have been proposed for these edit distance problems in the presence of duplicate genes. With the exemplar strategy, Sankoff [30] gave a first branch-and-bound algorithm to compute the breakpoint and inversion distances. Nguyen *et al.* gave a much faster divide-and-conquer approach for the breakpoint distance [38]. With the intermediate strategy, Angibaud *et al.* [31] proposed an exact algorithm using integer linear programming to compute the breakpoint distance. With the maximum matching strategy, Suksawatchon *et al.* proposed a heuristic to compute the inversion distance using integer linear programming [39]. Chen *et al.* proposed an efficient heuristic to compute the inversion distance through decomposing this problem into two new optimization problems, called *minimum common partition* and *maximum cycle decomposition*, for which efficient heuristic algorithms are given [37]. Later, their method was extended for a model including rearrangements and single-gene duplications [40], which was then implemented as a software package, called MSOAR. MSOAR has been applied to assign orthologs and paralogs between two genomes, and its performance has been shown outperformed other orthologous assignment softwares based on sequence similarity (like INPARANOID [41]).

Similar to multiple sequence alignment, multiple genome comparison is also fairly studied. The simplest formulation is the *median* problem, which is a generalization of the edit distance problem. The median problem is given three genomes, to construct a fourth one, such that the sum of the edit distances between itself and each of the three given genomes is minimized. The median problem is NP-hard for almost all formulations [42, 43] even if the given genomes are without duplicate genes. Under the inversion model, several exact algorithms [44, 45] and heuristics [46, 47] have been proposed. Under the DCJ model, Zhang *et al.* presented an exact solver using a branch-and-bound framework [48]. Xu *et al.* proposed a promising decomposition scheme by proving an optimal substructure of the problem [49, 50].

1.3 Contributions of this Dissertation

We focus on whole-genome comparison. The main contributions of this dissertation are in two folds. First, we designed novel algorithms for these edit distance problems and median problems, listed as follows.

1. We proposed a very fast and exact algorithm to compute the breakpoint distance with the exemplar strategy in the presence of duplicate genes (see Chapter 2) [51].
2. We proposed the first exact algorithm to compute the DCJ distance with the maximum matching strategy in the presence of duplicate genes (see Chapter 3) [52].
3. We proposed the first exact algorithm to compute the edit distance under a model including DCJ operations and segmental duplications with the maximum matching

strategy in the presence of duplicate genes (see Chapter 4) [53].

4. We devised a $(1.5+\epsilon)$-approximation algorithm to compute the edit distance under a model including DCJ operations, insertions and deletions with the maximum matching strategy in the presence of duplicate genes (see Chapter 5) [54].

5. We designed a polynomial-time algorithm to compute the *adequate subgraph*, a key step in solving the median problem. We also proved that an existing upper bound for the median problem is tight (see Chapter 6) [55].

6. We proposed a new universal graphical data structure, *trajectory graph*, which can represent both rearrangements and content-modifying events in a very natural way. We also gave a polynomial-time algorithm to simplify the trajectory graph (see Chapter 7) [56].

Second, we applied these algorithms to analyze whole-genome datasets and to infer functional relationships between them. For example, the algorithm proposed in [52, 51, 53] can be used to infer orthologs (and in-paralogs [53]) between two genomes. We tested our methods on both simulated datasets and biological datasets, and the results demonstrated that our methods can achieve very high accuracy (about 99% on five mammalian genomes) and outperformed the state-of-the-art orthology assignment method MSOAR [37, 40, 57].

2 Exemplar Breakpoint Distance

In this chapter, we propose a very fast and exact algorithm to compute the breakpoint distance with the exemplar strategy. In Section 1, we first formally state the problem using the equivalent definition of shared adjacencies (rather than breakpoints). In Section 2, we describe our algorithm, which consists of an integer linear program formulation and two specific techniques, one is to add a collection of new constraints to the ILP formulation and thereby reduce its search space, and the other is to identify optimal substructures and thereby reduce the size of the instance. In Section 3, we evaluate the efficiency of these two techniques and compare our full algorithm with the state-of-the-art divide-and-conquer algorithm through simulation. In Section 4, we propose a very fast heuristic through iteratively applying this exact algorithm to maximize the number of shared adjacencies under the intermediate strategy, which can be naturally used to infer orthologs. The performance of this method is then tested on five well-annotated genomes and compared with MSOAR.

2.1 Problem Statement

We model each genome as a set chromosomes and each chromosome as a (linear or circular) list of genes (see Section 1.2.2). For each linear chromosome in the genome, we always add one *capping* gene, represented by τ, to each of its two ends. Given a chromosome, we can reverse the list of symbols and switch all the signs, which will result in the same chromosome. For example, an equivalent representation of a linear chromosome $(\tau, a_1, a_2, \cdots, a_m, \tau)$ is $(\tau, -a_m, \cdots, -a_2, -a_1, \tau)$.

Genes in the input genomes are grouped into *gene families*, and genes in the same family are called *homologous*. We place all of the capping “genes" into a single gene family of their own, denoted by f_τ. Given a genome G, we denote by $\mathscr{A}(G)$ the set of all the gene families in G. We define $\widetilde{\mathscr{A}}(G) = \mathscr{A}(G) \setminus \{f_\tau\}$. For a gene family $f \in \mathscr{A}(G)$, we use $F(G, f)$ to denote the set of genes in G that come from f. Given a genome G, we can delete all but one gene for each non-capping gene family, resulting in a new genome satisfying $|F(G, f)| = 1$ for all $f \in \widetilde{\mathscr{A}}(G)$, called an *exemplar* of G.

Two consecutive genes a and b on the same chromosome form an *adjacency*, written as (a, b). Notice that (a, b) and $(-b, -a)$ are the same adjacency. (This definition of adjacency is equivalent with the form consisting of two extremities in Section 1.2.2. We can write $(a, b) = (-b, -a) = \{a_h, b_t\}$.) We say an adjacency (a, b) is *simple* if a and b come from different gene families. Given two genomes G_1 and G_2, we say two simple adjacencies $(a^1, b^1) \in G_1$ and $(a^2, b^2) \in G_2$ form a *pair of shared simple adjacencies* (PSSA), written as $\langle(a^1, b^1), (a^2, b^2)\rangle$, if a^1 and a^2 (and also b^1 and b^2) have the same sign and come from the same gene family. We use $s(G_1, G_2)$ to denote the number of PSSAs between G_1 and G_2. Given two genomes G_1 and G_2 with $\widetilde{\mathscr{A}}(G_1) = \widetilde{\mathscr{A}}(G_2)$, the *exemplar breakpoint distance problem* is to compute a pair of exemplars, G'_1 and G'_2 for G_1 and G_2 respectively, such that $s(G'_1, G'_2)$ is maximized (see Figure 2.1 for an example).

2.2 Algorithm

We describe a fast and exact algorithm for the exemplar breakpoint distance problem. The algorithm consists of a preprocessing phase to reduce the complexity of the instance, followed by an integer linear program (ILP) to solve the reduced instance. In the preprocessing phase, the algorithm iteratively identifies optimal substructures, fixing in place those it finds, until no such substructure can be found. The reduced instance is then formulated as an ILP. By using the properties of the optimal solution, we can add a collection of new constraints to the ILP, thereby reducing the search space and making the ILP very efficient. For ease of description, we present first the ILP formulation, in Section 2.2.1, then the algorithm for adding the extra constraints, in Section 2.2.2, and finally the algorithm for identifying the optimal substructures, in Section 2.2.3.

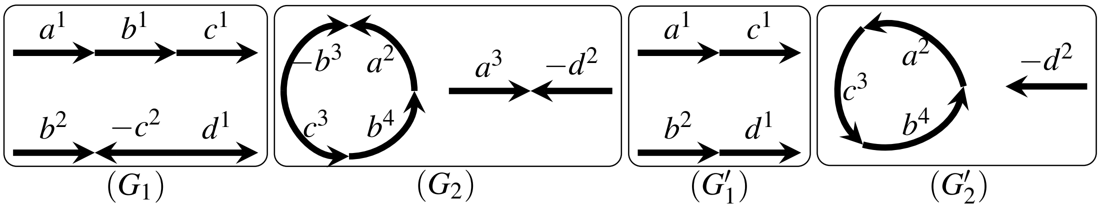


Figure 2.1 – G_1 contains two linear chromosomes, and G_2 contains one linear and one circular chromosomes. $\mathscr{A}(G_1) = \mathscr{A}(G_2) = \{\tau, a, b, c, d\}$. G'_1 and G'_2 are two exemplars of G_1 and G_2, respectively. There are two PSSAs, $q_1 = \langle(a^1, c^1), (a^2, c^3)\rangle$ and $q_2 = \langle(d^1, \tau), (\tau, -d^2)\rangle$, between G'_1 and G'_2, which means that $s(G'_1, G'_2) = 2$. We have $\mathscr{P}(G_1, G_2) = \{\langle[\tau, a^1], [\tau, a^3]\rangle, \langle[a^1, c^1], [a^2, c^3]\rangle, \langle[b^1, c^1], [b^4, c^3]\rangle, \langle[b^2, -c^2], [-b^3, c^3]\rangle, \langle[\tau, d^1], [-d^2, \tau]\rangle, \langle[d^1, \tau], [\tau, -d^2]\rangle\}$. There are two PSSPAs, $p_1 = \langle[a^1, c^1], [a^2, c^3]\rangle$ and $p_2 = \langle[d^1, \tau], [\tau, -d^2]\rangle$, that survive in G'_1 and G'_2, and p_1 and p_2 are the indicators of q_1 and q_2, respectively.

2.2.1 ILP Formulation

We first generalize the definition of adjacency. We say two genes a and b on the same chromosome form a *potential adjacency*, written as $[a,b]$, if we can remove all genes between a and b and yet retain at least one gene in every gene family in every genome. (Adjacencies are just potential adjacencies where nothing need to be removed.) We say a potential adjacency $[a,b]$ is *simple* if a and b come from different gene families. Given two genomes G_1 and G_2, we say two simple potential adjacencies $[a^1,b^1] \in G_1$ and $[a^2,b^2] \in G_2$ form a *pair of shared simple potential adjacencies* (PSSPA), written as $\langle [a^1,b^1],[a^2,b^2]\rangle$, if a^1 and a^2 (and also b^1 and b^2) have the same sign and come from the same gene family. Given a pair of exemplars G_1' and G_2' of G_1 and G_2 respectively, we say a PSSPA $p = \langle [a^1,b^1],[a^2,b^2]\rangle$ between G_1 and G_2 *survives* in G_1' and G_2' if $q = \langle (a^1,b^1),(a^2,b^2)\rangle$ is a PSSA between G_1' and G_2'; we then say that the PSSPA p is the *indicator* of the PSSA q. We denote by $\mathscr{P}(G_1,G_2)$ the set of all PSSPAs between G_1 and G_2 (see Figure 2.1). Clearly, all PSSAs in any pair of exemplars of G_1 and G_2 can find their indicators in $\mathscr{P}(G_1,G_2)$. We say a set of PSSPAs P can *coexist* w.r.t. G_1 and G_2 if there exists a pair of exemplars G_1' and G_2' of G_1 and G_2 such that all PSSPAs in P survive in G_1' and G_2' simultaneously. Thus, the exemplar breakpoint distance problem can be restated as computing a subset of $\mathscr{P}(G_1,G_2)$ with maximum cardinality that can coexist—and this new perspective leads to our ILP formulation.

We use two types of variables. First, for every gene a in the two given genomes, we use one binary variable x_a to indicate whether this gene appears in the final pair of exemplars. Second, for every PSSPA $p \in \mathscr{P}(G_1,G_2)$ we use one binary variable y_p to indicate whether p survives.

Our ILP has two types of general constraints. First, we require that for each non-capping gene family in each genome, exactly one gene appears in the final exemplar:

$$\begin{aligned} \sum_{a\in F(G_1,f)} x_a &= 1, \quad \forall f \in \widetilde{\mathscr{A}}(G_1),\\ \sum_{a\in F(G_2,f)} x_a &= 1, \quad \forall f \in \widetilde{\mathscr{A}}(G_2). \end{aligned}$$

Second, for each PSSPA $p = \langle [a^1,b^1],[a^2,b^2]\rangle$, we require that, if p survives, then the two pairs, $\langle a^1,a^2\rangle$ and $\langle b^1,b^2\rangle$, must appear in the final exemplars, while those genes between a^1 and b^1 (and also between a^2 and b^2) cannot appear in the final exemplars:

$$\begin{aligned} y_p &\le x_{a^1}, x_{a^2}, x_{b^1}, x_{b^2}, && \forall p = \langle [a^1,b^1],[a^2,b^2]\rangle \in \mathscr{P}(G_1,G_2),\\ x_a &\le 1 - y_p, && \forall a \text{ between } a^1 \text{ and } b^1 \text{ or between } a^2 \text{ and } b^2. \end{aligned}$$

The objective of the ILP is to maximize the sum of the variables for PSSPAs:

$$\max \sum_{p\in\mathscr{P}(G_1,G_2)} y_p.$$

Our ILP formulation is similar to the one proposed in [31], but has fewer variables; it uses some of the same ideas we used for the DCJ distance problem [52]. In our testing, all ILP instances are solved with GUROBI [58].

2.2.2 Adding Inference Constraints

The constraints we add are based on two new properties of optimal exemplars.

Lemma 2.2.1. *Let G_1 and G_2 be two exemplars. If we remove one non-capping gene from the same gene family for each genome, resulting in two new genomes $\overline{G_1}$ and $\overline{G_2}$ respectively, then we have $s(\overline{G_1},\overline{G_2}) \geq s(G_1,G_2)-1$.*

Proof. Let a^1 and a^2 be the two genes that are removed from G_1 and G_2, respectively. Assume that a^1 and a^2 have the same sign; otherwise we can reverse the chromosome on which a^2 is found. Let $\overleftarrow{a^1}$ and $\overrightarrow{a^1}$ (resp. $\overleftarrow{a^2}$ and $\overrightarrow{a^2}$) be the predecessor and successor of a^1 (resp. a^2). In terms of adjacencies, we have that $(\overleftarrow{a^1},a^1)$ and $(a^1,\overrightarrow{a^1})$ (resp. $(\overleftarrow{a^2},a^2)$ and $(a^2,\overrightarrow{a^2})$) are removed from G_1 (resp. G_2), and $(\overleftarrow{a^1},\overrightarrow{a^1})$ (resp. $(\overleftarrow{a^2},\overrightarrow{a^2})$) is added to G_1 (resp. G_2). If $(\overleftarrow{a^1},a^1)$ and $(\overleftarrow{a^2},a^2)$ do not form a PSSA, or $(a^1,\overrightarrow{a^1})$ and $(a^2,\overrightarrow{a^2})$ do not form a PSSA, then we lose at most one PSSA and thus the conclusion holds. Otherwise, $(\overleftarrow{a^1},\overrightarrow{a^1})$ and $(\overleftarrow{a^2},\overrightarrow{a^2})$ must form a new PSSA between $\overline{G_1}$ and $\overline{G_2}$, so the conclusion also holds. □

Let G'_1 and G'_2 be a pair of exemplars of G_1 and G_2. We say a PSSA $p = \langle (a^1,b^1),(a^2,b^2)\rangle$ between G_1 and G_2 *agrees with* G'_1 and G'_2 if either both $\langle a^1,a^2\rangle$ and $\langle b^1,b^2\rangle$ appear in G'_1 and G'_2, or neither $\langle a^1,a^2\rangle$ nor $\langle b^1,b^2\rangle$ appears in G'_1 and G'_2.

Lemma 2.2.2. *For any PSSA p between G_1 and G_2, there always exists an optimal pair of exemplars of G_1 and G_2 with which p agrees.*

Proof. Let $p = \langle (a^1,b^1),(a^2,b^2)\rangle$ be a PSSA. Suppose that there exists one optimal pair of exemplars G^*_1 and G^*_2 with which p does not agree; otherwise the lemma is proved. Without loss of generality, we assume that $\langle a^1,a^2\rangle$ appears in G^*_1 and G^*_2, but $\langle b^1,b^2\rangle$ does not (thus b^1 and b^2 are non-capping genes). Let b^3 and b^4 be the two genes in G^*_1 and G^*_2 that come from the same gene family as b^1 and b^2 (we may have $b^3 = b^1$ or $b^4 = b^2$, but not both). Let $\overline{G^*_1}$ and $\overline{G^*_2}$ be the two genomes after removing $\langle b^3,b^4\rangle$ from G^*_1 and G^*_2. According to Lemma 2.2.1, we have $s(\overline{G^*_1},\overline{G^*_2}) \geq s(G^*_1,G^*_2)-1$. We then insert $\langle b^1,b^2\rangle$ into $\overline{G^*_1}$ and $\overline{G^*_2}$ to create the PSSA $\langle (a^1,b^1),(a^2,b^2)\rangle$, resulting in two new genomes G'_1 and G'_2. Clearly, we have $s(G'_1,G'_2) = s(\overline{G^*_1},\overline{G^*_2})+1$. Combining these two formulas yields $s(G'_1,G'_2) \geq s(G^*_1,G^*_2)$, which implies that G'_1 and G'_2 are also an optimal pair of exemplars of G_1 and G_2. □

Given a PSSA $\langle (a^1,b^1),(a^2,b^2)\rangle$, Lemma 2.2.2 allows us to add the following *inference constraints* to the ILP, which guarantee that, for the two pairs of genes $\langle a^1,a^2\rangle$ and $\langle b^1,b^2\rangle$, the appearance of one pair implies the appearance of the other pair:

$$
\begin{aligned}
x_{b^1}, x_{b^2} &\geq x_{a^1} + x_{a^2} - 1,\\
x_{a^1}, x_{a^2} &\geq x_{b^1} + x_{b^2} - 1.
\end{aligned}
$$

However, we cannot add such constraints simultaneously for two or more PSSAs, as the following example demonstrates.

Example 2.2.1. Let $G_1 = (\tau, a^1, b^1, c^1, d^1, a^2, e^1, \tau)$ and $G_2 = (\tau, a^3, b^2, e^2, d^2, a^4, c^2, \tau)$. We have $|F(G_1, f)| = |F(G_2, f)| = 1$ for each $f \in \{b, d, e\}$. If the appearance of $\langle b^1, b^2\rangle$ implies that of $\langle a^1, a^3\rangle$ *and* the appearance of $\langle d^1, d^2\rangle$ implies that of $\langle a^2, a^4\rangle$, we get two genes from gene family a in each exemplar.

For a PSSA p, we denote by $\mathscr{A}(p)$ the two gene families in p. We say two PSSAs p and q are *independent*, if we have $\mathscr{A}(p) = \mathscr{A}(q)$, or $\mathscr{A}(p) \cap \mathscr{A}(q) = \emptyset$. Given two genomes G_1 and G_2, we denote by $\mathscr{Q}(G_1, G_2)$ the set of all PSSAs between G_1 and G_2 (see Figure 2.2 for an example). We say a subset $Q \subset \mathscr{Q}(G_1, G_2)$ is *mutually* independent, if every two PSSAs in Q are independent.

Lemma 2.2.3. *Let $Q \subset \mathscr{Q}(G_1, G_2)$ be a set of PSSAs. If Q is mutually independent, then there exists an optimal pair of exemplars with which every PSSA in Q agrees.*

Proof. Let G_1^* and G_2^* be any optimal pair of exemplars. Suppose that there exists one PSSA $p = \langle (a^1, b^1), (a^2, b^2)\rangle \in Q$ that does not agree with G_1^* and G_2^*; otherwise the lemma is proved. Without loss of generality, we assume that $\langle a^1, a^2\rangle$ appears in G_1^* and G_2^*, but $\langle b^1, b^2\rangle$ does not. As we did in the proof of Lemma 2.2.2, now we remove the pair of genes $\langle b^3, b^4\rangle$ that come from the same gene family as $\langle b^1, b^2\rangle$ in G_1^* and G_2^*, and then insert $\langle b^1, b^2\rangle$ into G_1^* and G_2^*, producing a new optimal pair of exemplars G_1' and G_2'. Clearly, p agrees with G_1' and G_2'.

We now show that those PSSAs that agree with G_1^* and G_2^* will also agree with G_1' and G_2'. To prove that, we need only consider those PSSAs affected by the removal of $\langle b^3, b^4\rangle$. If we have another PSSA $q \in Q$ that contains $\langle b^3, b^4\rangle$, say $q = \langle (b^3, x^1), (b^4, x^2)\rangle$, then $\langle x^1, x^2\rangle$ must come from the same gene family as $\langle a^1, a^2\rangle$, because p and q are independent. Since $\langle a^1, a^2\rangle$ appears in G_1^* and G_2^*, we know that $\langle x^1, x^2\rangle$ does not appear, which implies that q does not agree with G_1^* and G_2^*, but does agree with G_1' and G_2'. If we have another PSSA $q \in Q$ that contains b^3 or b^4 but not $\langle b^3, b^4\rangle$, e.g., $q = \langle (b^3, x^1), (b^5, x^2)\rangle$, then for the same reason we know that $\langle x^1, x^2\rangle$ comes from the same gene family as $\langle a^1, a^2\rangle$, which implies that q agrees with G_1^* and G_2^* and also agrees with G_1' and G_2'. Thus, comparing with G_1^* and G_2^*, we have more PSSAs agreeing with G_1' and G_2'. We can repeat this procedure and end up with an optimal pair of exemplars with which all PSSAs in Q agree. □

Hence, in order to add simultaneous inference constraints to the ILP, we need to find mutually independent PSSAs. We thus focus on this optimization problem: given $\mathscr{Q}(G_1, G_2)$, compute a mutually independent subset $Q \subset \mathscr{Q}(G_1, G_2)$ of maximum cardinality. We now give an efficient algorithm for this problem by reducing it to a maximum-weight matching problem.

We build the graph $G = (V, E)$ as follows. For each gene family in $\mathscr{A}(G_1) \cup \mathscr{A}(G_2)$, we add one vertex to V. For each PSSA $p \in \mathscr{Q}(G_1, G_2)$, we check whether there is already an edge between the two vertices corresponding to the two gene families in $\mathscr{A}(p)$. If such an edge already exists, we increase its weight by 1; otherwise, we create it and set its weight to 1. Clearly, two PSSAs in $\mathscr{Q}(G_1, G_2)$ are not independent if and only if their corresponding edges share exactly one vertex in G. Thus, the mutually independent subset $Q \subset \mathscr{Q}(G_1, G_2)$ with maximum cardinality corresponds to the matching in G with maximum total weight (see Figure 2.2 for an example). We can use Edmonds's algorithm to compute the maximum-weight matching in G, which gives us the mutually independent subset $Q \subset \mathscr{Q}(G_1, G_2)$ of maximum cardinality. We can then safely add these inference constraints to the ILP for all PSSAs in Q simultaneously. The efficiency of these constraints is studied in Section 2.3.

2.2.3 Identifying Optimal Substructures

We say n consecutive genes $(a_1, a_2, \cdots, a_n)$ on one chromosome form a *segment* of length n. Notice that $(a_1, a_2, \cdots, a_n)$ and $(-a_n, \cdots, -a_2, -a_1)$ are the same segment. For a segment t of length n, we use $\mathscr{A}(t)$ to denote the set of gene families among these genes in t, and we say t is *simple* if we have $|\mathscr{A}(t)| = n$. Given two genomes G_1 and G_2, we say two simple segments $(a_1, a_2, \cdots, a_n) \in G_1$ and $(b_1, b_2, \cdots, b_n) \in G_2$ form a *pair of shared simple segments* (PSSS for short), written as $\langle (a_1, a_2, \cdots, a_n), (b_1, b_2, \cdots, b_n) \rangle$, if a_i and b_i have the same sign and come from the same gene family for all $1 \le i \le n$, or a_i and b_{n-i+1} have the opposite sign and come from the same gene family for all $1 \le i \le n$ (see Example 2.2.2). Intuitively, those PSSSs between two genomes are more likely to stay in the optimal exemplars, since each PSSS of length n can contribute $(n-1)$ PSSAs to the exemplars. In this section we give a sufficient condition for a given PSSS to be contained in some optimal exemplars, and devise an algorithm to test it.

Let $p = \langle (a_1, a_2, \cdots, a_n), (b_1, b_2, \cdots, b_n) \rangle$ be a PSSS between two genomes G_1 and G_2. We say a PSSPA $\langle [x^1, y^1], [x^2, y^2] \rangle$ *spans* p if $(a_1, \cdots, a_n)$ is between x^1 and y^1, or $(b_1, \cdots, b_n)$ is between x^2 and y^2, or both. We denote by $S_1(p) \subset \mathscr{P}(G_1, G_2)$ the set of all PSSPAs that span p. Let $S_2(p)$ be the set of PSSPAs that consist of at least one gene pair in p, i.e., those PSS-

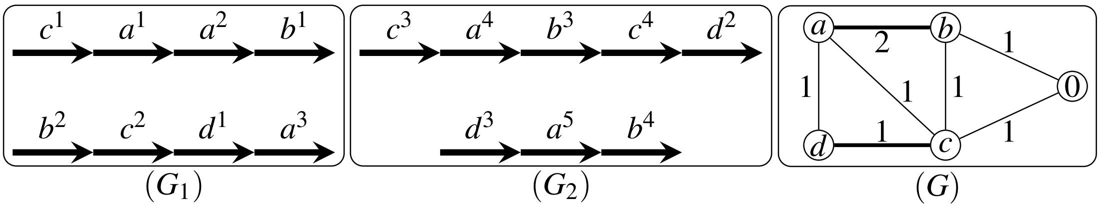


Figure 2.2 – $\mathscr{Q}(G_1, G_2) = \{\langle (\tau, c^1), (\tau, c^3) \rangle, \langle (c^1, a^1), (c^3, a^4) \rangle, \langle (a^2, b^1), (a^4, b^3) \rangle, \langle (a^2, b^1), (a^5, b^4) \rangle, \langle (b^1, \tau), (b^4, \tau) \rangle, \langle (b^2, c^2), (b^3, c^4) \rangle, \langle (c^2, d^1), (c^4, d^2) \rangle, \langle (d^1, a^3), (d^3, a^5) \rangle\}$. G is the corresponding graph, in which one maximum-weight matching is shown as bold edges.

PAs $\langle[x^1, y^1], [x^2, y^2]\rangle$ satisfying $\langle x^1, x^2\rangle = \langle a_i, b_i\rangle$, or $\langle y^1, y^2\rangle = \langle a_j, b_j\rangle$, or both, for some $1 \le i \ne j \le n$. Let $S_3(p)$ be the set of PSSPAs that consist of at least one gene pair from gene family in $\mathscr{A}(p)$, i.e., those PSSPAs $\langle[x^1, y^1], [x^2, y^2]\rangle$ satisfying that the gene family of $\langle x^1, x^2\rangle$ or of $\langle y^1, y^2\rangle$, or of both, comes from $\mathscr{A}(p)$. Clearly, we have $S_2(p) \subset S_3(p)$. Set $S(p) = S_1(p) \cup S_3(p) \setminus S_2(p)$ and let $m(p)$ be the maximum number of PSSPAs in $S(p)$ that can coexist w.r.t. G_1 and G_2 (see Example 2.2.2). Intuitively, $m(p)$ is the maximum number of PSSAs that can be destroyed by the appearance of p. Thus, if we have $m(p) \le n-1$, where $n-1$ is the number of PSSAs inside p, then some optimal solution must include p. Formally, we have the following lemma.

Lemma 2.2.4. *Let p be a PSSS with n gene pairs between G_1 and G_2 and assume $m(p) \le n-1$; then there exists an optimal pair of exemplars of G_1 and G_2 that contains p.*

Proof. We prove the lemma by contradiction. Write $p = \langle(a_1, a_2, \cdots, a_n), (b_1, b_2, \cdots, b_n)\rangle$ and let G_1^* and G_2^* be an optimal pair of exemplars of G_1 and G_2 that does not contain p. We can assume that G_1^* and G_2^* do not contain any pair of genes in p, i.e., do not contain $\langle a_i, b_i\rangle$ for any $1 \le i \le n$, since otherwise, we can iteratively apply Lemma 2.2.2 to build another optimal pair of exemplars that include all pairs of genes in p, which contradicts the assumption.

Now we modify G_1^* and G_2^* to create a new pair of exemplars of G_1' and G_2'. We first remove those n gene pairs coming from the gene families in $\mathscr{A}(p)$. Exactly those PSSAs between G_1^* and G_2^* that contain at least one gene pair with gene family in $\mathscr{A}(p)$ are destroyed. We denote by P_3 those PSSAs that are destroyed in this step. Second, we insert p back to G_1^* and G_2^*. This step can destroy at most two PSSAs between G_1^* and G_2^*, since the insertion of two segments (one in each genome) will break only two adjacencies. We denote by P_1 the set of PSSAs that are destroyed in this step.

Let I_3 and I_1 be the sets of the indicators of P_3 and P_1, respectively. According to the construction, we have $I_3 \subset S_3(p) \setminus S_2(p)$ and $I_1 \subset S_1(p)$. Moreover, these PSSPAs in $I_3 \cup I_1$ can coexist w.r.t. G_1 and G_2, since P_3 and P_1 can appear in a pair of exemplars, G_1^* and G_2^*, simultaneously. Using the condition, we get $|P_3 \cup P_1| = |I_3 \cup I_1| \le n-1$, which means that, during the modification, at most $(n-1)$ PSSAs are destroyed. On the other hand, the insertion of p introduces $(n-1)$ new PSSAs to G_1' and G_2'. Thus, we can write $s(G_1', G_2') \ge s(G_1^*, G_2^*)$, which contradicts the assumption. □

We now give an algorithm to compute an upper bound for $m(p)$. We create a new pair of genomes $G_1(p)$ and $G_2(p)$ by keeping only the genes appearing in some PSSPA in $S(p)$ and removing rest of the genes in G_1 and G_2. Those PSSPAs in $S(p)$ that can coexist w.r.t. G_1 and G_2 can also coexist w.r.t. $G_1(p)$ and $G_2(p)$, so that the maximum number of PSSPAs in $S(p)$ that can coexist w.r.t. $G_1(p)$ and $G_2(p)$ is an upper bound for $m(p)$. Thus, we can apply the algorithm described in Section 2.2.1 to compute the upper bound: in the ILP formulation, we just need to replace G_1, G_2, and $\mathscr{P}(G_1, G_2)$ by $G_1(p)$, $G_2(p)$, and $S(p)$, respectively, and the optimal objective value of this ILP will give us an upper bound for $m(p)$. If the upper bound is no larger than $(n-1)$, then according to Lemma 2.2.4, we know that p is contained in some optimal solution. We can then safely fix p by removing other genes from gene families in $\mathscr{A}(p)$ except those in p.

Example 2.2.2. Consider two genomes $G_1 = (\tau, -d^1, a^1, b^1, c^1, e^1, c^2, -f^1, b^2, a^2, \tau)$ and $G_2 = (\tau, -b^3, a^3, b^4, c^2, -e^2, -b^5, c^3, d^2, f^2, \tau)$. Let $p = \langle (a^1, b^1, c^1), (a^3, b^4, c^2) \rangle$ be a PSSS between G_1 and G_2. We have $S_1(p) = \{\langle(-d^1, e^1), (-e^2, d^2)\rangle\}$ and $S_3(p) \backslash S_2(p) = \{\langle(-d^1, b^1), (-b^5, d^2)\rangle, \langle(b^1, e^1), (-e^2, -b^5)\rangle, \langle(b^2, \tau), (\tau, -b^3)\rangle\}$. We then create $G_1(p)$ and $G_2(p)$ based on $S(p)$, obtaining $G_1(p) = (-d^1, b^1, e^1, b^2, \tau)$ and $G_2(p) = (\tau, -b^3, -e^2, -b^5, d^2)$. There are at most two PSSPAs in $S(p)$ that can coexist w.r.t. $G_1(p)$ and $G_2(p)$, i.e., $m(p) \le 2$. Thus, p is contained in some optimal exemplars of G_1 and G_2. We can then simplify G_1 and G_2 to $G_1' = (\tau, -d^1, a^1, b^1, c^1, e^1, -f^1, \tau)$ and $G_2' = (\tau, a^3, b^4, c^2, -e^2, d^2, f^2, \tau)$.

For an arbitrary PSSS p, the cardinality of $S(p)$ is usually small, since it consists of only the PSSPAs related with the gene families in $\mathscr{A}(p)$ and those spanning p. Thus, we expect the corresponding ILP instance to be small and efficiently solvable. The preprocessing phase applies this algorithm iteratively. At each iteration, we list all possible *maximal* PSSSs, i.e., those PSSSs that cannot be extended into longer ones. For each of these maximal PSSSs, we use the above algorithm to test whether it is contained in some optimal solution. If it is, then we fix it and start a new iteration. If the test fails for all maximal PSSSs, then the preprocessing phase terminates. The efficacy of this procedure is studied in Section 2.3.

2.3 Simulation Results

We simulate a pair of genomes as follows. We start from an ancestor genome with only one linear chromosome consisting of N gene families, each of which contains exactly one gene. The two extant genomes then evolve independently from this ancestor genome. The evolution process on each branch includes a series of genome-scale events: inversions, which occur with probability p, and segmental duplications, which occur with probability $(1-p)$. An inversion randomly chooses two positions on the same chromosome and then reverses the segment in-between. A segmental duplication randomly chooses a segment of length L and insert its copy to another random position. We make sure that the expected number of genes per gene family is two in each genome—so that the number of events on each branch is $N/((1-p)\cdot L)$. Thus, a simulation configuration is determined by the triple (N, p, L).

First, we evaluate the efficacy of the two features (preprocessing and additional constraints) in our algorithm. The preprocessing procedure to identify and fix the optimal substructures (Section 2.2.3) is referred to as feature 1, while the procedure to add the inference constraints (Section 2.2.2) is referred to as feature 2. We denote by ALGO0 the basic ILP formulation, denote by ALGO1 (resp. ALGO2) the basic ALGO0 with feature 1 (resp. feature 2), and denote by ALGO3 the basic ALGO0 with both features. We use the configuration $(1000, p, L)$ to evaluate the 4 algorithms, where p ranges from 0 to 0.9 and $L \in \{1, 5, 10\}$. The results are shown in Table 2.1. Observe that, as L increases, all versions of the program take longer, because the difficulty of the problem is closely related to the density of the shared adjacencies in each genome. With

$L = 1$, each duplication copies a single gene, thus adjacencies are rarely copied; the larger L is, the more shared adjacencies appear in each genome. Similarly, observe that, as p increases, all versions of the program use less time, because the increased number of inversions destroy many shared adjacencies. While each of the features improves the performance (with feature 2 bringing more significant improvements than feature 1, especially for small p), the two features are synergistic, as their combination gains more than the sum of the individual gains. Indeed, when both features are used, the resulting algorithm is very fast, finishing all instances in a very short time.

Next, we compare our algorithm ALGO3 with the divide-and-conquer algorithm [38], referred to here as ALGOX. The results are shown in Table 2.2. ALGO3 runs very fast for small L and large p. Moreover, for those instances with larger L ($L = 5$) and smaller p ($p \leq 0.3$), ALGO3 still finishes in a very short time (roughly one minute), even for genomes with 10000 genes. ALGOX, however, can finish in 20 minutes only when both N and p are small—in this case the number of shared adjacencies in the optimal exemplars is close to N, so that ALGOX can cut off most of the branches in the search process.

2.4 Application to Orthology Assignment

In this section, we study the maximum adjacency problem with the intermediate strategy: given two genomes, select the same number of genes in each gene family in each genome and compute a bijection between the selected genes in the same gene family, such that the number of shared adjacencies between the resulting two genomes induced by this bijection is maximized. Notice that this problem is different from the exemplar breakpoint distance

p	$L = 1$				$L = 5$				$L = 10$			
0.0	0	0	0	0	(0)	22(6)	169	8	(0)	145(4)	181(9)	107
0.1	0	0	0	0	(0)	39(6)	60	12	(0)	87(2)	233(8)	237
0.2	0	0	0	0	(0)	68	44	6	(0)	489(2)	184(8)	201
0.3	0	0	0	0	688(4)	62	45	4	(0)	268(2)	146(8)	189
0.4	0	0	0	0	271(8)	8	40	1	(0)	622(6)	130(8)	368
0.5	0	0	0	0	34	1	11	1	(0)	156(9)	88(9)	110
0.6	0	0	0	0	9	0	6	0	(0)	134	89(9)	36
0.7	0	0	0	0	0	0	0	0	7	8	102	6
0.8	0	0	0	0	0	0	0	0	20	1	9	1
0.9	0	0	0	0	0	0	0	0	0	0	0	0

Table 2.1 – Comparison of the four algorithms. For each value of L, the four columns correspond to ALGO0, ALGO1, ALGO2 and ALGO3, respectively. For each configuration, we generate and solve 10 independent instances, and compute the average running time (in seconds) for the instances that are finished in 20 minutes. If the number of instances that are finished in 20 minutes is less that 10, we put this number in a parentheses. Programs are run on a 64-core (2.3GHz) machine with 512GB memory.

with the intermediate strategy, since in the intermediate situation the sum of breakpoints and shared adjacencies is not a constant.

We propose a new algorithm, which we denote by IALGO3, to solve this problem. The algorithm works in two steps. In the first step, the algorithm starts with the two given genomes G_1 and G_2 and uses ALGO3 iteratively to compute an optimal pair of exemplars between the two current genomes; each pair of genes in the exemplars is assigned to a distinct new gene family of its own, and the updated genomes are returned. Successive iterations will decrease the size of gene families and produce new exemplar pairs. This step ends when two consecutive iterations give the same exemplars. In the final exemplars, there are $\min\{F(G_1, f), F(G_2, f)\}$ genes in each genome for each gene family $f \in \widetilde{\mathscr{A}}(G_1) \cap \widetilde{\mathscr{A}}(G_2)$. The second step then uses a local improvement strategy to optimize the objective, by iteratively examining all the gene pairs in the bijection specified by the exemplars and removing those pairs whose removal can increase the number of shared adjacencies.

The bijection infers a subset of the orthologs between the two given genomes under a parsimonious evolutionary model. We compare the results of IALGO3 with another method for orthology assignment based on position, MSOAR [40]. MSOAR uses several heuristics to build a bijection between genes in the same gene family such that the inversion distance induced by the bijection is minimized. We apply both methods to five well-annotated species, human, gorilla, orangutan, mouse, and rat. The datasets were downloaded from Ensembl (`http://www.ensembl.org`). For each species, we collected the sequences and positions on the chromosomes for all the protein-coding genes; in case a gene had multiple alternative products, we kept its longest isoform. We follow the pipeline of MSOAR to generate gene families: we first run the all-versus-all BLASTp comparison, and then build a graph with all genes as vertices and link two genes if they are among each other's top five bidirectional best hits; we finally take all the genes in a connected component as a gene family.

We do the pairwise comparison for all five species. For each pair of species, we run both methods to obtain two bijections. We then evaluate each bijection as follows. We call a pair

N	$L = 1$						$L = 5$					
	$p = 0.1$		$p = 0.3$		$p = 0.5$		$p = 0.1$		$p = 0.3$		$p = 0.5$	
100	0	28(3)	0	(0)	0	(0)	0	0	0	278(6)	0	(0)
200	0	(0)	0	(0)	0	(0)	2	2	0	(0)	0	(0)
500	0	(0)	0	(0)	0	(0)	4	68(2)	5	(0)	0	(0)
1000	0	(0)	0	(0)	0	(0)	7	(0)	11	(0)	1	(0)
2000	0	(0)	0	(0)	0	(0)	9	(0)	14	(0)	2	(0)
5000	0	(0)	0	(0)	0	(0)	24	(0)	30	(0)	12	(0)
10000	1	(0)	0	(0)	0	(0)	56	(0)	75	(0)	18	(0)

Table 2.2 – Comparison of ALGO3 and ALGOX. For each configuration, the two columns give the performance of ALGO3 and ALGOX, respectively. (The experimental setup is the same as that in Table 2.1.) The command line for the ALGOX program uses "`-ebp -fastED -delete`".

in a bijection *trivial* if each of the two genes in this pair forms a singleton gene family. We focus on nontrivial gene pairs in each bijection. To assess them, we downloaded the gene symbols (HGNC symbols for the primate genes, MGI symbols for mouse genes, and RGD symbols for rat genes) from Ensembl as the reference criterion. For a nontrivial gene pair a^1 and a^2 with symbols s_1 and s_2 respectively, we say it is *assessable* if there exists one gene (which could be a^2) in G_2 that has symbol s_1, or there exists one gene (which could be a^1) in G_1 that has symbol s_2, or both. We say this pair is assigned correctly, if there exists one gene (which could be a^2) with gene symbol s_1 in G_2 that is in the same tandemly arrayed genes with a^2, or there exists one gene (which could be a^1) with gene symbol s_2 in G_1 that is in the same tandemly arrayed genes with a^1, or both. The *accuracy* of a bijection is then defined as the ratio between the number of correctly assigned pairs and the number of assessable pairs.

The results are shown in Table 2.3. More than half of the gene pairs in the two bijections are nontrivial—duplicate genes are very common in these five species. Both methods reach very high accuracy, with IALGO3 doing slightly better than MSOAR—but IALGO3 ran very much faster (by at least two orders of magnitude) than MSOAR. We remark that the first step of the iteration in IALGO3 is to run ALGO3 on the two given genomes. Therefore, the running time of IALGO3 showed in Table 2.3 is an upper bound on the time to compute the optimal exemplars.

species pairs	gene pairs		nontrivial		assessable		accuracy		time	
gorilla & human	16457	16525	8689	8599	7812	7714	**98.2**%	98.1%	12	1472
gorilla & mouse	15381	15535	8266	8278	7030	7019	**98.1**%	97.9%	9	3509
gorilla & orangutan	15211	15368	7803	7804	5733	5698	**98.1**%	**98.1**%	9	1437
gorilla & rat	15468	15611	8663	8663	6098	6078	**96.3**%	96.0%	14	6964
human & mouse	15814	15868	8266	8185	7325	7267	**98.6**%	98.3%	10	2912
human & orangutan	15157	15245	7651	7609	5930	5857	**98.8**%	98.6%	6	875
human & rat	15749	15802	8613	8531	6433	6404	**96.4**%	96.3%	15	4846
mouse & orangutan	14437	14509	7429	7397	5390	5374	**98.3**%	**98.3**%	8	1921
mouse & rat	17722	17862	9772	9793	6759	6768	96.7%	**96.8**%	23	5360
rat & orangutan	14465	14559	7754	7740	4616	4617	**96.7**%	96.5%	9	4261

Table 2.3 – Pairwise comparison on five species. For each category, the first column shows the numbers for IALGO3 while the second column shows the numbers for MSOAR. The categories of *gene pairs, nontrivial, assessable* give the total number of gene pairs in the bijections, the number of nontrivial gene pairs, and the number of assessable gene pairs, respectively. The last category gives the running time (in seconds) of the two methods measured on a 6-core (3.5GHz) machine with 32GB memory.

2.5 Discussion

We remark that the high speed and good scaling properties of our algorithms make them practical for research on large-scale genomic evolution, but also for improved orthology assignment, as the exemplar concept finds broad applicability in comparative genomics.

The performance of our algorithms can be further improved. We expect additional structure can be discovered and turned into constraints for the ILP formulation, thereby reducing the search space for the ILP solver. We are also studying the use of a set of PSSSs (rather than a single PSSS) to define candidates for fixing in the optimal substructure, because it is possible that several PSSSs as a group pass the test, while any single one of these PSSSs fails.

3 DCJ Distance with Duplicate Genes

In this chapter, we give the first algorithm to compute the DCJ distance with the maximum matching strategy in the presence of duplicate genes. To achieve this, we first reduce this problem to the problem of finding the optimal consistent decomposition of the corresponding adjacency graph, then formulate the latter problem as an integer linear program (ILP). We also provide an efficient preprocessing approach to reduce the ILP formulation while preserving optimality. Finally, we apply our method to assign orthologs and also compare its performance with MSOAR on both simulated and biological datasets.

3.1 Problem Statement

In this chapter, we study two genomes with the same gene content: each gene family has the same number of genes in both genomes. We say a bijection between G_1 and G_2 is *valid* if it specifies n homologous gene pairs, where n is the number of genes in each genome. If G_1 and G_2 contain only singleton gene families (exactly one gene in each family in each genome), then there is a unique valid bijection between G_1 and G_2, and the DCJ distance between G_1 and G_2 can be computed in linear time [11]. If G_1 and G_2 contain gene families with multiple genes in each genome, then there are many valid bijections between G_1 and G_2. Different valid bijections define different one-to-one correspondences between homologous genes, yielding possibly different DCJ distances between G_1 and G_2. In this chapter, we study the DCJ distance problem with the maximum matching strategy: given two genomes G_1 and G_2 with the same gene content, find a valid bijection between G_1 and G_2 that minimizes the DCJ distance induced by this bijection. We denote the DCJ distance with the maximum matching strategy between G_1 and G_2 as $d(G_1, G_2)$. This problem is NP-hard, which can be proved by a reduction from the NP-hard problem of *breakpoint graph decomposition* [59].

We now introduce the data structure of adjacency graph, which plays a central role in computing the DCJ distance. We first introduce two more notations. If genes a and b are homologous, we say its corresponding extremities (a_h and b_h, a_t and b_t) are also *homologous.* The set of all extremities of a genome is called *extremity set.* Let G_1 and G_2 be two genomes with the same gene content, and let S_1 and S_2 be the extremity sets of G_1 and G_2, respectively. The *adjacency graph* with respect to G_1 and G_2 can be written as $AG = (V, E)$, where $V = S_1 \cup S_2$ and E is composed of two types of edges, *black edges* and *gray edges*. Two extremities in different extremity sets (one is in S_1 and the other one is in S_2) are connected by one black edge if they are homologous, and two extremities in the same extremity set are connected by one gray edge if they form an existing adjacency. Figure 3.1**(a)** gives an example.

We say that a cycle (or path) in the adjacency graph is *alternating* if any two adjacent edges in this cycle (or path) consist of one black edge and one gray edge. The *length* of a cycle (or path) is defined as the number of its black edges. A *decomposition* of the adjacency graph is a set of vertex-disjoint alternating cycles and paths that cover all vertices and all gray edges. We say a decomposition is *consistent* if for any two homologous genes a and b, either both (a_h, b_h) and (a_t, b_t) are in this decomposition, or neither of them is in this decomposition. Figure 3.1**(b)** and 3.1**(c)** give two examples of consistent decompositions.

Given two genomes G_1 and G_2 with the same gene content, there is a natural one-to-one correspondence between the set of all possible valid bijections from G_1 to G_2 and the set of all possible consistent decompositions of the adjacency graph. In fact, if one valid bijection is given, which maps gene a in G_1 to a homologous gene b in G_2, then we can keep the black edges (a_h, b_h) and (a_t, b_t) in the decomposition. We do the same thing for every pair of genes specified by this valid bijection; this process culminates in a consistent decomposition. On the other hand, if we are given a consistent decomposition of the corresponding adjacency graph, we can collect all homologous gene pairs (a, b) indicated by black edges (a_h, b_h) and (a_t, b_t), which form a valid bijection from G_1 to G_2. Given a consistent decomposition with c cycles and

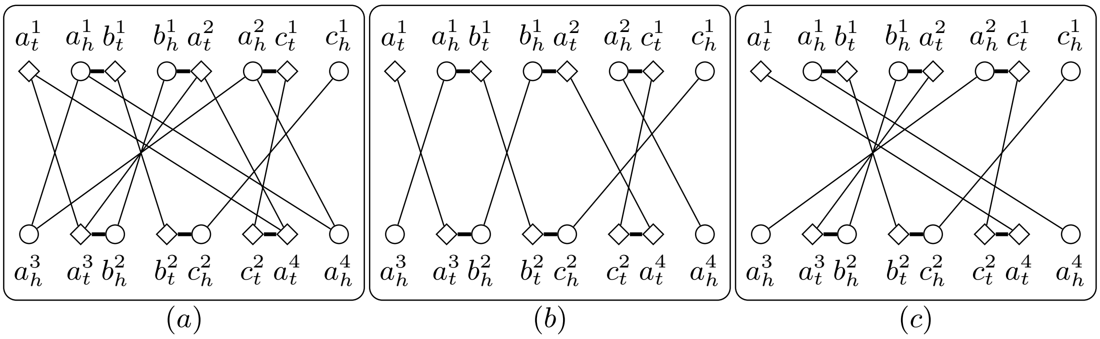

a_t^1 a_h^1 b_t^1 b_h^1 a_t^2 a_h^2 c_t^1 c_h^1
a_h^3 a_t^3 b_h^2 b_t^2 c_h^2 c_t^2 a_t^4 a_h^4
(a)
(b)
(c)

Figure 3.1 – An example of adjacency graph and its two consistent decompositions. Genome 1 contains one linear chromosome, (a^1, b^1, a^2, c^1), and genome 2 also contains one linear chromosome $(-a^3, -b^2, -c^2, a^4)$. Genes in the same gene family are represented by the same label, and distinguished by different superscripts. All black edges are represented by long thin lines, and all gray edges are represented by short thick lines. (**a**) The corresponding adjacency graph, in which head extremities are represented by circles, while tail extremities are represented by diamonds. (**b**) A consistent decomposition with 2 odd-length paths, whose corresponding valid bijection maps a^1 to a^3 and a^2 to a^4. (**c**) Another consistent decomposition with 2 odd-length paths and 1 cycle, whose corresponding valid bijection maps a^1 to a^4 and a^2 to a^3.

o odd-length paths, exactly $(|V|/4 - c - o/2)$ DCJ operations are needed to transform G_1 into G_2 [11]. Thus, we can write $d(G_1, G_2) = \min_{D\in\mathscr{D}}(|V|/4 - c_D - o_D/2) = |V|/4 - \max_{D\in\mathscr{D}}(c_D + o_D/2)$, where $\mathscr{D}$ is the space of all consistent decompositions, and c_D and o_D are the numbers of cycles and odd-length paths in a decomposition D, respectively. This formula transforms our DCJ distance problem into the *maximum cycle decomposition problem,* which asks for a consistent decomposition of the adjacency graph such that the number of cycles plus half the number of odd-length paths in this decomposition is maximized.

3.2 ILP for the Maximum Cycle Decomposition Problem

Shao *et al.* described a capping method to remove telomeres by introducing *null extremities* [54]. All null extremities are homologous to each other, but none is homologous to any other extremity. Let $AG = (V = S_1 \cup S_2, E)$ be the adjacency graph with respect to two given genomes G_1 and G_2. Suppose that G_1 and G_2 contain $2 \cdot k_1$ and $2 \cdot k_2$ telomeres respectively. The “telomere removal” proceeds as follows (see Figure 3.2 for an example). For each extremity $u \in S_1$ coming from each telomere in G_1, we add one null extremity τ to S_1 and add one gray edge to E that connects u and τ. Similarly, for each extremity $v \in S_2$ coming from each telomere in G_2, we add one null extremity τ to S_2 and add one gray edge to E that connects v and τ. If we additionally have $k_1 < k_2$, we then add $(k_2 - k_1)$ pairs of null extremities to S_1, each of which is connected by one more gray edge added to E. We finally add black edges connecting all possible pairs of null extremities between S_1 and S_2. We can prove that this telomere removal process does not change $d(G_1, G_2)$ using the same argument as in [54]. In the following we assume that each vertex is adjacent to exactly one gray edge in the adjacency graph, and that the consistent decompositions consist of only cycles.

Now we formulate the maximum cycle decomposition problem as an integer linear program. Let $AG = (V, E)$ be the adjacency graph with respect to two given genomes G_1 and G_2 with the same gene content. For each edge $e \in E$, we create binary variable x_e to indicate whether

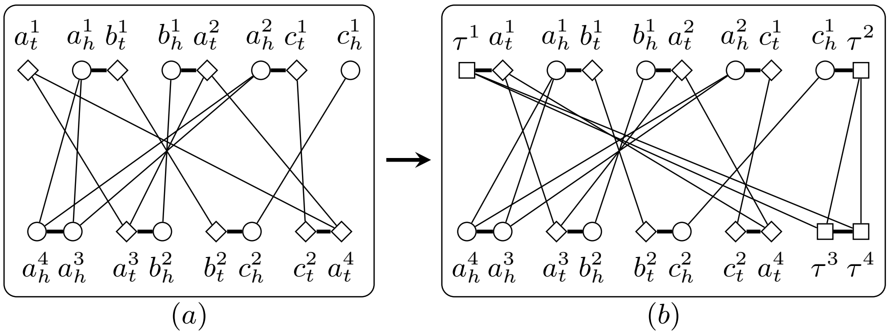

a_t^1 a_h^1 b_t^1 b_h^1 a_t^2 a_h^2 c_t^1 c_h^1
a_h^4 a_h^3 a_t^3 b_h^2 b_t^2 c_h^2 c_t^2 a_t^4
(a)
τ^1 a_t^1 a_h^1 b_t^1 b_h^1 a_t^2 a_h^2 c_t^1 c_h^1 τ^2
a_h^4 a_h^3 a_t^3 b_h^2 b_t^2 c_h^2 c_t^2 a_t^4 τ^3 τ^4
(b)

Figure 3.2 – An example of the telomere removal. Genome 1 contains one linear chromosome (a^1, b^1, a^2, c^1), and genome 2 contains one circular chromosome $(-a^3, -b^2, -c^2, a^4)$. (**a**) The corresponding adjacency graph. (**b**) The adjacency graph after telomere removal, in which null extremities are represented by squares.

e will be in the final decomposition. First, we require that all gray edges be in the final decomposition:

$$x_e = 1, \quad \forall e \text{ that are gray.}$$

Second, we require that the final decomposition be consistent:

$$x_{(a_h,b_h)} = x_{(a_t,b_t)}, \quad \forall a \in G_1 \text{ and } \forall b \in G_2 \text{ that are homologous.}$$

Third, we require that for each vertex exactly one adjacent black edge adjacent be chosen:

$$\begin{aligned} \sum_{(u,v)\in E,\ v\in S_2} x_{(u,v)} &= 1, \quad \forall u \in S_1; \\ \sum_{(u,v)\in E,\ u\in S_1} x_{(u,v)} &= 1, \quad \forall v \in S_2. \end{aligned}$$

Clearly, these three groups of constraints together guarantee that all selected edges form a consistent decomposition.

Now we count the number of cycles. We first index the vertices arbitrarily, $V = \{v_1, v_2, \cdots, v_{|V|}\}$. For each vertex v_i, we create variable y_i to indicate the *label* of v_i. We set a distinct positive bound i for each y_i:

$$0 \le y_i \le i, \quad 1 \le i \le |V|.$$

We require that all vertices in the same cycle in the final decomposition have the same label, which can be guaranteed by requiring that, for each selected edge, the two adjacent vertices have the same label:

$$\begin{aligned} &y_i \le y_j + i \cdot (1 - x_e), \quad \forall e = (v_i, v_j) \in E; \\ &y_j \le y_i + j \cdot (1 - x_e), \quad \forall e = (v_i, v_j) \in E. \end{aligned}$$

Then, for each vertex v_i, we create binary variable z_i to indicate whether y_i is equal to its upper bound i:

$$i \cdot z_i \le y_i, \quad 1 \le i \le |V|.$$

Since all vertices in the same cycle have the same label and all upper bounds are distinct, there is exactly one vertex in each cycle whose label can be equal to its upper bound. Finally, we set the objective to

$$\max \sum_{1 \le i \le |V|} z_i,$$

which is equal to the number of cycles.

There are $O(|E|)$ variables and $O(|E|)$ constraints in this ILP formulation.

3.3 Fixing Cycles of Length Two

A cycle of length two in the adjacency graph indicates one shared adjacency. The following theorem gives a sufficient condition to fix this cycle while preserving optimality, which can be used to narrow the search for an optimal bijection.

Theorem 3.3.1. *Given an adjacency graph $AG = (V, E)$, if a length-two cycle C contains some vertex with total degree 2, then there exists an optimal consistent decomposition of AG that contains C.*

Proof. Let $\{a_h^1, b_h^1, a_h^2, b_h^2\}$ be the four vertices of C, where a_h^1 and b_h^1 form an adjacency in G_1 while a_h^2 and b_h^2 form an adjacency in G_2, and (a_h^1, a_h^2) and (b_h^1, b_h^2) are the two black edges of C. Let a_h^1 be the vertex of total degree 2; then $\{a^1\}$ (resp. $\{a^2\}$) forms a singleton gene family in G_1 (resp. G_2), and thus edge (a_h^1, a_h^2) appears in every consistent decomposition. Now we prove the theorem by contradiction. Suppose that edge (b_h^1, b_h^2) is not in any optimal consistent decomposition. Take any optimal consistent decomposition D, in which b_h^1 is linked to b_h^4 and b_h^2 is linked to b_h^3. Since D is consistent, we know that edges (b_t^1, b_t^4) and (b_t^2, b_t^3) are also in D. We now transform D into a new decomposition D'' that contains edge (b_h^1, b_h^2) by exchanging two pairs of edges. Figure 3.3 illustrates this process. First, we remove edges (b_h^1, b_h^4) and

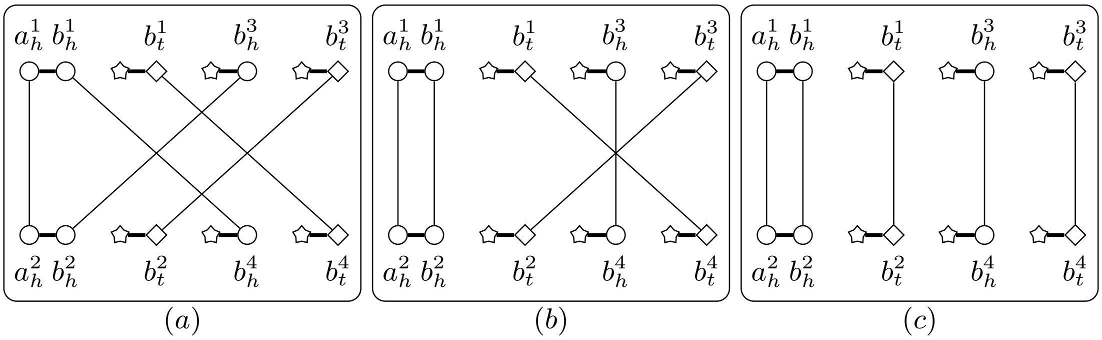


Figure 3.3 – The process of building a new optimal consistent decomposition that contains edge (b_h^1, b_h^2). (**a**) One optimal consistent decomposition D without edge (b_h^1, b_h^3). Star represent unrelated extremities. (**b**) The inconsistent decomposition D'. (**c**) The consistent decomposition D''.

(b_h^3, b_h^2) from D and add edges (b_h^1, b_h^2) and (b_h^3, b_h^4); denote this inconsistent decomposition by D'. Since in this step one cycle is split into two small cycles, we have that $c_{D'} = c_D + 1$. Now, we remove edges (b_t^1, b_t^4) and (b_t^3, b_t^2) from D' and add edges (b_t^1, b_t^2) and (b_t^3, b_t^4) to obtain the consistent decomposition D''. This step involves at most two cycles of D', and merges these two cycles together in the worst case. Thus, we have $c_{D''} \geq c_{D'} - 1$. Overall, we have that $c_{D''} \geq c_D$, which means D'' is also an optimal consistent decomposition—the desired contradiction. □

If all four vertices in a cycle of length two have degree larger than 2, then it is possible that this cycle is not part of any optimal consistent decomposition. Figure 3.4 gives such an example. Moreover, this example also shows that if a shared adjacency appears exactly once in each genome, it is still possible that the corresponding cycle of length two is not part of any optimal consistent decomposition.

3.4 Experimental Results

We compare our method with MSOAR on both simulated and biological datasets. The input for both methods is two genomes with the same gene content, and the output is a bijection between the two genomes, plus the DCJ distance calculated as $(n - c - o/2)$, where n is the

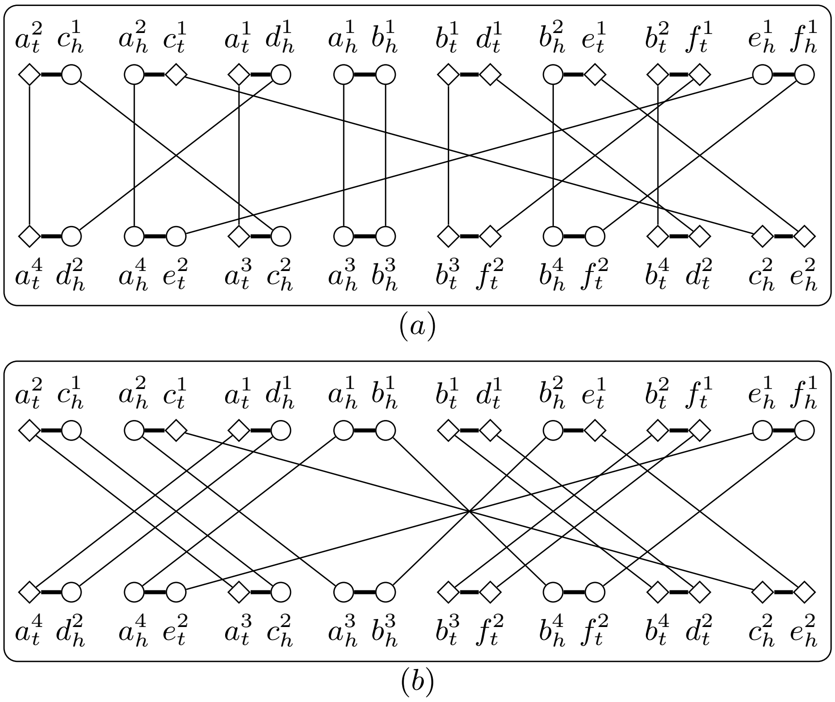


Figure 3.4 – An example of a cycle of length two that is not part of any optimal consistent decomposition. (**a**) A consistent decomposition with 4 cycles that contains the cycle of length two of $\{a_h^1, b_h^1, a_h^3, b_h^3\}$. (**b**) An optimal consistent decomposition with 5 cycles.

number of genes in each genome, and c and o are the numbers of cycles and odd-length paths in the adjacency graph induced by the bijection. We use both the accuracy of the bijection, which is defined as the percentage of correct gene pairs (compared with a reference bijection), and the deviation from the true evolutionary distances, to evaluate the performance of the two methods.

For our method, given two genomes, we first build the adjacency graph and then employ the telomere removal technique to obtain a new adjacency graph without telomeres. Then we apply Theorem 3.3.1 to fix possible cycles of length two, and finally invoke GUROBI [58] to solve the ILP formulation. Since the ILP solver might take a long time, we set a time limit of two hours for each instance in our experiments—the best solution will be returned if the ILP solver does not terminate in two hours. For MSOAR, we run its binary version downloaded from `http://msoar.cs.ucr.edu/`. We compare our method with MSOAR, rather than the latest version MSOAR 2.0, because we focus on genomes with the same gene content, which implicitly requires that, after the speciation event, only DCJ operations are involved. Compared with MSOAR, MSOAR 2.0 aims to identify tandem duplications of genes *after* the speciation. Thus, under our evolutionary model that does not contain post-speciation duplications, MSOAR and MSOAR 2.0 are equivalent.

3.4.1 Simulation Results

We simulate artificial genomes under an evolutionary model including segmental duplications and DCJs. We introduce duplicated genes through segmental duplications. For each segmental duplication, we uniformly select a position to start duplicating a segment of the genome and place the new copy to a new position. Since the average copy number of each gene in human, mouse and rat genomes, are 1.46, 1.55 and 1.28, respectively, we set the average copy number to 1.5 in our simulation. From a genome with S distinct genes, we generate an ancestor genome with $1.5 \cdot S$ genes, by randomly performing $S/(2 \cdot L)$ segmental duplications of length L (in terms of the number of genes in the segment). We then simulate two extant genomes from the ancestor by randomly performing N DCJs (in terms of inversions) independently. Thus, the true evolutionary distance between the two extant genomes is $2 \cdot N$. The reference bijection consists of those gene pairs that correspond to the same gene in the ancestor.

We first set $S = 1000$, and test four different lengths for segmental duplications ($L = 1, 2, 5, 10$). The results illustrate the trends and capabilities of the two methods in handling genomes with duplicated segments. We also vary the number of DCJs over a broad range ($N = 200, 210, \cdots, 500$) that reaches beyond the saturation point. For each combination of L and N, we randomly simulate 5 independent instances, and calculate the average accuracy of the bijection and the average deviation from the true evolutionary distances over these 5 instances.

Figure 3.5 shows the deviation from the true evolutionary distances for both methods. The first observation is that saturation starts occurring for a true evolutionary distance of 720: the DCJ distance obtained from the reference bijection is smaller than the true evolutionary distance, and the gap increases along with the increase of the true evolutionary distance. Second, when the true evolutionary distance is less than 720, our method obtains very accurate DCJ distances while MSOAR usually overestimates the DCJ distance. The difference is particularly pronounced for $L \geq 2$: in such cases, there exist identical segments in each genome, a situation that creates problems when MSOAR tries to partition each genome into a minimum number of common segments [37]. Figure 3.6 shows the the accuracy of the bijections for both methods. For $L = 1$, both methods can correctly identify most gene pairs. For $L \geq 2$, our method significantly outperforms MSOAR. For large L, the accuracy of our method decreases rapidly beyond saturation, but continues to dominate MSOAR.

We also simulate very large genomes by setting $S = 5000$. Again we test different segmental duplications ($L = 1, 2, 5, 10$) and different number of DCJs ($N = 1000, 1100, \cdots, 2000$). Figure 3.7 shows the average accuracy of the bijection. For large L and small N, our method can identify almost all correct gene pairs, while MSOAR outputs a significant portion of incorrect pairs. Similar to the case of $S = 1000$, when the true evolutionary is large, the accuracy of our method decreases quickly, but still outperforms MSOAR.

The running time of MSOAR grows slowly as the the true evolutionary distance increases. For the most complicated case of $S = 5000$, $L = 10$ and $N = 2000$, MSOAR needs roughly 1 hour to finish. Regarding our method, when the true evolutionary distance is relatively small (for example, $N \leq 320$ when $S = 1000$ and $L = 5$, $N \leq 1500$ when $S = 5000$ and $L = 5$), the preprocessing method can fix a considerable portion of the adjacency graph, leaving a

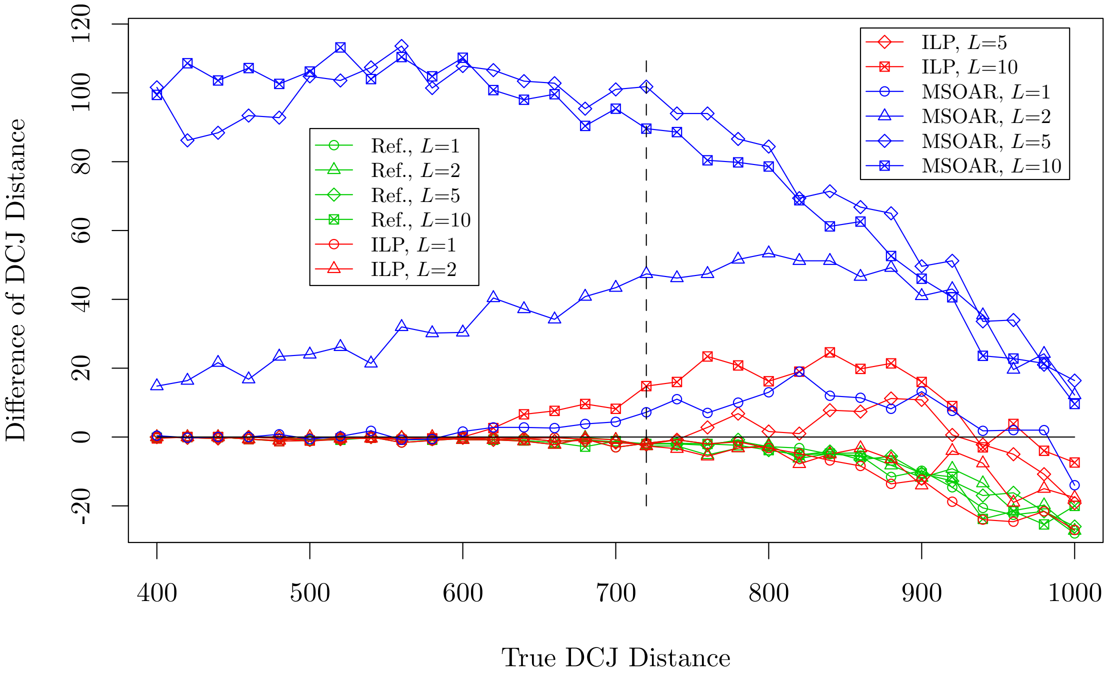


Figure 3.5 – Deviation from the true evolutionary distances for $S = 1000$. Green lines track reference bijection, red lines track our method, and blue lines track MSOAR.

small ILP instance that can be solved very quickly (it takes only a few seconds and is faster than MSOAR). When the true evolutionary distance is relatively large, the ILP solver cannot terminate in two hours and a sub-optimal solution is obtained. Usually, this solution is equal

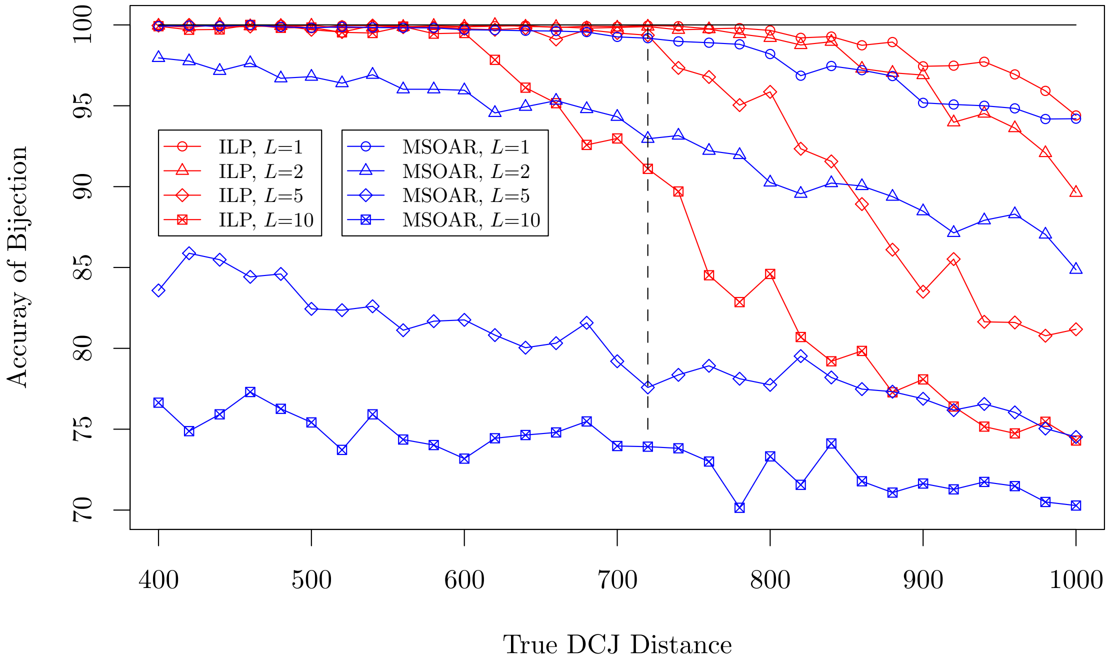


Figure 3.6 – The accuracy of the bijections for $S = 1000$. Red lines circles track our method, while blue lines track MSOAR.

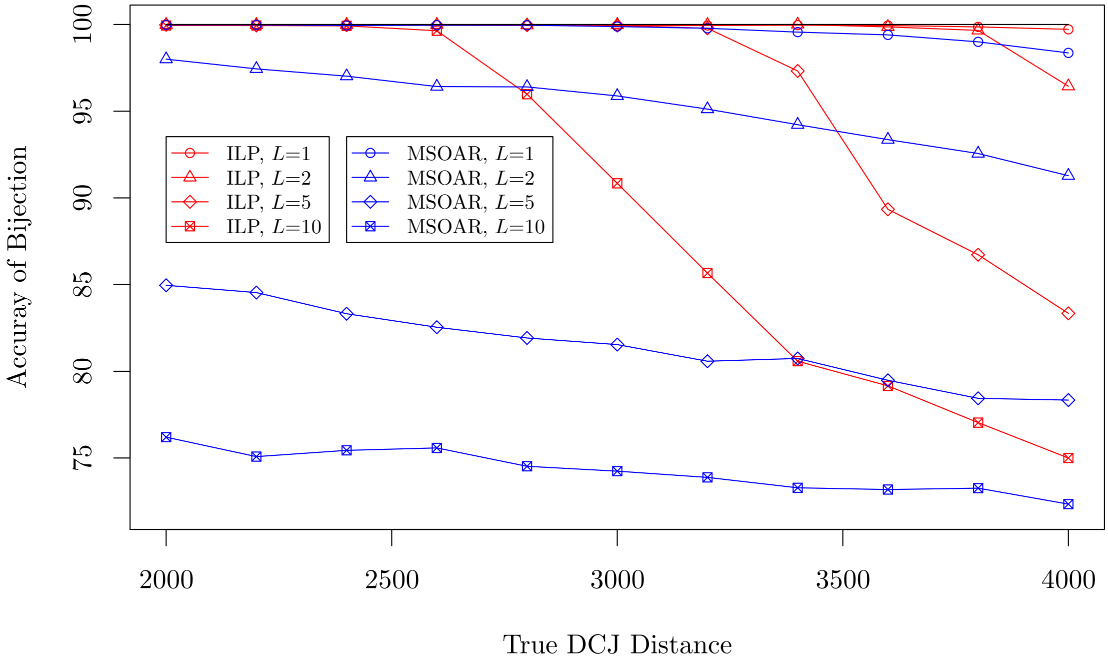


Figure 3.7 – The accuracy of the bijections for $S = 5000$. Red lines circles track our method, while blue lines track MSOAR.

or very close to the optimal solution, because the ILP solver can find the optimal solution very quickly, but must spend more time to verify that it is optimal. This observation is also verified by the very high accuracy before the saturation point shown in Figure 3.6.

3.4.2 Application to Orthology Assignment

Under a parsimonious evolutionary scenario, the optimal valid bijection between two genomes with the same gene content minimizes the number of DCJs after speciation, and thus infers the orthologous gene pairs [37]. We test both methods for assigning orthologous genes between pairs of genomes. Human, mouse, and rat genomes are well annotated, so we chose them to evaluate the performance of the two methods. For each species, we downloaded the information for all protein-coding genes from Ensembl (`http://www.ensembl.org`), including gene family names, positions on the chromosomes and gene symbols. If a gene has multiple alternative products, we keep its longest isoform. Two genes are considered homologous if they have the same Ensembl gene family name; they are considered orthologous if they have the same gene symbol. (Note that two orthologous genes are necessarily homologous, but two homologous genes need not be orthologous.) For a pair of genomes, we keep only orthologous gene pairs, thereby obtaining two genomes with the same gene content; our reference bijection is then defined by these orthologous gene pairs. For both methods, we use gene family and position information to infer orthologous relationships and compare them to the reference bijection.

The results of comparing these three genomes are shown in Table 3.1. Both methods mostly agree with annotation, indicating that the parsimonious model is appropriate when comparing these genomes; our method obtains slightly better accuracy. On human and mouse for example, our bijection has 122 different gene pairs compared with the reference bijection. Among these pairs, 34 of them can be explained by a simple structure, illustrated in Figure 3.8. For two identical segments, our method outputs a sequential bijection for which no DCJ operation is needed, while the reference bijection contains a crossover, for which at least two DCJ operations are needed. The other 87 pairs can be explained by 32 pairs of segments, for each of which our bijection needs fewer DCJ operations than the reference bijection. On the

species pairs	gene pairs	accuracy of bijection (%)		DCJ distance	
		MSOAR	our method	MSOAR	our method
human mouse	14876	98.63	99.18	933	894
human rat	12971	98.79	99.28	1320	1294
mouse rat	13525	98.60	99.26	968	916

Table 3.1 – Comparison of human, mouse and rat genomes. The column of *gene pairs* gives the total number of orthologous gene pairs for each pair of species. The two columns in the third category give the accuracy of the bijections returned by MSOAR and our method respectively. The two columns in the last category give the DCJ distance induced by the corresponding bijections returned by MSOAR and our method respectively.

comparison of the DCJ distance, our method gets fewer DCJ operations than MSOAR in all three pairs.

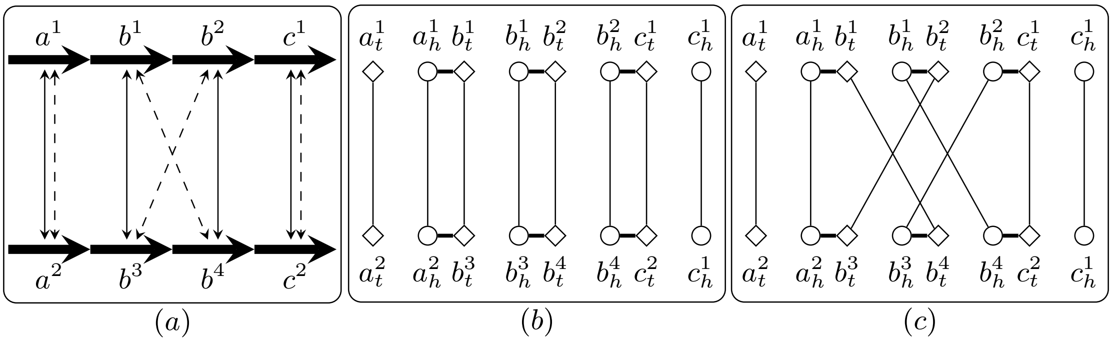

a^1 b^1 b^2 c^1
a^2 b^3 b^4 c^2
a_t^1 a_h^1 b_t^1 b_h^1 b_t^2 b_h^2 c_t^1 c_h^1
a_t^2 a_h^2 b_t^3 b_h^3 b_t^4 b_h^4 c_t^2 c_h^1
(a)
(b)
(c)

Figure 3.8 – Comparison of the reference bijection and our bijection. (**a**) Two identical segments. Our bijection is shown by solid lines while reference bijection is shown by dashed lines. (**b**) The adjacency graph corresponding to our bijection, in which there are 3 cycles. (**c**) The adjacency graph corresponding to the reference bijection, in which there is only 1 cycle.

3.5 Discussion

The ILP formulation can be extended in various ways. First, we can use the relaxed LP (linear programming) techniques to design possible approximation algorithms. Second, when we apply it to do orthology assignment, we can also take the sequence similarity information into account, by adding a term of the form $\sum_{e \in E} w_e \cdot x_e$ to the objective function, where w_e can be set to the similarity of the two genes. How to combine sequence similarity and DCJ distances remains an unexplored problem, but our ILP formulation provides a first step by allowing us to study linear combinations of the two.

4 Comparing Genomes with DCJs and Segmental Duplications

As we have seen, both rearrangements and content-modifying events are commonly observed in the course of evolution. However, algorithms that can handle them simultaneously in the presence of duplicate genes remain few and limited in their applicability. In this chapter, we study the comparison of two genomes with duplicate genes under a model including DCJ operations and segmental duplications. Formally, the problem is to compute a set of segmental duplications in each genome and a bijection between the nonduplicated genes, such that the total cost of the segmental duplications and the DCJs induced by the bijection is minimized. We propose an exact algorithm for this problem by formulating it as an integer linear program. Based on studying the the underlying structure of problem, we also devise a sufficient condition and an efficient algorithm to identify optimal substructures, which can simplify the problem while preserving optimality. Using the optimal substructures with the ILP formulation yields a practical and exact algorithm to solve the problem. We also discuss and propose a reasonable way to balance the costs between DCJs and segmental duplications. Finally, we apply our method to assign in-paralogs and orthologs and compare its performance with MSOAR on both simulated and biological datasets.

4.1 Problem Statement

We first give some notations. For a genome X, we use $\mathscr{A}(X)$ to denote all the gene families in X, and use $F(X,f)$ to denote the set of genes in X that come from gene family f. We say consecutive genes on one chromosome form a *segment*. The *length* of a segment s is defined as the number of genes in s, denoted by $|s|$. We say two segments in the same genome are *independent* if they do not contain the same gene. We say segments $s = (a_1, a_2, \cdots, a_n)$ and $t = (b_1, b_2, \cdots, b_n)$ are *homologous* if a_i and b_i are homologous and have the same sign for all $1 \le i \le n$, or a_i and b_{n+1-i} are homologous and have the opposite sign for all $1 \le i \le n$. We say segment s is *possibly duplicated*, if there exists segment t in the same genome such that s and t are independent and homologous. For a genome X, we use $\mathscr{D}(X)$ to denote the set of all the possibly duplicated segments in X (see Figure 4.1(**a**)). We say a subset $S \subset \mathscr{D}(X)$ is independent if every two segments in S are independent. For an independent subset $S \subset \mathscr{D}(X)$, we use $X \setminus S$ to denote the new genome after removing all genes appearing in the segments in S from X. Given two genomes X and Y, we say two independent subsets $S \subset \mathscr{D}(X)$ and $T \subset \mathscr{D}(Y)$ are *consistent* if $X \setminus S$ and $Y \setminus T$ have the same gene content, i.e., for each gene family $f \in \mathscr{A}(X) \cup \mathscr{A}(Y)$, we have $|F(X \setminus S, f)| = |F(Y \setminus T, f)|$ (see Figure 4.1(**a**)). In this chapter we assume that the given two genomes X and Y satisfy that $\mathscr{A}(X) = \mathscr{A}(Y)$; otherwise we modify them by removing all the genes that are not in $\mathscr{A}(X) \cap \mathscr{A}(Y)$. With this assumption, there always exist two independent subsets $S \in \mathscr{D}(X)$ and $T \in \mathscr{D}(Y)$ that are consistent.

Suppose we are given two independent consistent subsets $S \in \mathscr{D}(X)$ and $T \in \mathscr{D}(Y)$. We denote by $\mathscr{B}(X \setminus S, Y \setminus T)$ the set of bijections that map each gene in $X \setminus S$ to a homologous gene in $Y \setminus T$. If $X \setminus S$ and $Y \setminus T$ contain only *singletons*, i.e., we have $|F(X \setminus S, f)| = |F(Y \setminus T, f)| = 1$ for all $f \in \mathscr{A}(X) \cup \mathscr{A}(Y)$, then we have $|\mathscr{B}(X \setminus S, Y \setminus T)| = 1$, and the DCJ distance between $X \setminus S$ and $Y \setminus T$ is well defined and can be computed in linear time [11]. Once a bijection $B \in \mathscr{B}(X \setminus S, Y \setminus T)$ is given, we can relabel $X \setminus S$ and $Y \setminus T$ by assigning each pair of genes in B with a distinct gene family, and thus results in two new genomes with only singletons. We denote by $d(B)$ the DCJ distance between these two new genomes induced by bijection B.

In this chapter, we study the following problem: given two genomes X and Y satisfying $\mathscr{A}(X) = \mathscr{A}(Y)$, and a *cost* function $c(\cdot)$, which maps each segment in $\mathscr{D}(X) \cup \mathscr{D}(Y)$ to a positive value,

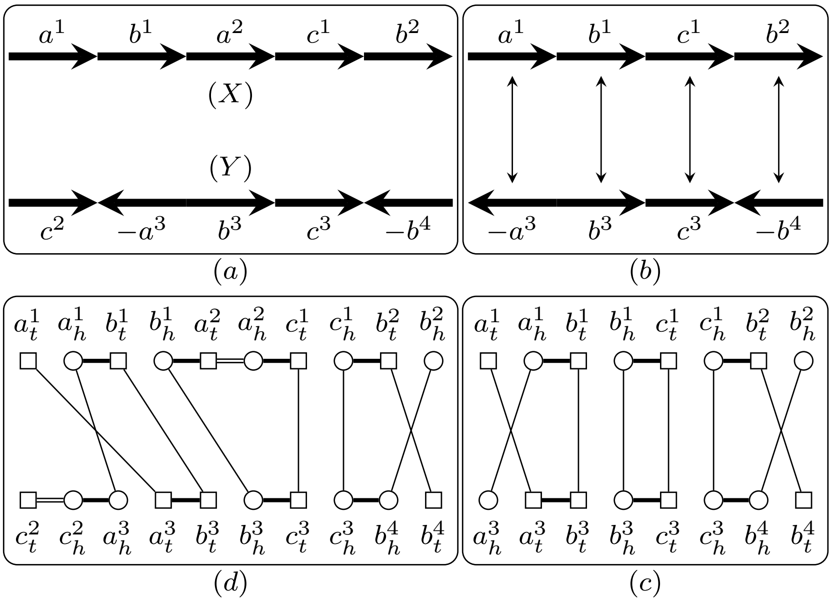

a^1 b^1 a^2 c^1 b^2
(X)
(Y)
c^2 $-a^3$ b^3 c^3 $-b^4$
a^1 b^1 c^1 b^2
$-a^3$ b^3 c^3 $-b^4$
(a)
(b)
a^1_t a^1_h b^1_t b^1_h a^2_t a^2_h c^1_t c^1_h b^2_t b^2_h
c^2_t c^2_h a^3_h a^3_t b^3_t b^3_h c^3_t c^3_h b^4_h b^4_t
a^1_t a^1_h b^1_t b^1_h c^1_t c^1_h b^2_t b^2_h
a^3_h a^3_t b^3_t b^3_h c^3_t c^3_h b^4_h b^4_t
(d)
(c)

Figure 4.1 – (**a**) Two genomes X and Y. Genes in the same gene family are represented by the same symbol with different superscripts. We have $\mathscr{D}(X) = \{(a^1), (a^2), (b^1), (b^2)\}$ and $\mathscr{D}(Y) = \{(c^2), (b^3), (c^3), (-b^4)\}$, and $S = \{(a^2)\}$ and $T = \{(c^2)\}$ are two consistent subsets. (**b**) The genomes $X \setminus S$ and $Y \setminus T$, and the bijection B. (**c**) The adjacency graph *w.r.t.* $Q = (S, T, B)$. Black edges are represented by long thin lines while gray edges by short thick lines. Head extremities are represented by circles while tail extremities by squares. (**d**) The extended adjacency graph *w.r.t.* Q, in which internal edges are represented by double lines.

compute a triple $Q = (S, T, B)$, where $S \subset \mathscr{D}(X)$ and $T \subset \mathscr{D}(Y)$ are two independent consistent subsets and $B \in \mathscr{B}(X \setminus S, Y \setminus T)$, such that the *total cost* of Q, $\gamma(Q) = \sum_{s \in S \cup T} c(s) + d(B)$, is minimized.

4.2 ILP Formulation

We now formulate the above problem as an integer linear program. To achieve that, we first introduce the *adjacency graph* in section 4.2.1, which is the essential data structure to compute the DCJ distance. We also propose a new extension of the adjacency graph, called the *extended adjacency graph*, which can incorporate duplicated genes and thus forms the basis for the following ILP formulation. We then describe a capping method to remove the telomeres, in section 4.2.2, which allows us only to count the number of cycles when computing the DCJ distance. Based on them, we finally give the ILP formulation in section 4.2.3.

4.2.1 Adjacency Graph

The set of all extremities in genome X is called the *extremity set* of X, denoted by $E(X)$. If genes a and b are homologous, we also say the two corresponding extremity pairs, a_h and b_h, a_t and b_t, are *homologous*. Suppose that we are given a triple $Q = (S, T, B)$, where $S \subset \mathscr{D}(X)$, $T \subset \mathscr{D}(Y)$ are two independent consistent subsets, and $B \in \mathscr{B}(X \setminus S, Y \setminus T)$. We can build the *adjacency graph w.r.t. Q*, denoted by $G(Q)$, as follows. We first build $X \setminus S$ and $Y \setminus T$ through removing all genes in $S \cup T$, and take all the extremities in them, i.e., $E(X \setminus S) \cup E(Y \setminus T)$, as the vertices of $G(Q)$. Then for each adjacency in $X \setminus S$ and $Y \setminus T$ we add one *gray edge* to connect the two extremities in it. Finally for each pair of homologous extremities specified by B (each homologous gene pair in B specifies two pairs of homologous extremities), we add one *black edge* to connect them (see Figure 4.1(**a,b,c**)). Clearly, in $G(Q)$ the degree of each vertex is at most 2, and thus it consists of a set of vertex-disjoint cycles and paths. The *length* of a cycle (or a path) is defined as the number of black edges in it. Let c be the number of cycles and o be the number of odd-length paths in $G(Q)$. We have that the DCJ distance induced by B can then be computed as $d(B) = n - c - o/2$, where n is the number of genes in $X \setminus S$ [11].

Given a triple $Q = (S, T, B)$ defined above, we propose an equivalent form of $G(Q)$, called the *extended adjacency graph w.r.t. Q*, denoted by $G'(Q)$. The set of vertices of $G'(Q)$ includes all the extremities in X and Y, i.e., $E(X) \cup E(Y)$. For each adjacency in X and Y, there is one gray edge connecting the two extremities in it. For each pair of homologous extremities specified by B, there is one black edge connecting them. For each gene contained in some segment in $S \cup T$, there is one *internal edge* connecting the two extremities in this gene (see Figure 4.1(**d**)). The difference between $G'(Q)$ and $G(Q)$ is that, the latter one explicitly removes those extremities in the genes in $S \cup T$, while the former one keeps them but adds internal edges connecting the two extremities in those genes. Clearly, $G'(Q)$ also consists a set of vertex-disjoint cycles and paths, and there is a one-to-one correspondence between the connected components in $G(Q)$ and that in $G'(Q)$. Thus, the DCJ distance induced by B can also be computed as

$d(B) = n - c' - o'/2$, where c' is the number of cycles and o' is the number of odd-length paths in $G'(Q)$, and n is the number of genes in $X \setminus S$. As we will see later, this extended adjacency graph is the key point in devising the ILP formulation.

4.2.2 Adding Capping Genes

In [54], we described a method to remove telomeres by introducing *capping genes*. A capping gene contains only one extremity, which combines with the adjacent telomere (or another capping gene) to form one adjacency. All capping genes are homologous to each other, forming a distinct gene family, denoted by f_τ. Given two genomes X and Y with l_X and l_Y linear chromosomes respectively (without loss of generality, we assume that $l_X \geq l_Y$), we first add one capping gene to each end of all the linear chromosomes in X and Y; then we add $(l_X - l_Y)$ *dummy* chromosomes, each of which contains only a pair of capping genes, to genome Y (see Figure 4.2). We denote by $\widehat{X}$ and $\widehat{Y}$ the two new genomes after adding capping genes for X and Y. Clearly, we have $|F(\widehat{X}, f_\tau)| = |F(\widehat{Y}, f_\tau)|$. Thus, given a pair of independent consistent subsets $S \subset \mathscr{D}(X)$ and $T \subset \mathscr{D}(Y)$, we know that $\widehat{X} \setminus S$ and $\widehat{Y} \setminus T$ also have the same gene content. Using the same argument as in [52], we can prove that

$$\min_{\widehat{B} \in \mathscr{B}(\widehat{X} \setminus S, \widehat{Y} \setminus T)} d(\widehat{B}) = \min_{B \in \mathscr{B}(X \setminus S, Y \setminus T)} d(B),$$

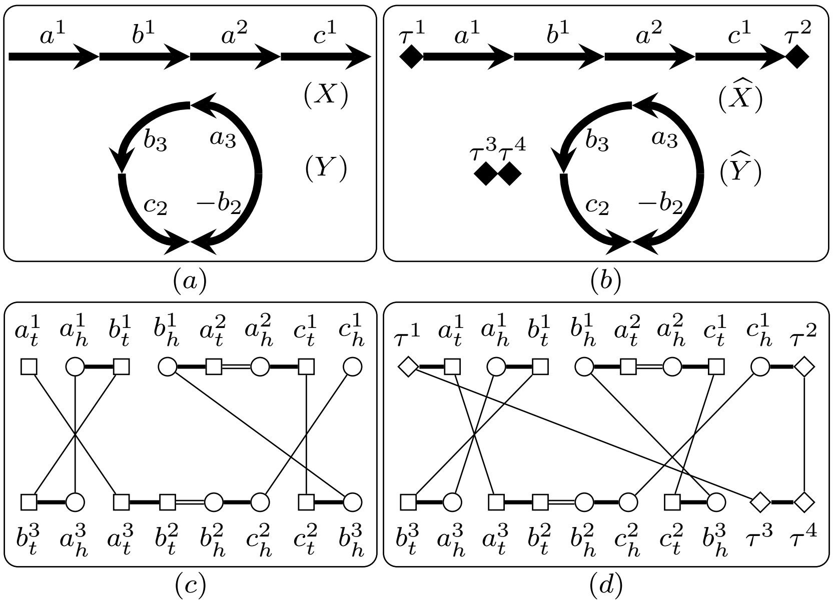

a^1
b^1
a^2
c^1
(X)
b_3
a_3
(Y)
c_2
$-b_2$
(a)
τ^1
a^1
b^1
a^2
c^1
τ^2
$(\widehat{X})$
$\tau^3 \tau^4$
b_3
a_3
$(\widehat{Y})$
c_2
$-b_2$
(b)
a_t^1 a_h^1 b_t^1 b_h^1 a_t^2 a_h^2 c_t^1 c_h^1
b_t^3 a_h^3 a_t^3 b_t^2 b_h^2 c_h^2 c_t^2 b_h^3
(c)
τ^1 a_t^1 a_h^1 b_t^1 b_h^1 a_t^2 a_h^2 c_t^1 c_h^1 τ^2
b_t^3 a_h^3 a_t^3 b_t^2 b_h^2 c_h^2 c_t^2 b_h^3 τ^3 τ^4
(d)

Figure 4.2 – (**a**) Two genomes X and Y. (**b**) The genomes $\widehat{X}$ and $\widehat{Y}$ after adding capping genes, where capping genes are represented by diamonds. (**c**) The extended adjacency graph *w.r.t.* (S, T, B), where $S = \{(a^2)\}$, $T = \{(b^2)\}$ and B maps a^1, b^1 and c^1 to a^3, b^3 and c^2 respectively. (**d**) The extended adjacency graph *w.r.t.* $(S, T, \widehat{B})$, where $\widehat{B}$ consists of the two pairs mapping τ^1 and τ^2 to τ^3 and τ^4 respectively, and those pairs in B.

and the optimal B can be recovered from the optimal $\widehat{B}$ through discarding the pairs with capping genes. This statement allows us to add capping genes to remove telomeres on the two given genomes without affecting the optimal bijection. Since the two new genomes $\widehat{X}$ and $\widehat{Y}$ do not contain telomeres, we have that for any triple $Q = (S, T, \widehat{B})$, where $S \subset \mathscr{D}(X)$, $T \subset \mathscr{D}(Y)$ and $\widehat{B} \in \mathscr{B}(\widehat{X} \setminus S, \widehat{Y} \setminus T)$, both $G(Q)$ and $G'(Q)$ contain only cycles (see Figure 4.2). This property allows us only to count the number of cycles when computing the DCJ distance, which simplifies the following ILP formulation.

4.2.3 ILP Formulation

Let X and Y be two given genomes after adding capping genes. Let $Q^* = (S^*, T^*, B^*)$ be the optimal triple minimizing $\sum_{s \in S^* \cup T^*} c(s) + d(B^*)$, where we have $S^* \subset \mathscr{D}(X)$, $T^* \subset \mathscr{D}(Y)$ and $B^* \in \mathscr{B}(X \setminus S^*, Y \setminus T^*)$. (Notice that here X and Y may contain capping genes, but we define $\mathscr{D}(X)$ and $\mathscr{D}(Y)$ are in terms of the original genomes, which do not contain segments with capping genes.) To facilitate our description, we use $a \in X$ to denote that gene a is contained in genome X. We use $a \in s$ to denote that gene a is contained in segment s. We denote by f_a the gene family to which gene a belongs. We say gene a is *duplicated* in Q^*, if there exists one segment $s \in S^* \cup T^*$ such that $a \in s$, and *nonduplicated* otherwise.

We now give the ILP formulation to compute Q^*. For each segment $s \in \mathscr{D}(X) \cup \mathscr{D}(Y)$, we have one binary variable x_s to indicate whether $s \in S^* \cup T^*$. For each gene $a \in X \cup Y$, we have one binary variable y_a to indicate whether a is duplicated in Q^*. We use the following two sets of constraints to guarantee that $y_a = 1$ if and only if there exists one segment $s \in S^* \cup T^*$ such that $a \in s$:

$$\begin{array}{rcl} y_a & \geq & x_s, \quad \forall s \in \mathscr{D}(X) \cup \mathscr{D}(Y) \text{ and } \forall a \in s; \\ y_a & \leq & \sum\limits_{s \in \mathscr{D}(X) \cup \mathscr{D}(Y) : a \in s} x_s, \quad \forall a \in X \cup Y. \end{array}$$

We require that these segments in $S^* \cup T^*$ are independent, i.e., there do not exist two of them that contain the same gene:

$$\sum_{s \in \mathscr{D}(X) \cup \mathscr{D}(Y) : a \in s} x_s \leq 1, \quad \forall a \in X \cup Y.$$

We also require that $X \setminus S^*$ and $Y \setminus T^*$ have the same gene content, i.e., for each gene family there must be an equal number of nonduplicated genes in Q^* in this family in each genome:

$$\sum_{a \in F(X,f)} (1 - y_a) = \sum_{b \in F(Y,f)} (1 - y_b), \quad \forall f \in \mathscr{A}(X).$$

And for each gene family, at least one gene is nonduplicated in Q^*:

$$\begin{array}{rcl} \sum_{a \in F(X,f)} (1-y_a) & \geq & 1, \quad \forall f \in \mathscr{A}(X); \\ \sum_{b \in F(Y,f)} (1-y_b) & \geq & 1, \quad \forall f \in \mathscr{A}(Y). \end{array}$$

For each pair of homologous genes $a \in X$ and $b \in Y$, we add one binary variable $z_{a,b}$ to indicate whether B^* contains this pair. We require that for each gene in $X \cup Y$, it is mapped to exactly one homologous gene in the opposite genome if and only if it is nonduplicated in Q^*:

$$\begin{array}{rcl} \sum_{b \in F(Y,f_a)} z_{a,b} & = & 1-y_a, \quad \forall a \in X; \\ \sum_{a \in F(X,f_b)} z_{a,b} & = & 1-y_b, \quad \forall b \in Y. \end{array}$$

These constraints guarantee that these pairs in B^* form a valid bijection between the genes in $X \setminus S^*$ and those in $Y \setminus T^*$. To compute $d(B^*)$, we need to count the number of cycles in $G'(Q^*)$. We add a variable l_e for each extremity $e \in E(X) \cup E(Y)$ to represent the *label* of e. We then assign a *distinct* upper bound for l_e, denoted by U_e (for example, we can just sort all the extremities in $E(X) \cup E(Y)$ in an arbitrary order and assign U_e as the index of e in the sorted list):

$$0 \leq l_e \leq U_e, \quad \forall e \in E(X) \cup E(Y).$$

We then require that all the extremities in the same cycle in $G'(Q^*)$ have the same label. This can be achieved by forcing that the two extremities connected by any edge in $G'(Q^*)$ have the same label. To guarantee this, we add the following three groups of constraints, each of which corresponds to one type of edges. First, we require that the two extremities in each adjacency have the same label (these constraints correspond to the gray edges):

$$l_{e_i} = l_{e_j}, \quad \forall \{e_i, e_j\} \text{ form an adjacency in } X \text{ or in } Y.$$

Second, we require that each pair of extremities specified by B^* have the same label (these constraints correspond to the black edges). To achieve that, we add the following four constraints for each pair of homologous genes $a \in X$ and $b \in Y$ (if a and b are capping genes, then we have $a_h = a_t$ and $b_h = b_t$ and thus the following four constraints degenerate into two):

$$\begin{array}{rcl} l_{a_h} & \leq & l_{b_h} + (1-z_{a,b}) \cdot U_{a_h}; \\ l_{b_h} & \leq & l_{a_h} + (1-z_{a,b}) \cdot U_{b_h}; \\ l_{a_t} & \leq & l_{b_h} + (1-z_{a,b}) \cdot U_{a_t}; \\ l_{b_t} & \leq & l_{a_h} + (1-z_{a,b}) \cdot U_{b_t}. \end{array}$$

Third, we require that the two extremities in each duplicated gene have the same label (these constraints correspond to the internal edges):

$$\begin{array}{rcll} l_{a_h} & \leq & l_{a_t} + (1 - y_a) \cdot U_{a_h}, & \forall a \in X \cup Y; \\ l_{a_t} & \leq & l_{a_h} + (1 - y_a) \cdot U_{a_t}, & \forall a \in X \cup Y. \end{array}$$

We then add a binary variable w_e for extremity e to indicate whether l_e reaches its upper bound:

$$w_e \cdot U_e \leq l_e, \quad \forall e \in E(X) \cup E(Y).$$

Since all the extremities in the same cycle in $G'(Q^*)$ are forced to have the same label, and all label variables have distinct upper bounds, we know that for each cycle in $G'(Q^*)$ at most one extremity can have its label reaching its upper bound. Thus, we have that

$$\sum_{e \in E(X) \cup E(Y)} w_e$$

is exactly the number of cycles in $G'(Q^*)$. And $d(B^*)$ can then be computed by

$$|X| - \sum_{a \in X} y_a - \sum_{e \in E(X) \cup E(Y)} w_e,$$

where the first two items give the number of genes in $X \setminus S^*$.

Finally, we set the objective function of the ILP as

$$\min \sum_{s \in \mathscr{D}(X) \cup \mathscr{D}(Y)} c(s) \cdot x_s + |X| - \sum_{a \in X} y_a - \sum_{e \in E(X) \cup E(Y)} w_e.$$

4.3 Identifying Optimal Substructures

Given two genomes X and Y after adding capping genes, we say two homologous segments s in X and t in Y form a pair of *shared* segments, denoted by $\langle s, t \rangle$. Intuitively, shared segments are more likely to be nonduplicated and mapped to each other. Below, we give one sufficient condition and one algorithm to decide whether a pair of shared segments is *in* some optimal solution, i.e., in this optimal solution a_i and b_i are nonduplicated and a_i is mapped to b_i, for all $1 \leq i \leq n$. From now on, we assume that the cost function only depends on the length of the segments, i.e., we assume that if $|s| = |t|$ then we have $c(s) = c(t)$.

4.3.1 A Sufficient Condition

We say gene a in genome X is *isolated*, if there does not exist any segment $s \in \mathscr{D}(X)$ such that $a \in s$ and $|s| \geq 2$. The following theorem gives a sufficient condition to decide whether a pair of shared segments of length two is an optimal substructure.

Theorem 4.3.1. *Let $p = \langle (a^1, b^1), (a^2, b^2) \rangle$. If we have a^1 and a^2 are singletons, and b^1 and b^2 are isolated, then p is in some optimal solution.*

Proof. Let $Q = (S, T, B)$ be an arbitrary triple such that either b^1 or b^2 is duplicated in Q, or B does not contain $\langle b^1, b^2 \rangle$. Below, we will show that we can always build a new triple $Q' = (S', T', B')$ in which both b^1 and b^2 are nonduplicated and B' contains $\langle b^1, b^2 \rangle$, and also verify that $\gamma(Q') \leq \gamma(Q)$. Since Q is arbitrary, this proves the theorem.

First, assume that in Q both b^1 and b^2 are duplicated. Let $s \in S$ and $t \in T$ be the segments containing b^1 and b^2 respectively. Since both b^1 and b^2 are isolated, we know that $|s| = |t| = 1$. Let $S' = S \setminus \{s\}$ and $T' = T \setminus \{t\}$. We have that $X \setminus S'$ and $Y \setminus T'$ still have the same content. Let $B' = B \cup \{\langle b^1, b^2 \rangle\}$. We have that $d(B') = d(B)$, since $X \setminus S'$ has one more gene than $X \setminus S$, while $G(Q')$ has one more cycle than $G(Q)$ (see Figure 4.3(**a,b**)). Thus, we have $\gamma(Q') = \sum_{u \in S' \cup T'} c(u) + d(B') = \sum_{u \in S \cup T} c(u) - c(s) - c(t) + d(B) \leq \sum_{u \in S \cup T} c(u) + d(B) = \gamma(Q)$.

Second, assume that in Q gene b^2 is duplicated while b^1 is not (or symmetrically, b^1 is duplicated while b^2 is not). Suppose that b^1 is mapped to b^3 in B, i.e., $\langle b^1, b^3 \rangle \in B$. Let $S' = S$ and $T' = T \setminus \{t\} \cup \{t'\}$, where $t \in T$ is the segment containing b^2, and t' is the segment containing only gene b^3. Clearly, we also have that $X \setminus S'$ and $Y \setminus T'$ have the same content. Let $B' = B \setminus \{\langle b^1, b^3 \rangle\} \cup \{\langle b^1, b^2 \rangle\}$. To compare $d(B')$ with $d(B)$, consider the difference between $G(Q')$ and $G(Q)$. In fact, we can transform $G(Q)$ into $G(Q')$ through two DCJs on genome Y (after that we need to rename b^3 as b^2). We first perform one DCJ to cut b^3 out to create the adjacency $\{b_h^3, b_t^3\}$ (see Figure 4.3(**c,d**)). This operation might decrease the number of cycles, but the number decreased is at most 1 according to the prop-

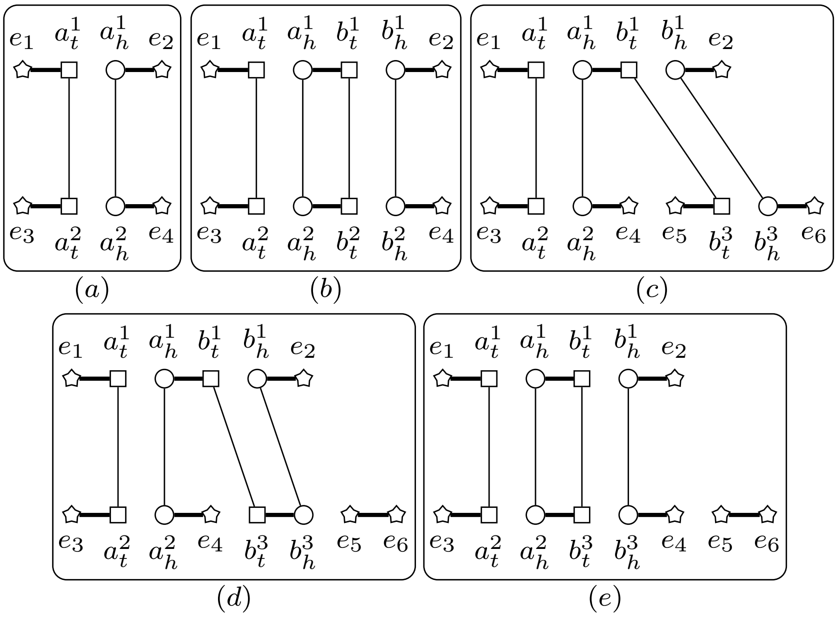

e_1 a_t^1 a_h^1 e_2
e_3 a_t^2 a_h^2 e_4
e_1 a_t^1 a_h^1 b_t^1 b_h^1 e_2
e_3 a_t^2 a_h^2 b_t^2 b_h^2 e_4
e_1 a_t^1 a_h^1 b_t^1 b_h^1 e_2
e_3 a_t^2 a_h^2 e_4 e_5 b_t^3 b_h^3 e_6
e_1 a_t^1 a_h^1 b_t^1 b_h^1 e_2
e_3 a_t^2 a_h^2 e_4 b_t^3 b_h^3 e_5 e_6
e_1 a_t^1 a_h^1 b_t^1 b_h^1 e_2
e_3 a_t^2 a_h^2 b_t^3 b_h^3 e_4 e_5 e_6
(a) (b) (c) (d) (e)

Figure 4.3 – (**a,b**) The adjacency graph before and after adding $\langle b^1, b^2 \rangle$. (**c,d,e**) Transforming $G(Q)$ into $G(Q')$ using two DCJs. Irrelevant extremities are represented by stars.

erty of the DCJ model. We then insert b^3 back as the neighbor of a^2 to form the segment (a^2, b^3), which will increase the number of cycles by 1 (see Figure 4.3(**d,e**)). This implies that the number of cycles in $G(Q')$ is no less than that in $G(Q)$. In addition to the fact that $X \setminus S'$ and $X \setminus S$ have the same number of genes, we have that $d(B') \leq d(B)$. Thus, we have $\gamma(Q') = \sum_{u \in S' \cup T'} c(u) + d(B') \leq \sum_{u \in S \cup T} c(u) - c(t) + c(t') + d(B) = \sum_{u \in S \cup T} c(u) + d(B) = \gamma(Q)$. The last equality uses the assumption that the cost function only depends on the length of the segments.

Third, assume that in Q both b^1 and b^2 are nonduplicated, and b^1 is mapped to b^3 while b^4 is mapped b^2. Let $S' = S$, $T' = T$ and $B' = B \setminus \{\langle b^1, b^3 \rangle, \langle b^4, b^2 \rangle\} \cup \{\langle b^1, b^2 \rangle, \langle b^4, b^3 \rangle\}$. Using the same technique in the Theorem 1 [52], we can prove that $d(B') \leq d(B)$. Thus, we still have $\gamma(Q') = \sum_{u \in S' \cup T'} c(u) + d(B') \leq \gamma(Q)$. □

4.3.2 An Algorithm

We say a pair of shared segments $p = \langle (a_1, a_2, \cdots, a_n), (b_1, b_2, \cdots, b_n) \rangle$ between genomes X and Y is *half fixed*, if b_i is singleton for all $1 \leq i \leq n$ (and thus none of them can be duplicated) and all genes in $F(X, f_{a_i})$ are isolated for all $1 \leq i \leq n$. Let p be such a pair of half fixed shared segments (PHFSS for short). We use $\mathscr{A}(p)$ to denote all the gene families in p, i.e., $\mathscr{A}(p) = \{f_{a_1}, f_{a_2}, \cdots, f_{a_n}\}$. In this section, we propose an algorithm to decide whether a PHFSS is in some optimal solution. Notice that for a PHFSS p, if we further know that in some optimal solution a_k is mapped to b_k for some $1 \leq k \leq n$, then we can immediately conclude that the whole p is in some optimal solution by iteratively applying theorem 4.3.

Let $Q^*_{\bar{p}} = (S, T, B)$ be the triple with smallest total cost among these triples that do not contain p (i.e., b_i is not mapped to a_i for all $1 \leq i \leq n$). We now modify $Q^*_{\bar{p}}$ to replace a'_i with a_i, where a'_i is the gene that are mapped to b_i in $Q^*_{\bar{p}}$. Notice that $\{(a_1), (a_2), \cdots, (a_n)\} \subset S$, since a_i is duplicated in $Q^*_{\bar{p}}$ (because b_i is singleton and a'_i is nonduplicated in $Q^*_{\bar{p}}$) and all genes in $F(X, f_{a_i})$ are isolated. Let $S' = S \setminus \{(a_1), \cdots, (a_n)\} \cup \{(a'_1), \cdots, (a'_n)\}$, $B' = B \setminus \{\langle a'_1, b_1 \rangle, \cdots, \langle a'_n, b_n \rangle\} \cup \{\langle a_1, b_1 \rangle, \cdots, \langle a_n, b_n \rangle\}$ and $Q^*_p = (S', T, B')$. Clearly Q^*_p contains p. According to the definition of $Q^*_{\bar{p}}$, if we can show that $\gamma(Q^*_p) \leq \gamma(Q^*_{\bar{p}})$, then p is in some optimal solution. From the construction of Q^*_p, we can see clearly that the cost of the segmental duplications in $Q^*_{\bar{p}}$ is equal to that in Q^*_p. Thus we only need to compare the number of DCJs between $Q^*_{\bar{p}}$ and Q^*_p.

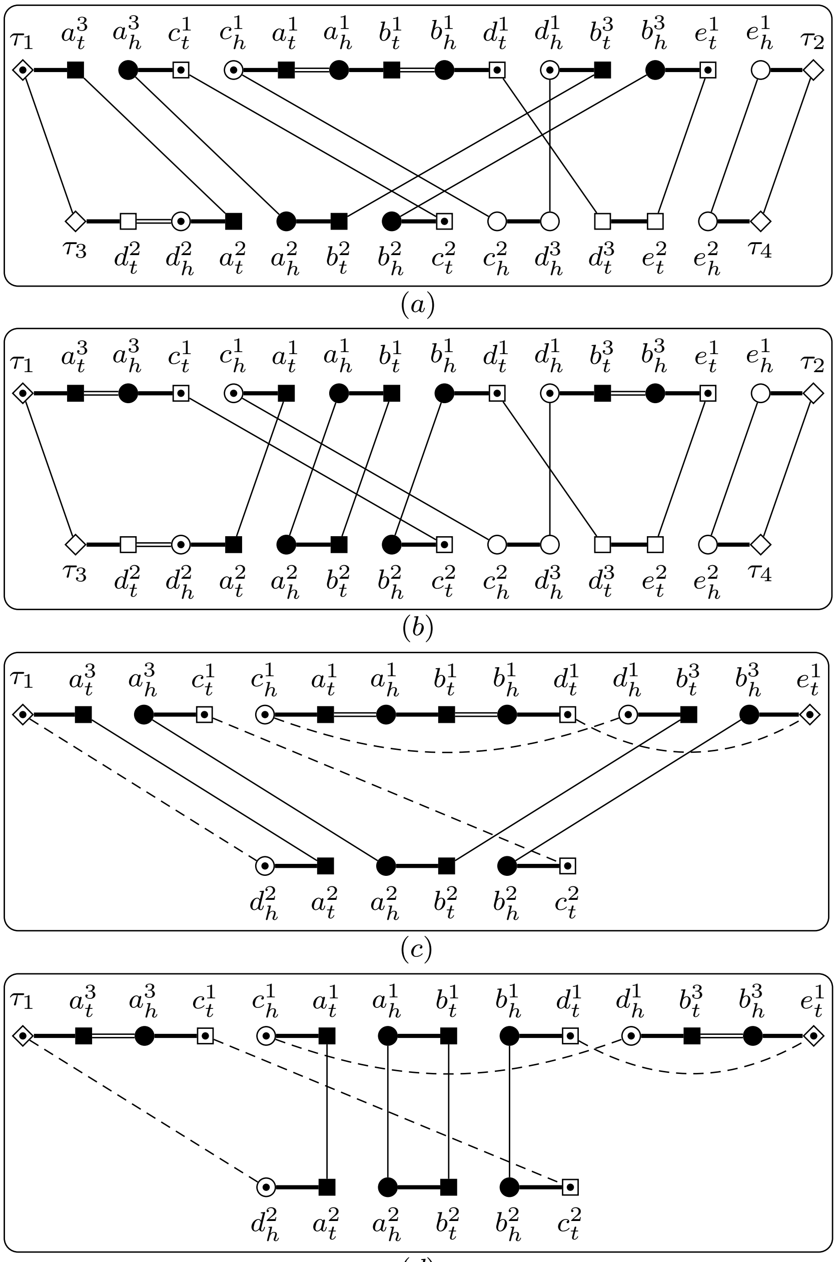
We compare the number of cycles in $G'(Q^*_{\bar{p}})$ and $G'(Q^*_p)$. Notice that $G'(Q^*_{\bar{p}})$ and $G'(Q^*_p)$ differ only on these gene families in $\mathscr{A}(p)$. We now define a new graph to focus on $\mathscr{A}(p)$ while hiding others. Let Q be a triple and p be a PHFSS. We can build the *reduced adjacency graph w.r.t.* Q and p, denoted by $R(Q, p)$, as follows. The vertices of $R(Q, p)$ are divided into two types, the *core vertices*, which are exactly those extremities in the genes in the gene families in $\mathscr{A}(p)$, and the *boundary vertices*, which consist of these extremities that form adjacencies with core vertices (see Figure 4.4(**a,b**)). The edges of $R(Q, p)$ are divided into four types, gray edges, black edges, internal edges, and *reduced edges*. For any two vertices in $R(Q, p)$, they are connected by gray edges or internal edges, if and only if they are connected by the same type of edge in

$G'(Q)$. For any two core vertices in $R(Q,p)$, they are connected by one black edge if and only if they are connected by one black edge in $G'(Q)$. For any two boundary vertices in $R(Q,p)$, they are connected by one reduced edge if there exists one path connecting them in $G'(Q)$ without going through any core vertices or boundary vertices (except its two ends). Clearly, $R(Q,p)$ also consists of a set of vertex-disjoint cycles (see Figure 4.4(**c,d**)).

We claim that the difference of the number of cycles between $G'(Q^*_{\overline{p}})$ and $G'(Q^*_p)$ is the same as that between $R(Q^*_{\overline{p}},p)$ and $R(Q^*_p,p)$. In fact, the cycles that do not contain any core vertices or boundary vertices are the same between $Q^*_{\overline{p}}$ and Q^*_p according to the construction of Q^*_p; those cycles do not appear in either $R(Q^*_{\overline{p}},p)$ or $R(Q^*_p,p)$. Moreover, for each cycle in $G'(Q^*_{\overline{p}})$ that contains some core vertices, there exists one corresponding cycle in $R(Q^*_{\overline{p}},p)$, since the reduction procedure in constructing $R(Q^*_{\overline{p}},p)$ can only shorten the length of each cycle, while it cannot merge or split it; it is the same for $G'(Q^*_p)$ and $R(Q^*_p,p)$. Thus, the claim holds. Furthermore, the reasoning used here also implies that we can construct $R(Q^*_p,p)$ directly from $R(Q^*_{\overline{p}},p)$, rather than from $G'(Q^*_p)$: we can first replace the black edges corresponding to $\langle a'_i, b_i\rangle$ with that corresponding to $\langle a_i, b_i\rangle$, and then replace the internal edges corresponding to a_i with that corresponding to a'_i.

In summary, once we know $R(Q^*_{\overline{p}},p)$, we can then construct $R(Q^*_p,p)$, and compare the number of cycles in them. If the number of cycles in $R(Q^*_p,p)$ is no less than that in $R(Q^*_{\overline{p}},p)$, then p is in some optimal solution. However, the problem is that we do not know $R(Q^*_{\overline{p}},p)$. Our strategy is to enumerate all the possibilities of $R(Q^*_{\overline{p}},p)$. The vertices of $R(Q^*_{\overline{p}},p)$, i.e., all the core vertices and all the boundary vertices *w.r.t.* p, can be computed in advance very easily. All the genes of a_i, $1 \le i \le n$, are duplicated in $R(Q^*_{\overline{p}},p)$ by definition, and thus the two extremities in a_i are always connected by one internal edge in $R(Q^*_{\overline{p}},p)$. All genes in $F(X, f_{a_i}) \setminus \{a_i\}$ are possibly mapped to b_i in $R(Q^*_{\overline{p}},p)$. For any two boundary vertices (maybe in the same genome), we need to check whether they can be connected by one reduced edge in $R(Q^*_{\overline{p}},p)$, i.e., whether there exists one possible path connecting them that does not go through any other core vertices or boundary vertices. Notice that this path must be *alternating*, i.e., the edges with odd indices must be either black edges or internal edges, and the edges with even indices must be gray edges (see Figure 4.4(**c,d**)).

There exists a linear time algorithm to decide the existence of an alternating path between two given vertices [60]. We now adapt it for our use. Given a PHFSS p and two boundary vertices x and y, the algorithm first build a graph with $V_1 \cup V_2 \cup \{x,y\}$ as its vertices, where V_1 is the set of all extremities except all the core vertices and boundary vertices, and V_2 is a copy of V_1. Two extremities in V_1 are connected by one gray edge if they form one adjacency. Two homologous extremities in V_2 in different genomes are connected by one black edge, and the two extremities in V_2 in a possibly duplicated gene are connected by one internal edge. We connect x (resp. y) to its all homologous extremities in V_2 in the opposite genome by black edges. Finally, all the counterparts between V_1 and V_2 are connected by *bridging* edges (see Figure 4.5). Clearly, all the bridging edges form a matching of size $|V_1|$, denoted by M. The algorithm then computes an augmenting path *w.r.t.* M using the Blossom algorithm, which takes linear time. We claim that such an augmenting path exists if and only if there exists one alternating path connecting x and y without going through any core vertices or boundary vertices. In fact, if such an augmenting path exists, then the two ends of this path must be x and y, since they are the only two unmatched vertices. We claim that the edges in


τ_1 a_t^3 a_h^3 c_t^1 c_h^1 a_t^1 a_h^1 b_t^1 b_h^1 d_t^1 d_h^1 b_t^3 b_h^3 e_t^1 e_h^1 τ_2
τ_3 d_t^2 d_h^2 a_t^2 a_h^2 b_t^2 b_h^2 c_t^2 c_h^2 d_h^3 d_t^3 e_t^2 e_h^2 τ_4
τ_1 a_t^3 a_h^3 c_t^1 c_h^1 a_t^1 a_h^1 b_t^1 b_h^1 d_t^1 d_h^1 b_t^3 b_h^3 e_t^1 e_h^1 τ_2
τ_3 d_t^2 d_h^2 a_t^2 a_h^2 b_t^2 b_h^2 c_t^2 c_h^2 d_h^3 d_t^3 e_t^2 e_h^2 τ_4
τ_1 a_t^3 a_h^3 c_t^1 c_h^1 a_t^1 a_h^1 b_t^1 b_h^1 d_t^1 d_h^1 b_t^3 b_h^3 e_t^1
d_h^2 a_t^2 a_h^2 b_t^2 b_h^2 c_t^2
τ_1 a_t^3 a_h^3 c_t^1 c_h^1 a_t^1 a_h^1 b_t^1 b_h^1 d_t^1 d_h^1 b_t^3 b_h^3 e_t^1
d_h^2 a_t^2 a_h^2 b_t^2 b_h^2 c_t^2

(a)

(b)

(c)

(d)

Figure 4.4 – $X = (a^3, c^1, a^1, b^1, d^1, b^3, e^1)$, $Y = (d^2, a^2, b^2, c^2, -d^3, e^2)$. $p = \langle(a^1, b^1), (a^2, b^2)\rangle$. The four subgraphs show $G'(Q^*_{\overline{p}})$, $G'(Q^*_p)$, $R(Q^*_{\overline{p}}, p)$ and $R(Q^*_p, p)$, respectively. The core vertices are shown as solid patterns, and the boundary vertices are shown as patterns with one inner point. Reduced edges are shown as dashed lines.

the augmenting path that are not in M form a alternating path connecting x and y. This is because edges in M are spanning V_1 and V_2, while gray edges are all inside in V_1 and black edges and internal edges are inside in V_2. The opposite side of statement can be reasoned in a similar way.

The algorithm to decide whether a given PHFSS p is in some optimal solution proceeds as follows. The first phase of the algorithm is to compute the core vertices and the boundary vertices *w.r.t.* p, and then for each pair of boundary vertices, to check whether they can be connected by a reduced edge. If the total number of edges (reduced edges plus those among core vertices) is larger than $\log n$, the algorithm terminates. Otherwise, the algorithm comes to the second phase. It enumerates all the possibilities of $R(Q^*_{\bar{p}}, p)$: for each possible valid combination of the reduced edges (i.e., they form a matching that covers all the boundary vertices), it enumerates all the possible valid mappings for the genes in $\mathscr{A}(p)$ (a_i cannot be mapped to b_i by the definition of $R(Q^*_{\bar{p}}, p)$), and the mapping that yields the maximum number of cycles, plus the current combination of the reduced edges, forms one possibility of $R(Q^*_{\bar{p}}, p)$. After that, for each possibility of $R(Q^*_{\bar{p}}, p)$, it then builds $R(Q^*_{p}, p)$, and compares the number of cycles between them. If the number of cycles in $R(Q^*_{p}, p)$ is always no less than that in $R(Q^*_{\bar{p}}, p)$ for all the possibilities, then the algorithm concludes that p is in some optimal solution.

The above algorithm runs in polynomial-time. In fact, the first phase runs in polynomial-time, since we can decide the existence of a reduced edge for each pair of boundary vertices in linear time. In the second phase, the number of edges is in logarithmic-size, which implies that the number of possibilities of $R(Q^*_{\bar{p}}, p)$ is in polynomial-size. Thus, the second phase also runs in polynomial-time.

We remark that usually not all pairs of boundary vertices can be connected by a reduced edge (see Figure 4.6). In fact, if this is not the case, then there always exists one possibility such that $R(Q^*_{\bar{p}}, p)$ contains more cycles than $R(Q^*_{p}, p)$, in which case the algorithm fails. In other words, the first phase to identify possible reduced edges is very essential, which not only decreases the number of possibilities, but more importantly, makes the algorithm capable of

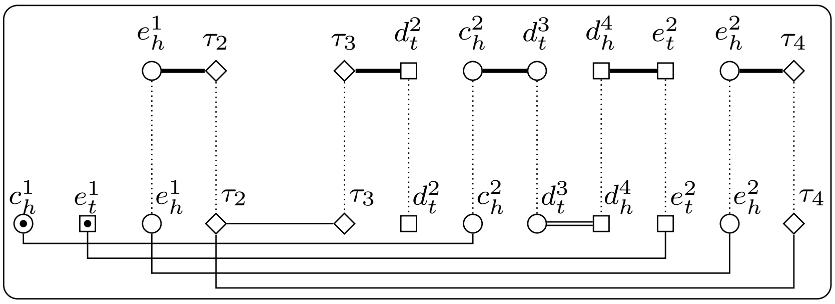


Figure 4.5 – The underlying graph used to decide the existence of an alternating path connecting c_h^1 and e_t^1 *w.r.t.* p for the same instance in Figure 4.4. All bridging edges are shown as dotted lines.

identifying optimal substructures. We also remark that this algorithm is a sufficient test, i.e., if it returns “yes”, then p is guaranteed in some optimal solution. However, if it returns “no”, then it is still possible that p is in some optimal solution. This is because two reduced edges in $R(Q^*_{\overline{p}}, p)$ might not be able to coexist in $G'(Q^*_{\overline{p}})$.

We can apply the theorem in section 4.3.1 and the algorithm in section 4.3.2 on all shared segments to verify their optimality. If such an optimal substructure is identified, we immediately fix it and update the genomes through assigning each pair of genes in it a distinct gene family. We can iteratively repeat this process until no such optimal substructure can be found. This serves as a preprocessing algorithm to simplify the problem before calling the ILP solver. The performance of this preprocessing algorithm on real genomes is analyzed in Table 4.4.

4.4 Setting the Cost

Under a parsimonious model, it is natural to set a unit cost for all segmental duplications (as we do for all DCJs). However, in this case, two segmental duplications, one in each genome, that creates a pair of shared segments can be always explained as two DCJs with the same total cost. Consider the example in Figure 4.7(**a**), for which we have two optimal solutions with total cost of 2: one is to regard a^2 and a^4 as duplicated genes, and the other uses two DCJs, which first cut a^2 out from X and then insert it back between c^1 and d^1. The scenario in the second case (two DCJs using one circular chromosome as intermediate) requires 3 inversions to explain, and therefore it is much less unlikely to happen comparing with the first scenario. Thus, to avoid the second case we set $c(\cdot) < 1$.

On the other hand, if we have $c(\cdot) \leq 0.5$, then every DCJ that inverts a possibly duplicated segment can be always explained by two segmental duplications with the same or even better total cost. Consider the example in Figure 4.7(**b**), for which one solution is to use only one DCJ to invert the segment (a^2, b^2) on X. However, if we have $c(\cdot) \leq 0.5$, then we can regard (a^2, b^2) and $(-b^4, -a^4)$ as duplicated segments, whose total cost is at most 1. Thus, to avoid the second case, we need to set $c(\cdot) > 0.5$.

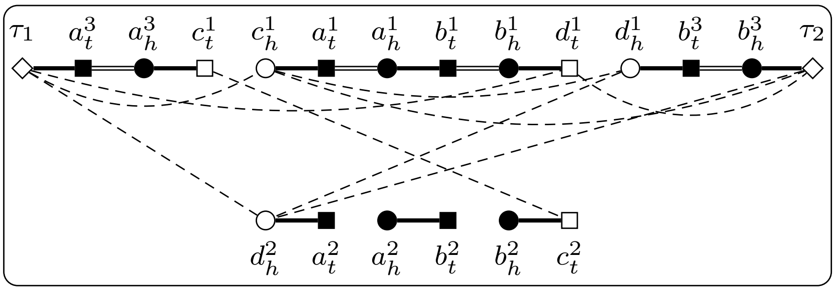


Figure 4.6 – All the possible reduced edges in $R(Q^*_{\overline{p}}, p)$ for the same instance in Figure 4.4. We can verify that among all possibilities of $R(Q^*_{\overline{p}}, p)$, the number of cycles in $R(Q^*_p, p)$ is always no less than that in $R(Q^*_{\overline{p}}, p)$. Thus, in this instance, p is optimal.

Combining the above two facts, in the following experiments, we set $c(\cdot) = 0.75$.

4.5 Inferring In-paralogs and Orthologs

Under a most parsimonious evolutionary scenario, the duplicated genes in the optimal triple infer the in-paralogs in each genome, while the bijection between the nonduplicated genes in the two genomes infers a subset of the orthology pairs (more specifically, positional orthologs [61]). In the following, we apply our method to infer in-paralogs and orthologs on both simulated datasets and biological datasets, and compare its performance with MSOAR.

4.5.1 Results on Simulated Datasets

We simulate a pair of genomes as follows. We start from an ancestor genome with only one linear chromosome consisting of $N = 5000$ singletons (we also test $N = 1000$ and $N = 2000$; the results are not presented since they agree with $N = 5000$). We then perform S_1 segmental duplications on the ancestor genome to make some gene families contain more than one copy. A segmental duplication randomly chooses a segment of length L and inserts its copy to another random position. The two extant genomes then speciate independently from this ancestor genome. The speciation process on each branch includes randomly mixed S_2 segmental duplications and D DCJs. A DCJ randomly chooses two positions in the genome and then reverses the segment in between. We make sure that the expected number of genes per gene family in each extant genome is 1.5 (this number is comparable to that in human genome, which is 1.46), therefore we have that $S_1 + S_2 = 0.5 \cdot N/L$. We further fix $S_1 = 0.2 \cdot N/L$ and $S_2 = 0.3 \cdot N/L$ (we also test $S_1 = 0$ and $S_2 = 0.5 \cdot N/L$, and the results are almost the same). Thus, a simulation configuration is determined by parameters L and D.

For each pair of simulated genomes X and Y, we take them as input to run MSOAR and our method. For MSOAR, we run its binary version downloaded from http://msoar.cs.ucr.edu/. For our method, we first apply the preprocessing algorithm described in section 4.3, and then formulate the simplified problem as an ILP instance, which is solved using the GUROBI solver. We set the time limit to two hours for each instance, i.e, if the ILP solver does not return the

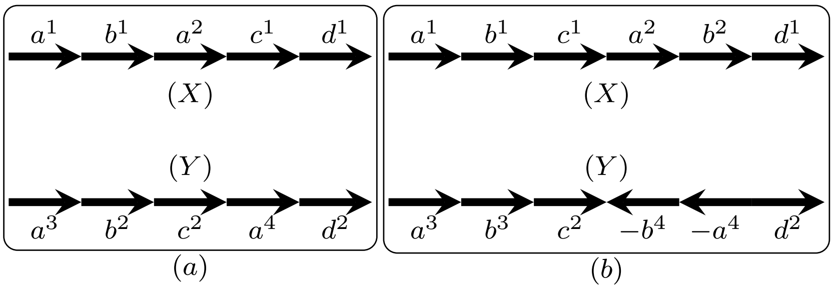

a^1 b^1 a^2 c^1 d^1
(X)
(Y)
a^3 b^2 c^2 a^4 d^2
(a)
a^1 b^1 c^1 a^2 b^2 d^1
(X)
(Y)
a^3 b^3 c^2 $-b^4$ $-a^4$ d^2
(b)

Figure 4.7 – (**a**) An example in which there are two optimal solutions if $c(\cdot) = 1$. (**b**) An example in which there are two optimal solutions if $c(\cdot) = 0.5$.

optimal solution in two hours, we terminate it and return the current sub-optimal solution.

Both methods return triples (S,T,B), where S and T infers the in-paralogs in the two extant genomes respectively, and B infers the orthology pairs. We now give the measures to evaluate them. First, we regard the problem to infer in-paralogs as a standard binary classification problem: those genes that are generated by segmental duplications in the speciation process are considered as gold standard positive in-paralogs, and those genes that are in the segments in $S \cup T$ are considered as predicted positive in-paralogs. Thus, we use the sensitivity and specificity to measure (S,T). To evaluate the performance of B, we refer to those gene pairs in the two extant genomes that correspond to the same gene in the ancestor genome as the *true* orthology pairs. We therefore use the following way to evaluate B: we say a pair in B is *assessable*, if at least one of its two genes can be found in some true orthology pair, and the *accuracy* of B is then defined as the ratio between the number of true orthology pairs in B and the number of assessable pairs in B.

For each parameter configuration, we simulate 10 instances and compute the average sensitivity, specificity and accuracy for both methods. The performance of the two methods is shown in Figure 4.8, Figure 4.9 and Figure 4.10, where the parameters $L \in \{1,2,5\}$ and D ranges from 250 to 2000. First, we can observe that both methods get very high sensitivity (above 90% on all configurations). However, MSOAR gets relatively low specificity. One reason for this is that MSOAR uses unit cost for both rearrangements and single-gene duplications. According to the discussion in section 4.4, unit cost for all operations might misclassify in-paralogs. Second, as D increases, the performance of both methods decreases. This is because the number of DCJs is highly positively correlated to the difficulty of the problem. When $D \leq 500$, i.e., roughly 10% of the size of the simulated genome (which is usually the case for real genomes, see Table 4.2 columns $d(B)$ for some examples), we can see that our method almost gets perfect

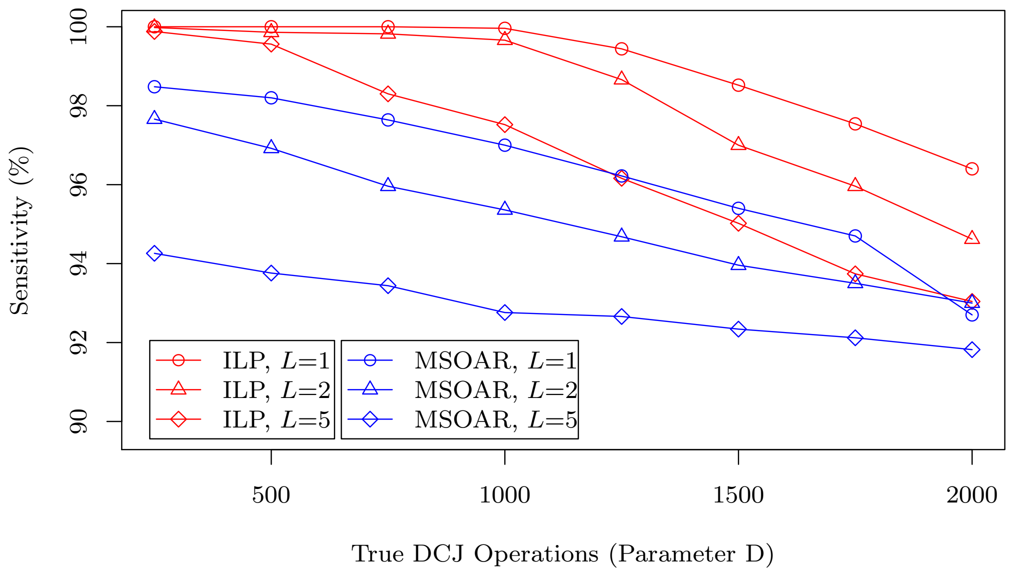


Figure 4.8 – Sensitivity of the inferred in-paralogs.

performance. Third, observe that MSOAR is very sensitive to L even when D is very small. This might be because the evolutionary model for in-paralogs in MSOAR is single-gene duplication, which creates trouble when genomes contain long segmental duplications. Finally, our method outperforms MSOAR on all the configurations.

4.5.2 Results on Biological Datasets

We compare both methods on five well-annotated mammalian species, human (*H.s.*), gorilla (*G.g.*), orangutan (*P.a.*), mouse (*M.m.*) and rat (*R.n.*). For each species, we collect all the

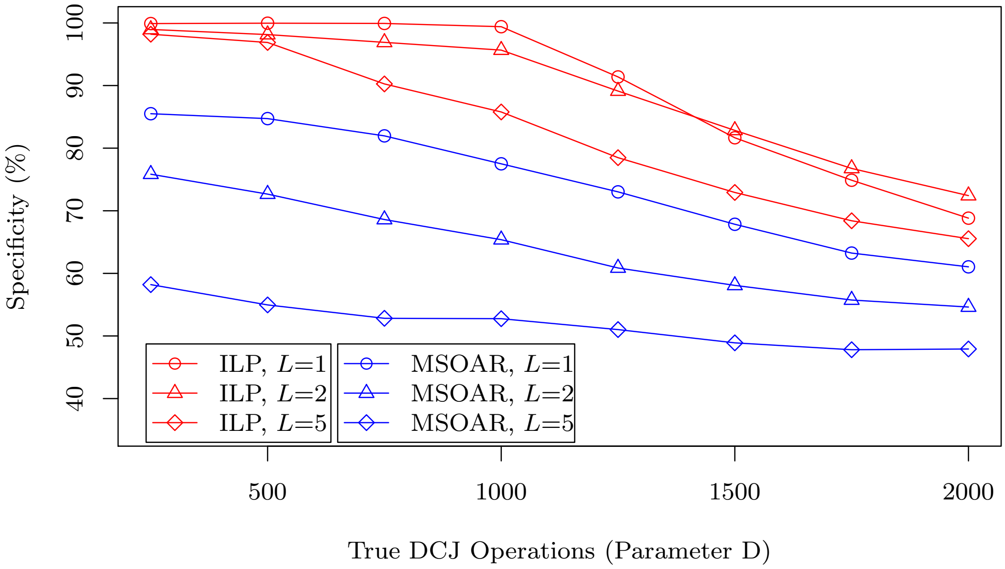


Figure 4.9 – Specificity of the inferred in-paralogs.

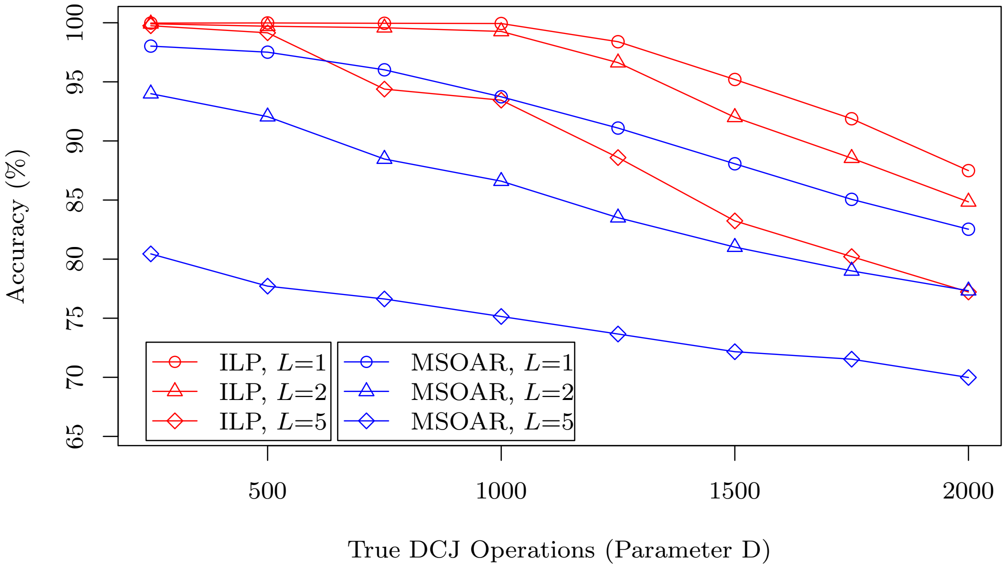


Figure 4.10 – Accuracy of the inferred orthologs.

protein-coding genes, and download their positions on the chromosomes and the Ensembl gene family names from Ensembl (http://www.ensembl.org). Two genes are considered as homologous if they have the same Ensembl gene family name. Since the tandemly arrayed genes (TAGs) have a different evolutionary model from segmental duplications, we merge each group of TAGs into only one gene through only keeping the first gene in the group while removing all the following ones.

We do the pairwise comparison for all five species, and for each pair of species, we run both methods to obtain triples (S, T, B). We use the same *accuracy* defined in section 4.5.1 to evaluate B. To compute the accuracy, we use the gene symbols (HGNC symbols for primate genes, MGI symbols for mouse genes, and RGD symbols for rat genes) to define true orthology pairs: those gene pairs that have the same gene symbol form the set of true orthology pairs for each pair of species. We do not have annotation data to serve as gold standard positive in-paralogs (we cannot just regard those genes that are not in the true orthology pairs as gold standard positive in-paralogs, since many genes have not yet been assigned a valid gene symbol). Thus we are not able to compute the sensitivity and specificity of (S, T).

The comparison on accuracy is shown in Table 4.1. We can observe that both methods have very high accuracy, indicating that the inferred orthology pairs from gene order data mostly agree with the annotations. On the other hand, our method gets higher accuracy than MSOAR on all pairs. The running time of MSOAR is also shown in Table 4.1. On average for each instance it takes 108 minutes, which is on the same level with our method (120 minutes).

In Table 4.2 we compare the number of operations and total score inferred by the two methods, to evaluate their ability as an optimizer. First, we can see that our method gets more segmental duplications and many fewer DCJs than MSOAR. One reason for this is that we use smaller cost

species pairs			assessable		accuracy		time
			MSOAR	ILP	MSOAR	ILP	
G.g.	&	*H.s.*	14898	14807	98.9%	**99.1**%	43
G.g.	&	*M.m.*	12946	12923	98.7%	**99.0**%	100
G.g.	&	*P.a.*	11308	11262	98.7%	**99.0**%	71
G.g.	&	*R.n.*	10831	10779	97.2%	**98.0**%	292
H.s.	&	*M.m.*	14030	13989	99.1%	**99.3**%	61
H.s.	&	*P.a.*	12004	11955	99.1%	**99.3**%	32
H.s.	&	*R.n.*	11748	11685	97.5%	**98.1**%	127
M.m.	&	*P.a.*	10574	10537	98.9%	**99.3**%	68
M.m.	&	*R.n.*	12332	12280	97.7%	**98.2**%	130
R.n.	&	*P.a.*	8788	8745	97.6%	**98.2**%	157

Table 4.1 – Comparison with MSOAR on accuracy. The two columns in the category of *assessable* show the number of assessable gene pairs in the bijections returned by MSOAR and our method respectively. The category of *accuracy* give the accuracy of the bijections returned by the two methods. The last column shows the running time of MSOAR (in minutes).

for segmental duplications. Second, our method gets smaller total cost on all the 10 pairs. This shows the advantage of our exact algorithm over the heuristic applied in MSOAR. Notice that the total cost shown in Table 4.2 is computed using our cost scheme, i.e., $d(B) + 0.75 \cdot (|S| + |T|)$, for both methods. However, if the total cost is computed using MSOAR's cost scheme, i.e., $d(B) + |S| + |T|$, our method still has less total cost on all pairs.

In Table 4.3 we analyze the distribution of the length of the inferred duplicated segments by our method. We can see that most of them are single-gene duplications. We can also observe that the rat genome contains more duplications than the other four genomes.

In Table 4.4 we analyze the composition of B returned by our method. If a gene family is a singleton in both genomes, then this pair of genes cannot be duplicated and must be mapped

species pairs	$\|S\|+\|T\|$		$d(B)$		total cost	
	MSOAR	ILP	MSOAR	ILP	MSOAR	ILP
G.g. & *H.s.*	1738	1962	670	361	1973.50	1832.50
G.g. & *M.m.*	2183	2369	1214	891	2851.25	2667.75
G.g. & *P.a.*	1985	2259	896	530	2384.75	2224.25
G.g. & *R.n.*	3389	3620	1969	1394	4510.75	4109.00
H.s. & *M.m.*	1320	1381	909	743	1899.00	1778.75
H.s. & *P.a.*	1336	1444	497	306	1499.00	1389.00
H.s. & *R.n.*	2897	2885	1366	1069	3538.75	3232.75
M.m. & *P.a.*	1731	1825	906	707	2204.25	2075.75
M.m. & *R.n.*	2621	2739	1176	763	3141.75	2817.25
R.n. & *P.a.*	3109	3208	1535	1101	3866.75	3507.00

Table 4.2 – Comparison with MSOAR on inferred operations and total score. The two columns in the category of $|S| + |T|$ give the number of segmental duplications predicted by MSOAR and our method respectively. The category of $d(B)$ give the DCJ distance induced by the bijection returned by the two methods. The last category gives the total cost of the triples returned by the two methods, computed as $0.75 \cdot (|S| + |T|) + d(B)$.

species pairs	S_1	S_2	$S_{\geq 3}$	$\|S\|$	T_1	T_2	$T_{\geq 3}$	$\|T\|$
G.g. & *H.s.*	98.7	1.2	0.0	1347	95.2	4.3	0.3	615
G.g. & *M.m.*	98.5	1.3	0.1	1421	96.7	2.9	0.3	948
G.g. & *P.a.*	97.9	1.8	0.1	1579	98.8	1.1	0.0	680
G.g. & *R.n.*	98.1	1.6	0.2	1377	94.9	3.7	1.2	2243
H.s. & *M.m.*	94.8	4.9	0.1	563	96.2	3.5	0.2	818
H.s. & *P.a.*	93.9	4.8	1.2	807	99.3	0.4	0.1	637
H.s. & *R.n.*	94.9	4.2	0.7	631	95.3	3.2	1.3	2254
M.m. & *P.a.*	96.0	3.3	0.6	1109	99.1	0.5	0.2	716
M.m. & *R.n.*	95.5	3.5	0.9	648	94.5	3.5	1.8	2091
R.n. & *P.a.*	95.2	3.5	1.2	2472	99.0	0.6	0.2	736

Table 4.3 – Distribution of the length of the inferred duplicated segments by our method, where S_k (resp. T_k) gives the percentage of the segments of length k in S (resp. T).

to each other by definition. We call such pair a *trivial* pair. Observe that roughly half of the pairs in B are trivial pairs (*trivial* column). We also show the percentage of the pairs that are fixed through the preprocessing algorithm (*predetermined* column). We can see that this preprocessing algorithm is very efficient, which can fix almost all the nontrivial pairs, leaving a very small portion (*remaining* column) that are to be determined by the ILP. This is because these species contain many shared segments and many isolated genes (because most of the segmental duplications are single-gene duplications), and thus there are many optimal substructures that can be identified by our algorithm.

species pairs	trivial	predetermined	remaining	$\|B\|$
G.g. & *H.s.*	51.5%	47.9%	0.5%	16213
G.g. & *M.m.*	48.7%	49.5%	1.6%	15015
G.g. & *P.a.*	50.7%	48.5%	0.7%	15271
G.g. & *R.n.*	46.3%	50.5%	3.0%	14983
H.s. & *M.m.*	51.0%	47.5%	1.3%	15572
H.s. & *P.a.*	52.3%	47.1%	0.4%	15481
H.s. & *R.n.*	48.5%	49.5%	1.8%	15379
M.m. & *P.a.*	50.0%	48.6%	1.2%	14620
M.m. & *R.n.*	48.9%	49.4%	1.6%	16347
R.n. & *P.a.*	47.7%	50.1%	2.0%	14534

Table 4.4 – Composition of the bijection returned by our method. The three columns in the middle are explained in the main text while the last column gives the total number of gene pairs in the bijection.

4.6 Discussion

The algorithm described in section 4.3.2 has potential to extend. For example, it can be directly used to test whether a general substructure, rather than a single PHFSS, is optimal. Moreover, we made a strong assumption that all genes in the related gene families are isolated, which immediately makes the cost of the segmental duplications trivial to compare, and thus allows us to focus on the number of cycles. In fact, we can relax this assumption, as long as we can guarantee that the segmental duplications induced by the substructure that is tested is optimal.

5 Approximating the Edit Distance with DCJs, Insertions and Deletions

In this chapter, we study the edit distance for genomes with duplicate genes under a model including DCJ operations, insertions and deletions. We prove that computing such edit distance is equivalent to finding the optimal cycle decomposition of the corresponding adjacency graph, and give an approximation algorithm with an approximation ratio of $(1.5+\epsilon)$.

5.1 Problem Statement

For a given genome, we call all its adjacencies and telomeres *adjacency set*. We define three operations on an adjacency set.

1. *DCJ (double-cut-and-join)* [11], which acts on one or two elements (adjacencies or telomeres) in one of the following ways:
   - $\{p,q\}+\{r,s\} \to \{p,r\}+\{q,s\}$, or $\{p,s\}+\{q,r\}$;
   - $\{p,q\}+\{r\} \to \{p,r\}+\{q\}$, or $\{p\}+\{q,r\}$;
   - $\{p,q\} \to \{p\}+\{q\}$;
   - $\{p\}+\{q\} \to \{p,q\}$.
2. *Insertion*, which inserts a single gene g in one of the following ways:
   - $\{p,q\} \to \{p,g_h\}+\{q,g_t\}$, or $\{p,g_t\}+\{q,g_h\}$;
   - $\{p\} \to \{p,g_h\}+\{g_t\}$, or $\{p,g_t\}+\{g_h\}$;
   - $\emptyset \to \{g_h,g_t\}$;
   - $\emptyset \to \{g_h\}+\{g_t\}$.
3. *Deletion*, which deletes a single gene g in one of the following ways:
   - $\{p,g_h\}+\{p,g_t\} \to \{p,q\}$;
   - $\{p,g_h\}+\{g_t\} \to \{p\}$;

- $\{g_h, g_t\} \to \emptyset$;
- $\{g_h\} + \{g_t\} \to \emptyset$.

Under a model that includes the above three operations, the *edit distance* between two adjacency sets S_1 and S_2, denoted as $d(S_1, S_2)$, is defined as the minimum number of operations that can transform S_1 into S_2. Note that the edit distance is defined at the adjacency set level. For genomes without duplicate genes, an adjacency set denotes a unique genomic structure. However, for genomes with duplicate genes, two genomes with different structures may share the same adjacency set as illustrated in Figure 5.1. Thus, $d(S_1, S_2)$ defined above is a lower bound for the edit distance between the two genomic structures.

Given two adjacency sets S_1 and S_2 from two genomes, let E_1 and E_2 be the multiset of extremities collected from all elements in S_1 and S_2 respectively. Let $T_1 = E_1 \setminus E_2$ and $T_2 = E_2 \setminus E_1$. Notice that all extremities in T_1 (resp. T_2) comes from $|T_1|/2$ (resp. $|T_2|/2$) genes, since S_1 and S_2 are the adjacency sets obtained from genomes. Thus we can pair extremities in T_1 (resp. T_2) into $|T_1|/2$ (resp. $|T_2|/2$) *ghost adjacencies*, each of which contains two extremities comes from a single gene. Clearly, to transform S_1 into S_2, at least $|T_1|/2$ deletions and $|T_2|/2$ insertions are needed. The following theorem shows that these insertions and deletions are both necessary and sufficient.

Theorem 5.1.1. *Given two adjacency sets S_1 and S_2, there exists an optimal series of operations with exactly $|T_1|/2$ deletions, exactly $|T_2|/2$ insertions and some DCJ operations that transforms S_1 into S_2.*

Proof. We prove this theorem by contradiction. Suppose that every optimal series of operations contains more than $|T_1|/2$ deletions and more than $|T_2|/2$ insertions. Assume that $O_1 O_2 \cdots O_m$ is an optimal series of operations that contains a minimum number of insertions and deletions. Let $S^0 S^1 S^2 \cdots S^m$ be the trace of S_1 in the process of transformation, where $S^0 = S_1$ and $S^m = S_2$. Note that for any insertion (or deletion) beyond the $|T_1|/2$ deletions and $|T_2|/2$ insertions, there must be a matching deletion (or insertion) to preserve gene content. Thus every optimal series of operations has at least a pair of insertion and deletion on the

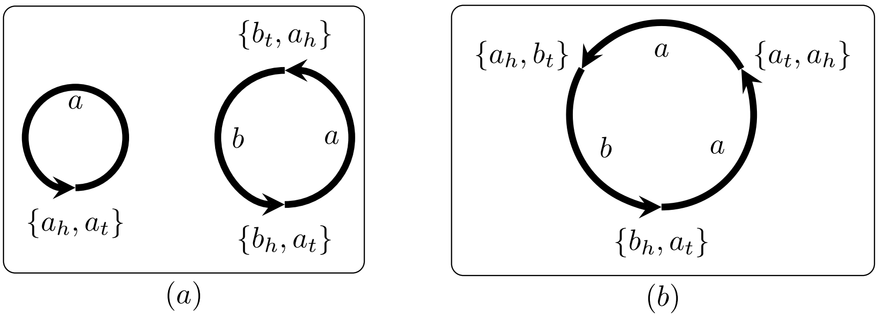


Figure 5.1 – An example illustrating that two genomes with different structures share the same adjacency set of $\{\{a_h, a_t\}, \{b_h, a_t\}, \{b_t, a_h\}\}$. In this chapter, we do not distinguish genes from the same gene family.

same gene. Without loss of generality, assume O_i inserts a pair of extremities $\{g_h, g_t\}$ and O_j deletes $\{g_h, g_t\}$ ($i < j$), and operations between O_i and O_j do not contain insertion or deletion on $\{g_h, g_t\}$. Now we will build a new series of operations $O'_i O'_{i+1} \cdots O'_j$ without the pair of insertion and deletion on $\{g_h, g_t\}$ to replace $O_i \cdots O_j$, which produce the trace $S^{i'} S^{i+1'} \cdots S^{j'}$ and satisfy $S^{j'} = S^j$. This process is shown in Figure 5.2.

Denote by g_h^* and g_t^* the two extremities inserted in O_i to distinguish them from other g_h and g_t. For $k = i, \cdots, j-1$, we will keep the invariant $S^{k-1'} = (S^k \setminus \{\{p^k, g_h^*\}, \{q^k, g_t^*\}\}) \cup \{\{p^k, q^k\}\}$, where p^k (resp. q^k) is the extremity that shares an adjacency with g_h^* (resp. g_t^*) in S^k. Note that p^k or q^k might be empty if g_h^* or g_t^* forms a telomere, or $\{g_h^*, g_t^*\}$ forms an adjacency in S^k. Clearly this holds for $k = i$, since we have both $S^{i-1'} = S^{i-1}$ and $S^i = (S^{i-1} \setminus \{\{p^i, q^i\}\}) \cup \{\{p^i g_h^*, q^i g_t^*\}\}$. To make this invariant hold for $k = i+1, \cdots, j-1$, our new operation O'_{k-1} will mimic operation O_k as follows: if O_k does not affect the adjacencies or telomeres containing g_h^* or g_t^*, then set $O'_{k-1} = O_k$, and the invariant holds; if O_k acts on at least one of g_h^* or g_t^*, we will build O'_{k-1} from O_k by replacing g_h^* (resp. g_t^*) with p^k (resp. q^k) in O_k. For example, if O_k is the DCJ operation given by $\{p^{k-1}, g_h^*\} + \{c, d\} \rightarrow \{p^{k-1}, c\} + \{g_h^*, d\}$, then O'_{k-1} would be $\{p^{k-1}, q^{k-1}\} + \{c, d\} \rightarrow \{p^{k-1}, c\} + \{q^{k-1}, d\}$. Since O_k does not affect g_t^*, we have $q^k = q^{k-1}$. Besides, we have $p^k = d$. Thus we have $S^k \setminus \{\{p^k, g_h^*\}, \{q^k, g_t^*\}\} \cup \{\{p^k, q^k\}\} = S^{k-1'}$. Other types of operations can be expressed similarly.

Recall that O_j is a deletion, i.e., $\{a, g_h\} + \{b, g_t\} \rightarrow \{a, b\}$. If g_h and g_t are the same as g_h^* and g_t^*, then we have $S^{j-2'} = S^j$, and we can skip O'_{j-1} and O'_j in our constructed series. If g_h and g_t are different from g_h^* and g_t^*, then we have $\{\{a, g_h\}, \{b, g_t\}, \{p^{j-1}, g_h^*\}, \{q^{j-1}, g_t^*\}\} \subset S^{j-1}$. We can set O'_{j-1} to be $\{a, g_h\} + \{b, g_t\} \rightarrow \{a, b\} + \{g_h, g_t\}$, and set O'_j to be $\{p^{j-1}, q^{j-1}\} + \{g_h, g_t\} \rightarrow \{p^{j-1}, g_h\} + \{q^{j-1}, g_t\}$. We can verify $S^{j'} = S^j$, and our constructed series contradicts the optimality of $O_1 O_2 \cdots O_m$. □

5.2 Adjacency Graph Decomposition

Given two adjacency sets S_1 and S_2 from two genomes with E_1 and E_2 be their corresponding extremity sets, their corresponding *adjacency graph* is defined as a bipartite multigraph,

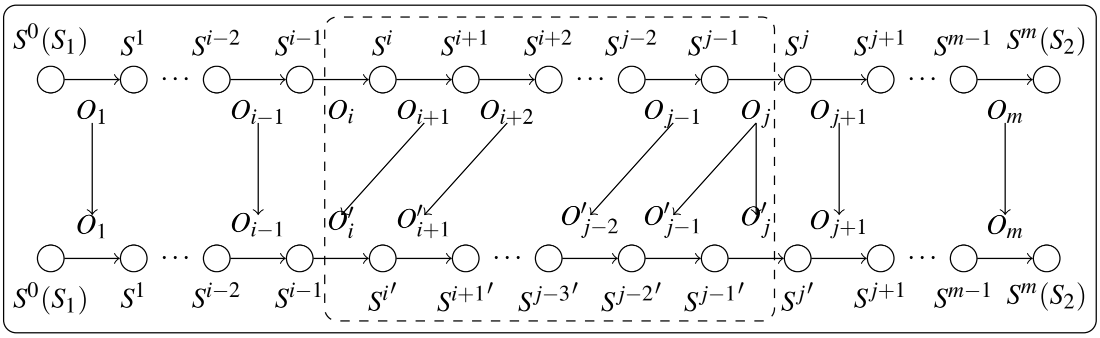


Figure 5.2 – Building a new series of operations to replace $O_i O_{i+1} \cdots O_j$.

$A = \{E_1 \cup T_2, E_2 \cup T_1, E\}$, in which $u \in E_1 \cup T_2$ and $v \in E_2 \cup T_1$ are linked by a *black* edge if u and v are the same extremity, and $u, v \in E_1 \cup T_2$ (resp. $u, v \in E_2 \cup T_1$) are linked by a *gray* edge if u and v form an adjacency in S_1 (resp. S_2) or they form a ghost adjacency in T_2 (resp. T_1). Note that $E_1 \cup T_2$ and $E_2 \cup T_1$ have the same set of extremities; we use n to denote half of the number of extremities, i.e., $n = |E_1 \cup T_2|/2$.

In the case of genomes with the same gene content and without duplicate genes, $T_1 = T_2 = \emptyset$, and each vertex in the adjacency graph has degree 2, which means that the adjacency graph consists of vertex-disjoint cycles and paths. We define the *length* of a cycle or a path to be the number of black edges it contains. Based on Theorem 5.1.1, $T_1 = T_2 = \emptyset$ implies there exists an optimal solution without insertion and deletion, thus $d(S_1, S_2)$ is just the minimum number of DCJ operations needed to transform S_1 into S_2. When S_1 has been transformed into S_2, the corresponding adjacency graph only consists of cycles of length 2 and paths of length 1. Since each DCJ operation can increase the number of cycles at most by 1, or increase the number of odd-length paths at most by 2, and we can always find out this kind of operation when S_1 and S_2 are different, we have $d(S_1, S_2) = n - c - o/2$, where c is the number of cycles and o is the number of odd-length paths in the corresponding adjacency graph [11].

In the presence of duplicate genes, the adjacency graph may contain vertices with degree larger than 2, so that there may be multiple ways of choosing vertex-disjoint cycles and paths that cover all vertices (see Figure 5.3). We say that a cycle (or path) in the adjacency graph is *alternating* if two adjacent edges in this cycle (or path) contain exactly a gray edge and a black edge. A valid *decomposition* of the adjacency graph is a set of vertex-disjoint alternating cycles and paths that cover all vertices. We say that a cycle of length ℓ is *helpful* if at most $(\ell/2 - 1)$ vertices are ghost adjacencies, *unhelpful* if at least $\ell/2$ vertices are ghost adjacencies. In fact, an unhelpful cycle has exactly $\ell/2$ ghost adjacencies (all in T_1 or all in T_2), since ghost adjacencies from T_1 and ghost adjacencies from T_2 do not have common extremities and thus cannot be linked in the adjacency graph. Now we show how to perform DCJ operations, insertions and

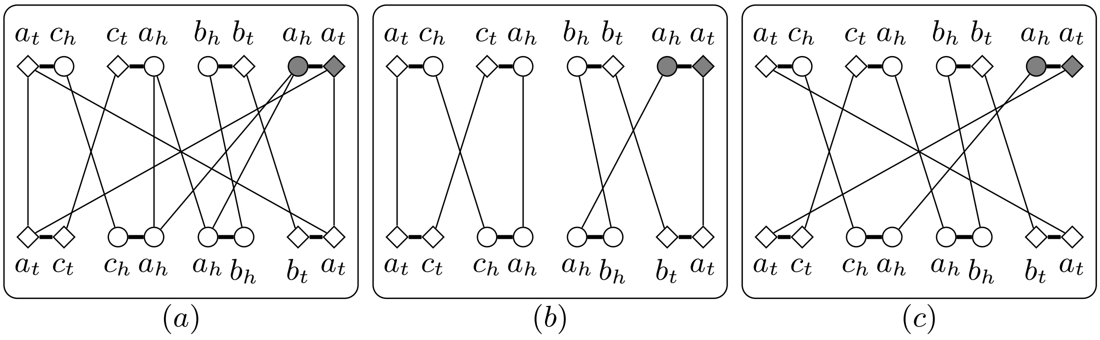

a_t c_h c_t a_h b_h b_t a_h a_t
a_t c_t c_h a_h a_h b_h b_t a_t
(a)
(b)
(c)

Figure 5.3 – An example of adjacency graph with duplicate genes. G_1 contains two circular chromosomes of (a, c) and (b), while G_2 contains two circular chromosomes of $(a, -c)$ and $(a, -b)$. Head extremities are represented by circles and tail extremities are represented by diamonds. Extremities in ghost adjacencies are then filled gray. (**a**) Adjacency graph w.r.t. G_1 and G_2. (**b**) A decomposition with 2 cycles. (**c**) A decomposition with only 1 cycle.

deletions to transform S_1 into S_2 based on a decomposition of the corresponding adjacency graph.

Lemma 5.2.1. *Given two adjacency sets S_1 and S_2 from two genomes, and a decomposition D of the corresponding adjacency graph with c helpful cycles and o odd-length paths, we can perform $(n - c - o/2)$ operations to transform S_1 into S_2, among which there are $|T_1|/2$ deletions, $|T_2|/2$ insertions and $(n - c - o/2 - |T_1|/2 - |T_2|/2)$ DCJ operations.*

Proof. We prove this lemma in a constructive way. We will perform operations under the guidance of the given decomposition. The goal is to transform the adjacency graph into a collection of cycles of length 2 and paths of length 1 without ghost adjacencies, indicating that S_1 has been transformed into S_2. In the following, we will prove that an unhelpful cycle of length ℓ costs $\ell/2$ operations, a path of even length ℓ costs $\ell/2$ operations, a helpful cycle of length ℓ costs $(\ell/2 - 1)$ operations, and a path of odd length ℓ costs $(\ell - 1)/2$ operations. In other words, a helpful cycle requires one less operation than an unhelpful cycle or an even-length path of the same length.

For a helpful cycle of length ℓ with d ghost adjacencies from T_1 and i ghost adjacencies from

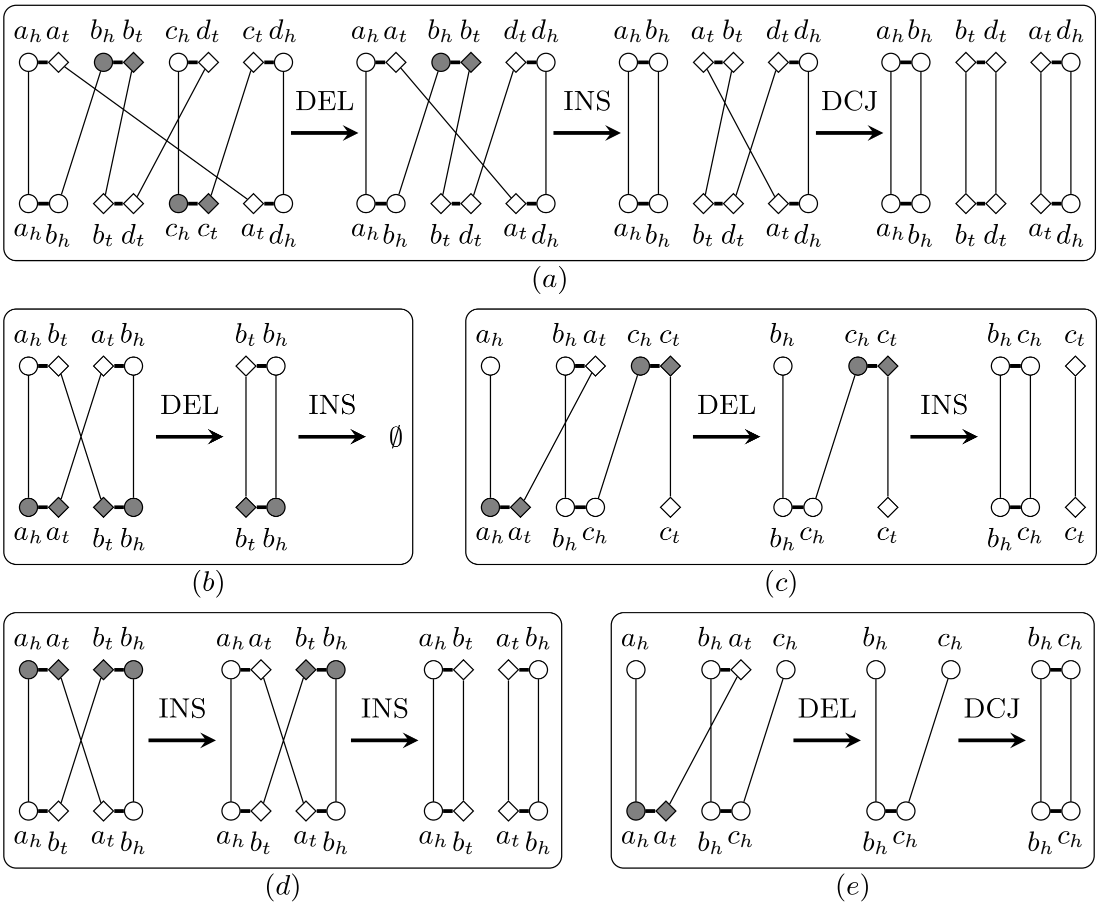


Figure 5.4 – Examples of performing operations under the guidance of decomposition.

T_2, we first perform d deletions guided by this cycle to reduce the size of the cycle to $(\ell - 2d)$. Then for each ghost adjacency from T_2, we choose one of its non-ghost neighbors in S_1 and perform an insertion to create one more helpful cycle of length 2. After all ghost adjacencies from T_2 are handled, we transform the cycle of length ℓ into one of length $(\ell - 2d - 2i)$ without ghost adjacencies, on which finally we can perform $(\ell/2 - d - i - 1)$ DCJ operations to create $(\ell/2 - d - i)$ cycles of length 2. An example is shown in Figure 5.4**(a)**.

For a unhelpful cycle of length ℓ with $\ell/2$ ghost adjacencies from T_1, we can perform $\ell/2$ deletions to remove the adjacencies in S_1. For a unhelpful cycle of length ℓ with $\ell/2$ ghost adjacencies from T_2, we can first insert a gene as initial operand, then perform $(\ell/2 - 1)$ insertions to create $\ell/2$ cycles of length 2—see Figure 5.4**(b)(d)**.

For a path with odd length ℓ, we need $(\ell - 1)/2$ operations, and for a path with even length ℓ, we need $\ell/2$ operations—see Figure 5.4**(c)(e)**.

In sum, we need $|T_1|/2$ deletions, $|T_2|/2$ insertions and $(n - c - o/2 - |T_1|/2 - |T_2|/2)$ DCJ operations to transform S_1 into S_2. □

Lemma 5.2.1 states that each decomposition gives an upper bound on the edit distance. The following lemma shows that an optimal decomposition also provides a lower bound.

Lemma 5.2.2. *We have $d(S_1, S_2) \geq n - \max_{D \in \mathscr{D}}(c_D + o_D/2)$, where $\mathscr{D}$ is the space of all decompositions of the corresponding adjacency graph, c_D and o_D is the number of helpful cycles and odd-length paths in D, respectively.*

Proof. Let $\Delta_P = \max_{D \in \mathscr{D}''}(c_D + o_D/2) - \max_{D \in \mathscr{D}'}(c_D + o_D/2)$, where $\mathscr{D}'$ and $\mathscr{D}''$ are the space of the decomposition before and after performing operation P, and $P \in \{DCJ, INS, DEL\}$. By Theorem 5.1.1, there exists an optimal series of operations with exactly $|T_1|/2$ deletions and $|T_2|/2$ insertions. Summing over all Δ_P for these operations in this optimal solution yields $\sum_{i=1}^{d(S_1,S_2)} \Delta_{P_i} = (n - |T_1|/2) - \max_{D \in \mathscr{D}}(c_D + o_D/2)$, where $(n - |T_1|/2)$ is the sum of the number of helpful cycles and half of the number of odd-length paths in the optimal decomposition of the adjacency graph when S_1 has been transformed into S_2. Define $\delta_{DCJ} = 1$, $\delta_{INS} = 1$ and $\delta_{DEL} = 0$. In the following, we will prove $\Delta_P \leq \delta_P$, $P \in \{DCJ, INS, DEL\}$, which implies that $\sum_{i=1}^{d(S_1,S_2)} \Delta_{P_i} \leq d(S_1, S_2) - |T_1|/2$. The combination of these two formulas proves this lemma.

We prove $\Delta_P \leq \delta_P$ by contradiction. Let A' and A'' be the adjacency graphs before and after performing the operation P. Let $\sigma(A')$ and $\sigma(A'')$ be the optimal decompositions of A' and A'', respectively. Suppose $\Delta_P > \delta_P$, namely, $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\sigma(A')} + o_{\sigma(A')}) > \delta_P$. Note that P is reversible; we denote the reversed operation as $\hat{P}$, and $\hat{P}$ simultaneously transforms $\sigma(A'')$ into a decomposition of A', denoted $\gamma(A')$. Since $\sigma(A')$ is optimal, we have $c_{\sigma(A')} + o_{\sigma(A')}/2 \geq c_{\gamma(A')} + o_{\gamma(A')}/2$. Thus, to get the contradiction, we only need to prove $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\gamma(A')} + o_{\gamma(A')}/2) \leq \delta_P$. Recall that $\gamma(A')$ is obtained from $\sigma(A'')$ by performing $\hat{P}$, and both $\sigma(A'')$ and $\gamma(A')$ are decompositions, which includes only vertex-disjoint cycles and paths.

If P is a DCJ operation, then $\hat{P}$ is still a DCJ operation. A DCJ operation may merge two cycles into one cycle, split one cycle into two cycles, merge two paths into one path, split one path into two paths, merge one path and one cycle into one path, split one path into one cycle and one path, rearrange two odd (resp. even)-length paths into two even (resp. odd) paths or make no change in the number of cycles and odd-length paths. Among those possible operations, the following four cases can reduce the number of helpful cycles or odd-length paths: (i) merge two helpful cycles into one helpful cycle; (ii) merge two odd-length paths into one even-length path; (iii) rearrange two odd-length paths into two even-length paths; (iv) merge one helpful cycle and one odd-length path into one odd-length path. For any of these four cases, we have $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\gamma(A')} + o_{\gamma(A')}/2) = 1$. For other possible DCJ operations, we have $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\gamma(A')} + o_{\gamma(A')}/2) \leq 0$.

If P is an insertion, then $\hat{P}$ is a deletion. Similarly, among all possible deletions, the following five cases can reduce the number of helpful cycles or odd-length paths: (i) merge two helpful cycles into one helpful cycle; (ii) merge two odd-length paths into one even-length path; (iii) rearrange two odd-length paths into two even-length paths; (iv) merge one helpful cycle and one odd-length path into one odd-length path; (v) change a helpful cycle into an unhelpful one. For any of these five cases, we have $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\gamma(A')} + o_{\gamma(A')}/2) = 1$. For other possible deletions, we have $(c_{\sigma(A'')} + o_{\sigma(A'')}/2) - (c_{\gamma(A')} + o_{\gamma(A')}/2) \leq 0$.

If P is a deletion, then $\hat{P}$ is an insertion. A insertion may split one cycle into two cycles, split one path into two paths, or split one path into one cycle and one path. All these possible insertions will not reduce the number of helpful cycles or odd-length paths. Thus, any deletion will not increase the number of helpful cycles or the number of odd-length paths, and we have $c_{\sigma(A'')} + o_{\sigma(A'')}/2 \leq c_{\gamma(A')} + o_{\gamma(A')}/2$. □

Combining Lemma 5.2.1 and Lemma 5.2.2, we have the following theorem.

Theorem 5.2.1. *We have $d(S_1, S_2) = n - \max_{D \in \mathscr{D}}(c_D + o_D/2)$, where $\mathscr{D}$ is the space of all decompositions of $A = \{E_1 \cup T_2, E_2 \cup T_1, E\}$, c_D and o_D are the numbers of helpful cycles and odd-length paths in D, respectively.*

5.3 Approximation Algorithm

We design an approximation algorithm by using techniques employed on the problem of Breakpoint Graph Decomposition [62, 15, 63, 64]. The basic idea is to find the maximum number of vertex-disjoint helpful cycles of length 4 in the adjacency graph. This problem can be reduced to the problem of k-Set Packing problem with $k = 4$, for which the best-to-date algorithm has an approximation ratio of $(2 + \epsilon)$ [65, 66].

To make use of such algorithm, we must remove telomeres and keep only cycles in the adjacency graph. This can be done by introducing *null extremity* τ and *null adjacency* $\{\tau, \tau\}$, which are different from other extremities and adjacencies (similar definition is introduced in [13]). Given two adjacency sets S_1 and S_2 with $2 \cdot k_1$ and $2 \cdot k_2$ telomeres respectively, we replace each telomere $\{x\}$ by the adjacency $\{x,\tau\}$. If we additionally have $k_1 < k_2$, we must add $(k_2 - k_1)$ null adjacencies $\{\tau,\tau\}$ to S_1 in order to balance the degrees. The corresponding adjacency graph is constructed in the same way as the case without null extremities. Now we prove that this "telomere-removal" process does not change $d(S_1,S_2)$.

Theorem 5.3.1. *Let S_1 and S_2 be two adjacency sets and denote by S_1' and S_2' the adjacency sets obtained from S_1 and S_2 by removing telomeres. Then we can write $d(S_1,S_2) = d(S_1',S_2')$.*

Proof. We first prove $d(S_1,S_2) \geq d(S_1',S_2')$. Let $A = \{E_1 \cup T_2, E_2 \cup T_1, E\}$ be the adjacency graph with respect to S_1 and S_2 and $\sigma(A)$ be the optimal decomposition of A. Let $A' = \{E_1' \cup T_2, E_2' \cup T_1, E\}$ be the adjacency graph with respect to S_1' and S_2' and $\sigma(A')$ be the optimal decomposition of A'. Suppose $\sigma(A)$ contains c helpful cycles, o odd-length paths and e even-length paths, and among these e even-length paths, e_1 of them contain two telomeres in S_1 and e_2 of them contain two telomeres in S_2. Suppose S_1 and S_2 contains $2 \cdot k_1$ and $2 \cdot k_2$ telomeres respectively (without loss of generality, we assume that $k_1 \leq k_2$). Since an odd-length path contains one telomere in each adjacency set while an even-length path contains two telomeres in one adjacency set, we have $o + 2 \cdot e_1 = 2 \cdot k_1$ and $o + 2 \cdot e_2 = 2 \cdot k_2$. We can perform the following modifications on $\sigma(A)$ to transform it into a decomposition of A' (see Figure 5.5). Nothing needs to be done for cycles. For odd-length paths, link their two telomeres to form a helpful cycle; for each even-length path with both telomeres in S_1, arbitrarily choose one even-length path with both telomeres in S_2 and link these two paths to form a helpful cycle; for the remaining $(e_2 - e_1)$ even-length paths, use $e_2 - e_1 = k_2 - k_1$ null adjacencies $\{\tau,\tau\}$ to transform

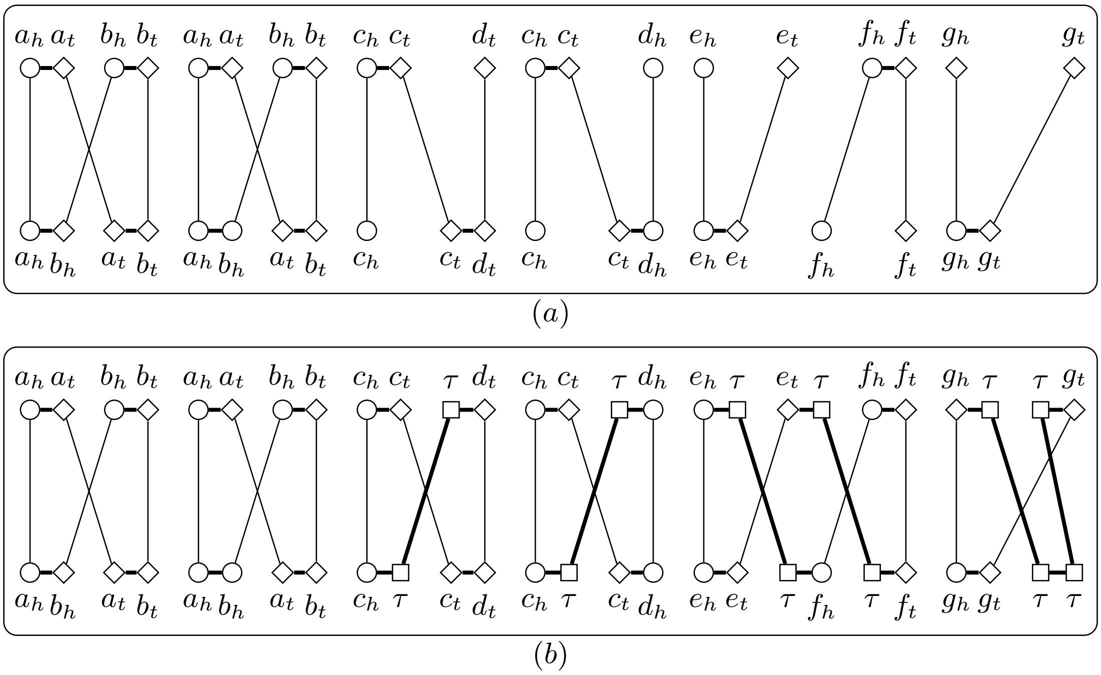

a_h a_t b_h b_t a_h a_t b_h b_t c_h c_t d_t c_h c_t d_h e_h e_t f_h f_t g_h g_t
a_h b_h a_t b_t a_h b_h a_t b_t c_h c_t d_t c_h c_t d_h e_h e_t f_h f_t g_h g_t
(a)
a_h a_t b_h b_t a_h a_t b_h b_t c_h c_t τ d_t c_h c_t τ d_h e_h τ e_t τ f_h f_t g_h τ τ g_t
a_h b_h a_t b_t a_h b_h a_t b_t c_h τ c_t d_t c_h τ c_t d_h e_h e_t τ f_h τ f_t g_h g_t τ τ
(b)

Figure 5.5 – Illustration of the "telomere-removal" process (from (**a**) to (**b**)) and "telomere-recovery" process (from (**b**) to (**a**)).

each such path into a helpful cycle. Thus, there are $(c+e_2)$ helpful cycles in this decomposition of A', so that the upper bound on $d(S'_1,S'_2)$ is $(n+k_2)-c-e_2 = n-c-o/2 = d(S_1,S_2)$.

Now we prove $d(S_1,S_2) \le d(S'_1,S'_2)$. Note that $\sigma(A')$ only consists of vertex-disjoint cycles, and unhelpful cycles cannot contain any null extremity. We claim that, for each helpful cycle in $\sigma(A')$, there is at most two null extremities τ on each side. Otherwise, we can always choose two nonadjacent edges that are linked through τ, exchange four ends of them, and divide this cycle into two (see Figure 5.6), contradicting the optimality of $\sigma(A')$. Now we transform $\sigma(A')$ into a decomposition of A by recovering all removed telomeres (see Figure 5.5). Each cycle falls into one of three cases: (a) it contains one $\{x,\tau\}$ adjacency on each side, then the recovery will yield one odd-length path; (b) it contains one $\{\tau,\tau\}$ adjacency on one side, then the recovery will yield one even-length path; (c) it contains two $\{x,\tau\}$-like adjacencies on each side, then the recovery will yield two even-length paths. In all cases $(n-c-o/2)$ remains unchanged, and after the recovery we obtain a decomposition of A. Thus we have $d(S_1,S_2) \le d(S'_1,S'_2)$. □

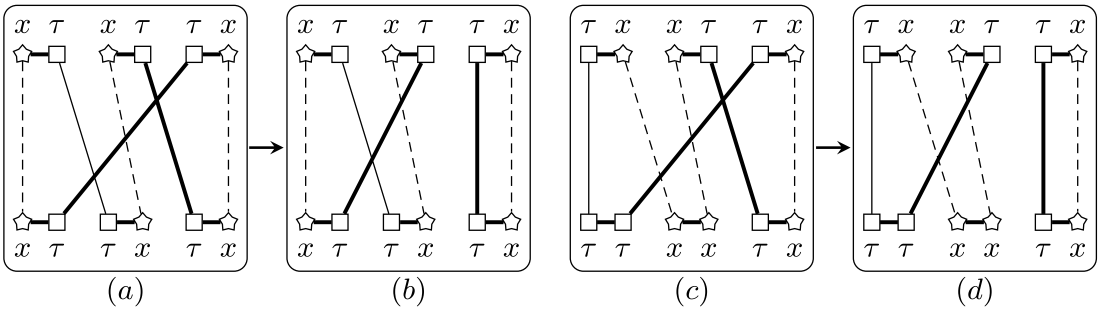


Figure 5.6 – Two cases of the adjacency graph with more than 2 edges linked through τ. Stars represent unrelated extremities, and dashed lines might represent more than one edge.

In summary, based on Theorems 5.2.1 and 5.3.1, we have stated the equivalence of the problem of computing the edit distance and that of finding a valid decomposition with a maximum number of helpful cycles in an adjacency graph without telomeres. The latter problem is NP-hard by a reduction from the NP-hard problem of Breakpoint Graph Decomposition [67], since any instance of the Breakpoint Graph Decomposition is indeed an adjacency graph without ghost adjacencies. Thus, the problem of computing the edit distance is also NP-hard.

Now we give the approximation algorithm and prove that its approximation ratio is $(1.5+\epsilon)$.

Approximation Algorithm
Input: Two adjacency sets S_1 and S_2 from two genomes
Output: A series of operations to transform S_1 into S_2.
Step 1 Add null adjacencies to S_1 and S_2 to obtain S'_1 and S'_2 without telomeres. Build the adjacency graph $A' = \{E'_1 \cup T_2, E'_2 \cup T_1, E\}$, where E'_1 and E'_2 are extremity sets of S'_1 and S'_2 respectively.
Step 2 Collect all helpful cycles of length 4 in A' as $\mathscr{C}$. Find a subset $\mathscr{S}$ of $\mathscr{C}$ in which no two cycles share the same extremity using the $(2+\epsilon)$-approximation algorithm for the k-Set

Packing problem with $k = 4$.
Step 3 Remove the adjacencies covered by cycles in $\mathscr{S}$. Arbitrarily decompose the remaining part of A' into cycles, denoting this set as $\mathscr{S}'$.
Step 4 Remove the null adjacencies of cycles in $\mathscr{S} \cup \mathscr{S}'$ to obtain a decomposition of A. Transform S_1 into S_2 according to Lemma 5.2.1 guided by these cycles and paths.

The running time of the above algorithm is dominated by the time complexity of the $(2+\epsilon)$-approximation algorithm for the k-Set Packing problem with $k = 4$, which is $O(|\mathscr{C}|^{\log_4 1/\epsilon})$ and $|\mathscr{C}| = O(n^4)$ [65, 66].

Theorem 5.3.2. *The approximation ratio of the above algorithm is* $(1.5+\epsilon)$.

Proof. Suppose the optimal decomposition of A' contain p helpful cycles of length 4 and q longer helpful cycles. Clearly, we have $n \geq 2 \cdot p + 3 \cdot q$. Based on Theorem 5.2.1 and Theorem 5.3.1, we know that $d(S_1, S_2) = n - p - q$. In the algorithm, we find at least $|\mathscr{S}|$ helpful cycles, which implies that the number of operations that our algorithm outputs is at most $(n - |\mathscr{S}|)$. Since $\mathscr{S}$ is a $(2+\epsilon)$-approximation solution, we have $(2+\epsilon) \cdot |\mathscr{S}| \geq OPT \geq p$, where OPT is the maximum number of independent helpful cycles of length 4 in $\mathscr{C}$. The approximation ratio is thus

$$r \leq \frac{n - |\mathscr{S}|}{n-p-q} \leq \frac{n - \frac{p}{2+\epsilon}}{n-p-q} \leq 1 + \frac{p+q-\frac{p}{2+\epsilon}}{n-p-q} \leq 1 + \frac{p+q-\frac{p}{2+\epsilon}}{2p+3q-p-q} \leq 1.5+\epsilon.$$

□

6 DCJ Median Distance

In this chapter, we study the *DCJ median problem*: given three genomes without duplicate genes, we want to construct a fourth genome, the *median*, that minimizes the sum of the DCJ distances between itself and each of the three given genomes. We contribute to this problem in three aspects. First, we describe a new strategy to find so-called adequate subgraphs in the multiple breakpoint graph [68], using a seed genome, which is a key step in computing the median. We show how to compute adequate subgraphs w.r.t. this seed genome using a network flow formulation. Second, we prove that the upper bound of the median distance computed from the triangle inequality is tight. Finally, we study the question of whether the median distance can reach its lower and upper bounds. We derive a necessary and sufficient condition for the median distance to reach its lower bound and a necessary condition for it to reach its upper bound and design algorithms to test for these conditions.

6.1 Preliminaries

We assume that each genome consists of the same set of n distinct genes and that those genes form one or more circular chromosomes in each genome. The head and tail of a gene g, represented by g_h and g_t, are called *extremities*. Two consecutive genes form one *adjacency*, represented as the set of its two extremities. Since all genes are distinct, each genome is uniquely determined by its n adjacencies. We build a graph (V, E), where V has $2 \cdot n$ vertices representing the extremities and E has n edges representing the adjacencies. Note that a genome thus corresponds to a perfect matching on V (see Fig. 6.1).

Given genomes G_1 and G_2 represented by perfect matchings M_1 and M_2 on V, the corresponding *breakpoint graph* is defined as the multigraph (V, M_1, M_2). In the multigraph, two vertices may be connected by two edges, one from M_1 and the other from M_2. These edges are distinguished by their provenance. Each vertex in this breakpoint graph has degree 2, so that the graph consists of vertex-disjoint cycles; let $c(M_1, M_2)$ denote the number of these cycles. The *DCJ distance* between G_1 and G_2, denoted as $d(M_1, M_2)$, can be expressed as $d(M_1, M_2) = n - c(M_1, M_2)$ [11]. We can extend this concept to three given genomes, M_1, M_2

and M_3, yielding a *multiple breakpoint graph* (MBG for short, see an example in Fig. 6.1), denoted by (V, M_1, M_2, M_3). Given a MBG (V, M_1, M_2, M_3), the DCJ median problem asks for a perfect matching M_0 on V (another genome) that minimizes $\sum_{k=1}^{3} d(M_0, M_k)$.

Now we generalize the definition of MBG by allowing nonperfect matchings, and we call those MBGs with three perfect matchings *complete* MBGs. Notice that if M_1' and M_2' are not perfect matchings on V', then the breakpoint graph (V', M_1', M_2') consists of isolated vertices, simple paths, and vertex-disjoint cycles; we still use $c(M_1', M_2')$ to denote the number of cycles.

Let B' be a MBG and B a complete MBG. We say B' is a *subgraph* of B if we have $V' \subseteq V$ and $M_k' \subseteq M_k$, $k = 1, 2, 3$. A matching M_0' on V' is a *median* of B' if it maximizes $\sum_{k=1}^{3} c(M_0', M_k')$ over all possible matchings on V'. We say B' is *adequate* if for any median M_0' of B' we have $\sum_{k=1}^{3} c(M_0', M_k') \geq 3 \cdot |V'|/4$.

Theorem 6.1.1. *[49] If B' is an adequate subgraph of B, then for any median M_0' of B', there exists one median M_0 of B such that $M_0' \subset M_0$.*

This result leads to a decomposition scheme to compute the median by iteratively finding adequate subgraphs and resolving each separately. To find adequate subgraphs in the complete MBG, ASMedian uses a precomputed set containing all adequate subgraphs with size less than 10, looking in turn for each subgraph in the complete MBG.

6.2 Adequate Subgraphs w.r.t. a Given Matching

We describe a new algorithm to compute adequate subgraphs in a complete MBG, based on the use of a "seed" genome—a perfect matching on V. In practice, this seed genome can be one of the three given matchings; it can also be computed with various heuristics.

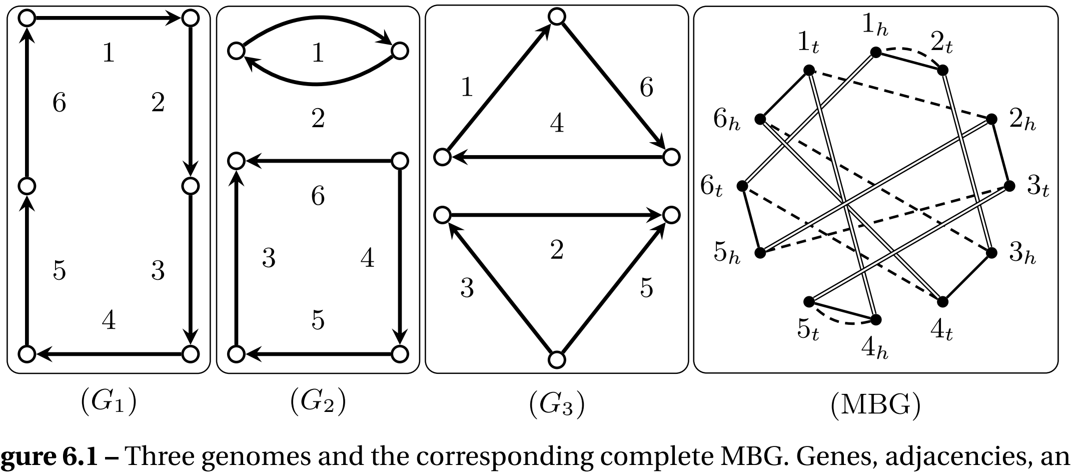


Figure 6.1 – Three genomes and the corresponding complete MBG. Genes, adjacencies, and extremities are represented by arrows, circles, and solid circles, respectively. Adjacencies in G_1, G_2, and G_3 are represented by solid, dashed and double lines respectively.

Let M be a perfect matching on V. We say a MBG $B' = (V', M'_1, M'_2, M'_3)$ is adequate w.r.t. M if there exists a matching M' on V' satisfying $\sum_{k=1}^{3} c(M', M'_k) \geq 3 \cdot |V'|/4$ and $M' \subseteq M$. If B' is adequate w.r.t. M, then clearly it is adequate. Given a complete MBG $B = (V, M_1, M_2, M_3)$ and a perfect matching M on V, let $\mathscr{C}_k$ be the set of cycles in the breakpoint graph (V, M_k, M), $k = 1, 2, 3$, and write $\mathscr{C} = \cup_{k=1}^{3} \mathscr{C}_k$. For a cycle $C \in \mathscr{C}$, let $V(C)$ be the set of vertices covered by C and $E(C)$ be the set of edges covered by C. For a subset $\mathscr{S} \subseteq \mathscr{C}$, set $V(\mathscr{S}) = \cup_{C \in \mathscr{S}} V(C)$ and $E(\mathscr{S}) = \cup_{C \in \mathscr{S}} E(C)$.

Lemma 6.2.1. *There exist adequate subgraphs of B w.r.t. M if and only if there exists a subset $\mathscr{S} \subseteq \mathscr{C}$ obeying $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$.*

Proof. If such $\mathscr{S}$ exists, we can define the subgraph as $(V(\mathscr{S}), M_1 \cap E(\mathscr{S}), M_2 \cap E(\mathscr{S}), M_3 \cap E(\mathscr{S}))$. Let $M' = M \cap E(\mathscr{S})$; then the sum $\sum_{k=1}^{3} c(M', M_k \cap E(\mathscr{S}))$ is exactly equal to $|\mathscr{S}|$, which is larger than or equal to $3 \cdot |V(\mathscr{S})|/4$. Thus, our subgraph is adequate w.r.t. M. Conversely, suppose that there exists one adequate subgraph (V', M'_1, M'_2, M'_3) of B w.r.t. M and let $M' \subseteq M$ be a matching on V' satisfying $\sum_{k=1}^{3} c(M', M'_k) \geq 3 \cdot |V'|/4$. Let $\mathscr{S}$ be the set of all cycles in the three breakpoint graphs (V', M', M'_k), $k = 1, 2, 3$. We can write $|\mathscr{S}| = \sum_{k=1}^{3} c(M', M'_k)$. Since M'_1, M'_2 and M'_3 are all matchings on V', we have that $|V'| \geq |V(\mathscr{S})|$. The combination of these formulas yields $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$. □

We now use a network flow formulation to compute such $\mathscr{S}$ as illustrated in Fig. 6.2.

Let N be the network. We add to N one vertex for each extremity in V, one vertex for each cycle in $\mathscr{C}$, one source vertex s and one sink vertex t. We add to N one directed edge of capacity $3/4$ from s to each extremity in V and one directed edge of unit capacity from each cycle in $\mathscr{C}$ to t. For each pair of $v \in V$ and $C \in \mathscr{C}$ with $v \in V(C)$, we add one directed edge of very large (large enough not to act as a constraint) capacity from v to C. Let f be a maximum s-t flow of N, N_f the residual network w.r.t. f, S the set of vertices reachable from s in N_f, and T the set of all other vertices.

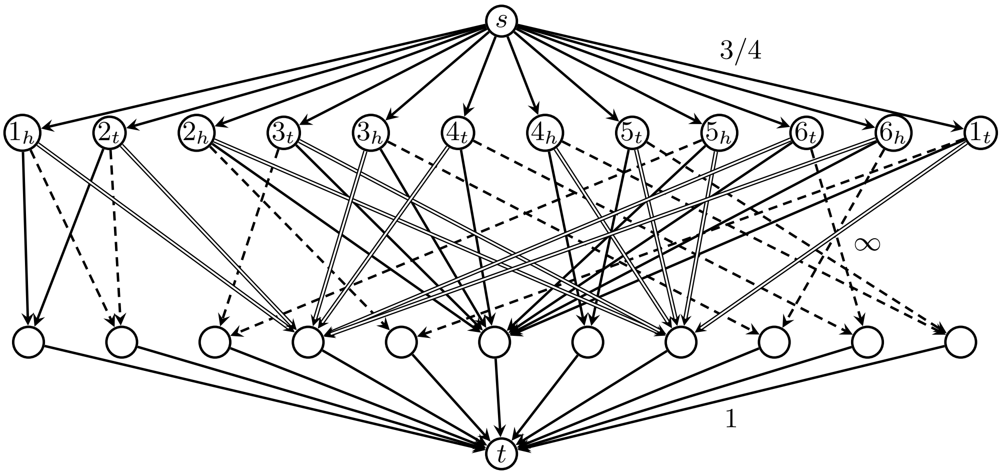


Figure 6.2 – The network for the complete MBG of Fig. 6.1 with the seed $M = M_2$.

Lemma 6.2.2. *There exists a subset $\mathscr{S} \subseteq \mathscr{C}$ with $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$ if and only if we have $\{t\} \subsetneq T$.*

Proof. Based on the construction of S and T, we must have $s \in S$ and $t \in T$; moreover, (S, T) is a minimum s-t cut of N. For any other minimum s-t cut (S', T'), we have $|S| \leq |S'|$. The total capacity of cut (S, T) is at most $|\mathscr{C}|$, since it is a minimum s-t cut and there is a trivial s-t cut (containing just the sink t on one side) whose total capacity is $|\mathscr{C}|$.

Assume we have $\{t\} \subsetneq T$. Let $\mathscr{S} \subseteq \mathscr{C}$ be the set of cycles in T and let $V' \subseteq V$ be the set of extremities that are in T. The edges assigned a very large capacity cannot belong to the (S, T) cut, so that the total capacity of the (S, T) cut is exactly $3 \cdot |V'|/4 + |\mathscr{C}| - |\mathscr{S}|$. Since the total capacity of any minimum s-t cut is at most $|\mathscr{C}|$, we must have $|\mathscr{S}| \geq 3 \cdot |V'|/4$. Because the edges assigned a very large capacity are not in the (S, T) cut, we also have $V(\mathscr{S}) \subseteq V'$. Thus, we can conclude $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$.

Conversely suppose there exists a subset $\mathscr{S}$ satisfying $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$. We prove $\{t\} \subsetneq T$ by contradiction. Assume $T = \{t\}$; then the total capacity of the cut (S, T) is $|\mathscr{C}|$. Now we construct another s-t cut (S', T'), where T' consists of the extremities in $V(\mathscr{S})$ and the cycles in $\mathscr{S}$ and sink t. The total capacity of this cut (S', T') is $3 \cdot |V(\mathscr{S})|/4 + |\mathscr{C}| - |\mathscr{S}|$, which is less than or equal to $|\mathscr{C}|$ since we have $|\mathscr{S}| \geq 3 \cdot |V(\mathscr{S})|/4$. Thus (S', T') is also a minimum s-t cut, but clearly we have $|S'| < |S|$, the desired contradiction. □

These two lemmas suggest a polynomial-time algorithm to decide the existence of the adequate subgraphs w.r.t. a seed perfect matching: based on the proof of Lemma 6.2.2, if such adequate subgraphs exist, we can find one from the residual network.

6.3 The Upper Bound is Tight

Let M_0 be a median of a complete MBG $B = (V, M_1, M_2, M_3)$. We denote by $d_m = \sum_{k=1}^{3} d(M_0, M_k)$ the median distance of B and by $d_t = d(M_1, M_2) + d(M_1, M_3) + d(M_2, M_3)$ the *triangle distance* of B. According to the triangle inequality (the DCJ distance is a metric), we have $d(M_0, M_i) + d(M_0, M_j) \geq d(M_i, M_j)$, $1 \leq i < j \leq 3$, which yields a lower bound for the median distance, $d_m \geq d_t/2$. On the other hand, by using any of M_1, M_2, and M_3 as a possible median, we get $d_m \leq d(M_1, M_2) + d(M_1, M_3)$, $d_m \leq d(M_2, M_1) + d(M_2, M_3)$, and $d_m \leq d(M_3, M_1) + d(M_3, M_2)$, which yields an upper bound for the median distance, $d_m \leq 2 \cdot d_t/3$. Fig. 6.3 shows a subgraph where the upper bound is reached. Notice that this subgraph is also adequate. Thus, the combination of any number of copies of this subgraph yields a graph that also reaches the upper bound.

6.4 Deciding Equality to the Bounds

We now study a pure decision problem: whether the median distance of a complete MBG reaches its lower or upper bound.

Let $u, v \in V$ be two distinct vertices. A DCJ operation *induced* by (u, v) on M_1 removes (u, u_1) and (v, v_1) from M_1 and adds (u, v) and (u_1, v_1) back to M_1, where u_1 and v_1 are the neighbors of u and v in M_1 respectively. If u is matched to v in M_1, then the DCJ operation induced by (u, v) on M_1 does not change anything; otherwise, we have the following property.

Property 6.4.1. *Let M and M_1 be two perfect matchings on V and $u, v \in V$ two distinct vertices with $(u, v) \in M$ and $(u, v) \not\in M_1$. Then we can write $d(M, M_1') = d(M, M_1) - 1$, where M_1' is the perfect matching obtained from M_1 after performing the DCJ operation induced by (u, v).*

We say (u, v) is *strong* w.r.t. M_1 and M_2 if u and v are in the same cycle of (V, M_1, M_2) and the distance between them (the number of edges on the shorter path from u to v) is odd—see Fig. 6.4. Otherwise, we say (u, v) is *weak* w.r.t. M_1 and M_2.

Property 6.4.2. *Let M_1' and M_2' be the two perfect matchings after performing the two DCJ operations induced by (u, v) on M_1 and M_2 respectively. Then (u, v) is strong w.r.t. M_1 and M_2 if and only if we have*

$$d(M_1', M_2') = \begin{cases} d(M_1, M_2) & \textit{if} \quad (u, v) \in M_1 \cap M_2; \\ d(M_1, M_2) - 1 & \textit{if} \quad (u, v) \in (M_1 - M_2) \cup (M_2 - M_1); \\ d(M_1, M_2) - 2 & \textit{if} \quad (u, v) \not\in M_1 \cup M_2. \end{cases}$$

We say two strong edges (u, v) and (u', v') w.r.t. M_1 and M_2 are *independent* w.r.t. M_1 and M_2 if (i) they are in different cycles of (V, M_1, M_2) or (ii) they do not "intersect" in the same cycle—where an intersection would mean that u' and v' are on the different paths from u to v.

Property 6.4.3. *Let (u, v) be a strong edge w.r.t. M_1 and M_2, and M_1' and M_2' be the matchings after performing two DCJ operations induced by (u, v) on M_1 and M_2 respectively.*

(a) If edge (u', v') is weak w.r.t. M_1 and M_2, then (u', v') is weak w.r.t. M_1' and M_2'.

(b) If edge (u', v') is strong w.r.t. M_1 and M_2 and (u, v) and (u', v') are independent w.r.t. M_1 and M_2, then (u', v') is strong w.r.t. M_1' and M_2'.

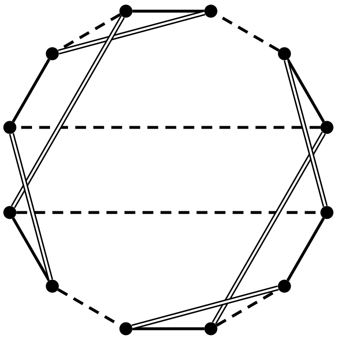


Figure 6.3 – Tightness of the upper bound. M_1, M_2, M_3 are represented by solid, dashed and double edges. We have $d(M_1, M_2) = d(M_1, M_3) = d(M_2, M_3) = 4$ and thus $d_t = 12$. Any M_k is a median with $d_m = \sum_{k=1}^{3} d(M_1, M_k) = 8$. Thus we have $3 \cdot d_m = 2 \cdot d_t$.

(c) *If edge (u', v') is strong w.r.t. M_1 and M_2 and (u, v) and (u', v') are not independent w.r.t. M_1 and M_2, then (u', v') is weak w.r.t. M'_1 and M'_2.*

(d) *If edges (u', v') and (u'', v'') are strong w.r.t. M_1 and M_2, (u, v) and (u', v') are independent w.r.t. M_1 and M_2, (u, v) and (u'', v'') are independent w.r.t. M_1 and M_2, but (u', v') and (u'', v'') are not independent w.r.t. M_1 and M_2, then (u', v') and (u'', v'') are (strong but) not independent w.r.t. M'_1 and M'_2.*

Lemma 6.4.1. *Let M, M_1 and M_2 be three perfect matchings on V. We have $d(M, M_1) + d(M, M_2) = d(M_1, M_2)$ if and only if M consists of n mutually independent strong edges w.r.t. M_1 and M_2.*

Proof. Choose one edge from M that is not in $M_1 \cup M_2$ and perform the DCJ operations induced by this edge on M_1 and M_2, and repeat until no more such operations can be performed. Let $2 \cdot o$ be the number of DCJ operations performed in this process and let M_1^* and M_2^* be the final matchings thus obtained. We must have $M = M_1^*$ or $M = M_2^*$ since at the final state we cannot find any edge in M that is not in $M_1 \cup M_2$. Without loss of generality, assume $M = M_1^*$. Using property 6.4.1, we have $d(M, M_1) = d(M, M_1^*) + o = d(M_1^*, M_1^*) + o = o$ and $d(M, M_2) = d(M, M_2^*) + o = d(M_1^*, M_2^*) + o$.

Suppose that M consists of n mutually independent strong edges w.r.t. M_1 and M_2. According to property 6.4.3(b), all edges used to perform DCJ operations must be strong w.r.t. their current states. Using property 6.4.2, we get $d(M_1^*, M_2^*) = d(M_1, M_2) - 2 \cdot o$ and thus also

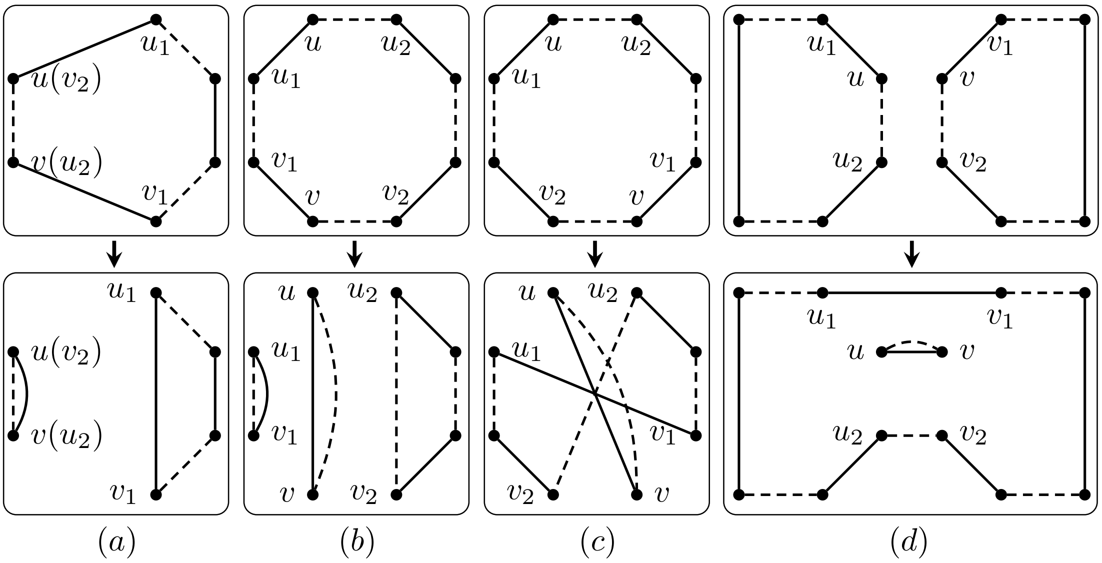


Figure 6.4 – The four cases for two DCJ operations induced by edge (u, v) on M_1 and M_2 (represented by solid and dashed edges respectively). u_1 and v_1 (u_2 and v_2) are the neighbors of u and v in M_1 (M_2). **(a)** u and v are neighbors in M_2; **(b)** u and v are in the same cycle at odd distance; **(c)** u and v are in the same cycle at even distance; and **(d)** u and v are in different cycles. In (a) ad (b), (u, v) is strong w.r.t. M_1 and M_2.

$d(M,M_1)+d(M,M_2)=d(M_1,M_2)$. Now suppose that there exists an edge in M that is weak w.r.t. M_1 and M_2 or that there exist two edges in M that are not independent. By the end of the iterative process, all edges in M are mutually strong w.r.t. M_1^* and M_2^*. Hence according to property 6.4.3, there exists a weak edge that is used to perform DCJ operations in the process. Thus, using property 6.4.2, we have $d(M_1^*,M_2^*) > d(M_1,M_2)-2\cdot o$, which implies $d(M,M_1)+d(M,M_2) > d(M_1,M_2)$. □

We say (u,v) is strong w.r.t. to $B=(V,M_1,M_2,M_3)$ if (u,v) is strong w.r.t. M_1 and M_2, M_1 and M_3, and M_2 and M_3. We say two strong edges (u_1,v_1) and (u_2,v_2) w.r.t. B are independent w.r.t. B if they are independent w.r.t. M_1 and M_2, M_1 and M_3, and M_2 and M_3. Now we are in a position to state a necessary and sufficient condition for the median distance to reach its lower bound.

Lemma 6.4.2. *We have $d_m = d_t/2$ if and only if there are n mutually independent strong edges w.r.t. B.*

Proof. We have $d_m = d_t/2$ if and only if there exists a perfect matching M_0 of V satisfying $d(M,M_i)+d(M,M_j)=d(M_i,M_j)$ for all $1\le i<j\le 3$. Following Lemma 6.4.1, such matching consists exactly of n mutually independent strong edges w.r.t. B. □

We can also give a necessary condition regarding the upper bound.

Lemma 6.4.3. *Assume $M_1\cap M_2\cap M_3=\emptyset$; then we have $d_m = 2\cdot d_t/3$ only if there is no strong edge w.r.t. B.*

Proof. Suppose there exists a strong edge (u,v) w.r.t. B. Let M_0 be one median of B. We analyze the situation case by case.

First, assume we have $(u,v)\not\in M_1\cup M_2\cup M_3$. We perform the DCJ operations induced by (u,v) on M_1, M_2 and M_3. Let M_k', $k=1,2,3$, be the corresponding new matchings and denote by $B'=(V,M_1',M_2',M_3')$ be the new complete MBG. Now we have $(u,v)\in\cap_{k=1}^3 M_k'$, and clearly the subgraph induced by $\{u,v\}$ is adequate. Thus, there exists one median of B', call it M_0', with $(u,v)\in M_0'$. Set $d_m'=\sum_{k=1}^3 d(M_0',M_k')$ and $d_t'=d(M_1',M_2')+d(M_1',M_3')+d(M_2',M_3')$. We know that each DCJ operation can increase the DCJ distance by at most one. Thus, we have

$$d_m \le \sum_{k=1}^{3} d(M_0',M_k) \le \sum_{k=1}^{3} (d(M_0',M_k')+1) = d_m'+3.$$

On the other hand, since (u,v) is strong w.r.t. B, by property 6.4.2, we have $d(M_1',M_2') = d(M_1,M_2)-2$, $d(M_1',M_3')=d(M_1,M_3)-2$, and $d(M_2',M_3')=d(M_2,M_3)-2$, which gives us $d_t'=d_t-6$. Applying the upper bound on B', we get $d_m'\le 2\cdot d_t'/3$. By combining these formulas, we finally get $d_m\le d_m'+3\le 2\cdot(d_t-6)/3+3=2\cdot d_t/3-1$. Thus, the upper bound cannot be achieved.

Second, assume we have $(u, v) \in M_1 - (M_2 \cup M_3)$. Now we perform the DCJ operations induced by (u, v) on just M_2 and M_3. We can thus write $d_m \leq d'_m + 2$. By applying property 6.4.2 and using the fact that M'_1 is just M_1, we can write $d(M'_1, M'_2) = d(M_1, M_2) - 1$, $d(M'_1, M'_3) = d(M_1, M_3) - 1$, and $d(M'_2, M'_3) = d(M_2, M_3) - 2$, which gives us $d'_t = d_t - 4$. Putting everything together, we get $d_m \leq d'_m + 2 \leq 2 \cdot (d_t - 4)/3 + 2 = 2 \cdot d_t/3 - 2/3$, which implies that the upper bound cannot be achieved.

Third, assume we have $(u, v) \in (M_1 \cap M_2) - M_3$. Now we perform the DCJ operation induced by (u, v) on M_3 only. By similar reasoning, we get $d_m \leq d'_m + 1$, $d(M'_1, M'_2) = d(M_1, M_2)$, $d(M'_1, M'_3) = d(M_1, M_3) - 1$, and $d(M'_2, M'_3) = d(M_2, M_3) - 1$. Thus, we have $d_m \leq d'_m + 1 \leq 2 \cdot (d_t - 2)/3 + 1 = 2 \cdot d_t/3 - 1/3$, which again implies that the upper bound cannot be achieved. □

While necessary, the condition of Lemma 6.4.3 is not sufficient, as illustrated in Fig. 6.5: the subgraph in the figure has no strong edge, but the median distance is not equal to its upper bound. Notice that the subgraph in Fig. 6.5 is adequate, so that we can build a general example by combining an arbitrary number of copies of this subgraph.

By Lemma 6.4.2, in order to decide whether the median distance reaches its lower bound, we need only check whether there exist n mutually independent strong edges w.r.t. B. This problem can be reduced to a maximum independent set problem, in which each strong edge is a vertex and linking two strong edges if they are not independent. Clearly, there exist n mutually independent strong edges if and only if the size of the maximum independent set is n. While the independent set problem is NP-hard, we can test in polynomial-time whether there exist n mutually independent strong edges w.r.t. M_1 and M_2, a necessary condition. The algorithm enumerates all possible strong edges w.r.t. M_1 and M_2; this can be done in $O(n^3)$ time. Let $C_1, C_2, \ldots, C_m$ be the cycles in the breakpoint graph (V, M_1, M_2). Because each strong edge must have both endpoints on the same cycle, we can handle each cycle separately. For cycle C_i with $V(C_i)$ vertices, we use dynamic programming to compute the maximum

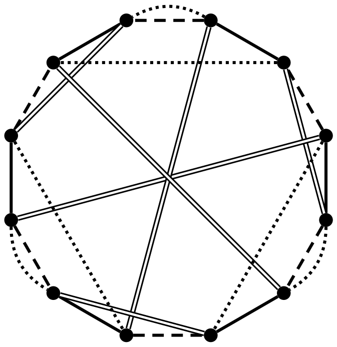


Figure 6.5 – A subgraph with no strong edge where the median distance does not reach its upper bound. Matchings M_1, M_2, M_3, and M_0 are represented by solid, dashed, double and dotted edges respectively. We have $d(M_1, M_2) = d(M_1, M_3) = d(M_2, M_3) = 5$, yet $d(M_0, M_1) = 2$, $d(M_0, M_2) = d(M_0, M_3) = 3$.

number of non-crossing edges, taking time in $O(|V(C_i)|^3)$. If this maximum number is less than $V(C_i)/2$, then we cannot find enough independent strong edges and thus the algorithm returns false. If the algorithm terminates after examining all cycles, it returns true. The total running time is $O(n^3)$.

The necessary condition for the upper bound stated in Lemma 6.4.3 can be tested effectively by just checking each pair of vertices to see whether it is strong w.r.t. B. The running time is also $O(n^3)$.

6.5 Discussion

In this chapter, we have given a new approach to the discovery of adequate subgraphs using a seed genome, thereby significantly extending the applicability of this powerful decomposition strategy. We have also given sharper characterizations of the upper and lower bounds for the median distance, along with polynomial-time algorithms to test necessary conditions for this distance to reach its lower or upper bound. Our work focused on genomes with equal gene content and without duplication and assumed circular chromosomes, the simplest case. Extension to linear chromosomes is the next step, while extension to unequal gene contents with duplications and losses of gene segments appears far more challenging.

7 Trajectory Graphs

As we have seen in the previous chapters, most of the algorithms for various edit distance problems use the same underlying data structure, the *adjacency graph*. Adjacency graphs have also been extended to study rearrangements with insertions and deletions [25, 69, 13], whole-genome duplications [70, 71] as well as incorporating sequence information [72]. In this chapter, we propose a new graphical data structure, the *trajectory graph*, which models not only the final states of two genomes but also an existing evolutionary trajectory between them. We begin by defining and illustrating the trajectory graph, then show correspondences between redundant rearrangements in the initial trajectory and certain cycles in the corresponding trajectory graph, and provide an effective algorithm to remove these redundant rearrangements by iteratively resolving the active cycles in the trajectory graph. We also prove that this algorithm converges to the optimal trajectory from any initial trajectory when the model is restricted to rearrangements.

7.1 The Trajectory Graph

In this chapter, we study an evolutionary model including DCJ operations and segmental duplications. We now view each of them as a function of adjacencies. For DCJ operation, the input is two adjacencies (a,b) and (c,d) and the output is two new adjacencies (a,d) and (c,b), or $(a,-c)$ and $(-b,d)$. A segmental duplication operation duplicates a segment and either creates a new circular chromosome out of the copy or inserts the copy in the genome at some location outside the original segment. A segment consisting of n genes $g_1,g_2,\cdots,g_n$ can be represented by its $(n-1)$ adjacencies, $(g_1,g_2),(g_2,g_3),\cdots,(g_{n-1},g_n)$. To keep notation shorter, we will simply write $(g_1,g_2,\cdots,g_n)$ to represent these $(n-1)$ adjacencies and, when appropriate, we simply use g' to represent the copy of the original gene g after a duplication. We now define the two types of segmental duplications in term of adjacencies. The first type of the segmental duplication [28, 29] inserts a copy of a segment $(g_1,g_2,\cdots,g_n)$ to a position specified by the adjacency (a,b); thus it takes $(g_1, g_2,\cdots,g_n)$ and (a,b) as input, and outputs (a,g_1'), $(g_1', g_2',\cdots,g_n')$, (g_n',b) and $(g_1,g_2,\cdots,g_n)$, or $(a,-g_n')$, $(-g_n',-g_{n-1},\cdots,g_{n-1}')$, $(-g_1',b)$ and $(g_1,g_2,\cdots,g_n)$. Note that $(g_1',g_2',\cdots,g_n')$ and $(-g_n', -g_{n-1}',\cdots,-g_1')$ represent the same adjacency set. The second type of the segmental duplication [73] creates a new circular chromosome with the copy of the segment $(g_1, g_2, \cdots, g_n)$, thus it takes only $(g_1, g_2, \cdots, g_n)$ as input and outputs (g'_n, g'_1), $(g'_1, g'_2, \cdots, g'_n)$ and $(g_1, g_2, \cdots, g_n)$.

Each operation can be represented as a directed subgraph, composed of one *operation node* representing the operation itself, a number of *input adjacency nodes* (one for each of its input adjacencies), a number of *output adjacency nodes* (one for each of its output adjacencies), one directed edge from each input adjacency node to the operation node, and one directed edge from the operation node to each output adjacency node. Figure 7.1 illustrates these graph components for DCJ and the two types of duplication.

We say that an edge is *active* if the adjacency associated with the node to which it is attached has been changed by the operation, *inactive* otherwise. In the graph for a DCJ operation, all four edges are active. In the graph for the first type of duplication, only the edge from (a, b) to the operation node and the two edges from the operation node to the output adjacencies (a, g'_1) and (g'_n, b), or $(a, -g'_n)$ and $(-g'_1, b)$ are active, while the other edges are inactive. In the graph for the second type of duplication, only the edge from the operation node to (g'_n, g'_1) is active; all other edges are inactive. We say that a cycle in the trajectory graph is an *active cycle* if all of its edges are active and define the *size* of a cycle as the number of the operation nodes it contains.

Given two adjacency sets X and Y, a *sorting path* $P = \{p_1, p_2, \cdots, p_n\}$ from X to Y is a series of operations that transform X into Y. The *trajectory graph* $G(P)$ with respect to P naturally delineates the input and output of each operation and the dependency relationships between

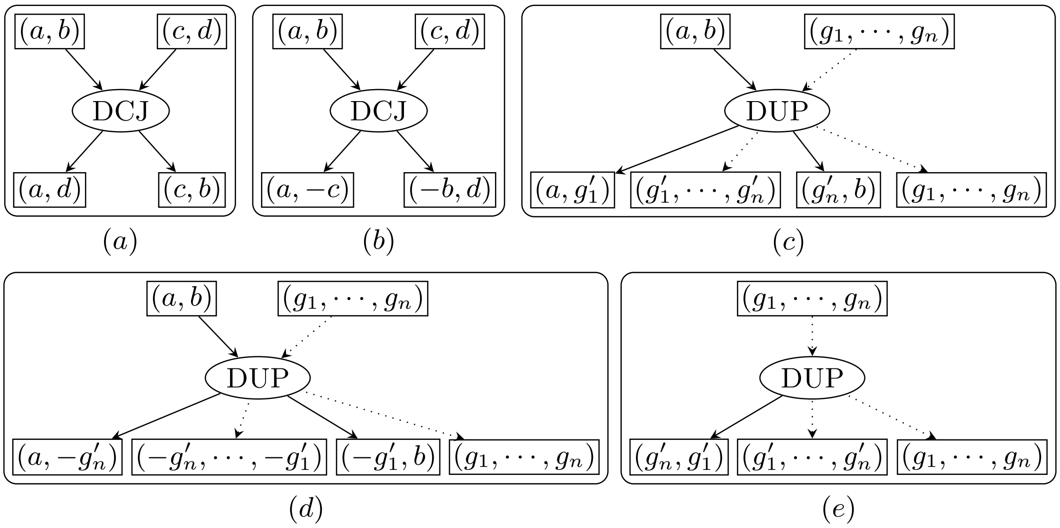


Figure 7.1 – Illustration of the DCJ and the segmental duplication as functions of adjacencies, among which part (**a**) and (**b**) are the DCJ, part (**c**) and (**d**) are the first type of the duplication, and part (**e**) is the second type of the duplication. Active edges are drawn with solid lines while inactive edges are drawn with dotted lines.

them. Let S_k be the adjacency set after sequentially performing operations $p_1, p_2, \cdots, p_k$ starting from X. The *trace* from X to Y with respect to P is $(X = S_0, S_1, S_2, \cdots, S_n = Y)$. To construct $G(P)$, the initial step is to draw one *adjacency node* for each adjacency in X. Then we sequentially handle each operation in P, ensuring that, before tackling operation p_k, all adjacency nodes of outdegree 0 in the current graph are exactly S_{k-1}. To add operation p_k, we connect the component graph for operation p_k to the current graph by replacing all the input adjacency nodes with their counterparts in the current graph. After all operations are added, the set of all adjacency nodes of outdegree 0 is then exactly Y (see Figure 7.2).

By construction, the trajectory graph has the following two properties. First, the adjacency nodes of indegree 0 form the adjacency set of X and the adjacency nodes of outdegree 0 form the adjacency set of Y, while all other adjacency nodes have indegree 1 and outdegree 1. Second, the trajectory graph is a directed acyclic graph and any topological sorting of all the operation nodes is a valid sorting path from X to Y.

7.2 An Iterative Algorithm to Improve any Trajectory

Given any two genomes and initial evolutionary trajectory P between them, we build the trajectory graph $G(P)$ and give the following sufficient condition in $G(P)$ to identify and resolve redundant rearrangements in P.

Theorem 7.2.1. *Let P be a sorting path from X to Y. If $G(P)$ contains active cycles, then we can find another sorting path P' from X to Y with fewer DCJ operations and an equal number of duplications.*

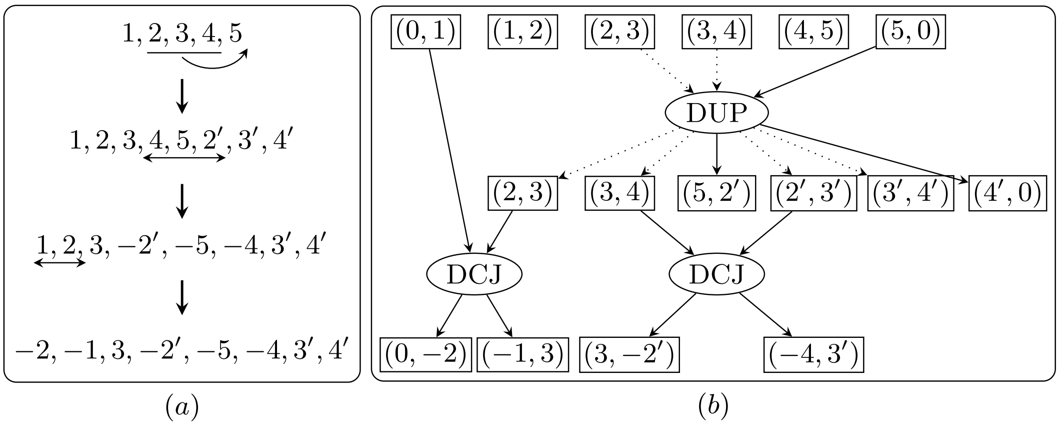


Figure 7.2 – A trajectory graph. The initial genome consists of one linear chromosome of 5 genes, (1,2,3,4,5). The duplication operation inserts a copy of (2,3,4) to the right end, which transforms the genome into (1,2,3,4,5,2′,3′,4′). Then two DCJ operations, one inverting the segment of (4,5,2′) and the other inverting the segment of (1,2), generate the final genome as (−2,−1,3,−2′,−5,−4,3′,4′). Adjacency nodes (1,2) and (4,5) form two trivial connected components, while the rest of the graph forms a nontrivial connected component.

Proof. Let C be an active cycle in $G(P)$. Since $G(P)$ is directed, we can represent C as two node-disjoint directed paths starting from the same *top node* and ending at the same *bottom node*. Clearly, neither of these two nodes can be an adjacency node: the top node must have outdegree 2 and the bottom node must have indegree 2. Moreover, the bottom node must be a DCJ node, since there is at most one active edge pointing to each duplication node. We now show how to exchange the bottom operation node with one of the two parent operation nodes, thereby moving one operation node out of C, until there are only two operation nodes left in C, which we can always replace with at most one operation.

We choose one of the two parent operation nodes to guarantee that the exchange will not create any directed cycles. If both parent nodes are independent (there is no directed path from one to the other), then we arbitrarily choose one; otherwise, we always choose the one that is on the directed path from the other to the bottom node.

Figure 7.3 shows how to exchange the bottom DCJ node with a parent DCJ node. We view the two DCJ nodes as a single supernode, with three input adjacencies and three output adjacencies. We replace the current two DCJ nodes with two new ones, keeping the inputs and outputs of the supernode unchanged. The new top DCJ node takes the two input adjacencies of the supernode that are linked in C as its inputs and outputs two adjacencies, one of which is among the outputs of the supernode. The new bottom DCJ node takes the other output adjacency of the new top DCJ node and the remaining input adjacency of the supernode as inputs and outputs the other two output adjacencies of the supernode. After the exchange, the new bottom DCJ node is out of the new active cycle, while the new top DCJ node becomes the bottom node of the new active cycle.

Figure 7.4 shows how to exchange the bottom DCJ node with its parent duplication node. Again, we consider these two operation nodes as a single supernode. The new DCJ node takes the two adjacency nodes linked in C as inputs and outputs two adjacencies, one of which is

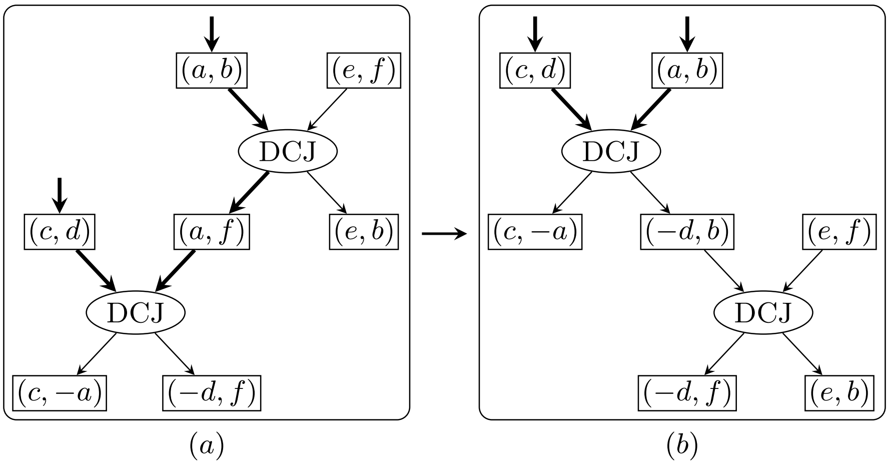


Figure 7.3 – Exchanging two DCJ nodes to reduce the size of the active cycle. Edges in the active cycle are in bold.

among the outputs of the supernode while the other is the insert position of the new bottom duplication node. After the exchange, the new bottom duplication node will be out of the new active cycle and the new parent DCJ node will be the bottom node of the new active cycle.

Through these exchanges, *C* will be reduced to just two operation nodes, the top one and the bottom one. Now we show that we can always replace these two operations with at most one operation.

Consider first the case in which the two operation nodes are both DCJ operations. If the input and output of the supernode are different, then we can use one new DCJ node to connect them, as shown in Figure 7.5(**a**,**b**); otherwise, we do not need any operation node, as shown in Figure 7.5(**c**,**d**).

Next consider the case in which the top node is a duplication and the bottom node is a DCJ. Assume the top duplication inserts a copy of the segment to position (a,b). If (a,b) is not one

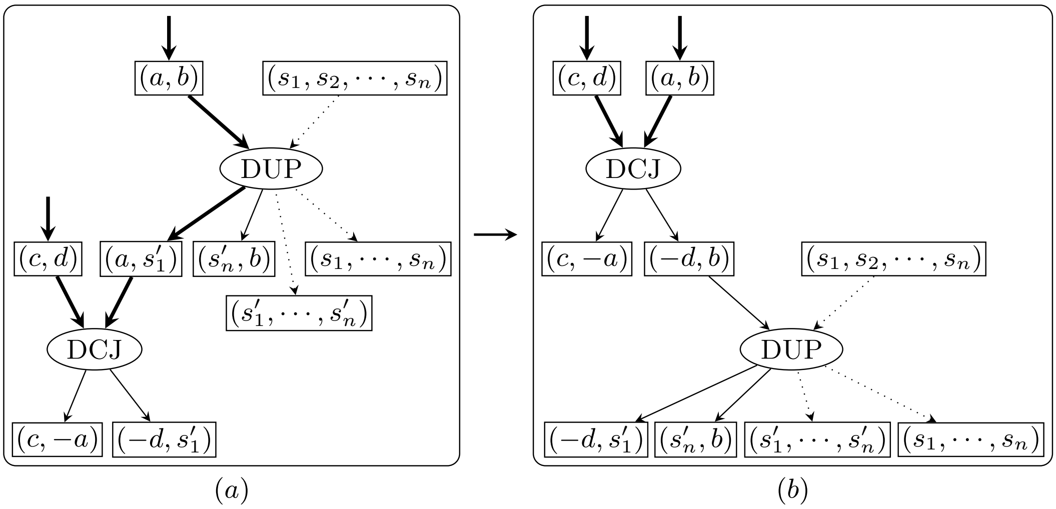


Figure 7.4 – Exchanging the bottom DCJ node with its parent duplication node.

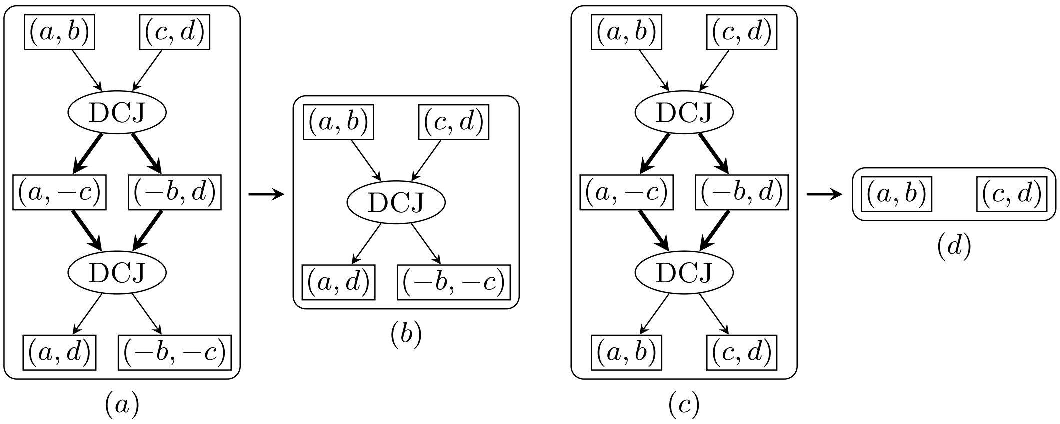


Figure 7.5 – Resolving the active cycle consisting of two DCJ operations.

of the output adjacencies of the bottom DCJ node, then these two operations can be replaced by one duplication which inserts the inverted segment to the same position, as shown in Figure 7.6(**a**,**b**); otherwise, these two operations can be replaced by one duplication which creates a circular chromosome from the copy of the segment, as shown in Figure 7.6(**c**,**d**).

Note that the trajectory graph remains a directed acyclic graph. Thus we can retrieve one sorting path from any topological sorting of the operation nodes in the final trajectory graph; since we also showed that the number of DCJ operation nodes has reduced by at least one and the number of duplication nodes is unchanged, the theorem is proved. □

Given the trajectory graph $G(P)$ constructed from two input genomes and an initial trajectory P, our algorithm iteratively applies Theorem 7.2.1 to reduce the number of operations until there is no active cycle in $G(P)$. This algorithm will always terminate since resolving each cycle will reduce the number of operation nodes in $G(P)$ and such number is always non-negative.

7.3 The Trajectory Graph Restricted to Rearrangements

We study the trajectory graphs with only DCJ operations. We show that the above iterative algorithm always converges to the optimal trajectory for any initial trajectory. We first investigate the structure of the trajectory graphs under a rearrangement-only model and illustrate the close relationship with adjacency graphs. Recall that given two adjacency sets X and Y,

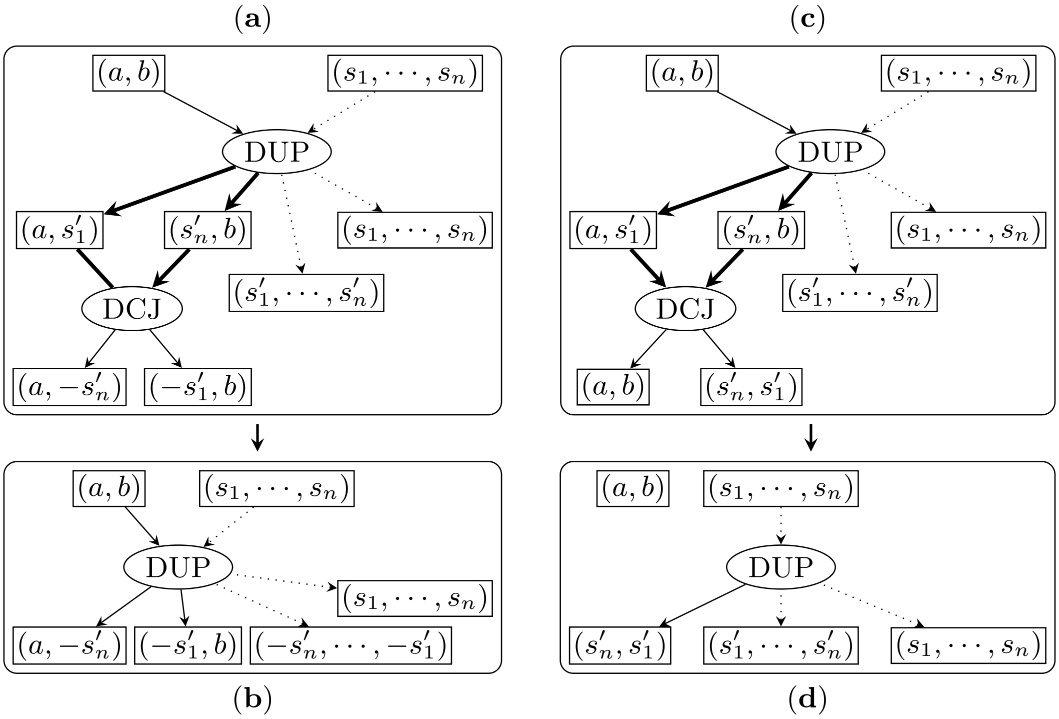


Figure 7.6 – Resolving the active cycle consisting of one DCJ operation and one duplication.

the adjacency graph is defined as a bipartite multigraph $A = \{X, Y, E\}$, in which $u \in X$ and $v \in Y$ are linked by one edge if u and v share one extremity and by two edges if they share two extremities. If X and Y have the same extremity set and each extremity appears only once, the adjacency graph consists of node-disjoint cycles and the minimum number of DCJ operations needed to transform X into Y is $|X| - c$, where c is the number of cycles in A [11].

Let a trajectory P consist of only DCJ operations. For a connected component C of the trajectory graph $G(P)$, we use $I(C)$ to denote the set of adjacency nodes of indegree 0 in C, $O(C)$ to denote the set of adjacency nodes of outdegree 0 in C, $A(C)$ to denote the set of adjacency nodes with indegree 1 and outdegree 1, and $D(C)$ to denote the set of DCJ nodes in C. We say that a connected component is *trivial* if it is a single adjacency node and *nontrivial* otherwise (for examples see Figure 7.2).

Lemma 7.3.1. *Let C be a connected component in the trajectory graph $G(P)$ where P consists of only DCJ operations. Then we have C is a tree if and only if $|D(C)| = |I(C)| - 1$.*

Proof. If C is trivial, it is a tree, and we have $|D(C)| = 0$ and $|I(C)| = 1$, hence the lemma holds. Assume then that C is nontrivial. The number of edges equals the sum of indegrees, which is $|O(C)| + |A(C)| + 2 \cdot |D(C)|$; the number of nodes is $|I(C)| + |O(C)| + |A(C)| + |D(C)|$, since $I(C)$ and $O(C)$ are disjoint when C is nontrivial. C is connected, so it is a tree exactly when it has one more vertices than it has edges, hence we can write $|O(C)| + |A(C)| + 2 \cdot |D(C)| = |I(C)| + |O(C)| + |A(C)| + |D(C)| - 1$, which yields $|D(C)| = |I(C)| - 1$, as desired. □

The following lemma shows that there is one-to-one correspondence between trees in the trajectory graph and cycles in the adjacency graph.

Lemma 7.3.2. *Let C be a tree in the trajectory graph $G(P)$ where P consists of only DCJ operations. The corresponding adjacency graph $A = \{I(C), O(C), E\}$ consists of exactly one cycle.*

Proof. If C is trivial, then A is a cycle of length 2, hence the lemma holds. For a nontrivial C, we proceed by contradiction. Suppose that A consists of two or more cycles. We partition A into two parts by taking one arbitrarily chosen cycle as the first part and the remaining cycle(s) as the second part. We then label all extremities in the first part as e_1 and all extremities in the second part as e_2. We divide the adjacency nodes in C into three categories: if its two extremities are both labeled e_1, then label the node a_1; if its two extremities are both labeled e_2, then label the node a_2; otherwise, label the node a_3. Now we can classify any DCJ operation into one of the following 7 types:

1. $\{a_1, a_1\} \to \{a_1, a_1\}$, or
2. $\{a_2, a_2\} \to \{a_2, a_2\}$, or
3. $\{a_1, a_2\} \to \{a_3, a_3\}$, or
4. $\{a_1, a_3\} \to \{a_1, a_3\}$, or
5. $\{a_2, a_3\} \to \{a_2, a_3\}$, or

6. $\{a_3, a_3\} \to \{a_3, a_3\}$, or
7. $\{a_3, a_3\} \to \{a_1, a_2\}$.

Note that the labels on the adjacency nodes in $I(C) \cup O(C)$ must be either a_1 or a_2, since these adjacencies are the nodes of the adjacency graph, whose two extremities have the same type. Note also that there must exist at least one adjacency node in C that is labeled as a_3: otherwise the DCJ operation can be only of type (1) or (2) and C can be divided into two disconnected subgraphs, one defined by adjacency nodes labeled a_1 and DCJ nodes of type (1), which contradicts the fact that C is connected. Now, remove all adjacency nodes labeled a_1 or a_2 and their adjacent edges: the remaining nodes must have even total degree—see Figure 7.7. The reason is that all adjacency nodes in $I(C) \cup O(C)$ are removed, while adjacency nodes in $A(C)$ either are removed or are labeled as a_3 and have degree 2. Moreover, all operation nodes must have even total degree, as easily verified by checking the 7 types. Hence there must be one cycle in C, a contradiction since C is a tree. □

Theorem 7.3.1. *A trajectory P consisting of only DCJ operations is optimal if and only if the corresponding trajectory graph $G(P)$ consists of trees.*

Proof. If $G(P)$ contains at least one cycle C , then this cycle C must be an active cycle since all edges in $G(P)$ are active for the DCJ-only model. Thus, according to Theorem 7.2.1, we have that P is not optimal.

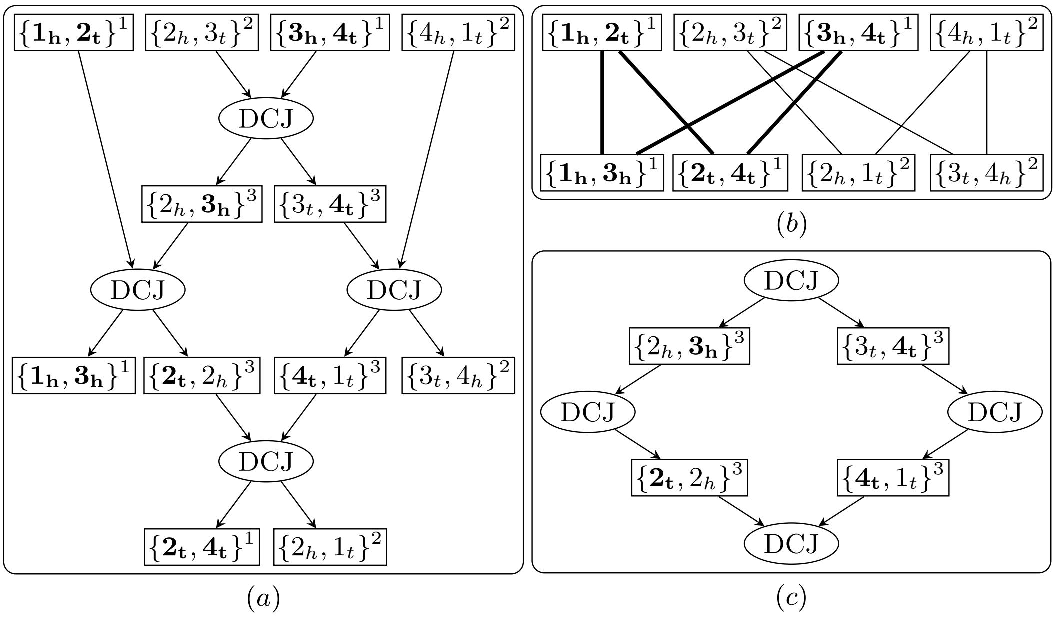

$\{\mathbf{1_h}, \mathbf{2_t}\}^1$
$\{2_h, 3_t\}^2$
$\{\mathbf{3_h}, \mathbf{4_t}\}^1$
$\{4_h, 1_t\}^2$
DCJ
$\{2_h, \mathbf{3_h}\}^3$
$\{3_t, \mathbf{4_t}\}^3$
DCJ
DCJ
$\{\mathbf{1_h}, \mathbf{3_h}\}^1$
$\{\mathbf{2_t}, 2_h\}^3$
$\{\mathbf{4_t}, 1_t\}^3$
$\{3_t, 4_h\}^2$
DCJ
$\{\mathbf{2_t}, \mathbf{4_t}\}^1$
$\{2_h, 1_t\}^2$
(a)
$\{\mathbf{1_h}, \mathbf{2_t}\}^1$
$\{2_h, 3_t\}^2$
$\{\mathbf{3_h}, \mathbf{4_t}\}^1$
$\{4_h, 1_t\}^2$
$\{\mathbf{1_h}, \mathbf{3_h}\}^1$
$\{\mathbf{2_t}, \mathbf{4_t}\}^1$
$\{2_h, 1_t\}^2$
$\{3_t, 4_h\}^2$
(b)
DCJ
$\{2_h, \mathbf{3_h}\}^3$
$\{3_t, \mathbf{4_t}\}^3$
DCJ
DCJ
$\{\mathbf{2_t}, 2_h\}^3$
$\{\mathbf{4_t}, 1_t\}^3$
DCJ
(c)

Figure 7.7 – Part (**a**) shows a non-trivial connected component C of a trajectory graph. Part (**b**) is the corresponding adjacency graph $A = \{I(C), O(C), E\}$, which has two cycles. All extremities in the first cycle are shown bold. The superscripts 1, 2 and 3 on each adjacency represent labels of a_1, a_2 and a_3 respectively. After removing all adjacency nodes in (**a**) labeled as a_1 or a_2, the remaining part is shown in part (**c**), in which all nodes have even total degree.

We now prove that, if $G(P)$ consists of only trees, then P is optimal. Now assume that $G(P)$ consists of m trees, $T_1, T_2, \cdots, T_m$. Applying Lemma 7.3.1 to each tree, we find that the total number of DCJ nodes in $G(P)$ is

$$\sum_{k=1}^{m} |D(T_k)| = \sum_{k=1}^{m} (|I(T_k)| - 1) = \sum_{k=1}^{m} |I(T_k)| - m.$$

On the other hand, according to Lemma 7.3.2, there are exactly m cycles in the adjacency graph $A = \{\cup_{k=1}^{m} I(T_k), \cup_{k=1}^{m} O(T_k), E\}$, which implies that the minimum number of DCJ operations needed is $\sum_{k=1}^{m} |I(T_k)| - m$. Thus P is optimal, as desired. □

Corollary 7.3.1. *The iterative algorithm converges to the optimal trajectory from any initial trajectory.*

Proof. The iterative algorithm terminates when there is no active cycles in the trajectory graph. According to Theorem 7.3.1, any trajectory retrieved from the trajectory graph is optimal. □

7.4 Discussion

Theorem 7.2.1 gives us a means to reduce the cost of a given sorting path. Unfortunately, the converse of the theorem does not hold: it is not hard to see how to take advantage of duplication nodes to produce a counterexample. Thus repeated applications of the constructive proof of Theorem 7.2.1 do not ensure optimality. However, the iterative improvement procedure can form the basis for strong heuristics or good approximation algorithms.

The trajectory graph naturally combines rearrangements and segmental duplications (or, in general, content-modifying operations) in a single model. Such a basis is crucial to the development of strong characterizations and good algorithms. We also took a step in that direction by showing that the trajectory graph is a proper extension of the adjacency graph, to which it reduces in the absence of duplication, and by describing an efficient iterative algorithm. Our current work focuses on using this improvement method within a large optimization framework (e.g. incorporating methods to find good initial trajectories) to derive fast and accurate approximations.

Different trajectories may correspond to the same trajectory graph, if they can be retrieved by different ways of topological sorting in the same graph. Thus the trajectory graph is useful in representing equivalent trajectories as well as characterizing the space of all optimal trajectories under both rearrangements and content-modifying operations (like the cases under inversions [74, 75] and DCJ operations [76] on adjacency graphs), and thus forming a basis for future statistical analysis.

Under the DCJ model, if the two cuts are in the same linear chromosome, one of the two nontrivial outcomes is to circularize a segment of DNA as a circular chromosome (also called circular intermediate), which has recently been inferred in the evolution of cow genomes [77].

Recent evidence also showed that segmental duplications may also be mediated by circular intermediates in fish genomes [73]. The proof of Theorem 7.2.1 makes natural use of the connection between rearrangements and segmental duplications through circular intermediates, and thus may be useful to identify possible circular intermediates in the evolutionary trajectory.

8 Conclusion

In this dissertation, we study the edit distance problems and median problems in whole-genome comparison. We proposed several novel exact algorithms and approximation algorithms for these problems and also applied them to analyze and annotate biological datasets. We showed that these methods can scale up-to whole-genomes and also guarantee very high accuracy.

We remark that in the process of designing these algorithms, we also proved many properties and identified many special structures inside these problems. For example, we proved that under certain condition, a shared adjacency can be fixed optimally for both the breakpoint distance and DCJ distance. We also showed that there is an one-to-one correspondence between each optimal sorting path of DCJ operations and each trajectory graph consisting of only trees. These structures and properties can help understanding these problems and thus help designing more efficient algorithms.

We emphasize that the framework of “ILP + identifying optimal substructures + adding extra constraints” proposed in this dissertation can also be used to design practical exact algorithms for other optimization problem, especially NP-hard problems. This framework first requires formulating the problem as an ILP. If this formulation is not efficient on some instances, then we need to study the properties of these instances, which can be incorporated into the algorithm in two ways. First, we can use these properties to directly simplify these instances and pipe the reduced instances to the ILP. Second, these properties can be transformed into additional constraints of the ILP, which usually can reduce the search space and thus speedup the ILP.

Bibliography

[1] W. Gu, F. Zhang, and J.R. Lupski. Mechanisms for human genomic rearrangements. *Pathogenetics*, 1(1):4, 2008.

[2] J.R. Lupski. Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits. *Trends in Genetics*, 14(10):417–422, 1998.

[3] M.R. Lieber, Y. Ma, U. Pannicke, and K. Schwarz. Mechanism and regulation of human non-homologous DNA end-joining. *Nat. Rev. Mol. Cell Biol.*, 4(9):712–720, 2003.

[4] J.A. Lee, C.M.B. Carvalho, and J.R. Lupski. A DNA replication mechanism for generating nonrecurrent rearrangements associated with genomic disorders. *Cell*, 131(7):1235–1247, 2007.

[5] P.J. Hastings, G. Ira, and J.R. Lupski. A microhomology-mediated break-induced replication model for the origin of human copy number variation. *PLoS Genetics*, 5(1):e1000327, 2009.

[6] P. Stankiewicz and J.R. Lupski. Genome architecture, rearrangements and genomic disorders. *Trends in Genetics*, 18(2):74–82, 2002.

[7] D. Simsek, E. Brunet, S.Y.-W. Wong, S. Katyal, Y. Gao, et al. DNA ligase III promotes alternative nonhomologous end-joining during chromosomal translocation formation. *PLoS Genetics*, 7(6):e1002080, 2011.

[8] A.R.J. Lawson, G.F.L. Hindley, T. Forshew, et al. RAF gene fusion breakpoints in pediatric brain tumors are characterized by significant enrichment of sequence microhomology. *Genome Research*, 21(4):505–514, 2011.

[9] C.G. Ghiurcuta and B.M.E. Moret. Evaluating synteny for improved comparative studies. *Bioinformatics*, 30(12):i9–i18, 2014.

[10] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[11] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *Proc. 6th Workshop Algs. in Bioinf. (WABI'06)*, volume 4175 of *Lecture Notes in Comp. Sci.*, pages 163–173. Springer Verlag, Berlin, 2006.

[12] M.A. Alekseyev and P.A. Pevzner. Whole genome duplications, multi-break rearrangements, and genome halving problem. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 665–679, 2007.

[13] S. Yancopoulos and R. Friedberg. Sorting genomes with insertions, deletions and duplications by DCJ. In *Proc. 6th RECOMB Workshop Comp. Genomics (RECOMB-CG'08)*, volume 5267 of *Lecture Notes in Comp. Sci.*, pages 170–183. Springer Verlag, Berlin, 2008.

[14] A. Bergeron, J. Mixtacki, and J. Stoye. A new linear-time algorithm to compute the genomic distance via the double cut and join distance. *Theor. Comput. Sci.*, 410(51):5300–5316, 2009.

[15] X. Chen. On sorting permutations by double-cut-and-joins. In *Proc. 16th Conf. Computing and Combinatorics (COCOON'10)*, volume 6196 of *Lecture Notes in Comp. Sci.*, pages 439–448. Springer Verlag, Berlin, 2010.

[16] X. Chen, R. Sun, and J. Yu. Approximating the double-cut-and-join distance between unsigned genomes. *BMC Bioinformatics*, 12(Suppl 9):S17, 2011.

[17] B.M.E. Moret, Y. Lin, and J. Tang. Rearrangements in phylogenetic inference: Compare, model, or encode? In C. Chauve, N. El-Mabrouk, and E. Tannier, editors, *Models and Algorithms for Genome Evolution*, volume 19 of *Computational Biology*, pages 147–172. Springer Verlag, Berlin, 2013.

[18] P. Feijão and J. Meidanis. SCJ: a breakpoint-like distance that simplifies several rearrangement problems. *ACM/IEEE Trans. on Comput. Bio. & Bioinf.*, 8(5):1318–1329, 2011.

[19] S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th Ann. ACM Symp. Theory of Comput. (STOC'95)*, pages 178–189. ACM Press, New York, 1995.

[20] D.A. Bader, B.M.E. Moret, and M. Yan. A fast linear-time algorithm for inversion distance with an experimental comparison. *J. Comput. Biol.*, 8(5):483–491, 2001.

[21] G. Jean and M. Nikolski. Genome rearrangements: a correct algorithm for optimal capping. *Inf. Proc. Letters*, 104(1):14–20, 2007.

[22] M. Ozery-Flato and R. Shamir. Two notes on genome rearrangement. *J. Bioinf. Comp. Bio.*, 1(1):71–94, 2003.

[23] G. Tesler. Efficient algorithms for multichromosomal genome rearrangements. *J. Comput. Syst. Sci.*, 65(3):587–609, 2002.

[24] N. El-Mabrouk. Sorting signed permutations by reversals and insertions/deletions of contiguous segments. *Journal of Discrete Algorithms*, 1(1):105–122, 2001.

[25] M.D. Braga, E. Willing, and J. Stoye. Genomic distance with DCJ and indels. *Algorithms in Bioinformatics*, pages 90–101, 2010.

[26] M.D. Braga, E. Willing, and J. Stoye. Double cut and join with insertions and deletions. *Journal of Computational Biology*, 18(9):1167–1184, 2011.

[27] P.E.C. Compeau. A simplified view of DCJ-Indel distance. In *Proc. 12th Workshop Algs. in Bioinf. (WABI'12)*, volume 7534 of *Lecture Notes in Comp. Sci.*, pages 365–377, 2012.

[28] J.A. Bailey and E.E. Eichler. Primate segmental duplications: crucibles of evolution, diversity and disease. *nrgenet*, 7(7):552–564, 2006.

[29] M. Lynch. *The Origins of Genome Architecture*. Sinauer, 2007.

[30] D. Sankoff. Genome rearrangement with gene families. *Bioinformatics*, 15(11):909–917, 1999.

[31] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. A pseudo-boolean programming approach for computing the breakpoint distance between two genomes with duplicate genes. In G. Tesler and D. Durand, editors, *Comparative Genomics*, volume 4751 of *Lecture Notes in Comp. Sci.*, pages 16–29. 2007.

[32] S. Angibaud, G. Fertin, I. Rusu, A. Thevenin, and S. Vialette. Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. *J. Comput. Biol.*, 15(8):1093–1115, 2008.

[33] G. Blin, C. Chauve, and G. Fertin. The breakpoint distance for signed sequences. In *Proc. 1st Conf. on Algs. and Comput. Methods for Biochemical and Evolutionary Networks (CompBioNets'04)*, volume 3, pages 3–16, 2004.

[34] D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, volume 1 of *Computational Biology*. 2000.

[35] Z. Chen, B. Fu, and B. Zhu. The approximability of the exemplar breakpoint distance problem. In S.-W. Cheng and C.K. Poon, editors, *Algorithmic Aspects in Information and Management*, volume 4041 of *Lecture Notes in Comp. Sci.*, pages 291–302. 2006.

[36] G. Blin, G. Fertin, F. Sikora, and S. Vialette. The exemplar breakpoint distance for non-trivial genomes cannot be approximated. In S. Das and R. Uehara, editors, *WALCOM: Algorithms and Computation*, volume 5431 of *Lecture Notes in Comp. Sci.*, pages 357–368. 2009.

[37] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *ACM/IEEE Trans. on Comput. Bio. & Bioinf.*, 2(4):302–315, 2005.

[38] C.T. Nguyen, Y.C. Tay, and L. Zhang. Divide-and-conquer approach for the exemplar breakpoint distance. *Bioinformatics*, 21(10):2171–2176, 2005.

[39] J. Suksawatchon, C. Lursinsap, and M. Bodén. Computing the reversal distance between genomes in the presence of multi-gene families via binary integer programming. *Journal of Bioinformatics and Computational Biology*, 05(01):117–133, 2007.

[40] Z. Fu, X. Chen, V. Vacic, P. Nan, Y. Zhong, and T. Jiang. MSOAR: a high-throughput ortholog assignment system based on genome rearrangement. *Journal of Computational Biology*, 14(9):1160–1175, 2007.

[41] M. Remm, C.E.V. Storm, and E.L.L. Sonnhammer. Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J. Comput. Biol.*, 314(5):1041–1052, 2001.

[42] A. Caprara. The reversal median problem. *INFORMS Journal on Computing*, 15(1):93–113, 2003.

[43] E. Tannier, C. Zheng, and D. Sankoff. Multichromosomal genome median and halving problems. In *Algorithms in Bioinformatics*, pages 1–13. Springer Verlag, Berlin, 2008.

[44] A.C. Siepel and B.M.E. Moret. Finding an optimal inversion median: experimental results. In *Algorithms in Bioinformatics*, pages 189–203. Springer Verlag, Berlin, 2001.

[45] B.M.E. Moret, A.C. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. In *Algorithms in Bioinformatics*, pages 521–536. Springer Verlag, Berlin, 2002.

[46] W. Arndt and J. Tang. Improving reversal median computation using commuting reversals and cycle information. *Journal of Computational Biology*, 15(8):1079–1092, 2008.

[47] V. Rajan, A.W. Xu, Y. Lin, K.M. Swenson, and B.M.E. Moret. Heuristics for the inversion median problem. *BMC bioinformatics*, 11(Suppl 1):S30, 2010.

[48] M. Zhang, W. Arndt, and J. Tang. An exact solver for the DCJ median problem. In *Pacific Symposium on Biocomputing*, page 138, 2009.

[49] A.W. Xu and D. Sankoff. Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem. In *Algorithms in Bioinformatics*, pages 25–37. Springer Verlag, Berlin, 2008.

[50] A.W. Xu. A fast and exact algorithm for the median of three problem: A graph decomposition approach. *J. Comput. Biol.*, 16(10):1369–1381, 2009.

[51] M. Shao and B.M.E. Moret. A fast and exact algorithm for the exemplar breakpoint distance. In *Proc. 19th Int'l Conf. Comput. Mol. Biol. (RECOMB'15)*, volume 9029 of *Lecture Notes in Comp. Sci.*, pages 309–322, 2015.

[52] M. Shao, Y. Lin, and B.M.E. Moret. An exact algorithm to compute the DCJ distance for genomes with duplicate genes. In *Proc. 18th Int'l Conf. Comput. Mol. Biol. (RECOMB'14)*, volume 8394 of *Lecture Notes in Comp. Sci.*, pages 280–292, 2014.

[53] M. Shao and B.M.E. Moret. Comparing genomes with rearrangements and segmental duplications. In *Proc. 23rd Int'l Conf. on Intelligent Systems for Mol. Biol. (ISMB'15)*, (accepted, to appear).

[54] M. Shao and Y. Lin. Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion. *BMC Bioinformatics*, 13(Suppl 19):S13, 2012.

[55] M. Shao and B.M.E. Moret. On the DCJ median problem. In *Proc. 25th Ann. Symp. Combin. Pattern Matching (CPM'14)*, volume 8486 of *Lecture Notes in Comp. Sci.*, pages 273–282, 2014.

[56] M. Shao, Y. Lin, and B.M.E. Moret. Sorting genomes with rearrangements and segmental duplications through trajectory graphs. *BMC Bioinformatics*, 14(Suppl 15):S9, 2013.

[57] G. Shi, L. Zhang, and T. Jiang. MSOAR 2.0: Incorporating tandem duplications into ortholog assignment based on genome rearrangement. *BMC bioinformatics*, 11(1):10, 2010.

[58] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2013.

[59] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1):180–210, 1995.

[60] J. Bang-Jensen and G. Gutin. Alternating cycles and trails in 2-edge-coloured complete multigraphs. *Discrete Mathematics*, 188(1–3):61–72, 1998.

[61] C.N. Dewey. Positional orthology: putting genomic evolutionary relationships into context. *Briefings in bioinformatics*, 12(5):401–412, 2011.

[62] A. Caprara and R. Rizzi. Improved approximation for breakpoint graph decomposition and sorting by reversals. *J. of Combin. Optimization*, 6(2):157–182, 2002.

[63] D.A. Christie. A 3/2-approximation algorithm for sorting by reversals. In *Proc. 9th Ann. ACM/SIAM Symp. Discrete Algs. (SODA'98)*, pages 244–252. SIAM Press, Philadelphia, 1998.

[64] G. Lin and T. Jiang. A further improved approximation algorithm for breakpoint graph decomposition. *J. of Combin. Optimization*, 8(2):183–194, 2004.

[65] M.M. Halldórsson. Approximating discrete collections via local improvements. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 160–169. Society for Industrial and Applied Mathematics, 1995.

[66] C.A.J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[67] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica*, 13(1):180–210, 1995.

[68] M.A. Alekseyev and P.A. Pevzner. Breakpoint graphs and ancestral genome reconstructions. *Genome Research*, 19(5):943–957, 2009.

[69] N. El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *Proc. 11th Ann. Symp. Combin. Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes in Comp. Sci.*, pages 222–234. Springer Verlag, Berlin, 2000.

[70] M.A. Alekseyev and P.A. Pevzner. Whole genome duplications and contracted breakpoint graphs. *SIAM J. on Computing*, 36(6):1748–1763, 2007.

[71] N. El-Mabrouk and D. Sankoff. The reconstruction of doubled genomes. *SIAM J. Computing*, 32(3):754–792, 2003.

[72] B. Paten, M. Diekhans, D. Earl, J.S. John, J. Ma, B. Suh, and D. Haussler. Cactus graphs for genome comparisons. *J. Comput. Biol.*, 18(3):469–481, 2011.

[73] K. Fujimura, M.A. Conte, and T.D. Kocher. Circular DNA intermediate in the duplication of Nile Tilapia vasa genes. *PLoS ONE*, 6(12):e29477, 2011.

[74] A. Bergeron, C. Chauve, T. Hartman, and K. St-Onge. On the properties of sequences of reversals that sort a signed permutation. In *Proceedings of JOBIM*, volume 2, pages 99–108. Citeseer, 2002.

[75] M.D.V. Braga, M.-F. Sagot, C. Scornavacca, and E. Tannier. Exploring the solution space of sorting by reversals, with experiments and an application to evolution. *ACM/IEEE Trans. on Comput. Bio. & Bioinf.*, 5(3):348–356, 2008.

[76] M.D.V. Braga and J. Stoye. The solution space of sorting by DCJ. *J. Comput. Biol.*, 17(9):1145–1165, 2010.

[77] K. Durkin et al. Serial translocation by means of circular intermediates underlies colour sidedness in cattle. *Nature*, 482(7383):81–84, 2012.

Curriculum Vitae

Mingfu Shao

EPFL IC IIF LCBB,
INJ 211 (Bâtiment INJ),
Station 14, CH-1015,
Lausanne, Switzerland

Office Phone:	+41 21 693 8183
Mobile Phone:	+41 78 685 6178
Email :	`mingfu.shao@epfl.ch`
Website:	`people.epfl.ch/mingfu.shao`

Education

Sep. 2011—Jul. 2015, Ph.D. of Computer Science (advisor: Prof. Bernard M.E. Moret), Laboratory for Computational Biology and Bioinformatics (LCBB), School of Computer and Communication Sciences (IC), École Polytechnique Fédérale de Lausanne (EPFL), Switzerland.

Sep. 2008—Jul. 2011, M.S. of Computer Science (advisor: Prof. Dongbo Bu), Advanced Computer Research Center, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China.

Sep. 2004—Jul. 2008, B.S. of Computer Science, Department of Computer Science, Beijing Institute of Technology, Beijing, China.

Teaching Experience

Fall 2014, teaching assistant and guest lecturer (topic: iterative improvement, Hungarian algorithm and Hopcroft-Karp algorithm), *Advanced Algorithms*, IC, EPFL.

Fall 2013, teaching assistant and guest lecturer (topic: competitive analysis, paging algorithms), *Advanced Algorithms*, IC, EPFL.

Fall 2012, teaching assistant, *Advanced Algorithms*, IC, EPFL.

Spring 2012, teaching assistant, *Computational Molecular Biology*, IC, EPFL.

Fall 2010, teaching assistant and guest lecturer (topic: PTAS for closest string and closest substring problems), *Algorithm Design and Analysis*, ICT, CAS.

Fall 2009, teaching assistant, *Algorithm Design and Analysis*, ICT, CAS.

Industrial Experience

Oct. 2007—Feb. 2008, internship in Microsoft Research Asia (mentor: Yingnong Dang), Beijing, China.

Academic Activity

Subreviewer, *ISAAC'14, RECOMB'15, ISMB'15, ISBRA'15.*

Awards

2014, Chinese Government Award for Outstanding Self-Financed Students Abroad (500 awardees worldwide and 8 awardees in Switzerland in 2014).

2012, Outstanding Teaching Assistant Award, IC, EPFL.

2009, Merit Student of Advanced Computing Research Center, ICT, CAS.

2008, Beijing Outstanding Graduates.

2007, China Encouragement Scholarship.

2006, First Prize in China Collegiate Physics Contest.

2005, China Scholarship.

Publications

M. Shao and B.M.E. Moret. Comparing genomes with rearrangements and segmental duplications. *Bioinformatics*, 31(12):i329–i338, 2015

M. Shao, Y. Lin, and B.M.E. Moret. An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. *J. Comput. Biol.*, 22(5):425–435, 2015

M. Shao and B.M.E. Moret. A fast and exact algorithm for the exemplar breakpoint distance. In *Proc. 19th Int'l Conf. Comput. Mol. Biol. (RECOMB'15)*, volume 9029 of *Lecture Notes in Comp. Sci.*, pages 309–322, 2015

M. Shao and B.M.E. Moret. On the DCJ median problem. In *Proc. 25th Ann. Symp. Combin. Pattern Matching (CPM'14)*, volume 8486 of *Lecture Notes in Comp. Sci.*, pages 273–282, 2014

M. Shao, Y. Lin, and B.M.E. Moret. An exact algorithm to compute the DCJ distance for genomes with duplicate genes. In *Proc. 18th Int'l Conf. Comput. Mol. Biol. (RECOMB'14)*, volume 8394 of *Lecture Notes in Comp. Sci.*, pages 280–292, 2014

M. Shao, Y. Lin, and B.M.E. Moret. Sorting genomes with rearrangements and segmental duplications through trajectory graphs. *BMC Bioinformatics*, 14(Suppl 15):S9, 2013

M. Shao and Y. Lin. Approximating the edit distance for genomes with duplicate genes under DCJ, insertion and deletion. *BMC Bioinformatics*, 13(Suppl 19):S13, 2012

M. Shao, S. Wang, C. Wang, X. Yuan, S.-C. Li, W.-M. Zheng, and D. Bu. Incorporating ab initio energy into threading approaches for protein structure prediction. *BMC Bioinformatics*, 12(Suppl 1):S54, 2011