The Double Cut and Join Operation and its Applications to Genome Rearrangements

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)

an der Technischen Fakultät
der Universität Bielefeld

vorgelegt von

Julia Zakotnik geb. Mixtacki

April 2008

Betreuer: Prof. Jens Stoye

Gutachter: Prof. Anne Bergeron
Prof. Robert Giegerich
Prof. Jens Stoye

Gedruckt auf alterungsbeständigem Papier – ISO 9706

Meiner Familie

Abstract

The aim of this thesis is to study mathematical models suitable for genome comparison in order to compute the evolutionary distance between two given genomes. We present a new model for genome comparison and apply the unifying double cut and join operation to this genome model. Different variants of the model are studied, yielding simpler formulations of earlier results.

Unlike the traditional sequence-based approach, evolution on the genomic level proceeds by large scale operations which rearrange the order and the direction of chromosomal regions. With an increasing number of sequenced genomes, it is now possible to compare whole genomes instead of short sequences. Different species often share similar genes that were inherited from common ancestors, and large scale mutations are observed in more distantly related genomes. It is widely accepted that the number of genome rearrangements needed to transform one genome into another is a measure of evolutionary distance between two species.

In the reconstruction of evolution based on genome rearrangement, the most common approach is to infer a sequence of rearrangements under the assumption of parsimony, motivating the following combinatorial problem: Given two genomes that have the same gene content, compute a shortest sequence of rearrangements that are needed to transform one genome into the other. The length of such a sequence is the **genomic distance** between these two genomes. For multi-chromosomal genomes, the most common operations are translocations, fusions, fissions, inversions and block interchanges. Remarkably, all these operations can be modeled by a single one, termed the **double cut and join** (DCJ) operation.

Besides genome rearrangements, another important source for genome evolution is whole genome duplication. Compared to the duplication at regional level, genome-wide doubling is a rare and spectacular event. This gives rise to the following combinatorial problem, called the **Genome Halving Problem**: Given a rearranged duplicated genome, find a perfectly duplicated genome such that the distance between these genomes is minimal with respect to some distance measure. Clearly, solutions to this problem depend on the underlying genome model and also on the rearrangement operations that are allowed.

The main contributions of this thesis are summarized as follows:

Graph-based genome model. The basic tools for our genome representation are graphs that are unions of paths and cycles. On this type of graphs, called genome graphs, it is possible to model all rearrangement operations on the most general genome structure that mixes both circular and linear chromosomes. Thus, the graph-based representation is used for modeling genomes, as well as genome rearrangements.

DCJ distance formula and sorting algorithm. With a suitable genome representation, the DCJ operation is applied to the most general type of genomes with a mixed collection of linear and circular chromosomes. A very simple data structure, the adjacency graph, is symmetric with respect to the two genomes under study and is closely related to the visual picture of the genomes themselves. Moreover, this graph simplifies the theory and distance computation considerably and yields an efficient sorting algorithm that can be tailored to optimize the use of certain types of operations.

Unifying treatment of the traditional HP distances. Our main result is an analysis of the relation between the general DCJ model and the three rearrangement models considered in the traditional Hannenhalli-Pevzner (HP) theory: the inversions-only, the translocations-only and a combination of inversions and translocations. The latter model motivates the general HP distance that can be solved using similar concepts as in the unichromomal case. A simple tree structure captures all the delicate features of the general HP problem. Moreover, we show how all three rearrangement models considered in the HP theory can be integrated in the more general DCJ model.

Reconstruction algorithm for duplicated genomes. The Genome Halving Problem under the DCJ operation, where the ancestral genome may contain linear and circular chromosomes, is revisited. Our genome model takes into account constraints required for genomes with only linear chromosomes, as well as the ones for genomes with only circular chromosomes. This yields a new proof and a simple algorithm for reconstructing an ancestral genome. Moreover, using our results, we correct an error in an analysis by Warren and Sankoff.

Contents

Chapter 1

Introduction

One of the major scientific breakthroughs of the 20th century was Watson and Crick's discovery of the molecular structure of *DNA*. The DNA is a helix-shaped molecule whose constituents are two parallel strands of *nuclecotides*, as shown in Fig. 1.1. The rungs of the ladder are formed by two complementary pairs of nucleotides – *adenine* (A) always pairs with *thymine* (T), and *cytosine* (C) always pairs with *guanine* (G). On an abstract level, the DNA is represented as a sequence over the four-letter alphabet $\{A, C, G, T\}$ and the opposite strand can be obtained by taking the reverse complement.

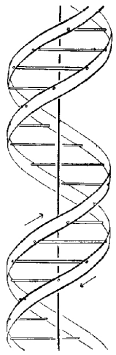


Figure 1.1: DNA double-helix. Picture taken from Watson and Crick [129].

A *genome* is the entire DNA sequence of a living organism and consists of smaller segments called *chromosomes*. A chromosome can be viewed as an ordered sequence of *genes*. A gene is a segment of DNA that is typically involved in producing proteins or encoding functional RNAs. Its *orientation* depends on the DNA strand that it is located on.

Genome rearrangement is a branch of *comparative genomics* that studies gene order among different species. The goal is to infer their phylogenetic relationships and estimate the number of genome rearrangements that have occurred during evolution. Despite a long history in molecular genetics, comparative genomics is still in its early days due to the large amounts of genomic data becoming available. Several key insights and new technologies have made this field take off in new directions and have

produced the need for sound mathematical models and efficient algorithms for genome comparison.

The measurement of evolutionary difference between organisms by comparing of their genomes has become possible through the development of molecular biology and modern genetics. Molecular evolution proceeds in two different forms: *local mutations* and *global rearrangements*. Local mutations such as nucleotide substitutions, deletions and insertions result in local changes in the DNA sequence. Most phylogenetic studies are based on these traditional sequence-based mutations [38, 57, 53].
On the other hand, evolution on the genomic level proceeds by large scale operations which rearrange the order and the orientation of genes along the genome. With an increasing number of sequenced genomes, it is now possible to compare whole genomes instead of short sequences. Different species often share similar genes that were inherited from common ancestors, and large scale mutations are observed in more distantly related genomes. It is widely accepted that the number of genome rearrangements needed to transform one genome into another is a measure of evolutionary distance between two species.

Surprisingly, genome rearrangements of chromosomes have already been identified in close species by Sturtevant [116] in the beginning of the 20th century. Shortly after, Dobzhansky and Sturtevant [42] compared the gene orders in giant chromosomes of strains of *Drosophila pseudoobscura* coming from different geographical regions. Chromosomal segments that are similar enough pair together due to the linkage in hybrids between different strains. If two chromosomes have a segment inverted, then the structure of these chromosomes builds a loop that is visible under the light microscope (Fig. 1.2).

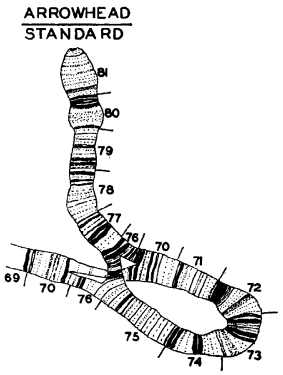


Figure 1.2: Configuration observed in the third chromosome of *Drosophila*. The chromosome coming from the region *Arrowhead* has an inverted segment compared to the *Standard* chromosome that is partitioned into segments from 1 to 100. Picture taken from Dobzhansky and Sturtevant [42].

Genome rearrangement occurs when a chromosome breaks at two or more locations, called *breakpoints*, and the resulting segments are rejoined in a different way. In the chromosome shown in Fig. 1.2, a rearrangement operation has occurred within the segments 70 and 76 and has resulted in $69, (70, 76), 75, 74, 73, 72, 71, (70, 76), 77$. Over

time, such rearrangements accumulate, creating a more and more divergent picture of the formerly identical chromosomes. In contrast to the traditional sequence-based approach in which local mutation accumulate rather quickly, genome rearrangements are usually extremely rare. For this reason, genome rearrangement studies allow for evolutionary reconstructions of more divergent species.

Genome rearrangement have been modeled by a variety of operations such as *inversions*, *translocations*, *fissions*, *fusions*, *transpositions* and *block interchanges*. For uni-chromosomal genomes, the operations are limited to inversions, which reverse a segment of genes, and block interchanges, which exchange two segments of genes. When two segments are adjacent, a block interchange is called a transposition. For multi-chromosomal genomes, the repertoire can be extended by translocations, which exchange segments of genes between two chromosomes. Translocations involving or creating empty chromosomes are called fusions and fissions.

In the reconstruction of evolution based on genome rearrangement, the most common approach is to infer a sequence of rearrangements under the parsimony assumption. This approach was pioneered by Palmer and Herbon [95] who presented a most parsimonious scenario for the mitochondrial genomes of cabbage and turnip and postulated that a minimum of three inversions has occurred during evolution, see Fig. 1.3.

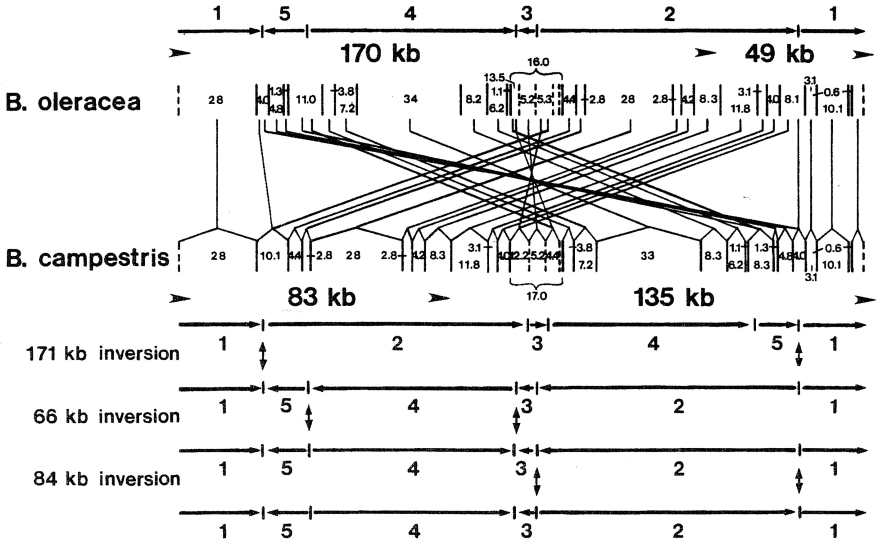


Figure 1.3: The mitochondrial genomes of *B. oleracea* (cabbage) and *B. campestris* (turnip) have essentially the same gene content, but they differ in gene order. The genomes can be divided into five linkage groups and a minimum of three inversions has occurred during evolution. Figure taken from Palmer and Herbon [95].

On an abstract level, the study of genome rearrangement problems involves the combinatorial task of finding a minimum sequence of rearrangements operations to transform one genome into another. The example of transforming cabbage into turnip became famous with the milestone publication by Hannenhalli and Pevzner [63]. At the time when the first genomes were completely sequenced, Hannenhalli and Pevzner presented an exact inversion distance formula and a polynomial time algorithm to compute it. Shortly after, two further papers [62, 60] followed and completed the series by Han-

nenhalli and Pevzner. Since then, genomic distances have been studied extensively and the major results are reviewed in the next section.

1.1 Genomic Distances

In comparative genomics, it is often convenient to represent the genes of a genome by positive integers and a chromosome as a sequence of these numbers. If the orientation of all genes is known, a (plus or minus) sign is associated to each integer. A genome is a set of chromosomes. Genomes consisting of just a single chromosome are *uni-chromosomal*, and genomes with one or more chromosomes are *multi-chromosomal*. Given two genomes that share exactly the same set of genes, and where each gene occurs exactly once, the genomes can be represented as *(signed) permutations* of the set of genes by chaining the chromosomes in an arbitrary order.

During the course of evolution, the genes in a genome can be shuffled around by genome rearrangements that move genes within a chromosome or among chromosomes. This motivates the following combinatorial problem: Given two genomes that have the same gene content, compute a shortest sequence of rearrangements that are needed to transform one genome into the other. The length of such a sequence is the *genomic distance* between these two genomes. Of course, the distance depends on the repertoire of operations that are considered. For example, as shown in Fig. 1.4, the human X chromosome can be transformed into the mouse X chromosome by six inversions, or alternatively by one inversion and three transpositions. As it turns out, not only the set of operations is crucial for the distance computation, but also the orientation of the segments. However, when the paper [8] was published in 1995, the orientation was known only for some segments of the X chromosome.

Among genomic distances, the most often studied distance is the *inversion distance*: In 1982, Watterson *et al.* [130] first formulated the problem of finding the minimum number of inversions required to transform one configuration of genes into another. It took more than ten years until Kececioglu and Sankoff [72] developed the first approximation algorithm for the problem of sorting an *unsigned* permutation by inversions. They also conjectured that this problem is NP-hard, which was proved later by Caprara [35]. Fortunately, the biologically more relevant *signed* version of the problem is polynomial-time solvable. In order to model the orientation of genes, Bafna and Pevzner [9] initiated the study of signed permutations. In 1995, Hannenhalli and Pevzner [63] gave the first inversion distance formula and polynomial-time algorithm for the problem of sorting a signed permutation by inversions using the concepts developed by Bafna and Pevzner [9]. Later, a linear-time algorithm for computing the inversion distance was given by Bader *et al.* [6], whereas the best known algorithm for sorting by inversions is subquadratic [119, 118].

For multi-chromosomal genomes with the same chromosome ends, Hannenhalli [60] gave the first polynomial-time algorithm for sorting by translocations. In 2004, the first linear-time algorithm for computing the translocation distance was given by Li *et al.* [79]. Shortly after, Wang *et al.* [126] presented a quadratic time algorithm for sorting by translocations. However, the last two algorithms rely on Hannenhalli's paper which is erroneous. In 2005, Bergeron *et al.* [21] corrected the error in the original paper

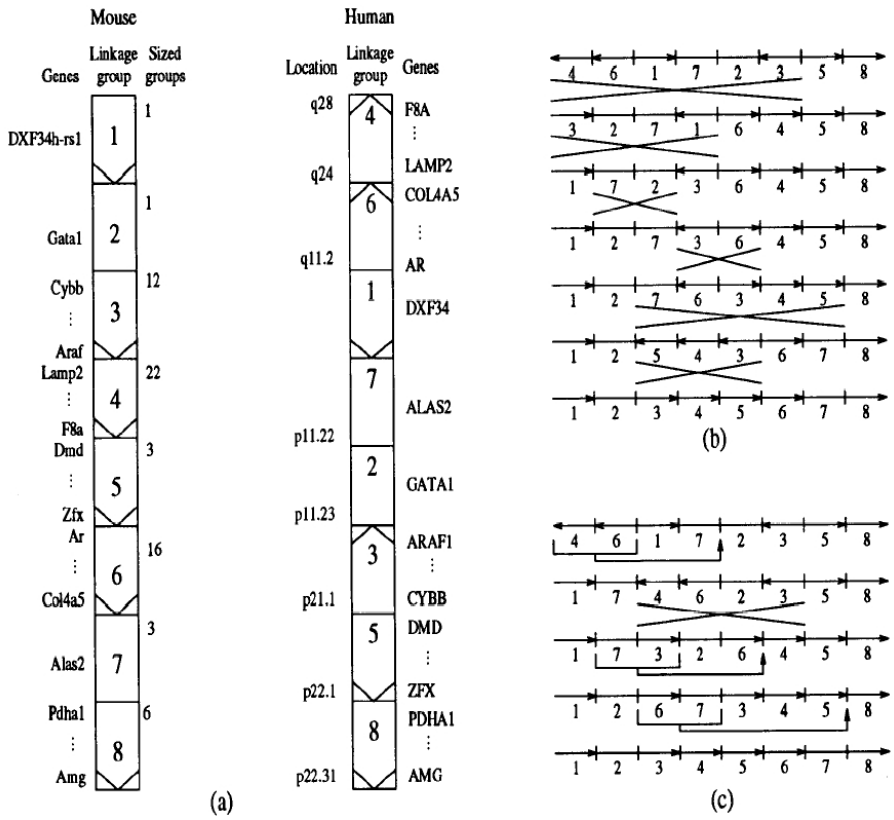


Figure 1.4: Transformation of human X chromosome into mouse X chromosome. The orientation of segments, if known, is indicated by an arrow. **(a)** Conserved linkage groups between human and mouse X chromosomes. **(b)** A most parsimonious scenario assuming that the chromosomes evolve only by inversions. **(c)** A rearrangement scenario involving both inversions and transpositions. Figure taken from Bafna and Pevzner [8].

and presented a new proof for the distance formula and a first correct sorting algorithm. Due to the similarity between the sorting by inversions problem and the sorting by translocations problem [93], Ozery-Flato and Shamir adapted the algorithm by Tannier *et al.* [118] and obtained a subquadratic algorithm for sorting by translocations [92].

In the more difficult case of genomes with different chromosome ends, the distance accounts not only for inversions and translocations, but also for fusions and fissions. In 1995, Hannenhalli and Pevzner [62] presented the first distance formula, called the *general HP-distance*. The rather complicated distance computation requires preprocessing steps such as *capping* and *concatenation* and involves seven parameters. In the last decade, a number of authors pointed to problems in the original formula and algorithm given by Hannenhalli and Pevzner. The first correction was given by Tesler [120]. In 2003, Ozery-Flato and Shamir [91] found a counter-example and modified one of the parameters of the distance formula. Very recently, another correction was presented by Jean and Nikolski [70]. These two recent results have not been implemented in a software tool yet. The only available tool is *GRIMM* that was implemented by Tesler [121] based on [120].

Given their prevalence in eukaryotic genomes [110], the usual choices of operations include translocations, fusions, fissions and inversions. There are some indications that transpositions should also be included in the set of operations [24]. However, the lack of theoretical results showing how to include transpositions in the models led to algorithms that simulate transpositions as sequences of inversions. The problem of sorting by transpositions was first studied by Bafna and Pevzner [10] who derived an approximation algorithm. Since then, many researchers [125, 66, 51, 76, 64] have investigated the problem and several approximation algorithms have been suggested. For combinations of inversions and transpositions, there exist approximation algorithms [65], too, including models where the operations are weighted differently [7]. There is little hope to find a polynomial-time algorithm since sorting by inversions and transpositions on a binary alphabet was proven to be NP-hard [40]. Up to now, the complexity of sorting by transpositions only is still open.
In contrast, the more general problem of sorting by block interchanges has a very simple quadratic-time solution [39] that was further improved by Feng and Zhu [54] and implemented by Martin [83].
Recently, Yancopoulos *et al.* [134] introduced a general operation called *double cut and join* operation (or shortly DCJ operation). Each of the classical operations can be viewed as making up to two cuts in a genome and joining the resulting segments in any order. In addition to inversions and translocations, the DCJ operation also models transpositions and block-interchanges by creating an intermediate circular chromosome that is re-integrated by another DCJ operation. Their general model accounts for the genomic evidence of the coexistence of both linear and circular chromosomes or plasmids in many genomes [37, 124].

Despite the recent efforts to generalize the underlying genome model [22], the computation of the genomic distance still relies on the strong assumption that each gene occurs exactly once in each genome. While this assumption may be appropriate for smaller genomes such as viruses and mitochondria, it is rather unrealistic for more divergent species where genes often have several copies across the genome. One way to overcome this restriction is to use the *exemplar method* studied by Sankoff [103]. The idea is to choose in both genomes one occurrence of each gene, called its *exemplar*, to delete all the other occurrences, and then to minimize the distance between the resulting *exemplar permutations*. Shortly after, it was shown that the problem of choosing exemplars so that the inversion distance between the exemplar permutations is minimized is NP-hard [34]. Another direction [45, 82] is to extend the approach by Hannenhalli and Pevzner [62, 63] to include insertions and deletions; see [47] for a survey chapter.
Compared to the duplication at regional level, genome-wide doubling is definitely the rarest and the most spectacular duplication event. Already in the early 1970s, Susumu Ohno [90] came up with the hypothesis that whole genome duplication has occurred in mammalian evolution. Not without controversy, this question has been addressed several times in the last three decades, both in the biological [1, 59, 74, 41] and in the computational literature [49, 50, 4, 128].
The related combinatorial problem, called the *Genome Halving Problem*, was first introduced in [49]: Given a rearranged duplicated genome, find a perfectly duplicated genome such that the distance between these genomes is minimal with respect to some

distance measure. Clearly, solutions to this problem depend on the underlying genome model and also on the rearrangement operations that are allowed.
El-Mabrouk and Sankoff [50] solved the Genome Halving Problem under the HP distance. Their algorithm for the reconstruction of doubled genomes is far from being trivial and is the final result of a whole series of papers [49, 48, 46]. In addition to the well-known *breakpoint graph*, they introduce further graphs, called *natural graph* and *signature graph*. Later, Alekseyev and Pevzner gave an alternative approach based on the notion of *contracted breakpoint graph* [3] and corrected in [4] an error in the El-Mabrouk-Sankoff analysis. Very recently, Warren and Sankoff [128] studied the Genome Halving Problem under the more general DCJ model. This generalization simplifies the problem, because some of the complicated components of the breakpoint graph, such as *hurdles* and *knots*, can be ignored. However, their solution still relies on the complex concepts introduced by El-Mabrouk and Sankoff.

1.2 Structure of the Thesis

In this thesis, we present a new genome model for genome comparison and we study the double cut and join operation under this model. The main results are (1) a new DCJ distance formula and sorting algorithm, (2) a unifying treatment of the traditional HP distances and (3) a new reconstruction algorithm for duplicated genomes.
The structure of the thesis is as follows. In Chapter 2, we give an introduction to genome rearrangement and their uses in comparative genomics and phylogenetic reconstruction. The relevance of genome rearrangement is exemplified by the human-mouse comparison and the whole genome duplication of the yeast genome. After discussing the two examples, we give a simplified formalization of genomes with coexisting circular and linear chromosomes. Finally, we explain how genomic distance matrices can be used for phylogenetic reconstruction.
In Chapter 3, we present a formal treatment of sorting genomes with a mixture of linear and circular chromosomes by the double cut and join operation. This yields a simple distance formula for the DCJ distance and a linear time sorting algorithm. An overview of our results and their link to related works can be found at the end of the chapter.
In Chapter 4, we study the relation between the general DCJ model and the three rearrangement models considered in the traditional Hannenhalli-Pevzner theory: inversions-only [63], translocations-only [60] and a combination of inversions and translocations [62]. The latter model motivates the general HP distance that can be solved using similar concepts as in the uni-chromosomal case [19]. A simple tree structure captures all the delicate features of the general HP problem. Moreover, we show how all three rearrangement models considered in the HP theory can be integrated in the more general DCJ model. Again, a summary and historical notes are included at the end of the chapter.
In Chapter 5, we study the Genome Halving Problem under the DCJ model where the ancestral genome may contain linear and circular chromosomes. Therefore, in our genome model, we take into account both: constraints required for genomes with only linear chromosomes, as well as the ones for genomes with only circular chromosomes.

Compared to the more general model studied in [128], these requirements on the ancestral genome increase the distance between the genomes. This yields a simple algorithm for reconstructing an ancestral genome. Moreover, by our results, we also correct an error in the Warren-Sankoff analysis. At the end of the chapter, we summarize our results and discuss related work.
Finally, Chapter 6 summarizes the results of this thesis and addresses open questions.

Parts of this thesis have been published in advance. The results on the DCJ distance given in Chapter 3 appeared in [22]. Moreover, the sorting algorithm was implemented and is publicly available at `http://bibiserv.techfak.uni-bielefeld.de/dcj/`.
Parts of Chapter 4 are already published [18, 19, 20, 21, 23] and a journal version about the latest results on the general HP distance is submitted [17].
Furthermore, a conference paper about the results of Chapter 5 is published [84].

1.3 Acknowledgements

This thesis would not have been possible without the help of many people. It is my pleasure that I have the opportunity to express my gratitude to them.
First and foremost, I would like to thank my advisor Jens Stoye for his continuous supervision and mentoring. He has always been supportive through many ups and downs. It helped me a lot that I could always discuss with him on any topic without any hesitation.
I am very grateful to Anne Bergeron for providing a lot of inspiration, useful comments and critical feedback. She is an expert in the field of comparative genomics, but also in finding nice and interesting places to work – and to dine. When I met her for the first time in Minneapolis in fall 2003, I could not imagine that the program that we started would last for so many years. It was a great pleasure to meet and work with her. *Merci beaucoup*!
My working and traveling experiences were further enriched by two research visits to Paris and Vancouver. I thank Mathieu Raffinot for scientific discussions and motivating me to start the PhD program. I also wish to thank Cedric Chauve with whom I was very lucky working, even if only for brief time. Although the results did not find their way into the thesis, they inspired a collaboration with Sebastian Böcker and Katharina Jahn. Thanks to all co-authors who showed me that research is more productive and much more fun in a team.
Sometimes, I have tried to implement algorithms on my own to check our ideas, but this effort has been less successful and has not taken long. I thank three students who have been courageous and patient enough to implement our algorithms: Rafael Friesen, Marcel Martin and Marvin Meinold.
I would like to acknowledge the financial, academic and spiritual support of the International NRW Graduate School in Bioinformatics and Genome Research at Bielefeld University. Apart from a three-year fellowship, it gave me the possibility to attend conferences and the opportunity to meet the Dalai Lama.
Many current and former colleagues made the Genome Informatics group a great environment to work in. In particular, I would like to mention my officemates: Conni

who was also a nice company for sports activities, and perhaps the closest contact to Biology I ever had during my work; Yasmin and Katharina who founded with me the first girls' office in the AGGI group. I am also thankful to Paul Medvedev, Martin Milanic and Wiebke Timm who carefully read earlier drafts of the thesis and gave valuable comments for improvement. And lastly, special thanks go to our secretary Heike Samuel for taking care of administration and for helping in all the things that a student in our group needs help with.
Inspired by the Klosters' winter seminar, members of the AGGI and closely related groups carried on this tradition with annual skiing trips. I very much appreciated the discussions with Sven on the lift, and the accompany of Arne and Thomas – whose office was my hideaway.
Thanks to the mentoring program "mo-ment-mal" of Bielefeld University. Especially, I want to thank my mentor Claudia Nölker for her strategic advice for career development. Moreover, some of us kept in touch even after completion of the program. I am very grateful to Beate, Carola, Nadine, Sonja, Tina, and Vera with whom I shared good times, common concerns, many worries and a lot of discussions that helped me to grow. I hope they will all achieve their goals.
At various times, I was fortunate for the company and encouragement of good friends like Anna, Annette, Elisabeth, Friedi, Jörg, Jule, Julia, Juliane, Maike, Thomas (in alphabetic order).
Last but not least, I am especially thankful to all members of my family who encouraged me throughout my whole life. I would not have made it so far without them. Most importantly, my husband Jure unconditionally loved me during my good and bad times. The last years have not been an easy ride, both academically and personally, and I truly thank Jure for being by my side.

Chapter 2

Genome Rearrangements

This chapter provides an introduction to genome rearrangements and, more specifically, its uses in comparative genomics and phylogenetic reconstruction. We will focus on two specific evolutionary events: *speciation* and *whole genome duplication*. Speciation is an evolutionary process in which organisms from the same species slowly diverge until they form two different species. Thus, speciation starts in offspring genomes that initially have identical gene content and order. In contrast, whole genome duplication creates a new genome with two identical copies of the ancestral genome embedded in it. In both cases, the offspring genomes will diverge over time and their gene order will change due to genome rearrangements. Without selective pressure on gene order, the order of genes would be randomized over time. Thus, similarities in the genomic architecture allow to infer evolutionary relationships among species.

The structure of this chapter is as follows: First, in Section 2.1, we demonstrate the relevance of genome rearrangement with two examples, the human-mouse comparison and the whole genome duplication of yeast. Then, in Section 2.2, two approaches for inferring phylogenetic trees, the Steiner tree reconstruction and the distance matrix based approach, are described.

2.1 Relevance of Genome Rearrangement

The analysis of genome rearrangements was pioneered by Dobzhansky and Sturtevant [42] early in the 20th century. Traditionally, comparative genome analysis begins with identifying *homologous* genes, where segments belonging to different genomes are said to be homologous if they descend from a common ancestor [56]. Homologous genes then give rise to graphical representations in form of *comparative maps*. However, sequencing of whole genomes during the 1990s allowed widespread comparative analysis of gene orders in complete genomic sequences. Even before the completion of the rough drafts of the human [69, 123] and mouse genomes, the human-mouse comparison has been extensively studied in the comparative analysis of mammalian genomes.

2.1.1 Human-Mouse Comparison

Humans and mice diverged from their common ancestor about 100 million years ago. A major question is how many translocations, fusions, fissions and inversions have

occurred during this time. This question is addressed by two different approaches, the statistical and the combinatorial.

The statistical approach was pioneered by Nadeau and Taylor [89] who defined homologous segments as segments with the same gene content and order in both genomes. At that time, only 83 homologous genes were known which led to 13 homologous segments. It turns out that the length of these segments is a very useful parameter. Nadeau and Taylor based their analysis on Ohno's hypothesis [90] that the breakpoints between homologous segments are uniformly distributed. The segment lengths approximate an exponential distribution with density function $f(x) = 1/Le^{-x/L}$, where L is the average length of all segments. Indeed, the average length $L = 8.1$ centiMorgan (cM) of the 13 conserved segments in the human-mouse comparison fits very well the expected lengths distribution for $L = 8$ (see Fig. 2.1), implying that the data is consistent with the assumptions of their analysis.

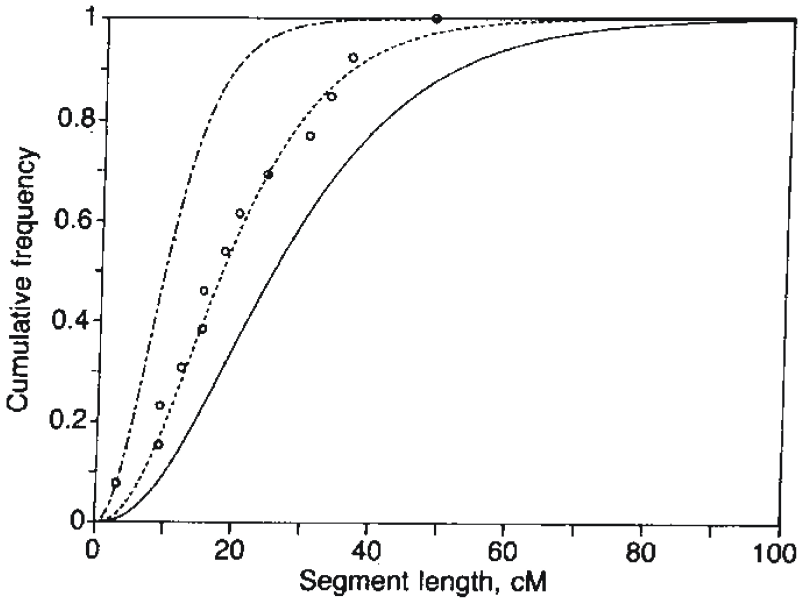


Figure 2.1: The curves illustrate the expected cumulative frequency distribution of segments containing two or more genes for different average segment lengths $L = 4$ cM $(- \cdot -)$, $L = 8$ cM $(\cdots)$, and $L = 12$ cM (——). The circles show the cumulative distribution of segments of the human-mouse comparative map. Picture taken from [89].

Based on a very small number of homologous segments, Nadeau and Taylor estimated the total number of *disruptions of homologies* between human and mouse, denoted by R. Since each disruption increases the number of conserved segments by one, the total number of conserved segments equals the number of disruptions plus the number of chromosomes N, that are present in the last common ancestor of human and mouse. On the other hand, the total number of conserved segments equals the total genome length, denoted by G, divided by the average length L. Altogether, we have that

$$R = (G/L) - N.$$

For the mouse genome with $N = 20$ chromosomes, $G = 1,600$ cM and $L = 8.1$ cM, Nadeau and Taylor estimated that the number of disruptions between human and mouse is approximately 180.
The estimated number of disruptions given by Nadeau and Taylor was confirmed by further studies with progressively increasing levels of resolution and newly discovered homologous segments. More than ten years later, Ehrlich *et al.* [44] studied a human-mouse comparative map containing 1152 homologous genes distributed among 91 *conserved syntenies*, which are segments where the order of genes is not preserved. They estimated 140 synteny disruptions and 180 segment disruptions, yielding 40 intrachromosomal disruptions. Thus, the estimated ratio between translocations and inversions is 140:40. The Nadeau-Taylor model was further improved by estimating not only the number of undiscovered segments, but also their lengths [88], and by modelling the dependency of the segment lengths on each other [68]. Recently, another direction for estimating separately the number of inversions and translocations was suggested by Sankoff and Mazowita [109]. They studied human-mouse comparative maps at different resolutions and showed that the number of estimated translocations is relatively stable, whereas the number of inversions increases with finer resolution.

In the combinatorial approach, the focus is on the process rather than on the end result: one tries to explain differences in gene order in genomes by most parsimonious rearrangement scenarios. Depending on the rearrangement operations that are considered, several genomic distances have been studied in the last two decades, as already mentioned in the previous chapter. Most algorithmic studies of genomic distances rely on *signed permutations* of the integers $\{1, \ldots, N\}$, where N is the number of genes in the comparative map. Therefore, genome rearrangement algorithms require that the orientation of each gene is known, that the genes are unique and that they are linearly ordered. For comparative mapping data, these requirements are not always fulfilled for several reasons: the lack of resolution of maps, missing data or no information about the orientation of genes, or several copies of a gene in a comparative map. These difficulties were addressed by Sankoff *et al.* [113] who studied the application of rearrangement methods to traditional comparative maps.

With the era of completely sequenced genomes, the human-mouse comparison shifted from comparative maps to genomic data. In order to bypass gene finding and homologous identification needed for traditional comparative maps, Pevzner and Tesler [97] used *syntenic blocks* constructed from sequence data as input for their rearrangement algorithm. More precisely, syntenic blocks are regions of high similarity that may nevertheless be interrupted by dissimilar regions or gaps. In order to find syntenic blocks, Pevzner and Tesler developed the *Grimm-Synteny* algorithm. The underlying idea of the algorithm is illustrated by the comparison of the human and mouse X chromosomes, shown in Fig. 2.2. Details can be found in [97].
The Grimm-Synteny algorithm yields 281 syntenic blocks shared by human and mouse of size at least one megabase. Even though the number of syntenic blocks is higher than that predicted by comparative mapping data, their lengths still fit the exponential distribution. In addition, Pevzner and Tesler found evidence for 3,170 *micro-rearrangements* within these blocks that were beyond the resolution of comparative

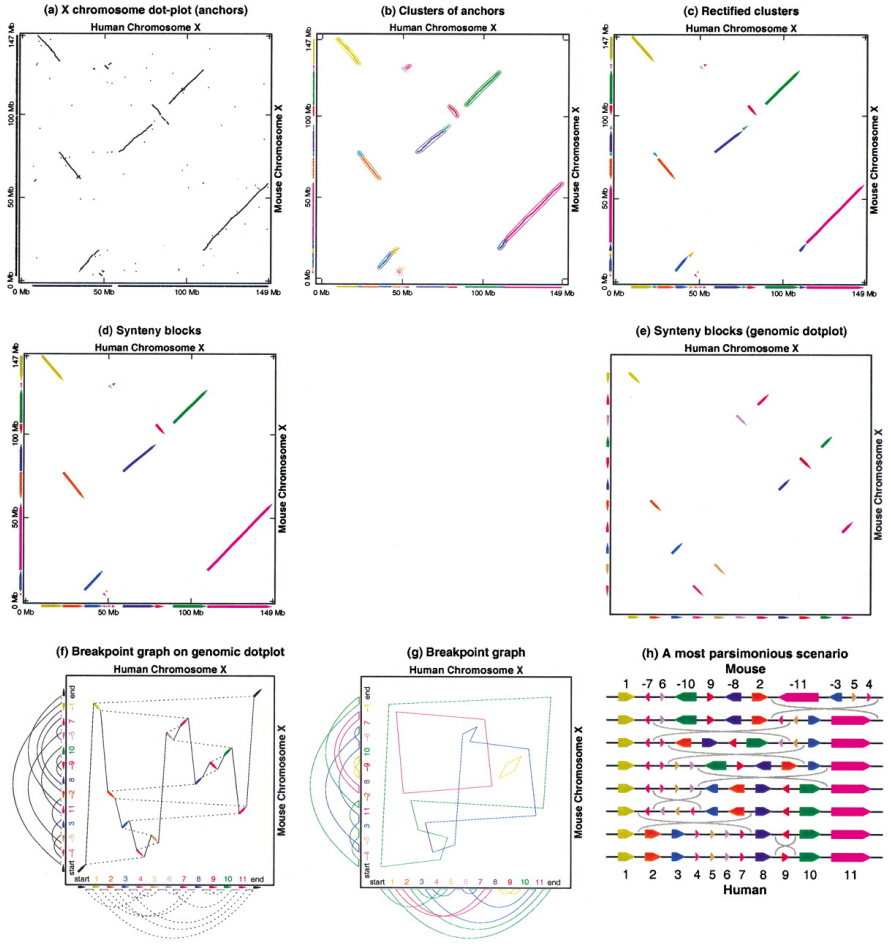


Figure 2.2: Comparison of the human and mouse X chromosome: **(a)** Bidirectional best local similarities are called *anchors*. **(b)** A distance measure in the 2-dimensional dot-plot yields 15 clusters. **(c)** *Rectified clusters* ignore the details of the internal anchor arrangement. **(d)** These rectified clusters are combined into diagonals that correspond to 11 synteny blocks. By assigning to each synteny block an integer from 1 to 11 **(e, f)** and constructing the breakpoint graph **(f, g)**, an optimal rearrangement scenario **(h)** is obtained. Picture taken from [97].

maps. On the other hand, using the 281 syntenic blocks as input for GRIMM [121] shows that 245 *macro-rearrangements* of these blocks have occurred since the divergence of human and mouse.

Inspired by their results from human-mouse comparison [97], Pevzner and Tesler had a closer look at the rearrangement scenario constructed by GRIMM and introduced the concept of *breakpoint reuse* [98, 99]. For example, there are at least three breakpoint reuses in any most parsimonious scenario of the X chromosomes of human and mouse because they consist of the eleven synteny blocks (see Fig. 2.2) and any inversion breaks the chromosome at two points. Thus, genome rearrangement analysis, together with formulas for computing breakpoint reuse, implies that there are 190 breakpoint reuses, yielding short "hidden" synteny blocks of length shorter than one megabase.

Although complete sequencing of genomes has confirmed the number of large-scale rearrangements originally found by comparative gene maps, a debate started about

the huge amount of short synteny blocks that do not fit the exponential distribution of the random breakage model. Pevzner and Tesler suggested the *fragile breakage model* [98, 99]: the probability of a breakpoint in a short *fragile* region follows the Poisson process, while the probability of a breakpoint in a long *solid* region is zero. Assuming that fragile regions are distributed randomly in the genome, both the random breakage and the fragile breakage model predict the same distribution for long synteny blocks. Since then, there has been an ongoing controversy, with more arguments against and in favor of this model being proposed [112, 104, 3, 5].

2.1.2 Whole-Genome Duplication of the Yeast Genome

Besides genome rearrangements, another source of genome evolution is whole genome duplication. First stated by Ohno [90], this hypothesis has found biological evidence among several eukaryotes. An excellent example is the duplication in the yeast genome, recently confirmed by biological experiments [74].
Already in 1997, Wolfe and Shields [131] suggested that the yeast genome has undergone a whole genome duplication 100 million years ago. They identified 55 duplicated regions, containing 376 homologous genes and representing 50% of the genome. Figure 2.3 illustrates the location of the 55 regions within the 16 chromosomes.

$$
\begin{array}{rcl}
\text{I} & : & 2, -1 \\
\text{II} & : & 4, -3, -7,\ 8, -5,\ 6 \\
\text{III} & : & 9, -10, -11 \\
\text{IV} & : & 20,\ 12,\ 12,\ 54,\ 15,\ 21, -3, -13, -16,\ 17, -24, -22, -14, -23, -19,\ 18, -9 \\
\text{V} & : & 28, -25, -27, -4, -26, -13 \\
\text{VI} & : & 55, -36 \\
\text{VII} & : & 36,\ 25,\ 26,\ 32,\ 6, -33,\ 5, -30, -34, -31, -29 \\
\text{VIII} & : & 35, -14, -37, -29, -1 \\
\text{IX} & : & 38,\ 39,\ 27 \\
\text{X} & : & 10,\ 40,\ 41, -28, -42 \\
\text{XI} & : & 42,\ 40,\ 43,\ 35, -41, -52, -38 \\
\text{XII} & : & 53, -53, -31, -55, -16, -18, -17, -45, -30, -15, -44 \\
\text{XIII} & : & 46,\ 44,\ 19, -43, -54, -48, -47, -46 \\
\text{XIV} & : & 49,\ 20,\ 37,\ 50,\ 39, -11 \\
\text{XV} & : & 49,\ 21, -22, -52, -50, -23, -45, -51, -47, -2 \\
\text{XVI} & : & 48,\ 32,\ 33,\ 51,\ 8,\ 24, -7, -34
\end{array}
$$

Figure 2.3: Locations of the 55 duplicated regions on the 16 chromosomes of the yeast genome [131].

Assuming that a genome is duplicated and then rearranged over time, can we reconstruct an ancestral genome from the gene order that we observe today? The key to

1 : $2, -1$
2 : $46,\ 47,\ 48,\ 54,\ 43,\ 35, -41, -40, -42$
3 : $9, -10, -11$
4 : $44,\ 15,\ 21, -22, -14, -23, -19,\ 18,\ 16,\ 13,\ 26,\ 32,\ 33,\ 51,\ 45,$
$17, -24, -8\ 7,\ 3, -4$
5 : $55,\ -36$
6 : $38,\ 39,\ 27,\ 25, -28$
7 : $29,\ 37,\ 50,\ 52, -53$
8 : $49,\ 20,\ 12,\ 31,\ 34,\ 30, -5,\ 6$

Figure 2.4: Ancestral genome suggested by El-Mabrouk and Sankoff [50].

the solution of this question is the structure of the genome right after duplication: it must have been *perfect*, i.e. each chromosome has existed in two identical copies. Of course, there exist many perfectly duplicated genomes that could have been the ancestral genome. Therefore, one wants to reconstruct a genome such that its distance to the current one is minimal. In [50], El-Mabrouk and Sankoff solved this problem under the HP distance. By applying their algorithm to Wolfe and Shields' data, they reconstructed an ancient genome shown in Fig. 2.4. The present-day genome can be obtained from this one by a genome duplication followed by 45 translocations.

2.2 Using Genomic Distances for Phylogenetic Reconstruction

Inferring the evolutionary relationship among different species is a major field in comparative genomics. Gene order comparison for phylogenetic inference has been proven to be powerful for studying the evolution of eukaryotes [33] and prokaryotes [43, 11].
Phylogenetic trees, where each leaf is labeled by a species, are used to represent the history of speciation events. One of the first phylogenetic trees in comparative genomics was given by Dobzhansky and Sturtevant [42]. After comparing the chromosomes of *Drosophila pseudoobscura* coming from different geographical regions, they eventually came up with a phylogenetic tree that represents a rearrangement scenario with altogether 17 inversions, shown in Fig. 2.5.
In the last 30 years, most phylogenetic inference has been based on DNA or protein sequences [38, 57, 53]. After whole genomic sequences became available, inferring phylogenetic relationship based on genome rearrangements became feasible. There exist two major approaches [27]: the reconstruction of a minimal Steiner tree and the distance matrix based methods.
The former approach was pioneered by Hannenhalli *et al.* [61] who studied most parsimonious scenarios for multiple genomes. As a proof of concept, they presented an exhaustive analysis of seven complete and three partial sequences of the herpes virus

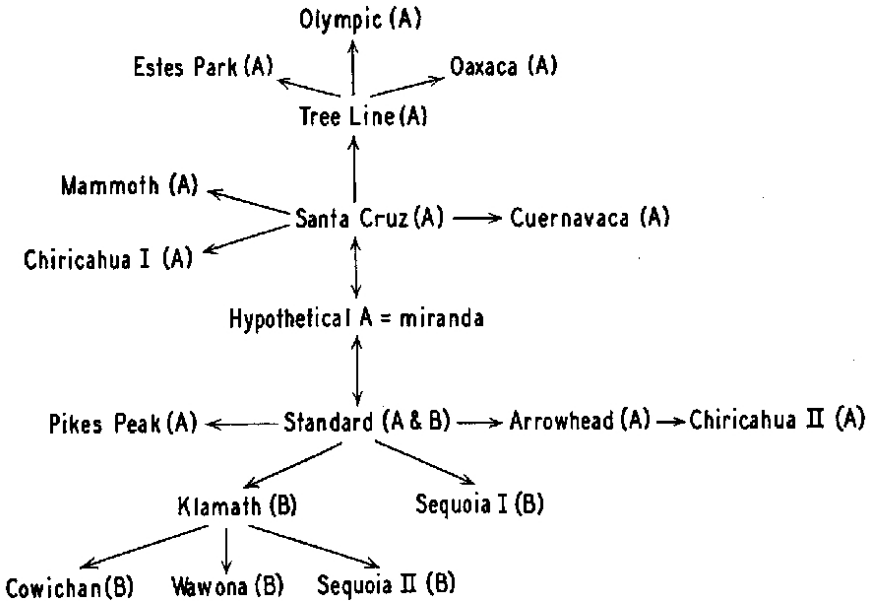


Figure 2.5: Phylogenetic tree of *Drosophila pseudoobscura* coming from different geographical regions. Two species are connected by an arrow if they differ by one inversion. The arrowhead indicates the derived species. The arrow $\leftrightarrow$ is used, if the ancestral species can not be determined. Picture taken from Dobzhansky and Sturtevant [42].

genomes. Their strategy led to phylogenetic trees whose quality was comparable to that of trees based on sequence comparison.

From a computational point of view, the reconstruction approach can be formulated as the problem of finding a phylogenetic tree describing the most parsimonious rearrangement scenario for multiple species. More formally, for a given a set of m genomes, the *Multiple Genome Rearrangement Problem* is to find a tree T whose leaves are the m genomes and to assign $m-2$ genomes to internal nodes such that

$$D(T) = \sum_{(A,B)\in E(T)} d(A,B)$$

is minimized, where d is the genomic distance between two genomes A and B and $E(T)$ is the set of edges of T [111]. In the special case of three genomes and only one internal node, the problem is called the *Genomic Median Problem* [105].

A well-studied distance for phylogenetic reconstruction is the *breakpoint distance* that is defined as the number of breakpoints between two genomes [28]. As in the Nadeau-Taylor model [89], this distance takes into account the number of disruptions and is not based on any specific rearrangement operations. Despite the rather simple computation of breakpoints between two genomes, the Steiner tree problem under the breakpoint distance, and even the median problem, are known to be NP-hard [96].

Beside the breakpoint distance, the inversion distance is of great interest for inferring phylogenetic relationships [115, 87], particularly after a linear time algorithm for its computation was given [6]. Moreover, experimental studies by Moret *et al.* [85] show

that inversion medians are strongly preferable over breakpoint medians. Unfortunately, the median problem for the inversion distance is also NP-hard [36].
Because of the computational difficulties, heuristics have been developed and implemented in programs: BPAnalysis [106] uses a heuristic for the Traveling Salesman Problem to minimize the breakpoint distance between genomes. GRAPPA [87] is a re-implementation of BPAnalysis with algorithmic improvements and inclusion of the inversion distance. MGR [32] is developed for the reconstruction of multi-chromosomal genomes under translocations, fusions, fissions and inversions and uses GRIMM [121] for distance computation. rEvoluzzer [26] searches for rearrangement scenarios with the additional property that gene groups should not be destroyed by inversions.

An alternative to the reconstruction approach is to view the data as a matrix of pairwise distances. Given a set of genomes $\{G_1, \ldots, G_m\}$, such an approach proceeds in two steps [127]:

1. For each pair of distinct genomes G_i and G_j, calculate their pairwise genomic distance $d(G_i, G_j)$ and store the results in a matrix D.

2. Apply a distance-based phylogenetic tree reconstruction method.

For the second step, distance-based methods developed in sequence-based phylogeny can be used, with the Fitch-Margoliash [57] and the Neighbor-Joining [102] methods being the most popular ones. However, the first step is very important as well and mathematical modeling of evolution on the genome level began with the availability of whole genome sequences.
In the early 1990s, this approach has been followed by Sankoff *et al.* [108] who studied the evolution of 16 mitochondrial gene orders from fungi and other eukaryotes. Their analysis is based on an edit distance $E = D+R$, where D is the number of genes present in either one of the genomes, but not in both, and R is the number of inversions and transpositions needed to transform one genome into the other. For computing the number of rearrangements, they developed a branch-and-bound algorithm that was implemented in the program DERANGE.
Ten years later, phylogenetic reconstruction based on rearrangement distances got a boost by the program GRIMM [121] that computes the minimal number of translocations, fusions, fissions and inversions needed to transform one genome into another. For example, GRIMM was used for the analysis of 30 complete γ-proteobacterial genomes by Belda *et al.* [11]. First, they took a subset of 244 genes shared by all genomes and then computed the inversion distance between all pairs of genomes with GRIMM. Their analysis showed a high correlation between the breakpoint and the inversion distance, indicating that inversions are the dominant rearrangement operations in γ-Proteobacteria. The phylogenetic tree obtained by using pairwise inversion distances computed by GRIMM is shown in Fig. 2.6. In the same year, another analysis of 12 γ-Proteobacteria based on the data set of [78] was presented by Blin *et al.* [30]. They handled the multiple copies of genes by computing a *matching* of the genomes, yielding permutations. In their analysis, they used three different distance measures that are based on breakpoints, conserved intervals and common intervals. The resulting phylogenetic trees agree well with the one given by Lerat *et al.* (shown in Fig. 5 in [78]).

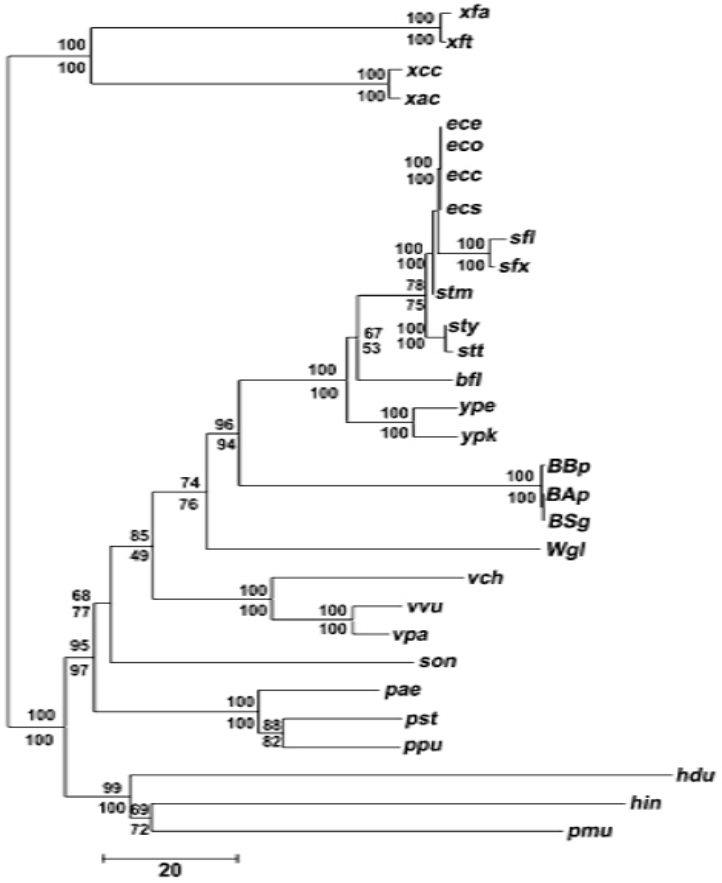


Figure 2.6: Phylogenetic relationship between 30 γ-proteobacterial genomes inferred from an inversion distance matrix. The bar represents 20 inversions. Figure taken from Belda *et al.* [11].

From a theoretical point of view, the quality of reconstructed trees is addressed in experimental studies by Kothari and Moret [75]. Clearly, genomic distances underestimate the true evolutionary distances. Moreover, the choice of distance computation might affect tree reconstruction. To address these issues, Kothari and Moret compare three distance measures, the inversion, the DCJ and the transposition distance. Recall that exact linear time algorithms exist only for the first two distances, whereas only approximation algorithms are known [66] for the transposition distance. Their approach is to simulate data, then to compute the pairwise distances, and finally measure the quality of the trees reconstructed from the distances using neighbor-joining. Surprisingly, inversion and DCJ distances gave very similar results, even on data generated using only transpositions.

Tools for analyzing gene-order data lag behind tools for sequence analysis, because of the almost 20 years of delay for obtaining genomic data and the rather complex mathematics of genome rearrangements. It is apparent, however, that simple and sound algorithms for genomic distance computation are the key for phylogenetic reconstruction based on genome rearrangement. Therefore, the focus of the next chapter is on a simple and unifying treatment of genomic distances.

Chapter 3

The Double Cut and Join Distance

In this chapter we present a simplified formalization of genomes with coexisting circular and linear chromosomes, and a formal treatment of sorting such genomes by the double cut and join operation. We introduce a very simple data structure, the *adjacency graph*, that is symmetric with respect to the two genomes under study and is closely related to the visual picture of the genomes themselves. We also show how the algebraic simplicity of the double cut and join operation yields efficient sorting algorithms.

The chapter is organized in the following manner: we begin by formally stating the DCJ distance problem in the next section. Then, in Section 3.2, we introduce our main construct, the adjacency graph. Using this graph, we give a simple sorting algorithm and a formula for the DCJ distance in Section 3.3. Finally, Section 3.5 concludes with a summary of our results and links them to the existing literature.

3.1 Problem Formulation

An essential ingredient in genome rearrangment studies are graphs where each vertex has degree one or two. Before showing how graphs of this type can be used for modeling genomes and genome rearrangements, we recall some of their properties and the definition of the DCJ distance.

3.1.1 Graphs with Vertices of Degree One and Two

Let G be a graph where each vertex has degree one or two (we allow for loops and multiple edges). We call a vertex of degree one *external* and a vertex of degree two *internal*. An internal vertex connecting edges p and q is denoted by the unordered multiset $\{p, q\}$ and an external vertex incident to an edge p by the singleton set $\{p\}$.

It follows immediately from the definition of G that any connected component of G is either circular, consisting only of internal vertices, or it is linear, consisting of internal vertices bounded by two external vertices, one at each end. We call circular components *cycles* and linear components *paths*. A cycle or path is *even* if it has an even number of edges, otherwise it is *odd*.

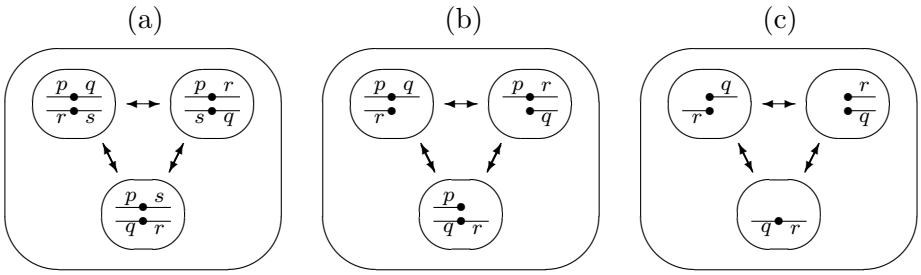
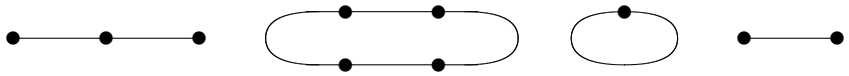


Figure 3.1: Definition of the double cut and join operation. Note that the operations between the two top graphs of part (c) are the identity.

Example 3.1 *The following graph has four vertices of degree one and six vertices of degree two. It has two cycles and two paths, one of which is even and one of which is odd.*



Definition 3.2 *The* double cut and join *(DCJ) operation acts on two vertices u and v of a graph with vertices of degree one or two in one of the following three ways:*

(a) If both $u = \{p, q\}$ and $v = \{r, s\}$ are internal vertices, these are replaced by the two vertices $\{p, r\}$ and $\{s, q\}$ or by the two vertices $\{p, s\}$ and $\{q, r\}$.

(b) If $u = \{p, q\}$ is internal and $v = \{r\}$ is external, these are replaced by $\{p, r\}$ and $\{q\}$ or by $\{q, r\}$ and $\{p\}$.

(c) If both $u = \{q\}$ and $v = \{r\}$ are external, these are replaced by $\{q, r\}$.

In addition, as an inverse of case (c), a single internal vertex $\{q, r\}$ can be replaced by two external vertices $\{q\}$ and $\{r\}$.

Figure 3.1 illustrates the definition.
The DCJ operation, although defined locally on a pair of vertices, has global effects on the connected components of the graph. In order to describe these effects, we use terminology essentially borrowed from biology.
First, consider Figure 3.2. If the two vertices are contained in two different paths and at least one of them is internal, then these paths exchange their ends, which is called a *path translocation*. If both are external vertices of different paths, as in Figure 3.2 (c), then these paths are merged, called a *path fusion*. The inverse of a path fusion is a *path fission*.
The case shown in Fig. 3.3, where both linear and circular components are mixed, is more intricate. If the DCJ operation acts on vertices contained in the same path and at least one of them is internal, then the intermediate part of the path is either reversed, called an *inversion*, or spliced out producing a new cycle, called an *excision*. The inverse operation of an excision is called an *integration*. If both are external vertices

of the same path, as in Fig. 3.3 (c), then a cycle is formed, called a *circularization*. Its opposite is a *linearization*.
If the vertices are contained in the same cycle, or in two different cycles, as shown in Fig. 3.4, then we have either an *inversion*, a *cycle fusion* or a *cycle fission*.

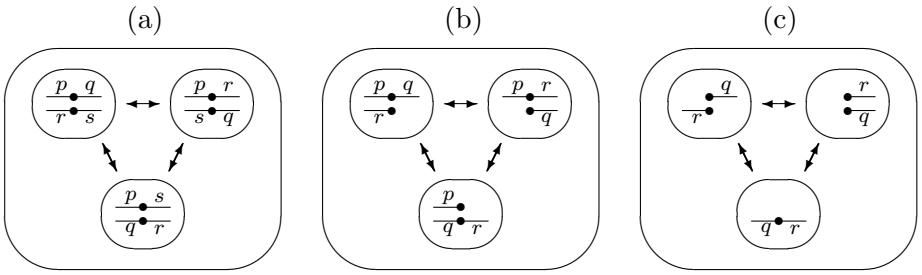


Figure 3.2: The DCJ operation acts on two internal **(a)**, one internal and one external **(b)**, or two external vertices **(c)**. Applying the DCJ operation on one or two paths yields path translocations, fusions and fissions.

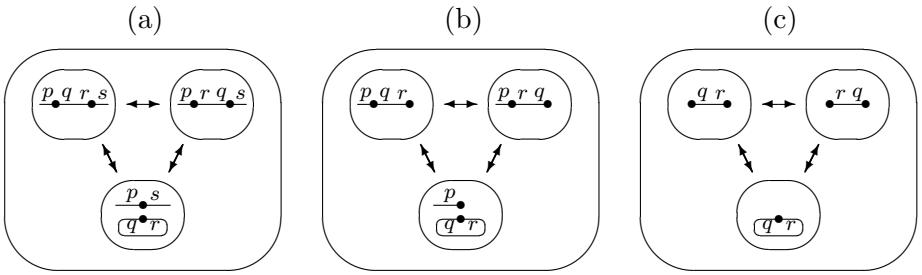


Figure 3.3: The DCJ operation acts on two internal **(a)**, one internal and one external **(b)**, or two external vertices **(c)**. Applying the DCJ operation a single path or a path and a cycle yields inversions, excisions, integrations, circularizations and linearizations.

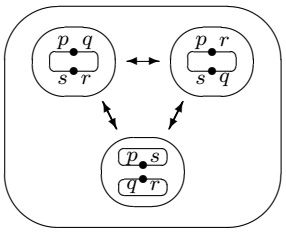


Figure 3.4: Applying the DCJ operation on two internal vertices of a single cycle or of two cycles yields inversions, cycle fusions and fissions.

The following lemma is an immediate consequence of the enumeration of all possible cases in Figures 3.2, 3.3 and 3.4:

Lemma 3.3 *The application of a single DCJ operation changes the number of circular or linear components by at most one.*

3.1.2 Genes, Chromosomes and Genomes

In this section, we introduce our notation of genomes and explain how they are modeled as graphs with vertices of degree one and two.
A *gene* is an oriented sequence of DNA that starts with a *tail* and ends with a *head*. These are called the *extremities* of the gene. The tail of a gene a is denoted by a^t, and its head is denoted by a^h. In biology, the tail of a gene is often called its 3' end and the head its 5' end.
Since two consecutive genes do not necessarily have the same orientation, an *adjacency* of two consecutive genes a and b, depending on their respective orientation, can be of four different types:

$$\{a^h, b^t\}, \{a^h, b^h\}, \{a^t, b^t\}, \{a^t, b^h\}.$$

An extremity that is not adjacent to any other gene is called a *telomere*, represented by a singleton set $\{a^h\}$ or $\{a^t\}$.
Given a set of genes, a *genome* is a set of adjacencies and telomeres of these genes such that the tail or the head of any gene appears in exactly one adjacency or telomere.
Given a genome, one reconstructs its *chromosomes* by representing the telomeres and adjacencies as vertices and then joining for each gene its tail and its head by an edge. Note that the *genome graph* obtained this way is a graph with vertices of degree one or two. The connected paths and cycles represent chromosomes of the genome, which are either linear or circular. Linear chromosomes are flanked by telomeres.
Chromosomes are often represented by lists of gene labels. These lists are obtained by choosing a telomere in a linear chromosome, or an arbitrary gene in a circular chromosome, and then enumerating the gene labels along the chromosome, using positive signs to indicate genes that are read from tail to head and negative signs to indicate genes that are read from head to tail. For linear chromosomes, the enumeration stops at its other telomere, while for circular chromosomes it stops when the initial gene appears for the second time in the list. Positive signs may be omitted where convenient.
In the list notation, an adjacency of two consecutive genes a and b is represented by an ordered pair of genes. According to the four types of an adjacency, the ordered pair can be of the form

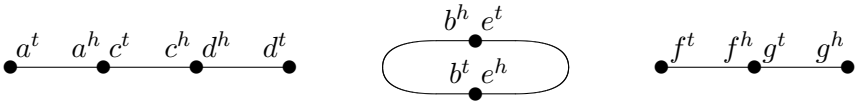
$$(a, b), (a, -b), (-a, b), (-a, -b).$$

Note that this representation depends on the telomere of a linear chromosome, or on the gene of a circular chromosome where we start the list representation.

Example 3.4 *Let*

$$A = \{\{a^t\}, \{a^h, c^t\}, \{c^h, d^h\}, \{d^t\}, \{b^h, e^t\}, \{e^h, b^t\}, \{f^t\}, \{f^h, g^t\}, \{g^h\}\}$$

be a genome with genes $\{a, b, c, d, e, f, g\}$*. The corresponding genome graph is the following:*



One possible list representation of A *is* $\{(a, c, -d), (b, e, b), (f, g)\}$.

Since the genome graph is a graph with vertices of degree one and two, the double cut and join operations defined in Section 3.1.1 can be applied to these graphs. This operation is the same as the one defined, with different notation, by Yancopoulos *et al.* [134].
We can now formulate the problem that we consider:

The DCJ Sorting and Distance Problem. Given two genomes A and B defined on the same set of genes, find a shortest sequence of DCJ operations that transforms A into B. The length of such a sequence is called the *DCJ distance* between A and B, denoted by $d_{DCJ}(A, B)$.

Example 3.5 *Consider the following two genomes that are defined over the set of genes* $\{a, b, c, d, e, f, g\}$*:*

$$\begin{aligned} A &= \{\{a^t\},\{a^h,c^t\},\{c^h,d^h\},\{d^t\},\{b^h,e^t\},\{e^h,b^t\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ B &= \{\{a^h,b^t\},\{b^h,a^t\},\{c^t\},\{c^h,d^t\},\{d^h\},\{e^t\},\{e^h\},\{f^h,g^t\},\{g^h,f^t\}\} \end{aligned}$$

Sorting A into B can, for example, be done in the following five steps, where the affected gene extremities are underlined:

$$\begin{aligned} A &= \{\{a^t\},\{a^h,\underline{c^t}\},\{c^h,d^h\},\{d^t\},\{b^h,e^t\},\{e^h,\underline{b^t}\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ &\quad \{\{\underline{a^t}\},\{a^h,b^t\},\{c^h,d^h\},\{d^t\},\{b^h,\underline{e^t}\},\{e^h,c^t\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ &\quad \{\{e^t\},\{a^h,b^t\},\{c^h,\underline{d^h}\},\{\underline{d^t}\},\{b^h,a^t\},\{e^h,c^t\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ &\quad \{\{e^t\},\{a^h,b^t\},\{c^h,d^t\},\{d^h\},\{b^h,a^t\},\{\underline{e^h},\underline{c^t}\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ &\quad \{\{e^t\},\{a^h,b^t\},\{c^h,d^t\},\{d^h\},\{b^h,a^t\},\{e^h\},\{c^t\},\{\underline{f^t}\},\{f^h,g^t\},\{\underline{g^h}\}\} \\ B &= \{\{a^h,b^t\},\{b^h,a^t\},\{c^t\},\{c^h,d^t\},\{d^h\},\{e^t\},\{e^h\},\{f^h,g^t\},\{g^h,f^t\}\} \end{aligned}$$

The DCJ distance between A and B is $d_{DCJ}(A, B) = 5$. Indeed, the sorting scenario is optimal as we will see in Section 3.3.

3.2 The Adjacency Graph

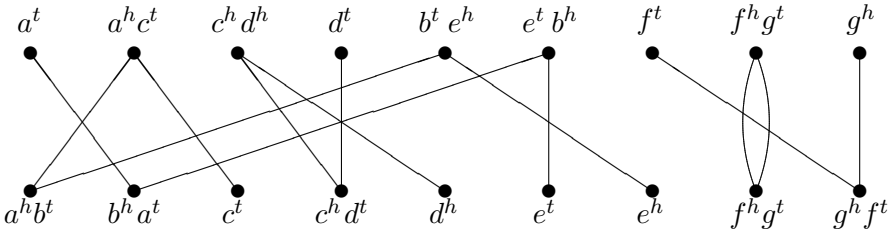
In order to solve the DCJ Distance Problem stated above, another graph of the type discussed in the previous section proves to be useful, this time defined on a pair of genomes A and B.

Definition 3.6 *The* adjacency graph *$AG(A, B)$ is a graph whose set of vertices is the disjoint union of the sets of adjacencies and telomeres of A and B. For each $u \in A$ and $v \in B$ there are $|u \cap v|$ edges between u and v.*

Example 3.7 *The adjacency graph of our two genomes*

$$\begin{aligned} A &= \{\{a^t\},\{a^h,c^t\},\{c^h,d^h\},\{d^t\},\{b^h,e^t\},\{e^h,b^t\},\{f^t\},\{f^h,g^t\},\{g^h\}\} \\ B &= \{\{a^h,b^t\},\{b^h,a^t\},\{c^t\},\{c^h,d^t\},\{d^h\},\{e^t\},\{e^h\},\{f^h,g^t\},\{g^h,f^t\}\} \end{aligned}$$

is the following:



Obviously, every vertex in the adjacency graph has degree one or two, therefore the graph is a union of cycles and paths. Since the graph is bipartite, all cycles have even length.
As we will see below, the adjacency graph allows a simple characterization of many of the properties of sorting by DCJ operations.

Lemma 3.8 *Let A and B be two genomes defined on the same set of N genes, then we have*

$$A = B \quad \textit{if and only if} \quad N = C + I/2$$

where C is the number of cycles and I the number of odd paths in $AG(A, B)$.

Proof Let a be the number of adjacencies and t the number of telomeres in $A = B$, then $N = a + t/2$. The adjacency graph $AG(A, B)$ has $C = a$ cycles and $I = t$ odd paths, hence $N = a + t/2 = C + I/2$.
To show that $N = C + I/2$ implies $A = B$, assume an adjacency graph $G = AG(A, B)$ such that $N = C + I/2$. Let a be the number of adjacencies and t the number of telomeres in A, then $N = a + t/2$. Each cycle in G contains at least one adjacency of A, thus $C \leq a$. Each odd path in G contains exactly one telomere of A, thus $I \leq t$. From $C + I/2 = N = a + t/2$ it follows that $C = a$ and $I = t$. Thus all cycles have length two and all odd paths have length one, which is only possible if the genomes are equal. ■

When a DCJ operation is applied to genome A, it acts on the adjacencies and telomeres of genome A. The same DCJ operation acts *also* on the adjacency graph since the adjacencies and telomeres of genome A are vertices of this graph. Since the adjacency graph is a union of paths and cycles, all the tools and terminology of Section 3.1.1 can be used.
In Lemma 3.3, we showed that the number of circular and linear components can change by at most one when a DCJ operation is applied to a graph that is a union of paths and cycles. In the case of adjacency graphs we also have constraints on the possible changes in the number of odd paths:

Lemma 3.9 *The application of a single DCJ operation to the adjacencies and telomeres of genome A changes the number of odd paths in the adjacency graph $AG(A, B)$ by –2, 0, or 2.*

Proof Consider operations that are path translocations, fusions or fissions (Figure 3.2). Two odd paths can be either transformed into two odd paths, or into one or two paths

of even length. Path(s) of even length(s) can be either transformed into path(s) of even length, or into two paths of odd length. One even and one odd path are always transformed into one even and one odd path. Finally, splitting one odd path always yields an even and an odd path.
Inversions, excisions, integrations, circularizations and linearizations (Figure 3.3) do not change the number of odd paths since all cycles have even length. No paths are involved in the DCJ operations of Figure 3.4. ■

Lemma 3.9 allows to derive the following lower bound for the DCJ distance:

Lemma 3.10 *Let A and B be two genomes defined on the same set of N genes, then we have*

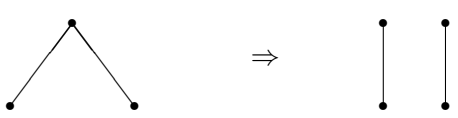
$$d_{DCJ}(A, B) \geq N - (C + I/2)$$

where C is the number of cycles and I the number of odd paths in $AG(A, B)$.

Proof Since none of the cases of the DCJ operation modifies the number of cycles and odd paths simultaneously, this follows immediately from Lemmas 3.3, 3.8 and 3.9. ■

The adjacency graph is also very useful when one wants to find an optimal sequence of sorting operations.
Observe that any pair of edges in the adjacency graph that connect two different vertices of genome A with an adjacency $\{p, q\}$ in genome B can be transformed by a single DCJ operation into a cycle of length two, plus the remaining structure, reduced by the two edges. This operation always increases $C + I/2$ by one since C is increased by one and we have already seen that no DCJ operation can simultaneously change C and I.

Now assume that all adjacencies of genome B are contained in cycles of length two. There might still be pairs of telomeres of B that form an adjacency in A. These adjacencies can be split into two telomeres, thus creating two odd paths of length one each, increasing I by two.



Pseudocode for this greedy sorting procedure is given in Algorithm 1. We will see later in Section 3.4 that the adjacency graph does not need to be constructed explicitly if the genomes are stored in an appropriate way. But first, we will show in the next section that Algorithm 1 provides an optimal solution to the DCJ sorting and distance problem.

Algorithm 1 (Greedy sorting by DCJ)

```
1: for each adjacency {p, q} in genome B do
2:     let u be the element of genome A that contains p
3:     let v be the element of genome A that contains q
4:     if u ≠ v then
5:         replace u and v in A by {p, q} and (u \ {p}) ∪ (v \ {q})
6:     end if
7: end for
8: for each telomere {p} in genome B do
9:     let u be the element of genome A that contains p
10:    if u is an adjacency then
11:        replace u in A by {p} and (u \ {p})
12:    end if
13: end for
```

3.3 Computing the DCJ Distance

Theorem 3.11 *Let A and B be two genomes defined on the same set of N genes, then we have*

$$d_{DCJ}(A, B) \;=\; N - (C + I/2)$$

where C is the number of cycles and I the number of odd paths in $AG(A, B)$. An optimal sorting sequence can be found in $O(N)$ time by Algorithm 1.

Proof Lemma 3.10 together with the fact that Algorithm 1 increments in each iteration either C by one or I by two prove the distance formula.
The linear time complexity follows from the fact that our genome representation allows to find and perform each sorting operation in constant time and the DCJ distance is never larger than N. ■

For example, our genomes

$$\begin{aligned} A &= \{\{a^t\}, \{a^h, c^t\}, \{c^h, d^h\}, \{d^t\}, \{b^h, e^t\}, \{e^h, b^t\}, \{f^t\}, \{f^h, g^t\}, \{g^h\}\} \\ B &= \{\{a^h, b^t\}, \{b^h, a^t\}, \{c^t\}, \{c^h, d^t\}, \{d^h\}, \{e^t\}, \{e^h\}, \{f^h, g^t\}, \{g^h, f^t\}\} \end{aligned}$$

have seven genes and their adjacency graph $AG(A, B)$ has $C = 1$ cycle, two even paths and $I = 2$ odd paths, as shown in Example 3.7. Thus, we have

$$d_{DCJ}(A, B) \;=\; N - (C + I/2) \;=\; 7 - (1 + 1) \;=\; 5.$$

Recall that an optimal sorting scenario with $d_{DCJ} = 5$ DCJ operations was given in Example 3.4.

3.4 Algorithm Details

The adjacency graph can easily be constructed as shown in Algorithm 2. Let N be the number of genes in genomes A and B, respectively. Then Algorithm 2 takes $O(N)$ time and uses $O(N)$ space if the genomes are stored in a data structure where, for each gene extremity, one can access in constant time the adjacency or telomere that contains it.

Algorithm 2 (Construction of the adjacency graph)

1: create a vertex for each adjacency and each telomere in genomes A and B
2: **for each** adjacency $\{p, q\}$ in genome A **do**
3: create an edge connecting $\{p, q\}$ and the vertex of genome B that contains p
4: create an edge connecting $\{p, q\}$ and the vertex of genome B that contains q
5: **end for**
6: **for each** telomere $\{p\}$ of genome A **do**
7: create an edge connecting $\{p\}$ and the vertex of genome B that contains p
8: **end for**

This can be done by using two tables for each genome. More precisely, we store in a table with two rows of length at most $2N$ the adjacencies and telomeres of a genome. Another table with two rows of length N stores for each gene the columns of the first table containing its head and its tail. Once again, consider the genome

$$A = \{\{a^t\}, \{a^h, c^t\}, \{c^h, d^h\}, \{d^t\}, \{b^h, e^t\}, \{e^h, b^t\}, \{f^t\}, \{f^h, g^t\}, \{g^h\}\}$$

from Example 3.4. The two tables are shown in Tables 3.1 and 3.2.

	1	2	3	4	5	6	7	8	9
first	a^t	a^h	c^h	d^t	b^h	e^h	f^t	f^h	g^h
second	–	c^t	d^h	–	e^t	b^t	–	g^t	–

Table 3.1: Table storing the adjacencies and telomeres of genome A. Adjacencies have two entries, telomeres just one.

	a	b	c	d	e	f	g
head	2	5	3	3	6	8	9
tail	1	6	2	4	5	7	8

Table 3.2: Table storing for each gene in A the location of its head and its tail in Table 3.1.

Finally, the number of cycles and paths of the adjacency graph $AG(A, B)$ are computed in linear time by Algorithm 3. The idea is to mark all vertices of $AG(A, B)$ and to process them in left-to-right order. Each time an unmarked vertex s is detected, we first check whether this adjacency belongs to a cycle or to a path. Therefore, we follow its edges until we reach a telomere or return to s (see lines 6-8 of Algorithm 3). In the first case, we increase the number of paths (line 10), and in the second case the number of cycles (line 12). Then, we go backward through the path/cycle and mark all vertices that belong to it (lines 15 and 18). Since the vertices of each path/cycle are traversed at most twice and the total number of vertices is in $O(N)$, the running time of Algorithm 3 is linear. Note that one can also compute the number of even and odd paths by a slight modification of lines 16 to 19, without affecting the running time.

3.5 Summary and Historical Notes

We have shown that, with a suitable representation, it is possible to model all rearrangement operations on the most general genome structure that mixes both circular and linear chromosomes.

It is worth mentioning that our distance formula is equivalent to the result $d_{DCJ} = b - c$ given by Yancopulos *et al.* [134], where b is the number of breakpoints and c is the number of cycles of the breakpoint graph after appropriate *capping* of the linear

Algorithm 3 (Compute the number of cycles and paths of $AG(A, B)$)

```
1: initially all vertices of AG(A,B) are unmarked
2: c ← 0, p ← 0 (* counters for the number of cycles and paths *)

3: for each vertex s of AG(A,B) do
4:    if s is not marked then
        (* Check whether s belongs to a cycle or a path *)
5:      choose one edge incident to s and call the other vertex of this edge v
6:      while v ≠ s and v is not a telomere do
7:        u ← v, v ← vertex incident to U
8:      end while
9:      if v is a telomere then
10:       p ← p + 1
11:     else
12:       c ← c + 1
13:     end if
        (* Go back through the path/cycle and mark all its vertices *)
14:     s ← v, v ← u
15:     mark s and v
16:     while v ≠ s and v is not a telomere do
17:       u ← v, v ← vertex incident to the other edge of u
18:       mark v
19:     end while
20:   end if
21: end for
```

chromosomes. To see this, let l_A and l_B be the number of linear chromosomes in genomes A and B, respectively. Then the total number of breakpoints, as defined in [134], is $b = N + l_B + aa = N + l_A + bb$ where aa is the number of even paths that start and end in genome A and bb is the number of even paths that start and end in genome B. The number of cycles is $c = C + I + E$ where C is the number of cycles, I the number of odd paths and E the number of even paths in the adjacency graph $AG(A, B)$ as defined in Section 3.2. Obviously $E = aa + bb$. Moreover, each linear chromosome is associated to two path ends, thus the number of linear chromosomes equals the number of paths, $l_A + l_B = I + E$. Together this implies that $2b = 2N + 2E + I$, giving $b - c = N - C - I/2$.

Another related rearrangement problem has recently been studied in [81], where operations are fusions and fissions between circular unsigned chromosomes, and block interchanges within a circular unsigned chromosome. The authors assign equal weight to the three operations, even though a block interchange requires two DCJ operations, and propose an $O(N^2)$ time algorithm to sort these circular genomes. Their algorithm first applies fusions to both source and target genome, until they have two genomes whose chromosomes have equal gene content. These fusions can be identified in linear time by a search of the adjacency graph. They then sort the resulting genomes by block interchanges using an $O(N^2)$ time algorithm described in [80]. This can be done with the same time complexity, but with elementary means, using a modification of our Algorithm 1 where every intermediate chromosome created by a fission is immediately

re-absorbed in the next step, such that only block interchanges are performed. The modification consists of searching, in the newly created circular chromosomes, a pair of genes that are adjacent in the target genome, but on different chromosomes in the source genome.

Chapter 4

HP Distances via the Double Cut and Join Distance

The main purpose of this chapter is to study the relation between the DCJ distance and other existing genomic distances. In a series of papers, Hannenhalli and Pevzner considered three rearrangement models: inversions only [63], translocations only [60] and a combination of inversions and translocations [62]. The latter is motivated by the Hannenhalli-Pevzner (HP) distance problem: Given two genomes whose chromosomes are linear, calculate the minimum number of inversions and translocations that transform one genome into the other. In this chapter, we present a new distance formula based on a simple tree structure that captures all the delicate features of this problem in a unifying way. Moreover, we show how all three rearrangement models considered in the HP theory can be integrated in the more general DCJ model.

This chapter is organized as follows. The next section introduces the notation needed to formulate the HP distance problem. In Section 4.2, we establish the conditions under which the DCJ distance and the HP distance are equal. The general case is treated in Section 4.3, where we introduce the basic concepts and the tree needed for the computation of the HP distance, and we give a new proof and formula for the Hannenhalli-Pevzner theorem. Two further genomic distances, the inversion and the translocation distance, are discussed in Section 4.4. Section 4.5 describes the algorithms needed for the HP distance computation. Finally, Section 4.6 concludes with a summary and historical notes.

4.1 Problem Formulation

In this chapter, we represent a *gene* by a signed integer where the sign represents its orientation. A *chromosome* is a sequence of genes and does not have an orientation. A *genome* is a set of chromosomes. We distinguish between *uni-chromosomal* genomes consisting of just a single chromosome and *multi-chromosomal* genomes consisting of one or more chromosomes.
In HP sorting, the shapes of chromosomes are restricted to linear forms throughout the sorting process. In order to be able to model this behavior, we need to introduce

explicit unsigned telomere markers. This leads to a slight modification in the definition of adjacencies, extending it to the case of genes that are *adjacent* to telomere markers. A linear chromosome will thus be represented by an ordered sequence of k signed genes, flanked by two telomere markers:

$$X = (\circ, x_1, \ldots, x_k, \circ).$$

Since a chromosome does not have an orientation, we can *flip* a chromosome X into $-X = (-\circ, -x_k, \ldots, -x_1, -\circ)$ and still have the same chromosome. Note that, by definition, $\circ = -\circ$.

An *interval* $(l, \ldots, r)$ in a genome is a sequence of at least two consecutive genes or telomere markers within a chromosome. The set $\{l, -r\}$ is the set of *extremities* of the interval $(l, \ldots, r) = (-r, \ldots, -l)$. In this context, we use the notation of representing an adjacency as an interval (x, y) of length 2. An adjacency that contains a telomere marker is called a *telomere*. It is sometimes convenient to represent an adjacency (x, y) by its set of extremities $\{x, -y\}$. In the following, we use the term adjacency also when referring to this set. It will be clear from the context if we mean the set of genes or two consecutive genes.

Looking at a single chromosome, the most common rearrangement operations are *inversions*. An inversion of an interval $(x_i, \ldots, x_j)$ of a chromosome

$$X = (\circ, x_1, \ldots, x_{i-1}, x_i, \ldots, x_j, x_{j+1}, \ldots, x_k, \circ)$$

reverses the order of the elements of the interval while changing their signs, yielding

$$X' = (\circ, x_1, \ldots, x_{i-1}, -x_j, \ldots, -x_i, x_{j+1}, \ldots, x_k, \circ).$$

Another type of genome rearrangements that occur in multichromosomal genomes are *translocations*. A translocation transforms two linear chromosomes

$$X = (\circ, x_1, \ldots, x_i, x_{i+1}, \ldots x_k, \circ) \quad \text{and} \quad Y = (\circ, y_1, \ldots, y_j, y_{j+1}, \ldots, y_l, \circ)$$

into linear chromosomes

$$X' = (\circ, x_1, \ldots, x_i, y_{j+1}, \ldots, y_l, \circ) \quad \text{and} \quad Y' = (\circ, y_1, \ldots, y_j, x_{i+1}, \ldots, x_k, \circ).$$

Fusions and fissions are translocations where chromosomes consisting of just two telomeres are involved or created. All other translocations are *internal*.

The HP Distance Problem. Given two multichromosomal genomes A and B defined on the same set of genes, compute the minimum number of inversions, translocations, fusions and fissions needed to transform A into B. This number is called the *HP distance* between A and B, denoted by $d_{HP}(A, B)$.

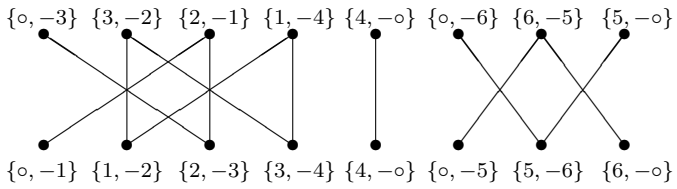
The solution of this problem is based on an extension of a simplified problem where the genomes to be compared are unichromosomal. In this case, the rearrangement operations are limited to inversions only. Another well known restriction of the general problem deals with genomes that have the same telomeres and that can be sorted by internal translocations only. Both variants of the general problem will be discussed in Section 4.4, but first we will present the solution for the general HP distance problem.

4.2 Components and Oriented Sorting

The goal of this section is to characterize the genomes for which we have $d_{DCJ} = d_{HP}$. We begin by recalling some of the results of the previous chapter. In particular, we define the adjacency graph in a slightly different way. Moreover, we introduce the notion of components.

4.2.1 Oriented DCJ Operations

In a given genome, any gene g is the extremity of two adjacencies, one as $+g$, and one as $-g$, in the set notation. For genomes A and B on the same set of genes, this remark allows to view the adjacency graph from a different perspective than as in Definition 3.6: In $AG(A, B)$, each gene g defines two edges, one connecting the two adjacencies of genome A and B in which g appears as extremity $+g$, and one connecting the two adjacencies in which g appears as extremity $-g$.
For example, the adjacency graph of the genomes $A = \{(\circ, 3, 2, 1, 4, \circ), (\circ, 6, 5, \circ)\}$ and $B = \{(\circ, 1, 2, 3, 4, \circ), (\circ, 5, 6, \circ)\}$ is then the following:



As stated in Definition 3.2, a *DCJ operation* applied to two adjacencies of the same genome disconnects the incident edges of the adjacency graph and reconnects them in one of the possible other ways. Recall that the *DCJ distance* between genomes A and B, denoted by $d_{DCJ}(A, B)$, is the minimum number of DCJ operations necessary to transform genome A into genome B. In the preceding section, we have shown in Theorem 3.11 that for two genomes A and B on the same N genes

$$d_{DCJ}(A, B) \;=\; N - (C + I/2)$$

where C is the number of cycles and I the number of paths of odd length in $AG(A, B)$. A DCJ operation that reduces the DCJ distance by 1 is called *DCJ-sorting*. Using Theorem 3.11, we have the following property of DCJ-sorting operations, based on the fact that a DCJ operation acts on at most two paths or cycles, and produces at most one new path or cycle:

Corollary 4.1 *A DCJ-sorting operation acts on a single path or cycle, or on two even paths of the adjacency graph.*

Proof A DCJ operation reduces the DCJ distance by one if it increases the number of cycles C by one or the number of odd paths by two. Recall that no DCJ operation can modify the number of cycles and odd paths simultaneously. First, suppose that the number of cycles is increased. Then the DCJ operation acts either on a single path, called an excision in Fig. 3.3, or on a single cycle, called a cycle fission in Fig. 3.4. Now,

suppose that the number of odd paths is increased. Then the DCJ operation is a path translocation, shown in Fig. 3.2, and the two paths affected are even. ■

Some DCJ operations can create intermediate circular chromosomes, even if both genomes A and B are linear, and we will want to avoid them in the HP model. The following definition is a generalization of a classical concept in rearrangement theory, *oriented operations* [63]:

Definition 4.2 *A DCJ-sorting operation is* oriented *if it does not create circular chromosomes.*

For two linear genomes, oriented operations are necessarily inversions, translocations, fusions or fissions. These operations are also called *HP operations*. Since DCJ operations are more general than HP operations, we have the following lower bound:

Proposition 4.3 *For two linear genomes A and B, we have that*

$$d_{DCJ}(A, B) \leq d_{HP}(A, B).$$

Proof Any optimal HP sorting scenario is also a DCJ scenario. Thus, by extending the set of operations, the distance cannot increase. ■

4.2.2 Components

We introduce here the notion of *components*. They roughly correspond to the classical concept of components, which are connected components in a graph called the *overlap graph* [63]. In the context of adjacency graphs, we will prove that they are unions of paths and cycles.

Definition 4.4 *Given two genomes A and B, an interval $(l, \ldots, r)$ of genome A is a* component *relative to genome B if there exists an interval in genome B with the same extremities, with the same set of genes, and that is not the union of two such intervals.*

Example 4.5 *Let*

$$\begin{aligned} A &= \{(\circ, 2, 1, 3, 5, 4, \circ), (\circ, 6, 7, -11, -9, -10, -8, 12, 16, \circ), (\circ, 15, 14, -13, 17, \circ)\}, \\ B &= \{(\circ, 1, 2, 3, 4, 5, \circ), (\circ, 6, 7, 8, 9, 10, 11, 12, \circ), (\circ, 13, 14, 15, \circ), (\circ, 16, 17, \circ)\}. \end{aligned}$$

The components of genome A relative to genome B are: $(\circ, 2, 1, 3)$, $(3, 5, 4, \circ)$, $(\circ, 6)$, $(6, 7)$, $(-11, -9, -10, -8)$, $(7, -11, -9, -10, -8, 12)$, $(\circ, 15, 14, -13)$ *and* $(17, \circ)$.

Note that components of length 2 are the same adjacencies in both genomes, up to flipping of a chromosome. Such components are called *trivial components*. All other components are *non-trivial*.

Two components are *nested* if one is included in the other and their extremities are different. As the following lemma shows, two components cannot share a telomere:

Lemma 4.6 *If* $(\circ, \ldots, r_1)$ *and* $(\circ, \ldots, r_1, \ldots, r_2)$ *are two components, then* $r_1 = r_2$, *and if* $(l_1, \ldots, l_2, \ldots, \circ)$ *and* $(l_2, \ldots, \circ)$ *are two components then* $l_1 = l_2$.

Proof Suppose that $(\circ, \ldots, r_1)$ and $(\circ, \ldots, r_1, \ldots, r_2)$ are two components. Since the corresponding intervals in genome B, $(\circ, \ldots, r_1)$ and $(\circ, \ldots, r_2)$, share the same gene content, the interval $(r_1, \ldots, r_2)$ shares the same gene content in both genomes, thus $(r_1, \ldots, r_2)$ is a component, and $(\circ, \ldots, r_1, \ldots, r_2)$ is the union of two components, a contradiction. The second statement has a similar proof. ■

It is further known that two components cannot overlap on two or more elements. We thus have the following generalization of a statement from [24]:

Proposition 4.7 *Two components are either disjoint, nested, or overlap on exactly one gene.*

Proof Consider two components $\mathcal{C}$ and $\mathcal{C}'$ of the form

$$\mathcal{C} = (l, \ldots, r) \quad \text{and} \quad \mathcal{C}' = (l', \ldots, r').$$

Suppose first that the interval $(l', \ldots, r')$ is nested in the interval $(l, \ldots, r)$ with $l = l'$. By Lemma 4.6, two components cannot share the same telomere implying that $l \neq \circ$. Since $\mathcal{C}'$ is a component, the interval $(l', \ldots, r')$ has the same gene content in both genomes. Hence, the interval $(r', \ldots, r)$ has also the same gene content in both genomes. This contradicts the fact that the component $\mathcal{C}$ is not the union of two shorter components. The case where $(l', \ldots, r')$ is nested in $(l, \ldots, r)$ with $r' = r$ can be treated similarly.
Now, suppose that the components $\mathcal{C} = (l, \ldots, r)$ and $\mathcal{C}' = (l', \ldots, r')$ overlap on more than one element. Without loss of generality, we can assume that both genomes have an interval of the form

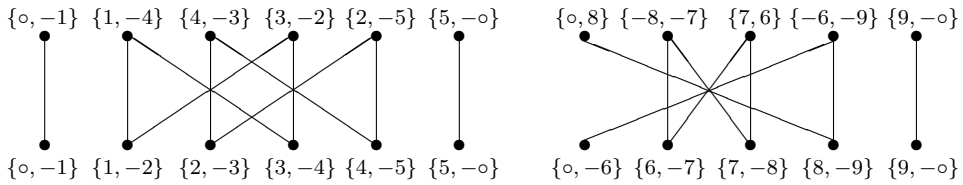
$$(l, \ldots, l', \ldots, r, \ldots, r')$$

with the same gene content. Since the interval $(l', \ldots, r')$ has the same gene content in both genomes, also the interval $(l, \ldots, l')$ must have the same gene content in both genomes. Thus, $\mathcal{C} = (l, \ldots, r)$ is the union of two shorter components, which leads to a contradiction. ■

Components can be partially ordered by inclusion, and, by Proposition 4.7, overlapping components will have the same parent (if they have one). An adjacency that is contained in one or more components, *properly belongs* to the smallest of these. This definition is not ambiguous since overlapping components overlap on exactly one gene.

Definition 4.8 *The* adjacency graph of a component $\mathcal{C}$ *is the subgraph of the adjacency graph of genomes* A *and* B *induced by the adjacencies that properly belong to* $\mathcal{C}$.

As an example, we consider the genomes $A = \{(\circ, 1, 4, 3, 2, 5, \circ), (\circ, -8, 7, -6, 9, \circ)\}$ and $B = \{(\circ, 1, 2, 3, 4, 5, \circ), (\circ, 6, 7, 8, 9, \circ)\}$ and their adjacency graph $AG(A, B)$:



The adjacencies $\{1, -4\}$, $\{4, -3\}$, $\{3, -2\}$, and $\{2, -5\}$ properly belong to the component $(1, \ldots, 5)$. Thus, the adjacency graph of $(1, \ldots, 5)$ consists of two cycles. There are two other non-trivial components: $(\circ, \ldots, -9)$ consisting of one path and $(-8, \ldots, -6)$ consisting of one cycle.

An important property of the adjacency graph is the following:

Proposition 4.9 *The adjacency graph of a component is the union of one or more connected components of the adjacency graph of genomes A and B.*

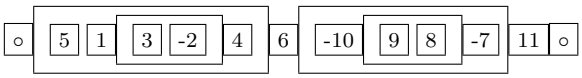
Proof Let $\mathcal{C} = (l, \ldots, r)$ be a component. Since it has the same gene content and the same extremities as the corresponding interval in genome B, its adjacency graph is the union of one or more connected components of the adjacency graph of genomes A and B.
Each component that is nested in $\mathcal{C}$ is also a union of connected components of the adjacency graph of genomes A and B, and none of them contains an adjacency that properly belongs to $\mathcal{C}$. We can thus remove them without compromising the connectivity of the adjacency graph of $\mathcal{C}$. ■

4.2.3 Oriented Sorting

In this section we will characterize genomes for which the DCJ distance and the HP distance are equal. We first consider sorting involving only one chromosome. In particular, we apply well known results from the inversion theory that are obtained by working on *permutations*.
Since the naming and orientation of genes is relative, we can always assume that all genes in a chromosome of genome B are positive and in increasing order. The proper adjacencies of a component $\mathcal{C} = (l, \ldots, r)$ induce a partition in the corresponding intervals of genomes A and B into sub-intervals that we will subsequently call *blocks*.
If we label the blocks in the interval of $\mathcal{C}$ in genome B with numbers from 1 to k, the corresponding blocks of the interval of $\mathcal{C}$ in genome A will be a signed permutation $(p_1, \ldots, p_k)$ of the *elements* $\{1, \ldots, k\}$. We will call this permutation – or its reverse – the *permutation associated to the component* $\mathcal{C}$.
Consider for example the following two genomes:

$$A = \{(\circ\ 5, 1, 3, -2, 4, 6, -10, 9, 8, -7, 11\ \circ)\},$$
$$B = \{(\circ\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9, 10, 11\ \circ)\}.$$

The components of A with respect to B can easily be seen in the following diagram:



The component $(\circ, \ldots, 6)$ consists of three blocks: the gene -5, the component $(1, \ldots, 4)$ and the gene 6. Thus, the permutation associated to the component $(\circ, \ldots, 6)$ is $(-2, 1, 3)$. For the other three non-trivial components, the associated permutations are $(1, 3, -2, 4)$, $(-4, 3, 2, -1)$ and $(1, -2, 3)$.
As shown in the proof of Corollary 4.1, a DCJ-sorting operation increases either the number of cycles or the number of odd paths. When the elements of the permutation associated to a component have both positive and negative signs, then there exists a pair of consecutive elements with opposite signs. This implies that there exists a DCJ-sorting inversion that increases the number of cycles by creating an adjacency of the form $(i, i+1)$ or $(-(i+1), -i)$. For instance, the permutation $(-2, 1, 3)$ associated to component $(\circ, \ldots, 6)$ in the above example admits the DCJ-sorting inversion of element 1, creating the adjacency of elements $(-2, -1)$. This corresponds to the adjacency of genes $(-5, -4)$ in the component $(\circ, \ldots, 6)$, yielding a new genome

$$A' = \{(\circ\ -5, -4, 2, -3, -1, 6, -10, 9, 8, -7, 11\ \circ)\}.$$

Components whose associated permutations have only elements with the same sign are more intricate. We will see later that some of them can be optimally sorted by DCJ-sorting operations, others not. For example, consider the pair of genomes:

$$A = \{(\circ, 4, 3, 2, 1, \circ)\} \quad \text{and} \quad B = \{(\circ, 1, 2, 3, 4, \circ)\},$$

whose associated permutation is $(4, 3, 2, 1)$. There exists a DCJ-sorting operation that is a path fission increasing the number of odd paths. The DCJ distance is 4, and it can be optimally sorted by inverting each of the four genes. However, we have:

Lemma 4.10 *If all elements of the permutation associated to a component have the same sign, then no inversion acting on one of its paths or cycles can create a new cycle.*

Proof By possibly flipping the chromosome, we can assume that all the elements of the permutation are positive. Suppose that an inversion is applied to two adjacencies $(+i, +j)$ and $(+k, +l)$ in a single path or cycle of the component, and that this creates a new cycle. The new adjacencies will be $(+i, -k)$ and $(-j, +l)$, where at most one of $+i$ and $+l$ can be a telomere. If both of these new adjacencies belong to the same path or cycle, there was no creation of a new cycle. Suppose that the adjacency $(+i, -k)$ belongs to the new cycle. Then, all other adjacencies of this cycle existed in the original component, and are composed of positive elements. This, however, is impossible by the construction of the adjacency graph. ■

Definition 4.11 *A component is* oriented *if there exists an oriented DCJ-sorting operation that acts on the vertices of its adjacency graph, otherwise it is* unoriented.

Oriented components are characterized by the following:

Proposition 4.12 *A component is oriented if and only if either its associated permutation has positive and negative elements, or its adjacency graph has two even paths.*

Proof If the associated permutation has positive and negative elements, then there is at least one change of signs between blocks labeled by consecutive integers. Thus, there exists an inversion that creates an adjacency in genome B, thus a new cycle, and the inversion is DCJ-sorting. If there are two even paths, then one of them must be a path from genome A to genome A, and the other one must be a path from genome B to genome B. An inversion in genome A that acts on one adjacency in each path creates two odd paths, thus is DCJ-sorting.
In order to show the converse, suppose that all elements of the associated permutation are positive, and all paths are odd. By Corollary 4.1, a DCJ-sorting operation must act on a single path or cycle. This operation cannot be a translocation or a fusion since all paths and cycles of a component are within the same chromosome. This operation cannot be an inversion, since inversions that create new cycles are ruled out by Lemma 4.10, inversions acting on a single odd path cannot augment the number of odd paths, and inversions acting on cycles never create paths. Finally, this operation cannot be a fission: a fission acting on a cycle creates an even path; and a fission acting on an odd path must circularize one of the chromosome parts in order to be DCJ-sorting, otherwise it would be split into an even path and an odd path. ■

As a consequence of Proposition 4.12, we have $d_{DCJ}(A, B) < d_{HP}(A, B)$ in the presence of unoriented components, since all DCJ-sorting operations will create circular chromosomes. On the other hand, well-known results from the Hannenhalli-Pevzner theory show that, when each component admits a sorting inversion, then it is possible to create a new cycle at each step of the sorting process with HP operations, without creating unoriented components [63]. We will see in the next propositions that the same type of result can be obtained in this context. According to Proposition 4.12, it will be useful to treat components with paths and components without paths separately.

Definition 4.13 *Components whose both extremities are genes are called* real *components. Components that contain one or two telomeres are* semi-real *components.*

First, let us consider oriented real components that are well studied in the context of sorting by inversions. In [63], it was first shown that oriented real components can be sorted optimally by oriented inversions. This relies on the fact that among all possible oriented inversions there always exists one that does not create new unoriented components. Such an inversion is called a *safe* inversion and can be found by trial and error. More sophisticated techniques to efficiently find safe inversions can be found in [14, 25, 71].

Proposition 4.14 ([63]) *An oriented real component has an oriented DCJ-sorting operation that does not create new unoriented components.*

As a consequence of Proposition 4.14, it is possible to sort real components with oriented DCJ-sorting inversions as shown in the next proposition.

Proposition 4.15 *A real component can be sorted with oriented DCJ-sorting operations if and only if it is oriented.*

Proof If a real component can be sorted by oriented DCJ-sorting operations, then, by definition, the component is oriented. If a real component is oriented, then there exists an oriented DCJ-sorting inversion and Proposition 4.14 guarantees that there will always be enough oriented DCJ-sorting inversions to sort the component. ■

Now, we come to the semi-real components. First, we consider semi-real components whose associated permutation has positive and negative elements. As it turns out, these components can be treated in the same way as oriented real components by adding extra genes.

Proposition 4.16 *An oriented semi-real component whose associated permutation has positive and negative elements can be sorted with oriented DCJ-sorting operations.*

Proof We will show that an oriented semi-real component whose associated permutation has positive and negative elements can be embedded in an oriented real component with the same DCJ distance. Then, by Proposition 4.15, we can sort the component with oriented DCJ-sorting operations.
Consider a component $\mathcal{C} = (l, \ldots, r)$ and its associated permutation $(p_1, \ldots, p_k)$. Assume that the adjacency graph of the component $\mathcal{C}$ has C cycles and I odd paths, then the number of DCJ-sorting operations equals

$$k - (C + \frac{I}{2}).$$

We will show that this distance is also achieved after having added extra genes to the component. Depending on the component, we distinguish the following cases:

1. If the component $\mathcal{C}$ has one telomere, then either $l = \circ$ or $r = \circ$. Suppose that $l = \circ$, then we add an extra gene 0 at the beginning of the associated permutation. This yields a real component $\mathcal{C}' = (l', \ldots, r) = (0, \ldots, k)$. In the adjacency graph, the – unique – odd path is transformed into a cycle and a new path of length one is created. Thus, the number of DCJ-sorting operations equals

   $$k + 1 - ((C + 1) + \frac{I}{2}) = k - (C + \frac{I}{2}).$$

   The case where $r = \circ$ can be treated similarly.

2. If the component has two telomeres, then $l = \circ$ and $r = \circ$. The adjacency graph has two paths, either both even or both odd.

   (a) Suppose that both paths are odd. First, if the telomere $(\circ, p_1)$ of A and the telomere $(\circ, 1)$ of B belong to the same odd path, then we add the extra gene 0 at the beginning and the extra gene $k + 1$ at the end of the associated permutation. This results in a real component $\mathcal{C}' = (l', \ldots, r') = (0, \ldots, k+1)$. This transforms the two odd paths into two cycles and creates two new paths of length one. Therefore, we have

   $$k + 2 - ((C + 2) + \frac{I}{2}) = k - (C + \frac{I}{2}).$$

Similarly, if the telomere $(\circ, p_1)$ of A and the telomere $(p_k, \circ)$ of B are the ends of an odd path, then we add the gene $-(k+1)$ at the beginning and -0 at the end of the associated permutation. Again, the DCJ distance is preserved.

(b) Suppose that both paths are even. By adding genes 0 and $k+1$ at the beginning and the end of the permutation, the component $\mathcal{C}$ is embedded in a real component $\mathcal{C}' = (0, \ldots, k+1)$. By doing this, the two even paths are joined into one cycle and two paths of length one are added. Therefore, the DCJ distance remains the same:

$$k + 2 - ((C+1) + \frac{I+2}{2}) = k - (C + \frac{I}{2}).$$

■

Oriented components must sometimes be sorted with fissions. This is the case for semi-real components with paths of even length. Such components have two paths, one of them connecting two telomeres of A and one connecting two telomeres of B. The latter path can be split by a fission into two odd paths without creating new unoriented components as stated in the next proposition.

Proposition 4.17 *A semi-real component whose adjacency graph has even paths can be sorted with oriented DCJ-sorting operations.*

Proof First, note that a semi-real component $\mathcal{C} = (l, \ldots, r)$ with one even path must have two even paths, i. e., $l = \circ$ and $r = \circ$.
Consider the permutation $(p_1, \ldots, p_k)$ associated to component $\mathcal{C}$. If the permutation is oriented, then it is possible to sort the component with oriented DCJ-sorting operations by Proposition 4.16.
Now, if the permutation is unoriented, then all genes p_1 to p_k have the same sign. Without loss of generality, we assume that they are all positive and that $p_i = k$ precedes $p_j = 1$. Then, one chromosome of genome A is given by

$$(\circ, p_1, \ldots, p_{i-1},\ k,\ p_{i+1}, \ldots, p_{j-1},\ 1,\ p_{j+1}, \ldots, p_k, \circ).$$

The path connecting the two telomeres of genome B implies two possible fissions: fission F_1 creating telomere $(k, \circ)$ and fission F_2 creating $(\circ, 1)$.
Suppose that both fissions create new unoriented components. Let $(l_1, \ldots, r_1)$ and $(l_2, \ldots, r_2)$ be unoriented components created by fissions F_1, respectively F_2. Since fission F_1 yields the chromosomes

$$(\circ, p_1, \ldots, p_{i-1},\ k, \circ) \text{ and } (\circ,\ p_{i+1}, \ldots, p_{j-1},\ 1,\ p_{j+1}, \ldots, p_k, \circ),$$

the newly created component $(l_1, \ldots, r_1)$ must be in the beginning of the second chromosome, otherwise it would have existed before. Moreover, gene 1 belongs to the component $(l_1, \ldots, r_1)$ because the interval is in the beginning of a chromosome of B. On the other hand, fission F_2 creates chromosomes:

$$(\circ, p_1, \ldots, p_{i-1},\ k,\ p_{i+1}, \ldots, p_{j-1}, \circ) \text{ and } (\circ, 1,\ p_{j+1}, \ldots, p_k, \circ).$$

By a similar argumentation, the component $(l_2, \ldots, r_2)$ is at the end of the first chromosome and contains the gene k. This means that the genes in the set $\{p_{i+1}, \ldots, p_{j-1}\}$ are in the beginning and at the end of a chromosome of B. But, since each gene occurs exactly once, the set $\{p_{i+1}, \ldots, p_{j-1}\}$ must be empty, implying that $i + 1 = j$. Thus, both fissions create the following two chromosomes

$$(\circ, p_1, \ldots, p_{i-1},\ k, \circ) \text{ and } (\circ, 1,\ p_{j+1}, \ldots, p_k, \circ).$$

But, a component $(l_1, \ldots, r_1)$ in the end of the first chromosome would be a real component. This contradicts the fact that the component was newly created by the fission. Also, contradicting our assumption, $(l_2, \ldots, r_2)$ cannot be created by a fission. Thus, we have shown that at least one of the two fissions does not create any new unoriented components. ■

Until now, we have dealt only with components, i.e. included in chromosomes. Now, we will turn to the general problem of sorting linear genomes. This requires also DCJ operations that act on different chromosomes. More formally, we have:

Definition 4.18 *A DCJ operation creating the adjacency (a, b) of B, where a and b are genes, is called* interchromosomal*, if (a, x) and (y, b) belong to different chromosomes in A.*

1. *If $x \neq \circ$ and $y \neq \circ$, the DCJ operation is a translocation.*
2. *If $x = \circ$ or $y = \circ$ (but not both), the DCJ operation is a semi-translocation.*
3. *If $x = \circ$ and $y = \circ$, the DCJ operation is a fusion.*

The next lemma is the key, it says that for any interchromosomal DCJ operation that creates an unoriented component there always exists an alternative interchromosomal DCJ-sorting operation that does not. This statement, already proven in the context of sorting by internal translocations [21], can be shown similarly for the general case.

Proposition 4.19 *Given two linear genomes A and B, if an interchromosomal DCJ operation creates an unoriented component, then there exists another interchromosomal DCJ-sorting operation that does not.*

Proof First note that an interchromosomal DCJ-sorting operation can be a translocation, a semi-translocation or a fusion. Without loss of generality, we assume that the DCJ operation is a translocation.
Suppose that a translocation T creates new unoriented components by acting on adjacencies (a, x) and (y, b), and let

$$\mathcal{C} = (l, \ldots, a, b, \ldots, r)$$

be the smallest such component in the resulting genome A' that contains a and b. We will show that then there must exist another DCJ-sorting translocation that either creates smaller components, or does not create non-trivial components.
We distinguish the two cases whether the component $\mathcal{C}$ is real or semi-real:

First, if the newly created component is real, then $l \neq \circ$ and $r \neq \circ$ and genome A contains the two chromosomes

$$(\circ, \ldots, l, \ldots, a, \ldots, \circ) \text{ and } (\circ, \ldots, b, \ldots, r, \ldots, \circ).$$

If the component $\mathcal{C}$ is a real component, then there exists an element M in $(l, \ldots, a)$ such that the element $M + 1$ is in $(b, \ldots, r)$. This follows from the definition of a component and the fact that either $l \neq a$ or $b \neq r$. Therefore, the two chromosomes of genome A can be written as

$$(\circ, \ldots, l, \ldots, M, \ldots, a, \ldots, \circ) \text{ and } (\circ, \ldots, b, \ldots, M+1, \ldots, r, \ldots, \circ).$$

Thus, there exists an alternative DCJ-sorting translocation creating the adjacency $(M, M + 1)$. Applying this translocation results in a genome A'' with chromosomes

$$(\circ, \ldots, l, \ldots, M, M+1, \ldots, r, \ldots, \circ) \text{ and } (\circ, \ldots, b, \ldots, a, \ldots, \circ).$$

If a or b belong to a new non-trivial component in A'', then this component must be strictly smaller than $(l, \ldots, r)$, since both a and b are contained in $\mathcal{C} = (l, \ldots, r)$, a contradiction.
Moreover, a new non-trivial component cannot contain both l and r, since the element $a \in \{l, \ldots, r\}$ is on a different chromosome. If it contains l and is longer than $(l, \ldots, r)$, then it must be an interval of the form $(l', \ldots, l, \ldots, r')$, where $l' < l < r' < r$. But all the elements at the right of l are greater than l, and all the elements at the left of l are smaller than l, implying that either $l' = l$ or $l = r'$, which are both contradictions. Similar arguments hold if the new non-trivial component contains r and is longer than $(l, \ldots, r)$.

Secondly, if the component $\mathcal{C} = (l, \ldots, a, b, \ldots, r)$ is semi-real, then either $l = \circ$, or $r = \circ$, or both. Suppose that $r = \circ$, then genome A contains the chromosomes

$$(\circ, \ldots, l, \ldots, a, \ldots, \circ) \text{ and } (\circ, \ldots, b, \ldots, \circ).$$

Suppose that the component $\mathcal{C}$ is a semi-real component with gene content $\{m, \ldots, M\}$, and that $l = m$ and $r = \circ$. Moreover, M is either in the interval $(l, \ldots, a)$ or in the interval $(b, \ldots, r)$. Assume that the first case holds, then the genome A has the chromosomes

$$(\circ, \ldots, l, \ldots, M, \ldots, a, \ldots, \circ) \text{ and } (\circ, \ldots, b, \ldots, \circ).$$

The translocation creating the telomere $(M, \circ)$ yields the genome A'':

$$(\circ, \ldots, l, \ldots, M, \circ) \text{ and } (\circ, \ldots, b, \ldots, a, \ldots, \circ).$$

Again, if a or b belong to a new non-trivial component in A'', then this component must be strictly smaller than $(l, \ldots, r)$, since both a and b are contained in $\mathcal{C} = (l, \ldots, r)$, a contradiction.
Moreover, a new non-trivial component cannot contain both l and M, since the element $a \in \{m, \ldots, M\}$ is on a different chromosome. If it contains l and is longer than $(l \ldots r)$, then it must be an interval of the form $(l', \ldots, l, \ldots, r')$, where $l' < l < r' < M$. But all the elements at the right of l are greater than l, and all the elements at the left of l are smaller than l, implying that either $l' = l$ or $l = M$, which are both contradictions. Similar arguments hold if the new non-trivial component contains M and is longer than $(l, \ldots, r)$. ■

Finally, we have all necessary ingredients for our main result of this section.

Theorem 4.20 *Given two linear genomes A and B, $d_{HP}(A, B) = d_{DCJ}(A, B)$ if and only if there are no unoriented components.*

Proof The "if" part comes from the fact that we can sort a genome without unoriented components with DCJ-sorting operations (Propositions 4.16, 4.17, 4.19), adding the fact that semi-real components whose graphs have even paths can be "destroyed" by fissions. The "only if" part comes from the fact that if there are unoriented components, then $d_{DCJ}(A, B) < d_{HP}(A, B)$, since we showed in Proposition 4.12 that in this case all DCJ-sorting operations create circular chromosomes. ■

4.3 Computing the General HP Distance

In this section we will show that, given the DCJ distance d_{DCJ}, one can express the Hannenhalli-Pevzner distance d_{HP} in the form

$$d_{HP} = d_{DCJ} + t,$$

where t represents the additional cost of not resorting to unoriented DCJ operations. First, we describe how to destroy unoriented components in Section 4.3.1 and after that, in Section 4.3.2, we compute the additional cost from the inclusion and linking tree of the unoriented components.

4.3.1 Destroying Unoriented Components

Destroying unoriented components is done by applying a DCJ operation either on one component in order to orient it, or on two components in order to destroy or to merge them, and possibly others, into a single oriented component. By using the nesting and linking relationship between components, one can minimize the number of operations necessary to destroy unoriented components.
When two components overlap on one element, we say that they are *linked*. Successive linked components form a *chain*. A chain that cannot be extended to the left or right is called *maximal*. We represent the nesting and linking relations between components of a chromosome in the following way:

Definition 4.21 *Given a chromosome X of genome A and its components relative to genome B, define the forest F_X by the following construction:*

1. *Each non-trivial component is represented by a round node.*
2. *Each maximal chain that contains non-trivial components is represented by a square node whose (ordered) children are the round nodes that represent the non-trivial components of this chain.*
3. *A square node is the child of the smallest component that contains this chain.*

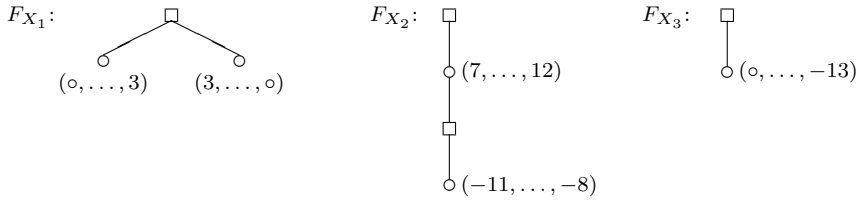


Figure 4.1: The forests associated to the genomes A and B of Example 4.5 where F_{X_1}, F_{X_2} and F_{X_3} are the forests of the first, the second and the third chromosome of genome A, respectively.

For example, the forests of the genomes A and B of Example 4.5 are shown in Fig. 4.1. Let us consider a single tree of a forest. Clearly, there is a unique path between any two components in this tree. Moreover, we have the following result:

Proposition 4.22 ([19]) *Let $\mathcal{C}$ be a component on the unique path joining components $\mathcal{A}$ and $\mathcal{B}$ in a tree of F_X, i. e. $\mathcal{C}$ contains either $\mathcal{A}$ or $\mathcal{B}$, or both.*

1. *If $\mathcal{C}$ contains both $\mathcal{A}$ and $\mathcal{B}$, then it is the only component on the path that contains $\mathcal{A}$ and $\mathcal{B}$.*

2. *No component of the path contains both $\mathcal{A}$ and $\mathcal{B}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ are included in two components that are in the same chain.*

Proof Consider the smallest component $\mathcal{D}$ that contains components $\mathcal{A}$ and $\mathcal{B}$. Suppose that $\mathcal{D}$ is on the path that joins $\mathcal{A}$ and $\mathcal{B}$, like in the left tree of Fig. 4.2. Then, any other component that contains $\mathcal{A}$ and $\mathcal{B}$ is an ancestor of $\mathcal{D}$, therefore not on the path. Now, suppose that $\mathcal{D}$ is not on the path that joins $\mathcal{A}$ and $\mathcal{B}$, as shown in the right tree of Fig. 4.2. The least common ancestor of components $\mathcal{A}$ and $\mathcal{B}$ is a square node q that is a child of the round node representing $\mathcal{D}$, thus $\mathcal{A}$ and $\mathcal{B}$ are included in two components that are in the chain represented by q. ■

Figure 4.2: Component $\mathcal{D}$ is the smallest component that contains components $\mathcal{A}$ and $\mathcal{B}$. **Left:** $\mathcal{D}$ is on the path between $\mathcal{A}$ and $\mathcal{B}$. **Right:** $\mathcal{D}$ is not on the path between $\mathcal{A}$ and $\mathcal{B}$.

Next, we define a tree associated to the components of a genome by combining the forests of all chromosomes into one rooted tree:

Definition 4.23 *Suppose genome A consists of chromosomes X_1, ... , X_K. The tree T associated to the components of genome A relative to genome B is given by the following construction:*

1. *The root of T is a round node.*

2. *All trees of the set of forests $\{F_{X_1}, F_{X_2}, \ldots, F_{X_K}\}$ are children of the root.*

The round nodes of T are *painted* according the following classification:

1. The root and all nodes corresponding to oriented components are painted *black*.
2. Nodes corresponding to unoriented, real components are painted *white*.
3. Nodes corresponding to unoriented, semi-real components are painted *gray*.

The tree associated to the components of the genomes A and B of Example 4.5 is shown in Fig. 4.3. Note that gray nodes are never included in other components.

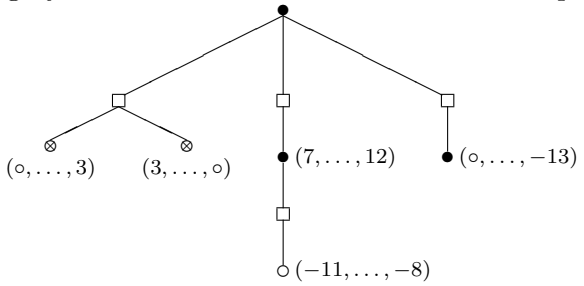


Figure 4.3: The tree T associated to the genomes A and B of Example 4.5 has two gray leaves, one white leaf and one black leaf.

The following two propositions are general remarks on components and are useful to show how to destroy unoriented components.

Proposition 4.24 *A translocation acting on two (unoriented) components cannot create new (unoriented) components.*

Proof Consider two components $\mathcal{A}$ and $\mathcal{B}$ and let $(p_1, \ldots, p_k)$ and $(q_1, \ldots, q_l)$ be their associated permutations. For contradiction, assume that a translocation acting on $\mathcal{A}$ and $\mathcal{B}$ creates a new component $\mathcal{D}$. Without loss of generality, we assume that the translocation transforms the adjacencies (p_i, p_{i+1}) and (q_j, q_{j+1}) into (p_i, q_{j+1}) and (q_j, p_{i+1}). The newly created component $\mathcal{D}$ must contain at least one of the two adjacencies, either (p_i, q_{j+1}) or (q_j, p_{i+1}), otherwise $\mathcal{D}$ would have existed before. Since a translocation acts on one cycle or one path in each component, we consider several possible cases:

1. If the translocation merges two cycles into one cycle, then both adjacencies (p_i, q_{j+1}) and (q_j, p_{i+1}) belong to the new cycle. This contradicts the fact that all adjacencies of a component are in the same chromosome.
2. If the translocation merges one cycle and one path into one path, then both adjacencies (p_i, q_{j+1}) and (q_j, p_{i+1}) belong to the new path. Again, this is a contradiction to the definition of a component.
3. If the translocation acts on two paths, then the adjacencies (p_i, q_{j+1}) and (q_j, p_{i+1}) are in two different paths. Because both paths have adjacencies that belong to two different chromosomes, this is again a contradiction to the definition of a component. ■

Proposition 4.25 *An inversion acting on two (unoriented) components $\mathcal{A}$ and $\mathcal{B}$ creates a new component $\mathcal{D}$ if and only if $\mathcal{A}$ and $\mathcal{B}$ are included in linked components.*

Proof Assume that an inversion acting on $\mathcal{A}$ and $\mathcal{B}$ creates a new component $\mathcal{D} = (l, \ldots, r)$, then the extremities of the interval $\mathcal{D}$ must be both outside the inverted interval. Indeed, if both extremities of $\mathcal{D}$ are inside the inverted interval, D existed in the original permutation. If one extremity of $\mathcal{D}$ is outside the interval, then component $\mathcal{D}$ must contain at least one endpoint of the inverted interval in order to be affected by the inversion. To see that the other endpoint of the inverted interval is also in $\mathcal{D}$, we distinguish between the three cases that the inversion is a cycle fusion, an integration, or a path translocation in the adjacency graph. Certainly, this is evident in the first two cases because the two endpoints of the inverted interval belong to the same cycle or the same path. If the inversion is a path translocation, then the resulting two paths span the whole chromosome and hence belong to the same component $\mathcal{D}$. Thus, in all cases, the second extremity of $\mathcal{D}$ is also outside the interval.
One can easily check that there does not exist a component $\mathcal{C}$ on the path from $\mathcal{A}$ to $\mathcal{B}$ that contains $\mathcal{A}$ and $\mathcal{B}$. In particular, if there would be such a component $\mathcal{C}$, then, after the inversion, the new component $\mathcal{D}$ would be included in $\mathcal{C}$. Thus, by Proposition 4.22, the components $\mathcal{A}$ and $\mathcal{B}$ are included in linked components.
To show the converse, suppose that components $\mathcal{A}$ and $\mathcal{B}$ are nested in components $\mathcal{A}' = (l_1, \ldots, r_1)$ and $\mathcal{B}' = (l_2, \ldots, r_2)$ such that $\mathcal{A}'$ and $\mathcal{B}'$ are linked in a chain. Then, an inversion acting on one adjacency of $\mathcal{A}$ and one adjacency of $\mathcal{B}$ creates the new component $\mathcal{D} = (l_1, \ldots, r_2)$. ■

Now, we have all necessary results to get rid of unoriented components. The following two propositions are straightforward generalizations of well-known results from the sorting by inversion theory. We will start by looking at one single unoriented component.

Proposition 4.26 *If a component $\mathcal{C}$ is unoriented, any inversion between adjacencies of the same cycle or the same path of $\mathcal{C}$ orients $\mathcal{C}$, and leaves the number of cycles and paths of the adjacency graph of $\mathcal{C}$ unchanged.*

Proof Inverting an interval changes the sign of the elements of the inverted interval. Therefore, component $\mathcal{C}$ will be oriented. Since the adjacencies belong to the same cycle or path, the inversion cannot merge cycles or paths. By Proposition 4.10, the inversion cannot split a cycle, nor a path. Therefore, the number of cycles and the number of paths remain unchanged. ■

Orienting a component as in Proposition 4.26 is called *cutting* the component. Note that this operation leaves the DCJ distance unchanged, and does not create new components. It is possible to destroy more than one unoriented component with a single DCJ operation acting on two unoriented components. The following proposition describes how to *merge* several components, and the relations of this operation to paths in the tree T.

Proposition 4.27 *There always exists an HP operation acting on adjacencies of two different components $\mathcal{A}$ and $\mathcal{B}$ destroys, or orients, all components on the path from $\mathcal{A}$ to $\mathcal{B}$ in the tree T, without creating new unoriented components or circular chromosomes.*

Proof If $\mathcal{A}$ and $\mathcal{B}$ are in different chromosomes, then we apply a translocation to destroy $\mathcal{A}$ and $\mathcal{B}$. As shown in Proposition 4.24, such a translocation cannot create new unoriented components. Moreover, all components that contain either $\mathcal{A}$ or $\mathcal{B}$ are also destroyed by the translocation. Thus, all components on the path are destroyed.
If $\mathcal{A}$ and $\mathcal{B}$ are in the same chromosome, then we apply an inversion to destroy $\mathcal{A}$ and $\mathcal{B}$. By Proposition 4.25, the only component eventually created by an inversion is the union of two or more linked components. Since linked components have extremities with the same sign, the sign of the links will be different from the sign of the extremities of the new component, thus it will be oriented.
Now, if the components $\mathcal{A}$ and $\mathcal{B}$ are not included in linked components, then the lowest common ancestor must be a round node, either the root or, by Proposition 4.22, a component $\mathcal{C}$ that is on the path from $\mathcal{A}$ to $\mathcal{B}$ and that contains both. In the latter case, components $\mathcal{A}$ and $\mathcal{B}$ must be real, thus the inversion merges two cycles, one from each component, into one new cycle. If $\mathcal{A}$ and $\mathcal{B}$ are unoriented, the new cycle contains at least one oriented adjacency. Since $\mathcal{C}$ is the smallest component that contains the new cycle, $\mathcal{C}$ will be oriented.
Finally, for any component $\mathcal{C}$ on the path from $\mathcal{A}$ to $\mathcal{B}$ and that contains either $\mathcal{A}$ or $\mathcal{B}$, but not both, the inversion changes the sign of one of the bounding elements of $\mathcal{C}$, and $\mathcal{C}$ will be destroyed. ■

Remark 4.28 *If the HP operation acts on two odd paths, thus on gray components, then merging the two components can be done without changing the number of odd paths, and the DCJ distance is unchanged. If the HP operation involves at least one cycle, then merging two components decreases the number of cycles by one, and the DCJ distance will increase by 1 in the resulting pair of genomes.*

4.3.2 Unoriented Sorting

Let T be the tree associated to the components of genome A relative to genome B, and let T' be the smallest subtree of T that contains all the unoriented components, that is, the white and gray nodes.

Definition 4.29 *A* cover *of T' is a collection of paths joining all the unoriented components, such that each terminal node of a path belongs to a unique path.*

A path that contains two or more white or gray components, or one white and one gray component, is called a *long* path. A path that contains only one white or one gray component, is a *short* path.
The *cost* of a cover is defined to be the sum of the costs of its paths, where the cost of path is the increase in distance caused by destroying the unoriented components along the path. More precisely, the cost of a path represent the sum of one extra HP operation and the variation of the DCJ distance. Using Remark 4.28, we have:

1. The cost of a short path is 1.
2. The cost of a long path with only two gray components is 1.
3. The cost of all other long paths is 2.

An *optimal* cover is a cover of minimal cost. Define t as the cost of any optimal cover of T'. We first establish that t is the difference between the DCJ distance and the HP distance, using the following terminology:

Definition 4.30 *Given genomes A and B, we call a DCJ operation applied to genome A*

- proper*, if it decreases $d_{DCJ}(A, B)$ by one, i. e., $\Delta(C + I/2) = 1$,*
- improper*, if $d_{DCJ}(A, B)$ remains unchanged, i. e., $\Delta(C + I/2) = 0$, and*
- bad*, if it increases $d_{DCJ}(A, B)$ by one, i. e., $\Delta(C + I/2) = -1$.*

Theorem 4.31 *If t is the cost of an optimal cover of T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B, then:*

$$d_{HP}(A, B) = d_{DCJ}(A, B) + t.$$

Proof First, we will show that $d_{HP}(A, B) \leq d_{DCJ}(A, B) + t$. Consider any cover of the tree T'. Let

- ww be the number of long paths with only white components,
- wg be the number of long paths with white and gray components,
- gg be the number of long paths with only gray components,
- w be the number of short paths with one white component,
- g be the number of short paths with one gray component.

Clearly, we have that the cost t' of this cover is $t' = 2ww + 2wg + gg + w + g$.
Suppose that the adjacency graph $AG(A, B)$ has C cycles and I odd paths. Applying $ww + wg$ bad DCJ operations and $gg + w + g$ improper DCJ operations yields a genome A'. Since each bad DCJ operation merges two cycles or one cycle and a path, the number of cycles in $AG(A', B)$ is $C - ww - wg$. Note that the number of odd paths remains unchanged. Therefore, by Theorem 4.20, we have that

$$\begin{aligned} d_{HP}(A, B) &\leq d_{HP}(A', B) + ww + wg + gg + w + g \\ &= N - ((C - ww - wg) + \frac{I}{2}) + ww + wg + gg + w + g \\ &= N - (C + \frac{I}{2}) + 2ww + 2wg + gg + w + g \\ &= d_{DCJ}(A, B) + t' \end{aligned}$$

Since the above equation is true for any cover, we have: $d_{HP}(A, B) \leq d_{DCJ}(A, B) + t$.

Now, we will show that $d_{HP}(A, B) \geq d_{DCJ}(A, B) + t$. Consider an optimal sequence of HP operations of length d. Observe that we can write

$$d = p + i + b$$

where p is the number of proper DCJ operations, i the number of improper DCJ operations and b denotes the number of bad DCJ operations. Since p operations increase $C + \frac{I}{2}$ by one and b operations decrease $C + \frac{I}{2}$ by one, and by Lemma 3.8, we have:

$$C + \frac{I}{2} + p - b = N, \text{ implying } d = N - (C + \frac{I}{2}) + 2b + i.$$

We will show that $2b + i \geq t$, implying $d_{HP}(A, B) \geq d_{DCJ}(A, B) + t$.
The sequence of d HP operations induces a cover of T. Indeed, any HP operation that merges a group of components traces a path in T, of which we keep the shortest segment that includes all unoriented components of the group. Of these paths, suppose that ww, wg and gg are the numbers of paths joining two white, one white and one gray, and two grey terminal nodes, respectively. Clearly we have $ww + wg \leq b$, since merging two white, or one white and one gray component is a bad HP operation. After applying the sequence of d operatiosn, all unoriented components are eventually destroyed. Therefore, the remaining unoriented components are all cut. Suppose that w and g are the numbers of remaining white and gray nodes, respectively. Because merging two gray components, as well as cutting unoriented components is done by improper HP operations, we have that $gg + w + g \leq i$. Altogether, we have

$$t \leq 2ww + 2wg + gg + w + g \leq 2b + i.$$

Thus, we get $d \geq n - c + t$, completing the proof. ■

It remains to establish a closed formula for t. A first easy but significant result on the size of t is the following lower bound. Let w be the number of white leaves and g be the number of gray leaves in T'. Since destroying a white leaf costs at least 1, and destroying a gray leaf costs at least 1/2, and t is an integer, we have:

$$w + \left\lceil \frac{g}{2} \right\rceil \leq t.$$

It is quite remarkable, as was observed in the original paper on HP distance [62], that this bound is at most within one rearrangement operation from the optimal solution.
To show the upper bound, we need the following observation that at most two gray leaves are paired with white leaves:

Lemma 4.32 *Consider an optimal cover of T' and the maximal number of paths covering one white and one gray leaf in such a cover. There exists an optimal cover such that this number is at most two.*

Proof Assuming the contrary, the number of paths covering one black and one gray leaf is greater than two. Consider three such pairs, then $t = x + 6$, where x is the cost of the remaining paths.
Now, we consider an alternative cover. We pair two black leaves such that the lowest common ancestor of the three black leaves is on the path. Finally, it remains one black and one gray leaf that are paired. This cover has cost $x + 5$, contradicting the minimality of the cost t. ■

A branch in a tree is called a *long branch* if it has two or more unoriented components. A tree is called a *fortress* if it has an odd number of leaves, all of them on long branches.

A standard theorem of the sorting by inversion theory states that the minimal cost to cover a tree that is not a fortress is ℓ, the number of leaves of the tree, and $\ell+1$ in the case of a fortress [18, 63].

Proposition 4.33 *Let w be the number of white leaves of T' and g the number of gray leaves of T'. Then we have*

$$t \leq w + \left\lceil \frac{g}{2} \right\rceil + 1$$

Proof We show the upper bound by a case-by-case analysis. For any values of w and g, we give a cover with cost t^* such that

$$t^* \leq w + \left\lceil \frac{g}{2} \right\rceil + 1.$$

Since $t \leq t^*$, the upper bound holds. Consider the following cases:
First, if $g = 0$, then the cost of an optimal cover of the white leaves is given by:

$$t^* = \begin{cases} w+1 & \text{if } T' \text{ is a fortress,} \\ w & \text{otherwise.} \end{cases}$$

If $g = 1$, then the cost for covering the white leaves is similar to the previous case. In addition, we have one short path for the gray leaf. In total, we have:

$$t^* = \begin{cases} w+2 & = w + \left\lceil \frac{1}{2} \right\rceil + 1 & \text{, if } T' \text{ is a fortress,} \\ w+1 & \leq w + \left\lceil \frac{1}{2} \right\rceil + 1 & \text{, otherwise.} \end{cases}$$

If $g = 2$ and w is even, then the only two gray leaves are either covered by one long path at cost 1, if $w = 0$, or they are paired with two white leaves, if $w \geq 2$. In the latter case, the remaining $w-2$ white leaves are paired at cost $w-2$. In both cases, we have $t^* \leq w + \left\lceil \frac{2}{2} \right\rceil + 1$.
If $g = 2$ and w is odd, then we choose one white leaf that is on a long branch, if possible. This white leaf is paired with one gray leaf at cost 2. In addition, the remaining $w-1$ white leaves can be covered at cost $w-1$, and the second gray leaf is destroyed at cost 1. Altogether, we have $t^* = 2 + (w-1) + 1 = w + \left\lceil \frac{2}{2} \right\rceil + 1$.
If $g \geq 3$ odd, then we pair $g-1$ gray leaves at cost $\frac{g-1}{2}$. As shown previously, the remaining gray leaf and the w white leaves can be destroyed with cost at most $w+2$. By summing up, we have $t^* \leq w + 2 + \frac{g-1}{2} = w + 2 + \left\lceil \frac{g-1}{2} \right\rceil = w + \left\lceil \frac{g}{2} \right\rceil + 1$.
Finally, if $g \geq 4$ even, then we pair $g-2$ gray leaves at cost $\frac{g-2}{2}$. By a previous case, the remaining two grey leaves and the w white leaves can be destroyed with cost at $w+2$. Thus, we have $t^* = w + 2 + \frac{g-2}{2} = w + \frac{g}{2} + 1 = w + \left\lceil \frac{g}{2} \right\rceil + 1$. ■

The closed formula for t is now given in the following two theorems.

Theorem 4.34 *Let w be the number of white leaves and g be the number of gray leaves in T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B. If the root of T' has more than one child with white leaves, then the minimal cost of a cover of T' is:*

$t = w + \left\lceil \frac{g}{2} \right\rceil$ *if the smallest subtree T'' that contains all the white leaves of T' is not a fortress, or g is odd,*
$t = w + \left\lceil \frac{g}{2} \right\rceil + 1$ *otherwise.*

Proof If the subtree T'' is not a fortress then it admits a cover of cost w, and pairing the maximum number of gray nodes yields a cover of T' costing $w + \lceil \frac{g}{2} \rceil$. If the subtree T'' is a fortress, then one of its white leaves is not paired with another leaf since the number of leaves is odd. A cover of T' can be obtained by pairing this white leaf with a gray leaf, which exists if g is odd. The resulting cost will be again $w + \lceil \frac{g}{2} \rceil$ which equals the lower bound, and thus the cover is optimal.
If the subtree is a fortress and g is even, we can construct a cover costing $(w+1)+g/2$, using the cover of the fortress and pairing the gray nodes. To show that this cost is minimal, suppose that k gray nodes are paired with k white nodes, and the remaining white and gray nodes are paired separately. If k is even, then the cost of such a cover would be $(w-k+1)+(g-k)/2+2k$, which is greater than $(w+1)+g/2$. If k is odd, then the cost of this cover is $(w-k)+(g-k+1)/2+2k$, which is again greater than or equal to $(w+1)+g/2$. ■

When all the white leaves belong to a single child of the root, the situation is more delicate. Define a *junior fortress* as a tree with an odd number of white leaves, all of them on long branches, except one that is alone on its branch, called the *top* of the fortress. We have the following:

Theorem 4.35 *Let $w > 0$ be the number of white leaves and $g > 0$ be the number of gray leaves in T', the smallest subtree of T that contains all the unoriented components of genome A relative to genome B. If the root of T' has only one child c with white leaves then the minimal cost of a cover of T' is:*

$t = w + \lceil \frac{g}{2} \rceil$ *if g is odd and the subtree T_c that is rooted at c is neither a fortress nor a junior fortress with c as its top,*
$t = w + \lceil \frac{g}{2} \rceil + 1$ *otherwise.*

Proof Suppose first that $g = 1$, then the only gray leaf either belongs to T_c or not. In the first case, this gray leaf must be the child c implying that T_c is not a junior fortress. If T_c is not a fortress, then there exists a cover with minimal cost equal to the number of leaves of T_c, which is given by $w + \lceil \frac{g}{2} \rceil$, since $g = 1$. If T_c is a fortress, then the minimal cost of a cover is $w + \lceil \frac{g}{2} \rceil + 1$.
In the other case, i.e. the gray leaf does not belong to T_c, then if T_c is a fortress or a junior fortress with c as its top, the whole tree T' is a fortress with $w + \lceil \frac{g}{2} \rceil$ leaves, yielding a cost of $w + \lceil \frac{g}{2} \rceil + 1$. Otherwise, if T_c is neither a fortress nor a junior fortress, then T' can not be a fortress, and hence can be destroyed with cost $w + 1 = w + \lceil \frac{g}{2} \rceil$. The same argumentation holds for any $g > 1$ if g is odd.
Now, we consider the case $g = 2$. If T_c is a fortress, two of the white leaves in T_c can be paired with the two gray leaves outside T_c at cost 4. This eliminates the two gray leaves, two of the long white branches, and the branch containing c. The remaining $w - 2$ long branches are paired at cost $w - 2$. Together, this gives a cover of cost $4 + w - 2 = w + \lceil \frac{g}{2} \rceil + 1$. This is optimal since the cost of T' is the same as for T_c. If T_c is not a fortress, we do not need to pair white and gray leaves. T_c can be covered with cost $w + 1$ and the g gray leaves are paired with cost $\lceil \frac{g}{2} \rceil$, giving again a total cost of $w + \lceil \frac{g}{2} \rceil + 1$.

If $g > 2$ and g is even, it is always possible to pair the gray leaves, as long as there are more than two left, and then apply the case $g = 2$. This gives the same cost $w + \lceil \frac{g}{2} \rceil + 1$. ■

For example, the genomes A and B of Example 4.5 have $N = 17$ genes. The adjacency graph $AG(A, B)$ has $C = 3$ cycles and $I = 6$ odd paths. After removing the dangling black leaf, the tree T' has $g = 2$ gray leaves and $w = 1$ white leaf (see Fig. 4.3). Therefore, by Theorem 4.35, we have $t = 2$ and thus

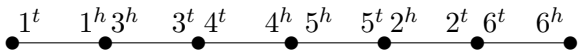
$$d_{HP}(A, B) = N - (C + \frac{I}{2}) + t = 17 - (3 + 3) + 2 = 13.$$

4.4 Relation to Other Genomic Distances

In this section, we are looking at two further genomic distances: the inversion distance (Section 4.4.1) and the translocation distance (Section 4.4.2). Similar to the general case studied in the previous sections, both distance computations are based on the nesting and linking relation of components, but this time, the components are directly defined on the elements of a permutation. Although both distance computations rely on the same concepts, there is a fundamental difference in destroying unoriented components: in the multichromosomal case, we are working with a forest instead of a single tree. Recently, this finding led to a correction of the original formula for the translocation distance. The connection to the existing literature is shortly given at the end of each subsection. For a detailed and complete discussion of the problems, we refer the reader to [20] for the inversion distance and to [21] for the translocation distance.

4.4.1 Inversion Distance

Let A be a unichromosomal, linear genome defined on the set of genes $\{1, \ldots, N\}$. The genome graph of A consists of one linear component. For example, consider the genome graph of the genome $A = \{(\circ,\ 1, -3,\ 4, -5, -2,\ 6, \circ)\}$:



We are interested in comparing two unichromosomal, linear genomes A and B defined on the same set genes $\{1, \ldots, N\}$. We can assume that both genomes have the same telomeres. Moreover, by renaming the genes, one of the genomes to be compared, say genome B, is represented by the identity $Id = (1, \ldots, N)$. Out of the set of HP operations, only inversions act on uni-chromosomal, linear genomes. This gives rise to the following problem:

The Inversion Distance Problem. Given two unichromosomal, linear genomes A and B, compute the minimum number of inversions needed to transform A into B. This number is called the *inversion distance* between A and B, denoted by $d_{inv}(A, B)$. By assuming that $B = Id$, we will simply write $d_{inv}(A)$ instead of $d_{inv}(A, B)$.

In order to compute the inversion distance between two linear genomes, we first have a look at their DCJ distance. Consider two unichromosomal, linear genomes A and B and let $AG(A, B)$ be the adjacency graph of Definition 3.6. Since the number of

odd paths equals two, and thus is constant, the DCJ distance between A and B only depends on the number of cycles. Thus, by Theorem 3.11, we have that

$$d_{DCJ}(A, B) = N - (C + 1)$$

where C is the number of cycles in in the adjacency graph $AG(A, B)$.
However, the distance given by this formula does not guarantee that all intermediate genomes are unichromosomal and linear. Therefore, an additional cost is needed that is computed from the components in a very similar way as for the HP distance. By definition, the forest of Definition 4.21 consists of one single tree and all components of two linear genomes are real components. For this reason, the tree associated to two linear genomes has only white and black, and no gray round nodes. For example, Fig. 4.4 represents the tree T'_{A_2} associated to the genomes $A_2 = \{(\circ,\ 1, -4,\ 2,\ 3,\ 5,\ 7,\ 6,\ 8,\ -16, -14, -15, -13, -11, -12, -10,\ 9,\ 17,\ \circ)\}$ and $B = Id$.

$A_2 = \{(\circ,\ 1\ -4,\ 2,\ 3,\ 5,\ 7,\ 6,\ 8,\ -16,\ -14,\ -15,\ -13,\ -11,\ -12,\ -10,\ 9,\ 17,\ \circ)\}.$

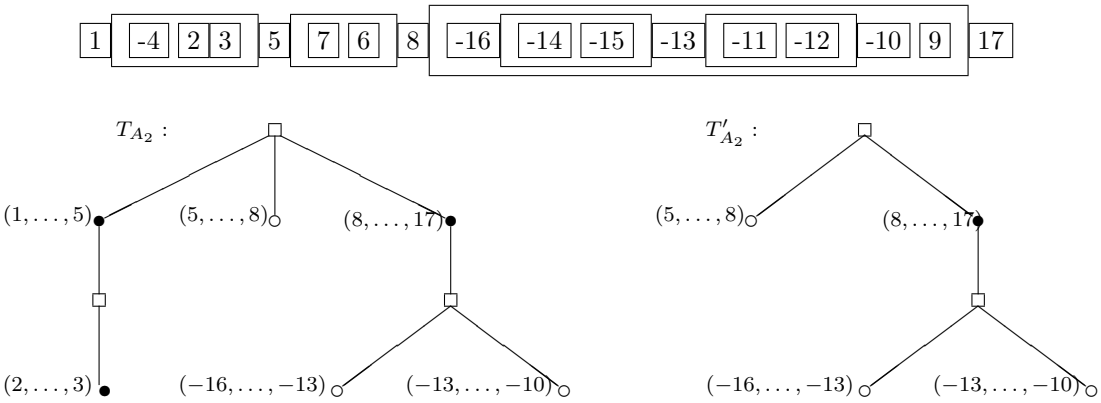


Figure 4.4: The tree T_{A_2} and the tree T'_{A_2} associated to the genomes $A_2 = \{(\circ,\ 1, -4,\ 2,\ 3,\ 5,\ 7,\ 6,\ 8,\ -16, -14, -15, -13, -11, -12, -10,\ 9,\ 17,\ \circ)\}$ and $B = Id$. White round nodes correspond to unoriented components, and black round nodes correspond to oriented components.

Let T' be the smallest unrooted subtree of T that contains all unoriented components of A with respect to B. For example, the tree T'_{A_2} is obtained from the tree T_{A_2} of Fig. 4.4 by removing the dangling oriented components $(2, \ldots, 3)$ and $(1, \ldots, 5)$. Thus, the tree T'_{A_2} contains three unoriented components and one oriented one. Since all components of the tree T' are real, the extra cost for destroying the white leaves is given by a simplified version of Theorem 4.34:

Theorem 4.36 ([18]) *Let A and B be two unichromosomal, linear genomes that are defined on N genes and have the same telomeres. If the adjacency graph $AG(A, B)$ has C cycles and the associated tree T has w white leaves, then*

$$\begin{aligned} d_{inv}(A, B) &= N - (C + 1) + t \\ &= d_{DCJ}(A, B) + t \end{aligned}$$

where

$$t = \begin{cases} w+1 & \textit{if } T \textit{ is a fortress} \\ w & \textit{otherwise.} \end{cases}$$

For example, the adjacency graph of the genomes

$$A_2 = \{(\circ,\ 1\ -4,\ 2,\ 3,\ 5,\ 7,\ 6,\ 8,\ -16,\ -14,\ -15,\ -13,\ -11,\ -12,\ -10,\ 9,\ 17,\ \circ)\}$$

and $B = Id$ has 6 cycles, as shown in Fig. 4.4. Its associated tree T' can be covered by one long path and one short path, since it has three leaves, all of them on short branches. Thus:

$$d(A_2) = N - (C+1) + t = 17 - (6+1) + 3 = 13.$$

Relation to Previous Literature. The inversion distance formula given in Theorem 4.36 was first developed by Hannenhalli and Pevzner [63] in 1995. They introduced the notions of *hurdles* and *fortresses* in order to express the inversion distance in terms of breakpoints, cycles and hurdles. Moreover, Hannenhalli and Pevzner gave the first polynomial-time algorithm for the problem of sorting by inversions using the concepts developed by Bafna and Pevzner [9]. A clear distinction between the problem of computing the inversion distance and finding an optimal sorting sequence was worked out by Kaplan *et al.* [71] and Bader *et al.* [6]. Currently, the most efficient algorithms to solve the inversion distance problem are linear [6, 18], while the most efficient algorithm to find optimal sorting sequences is subquadratic [119].
In the traditional analysis of the sorting by inversions problem, the cycles of the breakpoint graph and the connected components of the overlap graph play an important role. Therefore, a genome is modeled as a signed permutation. Consider a genome A on the set of genes $\{1, \ldots, N\}$. By replacing the telomere markers by 0 at the beginning and by $N+1$ at the end of the chromosome of A, we get a *signed permutation* P_A on the integers $\{0, \ldots, n\}$, where $n = N+1$. For the above genome $A = \{(\circ,\ 1, -3,\ 4, -5, -2,\ 6, \circ)\}$, we get:

$$P_A = (0 \quad 1 \quad -3 \quad 4 \quad -5 \quad -2 \quad 6 \quad 7)$$

Instead of working on graphs, an *inversion* can also be modeled by changing the order and the signs of an interval of genes of the signed permutation:

$$\begin{aligned} P &= (0 \quad 1 \quad -3 \quad 4 \quad \underline{-5 \quad -2} \quad 6 \quad 7) \\ P' &= (0 \quad 1 \quad -3 \quad 4 \quad 2 \quad 5 \quad 6 \quad 7) \end{aligned}$$

The most dominant parameter in the distance formula is the number of cycles. Usually, cycles are defined in the *breakpoint graph* whose most common version is based on an unsigned permutation of $2n$ elements defined as follows: replace any positive element x of a signed permutation by $2x-1, 2x$ and any negative element $-x$ by $2x, 2x-1$. The breakpoint graph is an edge-colored graph whose set of vertices are the elements $(p_0, \ldots, p_{2n-1})$ of this unsigned permutation.

For each $0 \leq i < n$, vertices p_{2i} and p_{2i+1} are joined by a black edge, and elements $2i$ and $2i+1$ of the permutation are joined by a gray edge. Thus, each vertex of the breakpoint graph has exactly two incident edges. This allows the unique decomposition of the breakpoint graph into *cycles*. The number of cycles c is maximized and equals n, if and only if the permutation P is the identity permutation Id. Note that $c = C + 2$ where C is the number of cycles defined in the adjacency graph.
Traditionally, components are defined as connected components of a graph. The *support* of a gray edge is the interval of elements between and including the endpoints. Two gray edges *overlap* if their supports intersect without proper containment. The *overlap graph* is the graph whose vertices are the gray edges of the breakpoint graph and whose edges join overlapping gray edges.
In the literature the notion of *hurdle* is handled in different ways: Hannenhalli and Pevzner [63] define *minimal hurdles* as unoriented components which are minimal with respect to the order induced by span inclusion. In addition, the greatest element is a hurdle, called *greatest hurdle*, if it does not separate any two minimal hurdles. Kaplan *et al.* [71] do not distinguish between minimal and greatest hurdles since they order the elements of unoriented components on a circle. They define a *hurdle* as an unoriented connected component whose elements occur consecutively on the circle. Regardless of the precise definition of a hurdle, hurdles can be classified as follows: A *simple hurdle* is defined as a hurdle whose elimination decreases the number of hurdles, otherwise the hurdle is called a *super-hurdle*. A *fortress* is a permutation that has an odd number of hurdles, all of which are super-hurdles.
Let P be a permutation on the set $\{0, \ldots, n\}$, Hannenhalli and Pevzner proved the following:

$$d(P) = \begin{cases} n - c + h + 1, & \text{if } P \text{ is a fortress} \\ n - c + h, & \text{otherwise} \end{cases}$$

where c is the number of cycles and h is the number of hurdles of permutation P. Note that the above distance formula is equivalent to the formula given in Theorem 4.36 since $n = N + 1$, $c = C + 2$ and $h = w$.
It should be mentioned that, alternatively to the breakpoint graph, cycles and components can also be defined directly on the permutation by using *elementary intervals* [18].

4.4.2 Translocation Distance

In this section, we are focusing on multichromosomal, linear genomes that are affected by translocations that do not act on the telomeres of their chromosomes. As defined previously, a translocation acts on two adjacencies belonging to different chromosomes. If none of the two adjacencies is a telomere, the translocation is *internal*. For example, a translocation applied on the adjacencies $(4, 3)$ and $(2, -7)$ of genome A

$$A = \{(4\ \underline{3}),\ (1\quad 2\quad \underline{\text{-7}}\quad \underline{5}),\ (6\quad \text{-8}\quad 9)\}$$

results in a new genome:

$$A' = \{(4\ \text{-7}\quad 5),\ (1\quad 2\quad 3),\ (6\quad \text{-8}\quad 9)\}$$

Multichromosomal, linear genomes can be sorted by internal translocations, if the genomes have the same chromosome ends. More precisely, for a chromosome $X = (\circ, x_1, \ldots, x_k, \circ)$, the elements x_1 and $-x_k$ are called its *tails*. Two genomes are *co-tailed*, if their sets of tails are equal. Since an internal translocation does not change the set of tails of a genome, sorting by internal translocations is restricted to co-tailed genomes.
Given two genomes A and B, we assume that both genomes are co-tailed and that the elements of each chromosome of the target genome B are positive and in increasing order. For example, we have that

$$A = \{(\,4 \quad 3), \quad (\,1 \quad 2 \quad -7 \quad 5), \quad (6 \quad -8 \quad 9)\}$$
$$B = \{(\,1 \quad 2 \quad 3), \quad (4 \quad 5), \quad (6 \quad 7 \quad 8 \quad 9)\}.$$

The Translocation Distance Problem. Given two co-tailed genomes A and B, compute the minimum number of internal translocations needed to transform A into B. This number is called the *translocation distance* between A and B, denoted by $d_{trans}(A, B)$. By assuming that the elements of B are positive and in increasing order, we will shortly use $d_{trans}(A)$ instead of $d_{trans}(A, B)$.

Suppose that the co-tailed genomes A and B have K chromosomes and that $AG(A, B)$ is the adjacency graph as given by Definition 3.6. Observe that the adjacency graph consists of $2K$ 1-paths and otherwise only cycles. Due to this, the DCJ distance given by Thereom 3.11 equals

$$d_{DCJ}(A, B) = N - (C + K)$$

where C is the number of cycles in $AG(A, B)$.
According to Definition 4.21, we define for each chromosome X a forest F_X by its components. Given a genome A consisting of chromosomes $\{X_1, X_2, \ldots, X_K\}$, the forest F_A is the set of trees from the forests $\{F_{X_1}, F_{X_2}, \ldots, F_{X_K}\}$. Note that, in contrast to the general HP formula, these trees are *not* joined into one large tree. Consider for example the genomes

$$A_3 = \{(\circ, \quad 1, \quad -2, \quad 3, \quad 8, \quad 4, \quad -5, \quad 6, \quad \circ),$$
$$(\circ, \quad 7, \quad 9, \quad -10, \quad 11, \quad -12, \quad 13, \quad 14, \quad -15, \quad 16, \quad \circ)\}.$$

and $B = Id$. Figure 4.5 shows the forest F_{A_3} that consists of three trees.
In the context of sorting by translocations, destroying unoriented components is delicate, and somehow dynamic. It is also possible to eventually destroy by a single translocation two components that initially belong to two different trees of the same chromosome. Before destroying them, such components have to be *separated* during the sorting processes [21]. Therefore, the forest instead of a tree is the basic construct for the sorting algorithm, as well as for the distance formula given in the next theorem.

Theorem 4.37 ([21]) *Let A and B be two co-tailed genomes defined on N genes. If the adjacency graph $AG(A, B)$ has C cycles and the forest F_A has L leaves and T trees, then*

$$\begin{aligned} d_{trans}(A) &= N - (C + K) + t \\ &= d_{DCJ}(A, B) + t \end{aligned}$$

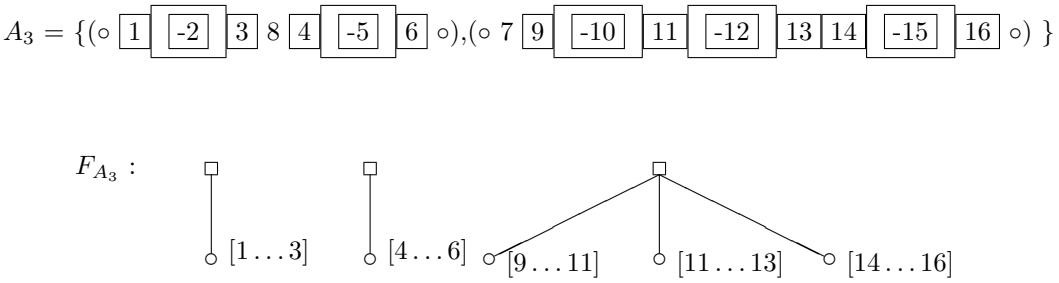


Figure 4.5: The non-trivial components of genome $A_3 = \{(1-2\ 3\ 8\ 4-5\ 6), (7\ 9-10\ 11-12\ 13\ 14-15\ 16)\}$ and the forest F_{A_3}.

where

$$t = \begin{cases} L+2 & \text{if } L \text{ is even and } T = 1 \\ L+1 & \text{if } L \text{ is odd} \\ L & \text{if } L \text{ is even and } T \neq 1. \end{cases}$$

For example, the genome A_3 of Figure 4.5consists of two chromosomes and 16 elements. The adjacency graph $AG(A_3, Id)$ has seven cycles. The forest F_{A_3} has three trees and five leaves. Therefore, we have

$$d(A_3) = N - (C + K) + t = 16 - (7 + 2) + 6 = 13.$$

Relation to Previous Literature. The algorithmic study of genome rearrangement by internal translocations was pioneered by Kececioglu and Ravi [73] in 1995. One year later, Hannenhalli [60] gave the first polynomial time algorithm for sorting a genome by internal translocations. In 2004, Li *et al.* [79] developed the first linear-time algorithm for computing the translocation distance. But, their algorithm relies on Hannenhalli's distance formula that, shortly after, was proven to be wrong [21]. Bergeron *et al.* corrected the distance formula and presented an $\mathcal{O}(n^3)$ time sorting algorithm. By adapting the concepts of the subquadratic algorithm for sorting by inversions [119], Ozery-Flato and Shamir obtained a subquadratic algorithm [92] for sorting by translocations. To make use of the concepts developed in the unichromosomal case, it is common to simulate translocations by inversions of intervals of signed permutations, see [62, 91, 120]. For a genome A with K chromosomes, there are $2^K K!$ possible ways to chain the K chromosomes, each of these is called a *concatenation*. Given a concatenation, we extend it by adding a first element 0 and a last element $N + 1$. This results in a signed permutation P_A on the set $\{0, \ldots, N + 1\}$. We can model translocations on the genome A by inversions on the signed permutation P_A. Sometimes it is necessary to flip a chromosome. This can also be modeled by the inversion of a chromosome, but does not count as an operation in computing the translocation distance since the represented genomes are identical. See Fig. 4.6 for an example.
Given any concatenation, *cycles* are defined on the signed permutation P_A similar to the unichromosmal case. In order to make the distance computation independent of the concatenation, we distinguish between dashed cycles and solid cycles, as shown in Fig. 4.6. More precisely, for two co-tailed genomes with K chromosomes, the dashed

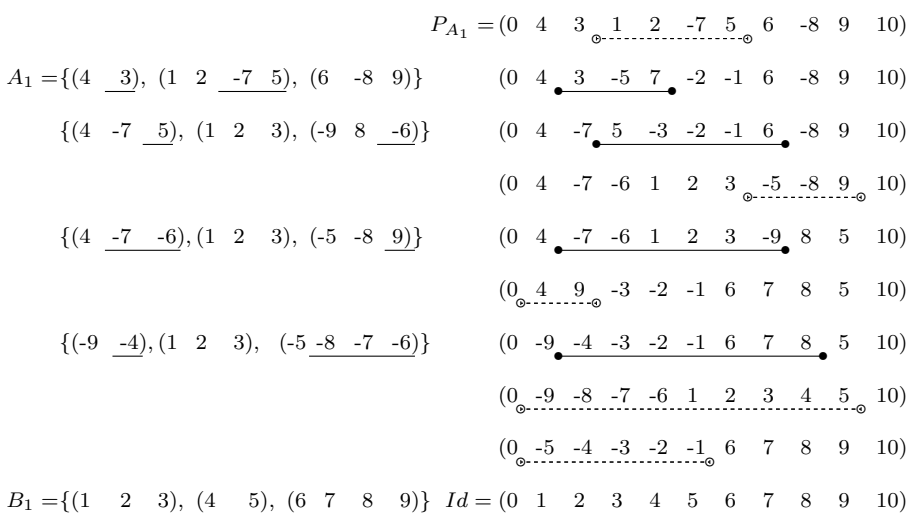


Figure 4.6: *Left:* An optimal sorting scenario for the translocation distance problem for the genomes A and B; the exchanged chromosome ends are underlined. *Right:* Given an arbitrary concatenation, the problem can be modeled by sorting the signed permutation P_A by inversions; solid lines denote inversions that represent translocations, dashed lines denote inversions that flip chromosomes.

cycles formed by the $K + 1$ dashed adjacencies depend on the concatenation. Since the order and the orientation of the chromosomes are irrelevant for the sorting by translocation problem, we focus on the solid cycles that are formed by the $N - K$ solid adjacencies. The number of solid cycles c of P_A is maximized, and equals $N - K$, if and only if genome A is sorted. In order to compute the translocation distance, Hannenhalli [60] introduced the notions of *subpermutations* and *even-isolation*. We call a component *intrachromosomal*, if all its elements belong to the same chromosome. *Subpermutations* are equivalent to the non-trivial intrachromosomal components defined in the previous section. A genome A has an *even-isolation* if all the minimal subpermutations of A reside on a single chromosome, the number of minimal subpermutations is even, and all the minimal subpermutations are contained within a single subpermutation. Hannenhalli showed that

$$d(P) = N - K - c + s + o + 2i$$

where s denotes the number of minimal subpermutations, $o = 1$ if the number of minimal subpermutations is odd and $o = 0$ otherwise, and $i = 1$ if P has an even-isolation and $i = 0$ otherwise. Although the distance formula given by Hannenhalli is correct, there is an error in its proof and in the sorting algorithm as shown in [22].

4.5 Algorithms

The goal of this section is to present a linear time algorithm to compute the HP distance between two linear genomes based on Theorem 4.31. The main part of the algorithm is the component identification of two genomes that is described in Section 4.5.1. The basic idea is to adapt the algorithm for real component identification used in the

unichromosomal case [19] to multichromosomal genomes. Thereafter, we present the overall algorithm for the distance computation in Section 4.5.2. Therefore, we compute the cycles and paths of the adjacency graph in the same way as for the DCJ distance, described earlier in Section 3.4. Finally, we construct the tree associated to the genomes and we compute the optimal cost of a cover of this tree by Theorems 4.34 and 4.35.

4.5.1 Component Identification

For two multichromosomal genomes A and B defined on the set of genes $\{1, \ldots, N\}$, we give an algorithm to compute the components of genome A with respect to B. In the following, we modify the algorithm for unichromosomal genomes [19] such that not only real, but also semi-real components are identfied. Throughout this section, we will follow the notation and terminology used in [19].

Equivalently to Definition 4.4, a real component can be described as an interval from i to $(i + j)$ or from $-(i + j)$ to $-i$, for some $j > 0$, whose set of unsigned elements is $\{i, \ldots, i + j\}$, and that is not the union of two such intervals [19]. The elements i and $i + j$ are called the *bounding elements*. If the bounding elements have positive sign, then the component is called *direct*, otherwise the component is a *reversed* component. In order to extend the notion of direct and reversed components to semi-real components, we replace the telomere markers by *caps* such that the left and the right end of a chromosome are different and that they also have a sign. More concretely, we transform the genome A into two strings: First, we add 0 at the beginning and $N+1$ at the end of each chromosome of genome A and chain its chromosomes into a string P^+. Similarly, we add $-(N+1)$ at the beginning and -0 at the end of each chromosome of genome A and chain its chromosomes into a string P^-. If genome A consists of K chromosomes, then both strings P^+ and P^- have length $l = N + 2K$.

Compared to the uni-chromosomal case, we need to do some extra work in order to identify components: The chromosomes of B are numbered consecutively and an array c stores for each gene in A its chromosome number in B. More precisely, if $P^+[i] = P^-[i]$, then $c[i]$ is the chromosome number in B of the element at index i. Otherwise, at index i is a chromosome end and $c[i]$ gets the chromosome number of its adjacent gene.

Now, with the help of array c, we are able to describe components of genome A and B as intervals in the strings P^+ and P^-. Depending whether the bounding elements are caps or genes, there are eight different types of components: four direct components shown at the left and four reversed components shown at the right of Fig. 4.7.

Example 4.38 *Consider the genomes A and B of Example 4.5 with $N = 17$ genes:*

$$\begin{aligned} A &= \{(\circ, 2, 1, 3, 5, 4, \circ), (\circ, 6, 7, -11, -9, -10, -8, 12, 16, \circ), (\circ, 15, 14, -13, 17, \circ)\}, \\ B &= \{(\circ, 1, 2, 3, 4, 5, \circ), (\circ, 6, 7, 8, 9, 10, 11, 12\circ), (\circ, 13, 14, 15, \circ), (\circ, 16, 17, \circ)\}. \end{aligned}$$

By replacing the telomere markers by the caps 0 and 18 (or -18 and -0), we get the string P^+ (or P^- respectively) and the array c that stores the chromosome in B

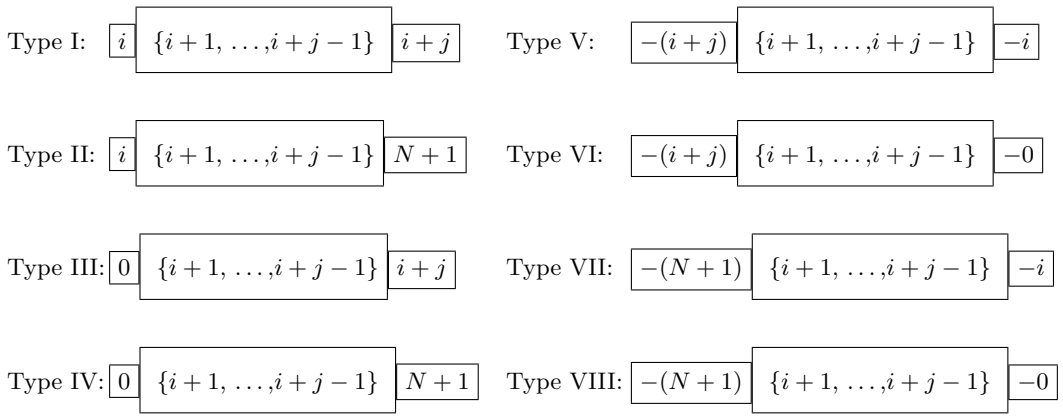


Figure 4.7: **Left:** Direct components. **Right:** Reversed components.

for each gene:

P^+	:	0	2	1	3	5	4	18	0	6	7	−11	−9	−10	−8	12	16	18	0	15	14	−13	17	18
P^-	:	−18	2	1	3	5	4	−0	−18	6	7	−11	−9	−10	−8	12	16	−0	−18	15	14	−13	17	−0
c	:	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	4	4	3	3	3	3	4	4

The components of genome A relative to genome B can be classified according to their bounding elements. There are six direct components: $(0,\ldots,3)$ *and* $(0,6)$ *are of type III,* $(3,\ldots,18)$ *and* $(17,18)$ *are of type II, and* $(6,7)$ *and* $(7,\ldots,12)$ *are of type I. The two remaining components are reversed, the real component* $(-11,\ldots,-8)$ *of type V and the semi-real component* $(-18,\ldots,-13)$ *of type VII.*

It should be noticed that real components are also known as *framed common intervals* and that the algorithm presented in [19] is based on the algorithm for finding framed common intervals given in [16]. The more general concept of *common intervals* was recently used in the context of gene cluster detection and efficient [122] and simple [15] algorithms were developed. Probably, with appropriate post-processing, it would be possible to use also these algorithms. However, our approach described in the following is a more direct one.

Preprocessing

The input for the component identification algorithm is a string of length l, either P^+ or P^-, separated into an array of unsigned elements $\pi = (\pi_0, \pi_1, \ldots, \pi_{l-1})$ and an array of signs $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{l-1})$.
For each string P^+ and P^-, two further arrays M and m are computed beforehand. The array M is defined as follows: $M[i]$ is the nearest unsigned element of π that precedes π_i, and is greater than π_i, and $N+1$ if no such element exists, or if index i is a cap. Similarly, $m[i]$ is the nearest unsigned element of π that precedes π_i and is smaller than π_i, and 0 if no such element exists or if index i is a cap.

Like in the unichromosomal case [19], the arrays M and m can efficiently be computed using two stacks M_1 and M_2, as shown for P^+ in Algorithm 4. For example, the arrays π^+, σ^+, M and m of the string P^+ of Example 4.38 are given by:

π^+	:	0	2	1	3	5	4	18	0	6	7	11	9	10	8	12	16	18	0	15	14	13	17	18
σ^+	:	+	+	+	+	+	+	+	+	+	+	−	−	−	−	+	+	+	+	+	+	−	+	+
M	:	18	18	2	18	18	5	0	18	18	18	18	11	11	10	18	18	0	18	18	15	14	18	0
m	:	0	0	0	1	3	3	0	0	0	6	7	7	9	7	8	12	0	0	0	0	0	13	0

The corresponding arrays for the other string P^- are computed analogously, and we get:

π^-	:	18	2	1	3	5	4	0	18	6	7	11	9	10	8	12	16	0	18	15	14	13	17	0
σ^-	:	−	+	+	+	+	+	−	−	+	+	−	−	−	−	+	+	−	−	+	+	−	+	−
M	:	18	3	3	5	18	18	18	18	7	11	12	10	12	12	16	18	18	18	17	17	17	18	18
m	:	0	1	0	0	4	0	0	0	0	0	9	8	8	0	0	0	0	0	14	13	0	0	0

Algorithm 4 (Preprocessing for $P^+ = (\pi^+, \sigma^+)$, adapted from [19])

1: M_1 and M_2 are stacks of integers; initially M_1 contains $N+1$ and M_2 contains 0

2: **for** $i \leftarrow 0, \ldots, l-1$ **do**
3: $M[i] \leftarrow N+1$
4: $m[i] \leftarrow 0$
5: i++
6: **while** $\pi^+[i] \neq N+1$ **do**

(* Compute the $M[i]$ *)
7: **if** $\pi^+[i-1] > \pi^+[i]$ **then**
8: push $\pi^+[i-1]$ on M_1
9: **else**
10: pop from M_1 all entries that are smaller than $\pi^+[i]$
11: **end if**
12: $M[i] \leftarrow$ the top element of M_1

(* Compute the $m[i]$ *)
13: **if** $\pi^+[i-1] < \pi^+[i]$ **then**
14: push $\pi^+[i-1]$ on M_2
15: **else**
16: pop from M_2 all entries that are larger than $\pi[i]$
17: **end if**
18: $m[i] \leftarrow$ the top element of M_2
19: i++
20: **end while**
21: $M[i] \leftarrow N+1$
22: $m[i] \leftarrow 0$
23: **end for**

Component Identification

Now, we are coming to the component identification algorithm, consisting of two steps. In the first step, we identify the direct components of types I, III and IV and the reversed components of type VII by going from left to right through string P^+, and then, in the second step, we compute the reversed components of types V, VI and VIII and the direct components of type III by going from right to left through string P^-. Thus, in both steps, direct as well as reversed components are reported.
To find components in Step 1 (and Step 2), we systematically test for each index i from 0 to $l-1$ (or from $l-1$ to 0, respectively), whether there is a component with right bounding element π_i, *i.e.* an interval of the form $(\pi_s \dots \pi_i)$. Potential positive starting positions for direct components are stored in stack S_1 and negative ones for reversed components in stack S_2.
Four additional arrays are required for the component identification: Min and Max for direct components, and min and max for reversed components. Concretely, if j is the top of stack S_1, then $Min[j]$ is the minimum and $Max[j]$ is the maximum between j and the current index i, excluding chromosome caps. In the same way, the arrays min and max are defined using the stack S_2. Since direct and reversed components are reported in both steps, all four arrays and both stacks S_1 and S_2 are updated in both steps.

The component identification in Step 1, shown as pseudocode in Algorithm 5 is as follows: First, let us consider direct real components. Additionally to the three conditions described in [19], we require that all elements of a component must be on the same chromosome in B. In line 12 of Algorithm 5, when a left bounding element s is tested, σ_s must be positive because s is the top of the stack S_1. Thus, we test s by the following criterion: $(\pi_s \dots \pi_i)$ is a component of type I if and only if:

1. σ_i is positive,
2. all elements between π_s and π_i in π are greater than π_s and smaller than π_i, the latter being equivalent to the simple test $M[i] = M[s]$,
3. $\pi[s] \neq 0$ and $Max[s] - Min[s] = i - s$, and
4. all elements from s to i are on the same chromosome in B.

It should be mentioned that the third condition is equivalent, but slightly different to the one given in [19]. For real components, we have that $\pi_i - \pi_s = Max[s] - Min[s]$. The reason for testing $Max[s] - Min[s] = i - s$ instead of $\pi_i - \pi_s = i - s$ is that the latter one is not extendable to semi-real components where one or both bounding elements are caps.

Now, we turn from real to semi-real components. Our strategy in Step 1 is to report semi-real components whose left bounding element is a cap. These are direct components of type III and reversed components of type VII. Semi-real components whose right bounding element is a cap will be reported in Step 2.
First, let us consider components of type III. Compared to real components, only the third condition has to be modified. Since the left bounding element is a cap, the number

of genes in the interval is one less than for real components. Moreover, $Min[s]$ is the minimum of the interval $(\pi_s, \ldots, \pi_i)$ and there must be a gene $Min[s]$ at the left end of a chromosome in B. Bringing this together, the two conditions are replaced by:

2. all elements between π_s and π_i in π are smaller than π_i, which is equivalent to the simple test $M[i] = M[s]$,

3. $\pi_s = 0$ and $Max[s] - Min[s] = i - s - 1$ and $Min[s]$ is a left chromosome end in B.

In line 13 of Algorithm 5, components of type I are reported together with components of type III since their conditions are the same, except for the third one (see line 12).

The identification of components of types V and VII is similar to the one of types I and III. To switch from direct to reversed, we change the sign of σ_i and replace Max and Min by max and min in the four conditions (see line 19). As a consequence, the following four conditions have to be tested for the identification of components of type VII:

1. σ_i is negative,

2. all elements between π_s and π_i in π are greater than π_i which is equivalent to the simple test $m[i] = m[s]$,

3. $\pi[s] = 0$ and $max[s] - min[s] = i - s - 1$ and $max[s]$ is at a right chromosome end in B, and

4. all elements from s to i are on the same chromosome in B.

Note that we only report reversed components of type VII in line 20 of Algorithm 5. Even though the direct components of type V are identified and the arrays are updated in Step 1, they will not be reported.

Finally, at the end of a chromosome, when $\pi_i = N + 1$, semi-real components whose bounding elements are both caps and have positive sign are identified. As long as we have $\pi_s \neq 0$ for the top element of stack S_1, we remove s (line 30) and update Min and Max (line 31). Then, we test whether the whole chromosome is a semi-real component. In particular, we have that $\pi_s = 0$ and $\pi_i = N+1$. Obviously, all elements between π_s and π_i are greater than π_s and smaller than π_i, making the second condition meaningless. Since both bounding elements are caps, the number of genes is two less than for real components. Furthermore, there must exist a chromosome in B of the form $(\circ, Min[s], \ldots, Max[s], \circ)$. Altogether, we have to check for components of type IV the following conditions in line 33:

1. σ_{s+1} is positive,

2. (not applicable)

3. $Max[s] - Min[s] = i - s - 2$ and $Min[s]$ is at a left chromosome end in B and $Max[s]$ is at a right chromosome end of B,

4. all elements from s to i are on the same chromosome in B

5. the component is not a chain of shorter components.

In Step 2, we apply Algorithm 5 to string P^- instead of P^+. By going backwards through P^-, we report the remaining components in the following order: First, the components of types V and VI are found in the same way as the components of types I and III in Step 1. Then, by switching from reversed to direct, we test for the components of types I and II, but we only report those of type II because the others are already found in Step 1. Finally, at the end of each chromosome, we compute the components of type VIII.

Component Classification

Up to now, Algorithm 5 reports all components of genome A with respect to B and it remains to classify which of them are unoriented, and which are not. For the classification of components of types I-III and of types V-VII, it is sufficient to test whether all elements of the component have the same sign. As shown in [19], this can be done by a slight modification of Algorithm 5, without affecting the running time. The classification for the components of types IV and VIII is more delicate and requires the computation of the paths and cycles that belong to the component.

In order to test whether all elements of a component have the same sign, we need an extra array o to store the signs of any two consecutive elements π_i and π_{i+1} of the permutation P^+ (for ease of notation shifted down by one position). For $0 \leq i < l$, the entries of the array o are initially defined as follows:

$$o[i] = \begin{cases} +, & \text{if } \sigma_i = + \text{ and } \sigma_{i+1} = + \\ -, & \text{if } \sigma_i = - \text{ and } \sigma_{i+1} = - \\ 0, & \text{otherwise.} \end{cases}$$

For example, the initial array o of permutation P^+ of Example 4.38 is:

P^+	:	0	2	1	3	5	4	18	0	6	7	−11	−9	−10	−8	12	16	18	0	15	14	−13	17	18
σ^+	:	+	+	+	+	+	+	+	+	+	+	−	−	−	−	+	+	+	+	+	+	−	+	+
o	:	+	+	+	+	+	+	+	+	+	0	−	−	−	0	+	+	+	+	+	0	0	+	

Now we define a function $f : \{-, 0, +\}^2 \to \{-, 0, +\}$ as:

$$f(x_1, x_2) = \begin{cases} x_1, & \text{if } x_1 = x_2 \\ 0, & \text{otherwise.} \end{cases}$$

Then, in the modified algorithm, whenever an index s is removed from the stack such that index r becomes the top of the stack, $o[r]$ will be replaced by $f(o[r], o[s])$. We also replace the entry of the left bounding element of an identified direct component by $+$, and the entry of the left bounding element of an identified reversed component by $-$. This way, when a component $(\pi_s \dots \pi_i)$ is reported in Algorithm 5, the signs of all adjacencies belonging to the component are folded by repeated application of function f to the leftmost index s of the component. Its orientation can then easily be derived: $(\pi_s \dots \pi_i)$ is unoriented if and only if $o[s]$ equals $+$ or $-$ (all its points have the same sign).

Algorithm 5 (Phase 1: Find components of $P^+ = (\pi, \sigma)$)

1: **for** $i \leftarrow 0, \ldots, l$ **do**
2: S_1 and S_2 are stacks of integers; initially S_1 contains i and S_2 contains i
3: $Min[i] \leftarrow N$, $Max[i] \leftarrow 0$, $min[i] \leftarrow N$, $max[i] \leftarrow 0$
4: **while** $\pi[i] \neq N + 1$ **do**

(* Update minima and maxima *)
5: $Min[i] \leftarrow \pi[i]$, $Min[\text{top element of } S_1] \leftarrow \min(Min[\text{top element of } S_1], \pi[i])$
6: $Max[i] \leftarrow \pi[i]$, $Max[\text{top element of } S_1] \leftarrow \max(Max[\text{top element of } S_1], \pi[i])$
7: update similarly min and max using stack S_2

(* Find components of types I and III *)
8: **while** $\pi[s] > \pi[i]$ **or** $M[s] < \pi[i]$ **do**
9: pop the top element s from S_1
10: update Min and Max as in line 5 and 6
11: **end while**
12: **if** $\sigma[i] = +$ **and** $M[i] = M[s]$ **and** (($\pi[s] \neq 0$ **and** $Max[s] - Min[s] = i - s$) **or** ($\pi[s] = 0$ **and** $Max[s] - Min[s] = i - s - 1$ **and** $Min[s]$ is at a left chromosome end in B)) **and** all elements from s to i are on the same chromosome in B **then**
13: report the component $(\pi_s \ldots \pi_i)$
14: **end if**

(* Find components of type VII *)
15: **while** ($\pi[s] < \pi[i]$ or $m[s] > \pi[i]$) **and** $\pi[s] > 0$ **do**
16: pop the top element s from S_2
17: update min and max as in line 7
18: **end while**
19: **if** $\sigma[i] = -$ **and** $m[i] = m[s]$ **and** ($\pi[s] = 0$ **and** $max[s] - min[s] = i - s - 1$ **and** $max[s]$ is at a right chromosome end in B) **and** all elements from s to i are on the same chromosome in B **then**
20: report the component $(\pi_s \ldots \pi_i)$
21: **end if**

(* Update stacks *)
22: **if** $\sigma[i] = +$ **then**
23: push i on S_1
24: **else**
25: push i on S_2
26: **end if**
27: $i{+}{+}$
28: **end while**

(* Find components of type IV *)
29: **while** $\pi[s] \neq 0$ **do**
30: pop the top element s from S_1
31: update Min and Max as in lines 5 and 6
32: **end while**
33: **if** $\sigma[s+1] = +$ **and** $Max[s] - Min[s] = i - s - 2$ **and** $Min[s]$ is at a left chromosome end in B **and** $Max[s]$ is at the right chromosome end of B **and** all elements from s to i are on the same chromosome in B **and** this chromosome is not a chain of shorter components **then**
34: report the component $(\pi_s \ldots \pi_i)$
35: **end if**
36: **end for**

Additionally to the array o, the classification of the components of types IV and VIII requires the computation of the cycles and paths that belong to the components. Recall that a component of type IV (or VIII) is unoriented if its elements are positive (respectively negative) and its adjacency graph does not contain an even path. The idea is to mark in an additional boolean array whether an adjacency belongs to an even path or not. This can be done in linear time by a slight modification of Algorithm 3. If a semi-real component $(\pi_s \dots \pi_i)$ is reported in line 34 of Algorithm 5, we check in constant time whether the two adjacencies $(\circ, \pi_{s+1})$ and $(\pi_{i-1}, \circ)$ belong to an even path, or not.

Summarizing, we have that:

Theorem 4.39 *All components of two linear genomes on the set of genes* $\{1, \dots, N\}$ *can be found and classified as oriented or unoriented with a modified version of Algorithm 5 in* $O(N)$ *time and space.*

4.5.2 Distance Computation

Now that we have an algorithm for the component identification, we present a linear-time algorithm for computing the general HP distance between two linear genomes A and B, consisting of five parts:

1. Construct the adjacency graph $AG(A, B)$;
2. Compute the cycles and paths of $AG(A, B)$;
3. Compute the components of A with respect to B;
4. Construct the trees T and T' associated to genomes A and B;
5. Compute the minimal cost of the cover of T'.

The first two steps can be solved by Algorithms 2 and 3 as described in Section 3.4. As discussed in detail in Section 4.5.1, the components can be computed in linear time.
Given the components, we construct the tree in the fourth step as follows: For each index i, $0 \leq i \leq l$, at most one component can start at position i, and at most one component can end at position i. Hence, it is possible to create a data structure that tells, in constant time, if there is a component beginning or ending at position i and, if so, reports such components. Given this data structure, it is a simple procedure to construct the tree T in one left-to-right scan along the permutation. Initially one round root node is created. Then, for each additional component, a new round node p is created as the child of a new or an existing square node q, depending if p is the first component in a chain or not. For details, see Algorithm 6.
Finally, we compute the extra cost. To generate tree T' from tree T, a bottom-up traversal of T recursively removes all dangling round leaves that represent oriented components, and square nodes, including the root if it has degree 1. Given the tree T', it is easy to compute the cover cost: Perform a depth-first traversal of T' and count the number of leaves and the number of long and short branches, including the root if

Algorithm 6 (Construct T from the components of A with respect to B)

```
create a round node p, the root of T
for i ← 1, ..., l − 1 do
    if there is a component C starting at position i then
        if there is no component ending at position i then
            create a new square node q as a child of p
        end if
        create a new round node p (representing C) as a child of q
    else if there is a component ending at position i then
        p ← parent of q
        q ← parent of p
    end if
end for
```

it has degree 1. Then use the formulas from Theorems 4.34 and 4.35 to obtain t, and the formula from Theorem 4.31 to obtain the distance d_{HP}.

Altogether we have:

Theorem 4.40 *The HP distance $d_{HP}(A, B)$ of two genomes A and B on the set $\{1, \ldots, N\}$ can be computed in linear time $O(N)$.*

4.6 Summary and Historical Notes

In this chapter, we have presented three formulas for genomic distances. The main contribution was a simpler formula for the HP distance problem. It requires only a few parameters that can easily be computed directly from the genomes and from simple graph structures derived from the genomes.
In 1995, the first exact distance formula for this problem was given by Hannenhalli and Pevzner and involves more than half a dozen parameters [62]. Although some of them are derived from the problem of sorting by inversions [63], the general formula is much more complicated. In the last decade, different authors pointed to problems in the original formula and algorithm. The first correction was by Tesler [120] whose main contribution was a constructive algorithm how to achieve optimal concatenations. In 2003, Ozery-Flato and Shamir [91] found a counter-example and modified one of the parameters of the distance formula. To show their distance formula, they followed to a large extent the notation introduced by Hannenhalli and Pevzner [62] and in the following, we briefly recapture their results.

Consider two multichromosomal, linear genomes Π and Γ defined on the same set of n genes. Let M and N be the number of chromosomes of Π and Γ. If one of the two genomes has fewer chromosomes than the other, then empty chromosomes are added to this genome, such that afterwards both genomes contain $L = \max(M, N)$ chromosomes each.
Now, we consider a set of $2L$ additional elements, called *caps*. These elements are added at the ends of the linear chromosomes of both genomes such that the resulting genomes, denoted by $\hat{\Pi}$ and $\hat{\Gamma}$, are co-tailed. Chaining the chromosomes of the genomes $\hat{\Pi}$ and $\hat{\Gamma}$

in any order results in two signed permutations π and γ. Let $G(\pi, \gamma)$ be the breakpoint graph as defined in the historical notes of Section 4.4.1. Note that the number of cycles of $G(\pi, \gamma)$ depends on the capping and the concatenation.

Let $G(\Pi, \Gamma)$ be the graph obtained after removing the $2(L+1)$ gray and black edges that define the concatenation and the $2L$ grey edges that define the capping. Since $G(\Pi, \Gamma)$ has $4L$ vertices of degree one, the graph is composed of cycles and $2L$ paths. A path is called a $\Pi\Pi$-path ($\Gamma\Gamma$-path) if both endpoints are telomeres of Π (of Γ). A $\Pi\Gamma$-path is a path that has one telomere of Π and one telomere of Γ.
Similar to the unichromosomal case described in Section 4.4.1, cycles and paths *overlap* in $G(\Pi, \Gamma)$. This allows to define *components* as unions of cycles and paths. Let $\mathcal{RU}$ be the set of real, unoriented components in $G(\Pi, \Gamma)$. Hurdles, super-hurdles and fortresses for the set $\mathcal{RU}$ are called *real-knots*, *super-real-knots* and *fortresses-of-real-knots*.
A component in $G(\Pi, \Gamma)$ containing a $\Pi\Gamma$-path is *simple* if it is not a semi-real-knot. Let $\overline{G}(\Pi, \Gamma)$ be the graph obtained from $G(\Pi, \Gamma)$ by closing all $\Pi\Gamma$-paths in simple components.

Let $\mathcal{IU}$ be the set of intrachromosomal, unoriented components of $G(\Pi, \Gamma)$. A component from the set $\mathcal{IU} \setminus \mathcal{RU}$ is a *semi-real-knot* if (i) it does not contain a $\Gamma\Gamma$-path in its interval, and (ii) closing all the $\Pi\Gamma$-paths in it creates a minimal real-knot or a simple (not super-real-knot) greatest real-knot.
The breakpoint graph G is a *weak-fortress-of-real-knots* if:

1. the number of real-knots in G is odd,
2. one of the real-knots is the greatest real-knot
3. every real-knot but the greatest one is a super-real-knot
4. there exists semi-real knots in $G(\Pi, \Gamma)$.

Ozery-Flato and Shamir [91] have shown that for two genomes Π and Γ

$$d(\Pi, \Gamma) = b(\Pi, \Gamma) - c(\Pi, \Gamma) + r(\Pi, \Gamma) + \left\lceil \frac{s'(\Pi, \Gamma) - gr'(\Pi, \Gamma) + fr'(\Pi, \Gamma)}{2} \right\rceil$$

where

- $b(\Pi, \Gamma)$ is the number of black edges in the graph $G(\Pi, \Gamma)$,
- $c(\Pi, \Gamma)$ is the number of cycles, $\Pi\Gamma$-paths, and $\Gamma\Gamma$-paths,
- $r(\Pi, \Gamma)$ is the number of real-knots in $G(\Pi, \Gamma)$,
- $s'(\Pi, \Gamma)$ is the number of semi-real knots in $G(\Pi, \Gamma)$,
- $gr'(\Pi, \Gamma)$ is equal to 1, if $\overline{G}$ has the greatest real-knot and $s' > 0$, and is 0 otherwise,
- $fr'(\Pi, \Gamma)$ is equal to 1, if either (i) $\overline{G}$ is a fortress of real knots and the greatest semi-real knot does not exist in $\overline{G}$, or (ii) $\overline{G}$ is a weak-fortress-of-real-knots.

The relation to Theorems 4.34 and 4.35 is as follows: It can be shown in the same way as in Section 3.5 that $b(\Pi, \Gamma) - c(\Pi, \Gamma)$ is equal to the DCJ distance. Moreover, the real components are equivalent to the white components, implying that $r = w$. The last term of the distance formula given by Ozery-Flato and Shamir corresponds to $\lceil \frac{g}{2} \rceil$, respectively $\lceil \frac{g}{2} \rceil + 1$.

It should be mentioned that the distance computation is independent of capping and concatenation. However, finding a most parsimonious sorting scenario requires an optimal capping, as well as an optimal concatenation. Very recently, Jean and Nikolski [70] presented a new algorithm for capping the genomes. Moreover, they provide an extensive classification of the connected components of the breakpoint graph.

Chapter 5

Genome Halving under the Double Cut and Join Distance

The Genome Halving Problem is the following: Given a rearranged duplicated genome, find a perfectly duplicated genome such that the distance between these genomes is minimal with respect to some distance measure. Recently, Warren and Sankoff [128] studied this problem under the general DCJ model where the pre-duplicated genome contains both, linear and circular chromosomes. In this chapter we will revisit the Genome Halving Problem under the double cut and join operation where the ancestral genome may contain linear and circular chromosomes. In our genome model, we take into account constraints required for genomes with only linear chromosomes, as well as the ones for genomes with only circular chromosomes. Compared to the more general model studied in [128], these requirements on the ancestral genome increase the distance between the genomes. This yields a simple algorithm for reconstructing an ancestral genome. Moreover, by our results, we will also correct an error in the Warren-Sankoff analysis.

The structure of this chapter is as follows. We begin by formalizing the problem in the next section. Then, in Section 5.2, we study the effect of a DCJ operation on the natural graph. In Section 5.3 we present our distance formula and a linear-time algorithm to reconstruct an ancestral genome with the minimum number of DCJ operations. Finally, we will discuss the Warren-Sankoff formula in Section 5.4. The last section summarizes our results and gives an overview of existing results that can be found in the literature.

5.1 Problem Formulation

First, we briefly recall the terminology. A gene is represented by a directed identifier where the direction is indicated by a head and a tail. These are called the extremities of the gene. The tail of a gene a is denoted by a^t, and its head is denoted by a^h.
An adjacency of two consecutive genes a and b can be of four different types:

$$\{a^h, b^t\}, \{a^h, b^h\}, \{a^t, b^t\}, \{a^t, b^h\}.$$

An extremity that is not adjacent to any other gene is called a telomere, represented by a singleton set $\{a^h\}$ or $\{a^t\}$.

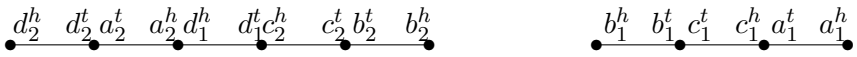
Definition 5.1 *A* duplicated genome *A is a set of adjacencies and telomeres such that the head and the tail of every gene appears exactly twice.*

Thus, a duplicated genome has two identical copies of each gene that are called *paralogs* and we distinguish them by a subscript, called an *assignment of the paralogs*. For a gene a, we denote its copies by a_1 and a_2 and the *paralogous extremities* by a_1^t, a_2^t and a_1^h, a_2^h.

Example 5.2 *Consider the following genome defined on the set of genes $\{a, b, c, d\}$:*

$$\{\{d_2^h\}, \{d_2^t, a_2^t\}, \{a_2^h, d_1^h\}, \{d_1^t, c_2^h\}, \{c_2^t, b_2^t\}, \{b_2^h\}, \{b_1^h\}, \{b_1^t, c_1^t\}, \{c_1^h, a_1^t\}, \{a_1^h\}\}$$

A genome can be represented as a graph, called the genome graph*, with vertices corresponding to the adjacencies and telomeres and edges joining the head and the tail of each paralogous extremity. Thus, we have:*



Suppose that the genome graph consists of K components $C_1, \ldots, C_K$. A *chromosome* is a set of adjacencies and telomeres that belong to the same component. Note that, by definition, each vertex in the genome graph has degree one or two, and thus the components of the genome graph are either *linear* or *circular*. We call a genome *linear* if all its chromosomes are linear. Similarly, a genome is *circular* if all its chromosomes are circular. For example, the above genome graph is a linear genome consisting of two linear chromosomes.

For paralogous extremities, we also use the following notation: if p is an extremity, then $\overline{p}$ is its corresponding paralogous extremity. By elevating this notation to sets of extremities, we can apply it to adjacencies and telomeres. For example, for an adjacency $x = \{a_1^h, b_2^t\}$, we have $\overline{x} = \{a_2^h, b_1^t\}$.

For a chromosome C, we define $\overline{C} = \{\overline{x} \mid x \text{ is an adjacency or telomere of } C\}$. This notation is useful to describe the different notions of a duplicated genome that can be found in the literature, for linear genomes in [50] and for circular genomes in [4]. By bringing this together for genomes with a mixture of linear and circular chromosomes, we have:

Definition 5.3 *A duplicated genome A consisting of chromosomes $C_1, \ldots, C_K$ is*

- linear-perfectly duplicated*, if for each linear chromosome C_i, we have $C_i = \overline{C_j}$ for some $j \in \{1, \ldots, K\} \backslash \{i\}$;*
- circular-perfectly duplicated*, if for each circular chromosome C_i, either we have $C_i = \overline{C_j}$ for some $j \in \{1, \ldots, K\} \backslash \{i\}$ or $C_i = C \cup \overline{C}$, where each adjacency of C_i occurs either in C or in $\overline{C}$, but not in both;*
- perfectly duplicated*, if it is linear- and circular-perfectly duplicated.*

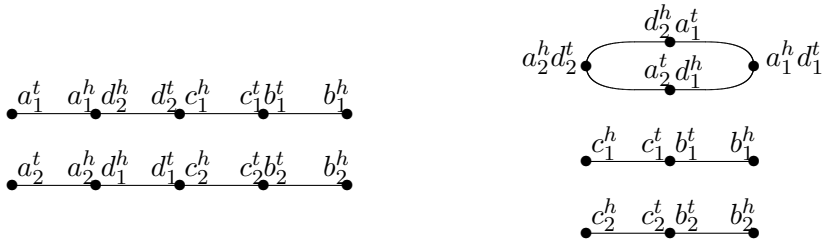


Figure 5.1: Two perfectly duplicated genomes.

Note that this definition does not depend on the assignment of the paralogs. Two examples of perfectly duplicated genomes are given in Fig. 5.1. From the right genome in that figure, we also see that the number of chromosomes of a perfectly duplicated genome is not necessarily even.
Alternatively to the formulation on the level of chromosomes, a perfectly duplicated genome can also be characterized locally, as stated by the next lemma.

Lemma 5.4 *A genome A is perfectly duplicated if and only if*

- *for each adjacency $\{u, v\}$ in A, also $\{\overline{u}, \overline{v}\}$ is in A and $u \neq \overline{v}$, and*
- *for each telomere $\{u\}$ in A, also $\{\overline{u}\}$ is in A.*

Now, let us consider rearrangement operations. Generally speaking, such an operation applied to two adjacencies or telomeres of a genome disconnects the incident edges of the genome graph, and reconnects them in one of the possible other ways. More formally, we have:

Definition 5.5 ([22]) *The* double cut and join *(DCJ) operation acts on two vertices u and v of a graph with vertices of degree one or two in one of the following three ways:*

(a) If both $u = \{p, q\}$ and $v = \{r, s\}$ are internal vertices, these are replaced by the two vertices $\{p, r\}$ and $\{s, q\}$ or by the two vertices $\{p, s\}$ and $\{q, r\}$.

(b) If $u = \{p, q\}$ is internal and $v = \{r\}$ is external, these are replaced by $\{p, r\}$ and $\{q\}$ or by $\{q, r\}$ and $\{p\}$.

(c) If both $u = \{q\}$ and $v = \{r\}$ are external, these are replaced by $\{q, r\}$.

In addition, as an inverse of case (c), a single internal vertex $\{q, r\}$ can be replaced by two external vertices $\{q\}$ and $\{r\}$.

Given two genomes A and B, the *DCJ distance* denoted by $d_{DCJ}(A, B)$ is the minimum number of DCJ operations necessary to transform genome A into genome B. Thus, we can formulate the following problem:

The Genome Halving Problem. Given a rearranged duplicated genome A, find a perfectly duplicated genome B such that the DCJ distance between A and B is minimal.

To solve this problem, we will construct another graph in the next section. Again, the graph is defined on the adjacencies and telomeres of A, but this time it represents the relation between paralogous extremities.

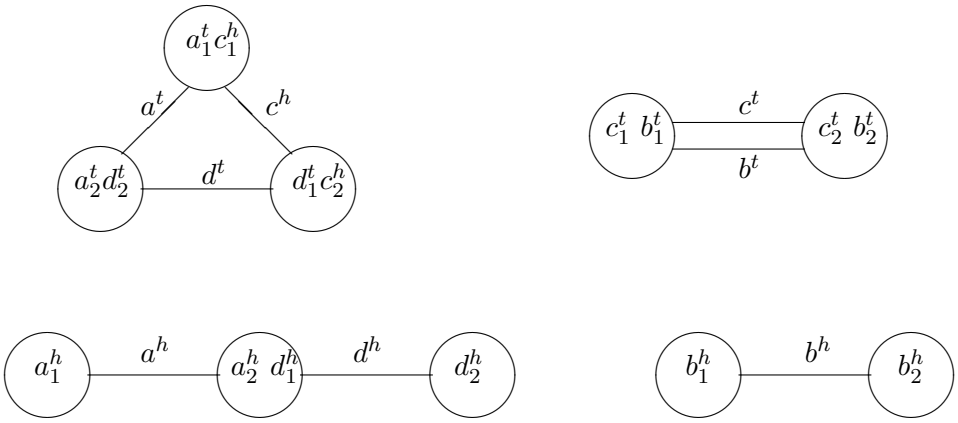


Figure 5.2: Natural graph $N(A)$ of genome A of Example 5.2.

5.2 Natural Graphs

Let us consider a duplicated genome A with N genes, each present in two copies. Assume that the two paralogs of every gene are assigned arbitrarily.

Definition 5.6 *The* natural graph $NG(A)$ *is a graph whose vertices are the adjacencies and telomeres of A and, for each extremity, the two paralogous extremities are connected by an edge, i. e., two vertices u and v are connected if $p \in u$ and $\overline{p} \in v$.*

Observe that the total number of edges in the graph equals two times the number of genes. The natural graph of genome A from Example 5.2 is given in Fig. 5.2.
In a natural graph, by definition, every vertex has degree one or two. Thus, the natural graph consists only of cycles and paths.

Definition 5.7 *A cycle (or a path) with k edges is a* k-cycle *(or* k-path*). If k is even, the cycle (or path) is called* even*, otherwise* odd*.*

Note that an adjacency $\{p, \overline{p}\}$ consisting of two paralogous extremities is a 1-cycle. The set of components of the natural graph can be partitioned into the following four disjoint subsets:

- EC := set of even cycles
- EP := set of even paths
- OC := set of odd cycles
- OP := set of odd paths

The following lemma is an immediate consequence of Lemma 5.4:

Lemma 5.8 *A genome A is perfectly duplicated if and only if all cycles in $NG(A)$ are 2-cycles and all paths in $NG(A)$ are 1-paths, i. e., $N = |EC| + |OP|/2$.*

5.3 Reconstructing an Ancestral Genome

In this section, we solve the genome halving problem by applying DCJ operations to the natural graph. This allows us to reconstruct a perfectly duplicated genome. We will first present our distance formula in Section 5.3.1 and then a linear time algorithm in Section 5.3.2.

5.3.1 Distance Formula

Consider a rearranged duplicated genome A. When a DCJ operation is applied to genome A, it acts on the adjacencies and telomeres of genome A. The same DCJ operation acts also on the natural graph $NG(A)$ since the adjacencies and telomeres of genome A are the vertices of this graph. Because the natural graph is a union of cycles and paths, all the properties of DCJ operations apply here as well, for instance: A DCJ operation can change the number of components only by one, as shown in [22]. Thus, we get a lower bound on the distance:

Lemma 5.9 *For a given genome A and any perfectly duplicated genome B over the same set of $2N$ genes, we have that*

$$d_{DCJ}(A, B) \geq N - (|EC| + \left\lfloor \frac{|OP|}{2} \right\rfloor).$$

In fact, there always exists a DCJ operation that increases either the number of even cycles or the number of odd paths. Thus, the distance decreases and the lower bound is strict as we see in the next theorem.

Theorem 5.10 *Let A be a rearranged duplicated genome with N genes each present in two copies, then the minimal distance between A and any perfectly duplicated genome B equals*

$$d_{DCJ}(A, B) = N - (|EC| + \left\lfloor \frac{|OP|}{2} \right\rfloor)$$

where $|EC|$ is the number of even cycles and $|OP|$ is the number of odd paths in the natural graph $NG(A)$.

Proof We explain how to find a sequence of DCJ operations that achieves the lower bound of Lemma 5.9.
Let J, K, L and M be the total number of edges in all even cycles, even paths, odd cycles and odd paths, respectively. Note that the number of genes equals half of the total number of edges in $NG(A)$, i. e. $N = (J + K + L + M)/2$.
Consider a connected component G of $NG(A)$.

1. If G is an even j-cycle, we can create $\frac{j}{2}$ 2-cycles with $\frac{j}{2} - 1$ DCJ operations. Thus, for $|EC|$ even cycles with J edges in total, we need $\frac{J}{2} - |EC|$ DCJ operations to create $\frac{J}{2}$ 2-cycles.

2. If G is an even k-path, we can create $\frac{k}{2}$ 2-cycles with $\frac{k}{2}$ DCJ operations. Thus, for $|EP|$ even paths with K edges in total, we need $\frac{K}{2}$ DCJ operations to create $\frac{K}{2}$ 2-cycles.

3. If $|OP|$ is even, then $|OC|$ is also even.

   (a) If G is an odd l-cycle, we can create $\frac{l-1}{2}$ 2-cycles and one 1-cycle with $\frac{l-1}{2}$ DCJ operations. Thus, for $|OC|$ odd cycles with L edges in total, we need $\frac{L-|OC|}{2}$ DCJ operations to create $\frac{L-|OC|}{2}$ 2-cycles and $|OC|$ 1-cycles. We can choose two 1-cycles and create one 2-cycle. Since $|OC|$ is even, we can thus create $\frac{|OC|}{2}$ 2-cycles with $\frac{|OC|}{2}$ DCJ operations. Thus, in total we need $\frac{L-|OC|}{2} + \frac{|OC|}{2} = \frac{L}{2}$ DCJ operations.

   (b) If G is an odd m-path, we can create $\frac{m-1}{2}$ 2-cycles and one 1-path with $\frac{m-1}{2}$ DCJ operations. Thus, for $|OP|$ odd paths with M edges in total, we need $\frac{M-|OP|}{2}$ DCJ operations to create $\frac{M-|OP|}{2}$ 2-cycles and $|OP|$ 1-paths.

   Since L and M are even, summing up (a) and (b) gives us in total $\frac{L+M}{2} - \frac{|OP|}{2}$ DCJ operations.

4. If $|OP|$ is odd, then $|OC|$ is also odd.

   (a) If G is an odd l-cycle, we can create $\frac{l-1}{2}$ 2-cycles and one 1-cycle with $\frac{l-1}{2}$ DCJ operations. Thus, for $|OC|$ odd cycles with L edges in total, we need $\frac{L-|OC|}{2}$ DCJ operations to create $\frac{L-|OC|}{2}$ 2-cycles and $|OC|$ 1-cycles. We can choose two 1-cycles and create one 2-cycle. Since $|OC|$ is odd, there is one remaining 1-cycle that can be transformed into a 1-path by one extra DCJ operation. Thus, in total we need $\frac{L-|OC|}{2} + \frac{|OC|-1}{2} + 1 = \frac{L+1}{2}$ DCJ operations.

   (b) If G is an odd m-path, we can create $\frac{m-1}{2}$ 2-cycles and one 1-path with $\frac{m-1}{2}$ DCJ operations. Thus, for $|OP|$ odd paths with M edges in total, we need $\frac{M-|OP|}{2}$ DCJ operations to create $\frac{M-|OP|}{2}$ 2-cycles and $|OP|$ 1-paths.

   Since L and M are odd, summing up (a) and (b) gives us in total $\frac{L+1}{2} + \frac{M-|OP|}{2} = \frac{L+M}{2} - \frac{|OP|-1}{2}$ DCJ operations.

By bringing together the results, the distance formula follows. ■

5.3.2 Algorithm

In this section, we show how the distance computation as well as an algorithm for reconstructing an ancestral genome can be implemented to run in linear time. Based on the proof of Theorem 5.10, our strategy for reconstructing a perfectly duplicated genome is the following:

1. Construct the natural graph
2. Maximize the number of even cycles and odd paths in the natural graph
3. Reconstruct the perfectly duplicated genome from the resulting natural graph

The natural graph can easily be constructed in $O(n)$ time and $O(n)$ space if we store the information about the adjacencies and the telomeres in two tables. The first table

represents the vertices of the natural graph. Each of its entries contains one or two extremities, depending whether it represents an adjacency or a telomere. The edges can be obtained from the second table that stores for each paralogous extremity the index of the vertex that contains it. The two tables for genome A of Example 5.2 are given in Tables 5.1 and 5.2. Thus, the natural graph $NG(A)$ has 10 vertices and 8 edges, for example one edge joining vertex 10 with vertex 3, another edge joining vertex 9 with vertex 2, and so on.

	1	2	3	4	5	6	7	8	9	10
first	d_2^h	d_2^t	a_2^h	d_1^t	c_2^t	b_2^h	b_1^h	b_1^t	c_1^h	a_1^h
second	–	a_2^t	d_1^h	c_2^h	b_2^t	–	–	c_1^t	a_1^t	–

Table 5.1: Table storing the adjacencies and telomeres of genome A. Adjacencies have two entries, telomeres just one.

	a_1	a_2	b_1	b_2	c_1	c_2	d_1	d_2
head	10	3	7	6	9	4	3	1
tail	9	2	8	5	8	5	4	2

Table 5.2: Table storing for each gene in A the location of its head and its tail in Table 5.1.

Using these tables, the connected components can be computed in linear time, and thus the distance as given by Theorem 5.10.
In order to reconstruct a perfectly duplicated genome, we maximize the number of even cycles and odd paths in the natural graph. This is done by Algorithm 7, following the idea used in the proof of Theorem 5.10. By marking each adjacency of Table 3.1, Algorithm 7 can be implemented in linear time. The adjacencies are processed in left-to-right order, and each time an unmarked adjacency is detected, all adjacencies on its path or cycle are marked and transformed into 2-cycles and 1-paths by successively applying DCJ operations. Note that, by applying a DCJ operation, at most 4 entries in each of the two tables have to be updated. Eventually, all cycles are 2-cycles and all paths are 1-paths and a perfectly duplicated genome can be obtained as follows: By ignoring the assignment of the paralogs, each 2-cycle consists of two adjacencies of the form $\{u^x, v^y\}$, where $x, y \in \{t, h\}$, and each 1-path connects two telomeres of the form u^x, where $x \in \{t, h\}$. Thus, a perfectly duplicated genome can be reconstructed by replacing each 2-cycle by the adjacency $\{u^x, v^y\}$ and each 1-path by the telomere u^x. So, the overall running time of the algorithm for reconstructing a perfectly duplicated genome is linear.

5.4 A Note on the Warren-Sankoff Formula

In [128], Warren and Sankoff consider a more general genome model where the ancestral genome has to be neither circular-perfectly duplicated, nor linear-perfectly duplicated. Therefore, we will use the notion *general-perfectly duplicated* in order to distinguish it from our definition of a perfectly duplicated genome. More precisely, a genome is general-perfectly duplicated if and only if for each adjacency $\{u, v\}$ in A, also $\{\overline{u}, \overline{v}\}$ is in A, and for each telomere $\{u\}$ in A, also $\{\overline{u}\}$ is in A. Observe that, in contrast to our definition, a general-perfectly duplicated genome can have adjacencies of the type $\{u, \overline{u}\}$. For example, the following genome is general-perfectly duplicated, but not perfectly duplicated:

Algorithm 7 Reconstruction of a perfectly duplicated genome

```
1: Construct NG(A), the natural graph of genome A
2: while there exists a k-path with k > 1 do
3:    Create a 2-cycle (and a (k − 2)-path if k > 2)
4: end while
   (* all remaining paths have length 1 *)
5: while there exists a k-cycle with k > 2 do
6:    Create a 2-cycle and a (k − 2)-cycle
7: end while
   (* all remaining cycles have length 1 or 2 *)
8: while there exists a 1-cycle do
9:    if there exists another 1-cycle then
10:      Create a 2-cycle
11:   else
12:      Create a 1-path
13:   end if
14: end while
```

Now, let us denote by $d_{DCJ}^{general}(A, B)$ the minimum number of DCJ operations needed to transform a rearranged duplicated genome A into a general-perfectly duplicated genome B. By showing an upper and a lower bound, Warren and Sankoff finally claim that

$$d_{DCJ}^{general}(A, B) = N - (|EC| + |OP| + \left\lfloor \frac{|OC|}{2} \right\rfloor).$$

As a counterexample, consider a genome with just one gene a. Assume that the genome has two linear chromosomes, each consisting of one paralog a_1 and a_2. Note that the genome is general-perfectly duplicated and the natural graph has two paths of length one. Thus, the distance should be zero, but the above formula gives us

$$N - |OP| = 1 - 2 = -1.$$

Even though their distance formula is formulated in terms also defined in the natural graph, Warren and Sankoff follow a different approach. Therefore, instead of using their techniques, we will present in the following a correction of their result by modifying our algorithm.
As mentioned above, the difference is that a general-perfectly duplicated genome may have adjacencies that correspond to 1-cycles in the natural graph. Thus, we have:

Lemma 5.11 *A genome A is general-perfectly duplicated if and only if all cycles in $NG(A)$ are 2-cycles or 1-cycles, and all paths in $NG(A)$ are 1-paths, i. e., $N = |EC| + (|OP| + |OC|)/2$.*

As a consequence of this lemma, we do not have to apply DCJ operations in order to get rid of 1-cycles in the natural graph as in our genome model. Since there are at most $\lceil |OC|/2 \rceil$ such DCJ operations, one can easily show that

$$d_{DCJ}(A,B) = d_{DCJ}^{general}(A,B) + \left\lceil \frac{|OC|}{2} \right\rceil .$$

By this fundamental relation, one can derive the distance formula for the general DCJ model studied by Warren and Sankoff in [128]:

Theorem 5.12 *Let A be a rearranged duplicated genome with N genes each present in two copies, then the minimal distance between A and any perfectly duplicated genome B equals*

$$d_{DCJ}^{general}(A,B) = N - (|EC| + \frac{|OP| + |OC|}{2})$$

where $|EC|$ is the number of even cycles, $|OC|$ the number of odd cycles and $|OP|$ the number of odd paths in the natural graph $NG(A)$.

It should be mentioned that an optimal algorithm for reconstructing a general-perfectly duplicated genome is obtained by just removing the last while-loop in our Algorithm 7.

5.5 Summary and Historical Notes

In this chapter, we have presented a new genome model with coexisting circular and linear chromosomes that unifies earlier genome models for linear genomes and for circular genomes. Under this model, we solve the Genome Halving Problem for the DCJ distance. Surprisingly, this can be done by working directly on the natural graph — all other graphs that are typically used in this context are bypassed.
El-Mabrouk and Sankoff [50] solved the Genome Halving Problem under the HP distance. Their algorithm for the reconstruction of doubled genomes is from of being trivial and is the final result of a whole series of papers [49, 48, 46]. In addition to the well-known *breakpoint graph*, they introduce further graphs, called *natural graph* and *signature graph*. Later, Alekseyev and Pevzner gave an alternative approach based on the notion of *contracted breakpoint graph* [3] and corrected in [4] an error in the El-Mabrouk-Sankoff analysis.
Very recently, Warren and Sankoff [128] studied the Genome Halving Problem under the more general DCJ model. This generalization simplifies the problem because some of the complicated components of the breakpoint graph, such as *hurdles* and *knots*, can be ignored. Unfortunately, their solution still relies on the complex concepts introduced by El-Mabrouk and Sankoff. Indeed, as we have seen in this chapter, our approach is also able to describe alternative genome models such as the one presented by Warren and Sankoff. Thus, our genome model represents a firm starting point for further studies and variants of the Genome Halving Problem.
One direction is to consider duplicated genomes with a higher multiplicity of each gene. This extension yields a natural graph with vertices of degree greater than two. It would have to be studied whether the DCJ operation can also be used on such a graph and how to partition the connected components.

Chapter 6

Conclusion and Future Directions

In this thesis, we presented a novel approach for genome comparison and studied the double cut and join (DCJ) operation on the most general genome structure that allows for both circular and linear chromosomes. With this genome model, the DCJ operation elegantly accounts for all classical rearrangement operations such as translocations, fusions, fissions, inversions and block interchanges. Our main result is an elementary and formal presentation of the DCJ distance problem. The basic tools for this representation are graphs that are unions of paths and cycles. Surprisingly, this type of graphs can be used for representing genomes, as well as for modeling genome rearrangements. We have introduced a very simple data structure, the adjacency graph, which is symmetric with respect to the two genomes under study and is closely related to the visual picture of the genomes themselves. This graph simplifies the theory and distance computation considerably and yields an efficient algorithm for suggesting rearrangement scenarios.

In a unifying way, the model of DCJ operations yields a global picture of genome rearrangements that includes existing models such as the classical Hannenhalli-Pevzner (HP) model that is restricted to linear chromosomes. The three rearrangement models considered in the HP theory (inversions-only, translocations-only and a combination of inversions and translocations) can be integrated in the more general DCJ model. Our main contribution here is a simpler formula for the HP genomic distance problem. Traditionally used concepts that were sometimes hard to access, like *weak-fortresses-of-semi-real-knots*, are bypassed. Indeed, the original HP distance formula involves more than half a dozen parameters, *"making it very hard to explain an intuition behind it"* [62]. On the other hand, our method is based on a simple tree structure that captures all the delicate features of this problem in a unifying way and allows for simple and efficient algorithms for distance computation. Our approach simplifies the classical HP results, both on the combinatorial and on the algorithmic level.

Another application of the DCJ operation is the Genome Halving Problem where the ancestral genome may contain both linear and circular chromosomes. With our genome model, constraints required for genomes with only linear chromosomes, as well as the ones for genomes with only circular chromosomes are taken into account. This yields a new proof and a simple algorithm for reconstructing an ancestral genome. Moreover, our results correct an error in the analysis of Warren and Sankoff.

The main focus of this thesis is the combinatorial and algorithmic study of the DCJ model as a framework for distance computation between two genomes. The presented approach is powerful and a firm starting point to explore further rearrangement problems in comparative genomics.

One direction to increase the applicability of a genome model would be to include *centromeres* that are special regions in a chromosome. A rearrangement operation that preserves a centromere in each chromosome is more biologically realistic since a chromosome that lacks a centromere is likely to be lost during subsequent cell division. Unlike genes, different centromeres are not distinguishable and only their position on the chromosome is known. This additional level of structure excludes some translocations, namely those that result in one chromosome with two centromeres and one with none. For the translocations-only model, Ozery-Flato and Shamir presented a solution that takes centromeres into account [94]. To extend their model by fusions and fissions, one should also allow for centromere creation and disappearance.

Beside extending the genome model, one can also consider a more general set of rearrangement operations, called *multi-break* rearrangements [4]. By this generalization, a DCJ operation is equivalent to a 2-break operation and a transposition can be modelled by a 3-break operation instead of two DCJ operations as in our model. Therefore, the results of [3] can be extended to genomes with linear and circular chromosomes that are studied in this work.

Another extension towards more realistic genome comparison would be to give different weights to different rearrangement operations. This motivates the *weighted rearrangement problem* that is given as: find a sorting sequence such that the sum of the weights of the operations in the sequence is minimal. One should note that a shortest sequence is not necessarily optimal. When operations are combined, the weighted rearrangement problem brings up the question how to assess the relative contribution of operations. First results are due to Blanchette *et al.* [29] and further investigations [7, 101] lead to approximation algorithms. It is believed that transpositions should cost twice as much as other operations, which is the case for the DCJ distance. But so far this has not been proven rigorously, despite some efforts [52]. In the context of the DCJ model, it would be an interesting question how variable weights should be assigned to the different operations.

For the inversions-only model, an alternative approach for assigning weights is motivated by a cost model in which the lengths of the reversed segments play a role [100, 12]. The additive inversion cost was also studied by Ajana *et al.* [2] who developed a method that allows to choose one or more solutions based on different criteria. As it turns out, their branch and bound algorithm for sorting by weighted inversions was useful for testing certain inversion hypotheses.

Usually, algorithms for computing sorting scenarios propose only one solution, and different ways around this limitation have been suggested in the context of sorting by inversions only. However, the enumeration of all sorting inversions [114], with no criteria to discriminate among them, is not helpful, since Bérard *et al.* [13] showed that the number of parsimonious sequences can be exponential. They suggested to group the solutions into equivalence classes and, following this approach, Braga *et al.* [58] developed an algorithm that gives one *representative* for each class of solutions and counts

the number of solutions in each class. Another approach to handle the huge amount of inversion sorting scenarios is to take into account structural constraints based on the notion of *common intervals* [122, 67]. Common intervals are sets of genes that occur as single contiguous blocks in two or more genomes. A sorting scenario is called *perfect* if it does not break any common interval. It was shown that the computation of parsimonious, perfect scenarios is NP-complete [55], but in some cases, an efficient computation is feasible [13]. Perfect scenarios based on common intervals are closely related to *strong interval trees* that can be represented by *PQ-trees* due to the relationship between common intervals and the modular decomposition of permutations graphs [13]. Another application of PQ-trees in comparative genomics is the representation and detection of gene clusters [77]. In fact, the tree that is used in our distance computation is similar to the PQ-tree introduced by Booth and Lueker [31].
Once the genomic distance between two genomes is computed, there is a need for the statistical validation of the results and the underlying assumptions. This led to studies of the probability distribution of the genomic distance under the hypothesis of random gene order. The statistical properties of random genomes have been worked out for one or more circular chromosomes [107] and for random genomes containing the same number of linear chromosomes [133]. Moreover, it was shown that the calculation of the cycle expectations can be simplified considerably, using the DCJ distance formula and collapsing all caps of the linear chromosomes into one single source [132].
The combinatorial approach of computing genomic distances has the advantage that it represents a clearly defined minimization problem. On the other hand, it underestimates the true evolutionary distance between two genomes. One attempt to more accurate evolutionary distances is based on *distance corrections*. Moret *et al.* [86] developed a formula to correct the underestimate of the inversion distance. This approach was further generalized by including duplications and insertions [117]. Indeed, our assumptions that no gene is duplicated and that both genomes have exactly the same gene content is clearly unrealistic. Thus, the most natural extension of our model would be to involve either gene duplications [47] or missing information about the actual order of genes in a genome [135].

The fundamental problem that must be solved first, before one can tackle higher level problems, is the distance computation between two genomes. In the last decade, several solutions have been suggested, but their benefit was sometimes restricted by rather complex mathematics, or erroneous results. The general DCJ model presented in this work contributes to the field of comparative genomics by a unifying theory of genome rearrangement problems and suggests a promising avenue for further exploration. The presented concepts and algorithms will eventually allow us to better understand the different genome rearrangement effects found in real genomic data.

Bibliography

[1] S. Ahn and S. D. Tanksley. Comparative linkage maps of rice and maize genomes. *Proceedings of the National Academy of Sciences*, 90(17):7980–7984, 1993.

[2] Y. Ajana, J.-F. Lefebvre, E. R. M. Tillier, and N. El-Mabrouk. Exploring the set of all minimal sequences of reversals – an application to test the replication-directed reversal hypothesis. In *Proceedings of the Second International Workshop on Algorithms in Bioinformatics (WABI 2002)*, volume 2452 of *LNCS*, pages 300–315. Springer Verlag, 2002.

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

[4] M. 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 (SODA 2007)*, pages 665–679, 2007.

[5] M. A. Alekseyev. Multi-break rearrangements: from circular to linear genomes. In *Proceedings of the Fifth RECOMB Satellite Workshop on Comparative Genomics*, volume 4751 of *LNBI*, pages 1–15. Springer Verlag, 2007.

[6] D. A. Bader, B. M. E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.

[7] M. Bader and E. Ohlebusch. Sorting by weighted reversals, transpositions, and inverted transpositions. In *Proceedings of the Tenth Annual Conference on Computational Molecular Biology (RECOMB 2006)*, volume 3909 of *LNCS*, pages 30–35, 2006.

[8] V. Bafna and P. A. Pevzner. Sorting by reversals: Genome rearrangements in plant organelles and evolutionary history of X chromosome. *Molecular Biology and Evolution*, 12(2):239–246, 1995.

[9] V. Bafna and P. A. Pevzner. Genome rearrangements and sorting by reversals. *SIAM Journal on Computing*, 25(2):272–289, 1996.

[10] V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM Journal on Discrete Mathematics*, 11(2):224–240, 1998.

[11] E. Belda, A. Moya, and F. J. Silva. Genome rearrangement distances and gene order phylogeny in γ-proteobacteria. *Molecular Biology and Evolution*, 22(6):1456–1467, 2005.

[12] M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting with length-weighted reversals. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004)*, pages 919–928, 2004.

[13] S. Bérard, A. Bergeron, C. Chauve, and C. Paul. Perfect sorting by reversals is not always difficult. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI 2005)*, volume 3692 of *LNCS*, pages 228–238. Springer, 2005.

[14] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, volume 2089 of *LNCS*, pages 106–117. Springer Verlag, 2001.

[15] A. Bergeron, C. Chauve, F. de Montgolfier, and M. Raffinot. Computing common intervals of k permutations, with applications to modular decomposition of graphs. In *Proceedings of the 13th Annual European Symposium Algorithms (ESA 05)*, volume 3669 of *LNCS*, pages 779–790. Springer, 2005.

[16] A. Bergeron, S. Heber, and J. Stoye. Common intervals and sorting by reversals: A marriage of necessity. *Bioinformatics*, 18(Suppl. 2):S54–S63, 2002. (Proceedings of ECCB 2002).

[17] A. Bergeron, J. Mixtacki, and J. Stoye. A new algorithm to compute the HP distance via the double cut and join distance in linear time. Submitted.

[18] A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, volume 3109 of *LNCS*, pages 388–399. Springer Verlag, 2004.

[19] A. Bergeron, J. Mixtacki, and J. Stoye. The inversion distance problem. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, chapter 10, pages 262–290. Oxford University Press, Oxford, UK, 2005.

[20] A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005)*, volume 3500 of *LNCS*, pages 615–629. Springer, 2005.

[21] A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *Journal of Computational Biology*, 13(2):567–578, 2006.

[22] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *Proceedings of the 6th International Workshop on Algorithms in Bioinformatics (WABI 2006)*, volume 4175 of *LNBI*, pages 163–173. Springer Verlag, 2006.

[23] A. Bergeron, J. Mixtacki, and J. Stoye. HP distance via double cut and join distance. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, volume 5029 of *LNCS*, pages 56–68. Springer Verlag, 2008.

[24] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proceedings of the 13th Annual International Conference on Computing and Combinatorics (COCOON 2003)*, volume 2697 of *LNCS*, pages 68–79. Springer Verlag, 2003.

[25] P. Berman and S. Hannenhalli. Fast sorting by reversal. In *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching (CPM 1996)*, volume 1075 of *LNCS*, pages 168–185. Springer Verlag, 1996.

[26] M. Bernt, D. Merkle, and M. Middendorf. Genome rearrangement based on reversals that preserve conserved intervals. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(3):275–288, 2006.

[27] M. Blanchette. Evolutionary puzzles: An introduction to genome rearrangement. In *Proceedings of the 9th International Conference on Conceptual Structures (ICCS 2001)*, volume 2074 of *LNCS*, pages 1003–1011. Springer Verlag, 2001.

[28] M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. In *Proceedings of Genome Informatics Workshop (GIW 1997)*, pages 25–34, 1997.

[29] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172:11–17, 1996.

[30] G. Blin, C. Chauve, and G. Fertin. Genes order and phylogenetic reconstruction: Application to γ-proteobacteria. In *Proceedings of the Third RECOMB Satellite Workshop on Comparative Genomics*, volume 3678 of *LNBI*, pages 11–20. Springer Verlag, 2005.

[31] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[32] G. Bourque and P. A. Pevzner. Genome-scale evolution: Reconstructing gene orders in the ancestral species. *Genome Research*, 12(1):26–36, 2002.

[33] G. Bourque, P. A. Pevzner, and G. Tesler. Reconstructing the genomic architecture of ancestral mammals: lessons from human, mouse and rat genomes. *Genome Research*, 14(4):507–516, 2004.

[34] D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J. H. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics*, pages 207–211. Kluwer, 2000.

[35] A. Caprara. Sorting by reversals is difficult. In *Proceedings of the First Conference on Computational Molecular Biology (RECOMB 1997)*, pages 75–83. ACM Press, 1997.

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

[37] S. Casjens, N. Palmer, R. van Vugt, W. M. Huang, B. Stevenson, P. Rosa, R. Lathigra, G. Sutton, J. Peterson, R. J. Dodson, D. Haft, E. Hickey, M. Gwinn, O. White, and C. M. Fraser. A bacterial genome in flux: The twelve linear and

nine circular extrachromosomal DNAs in an infectious isolate of the Lyme disease spirochete *Borrelia burgdorferi*. *Molecular Microbiology*, 35(3):490–516, 2000.

[38] L. Cavalli-Sforza and A. Edwards. Phylogenetic analysis, models and estimation procedures. *Evolution*, 32:550–570, 1967.

[39] D. A. Christie. Sorting permutations by block-interchanges. *Information Processing Letters*, 60(4):165–169, 1996.

[40] D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14(2):193–206, 2001.

[41] P. Dehal and J. L. Boore. Two rounds of whole genome duplication in the ancestral vertebrate. *PLoS Biology*, 3(10):e314, 2003.

[42] T. Dobzhansky and A. H. Sturtevant. Inversions in the chromosomes of *Drosophila pseudoobscura*. *Genetics*, 23:28–64, 1938.

[43] J. V. Earnest-DeYoung, E. Lerat, and B. M. E. Moret. Reversing gene erosion - reconstructing ancestral bacterial genomes from gene-content and order data. In *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics (WABI 2004)*, LNBI, pages 1–13, 2004.

[44] J. Ehrlich, D. Sankoff, and J. H. Nadeau. Synteny conservation and chromosome rearrangements during mammalian evoltion. *Genetics*, 147:289–296, 1997.

[45] N. El-Mabrouk. Genome rearrangement by reversals and insertions/deletions of contiguous segments. In *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM 2000)*, volume 1848 of *LNCS*, pages 222–234, 2000.

[46] N. El-Mabrouk. Reconstructing an ancestral genome using minimum segments duplications and reversals. *Journal of Computer and System Science*, 65(3), 2002.

[47] N. El-Mabrouk. Genome rearrangements with gene families. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, pages 291–320. Oxford University Press, 2005.

[48] N. El-Mabrouk, D. Bryant, and D. Sankoff. Reconstructing the pre-doubling genome. In *Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology (RECOMB 1999)*, pages 154–163, 1999.

[49] N. El-Mabrouk, J. Nadeau, and D. Sankoff. Genome halving. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM 1998)*, volume 1448 of *LNCS*, pages 235–250, 1998.

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

[51] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI 2005)*, volume 3692 of *LNCS*, pages 204–215. Springer, 2005.

[52] N. Eriksen. $(1 + \epsilon)$-approximation of sorting by reversals and transpositions. *Journal Theoretical Computer Science*, 289:517–529, 2002. (Proceedings of WABI 2001).

[53] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17(6):368–376, 1981.

[54] J. Feng and D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms*, 3(3):25, 2007.

[55] M. Figeac and J.-S. Varre. Sorting by reversals with common intervals. In *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics (WABI 2004)*, volume 3240 of *LNCS*, pages 26–37. Springer Verlag, 2004.

[56] W. M. Fitch. Homology: a personal view on some of the problems. *Trends in Genetics*, 16(5):227–231, 2000.

[57] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[58] Z. Fu, X. Chen, V. Vacic, P. Nan, Y. Zhong, and T. Jiang. The solution space of sorting by reversals. In *Proceedings of the Third International Symposium on Bioinformatics Research and Applications (ISBRA 2007)*, volume 3909 of *LNCS*, pages 293–304. Springer Verlag, 2007.

[59] R. Guyot and B. Keller. Ancestral genome duplication in rice. *Genome*, 47:610–614, 2004.

[60] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. *Discrete Applied Mathematics*, 71(1-3):137–151, 1996.

[61] S. Hannenhalli, C. Chappey, E. Koonin, and P. A. Pevzner. Genome sequence comparison and scenarios for gene rearrangements: A test case. *Genomics*, 30:299–311, 1995.

[62] S. Hannenhalli and P. A. Pevzner. Transforming men into mice (polynomial algorithm for genomic distance problem). In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS 1995)*, pages 581–592. IEEE Comuter Society Press, 1995.

[63] S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[64] T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204(2):275–290, 2006.

[65] T. Hartman and R. Sharan. A 1.5-approximation algorithm for sorting by transpositions and transreversals. *Journal of Computer and System Sciences*, 70(3):300–320, 2005. (Proceedings of WABI 04).

[66] T. Hartmann. A simpler 1.5− approximation algorithm for sorting by transpositions. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, LNCS, pages 156–169. Springer Verlag, 2003.

[67] S. Heber and J. Stoye. Finding all common intervals of k permutations. In *Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (CPM 2001)*, volume 2089 of *LNCS*, pages 207–218. Springer Verlag, 2001.

[68] E. A. Housworth and J. Postlethwait. Measures of synteny conservation between species pairs. *Genetics*, 162:441–448, 2002.

[69] International Human Genome Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[70] G. Jean and M. Nikolski. Genome rearrangements: a correct algorithm for optimal capping. *Information Processing Letters*, 104:14–20, 2007.

[71] H. Kaplan, R. Shamir, and R. E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM Journal on Computing*, 29(3):880–892, 1999.

[72] J. Kececioglu and D. Sankoff. Exact and approximation algorithms for the inversion distance between two chromosomes. In *Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (CPM 1993)*, volume 684 of *LNCS*, pages 87–105, 1993.

[73] J. D. Kececioglu and R. Ravi. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. In *Proceedings of the Sixth ACM-SIAM Symposium on Discrete Algorithm*, pages 604–613. Society of Industrial and Applied Mathematics, 1995.

[74] M. Kellis, B. W. Birren, and E. S. Lander. Proof and evolutionary analysis of ancient genome duplication in the yeast *Saccharomyces cerevisiae*. *Nature*, 428(6983):617–624, 2004.

[75] M. Kothari and B. M. E. Moret. An experimental evaluation of inversion- and transposition-based genomic distances through simulations. In *Proceedings of IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology (CIBCB 2007)*, pages 151–158, 2007.

[76] A. Labarre. A new tight upper bound on the transposition distance. In *Proceedings of the 5th Workshop on Algorithms in Bioinformatics (WABI 2005)*, volume 3692 of *LNCS*, pages 216–227. Springer, 2005.

[77] G. M. Landau, L. Parida, and O. Weimann. Using PQ trees for comparative genomics. In *Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM 2005)*, volume 3537 of *LNCS*, pages 128–143. Springer, 2005.

[78] E. Lerat, V. Daubin, and N. A. Moran. From gene tree to organismal phylogeny in prokaryotes: the case of γ-proteobacteria. *PLoS Biology*, 1(1):101–109, 2003.

[79] G. Li, X. Qi, X. Wang, and B. Zhu. A linear-time algorithm for computing translocation distance between signed genomes. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, volume 3109 of *LNCS*, pages 323–332. Springer Verlag, 2004.

[80] Y. C. Lin, C. L. Lu, H.-Y. Chang, and C. Y. Tang. An efficient algorithm for sorting by block-interchanges and its application to the evolution of vibrio species. *Journal of Computational Biology*, 12(1):102–112, 2005.

[81] L. Lu, Y. L. Huang, T. C. Wang, and H.-T. Chiu. Analysis of circular genome rearrangement by fusions, fissions and block-interchanges. *BMC Bioinformatics*, 7(295), 2006.

[82] M. Marron, K. Swenson, and B. M. E. Moret. Genomic distances under deletions and insertions. *Theoretical Computer Science*, 325:347–360, 2004. (Proceedings of COCOON 2003).

[83] M. Martin. SBBI - sorting by block-interchanges. `http://bibiserv.techfak.uni-bielefeld.de/sbbi/`, 2007.

[84] J. Mixtacki. Genome halving under DCJ revisited. In *Proceedings of the 14th Annual International Conference on Computing and Combinatorics (COCOON 2008)*, volume 5092 of *LNCS*, pages 276–286. Springer Verlag, 2008.

[85] 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 *Proceedings of the Second International Workshop on Algorithms in Bioinformatics (WABI 2002)*, volume 2452 of *LNCS*, pages 521–536. Springer Verlag, 2002.

[86] B. M. E. Moret, J. Tang, L.-S. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *Journal of Computer and System Science*, 65(3):508–525, 2002.

[87] B. M. E. Moret, L. Wang, T. Warnow, and S. Wyman. New approaches for reconstructing phylogenies from gene-order data. *Bioinformatics*, 17:S165–S173, 2001. (Proceedings 9th International Conference on Intelligent Systems for Molecular Biology, ISMB 2001).

[88] J. H. Nadeau and D. Sankoff. The lengths of undiscovered conserved segments in comparative maps. *Mammalian Genome*, 9:491–495, 1998.

[89] J. H. Nadeau and B. A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences USA*, 81:814–818, 1984.

[90] S. Ohno. Ancient linkage group and frozen accidents. *Nature*, 244:259–262, 1973.

[91] M. Ozery-Flato and R. Shamir. Two notes on genome rearrangements. *Journal of Bioinformatics and Computational Biology*, 1(1):71–94, 2003.

[92] M. Ozery-Flato and R. Shamir. An $\mathcal{O}(n^{3/2}\sqrt{\log(n)})$ algorithm for sorting by reciprocal translocations. In *Proceedings of the 17th Annual Symposium on Combinatorial Pattern Matching (CPM 2006)*, volume 4009 of *LNCS*, pages 258–269. Springer Verlag, 2006.

[93] M. Ozery-Flato and R. Shamir. Sorting by translocations via reversals theory. In *Proceedings of RECOMB 2006 International Workshop on Comparative Genomics, (RCG 2006)*, volume 4205 of *LNCS*, pages 87–98. Springer, 2006.

[94] M. Ozery-Flato and R. Shamir. Rearrangements in genomes with centromeres part I: Translocations. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2007)*, volume 4453 of *LNCS*, pages 339–353. Springer Verlag, 2007.

[95] J. D. Palmer and L. A. Herbon. Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. *Journal of Molecular Evolution*, 28:87–97, 1988.

[96] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. Technical report, Elec. Colloq. on Comput. Complexity, Report 71, 1998.

[97] P. A. Pevzner and G. Tesler. Genome rearrangements in mammalian evolution: Lessons from human and mouse genomes. *Genome Research*, 13:37–45, 2003.

[98] P. A. Pevzner and G. Tesler. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of National Academy of Sciences*, 100(13):7672–7677, 2003.

[99] P. A. Pevzner and G. Tesler. Transforming men into mice: the Nadeau-Taylor chromosomal breakage model revisited. In *Proceedings of the Seventh Annual Conference on Computational Molecular Biology (RECOMB 2003)*, pages 247–256. ACM Press, 2003.

[100] R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:103–111, 2002.

[101] A. Rahman, S. Shatabda, and M. Hasan. An approximation algorithm for sorting by reversals and transpositions. In *Proceedings of Workshop on Algorithms and Computation (WALCOM 2007)*, pages 97–108, 2007.

[102] N. Saitau and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

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

[104] D. Sankoff. The signal in the genomes. *PLoS Computational Biology*, 2(4):e35, 2006.

[105] D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. In *Proceedings of Third Annual International Conference on Computing and Combinatorics (COCOON 1997)*, pages 251–264, 1997.

[106] D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570, 1998.

[107] D. Sankoff and L. Haque. The distribution of genomic distance between random genomes. *Journal of Computational Biology*, 13:1005–1012, 2006.

[108] D. Sankoff, G. Leduc, N. Antoine, B. Paquin, B. Lang, and R. Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences USA*, 89:6575–6579, 1992.

[109] D. Sankoff and M. Mazowita. Estimators of translocations and inversions in comparative maps. In *Proceedings of the Second RECOMB Satellite Workshop on Comparative Genomics (RCG 2004)*, volume 3388 of *LNCS*, pages 109–122. Springer, 2004.

[110] D. Sankoff and M. Mazowita. Stability of rearrangement measures in the comparison of genome sequences. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology, (RECOMB 2005)*, volume 3500 of *LNCS*, pages 603–614. Springer, 2005.

[111] D. Sankoff, G. Sundaram, and J. D. Kececioglu. Steiner points in the space of genome rearrangements. *International Journal of Foundations of Computer Science*, 7(1):1–9, 1996.

[112] D. Sankoff and P. Trinh. Chromosomal breakpoint re-use in the inference of genome sequence rearrangement. In *Proceedings of the Eighth Annual Conference on Computational Molecular Biology (RECOMB 2004)*, pages 30–35. ACM Press, 2004.

[113] D. Sankoff, C. Zheng, and A. Lenert. Reversals of fortune. In *Proceedings of RECOMB 2005 International Workshop on Comparative Genomics (RCG 2005)*, volume 3678 of *LNCS*, pages 131–141. Springer, 2005.

[114] A. Siepel. An algorithm to enumerate all sorting reversals. In *Proceedings of the Sixth Annual International Conference on Research in Computational Molecular Biology (RECOMB 2002)*, pages 281–290. ACM Press, 2002.

[115] A. C. Siepel and B. M. E. Moret. Finding an optimal inversion median: Experimental results. In *Proceedings of the First International Workshop on Algorithms in Bioinformatics (WABI 2001)*, volume 2149 of *LNCS*, pages 189–203, 2001.

[116] A. H. Sturtevant. A crossover reducer in *Drosophila melanogaster* due to inversion of a section of the third chromosome. *Biologisches Zentralblatt*, 46(12):697–702, 1926.

[117] K. M. Swenson, M. Marron, J. V. Earnest-DeYoung, and B. M. E. Moret. Approximating the true evolutionary distance between two genomes. In *Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments and the Second Workshop on Analytic Algorithmics and Combinatorics (ALENEX / ANALCO 2005)*, pages 121–129, 2005.

[118] E. Tannier, A. Bergeron, and M.-F. Sagot. Advances in sorting by reversals. *Discrete Applied Mathematics*, 155(6-7):881–888, 2007.

[119] E. Tannier and M.-F. Sagot. Sorting by reversals in subquadratic time. In *Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, volume 3109 of *LNCS*, pages 1–13. Springer Verlag, 2004.

[120] G. Tesler. Efficient algorithms for multichromosomal genome rearrangements. *Journal of Computer and System Sciences*, 65(3):587–609, 2002.

[121] G. Tesler. GRIMM: Genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

[122] T. Uno and M. Yagiura. Fast algorithms to enumerate all common intervals of two permutations. *Algorithmica*, 26(2):290–309, 2000.

[123] J. C. Venter *et al.* The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[124] J.-N. Volff and J. Altenbuchner. A new beginning with new ends: Linearisation of circular chromosomes during bacterial evolution. *FEMS Microbiology Letters*, 186:143–150, 2000.

[125] M. E. M. T. Walter, L. R. A. F. Curado, and A. G. Oliveira. Working on the problem of sorting by transpositions on genome rearrangements. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, LNCS, pages 372–383. Springer Verlag, 2003.

[126] L. Wang, D. Zhu, X. Liu, and S. Ma. A $\mathcal{O}(n^2)$ algorithm for sigend translocation. *Journal of Computer and System Sciences*, 70(3):284–299, 2005.

[127] L.-S. Wang and T. Warnow. Distance-based genome rearrangement phylogeny. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, chapter 13, pages 353–383. Oxford University Press, Oxford, UK, 2005.

[128] R. Warren and D. Sankoff. Genome halving with double cut and join. In *Proceedings of APBC 2008*, volume 6 of *Series on Advances in Bioinformatics and Computational Biology*, 2008.

[129] J. D. Watson and F. H. C. Crick. A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[130] G. A. Watterson, W. J. Ewens, and T. E. Hall. The chromosome inversion problem. *Journal of Theoretical Biology*, 99:1–7, 1982.

[131] K. H. Wolfe and D. C. Shields. Molecular evidence for an ancient duplication of the entire yeast genome. *Nature*, 387:708–713, 1997.

[132] W. Xu. The distance between randomly constructed genomes. In *Proceedings of the Fifth Asia Pacific Bioinformatics Conference (APBC 2007)*, pages 227–236, 2007.

[133] W. Xu, C. Zheng, and D. Sankoff. Paths and cycles in breakpoint graphs of random multichromosomal genomes. In *Proceedings of the Fourth RECOMB Satellite Workshop on Comparative Genomics (RCG 2006)*, pages 51–62, 2006.

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

[135] C. Zheng, A. Lenert, and D. Sankoff. Reversal distance for partially ordered genomes. *Bioinformatics*, 21(Suppl. 1):i502–i508, 2005. (Proceedings of ISMB 2005).