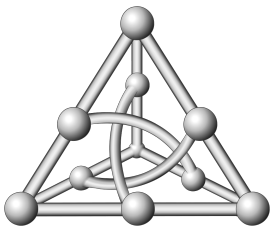
Distance and Similarity Measures in Comparative Genomics

Diego P. Rubert

Doctoral Thesis

Area of Concentration: Theoretical Computer Science

Advisor: Prof. Fábio Henrique Viduani Martinez, Dr.
Co-advisor: Prof. Jens Stoye, Dr.



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
December, 2019

Distance and Similarity Measures in Comparative Genomics

Diego P. Rubert

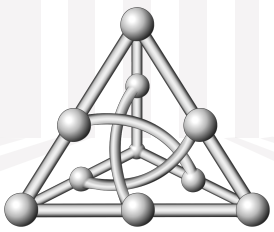
Doctoral Thesis

Area of Concentration: Theoretical Computer Science

Advisor: Prof. Fábio Henrique Viduani Martinez, Dr.

Co-advisor: Prof. Jens Stoye, Dr.

Submitted in partial fulfilment of the requirements for
the degree of Doctor in Computer Science



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
December, 2019

Abstract

Comparative genomics is a field of biological research in which genomic features, such as DNA sequence, genes, gene order, regulatory sequences or other structural aspects, are evaluated to compare different species. To this end, whole or parts of genomes are compared to find biological similarities and differences as well as evolutionary relationships between organisms. Based on these findings, genome and molecular evolution can be inferred and this may in turn be put in the context of, for instance, phenotypic evolution, phylogenetic evaluation, population genetics or ancestral genome reconstruction.

Genome rearrangements—large-scale mutations responsible for complex changes and structural variations—are subject of extensive studies in comparative genomics. Given two genomes, we are interested in the problems of computing the rearrangement distance between them, i.e., finding the minimum number of rearrangement operations that transform one genome into the other, and the genomic similarity between them, i.e., finding structural similarities given some rearrangement operation. The study of these problems supports the investigation of important questions in a number of other fields, such as molecular biology, genetics, biomedicine and paleogenomics.

Most rearrangements that modify the organization of a genome can be represented by the double-cut-and-join (DCJ) operation. In this work, we propose a linear time approximation algorithm with approximation ratio $O(k)$ for the a restrict case of the DCJ distance problem where, given two unichromosomal genomes, each gene occurs the same amount of times in both genomes and no gene occurs more than k times. The general case of this problem is NP-hard and there already exists an ILP exact algorithm for solving it.

We also study the problem of computing the genomic similarity under the DCJ model in a setting that does not assume genes of the compared genomes are grouped into gene families. This problem is called family-free DCJ similarity and is NP-hard. We show it is also APX-hard, and then propose an exact ILP algorithm and four combinatorial heuristics to solve the problem.

For both problems above, we run computational experiments comparing the proposed algorithms with the exact ILP algorithms. Experiments show that the approximation algorithm and the heuristics are very competitive both in efficiency and in quality of the solutions with respect to the ILP algorithms.

Lastly, we propose a local similarity measure based on DCJ operations. Analogous to local sequence alignment, the local DCJ similarity scores local regions in compared genomes with high levels of structural similarity. Such a local measure is often convenient when comparing highly dissimilar genomes containing some or many conserved regions. We show its usefulness by modifying a popular ancestral genome reconstruction pipeline, performing the ancestral reconstruction for an eudicots dataset, and obtaining improved results compared to those presented in a recent publication using the original pipeline.

Keywords: double-cut-and-join (DCJ), Genome rearrangements, Comparative genomics, Approximation algorithms, Heuristics, Integer linear programming, Local genome rearrangements, Ancestral genome reconstruction

Resumo

A genômica comparativa é um campo de pesquisa da biologia em que características genômicas, como sequência de DNA, genes, ordem de genes, sequências reguladoras ou outros aspectos estruturais, são avaliadas para comparar diferentes espécies. Para tal, todo ou parte dos genomas são comparados para encontrar semelhanças e diferenças biológicas, bem como relações evolutivas entre organismos. Com base nessas descobertas, a evolução molecular e dos genomas pode ser inferida, podendo ser avaliada no contexto de, por exemplo, evolução fenotípica, avaliação filogenética, genética de populações ou reconstrução de genomas ancestrais.

Os rearranjos de genomas — mutações em larga escala responsáveis por mudanças complexas e variações estruturais — são objeto de extensos estudos em genômica comparativa. Dados dois genomas, estamos interessados nos problemas de calcular a distância de rearranjo entre eles, ou seja, encontrar o número mínimo de operações de rearranjo que transformam um genoma em outro, e a similaridade genômica entre eles, ou seja, encontrar semelhanças estruturais estáticas ou semelhanças estruturais dada alguma operação de rearranjo. O estudo desses problemas apoia a investigação de questões importantes em vários outros campos, como biologia molecular, genética, biomedicina e paleogenômica.

A maioria dos rearranjos que modificam a organização de um genoma podem ser representados pela operação *double-cut-and-join* (DCJ). Propomos um algoritmo de aproximação de tempo linear com razão de aproximação $O(k)$ para um caso restrito do problema da distância DCJ, em que, dados dois genomas unicromossomais, cada gene ocorre a mesma quantidade de vezes em ambos genomas e nenhum ocorre mais que k vezes. O caso geral do problema é NP-difícil e já existe um algoritmo PLI exato para resolvê-lo.

Também estudamos o problema de calcular a similaridade genômica sob o modelo DCJ em um cenário que não pressupõe que os genes dos genomas comparados sejam agrupados em famílias de genes. Esse problema é chamado de similaridade DCJ livre de famílias, e é NP-difícil. Mostramos que também é APX-difícil e, em seguida, propomos um algoritmo PLI exato e quatro heurísticas combinatórias para resolver o problema.

Para ambos os problemas, realizamos experimentos computacionais comparando os algoritmos propostos com os algoritmos PLI exatos. Os experimentos mostram que o algoritmo de aproximação e as heurísticas são muito competitivos tanto em eficiência quanto em qualidade das soluções em relação aos algoritmos PLI.

Por fim, propomos uma medida de similaridade local baseada em operações DCJ. De forma análoga ao alinhamento local de sequências, a similaridade DCJ local pontua regiões locais com altos níveis de similaridade estrutural nos genomas comparados. Medidas locais tais como esta são frequentemente convenientes quando se compara genomas altamente diferentes contendo algumas ou muitas regiões conservadas. Mostramos a utilidade desta medida modificando um procedimento (*pipeline*) popular para reconstrução de genomas ancestrais, realizando a reconstrução ancestral para um conjunto de dados de eudicots, e obtendo resultados melhorados em comparação com os apresentados em uma publicação recente usando o *pipeline* original.

Palavras-chave: *double-cut-and-join* (DCJ), Rearranjos de genomas, Genômica comparativa, Algoritmos de aproximação, Heurísticas, Programação linear inteira, Rearranjos locais de genomas, Reconstrução ancestral de genomas

To mom.

"It has documentation."

Dumbfounded reviewer (concerning ANGORA).

Acknowledgements

I would like to thank my advisor Fábio Martinez, who has guided me patiently through this journey for years, contributed with so many ideas during my research, and with whom I shared some failures—because we all must learn to fail. Jens Stoye, for his precious contribution in every work of this thesis, who knows a lot about several things and seems to know a little bit about everything else, and mentored me helpfully during my stay in Bielefeld. Daniel Doerr (*oder Dörr*?), who worked close to me, contributed to my research with numerous insightful ideas, pushed me beyond my limits, and from whom I have learned countless concepts, new working methodologies and useful tools.

I am specially grateful to all my co-authors for allowing me to use the manuscripts we have written as a base for several parts of this thesis. I am also indebted to Michel Theodor Henrichs for providing the sequential implementation of IMP in `C++` and for providing Figure 5.2, and to Jerome Salse for providing the data of the reconstructed eudicot ancestor from [10].

Lastly, the most important acknowledgement, I am grateful to my wife Evelyn, who encouraged me (even unwittingly) countless times, has done her best to provide me through emotional and material support, and faced solitude by moving away from everyone she knows so I could have a better doctoral study. I could not have a better lifemate.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 – PDSE scholarship no. 88881.188040/2018-01, and DAAD funding programme Research Stays for University Academics and Scientists 2018, ID 57378441.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Chromosomes are molecules present in all living organisms and carry their genetic information. The set of all chromosomes of an organism is its genome. A chromosome is composed of a double-stranded molecule called DNA (Fig. 1.1), where each strand is a sequence of nucleotides complementary to nucleotides in the other strand. Chromosomes can be linear (eukaryotes and some prokaryotes) or circular (most prokaryotes). A nucleotide is composed of, besides other molecules, one among four nucleobases: adenine (A), which pairs with thymine (T), and cytosine (C), which pairs with guanine (G). Since strands are complementary, a DNA molecule can be seen as a single sequence over the alphabet {A,C,G,T}.

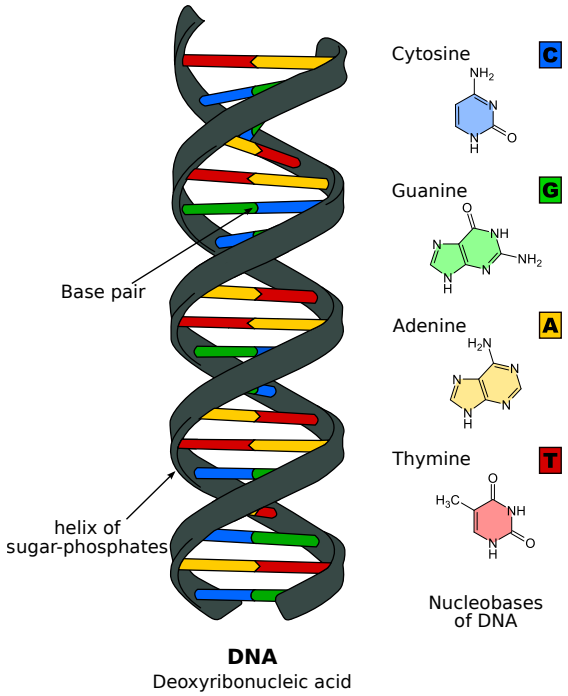


Figure 1.1: **Structure of DNA.** Each base pair is formed by a pair of complementary nucleotides. Source: Wikimedia commons, licensed under CC BY-SA 3.0.

Problems and applications in molecular biology can work with the DNA sequence in low level, i.e., at the level of nucleotides, or in higher, that is, looking into each chromosome as a sequence of blocks, each one formed by a nucleotide sequence of arbitrary length. Such blocks are loosely defined as genomic markers. Naturally, the length of the sequence that composes a marker must be of a reasonable size. A marker formed by a single nucleotide takes us back to the low level view of the chromosome. At the other extreme is a marker comprising the whole chromosome. At the same time, a large proportion of nucleotide sequences play, however, no role in any known function in genomes. For this reason, we usually consider only nucleotide sequences relevant in some aspect. There are different approaches for defining genomic markers, none of them being superior in every aspect, though. One such approach is to take sequences of a minimum length that have been kept minimally conserved through evolution [47]. Another usual approach is to simply take genes, which can be defined as DNA sequences that code the information needed to produce other molecules (e.g. proteins) [2, 135]. Even the formal definition of the term gene is, however, subject to discussion [58,98]. Despite of that, the set of genomic markers of a genome is very often defined directly as the set of its genes. Further, in many cases, the approach used to define genomic markers is simply irrelevant. Therefore, the term gene will be used in this text as a synonym of genomic marker, unless noted otherwise.

The diversity of living organisms is possible mainly due to a process known as DNA replication, where one genome is the basis to construct another similar genome. The inaccuracy of this process is the principle of the molecular evolution. Changes in a DNA molecule may occur by point mutations, at the level of nucleotides, and by large-scale mutations or rearrangements, changing the number of chromosomes and/or the positions and orientations of genes. Examples of such rearrangements are inversions (also called reversals), translocations, fusions, and fissions.

Genome comparison methods by means of large-scale mutations support the investigation of important questions of molecular biology, genetics, biomedicine, and other areas. The research on the field of comparative genomics often involves definitions of distance or similarity measures between genomes, in order to use such measures to obtain phylogenetic trees, predict orthologous genes, or identify propagation of gene functions across different species. A classical problem in comparative genomics is to compute the rearrangement distance, that is, the smallest number of rearrangements required to transform a given genome into another given genome [110].

Some branches of comparative genomics are summarized below.

Detection of conserved structures

The study of gene order in genomes is responsible for detecting similarities or quantifying conserved structures based on close relationships between pairs or groups of genes considering their sequences in chromosomes. The most representative relations of proximity or distance between pairs of genes include breakpoints [26, 112] and adjacencies [3, 33], while relations between groups of genes encompass common intervals [72, 73, 113], approximate common intervals [77, 100], and *max-gap* clusters [13, 71].

Reconstruction of ancestral genomes

A more complete and in-depth view of evolutionary mechanisms, gene functions, and the reconstruction of phylogenetic trees can be obtained through the study

of the conservation of gene orders or rearrangement processes in the context of the phylogeny of involved organisms. The reconstruction of ancestral genomic structures from homologies between species is a problem that has been extensively studied [1, 10, 39, 48, 93, 97, 107, 108, 123]. Fig. 1.2 depicts an overview of an anscestral genome reconstruction.

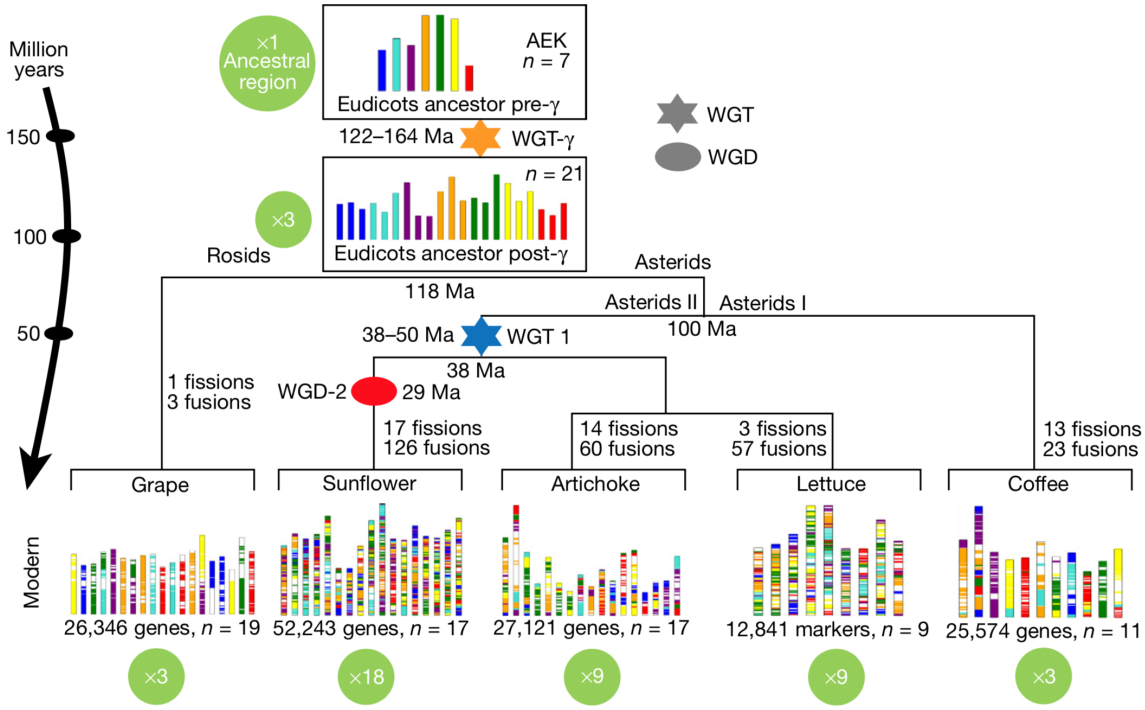


Figure 1.2: **Example of ancestral genome reconstruction overview.** Different colors in modern genomes (bottom) reflect the origin of chromosome segments with respect to the seven ancestral chromosomes from the ancestral eudicot karyotype (AEK) on top. Ellipses and stars show whole genome duplication (WGD) and triplication (WGT) events, respectively. Source: *The sunflower genome provides insights into oil metabolism, flowering and Asterid evolution*, Badouin *et al.* [10], licensed under CC BY 4.0.

Genome rearrangements

In contrast to methods for detecting conserved structures, where only the static properties of the genomes are studied and compared, genome rearrangement is a research area in comparative genomics that attempts to understand the dynamics of structural modifications of genomes over time. Typical operations of genome rearrangements are *reversals* (or *inversions*) of a chromosome segment, *block interchanges* which exchange pairs of segments, *transpositions* which are block interchanges of adjacent segments, *translocation* of genetic material between two chromosomes, and *fusion* and *fission* of chromosomes (see Fig. 1.3). One important rearrangement operation studied in the recent years is the *double-cut-and-join* (DCJ) [134], that consists of cutting a genome in two distinct positions (possibly in two distinct chromosomes) and joining the four resultant open ends in a different way, represents most of large-scale rearrangements that modify genomes (see Fig. 1.3). Additionally, some methods allow *indels* (Fig. 1.3), meaning the *insertion* or *deletion* of chromosome

segments. In this branch, we often want to obtain (i) a distance measure under some rearrangement operation or (ii) a sequence of rearrangement operations that transforms one genome into another (counting the number of operations in that sequence is one way to obtain (i)).

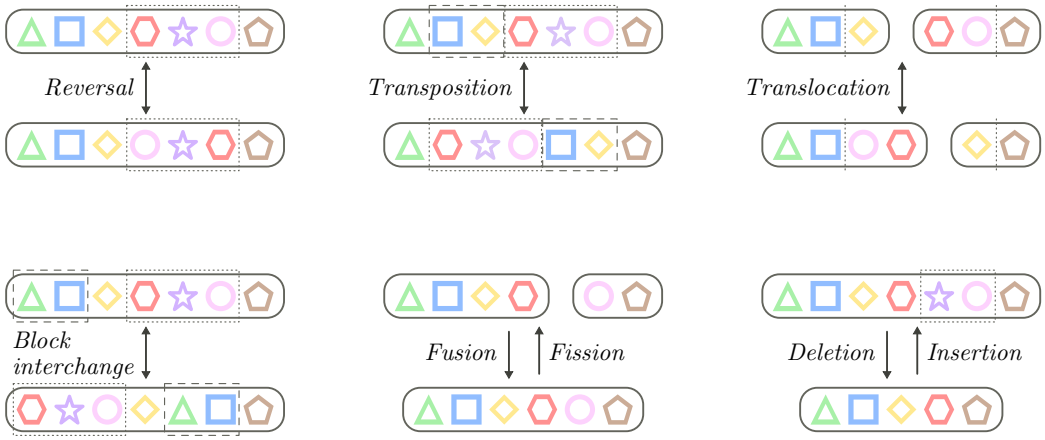


Figure 1.3: **Examples of rearrangement events.** Symbols are genes, solid lines group genes in chromosomes and dashed or dotted lines represent regions or positions affected by a rearrangement event. One DCJ can mimic a reversal, a fusion, a fission, or a translocation, whereas two DCJs are required to model a block interchange or a transposition (details in the Section 2.1).

The purpose of this work is to investigate distance and similarity measures for genomes under rearrangement events and applications of such measures. The most basic scenario in comparing genomes is where each gene occurs only once in each of the genomes. In this scenario, several measures have been studied, many of which can be efficiently computed [11,15,65,134]. However, this model does not match strictly what is found in nature, where several copies of the same gene (or orthologous genes) occur in the same genome due to duplications. When orthologous genes occur, *gene families* are defined, which are gene sets with similar biochemical functions. In order to address the occurrence of multiple genes from the same family in genomes, the *family-based* approaches have been proposed, establishing measures usually hard to compute [3–5, 27, 29, 111, 115].

Although family information can be obtained by accessing public databases or by direct computing, data can be incorrect, and inaccurate families could be providing support to erroneous assumptions of homology between segments [45]. Thus, it is not always possible to classify each gene unambiguously into a single family, and an alternative to the family-based setting was proposed recently [22, 44, 45, 89]. It consists of studying genome rearrangements without prior family assignment, by directly accessing the pairwise similarities between genes of the compared genomes. This approach is said to be *family-free* (FF). Such problems in general are at least as difficult as those based on gene families.

More recently, an intermediate modeling between the family-based and the family-free approaches has been proposed, called *gene connections* [43]. Some problems on this model are proven to be polynomial while others are as difficult as family-based and family-free harder problems.

In the following sections, solutions, hardness, inapproximability, and approximation results are summarized for family-based and family-free measures under the most common rearrangement operations. Measures for unichromosomal genomes are far more common than for multichromosomal genomes, since the simplest operations do not exchange genes between chromosomes (e.g. reversals or transpositions). Therefore, when not specified, we assume unichromosomal genomes. However, considering translocation, fusion, fission, or any rearrangement operation that acts solely over two chromosomes (DCJs not included) only makes sense for and imply multichromosomal genomes. Hence, when these rearrangement operations are involved, we assume multichromosomal genomes. In addition, problems for multichromosomal genomes (general case) are harder than for unichromosomal genomes (restricted case), therefore any solution or approximation for the former is also for the latter and any hardness or inapproximability result for the latter is also for the former.

Lastly, since the usefulness of a new measure proposed in this work is shown by improving a popular ancestral genome reconstruction pipeline, we also provide a succinct overview on this field.

1.1 Some background on family-based measures

In this section we overview family-based measures for most common rearrangement operations. See [55] for a detailed review of other problems and variants. In the family-based setting, some preprocessing is required as a first step before genomes can be compared. The most common method, adopted for about 20 years [110, 111], is to base the analysis on the order of conserved syntenic DNA segments across different genomes and group homologous segments into families.

It is worth mentioning that many references for family-based measures on genomes without duplicate genes concern sorting or measures for permutations, signed or unsigned, sometimes not even in the context of genome rearrangements. However, signed linear (circular) permutations are equivalent to linear (circular) chromosomes without duplicate genes. Besides, distance measures are often obtained by counting the number of operations to "sort" one permutation into another, e.g., the reversal distance is calculated by counting the minimum number of reversals required to transform one unichromosomal genome into another. In addition, when considering only operations that do not change the sign of elements in permutations (orientation of genes), such as transpositions, the sign is irrelevant and the problem is equivalent for signed and unsigned permutations. Similarly, many references for measures with duplicate genes concern sorting or measures for strings, which are equivalent to linear chromosomes with duplicate genes. In this context, the term *rearranging* is often used instead of sorting because we want to transform one string into another.

First, we present older and simpler problems, for which genomes have no duplicate genes. Next, we consider more recent and harder problems, where duplicate genes are present in genomes. In both cases, genomes can be balanced or unbalanced. Two genomes are *balanced* when each family has the same number of genes in each genome, otherwise they are *unbalanced*. The same relation between multichromosomal and unichromosomal genomes exists for unbalanced ("harder", general case) and balanced ("easier", restricted

case) genomes. Therefore, any solution or approximation for the former is also for the latter and any hardness or inapproximability result for the latter is also for the former. Few rearrangement operations under family-based methods have been studied for unbalanced genomes. Henceforward, when not specified, we consider results on family-based measures for balanced genomes.

Genomes without duplicate genes

Without duplicate genes, i.e., with the additional restriction that at most one representative of each family occurs in any genome, several polynomial time algorithms have been proposed to compute genomic distances and similarities. Recall that, when not specified, we assume linear chromosomes.

A *conserved segment* is a maximal common sequence of genes to both genomes. For any pair of consecutive genes, if they belong to the same conserved segment they form an *adjacency*, otherwise they form a *breakpoint*. One of the first distance measures for genomes, defined by Sankoff and Blanchette, the breakpoint distance is trivially calculated for uni- or multichromosomal linear or circular genomes [26, 112, 125]. Unlike most measures, the breakpoint distance is a dissimilarity measure that does not take into account any rearrangement operation. In fact, the breakpoint distance is not really a rearrangement distance, but a simple and one of the first measures of dissimilarity between genomes.

The reversal distance is solvable in polynomial time for linear [11, 66, 124] and circular [90] genomes. This is also true for block interchanges for both linear [34, 53, 76, 86] and circular [76, 86] genomes. While the sign of elements plays an important role for defining more realistic models in genome rearrangement problems, the reversal distance is a classic example where the unsigned version of a problem is much more difficult to solve. In fact, the unsigned reversal distance is NP-hard [30, 31].

Bafna and Pevzner [12] analyzed the problem with respect to transpositions, presenting an 1.5-approximation algorithm, which was outperformed later by means of time [53, 68] and approximation ratio [40, 50]. After 15 years the problem was found to be NP-hard [28]. Hartman and Shamir [68] also found that the problem of sorting circular permutations (unichromosomal circular genomes) by transpositions is equivalent to the problem of sorting linear permutations (unichromosomal linear genomes) by transpositions, hence all algorithms for the latter problem can be used for the former.

The complexity of sorting unichromosomal genomes by reversals and transpositions is unknown. For reversals with weight 1 and transpositions with weight α, the best approximation ratios are 2, 1.5 and $1 + \varepsilon$ for $\alpha = 1$ [61, 85, 129], $1 \leq \alpha \leq 2$ [9], and $\alpha = 2$ for any $\varepsilon > 0$ [51], respectively.

For translocations only, the distance can be computed in polynomial time [14, 64]. Hannenhalli and Pevzner [65] studied the distance problem for genomes involving reversals, fusions, fissions, and translocations, devising a polynomial time algorithm, further corrected or improved [16, 78, 96, 126]. The problems for genomes under fusions, fissions, and transpositions (weighting twice as much as fusions and fissions) [42], circular genomes under fusions, fissions and block interchanges [87], or circular genomes under block interchanges and reversals [91] are also polynomial. The same holds for both uni- or multichromosomal linear or circular genomes under DCJs [15, 134].

The problems presented above consider the default scenario for genome rearrangements without duplicate genes (one gene per family), where the input is two balanced genomes (with the same genes, in different orders). Despite that, some models admit as input unbalanced genomes (some genes appear in only one of them). Therefore, some genes must be inserted or removed so that both genomes end up having the same genes. The reversal distance for unbalanced circular genomes where additional genes are present in only one genome, thus allowing deletions or insertions only, is polynomial [130]. The DCJ distance for unbalanced multichromosomal linear or circular genomes allowing indels can be computed in linear time [24].

Genomes with duplicate genes

When duplicates are allowed, problems become more intricate and many of them are NP-hard. For measures in this context, first we have to consider three standard models for dealing with duplicates (Fig. 1.4):

- The *exemplar model*, in which, for each genome, exactly one gene in each family is selected, and then the seleced pairs (one in each genome) are associated;
- The *full model*, in which as many genes as possible must be associated between genomes (a maximal matching for gene associations between genomes must be defined);
- The *intermediate model*, which is similar to the full model but requires an association of at least one gene in each family in each genome, instead of as many as possible.

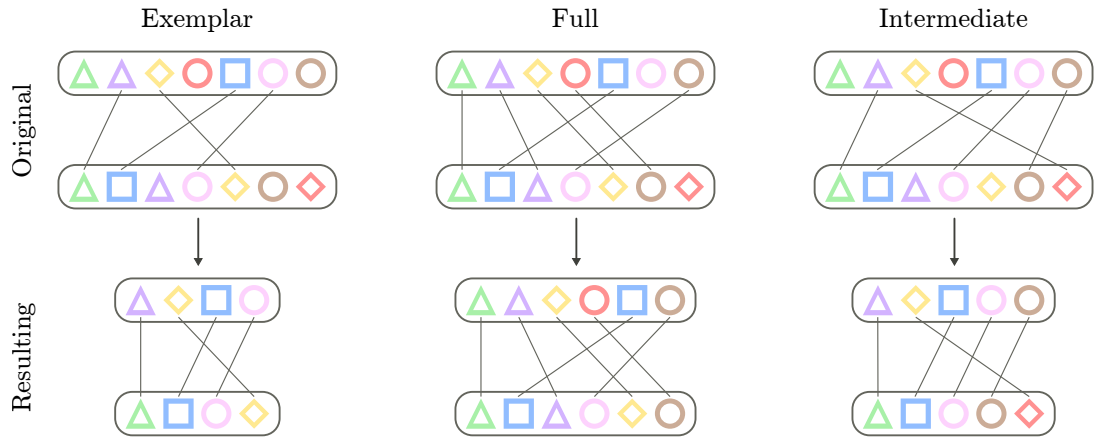


Figure 1.4: **Examples of original and resulting genomes for exemplar, full and intermediate models.** Genes in the same family are represented by the same symbol, genes with multiple copies are distinguished by different colors, and gene associations are indicated by solid lines. The same color in different genomes has no special meaning. The measure between genomes is usually calculated from the resulting genomes.

In all cases, an association of genes is made such that some measure between the two resulting genomes (usually ignoring unassociated genes) is the best possible. Although unassociated genes are usually simply ignored (removed from resulting genomes), some measures model this by indels events. Notice that resulting genomes under the exemplar

model have no duplicate genes. In this work we regard only measures under the full model, i.e., we want to keep as many genes as possible.

Additionally, an important note is that all the three models are identical when one of the two genomes contains no duplicates. Hence, hardness and inapproximability results for problems in this restrict case (no duplicates in one genome) under one model are carried to the same restrict case under all models, and then to the general case (with duplicates in both genomes) for all models.

The breakpoint distance for both linear and circular genomes is NP-hard [3, 21] and APX-hard [4]. Moreover, when the number of genes in the largest family is k, the problem admits 1.1037, 4, and $8k$-approximation when $k = 2$ [60], $k = 3$ [60], and k is free [83], respectively. The complementary problem, the adjacency similarity (or non-breaking similarity) is also NP-hard [3] for both linear and circular genomes and admits approximations with ratios 1.1442 when $k = 2$, $(3 + \varepsilon)$ when $k = 3$ (for any $\varepsilon > 0$), and 4 in the general case for linear and circular genomes [4]. Further, for the general case of unbalanced linear and circular genomes, this problem does not admit a polynomial time approximation for any ratio $n^{1-\varepsilon}$, for $0 \leq \varepsilon \leq 1$ [33].

For signed strings (equivalent to linear unichromosomal genomes), the minimum common string partition (MCSP) problem [32] is closely related to the breakpoint distance problem. The breakpoint distance can be viewed as the minimum number of places where a rearrangement scenario has to break, whereas the number of contiguous segments without breaks is the solution of MCSP. Hence, the number of these segments is exactly the breakpoint distance plus one. On the other hand, the breakpoint distance (or MCSP) is known to differ only by a constant multiplicative factor from the reversal distance [32]. More precisely, for two genomes let b be the breakpoint and let r be the reversal distance between them, then $\lceil \frac{b}{2} \rceil \leq r < b$. Because of this, approximation results for the breakpoint distance can often be used for reversal distance, although approximation ratios are not kept the same. The reversal distance is NP-hard [99] and is $O(k)$-approximate when the largest family has k genes [83]. For the general unbalanced case, the reversal distance is APX-hard [4].

Recall that problems are equivalent for signed or unsigned strings when considering only operations that do not change the sign of genes, such as transpositions or block interchanges. The transposition distance for unsigned strings is NP-hard [99] and there exist an $O(\log n \log^* n)$-approximation [36,116] for the general case and an $O(k)$-approximation [83] when the largest family has k genes, which comes from the close relation between transposition distance and MCSP for unsigned strings [82, 117]. Similarly, the block interchange distance for unsigned strings is NP-hard [35] and is closely related to the transposition distance [55]. Approximation algorithms for the former are approximation algorithms for the latter with ratios multiplied by a constant not exceeding 2 [55]. Therefore, this problem can also be approximated with ratios $O(\log n \log^* n)$ [36, 116] and $O(k)$ [83] when the largest family has k genes.

Regarding the DCJ operation, even when the genomes are balanced the DCJ distance is NP-hard [115].

1.2 Some background on family-free measures

The family-free approach was proposed recently [22, 45] as an alternative to family-based methods. In the family-based setting, comparative studies can lead to inaccurate results, as most gene families are uncurated and this can result in erroneous homology assumptions. Family-free methods can sometimes even better reflect the biological reality, where the similarity of each pair of genes is calculated directly. On the other hand, problems in this setting are usually at least as difficult as family-based problems. We present in this section the few problems studied under family-free approaches. In this context, many rearrangement operations have not been studied at all.

In family-free methods, no prior family assignment is made. Instead, we directly access the pairwise similarities between genes of the compared genomes. For such, it is a common sense that comparing protein sequences (amino acids) coded by codons is almost always preferable than DNA sequences (nucleotides). Also, some amino acids can be easily replaced by others because of their similar chemical and spatial properties, but such information is contained in protein substitution matrices [74, 118] used by alignment softwares (such as BLASTP).

The first family-free measure proposed was an adjacency similarity measure named FF-Adjacencies, shown to be NP-hard [45] to compute for linear or circular genomes. The FF-Adjacencies is modeled similarly to the intermediate model for family-based measures with duplicate genes [89]. This adjacency similarity is related to the family-based adjacency similarity, which is complementary to the family-based breakpoint distance.

The DCJ similarity was first proposed following the ideas of FF-Adjacencies [22], being redefined later [89] under a model similar to the full model for family-based measures with duplicate genes and proved to be NP-hard [89]. The DCJ distance is NP-hard and APX-hard [89].

1.3 Some background on ancestral genome reconstruction

The growing field of paleogenomics aims at reconstruction and analysis of genomic data in extinct species. Research in this field is supported by two different approaches focused on tracing genome evolution back in time [97]. The first one, the direct or allochronic approach, relies on methods for the extraction of ancient DNA from subfossils. The second one, indirect or synchronic, reconstructs ancestral genomes by comparing modern descendant genomes.

Ancestral genomes, that is, genome sequences of extinct species, are constituent for inferring phylogenies and for our understanding of evolutionary processes, such as adaptations to changing environmental conditions, the dynamics of genomes within populations and across species, and the study of pathogen-host interactions. At the same time, the study of ancestral sequences can give insights into gene function, regulatory networks, and molecular processes.

Next-generation sequencing technologies and the following growing of genomic databases have allowed, specially for plants, the comparison of related genomes and the inference of their ancestral genome known as *most recent common ancestor* (MRCA).

Reconstructing MRCAs is relevant not only for understanding the evolution of a particular set of organisms, but also for providing insight on other research questions, such as the use of conserved orthologous gene clusters to define efficient strategies for genetic studies and gene isolation or to improve the accuracy of gene annotation [108].

The field of computational paleogenomics has established several methods to infer genome sequences of ancestral organisms from genome sequences of their extant descendants and relatives. In general, ancestral genome reconstruction is divided into two largely complementary—although interdependent—tasks: One is the inference of the genome's *architecture*, i.e., the number, appearance, and composition of ancestral chromosomes. The other concerns the reconstruction of the *genome content* constituting the set of "building blocks" of the genome architecture that are the genomic markers, often represented by (protein coding) genes. Provided that the genomic coordinates of extant markers are known, the latter task coincides with the inference of homology classes (families), and the determination of whether and how many members of each family are part of the ancestral genome content [97]. Most popular methods for reconstructing the ancestral genome architecture follow one of two strategies: either they make use of a *genome rearrangement model* to derive parsimonious rearrangement scenarios that explain the observed differences in modern genome architectures, or they infer *syntenic blocks*. These constitute conserved neighborhoods of individual pairs of markers, also denoted *adjacencies*, or neighborhoods of marker sets comprising more than two markers [6].

Model-based reconstruction methods

For mostly all known rearrangement models, if duplications are not considered, the minimum number of operations (distance) to transform one given genome into another given genome can be computed efficiently. However, considering one step further, the reconstruction of an ancestral genome for three given genomes under the DCJ model, also called the DCJ median problem, is already an NP-hard problem [125]. When duplications are taken into account, even pairwise distances between given genomes are NP-hard to compute for mostly all rearrangement models.

Consequently, to deal with the aforementioned issues for the reconstruction of ancestral genomes, there are some proposed heuristic methods such as GASTS [133], MGRA [8] and Badger [84]. GASTS and MGRA operate under the command of parsimony, i.e., they aim to minimize the number of DCJ operations occurring along the edges of a given phylogeny. Conversely, Badger [84] considers a probabilistic model, using Bayesian analysis, aiming to solve the corresponding maximum likelihood problem. All methods assume that each marker is unique, with MGRA supporting that some markers may be missing in some of the genomes. As mentioned before, despite this unrealistic limitation, both objectives are computationally intractable, hence neither of the methods is exact but both implement fast heuristics that permit the analysis of biological datasets in practice.

Synteny-based reconstruction methods

Syntenic blocks are blocks of two or more extant genome sequences that are *homologous*, i.e., they originate from the same block of a common ancestral sequence. Methods that make use of inferred syntenic blocks must resolve conflicts between contradicting neighborship relations of genomic markers imposed by these blocks in order to derive a total or

partial, sequential or circular order of common ancestral markers. The most popular such method, ANGES [81], identifies a subset of neighborship relations that can be displayed by a PQ-tree, a data structure for capturing local variations in a set of permutations. ANGES' procedure implies that each family can contribute at most with one marker to the ancestral genome content. This severely limits the applicability of the method for the reconstruction of ancestral plant genomes, where multiple rounds of polyploidy have frequently occurred, resulting in multiple copies of each gene. Alternative methods such as PMAG [75] and DeCoStar [46] use likelihood estimation to infer ancestral gene orders, yet are limited to process adjacencies only. Nevertheless, DeCoStar infers evolutionary trees of marker adjacencies and therefore can handle evolutionary events such as duplication, insertion, and loss [6].

Contiguous ancestral regions

Independent of the strategy used for reconstructing ancestral genomes, the outcome are *contiguous ancestral regions* (CARs), that detail the composition of ancestral chromosomes (or parts thereof) as well as the relative order of their contained genomic markers. Such an order may not be entirely fixed—the resolution of ancestral marker orders depends on the input data [136], the method of choice [6], and its alacrity to proclaim neighborship relations derived from the analysis of extant genomes as ancestral. Many methods output multiple candidates for ancestral gene order, either because their strategy is based on sampling, or because it is subject to optimization criteria that give rise to many co-optimal solutions.

1.4 Outline of this thesis

Chapter 2 presents formal definitions of some terms described previously or common to both family-based and family-free methods, along with the basics of complexity classes, reductions and approximation-preserving reductions. Chapter 3 describes a linear time approximation algorithm with approximation ratio $O(k)$ for the DCJ distance problem for balanced unichromosomal linear genomes with duplicate genes, where k is the maximum number of occurrences of any gene in the input genomes [102, 103]. The approximation algorithm is also extended to circular unichromosomal balanced genomes in linear time. In Chapter 4 the family-free DCJ similarity for linear or circular multichromosomal genomes is investigated, showing its APX-hardness, proposing an exact ILP algorithm to solve it and presenting four combinatorial heuristics [104, 106]. Next, Chapter 5 proposes the local DCJ similarity, a novel family-based measure by means of DCJ operations for genomes with duplicate genes. Its usefulness is evaluated by showing that an estabilished pipeline for ancestral reconstruction in plants used in multiple studies [10, 93, 97, 107, 132] can be improved by using this novel measure [105]. Finally, Chapter 6 summarizes the results presented in this doctoral study.

Chapter 2

Preliminaries

The first section of this chapter introduces common definitions for both family-based and family-free measures under DCJ. Even though many definitions are similar for both approaches, most of them differ slightly. Consequently, some general definitions are presented here and the particular remaining definitions are presented in their own chapters. In addition, we assume that the reader is familiar with graphs and their notations.

To fully understand a couple of results in the later chapters, some background in complexity classes is required. The second section of this chapter thus describes briefly a few complexity classes related to hardness and inapproximability of problems and how to demonstrate problems belong to these classes.

2.1 Genomic definitions

A *(genomic) marker* g is an oriented DNA fragment. The term *gene* will be used in this text as a synonym of genomic marker, unless noted otherwise. The double-strandedness of the DNA imposes a relative orientation to each gene g: If g's orientation conforms with the (predetermined) reading direction of its sequence, g is denoted by the symbol g itself. Otherwise, it has reverse orientation and is denoted by $\overline{g}$ or $-g$. In addition, $(\overline{\overline{g}} = g)$. If the orientation of a gene g is irrelevant, we denote by $|g|$ the gene itself, omitting its orientation.

A *chromosome* is a linear or a circular sequence of genes, and a *genome* is a set of chromosomes. Each one of the two ends of a linear chromosome is a *telomere*, represented by the symbol $\circ$. The two distinct *extremities* of a gene g are called *tail* and *head*, denoted by g^t and g^h, respectively. An *adjacency* in a chromosome is then an unordered pair of consecutive extremities (one of the two extremities can be a telomere). As an example, observe that the adjacencies e^h, $e^t b^t$, $b^h d^t$, $d^h c^t$, $c^h f^t$, $f^h a^h$ and a^t can define a linear chromosome. Often adjacencies with telomeres are represented explicitly, for instance $\circ e^h$ and $a^t \circ$. Another representation of the same linear chromosome, flanked by parentheses for the sake of clarity, would be $(\circ\ \overline{e}\ b\ d\ c\ f\ \overline{a}\ \circ)$. As noted above, a genome is represented by the set of its chromosomes, for instance $\{(\circ\ \overline{e}\ b\ d\ \overline{a}\ \circ), (\circ\ \overline{f}\ \overline{g}\ c\ \circ)\}$. However, a unichromosomal genome may be optionally represented by its only chromosome, for instance $(\circ\ \overline{e}\ b\ d\ \overline{a}\ \circ)$.

A *double-cut-and-join* or DCJ operation [134] applied to a genome A is the operation that cuts two adjacencies of A and joins the separated extremities in a different way, creating two new adjacencies. For example, a DCJ acting on two adjacencies pq and rs would create either the adjacencies pr and qs, or the adjacencies ps and qr (this could correspond to an inversion, a reciprocal translocation between two linear chromosomes, a fusion of two circular chromosomes, or an excision of a circular chromosome, that is, splitting it in two). In the same way, a DCJ acting on two adjacencies pq and r would create either pr and q, or p and qr (in this case, the operation could correspond to an inversion, a translocation, or a fusion of a circular and a linear chromosome). For the cases described so far we can notice that for each pair of cuts there are two possibilities of joining. There are two special cases of a DCJ operation, in which there is only one possibility of joining. The first is a DCJ acting on two adjacencies p and q, that would create only one new adjacency pq (that could represent a circularization of one or a fusion of two linear chromosomes). Conversely, a DCJ can act on only one adjacency pq and create the two adjacencies p and q (representing a linearization of a circular or a fission of a linear chromosome).

Consider, a DCJ applied to the unichromosomal genome $(\circ\ c\ \overline{a}\ d\ b\ \overline{e}\ f\ \circ)$ that cuts before and after $\overline{a}\ d$, creating the segments $(\circ\ c\ \bullet)$, $(\bullet\ \overline{a}\ d\ \bullet)$ and $(\bullet\ b\ \overline{e}\ f\ \circ)$, where the symbol $\bullet$ represents the open ends. If we then join the first with the third and the second with the fourth open end, we obtain $(\circ\ c\ \overline{d}\ a\ b\ \overline{e}\ f\ \circ)$. This DCJ corresponds to the inversion of contiguous genes $\overline{a}\ d$.

In unichromosomal genomes, DCJ operations can correspond to various rearrangement operations such as reversals, block interchanges (transpositions included), or transreversals (transpositions followed by a reversal on one of the transposed segments). Notice that block interchanges and transreversals are modeled by two DCJs representing one fission, which creates a "temporary" chromosome, followed by one fusion, even though input genomes are unichromosomal. In multichromosomal genomes, DCJ operations can also correspond to other rearrangements, such as translocations, fusions, and fissions.

Lastly, when comparing two distinct genomes, we usually denote them by A and B.

2.2 Complexity classes

This is a very succinct and somewhat informal reference for complexity classes. More complete references are [7, 37, 38, 54, 57], on which this section is based.

Problems are defined as decision or optimization problems, which in turn are either maximization or minimization problems. In a *decision* problem we want to answer an YES/NO question (e.g. given a graph G and an integer k, does there exist an independent set of G of size k?). On the other hand, in an *optimization* problem we want to find an optimal solution from all feasible solutions (e.g. given a graph G, find an independent set of G of maximum size). A *solution* (for an instance) of a problem is the correct YES/NO answer, if it is a decision problem, or an optimal solution, if it is an optimization problem. This common definition of solution for both decision and optimization problems allow us to simplify some definitions in this chapter. We say that an algorithm *solves* a problem if it returns a solution for any instance taken as input.

Completeness and hardness

The *class P* (polynomial time) contains all decision problems that can be solved by a deterministic algorithm in polynomial time. Similarly, the *class NP* (nondeterministic polynomial time) is the class of all decision problems that can be solved by polynomial time nondeterministic algorithms (i.e. using a nondeterministic Turing machine), and the *class NEXP* is the class of all decision problems that can be solved by nondeterministic algorithms in exponential time. In an equivalent definition, the class NP consists of those problems that are "verifiable" in polynomial time, that is, the YES answer have a "certificate" that can be verified in deterministic polynomial time. For instance, for the hamiltonian cycle problem, which is in NP, a certificate could be a sequence of vertices composing a hamiltonian cycle, which can be checked easily. Clearly, P $\subseteq$ NP. It is also known that NP $\subsetneq$ NEXP.

A (*polynomial-time*) *reduction* from a problem $\mathcal{P}$ to a problem $\mathcal{P}'$ is a pair of functions (f, g) such that f transforms instances x of $\mathcal{P}$ into instances $f(x) = x'$ of $\mathcal{P}'$, in polynomial time, in such a way that, given a solution y' for x', g transforms y' into a solution $g(x, y') = y$ for x in polynomial time. We write $\mathcal{P} \leq_{\mathrm{P}} \mathcal{P}'$ to denote that $\mathcal{P}$ is *reduced* to $\mathcal{P}'$. In general, this means that $\mathcal{P}'$ is at least as difficult as $\mathcal{P}$.

As consequence of $\mathcal{P} \leq_{\mathrm{P}} \mathcal{P}'$, if an algorithm is known to solve $\mathcal{P}'$ in polynomial time, $\mathcal{P}$ can also be solved in polynomial time ($\mathcal{P}' \in$ P implies $\mathcal{P} \in$ P) as follows:

1. given x, apply the reduction and obtain x',
2. solve $\mathcal{P}'$ for the instance x' and obtain y', and
3. transform y' into y to solve $\mathcal{P}$.

Further, if $\mathcal{P}$ cannot be solved in polynomial time, then $\mathcal{P}'$ cannot be solved in polynomial time ($\mathcal{P} \notin$ P implies $\mathcal{P}' \notin$ P).

A problem $\mathcal{P}$ is (in the *class*) *NP-complete* if:

(i) $\mathcal{P} \in$ NP, and

(ii) $\mathcal{P}' \leq_{\mathrm{P}} \mathcal{P}$ for every $\mathcal{P}' \in$ NP ($\mathcal{P}$ is as "hard" as any problem in NP).

Thus, NP-complete contains the most difficult problems in NP. If a problem $\mathcal{P}$ satisfies (ii) but not necessarily (i), we say that $\mathcal{P}$ is (in the *class*) *NP-hard*. For instance, the NP-hard problem of finding the longest path between two vertices in a graph is an optimization problem (and thus cannot be in NP), and the NP-hard problem of deciding whether there is a Hamiltonian path in a graph represented by a succinct circuit [56] is a decision problem in NEXP but not in NP, therefore both cannot be NP-complete. A deterministic polynomial time algorithm that solves a NP-complete or NP-hard problem would also solve every problem in NP, what implies that such an algorithm exists if and only if P = NP.

By the definition of NP-complete and because reductions are transitive, a problem $\mathcal{P}$ can be proved to be NP-complete by showing a polynomial-time nondeterministic algorithm that solves $\mathcal{P}$(i.e. $\mathcal{P} \in$ NP) and then showing $\mathcal{P}' \leq_{\mathrm{P}} \mathcal{P}$ for some problem $\mathcal{P}' \in$ NP-complete. Similarly, a problem $\mathcal{P}$ can be proved to be NP-hard just by showing that $\mathcal{P}' \leq_{\mathrm{P}} \mathcal{P}$ for some problem $\mathcal{P}' \in$ NP-complete or $\mathcal{P}' \in$ NP-hard.

Approximations, inapproximability, and related classes

For optimization problems, there are classes equivalent to P and NP. Formally, an NP-*optimization* problem $\mathcal{P} = (I, sol, val, goal)$ is such that:

(i) the set of instances I can be recognized in polynomial time, that is, it can be verified if some instance x is valid ($x \in I$) in polynomial time;

(ii) given $x \in I$, $sol(x)$ is the set of feasible solutions of x and the length of any $y \in sol(x)$ is polynomial in the length of x;

(iii) given $x \in I$ and any y of length polynomial in the length of x, it can be verified if $y \in sol(x)$ in polynomial time;

(iv) given $x \in I$ and $y \in sol(x)$, the *value* (often called *measure*) $val(x, y)$ of the objective function is a positive integer computable in polynomial time; and

(v) $goal \in \{\max, \min\}$ which defines whether we want to maximize or minimize the value of the objective function.

The *class NPO*, an extension of NP for optimization problems, contains all NP-optimization problems, while the *class PO* contains problems in NPO that can be solved in polynomial time, hence it is an extension of P for optimization problems.

The value of an optimal solution (which maximizes or minimizes the value of the objective function, depending on *goal*) is defined as $opt(x)$. Thus, the *performance ratio* (also called *performance factor*) of y with respect to x is defined as:

$$R_{\mathcal{P}}(x, y) = \begin{cases} \dfrac{opt(x)}{val(x, y)}, & \text{if } goal = \max\ ; \\ \\ \dfrac{val(x, y)}{opt(x)}, & \text{if } goal = \min\ . \end{cases} \tag{2.1}$$

The performance ratio can be seen as the quality of a solution. It is a number greater than or equal to 1 and, the closer it is to 1, the closer y is to an optimal solution.

Approximation algorithms find approximate solutions with provable performance ratios for any instance in polynomial time, usually concerning NP-hard problems. Given an optimization problem $\mathcal{P}$, an approximation algorithm is an *α-approximation algorithm* (or *α-approximate algorithm*) for $\mathcal{P}$ if, given any $x \in I$, it returns an *approximate solution* $y \in sol(x)$ such that $R_{\mathcal{P}}(x, y) \leq \alpha$. We also say that such an algorithm approximates $\mathcal{P}$ within ratio α, that $\mathcal{P}$ can be approximated within ratio α, or that $\mathcal{P}$ is α-approximable. The *class APX* is formed by problems in NPO for which there is an α-approximation algorithm for some constant α.

In fact, there are approximation algorithms with small constant approximation ratios for many hard problems. Nevertheless, some problems allow polynomial-time approximation algorithms with increasingly ratios by the cost of progressively larger computation times. Obviously, this cost increases with the inverse of the performance ratio. An algorithm for the problem $\mathcal{P} \in$ NPO is said to be a *polynomial-time approximation scheme (PTAS)* if, for any $x \in I$ and any given rational $\varepsilon > 0$, it returns in polynomial time

an approximate solution $y \in sol(x)$ such that $R_{\mathcal{P}}(x, y) \leq 1 + \varepsilon$. The running time of a PTAS can increase quickly as ε decreases, for example, when the running time is $O(n^{1/\varepsilon})$. In most polynomial-time approximation schemes, choosing a very small ε indeed leads to solutions very close to the optimal solution, but at expense of a dramatic increase in the running time. The *class PTAS* is the set of problems in NPO that admit a polynomial-time approximation scheme. As consequence, problems in this class allow arbitrarily good solutions in polynomial time.

From the classes defined in this section, we have that:

$$\text{PO} \subseteq \text{PTAS} \subseteq \text{APX} \subseteq \text{NPO}\,.$$

If any of the subsets above is not proper, then P = NP [54]. On the other hand, if P = NP, then PO = NPO.

A reduction $\mathcal{P} \leq_{\text{P}} \mathcal{P}'$ *preserves membership* in a class $\mathcal{C}$ if $\mathcal{P}' \in \mathcal{C}$ implies $\mathcal{P} \in \mathcal{C}$. Depending on its type, an *approximation-preserving* reduction preserves membership in either APX (generically denoted by $\leq_{\text{APX}}$), PTAS (generically denoted by $\leq_{\text{PTAS}}$), or both classes. Given $\mathcal{P} \in \text{APX}$, a reduction $\mathcal{P}' \leq_{\text{APX}} \mathcal{P}$ can be used to find an approximation algorithm for $\mathcal{P}'$. Similarly, given $\mathcal{P} \in \text{PTAS}$, a reduction $\mathcal{P}' \leq_{\text{PTAS}} \mathcal{P}$ can be used to find an polynomial-time approximation scheme (PTAS) for $\mathcal{P}'$. There are many types of approximation-preserving reductions (see [37]), such as L-reduction, PTAS-reduction or AP-reduction, some of them preserving membership in APX, PTAS or both. The *strict reduction* (denoted by $\leq_{\text{strict}}$), which is the simplest type of approximation-preserving reduction, preserves membership in both APX and PTAS classes and must satisfy the following condition:

$$R_{\mathcal{P}}(x, g(x, y)) \leq R_{\mathcal{P}'}(f(x), y)\,. \tag{2.2}$$

A problem $\mathcal{P} \in \text{NPO}$ is (in the *class*) *APX-complete* if:

(i) $\mathcal{P} \in \text{APX}$ (allows an approximation with constant ratio), and

(ii) $\mathcal{P}' \leq_{\text{PTAS}} \mathcal{P}$ for every $\mathcal{P}' \in \text{APX}$.

If a problem $\mathcal{P} \in \text{NPO}$ satisfies (ii) but not necessarily (i), we say that $\mathcal{P}$ is (in the *class*) *APX-hard*. Similarly to showing that a problem $\mathcal{P} \in$ NP-hard belongs to P, showing that a problem $\mathcal{P} \in$ APX-hard (or APX-complete) belongs to PTAS (has a PTAS) implies P = NP. In other words, if PTAS = APX, then P = NP, otherwise, every APX-complete problem is in APX \ PTAS. Related to this, the class APX-hard is also defined as the class formed by problems $\mathcal{P} \in \text{NPO}$ such that the existence of a PTAS for $\mathcal{P}$ implies P = NP. Therefore, to show that a problem $\mathcal{P}$ is APX-hard, it usually suffices to show that $\mathcal{P}' \leq_{\text{PTAS}} \mathcal{P}$ for some $\mathcal{P}' \in$ APX-hard, as $\leq_{\text{PTAS}}$ is transitive.

For some problems, it is possible to prove that even the existence of an approximation algorithm with ratio smaller than some constant r is impossible, unless P = NP. We also say that is NP-hard to approximate these problems with ratios better (smaller) than r. These are called *inapproximability* results and also imply that the problems are APX-hard, because the existence of a PTAS for them would lead to approximation ratios smaller than r. Finally, notice that APX-hardness implies NP-hardness, since a problem hard to approximate (in polynomial time) certainly is also hard to solve (in polynomial time).

Chapter 3

Family-based DCJ distance

Under the context of family-based approaches, here we describe in details the DCJ distance and present an approximation algorithm for computing it between two balanced unichromosomal genomes [102, 103]. The main step of our approximation algorithm is similar to approximating the problem of computing the *Breakpoint Distance* (BD). The breakpoint distance is NP-hard in the presence of duplicate genes [27] (from the exemplar breakpoint distance when one genome has no duplicates), even if genomes are balanced [21]. Let k be the maximum number of occurrences of any gene in the input genomes. With this parameter, BD has a 1.1037-approximation if $k = 2$ and a 4-approximation if $k = 3$ [60]. Otherwise, for general values of k, it has an $O(k)$-approximation [79, 120]. The latter result is based on a linear time approximation algorithm for the Minimum Common String Partition (MCSP) problem [60] with approximation ratio $O(k)$ [83].

As we will show, the algorithm we developed to compute the DCJ distance of balanced genomes also has an approximation ratio $O(k)$ and linear running time. It works properly on inputs that are balanced unichromosomal linear genomes. Moreover, we describe how to extend it for balanced unichromosomal circular genomes. Aditionally, experiments on simulated data sets show that the approximation algorithm is very competitive both in efficiency and in quality of the solutions.

After some concluding remarks, few techincal details on how to obtain data used in our experiments are presented in an appendix.

3.1 Preliminaries

Before addressing the approximation algorithm, we first present formal definitions related to the subject of this chapter.

Genes, families, and chromosomes

Given a gene g, let $m_A(g)$ be the number of occurrences of g in a genome A. To refer to each occurrence of a gene g unambiguously, we number the occurrences of g from 1 to $m_A(g)$. When there exists at least one gene that occurs more than once in genome A, we say that A has *duplicate genes*.

In this chapter we consider only unichromosomal genomes, that are genomes composed of a single chromosome. Consider for instance the unichromosomal linear genome $A = (\circ\ c_1\ \overline{a}_1\ d_1\ b_1\ \overline{a}_2\ c_2\ \circ)$. In A we have one occurrence of genes b and d and two occurrences of genes a and c, that is, A has duplicate genes, and $m_A(a) = 2$, $m_A(b) = 1$, $m_A(c) = 2$, and $m_A(d) = 1$.

We use the notations $\mathcal{G}(A)$ and $\mathcal{G}^N(A)$, respectively, to refer to the set of (non-numbered) genes and to the set of numbered genes of a genome A. Considering again the genome A above, we have $\mathcal{G}(A) = \{a, b, c, d\}$ and $\mathcal{G}^N(A) = \{a_1, a_2, b_1, c_1, c_2, d_1\}$. Observe that the genomes $A' = (\circ\ c_2\ \overline{a}_1\ d_1\ b_1\ \overline{a}_2\ c_1\ \circ)$, $A'' = (\circ\ c_1\ \overline{a}_2\ d_1\ b_1\ \overline{a}_1\ c_2\ \circ)$, and $A''' = (\circ\ c_2\ \overline{a}_2\ d_1\ b_1\ \overline{a}_1\ c_1\ \circ)$ are equivalent to $A = (\circ\ c_1\ \overline{a}_1\ d_1\ b_1\ \overline{a}_2\ c_2\ \circ)$. Given a genome A, possibly with duplicate genes, we denote by $[A]$ the equivalence class of genomes that can be obtained from A by swapping indices between occurrences of the same gene.

Balanced genomes

Let A and B be two unichromosomal genomes, possibly with duplicate genes. If they contain the same number of occurrences of each gene, i.e. $\mathcal{G}^N(A) = \mathcal{G}^N(B)$, we say that A and B are *balanced*. We can then simply denote by $\mathcal{G} = \mathcal{G}(A) = \mathcal{G}(B)$ the set of (non-numbered) genes and by $\mathcal{G}^N = \mathcal{G}^N(A) = \mathcal{G}^N(B)$ the set of numbered genes of A and B. For example, for balanced genomes $A = (\circ\ c_1\ \overline{a}_1\ d_1\ b_1\ c_2\ c_3\ \circ)$ and $B = (\circ\ a_1\ c_3\ \overline{c}_1\ \overline{b}_1\ d_1\ c_2\ \circ)$ we have $\mathcal{G} = \{a, b, c, d\}$ and $\mathcal{G}^N = \{a_1, b_1, c_1, c_2, c_3, d_1\}$.

DCJ distance and adjacency graph

Observe that the DCJ operation alone can only sort balanced genomes. We formally define the DCJ distance problem:

> **Problem** DCJ-DISTANCE(A, B): Given two balanced genomes A and B, compute their DCJ distance $\mathrm{d}_{\text{DCJ}}(A, B)$, i.e., the minimum number of DCJ operations required to transform A into B', such that $B' \in [B]$.

Any sequence of $\mathrm{d}_{\text{DCJ}}(A, B)$ DCJ operations transforming A into $B' \in [B]$ is called an *optimal* sequence of DCJ operations.

Given two balanced genomes A and B, $\mathrm{d}_{\text{DCJ}}(A, B)$ can be computed with the help of some concepts. A genome A can also be defined as a *set of adjacencies* $\mathrm{adj}(A)$ of its numbered genes. For the genome $A = \{(\circ\ c_1\ \overline{a}_1\ d_1\ b_1\ \overline{a}_2\ c_2\ \circ)\}$, for instance, we have $\mathrm{adj}(A) = \{\ \circ c_1^t\ ,\ c_1^h a_1^h\ ,\ a_1^t d_1^t\ ,\ d_1^h b_1^t\ ,\ b_1^h a_2^h\ ,\ a_2^t c_2^t\ ,\ c_2^h \circ\ \}$.

Given two balanced genomes A and B, the *adjacency graph* $AG(A, B)$ [15] is a bipartite multigraph such that each partition corresponds to the set of adjacencies of one of the two input genomes, and an edge connects the same gene extremities of adjacencies in both partitions, regardless of their index numbers. We say that the edge *represents* those extremities. If A and B are linear, each of the two telomeres of A must be connected by an edge to each of the two telomeres of B. After this operation, called *capping* of telomeres, the adjacency graph has no vertex of degree one.

Without duplicate genes

First we consider the case when the genomes A and B contain no duplicate genes. If A and B are circular, there is a one-to-one correspondence between the set of edges in $AG(A, B)$ and the set of gene extremities. In this case, all vertices have degree two and thus the adjacency graph is a collection of disjoint cycles. Here, problem DCJ-DISTANCE can easily be solved in linear time [15, 134] using the formula

$$\mathrm{d}_{\text{DCJ}}(A, B) = n - c\,,$$

where $n = |\text{adj}(A)| = |\text{adj}(B)| = |\mathcal{G}|$ is the number of adjacencies or genes in any of the two genomes and c is the number of cycles in $AG(A, B)$.

If A and B are linear, besides the edges connecting gene extremities, each telomere of A must be connected by an edge to each telomere of B. There is then an ambiguity concerning the vertices that contain a telomere, that have degree three. This means that we need to choose one of the two possible matchings of telomeres to obtain a graph in which all vertices have degree two, that is, a graph that is composed of cycles only. We must choose a matching that maximizes the number of cycles in the resulting $AG(A, B)$. To accomplish this task, we just need to do a walk on the graph starting in one telomere of A until we find the next telomere in $AG(A, B)$. If the second telomere is also in A, then we can pick any of the two possible matchings. In this case we have one big cycle covering all four vertices that contain a telomere. If the second telomere is in B, then we can pick the matching that connects these two telomeres (and consequently connects the other two telomeres, that were not covered by this walk). In this case we have two cycles covering the four vertices that contain a telomere. Once this matching is defined, problem DCJ-DISTANCE can again be solved in linear time [134] using the formula

$$\mathrm{d}_{\text{DCJ}}(A, B) = n - c\,,$$

where $n = |\text{adj}(A)| = |\text{adj}(B)| = |\mathcal{G}| + 1$ is the number of adjacencies in any of the two genomes and c is the number of cycles in $AG(A, B)$.

With duplicate genes

When genomes have duplicate genes, problem DCJ-DISTANCE becomes NP-hard [115]. In the same paper, the authors present an exact, exponential-time algorithm for its solution, phrased in form of an Integer Linear Program (ILP).

An approach to compute the DCJ distance with duplicate genes

Observe that, in the presence of duplicate genes, the adjacency graph may contain vertices of degree larger than two. A *decomposition* of $AG(A, B)$ is a collection of disjoint cycles covering all vertices of $AG(A, B)$.

There can be multiple ways of selecting a decomposition of the adjacency graph. We need to find one that allows to match each occurrence of a gene in genome A with exactly one occurrence of the same gene in genome B and each telomere of A to one telomere of B. In order to build such a decomposition, we need the following definitions.

Let g_i and g_j be, respectively, occurrences of the same gene g in genomes A and B. The edge e that represents the connection of the head of g_i to the head of g_j and the edge

f that represents the connection of the tail of g_i to the tail of g_j are called *siblings*. Two edges are *compatible* if they are siblings, if they represent the connection of extremities of distinct occurrences of the same gene, or if they represent the connection of extremities of distinct genes. Otherwise they are *incompatible*. A set of edges is *consistent* if it has no pair of incompatible edges. A cycle C of $AG(A, B)$ is *consistent* if the set $E(C)$ of edges of C is consistent. Note that, when constructing a decomposition, by choosing consistent cycles one may still select incompatible edges that occur in separate cycles (see the three dotted cycles of length 2 in Fig. 3.1). Thus, consistency cannot be taken into account in cycles separately.

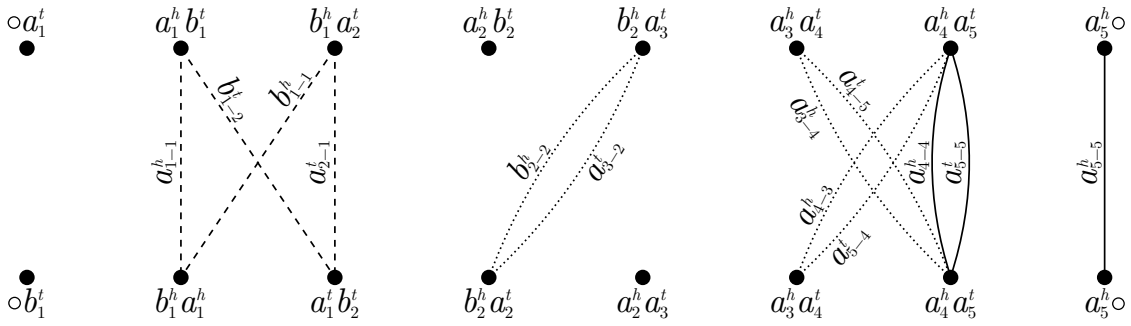


Figure 3.1: **Examples of an inconsistent cycle (dashed edges) and an inconsistent set of cycles (dotted edges).** The adjacency graph for $A = (\circ\ a_1\ b_1\ a_2\ b_2\ a_3\ a_4\ a_5\ \circ)$ and $B = (\circ\ b_1\ \overline{a}_1\ b_2\ a_2\ a_3\ a_4\ a_5\ \circ)$, with some edges omitted. For the sake of clarity, edges are labeled with extremities they represent. For example, an edge labeled g^t_{i-j} represents extremities g^t_i from A and g^t_j from B.

A set of cycles $\{C_1, C_2, \ldots, C_k\}$ of $AG(A, B)$ is *consistent* if and only if $E(C_1) \cup E(C_2) \cup \cdots \cup E(C_k)$ is consistent. A *consistent decomposition* D of $AG(A, B)$ is a consistent set of disjoint cycles that cover all vertices in $AG(A, B)$. Observe that in a consistent decomposition D we have only pairs of siblings, i.e., either an edge e and its sibling f are in D or both e and f are not in D. Thus, a consistent decomposition corresponds to a matching of occurrences of genes and telomeres in both genomes and allows us to compute the value

$$d_D = n - c_D\ ,$$

where $n = |\text{adj}(A)| = |\text{adj}(B)|$ and c_D is the number of cycles in D. Observe that $n = |\mathcal{G}^N|$ if A and B are circular. If A and B are linear genomes, then $n = |\mathcal{G}^N| + 1$.

We can now compute the DCJ distance of two balanced unichromosomal genomes.

Theorem 3.1 *Given two balanced unichromosomal genomes A and B, the solution for the problem* DCJ-DISTANCE *is given by the following equation:*

$$\text{d}_{\text{DCJ}}(A, B) = \min_{D \in \mathcal{D}}\{d_D\}\ ,$$

where $\mathcal{D}$ is the set of all consistent decompositions of $AG(A, B)$.

Proof. Since a consistent decomposition allows to match duplicates in both genomes, clearly $\text{d}_{\text{DCJ}}(A, B) \leq \min_{D \in \mathcal{D}}\{d_D\}$. Now, assume that $\text{d}_{\text{DCJ}}(A, B) < \min_{D \in \mathcal{D}}\{d_D\}$. By definition, this distance corresponds to an optimal rearrangement scenario from A to some

$B' \in [B]$ and therefore implies a matching between the genes of A and the genes of B'. Furthermore, this matching gives rise to a consistent decomposition D' of $AG(A, B)$ such that $d_{D'} < \min_{D \in \mathcal{D}}\{d_D\}$, which is a contradiction. □

A consistent decomposition D such that $d_D = \mathrm{d}_{\text{DCJ}}(A, B)$ is said to be *optimal*.

Once a consistent decomposition D of the adjacency graph $AG(A, B)$ is found, following [15] it is easy to derive in linear time a DCJ rearrangement scenario with d_D DCJ operations transforming A into B. Moreover, an optimal consistent decomposition allows to find all optimal rearrangement scenarios [23].

3.2 The $O(k)$-approximation

Actually, all definitions and properties for the DCJ distance of balanced genomes presented from the beginning of the chapter to here work properly for the general case, where genomes can be multichromosomal. However, as we will see in this section, to solve the DCJ distance problem we use an intermediate procedure whose inputs are strings. For this reason we restricted our inputs to unichromosomal genomes. Moreover, for the moment we will additionally consider only unichromosomal linear genomes, discussing later how to deal with unichromosomal circular genomes. The extension to multichromosomal genomes is left as an open problem.

Approximating the DCJ distance by cycles of length 2

As mentioned above, given two balanced unichromosomal linear genomes A and B, we have to find a consistent decomposition of $AG(A, B)$ to compute the DCJ distance according to Theorem 3.1. Recall that this is an NP-hard problem [115].

Given a consistent decomposition $D \in \mathcal{D}$ of the adjacency graph $AG(A, B)$, we can see that

$$d_D = n - c_D = n - c_2 - c_> \,,$$

where $n = |\text{adj}(A)| = |\text{adj}(B)|$, c_2 is the number of cycles of length 2, and $c_>$ is the number of cycles of length longer than 2 in D.

Building a consistent decomposition by maximizing c_2 as a first step is itself an NP-hard problem [3]. Furthermore, this strategy is not able to optimally solve the DCJ distance, as we can see in Fig. 3.2. Nevertheless, it allows us to approximate the DCJ distance:

Lemma 3.2 *A consistent decomposition D' of $AG(A, B)$ containing the maximum number of cycles of length 2 is a 2-approximation for the* DCJ-DISTANCE *problem.*

Proof. Let c_2^* and $c_>^*$ be the number of cycles of length 2 and longer than 2, respectively, of an optimal consistent decomposition D^* of $AG(A, B)$. Let c_2' and $c_>'$ be the numbers analogous to c_2^* and $c_>^*$ with respect to the decomposition D'. It it easy to see that $c_2^* + 2c_>^* \leq n$, thus

$$\begin{aligned} 0 &\leq n - c_2^* - 2c_>^* \\ n - c_2^* &\leq n - c_2^* - 2c_>^* + n - c_2^* \\ n - c_2^* &\leq 2(n - c_2^* - c_>^*) \,. \end{aligned} \tag{3.1}$$

Therefore, we have

$$\begin{aligned}
\frac{d_{D'}}{d_{D^*}} &= \frac{n - c'_2 - c'_>}{n - c^*_2 - c^*_>} \\
&\leq \frac{n - c^*_2 - c'_>}{n - c^*_2 - c^*_>} && (3.2) \\
&\leq \frac{n - c^*_2}{n - c^*_2 - c^*_>} \\
&\leq \frac{2(n - c^*_2 - c^*_>)}{n - c^*_2 - c^*_>} && (3.3) \\
&= 2\,, && (3.4)
\end{aligned}$$

where (3.2) holds since $c'_2 \geq c^*_2$, and (3.3) is true from (3.1). □

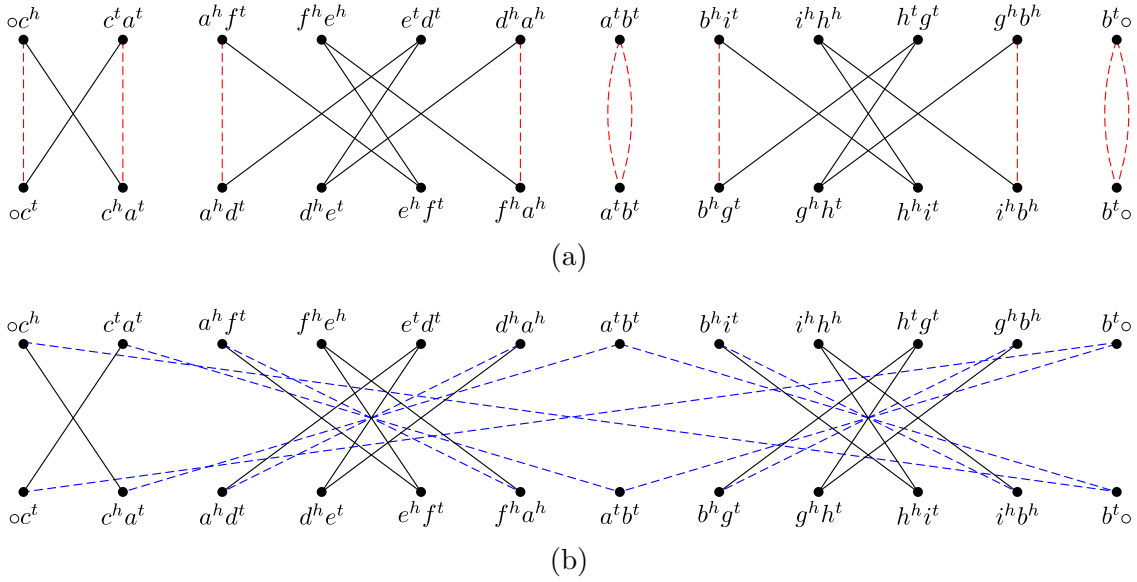


Figure 3.2: **Two consistent decompositions for the same pair of genomes.** The genomes (with gene indices omitted) are $A = (\circ\ \overline{c}\ a\ f\ \overline{e}\ d\ \overline{a}\ b\ i\ \overline{h}\ g\ \overline{b}\ \circ)$ and $B = (\circ\ c\ a\ d\ e\ f\ \overline{a}\ b\ g\ h\ i\ \overline{b}\ \circ)$. Solid edges are in both decompositions. (a) A consistent decomposition D' containing the maximum number of cycles of length 2, composed of 2 cycles of length 2, 1 cycle of length 4 and 2 cycles of length 8, resulting in $d_{D'} = 12-5 = 7$. (b) An optimal consistent decomposition D^*, composed of 6 cycles of length 4, resulting in $d_{D^*} = 12 - 6 = 6$.

The problem of finding a decomposition maximizing c_2 is equivalent to the Adjacency Similarity (AS) problem [3], the complement of the Breakpoint Distance (BD) problem, where one wants to minimize $n - c_2$.

Minimum common string partition

The main result of this chapter relies on a restricted version of the Minimum Common String Partition (MCSP) problem [60, 83], described briefly as follows.

Given a string s, a *partition* of s is a sequence $\mathcal{S} = [\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m]$ of substrings called *blocks* whose concatenation is s, i.e., $\mathcal{S}_1\mathcal{S}_2 \cdots \mathcal{S}_m = s$, and m is the *size* of $\mathcal{S}$.

Two strings s and t are *balanced* if any character has the same number of occurrences in s and in t, disregarding signs. Given two balanced strings s and t, and partitions $\mathcal{S} = [\mathcal{S}_1, \ldots, \mathcal{S}_m]$ of s and $\mathcal{T} = [\mathcal{T}_1, \ldots, \mathcal{T}_m]$ of t, the pair $(\mathcal{S}, \mathcal{T})$ is a *common partition* of s and t if there exists a permutation f on $\{1, \ldots, m\}$ such that $\mathcal{S}_i = \mathcal{T}_{f(i)}$ for each $i = 1, \ldots, m$. The minimum common string partition problem (MCSP) is to find a common partition $(\mathcal{S}, \mathcal{T})$ of two balanced strings s and t with minimum size.

We are interested in a restricted version of MCSP:

> **Problem** k-MCSP(s, t): Given two balanced strings s and t such that the number of occurrences of any character in s and t is bounded by k, find a common partition $(\mathcal{S}, \mathcal{T})$ of s and t with minimum size.

Now let $occ(A) = \max_{g \in \mathcal{G}(A)}\{m_A(g)\}$ be the maximum number of occurrences of any gene in a genome A. If two genomes A and B are balanced, we have $occ(A) = occ(B)$. For simplicity, in this case we use only occ.

For a given unichromosomal linear genome A, let the *index-free* string $\widehat{A}$ be the gene sequence of the chromosome of A ignoring telomeres and gene indices. For example, for genome $A = (\circ\ c_1\ \overline{a}_1\ d_1\ b_1\ c_2\ c_3\ \circ)$, we have $\widehat{A} = c\overline{a}dbcc$.

Finding consistent decompositions

In this section we present a linear time approximation algorithm Consistent-Decomposition, which receives two balanced unichromosomal linear genomes A and B with $occ = k$ and returns a consistent decomposition of $AG(A, B)$, which is an $O(k)$-approximation for the DCJ distance. The main steps of Consistent-Decomposition can be briefly described as follows.

First, from the input genomes A and B, we build their adjacency graph $AG(A, B)$. We can then find a consistent decomposition by computing an approximation for k-MCSP$(\widehat{A}, \widehat{B})$, which gives an approximation for the number of breakpoints between genomes A and B. After that we remove the chosen cycles of length 2 from $AG(A, B)$. Following, we iteratively collect arbitrary cycles of length longer than 2, removing them from the remaining graph after each iteration. Finally, we return the set of collected cycles as a consistent decomposition of $AG(A, B)$. Pseudocode of Consistent-Decomposition is given in Algorithm 3.1. The individual steps are detailed in the following.

Step 1 of Consistent-Decomposition consists of building the adjacency graph of the given balanced genomes A and B as described previously. After that, Step 2 collects cycles of length 2 of $AG(A, B)$ using an $O(k)$-approximation algorithm for k-MCSP$(\widehat{A}, \widehat{B})$ [83]. Step 3 removes from $AG(A, B)$ vertices covered by cycles in $\mathcal{C}_2$ and edges incompatible with edges of cycles in $\mathcal{C}_2$.

Step 4 constructs the set $\mathcal{C}_>$ by decomposing the remaining graph into consistent cycles. Iteratively, it chooses a consistent cycle C and then removes from the remaining graph vertices covered by C. To find C, it can start with an empty path, choose some edge e from the remaining graph that extends the path and then remove from the remaining graph edges incompatible with e (just inspecting edges incident to vertices which are adjacent to e and to its sibling), repeating both edge selection and removal steps until the cycle is closed (it is easy to verify that this procedure will always close a consistent cycle). Hence

Algorithm 3.1 CONSISTENT-DECOMPOSITION(A, B)

Input: balanced unichromosomal linear genomes A and B such that $occ = k$, $|A| = |B| = n$
Output: a consistent decomposition of $AG(A, B)$

1: Build the adjacency graph $AG(A, B)$
2: Obtain an $O(k)$-approximation $\mathcal{C}_2$ for the set of cycles of length 2 in $AG(A, B)$ using the $O(k)$-approximation algorithm for k-MCSP$(\widehat{A}, \widehat{B})$ [83]
3: Remove from the adjacency graph vertices covered by $\mathcal{C}_2$ and all edges incompatible with edges of $\mathcal{C}_2$
4: Decompose the remaining graph into consistent cycles by iteratively finding a consistent cycle C and then removing from the graph vertices covered by C and edges incompatible with edges of C, collecting them in $\mathcal{C}_>$
5: Return $\mathcal{C}_2 \cup \mathcal{C}_>$

the algorithm does not form an inconsistent cycle nor choose an inconsistent set of cycles. Further, this guarantees that for every edge in the decomposition, its sibling edge will also be in the decomposition. Note that $\mathcal{C}_>$ may contain cycles of length 2 not collected in $\mathcal{C}_2$.

A consistent decomposition of $AG(A, B)$ is then the set $\mathcal{C}_2 \cup \mathcal{C}_>$, returned in Step 5.

To conclude this section, we present the following result which, together with the $O(k)$-approximation algorithm for k-MCSP from [83], establishes an approximation ratio for DCJ-DISTANCE.

Theorem 3.3 *Let A and B be balanced unichromosomal linear genomes with $occ = k$. Let $(\mathcal{A}, \mathcal{B})$ be a common string partition with approximation ratio $O(k)$ for k-MCSP$(\widehat{A}, \widehat{B})$. A consistent decomposition D of $AG(A, B)$, containing cycles of length 2 reflecting preserved adjacencies in $(\mathcal{A}, \mathcal{B})$, is an $O(k)$-approximation for the* DCJ-DISTANCE *problem.*

Proof. Let c_2^* and $c_>^*$ be the number of cycles of length 2 and longer than 2, respectively, of an optimal consistent decomposition D^* of $AG(A, B)$. Let $\mathcal{C}_2$ be the set of cycles of length 2 reflecting preserved adjacencies in $(\mathcal{A}, \mathcal{B})$, and let $\mathcal{C}_>$ be an arbitrary consistent decomposition of cycles in $AG(A, B) \setminus \mathcal{C}_2$. Let $D = \mathcal{C}_2 \cup \mathcal{C}_>$ be a consistent decomposition, $c_2 = |\mathcal{C}_2|$, and $c_> = |\mathcal{C}_>|$. Since $(\mathcal{A}, \mathcal{B})$ is an $O(k)$-approximation of k-MSCP, it follows that $n - c_2 \leq \ell(n - c_2')$, where $\ell = O(k)$ and c_2' is the number of cycles of length 2 in a consistent decomposition with maximum number of cycles of length 2. Hence,

$$\begin{aligned}
\frac{d_D}{d_{D^*}} &= \frac{n - c_2 - c_>}{n - c_2^* - c_>^*} \\
&\leq \frac{\ell\,(n - c_2') - c_>}{n - c_2^* - c_>^*} \\
&\leq \frac{\ell\,(n - c_2')}{n - c_2^* - c_>^*} \\
&\leq 2\ell \left(\frac{n - c_2' - c_>'}{n - c_2^* - c_>^*} \right) \qquad (3.5) \\
&\leq 4\ell\,, \qquad (3.6)
\end{aligned}$$

where (3.5) is analogous to (3.1) and (3.6) holds from (3.4), both in the proof of Lemma 3.2. □

3.3 Running time

Prior to addressing the running time of Consistent-Decomposition, we must consider one implicit but important step in the algorithm, which is to obtain the set $\mathcal{C}_2$ given the output of the k-MCSP approximation algorithm [83]. This algorithm takes as input $\widehat{A}$ and $\widehat{B}$ and outputs a common string partition $(\mathcal{A}, \mathcal{B})$.

Both $\mathcal{A}$ and $\mathcal{B}$ are composed of the same set of substrings, in different orders and possibly reversed, e.g., $\mathcal{A} = [\overline{ba}, a, ab]$ and $\mathcal{B} = [ab, ab, a]$ for index-free strings $\widehat{A} = \overline{ba}aab$ and $\widehat{B} = ababa$. Each substring of length $l > 1$ in $\mathcal{A}$ and $\mathcal{B}$ induces a sequence of $l - 1$ preserved adjacencies in $\widehat{A}$ and $\widehat{B}$. Then we just have to map each substring in $\mathcal{A}$ to the same substring in $\mathcal{B}$ (in case of multiple occurrences, we choose any of them). Considering $\mathcal{A}$ and $\mathcal{B}$ in the example above, ab and $\overline{ba}$ in $\mathcal{A}$ could be mapped to the first and second occurrences of ab in $\mathcal{B}$, respectively, since both ab and $\overline{ba}$ contain exactly the same preserved adjacency $a^h b^t$. We describe carefully in the next paragraphs the algorithm Substring-mapping (Algorithm 3.2) and how to use it to find such a mapping while preserving the linear time complexity of Consistent-Decomposition.

The nontriviality of finding such a mapping in linear time comes from the fact that alphabets of strings representing genomes are not constant size alphabets. They can and most likely will be of size $O(n)$. One could obtain this mapping by sorting substrings in $\mathcal{A}$ and $\mathcal{B}$ or building one trie containing all substrings of $\mathcal{A}$ and $\mathcal{B}$. However, linear time algorithms for sorting strings or building tries assume constant size alphabets. Thus, these approaches do not lead to a mapping in linear running time. Another option could be to add all substrings of $\mathcal{A}$ and $\mathcal{B}$ into a hash map, but since we are working with strings and not integers, perfect hashing techniques does not achieve linear running time.

Before describing the Substring-mapping algorithm, some observations and preprocessing must be addressed. We assume that the value $v(g)$ of each symbol (gene family) g in the alphabet $\mathcal{G}$ is unique and in the range $[1, n]$. For reversed symbols we define $v(\overline{g}) = v(g) + n$, therefore their values will be in the range $[n + 1, 2n]$. Given different strings $s = s_1, \ldots, s_\ell$ and $t = t_1, \ldots, t_\ell$ of the same length ℓ such that i is the first position in which they differ, s is *lexicographically smaller* than t if $v(s_i) < v(t_i)$. (Note that $v(g) < v(\overline{g})$, therefore g comes lexicographically before $\overline{g}$ for any symbol g.)

As preprocessing, we first create normalized versions $\widetilde{\mathcal{A}}$ of $\mathcal{A}$ and $\widetilde{\mathcal{B}}$ of $\mathcal{B}$, to ensure that for any substring s, only s or only its reverse $\overline{s}$ occurs in $\widetilde{\mathcal{A}} \cup \widetilde{\mathcal{B}}$. Therefore, for each string s in $\mathcal{A}$ (respectively $\mathcal{B}$), the normalized partition $\widetilde{\mathcal{A}}$ (respectively $\widetilde{\mathcal{B}}$) contains s itself, if s is lexicographically smaller than $\overline{s}$, or $\overline{s}$ otherwise. For instance, normalizing $\mathcal{A} = [\overline{ba}, a, ab]$ would change it to $\widetilde{\mathcal{A}} = [ab, a, ab]$. Also as a preprocessing step, given that we must find the same substrings in $\mathcal{A}$ and $\mathcal{B}$, it only makes sense to analyze substrings in both sets of the same length. Then, if there are substrings of multiple lengths in $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$, in one pass through them (i.e. linear time) we can gather substrings of same length in buckets. Therefore, we define multisets $\widetilde{\mathcal{A}}_l = \{s \text{ in } \widetilde{\mathcal{A}} : |s| = l\}$ (analogously $\widetilde{\mathcal{B}}_l$) and the generic bucket (multiset) $\widetilde{\mathcal{A}\mathcal{B}}_l = \widetilde{\mathcal{A}}_l \cup \widetilde{\mathcal{B}}_l$ (also recording in some manner whether a string in $\widetilde{\mathcal{A}\mathcal{B}}_l$ comes from $\mathcal{A}$ or $\mathcal{B}$), running the algorithm Substring-mapping for each bucket $\widetilde{\mathcal{A}\mathcal{B}}_l$. See Fig. 3.3 for an example of this preprocessing step.

The main idea of the algorithm Substring-mapping is, given a set of strings of length l, to obtain a set of buckets for some value of i (from 1 to l), each one containing strings

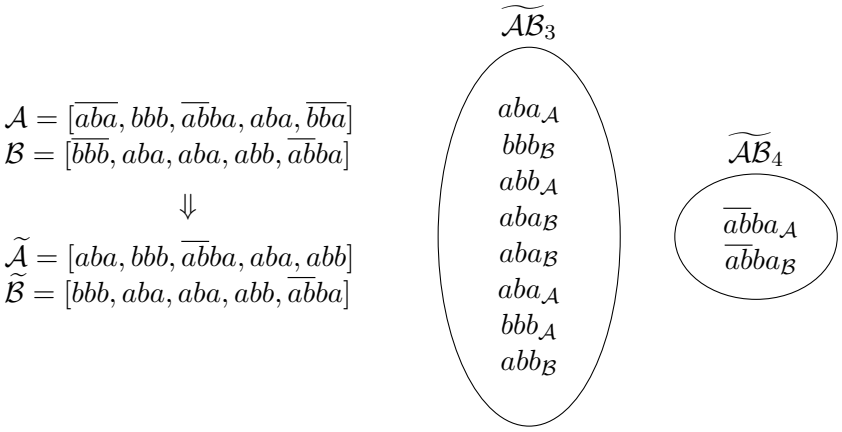


Figure 3.3: **Example of the preprocessing step for the mapping of substrings.** The subscript represents the origin of the string ($\mathcal{A}$ or $\mathcal{B}$).

which are found to be equal to the i-th symbol, by splitting buckets for which strings are equal to the $(i-1)$-st symbol. At the end, each bucket holds equal strings and we just have to map them taking into account their origin, $\mathcal{A}$ or $\mathcal{B}$. See an example in Fig. 3.4. Of course, instead of working with the substrings themselves we work just with references.

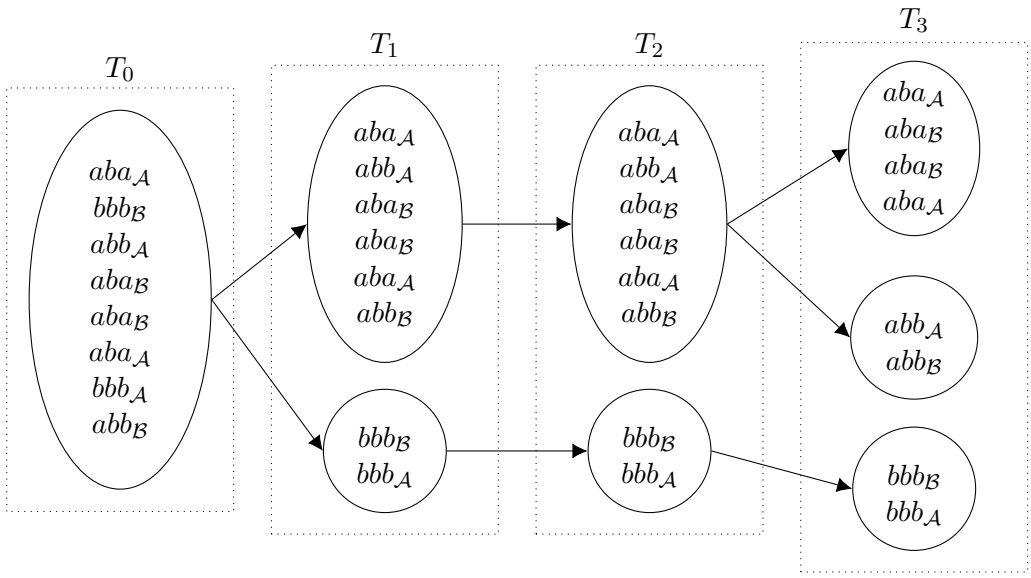


Figure 3.4: **Example of the algorithm SUBSTRING-MAPPING for the bucket $\widetilde{\mathcal{AB}}_3$ of Fig. 3.3.**

It is worth mentioning that the array w plays an important role in this algorithm. Each position j is associated to some symbol g of the alphabet whose value $v(g) = j$. It helps us not having to iterate over alphabet (which can be $O(n)$) repeatedly and further finding the bucket related to some symbol in $O(1)$. Related to this, we avoid iterating over w repeatedly (for instance in the line 6 of this algorithm), which is of size $O(n)$.

Algorithm 3.2 SUBSTRING-MAPPING($\widetilde{\mathcal{AB}}_l$)

Input: the bucket $\widetilde{\mathcal{AB}}_l$, for some $l \in [1, n]$, and an array $w_{1..2n}$ of $2n$ empty buckets
Output: a mapping of strings that come from $\widetilde{\mathcal{A}}_l$ to equal strings that come from $\widetilde{\mathcal{B}}_l$

1: $T_0 = \{\widetilde{\mathcal{AB}}_l\}$
2: **for** $i \leftarrow 1$ **to** l **do**
3: **for each** bucket $S \in T_{i-1}$ **do**
4: **for each** string $s \in S$ **do**
5: add s to bucket $w_{v(s_i)}$
6: **for each** string $s \in S$ **do**
7: **if** bucket $w_{v(s_i)}$ is not empty **then**
8: move contents of bucket $w_{v(s_i)}$ to a new bucket in set T_i
9: **for each** bucket $S \in T_l$ **do**
10: $S_{\mathcal{A}} = \{s \in S : s \text{ comes from } \mathcal{A}\}$
11: $S_{\mathcal{B}} = \{s \in S : s \text{ comes from } \mathcal{B}\}$
12: **for each** $s \in S_{\mathcal{A}}$ **do**
13: map s to some string $s' \in S_{\mathcal{B}}$
14: remove s from $S_{\mathcal{A}}$ and s' from $S_{\mathcal{B}}$

We shall demonstrate in the following lemma that this implicit mapping step can be performed in $O(n)$ time:

Lemma 3.4 *The running time of* SUBSTRING-MAPPING *is proportional to the sum of lengths of strings in $\widetilde{\mathcal{AB}}_l$, for some l.*

Proof. Operations in lines 5, 7, and 8 can be done in constant time and are performed at most once per symbol of strings in $\widetilde{\mathcal{AB}}_l$. Operations after line 9 are performed $O(1)$ times for each string in $\widetilde{\mathcal{AB}}_l$. Therefore, the total running time of SUBSTRING-MAPPING is $O(\sum_{s \in \widetilde{\mathcal{AB}}_l} |s|)$. □

Since the buckets $\widetilde{\mathcal{AB}}_l$ are disjoint, we also have:

Lemma 3.5 *The set $\mathcal{C}_2$ can be obtained from the output of the k-*MCSP *approximation algorithm in $O(n)$ time.*

Proof. Let $\widetilde{\mathcal{S}} = \{\widetilde{\mathcal{AB}}_l$: there exists at least one substring of length l in $\widetilde{\mathcal{A}}$ (and therefore also in $\widetilde{\mathcal{B}}$)$\}$. To obtain $\mathcal{C}_2$, SUBSTRING-MAPPING is called for each $\widetilde{\mathcal{AB}}_l \in \widetilde{\mathcal{S}}$. The time complexity is the sum of time spent in all calls plus some extra preprocessing time. It is easy to see that $\widetilde{\mathcal{S}}$ can be obtained in one pass through $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{B}}$, therefore in linear time. The array of buckets $w_{1..2n}$ can be defined in linear time once before calling SUBSTRING-MAPPING the first time and the buckets are empty at the end of each call. Finally, by Lemma 3.4 the running time of SUBSTRING-MAPPING for some $\widetilde{\mathcal{AB}}_l$ is linear in the sum of lengths of strings in $\widetilde{\mathcal{AB}}_l$, and the total sum of the lengths of strings in buckets $\widetilde{\mathcal{AB}}_l \in \widetilde{\mathcal{S}}$ is $2n$ (each substring of $\widetilde{\mathcal{A}}$ or $\widetilde{\mathcal{B}}$ appears once in exactly one $\widetilde{\mathcal{AB}}_l$). Hence, the total time complexity is $O(n)$. □

Having the running time of the implicit step of obtaining $\mathcal{C}_2$ by the output of the k-MCSP approximation algorithm, we can now analyze the complexity of CONSISTENT-DECOMPOSITION.

Theorem 3.6 *Given balanced unichromosomal linear genomes A and B such that $|A| = |B| = n$ and $occ = k$, the running time of the algorithm* Consistent-Decomposition *is linear in the size of the genomes, i.e., $O(n)$.*

Proof. First, note that $AG(A, B)$ is a bipartite graph composed of $2(n + 1)$ vertices and at most $2kn + 4$ edges. This worst case occurs if there are $\lfloor n/k \rfloor$ gene families of size k, yielding $2k^2$ edges each (k^2 for the gene heads and k^2 for the gene tails), thus $2kn$ edges in total; plus 4 edges from the capping. Therefore, assuming that k is a constant, $AG(A, B)$ is of size $O(n)$.

It is easy to see that Step 1 of Algorithm 1 has linear running time with respect to the size of $AG(A, B)$, i.e., $O(n)$. Computing the k-MCSP approximation [83] in Step 2 (with suffix trees for integer alphabets [52]) takes $O(n)$ time. The same holds for the implicit step described above. The running time of Step 3 is $O(n)$ since we have just to traverse vertices and edges of the remaining adjacency graph. Step 4 consists of collecting cycles arbitrarily and its running time is also linear, since we just have to walk in the remaining graph finding cycles and this can be done looking at each edge and each vertex at most $O(1)$ times. The last step (Step 5) has running time $O(1)$. Therefore, Consistent-Decomposition has running time $O(n)$. □

3.4 Extension to unichromosomal circular genomes

Meidanis *et al.* [90] showed that the problem of calculating the reversal distance for signed circular chromosomes without duplicate genes is essentially equivalent to the analogous problem for linear chromosomes (similar for transpositions in the unsigned case [68]). Therefore, any algorithm for the latter works for the former. The main idea is that each reversal on some region of a circular chromosome can be performed in two ways: reversing it directly or reversing everything else (Fig. 3.5). In the following we show that similar ideas can also be applied to genomes with duplicate genes.

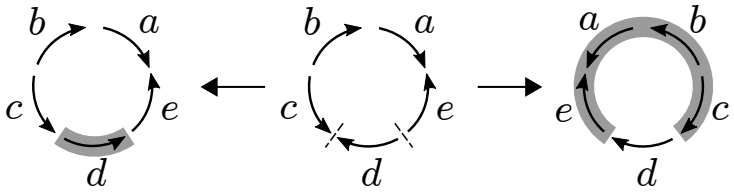


Figure 3.5: **Example of two ways of performing a reversal in a circular chromosome (center).** Dashed lines denote where cuts are made, shaded regions denote the reversed region. The two resulting chromosomes (sides) are the same.

Let A and B be balanced unichromosomal circular genomes such that $occ = k$. For some gene family g, there are genes $g_1, g_2, \ldots, g_l$, with $l \leq k$, in both A and B. Gene g_1 of A can be associated with l genes of B. We linearize A having g_1 with positive sign in the first position and linearize B l times, each one of them having one of the genes $g_1, g_2, \ldots, g_l$ with positive sign in the first position, associating it with g_1 (and assuming that both already are in the correct position). Next, we run Consistent-Decomposition on each one of the l linearizations, taking into account only the sequence of genes from position 2

to position n, keeping the best result. Thus, the running time of this strategy is $l \cdot O(n)$, that is, $O(n)$ since $l \leq k$ and k is a constant.

Corollary 3.7 *For unichromosomal circular genomes A and B, the strategy of keeping the minimum output of* Consistent-Decomposition *for one linearization of A and l linearizations of B as described above leads to an $O(k)$-approximation for problem* DCJ-distance.

Proof. Let d be the DCJ distance between A and B and let g_c be the copy of gene g in B associated to g_1 in A of the correct gene association to obtain d. One of the l linearizations of B associates g_c in B with g_1 in A, approximating d with an $O(k)$ ratio by the Consistent-Decomposition algorithm. Clearly, the minimum output of all l linearizations cannot be higher. □

3.5 Experimental results

We have implemented our approximation algorithm in C++, with the addition of a linear time greedy heuristic for the decomposition of cycles not induced by the k-MCSP approximation (available at https://git.facom.ufms.br/diego/k-dcj).

We compare our algorithm with Shao *et al.*'s ILP [115] (GREDU software package[1]) on simulated datasets. Given two genomes, the ILP based experiments first build the adjacency graph, followed by capping of the telomeres, fixing some safe cycles of length two, and finally invoking an ILP solver to obtain an optimal solution with a time limit of 2 hours. The experiments for were performed on an Intel i7 3.4GHz (4 cores) machine.

Following [115], we simulate artificial genomes with segmental duplications and DCJs. We uniformly select a position to start duplicating a segment of the genome and place the new copy to a new position. From a genome of s distinct genes, we generate an ancestor genome with $1.5s$ genes by randomly performing $s/2l$ segmental duplications of length l, resulting in an average $k = 1.5$. Then we simulate two extant genomes from the ancestor by randomly performing r DCJs (reversals) independently. Thus, the simulated evolutionary distance between the two extant genomes is $2r$. For each gene copy in the extant genomes we keep track of which gene copy in the ancestor it corresponds to, establishing the *reference bijection*, allowing us to compute the *true positive rate*, that is, for two genomes A and B, the rate of matchings of gene occurrences in A and B corresponding to the same gene occurrence in the ancestor genome.

We first set $s = 1000$, test three different lengths for segmental duplications ($l = 1, 2, 5$) and vary the r value over the range $200, 220, \ldots, 500$. We also simulate genomes having $s = 5000$, $l = 1, 2, 5, 10$, and r over the range $1000, 1100, \ldots, 2000$. Figs. 3.6 and 3.9 show the average difference "*computed number of DCJs minus simulated evolutionary distance*", taking as input three pairs of genomes for each combination of l and r, Figs. 3.7 and 3.10 show the *true positive rate*, while Figs. 3.8 and 3.11 show the average running times. Note that, although the DCJ distance is unknown, it is always less than or equal to the simulated evolutionary distance for these artificial genome pairs.

[1] https://github.com/shaomingfu/gredu

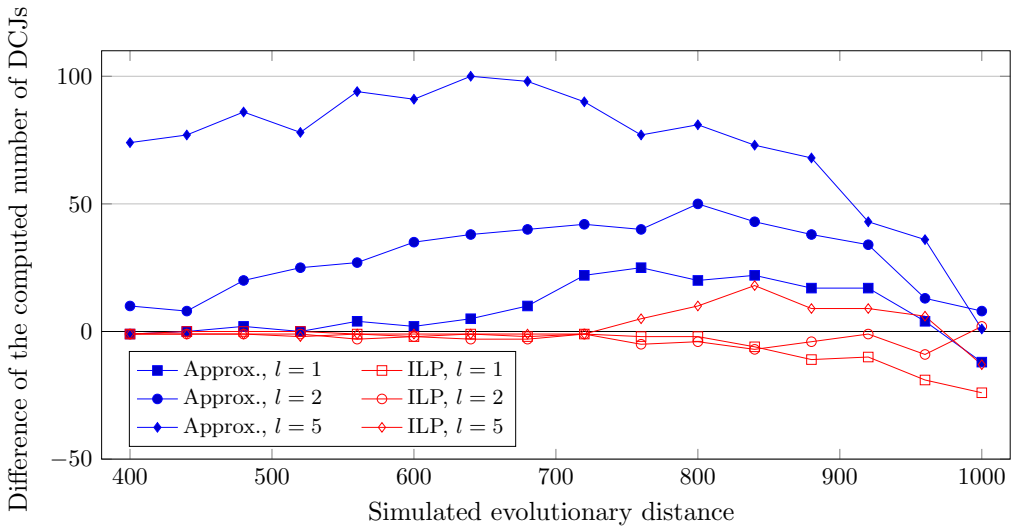


Figure 3.6: **The computed number of DCJs vs. the simulated evolutionary distance for** $s = 1000$**.**

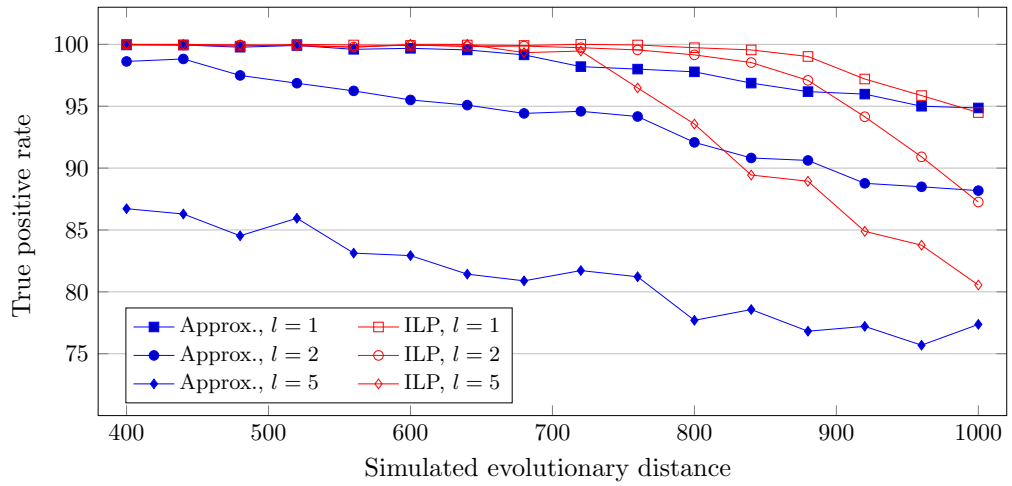


Figure 3.7: **True positive rate for** $s = 1000$**.**

The difference of the number of DCJs (blue lines in Figs. 3.6 and 3.9) calculated by our approximation algorithm remains very close to the simulated evolutionary distance for small values of l. Moreover, it remains roughly the same for the same value of l even for greater values of r. The values obtained by the ILP approach (red lines in Figs. 3.6 and 3.9) are very close to those obtained by the approximation algorithm and to the simulated evolutionary distance from the simulations for $l \leq 2$ and smaller values of r. However, beyond some point the DCJ distance calculated by the ILP gets even lower than the simulated evolutionary distance, showing the limitations of parsimony for larger distance ranges.

While the true positive rate is higher than 95% for most of datasets (Figs. 3.7 and 3.10), the rate remains between 75% and 85% when $l \geq 5$ for the approximation approach and

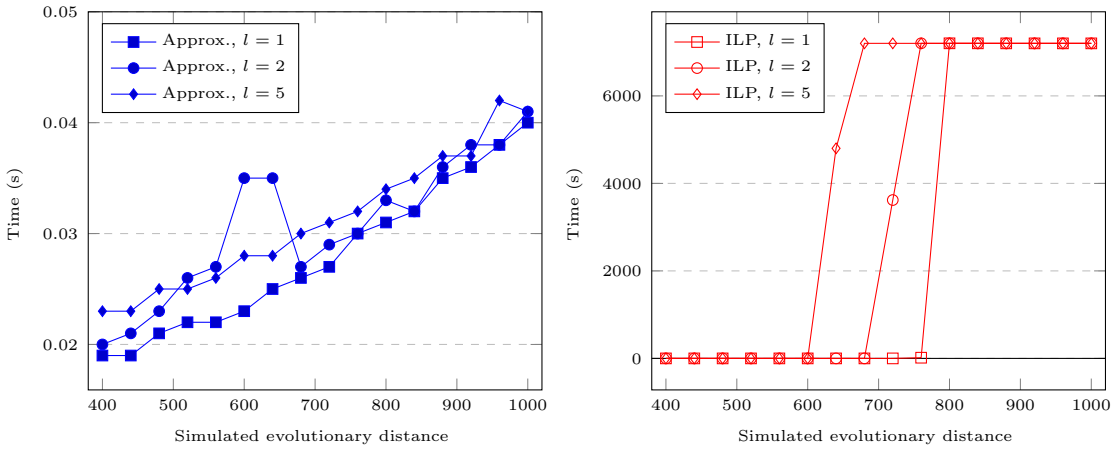


Figure 3.8: **Execution time for $s = 1000$ of (a) approximation and (b) ILP based programs.**

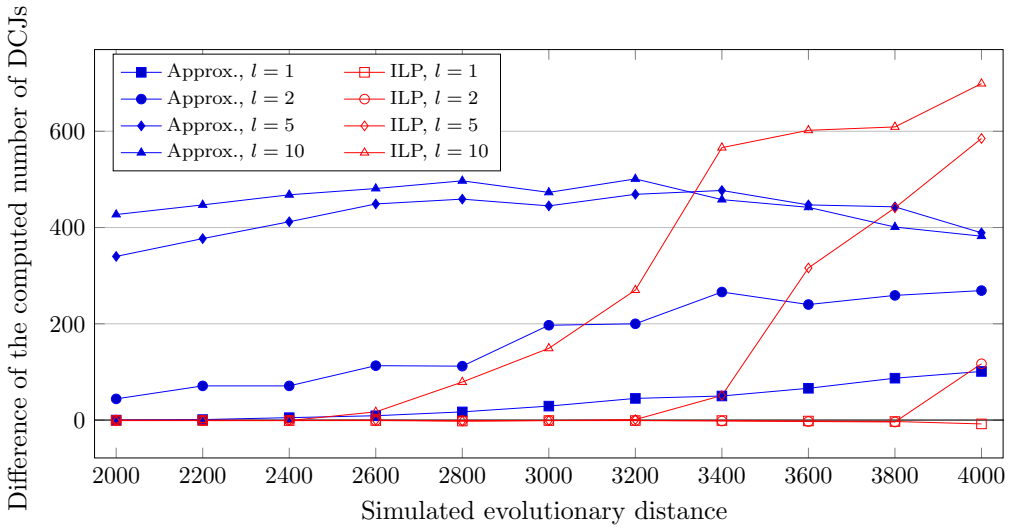


Figure 3.9: **The computed number of DCJs vs. the simulated evolutionary distance for $s = 5000$.**

even for the ILP approach in some cases. For $s = 5000$ and $l \geq 5$, the computed number of DCJs increases while the true positive rate decreases significantly beyond some point for the ILP results. Notice that the approximation algorithm results for the same sets have small rates of increase or decrease, even for greater values of r.

The running time of our implementation of Consistent-Decomposition increases slowly from ≈ 0.03 seconds ($2r = 400$) to ≈ 0.08 seconds ($2r = 1000$) on average, when $s = 1000$, see Fig. 3.8(a). The ILP approach takes ≈ 0.3 seconds for smaller values of r (where the preprocessing step fixes a considerable amount of cycles of length 2 in the adjacency graph), while always reaching the time limit of 2 hours beyond some point, see Fig. 3.8(b). A similar behavior is observed for $s = 5000$ (Fig. 3.11).

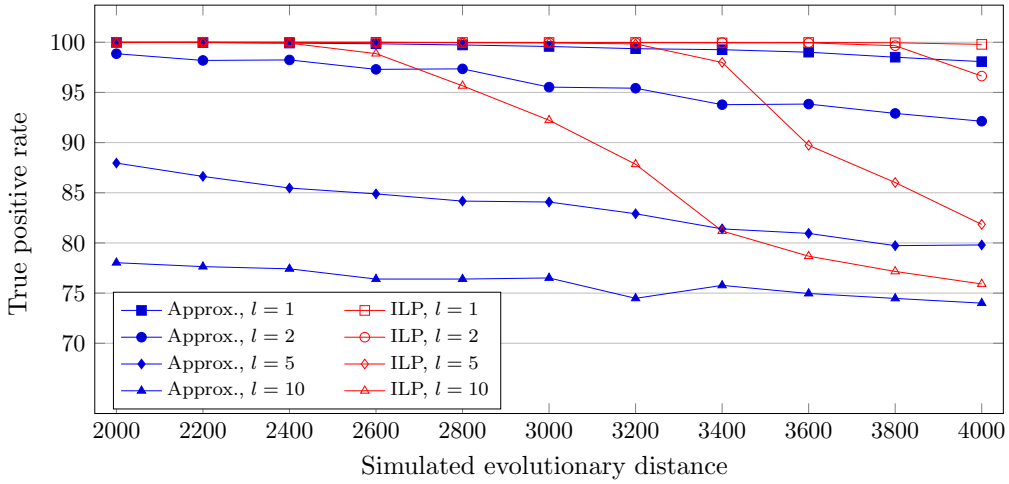
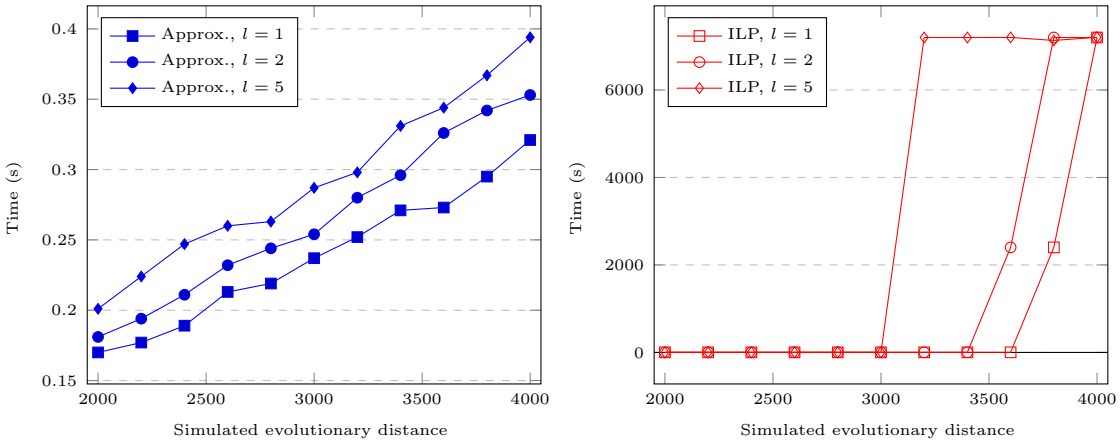


Figure 3.10: **True positive rate for $s = 5000$.**



(a) (b)

Figure 3.11: **Execution time for $s = 5000$ of (a) approximation and (b) ILP based programs.**

3.6 Concluding remarks

This chapter presented a new approximation algorithm for the DCJ distance for balanced unichromosomal linear genomes, an NP-hard problem, and showed how to make it work for circular genomes as well. The algorithm runs in linear time in the size of the genomes and approximates the problem by a ratio of $O(k)$, where k is a constant representing the size of the largest family in both genomes. As experiments on simulated genomes have shown, our algorithm is very competitive both in efficiency and quality of the solutions, in comparison to an exact ILP solution.

Appendix

3.A How to get data

The experiments presented above were performed using files generated in specific formats by a tool. Here we describe them briefly and provide online references for further details.

The simulated data is given by RINGO project[2], which generates files in the UniMog file format[3], as required by the implementation of the approximation algorithm[4] and by the ILP[5]. These files are very simple and define the sequences of genes in two genomes, where genes are represented by numbers and equal numbers represent duplicates of the same gene, that is, genes in the same family. For example, the following file represents two unichromosomal genomes $A = (\circ\ 1\ 2\ {-1}\ {-3}\ \circ)$, named T1, and $B = (\circ\ {-1}\ 3\ {-1}\ 2\ \circ)$, named T2:

```
>T1
1  2 -1 -3 |
>T2
-1 3 -1  2 |
```

The `simulation.py` script of RINGO project generates an ancestral genome and then descendant genomes forming an evolution tree, where the resulting output genomes are the leaves of this tree. Since genomes must be balanced, we must allow duplications only in the ancestral genome by using the `--predup` parameter followed by the number of duplications.

Since UniMog files do not distinguish between different copies of the same gene, GREDU also requires, for each input genome file, a file with the same name of the genomes and extension `.coser` in the COSER format[6]. Files in the COSER format define labels for each gene copy in the corresponding UniMog file. Labels in the these files are usually defined as `<gene_number>_<copy_number>` and should appear in the same order as in the genome files. For instance, for the UniMog file above, we could use the file `T1.coser`:

```
1_1  1  chr1  1
2_1  2  chr1  1
```

[2] https://github.com/pedrofeijao/RINGO

[3] Info about this format in http://bibiserv.techfak.uni-bielefeld.de/dcj, "Manual" tab

[4] https://git.facom.ufms.br/diego/k-dcj

[5] https://github.com/shaomingfu/gredu

[6] Info about this format in http://lcbb.epfl.ch/softwares/coser or https://github.com/shaomingfu/gredu

```
1_2 -1  chr1  1
3_1 -3  chr1  1
```

and the file `T2.coser`:

```
1_2 -1  chr1  1
3_1  3  chr1  1
1_1 -1  chr1  1
2_1  2  chr1  1
```

Finally, our implementation also requires COSER files if we want to output the association of genes between genomes instead of only the approximate distance.

Chapter 4

Family-free DCJ similarity

In this chapter we are interested in the problem of computing the overall similarity of two given linear or circular multichromosomal genomes in a family-free setting under the DCJ model [104, 106]. This problem is called FFDCJ similarity. The complexity of computing the FFDCJ similarity was proven to be NP-hard [89], while the counterpart problem of computing the FFDCJ distance was already proven to be APX-hard [89].

In the remainder of this chapter, after preliminaries and a formal definition of the FFDCJ similarity problem, we first demonstrate its APX-hardness and an inapproximability result. We then present an exact ILP algorithm to solve it and four combinatorial heuristics, with computational experiments comparing their results for datasets simulated by a framework for genome evolution. Lastly, concluding remarks are discussed, followed by an appendix containing information on how simulated and real data were obtained for experiments.

4.1 Preliminaries

Here we present definitions related to the FFDCJ similarity and extend the notation introduced previously [89]. In the following sections, we consider two given genomes A and B, and denote by $\mathcal{A}$ the set of genes in genome A, and by $\mathcal{B}$ the set of genes in genome B. To contextualize the FFDCJ similarity, we begin presenting the DCJ similarity, a correlate problem under the family-based setting.

Family-Based DCJ Similarity and Adjacency Graph

In most versions of the family-based setting the two genomes A and B have the same content, that is, $\mathcal{A} = \mathcal{B}$. When in addition there are no duplicates, that is, when there is exactly one representative of each family in each genome, we can easily build the *adjacency graph* of genomes A and B, denoted by $AG(A, B)$ and described in details in the Section 3.1. Since the graph is bipartite and vertices have degree one or two, the adjacency graph is a collection of paths and even cycles. An example of an adjacency graph is presented in Fig. 4.1.

It is well known that a DCJ operation that modifies $AG(A, B)$ by increasing either the number of even cycles by one or the number of odd paths by two decreases the DCJ

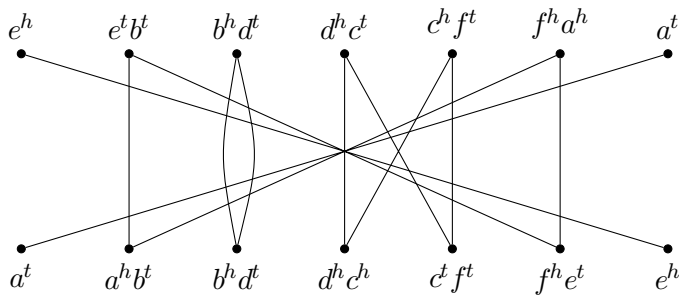


Figure 4.1: **Example of adjacency graph.** The genomes for $AG(A, B)$ are $A = \{(\circ\ \overline{e}\ b\ d\ c\ f\ \overline{a}\ \circ)\}$ and $B = \{(\circ\ a\ b\ d\ \overline{c}\ f\ e\ \circ)\}$.

distance between genomes A and B [15]. This type of DCJ operation is said to be *optimal*. Conversely, if we are interested in a DCJ similarity measure between A and B, rather than a distance measure, then it should be increased by such an optimal DCJ operation. This suggests that a formula for a DCJ similarity between two genomes should correlate to the number of connected components (in the following just *components*) of the corresponding adjacency graph.

When the genomes A and B are identical, their corresponding adjacency graph is a collection of c 2-cycles and b 1-paths [15], so that $c + \frac{b}{2} = |\mathcal{A}| = |\mathcal{B}|$. This should be the upper bound of our DCJ similarity measure, and the contribution of each component in the formula should be upper bounded by 1.

We know that an optimal operation can always be applied to adjacencies that belong to one of the two genomes and to one single component of $AG(A, B)$, until the graph becomes a collection of 2-cycles and 1-paths. In other words, each component of the graph can be *sorted*, that is, converted into a collection of 2-cycles and 1-paths independently of the other components. Furthermore, it is known that each of the following components—an even cycle with $2d + 2$ edges, or an odd path with $2d + 1$ edges, or an even path with $2d$ edges—can be sorted with exactly d optimal DCJ operations. Therefore, for the same d, components with more edges should actually have higher contributions in the DCJ similarity formula.

With all these considerations, the contribution of each component C in the formula is then defined to be its *normalized length* $\widehat{\ell}(C)$:

$$\widehat{\ell}(C) = \begin{cases} \dfrac{|C|}{|C|} = 1\,, & \text{if } C \text{ is a cycle}\,, \\ \dfrac{|C|}{|C|+1}\,, & \text{if } C \text{ is an odd path}\,, \\ \dfrac{|C|}{|C|+2}\,, & \text{if } C \text{ is an even path}\,. \end{cases}$$

Let $\mathcal{C}$ be the set of all components in $AG(A, B)$. The formula for the family-based DCJ similarity is the sum of their normalized lengths:

$$\mathrm{s}_{\mathrm{DCJ}}(A, B) = \sum_{C \in \mathcal{C}} \widehat{\ell}(C) \,. \tag{4.1}$$

Observe that $\mathrm{s}_{\mathrm{DCJ}}(A, B)$ is a positive value, indeed upper bounded by $|\mathcal{A}|$ (or, equivalently, by $|\mathcal{B}|$). In Fig. 4.1 the DCJ similarity is $\mathrm{s}_{\mathrm{DCJ}}(A, B) = 2 \cdot \frac{1}{2} + 3 \cdot 1 = 4$. The formula of Equation (4.1) is the family-based version of the family-free DCJ similarity defined by Martinez *et al.* [89], as we will see in the following subsections.

From the family-based to the family-free setting

In the family-free setting, each gene in each genome is represented by a unique (signed) symbol. In contrast to letters used in family-based methods, genes in family-free methods are usually labeled as $1, \ldots, |\mathcal{A}|$ in A and as $|\mathcal{A}| + 1, \ldots, |\mathcal{A}| + |\mathcal{B}|$ in B. Notice that $\mathcal{A} \cap \mathcal{B} = \emptyset$ and the cardinalities $|\mathcal{A}|$ and $|\mathcal{B}|$ may (and in fact very often will) be distinct. Moreover, as we label genes with numbers, we prefer $-g$ instead of $\overline{g}$ to represent a gene g with reverse orientation.

Additional concepts of the family-free DCJ similarity are presented in a timely manner in the following subsections.

Gene Similarity Graph

Let a be a gene in A and b be a gene in B, then their *normalized gene similarity* is given by some value $\sigma(a, b)$ such that $0 \leq \sigma(a, b) \leq 1$. Values closer to 1 mean gene sequences have high similarity and therefore genes are evolutionary "closer" to each other, while values closer to 0 mean genes are not related.

We can represent the gene similarities between the genes of genome A and the genes of genome B with respect to σ in the so called *gene similarity graph* [22], denoted by $GS_\sigma(A, B)$. This is a weighted bipartite graph whose partitions $\mathcal{A}$ and $\mathcal{B}$ are the sets of (signed) genes in genomes A and B, respectively. Furthermore, for each pair of genes (a, b) such that $a \in \mathcal{A}$ and $b \in \mathcal{B}$, if $\sigma(a, b) > 0$ then there is an edge e connecting a and b in $GS_\sigma(A, B)$ whose weight is $\sigma(e) := \sigma(a, b)$. An example of a gene similarity graph is given in Fig. 4.2.

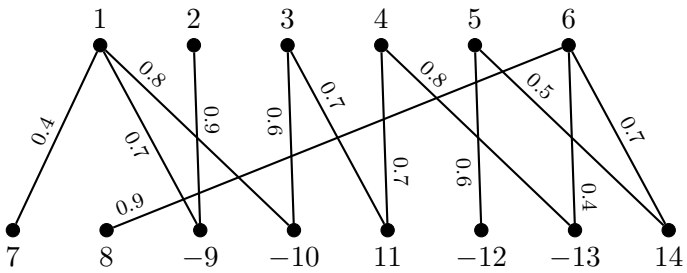


Figure 4.2: **Representation of a gene similarity graph.** Unichromosomal linear genomes for $GS_\sigma(A, B)$ are $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ -9\ -10\ 11\ -12\ -13\ 14\ \circ)\}$.

Weighted Adjacency Graph

The *weighted adjacency graph* $AG_\sigma(A, B)$ of two genomes A and B has a vertex for each adjacency in A and a vertex for each adjacency in B. For a gene a in A and a gene b in B with gene similarity $\sigma(a, b) > 0$ there is one edge e^h connecting the vertices containing the two heads a^h and b^h and one edge e^t connecting the vertices containing the two tails a^t and b^t. The weight of each of these edges is $\sigma(e^h) = \sigma(e^t) = \sigma(a, b)$. Differently from the simple adjacency graph, the weighted adjacency graph cannot be easily decomposed into cycles and paths, since its vertices can have degree greater than 2. As an example, the weighted adjacency graph corresponding to the gene similarity graph of Fig. 4.2 is given in Fig. 4.3.

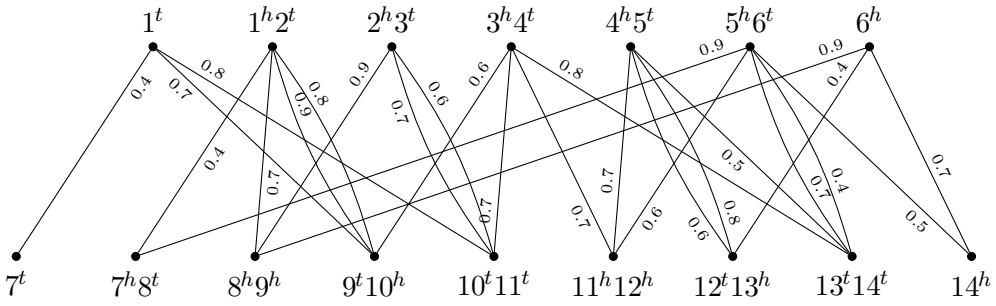


Figure 4.3: **Example of weighted adjacency graph.** The unichromosomal linear genomes for $AG_\sigma(A, B)$ are $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ {-9}\ {-10}\ 11\ {-12}\ {-13}\ 14\ \circ)\}$, and gene similarities are given in Fig. 4.2.

We denote by $w(G)$ the weight of a graph or subgraph G, that is given by the sum of the weights of all its edges, that is, $w(G) = \sum_{e \in G} \sigma(e)$. Observe that, for each edge $e \in GS_\sigma(A, B)$, we have two edges of weight $\sigma(e)$ in $AG_\sigma(A, B)$, thus, the total weight of the weighted adjacency graph is $w(AG_\sigma(A, B)) = 2\, w(GS_\sigma(A, B))$.

Reduced Genomes

Let A and B be two genomes and let $GS_\sigma(A, B)$ be their gene similarity graph. Now let $M = \{e_1, e_2, \ldots, e_n\}$ be a matching in $GS_\sigma(A, B)$ and denote by $w(M) = \sum_{e_i \in M} \sigma(e_i)$ the weight of M, that is, the sum of its edge weights. Since the endpoints of each edge $e_i = (a, b)$ in M are not saturated by any other edge of M, we can unambiguously define the function $\ell^M(a) = \ell^M(b) = i$ to relabel each vertex in A and B [89]. The *reduced genome* A^M is obtained by deleting from A all genes not saturated by M, and renaming each saturated gene a to $\ell^M(a)$, preserving its orientation (sign). Similarly, the reduced genome B^M is obtained by deleting from B all genes that are not saturated by M, and renaming each saturated gene b to $\ell^M(b)$, preserving its orientation. Observe that the set of genes in A^M and in B^M is $\mathcal{G}(M) = \{\ell^M(g) : g \text{ is saturated by the matching } M\} = \{1, 2, \ldots, n\}$.

Weighted Adjacency Graph of Reduced Genomes

Let A^M and B^M be the reduced genomes for a given matching M of $GS_\sigma(A, B)$. The weighted adjacency graph $AG_\sigma(A^M, B^M)$ can be obtained from $AG_\sigma(A, B)$ by deleting all edges that are not elements of M and relabeling the adjacencies according to ℓ^M. Vertices that have no connections are then also deleted from the graph. Another way to obtain the same graph is building the adjacency graph of A^M and B^M and adding weights to the edges as follows. For each gene i in $\mathcal{G}(M)$, both edges $i^t i^t$ and $i^h i^h$ inherit the weight of edge e_i in M, that is, $\sigma(i^t i^t) = \sigma(i^h i^h) = \sigma(e_i)$. Consequently, the graph $AG_\sigma(A^M, B^M)$ is also a collection of paths and even cycles and differs from $AG(A^M, B^M)$ only by the edge weights.

For each edge $e \in M$, we have two edges of weight $\sigma(e)$ in $AG_\sigma(A^M, B^M)$, therefore $w(AG_\sigma(A^M, B^M)) = 2\, w(M)$. Examples of weighted adjacency graphs of reduced genomes are shown in Fig. 4.4.

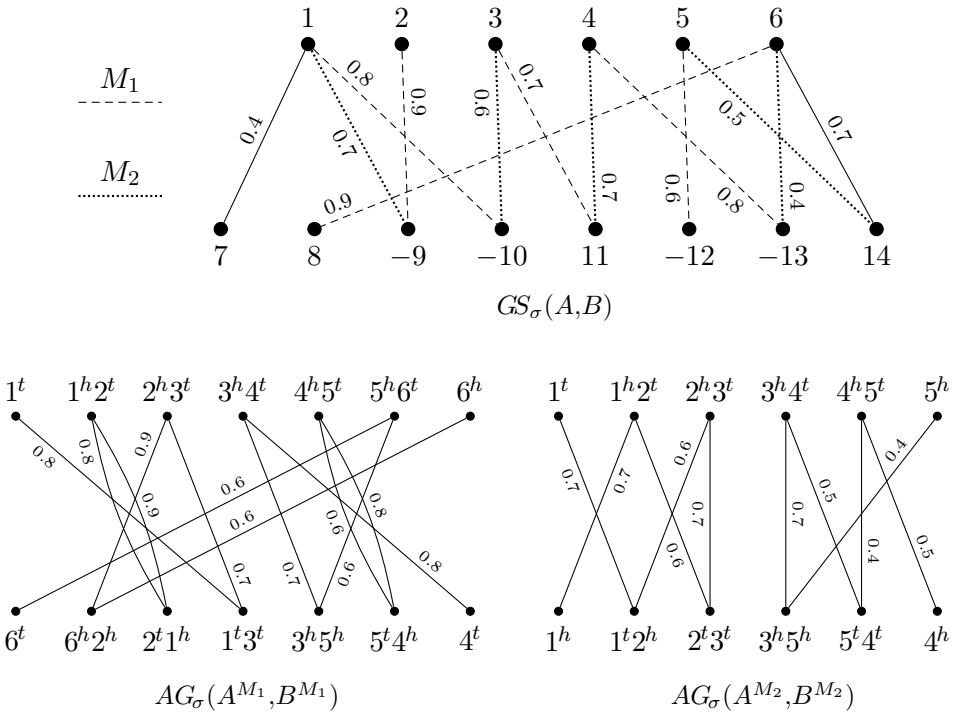


Figure 4.4: **Two distinct maximal matchings in a gene similarity graph inducing two different reduced genomes.** Considering, as in Fig. 4.2, the genomes $A = \{(\circ\ 1\ 2\ 3\ 4\ 5\ 6\ \circ)\}$ and $B = \{(\circ\ 7\ 8\ -9\ -10\ 11\ -12\ -13\ 14\ \circ)\}$, let M_1 (dashed edges) and M_2 (dotted edges) be two distinct maximal matchings in $GS_\sigma(A, B)$, shown in the upper part. The two resulting weighted adjacency graphs $AG_\sigma(A^{M_1}, B^{M_1})$, that has two cycles and two even paths, and $AG_\sigma(A^{M_2}, B^{M_2})$, that has two odd paths, are shown in the lower part.

The Family-Free DCJ Similarity

For a given matching M in $GS_\sigma(A, B)$, a first formula for the weighted DCJ (wDCJ) similarity s_σ of the reduced genomes A^M and B^M was proposed by Braga *et al.* [22] only considering the cycles of $AG_\sigma(A^M, B^M)$. After that, this definition was modified and extended [89] in order to consider all components of the weighted adjacency graph.

First, let the *normalized weight* $\widehat{w}(C)$ of a component C of $AG_\sigma(A^M, B^M)$ be:

$$\widehat{w}(C) = \begin{cases} \dfrac{w(C)}{|C|}\,, & \text{if } C \text{ is a cycle}\,, \\ \dfrac{w(C)}{|C|+1}\,, & \text{if } C \text{ is an odd path}\,, \\ \dfrac{w(C)}{|C|+2}\,, & \text{if } C \text{ is an even path}\,. \end{cases}$$

Let $\mathcal{C}$ be the set of all components in $AG_\sigma(A^M, B^M)$. Then the wDCJ similarity s_σ is given by the following formula [89]:

$$s_\sigma(A^M, B^M) = \sum_{C \in \mathcal{C}} \widehat{w}(C)\,. \tag{4.2}$$

Observe that, when the weights of all edges in M are equal to 1, this formula is equivalent to the one in Equation (4.1).

The goal now is to compute the family-free DCJ similarity, i.e., to find a matching in $GS_\sigma(A, B)$ that maximizes s_σ. However, although $s_\sigma(A^M, B^M)$ is a positive value upper bounded by $|M|$, the behaviour of the wDCJ similarity does not correlate with the size of the matching, since smaller matchings, that possibly discard gene assignments, can lead to higher wDCJ similarities [89]. For this reason, the wDCJ similarity function is restricted to *maximal matchings* only, ensuring that no pair of genes with positive gene similarity score is simply discarded, even though it might decrease the overall wDCJ similarity. We then have the following optimization problem:

> **Problem** FFDCJ-SIMILARITY(A, B): Given genomes A and B and their gene similarities σ, calculate their family-free DCJ similarity
>
> $$\mathrm{s}_{\text{FFDCJ}}(A, B) = \max_{M \in \mathbb{M}} \{s_\sigma(A^M, B^M)\}, \tag{4.3}$$
>
> where $\mathbb{M}$ is the set of all maximal matchings in $GS_\sigma(A, B)$.

Problem FFDCJ-SIMILARITY is NP-hard [89]. Moreover, one can directly correlate the problem to the adjacency similarity problem, where the goal is to maximize the number of preserved adjacencies between two given genomes [3, 33]. However, since there the objective is to maximize the number of cycles of length 2, even an approximation for the adjacency similarity problem is not a good algorithm for the FFDCJ-SIMILARITY problem, where cycles of higher lengths are possible in the solution [102].

Capping Telomeres

A very useful preprocessing to $AG_\sigma(A, B)$ is the *capping* of telomeres, a general technique for simplifying algorithms that handle genomes with linear chromosomes, commonly used in the context of family-based settings [65, 114, 134]. Given two genomes A and B with i and j linear chromosomes, respectively, for each vertex representing only one extremity we add a *null extremity* τ to it (e.g., 1^t of Fig. 4.4 becomes $\tau 1^t$). Furthermore, in order to add the same number of null extremities to both genomes, $|j - i|$ *null adjacencies* $\tau\tau$ (composed of two null extremities) are added to genome A, if $i < j$, or to genome B, if $j < i$. Finally, for each null extremity of a vertex in A we add to $AG_\sigma(A, B)$ a *null edge* with weight 0 to each null extremity of vertices in B. Consequently, after capping of telomeres the graph $AG_\sigma(A, B)$ has no vertex of degree one. Notice that, if before the capping p was a path of weight w connecting telomeres in $AG_\sigma(A, B)$, then after the capping p will be part of a cycle closed by null extremities with normalized weight $\frac{w}{|p|+1}$ if p is an odd path, or of normalized weight $\frac{w}{|p|+2}$ if p is an even path. In any of the two cases, the normalized weight is consistent with the wDCJ similarity formula in Equation (4.2).

4.2 APX-hardness and inapproximability

This section contains the APX-hardness proof of problem FFDCJ-SIMILARITY and a lower bound for the approximation ratio. Here we restrict ourselves to feasible solutions.

Consider the following optimization problem, to be used within the results of this section:

> **Problem** MAX-2SAT3(ϕ): Given a 2-CNF formula (i.e., with at most 2 literals per clause) $\phi = \{C_1, \ldots, C_m\}$ with n variables $X = \{x_1, \ldots, x_n\}$, where each variable appears in at most 3 clauses, find an assignment that satisfies the largest number of clauses.

The formula ϕ as defined above is called a 2SAT3 formula. MAX-2SAT3 [7, 19] is a special case of MAX-2SATk (also known as MAX-2SAT(k) or k-OCC-MAX-2SAT), where each variable occurs in at most k clauses for some fixed k, which in turn is a restricted version of MAX-2SAT [101]. Some authors consider restrict versions of SAT for El-CNF [69] formulae, which have *exactly* l literals, nevertheless, any hardness or inapproximability result for El-CNF is also for l-CNF. There are also other variants such as symmetric (l,Bk)-SAT or relaxed (l,k)-SAT instances, where clauses have exactly l literals and each literal or variable, respectively, occurs exactly k times [20].

To prove that FFDCJ-SIMILARITY is APX-hard, we present a strict reduction MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY in the form (f, g), showing that

$$R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) \leq R_{\text{FFDCJ-SIMILARITY}}(f(\phi), \gamma)\,,$$

for any instance ϕ of MAX-2SAT3 and solution γ of FFDCJ-SIMILARITY with instance $f(\phi)$. Since variables occurring only once imply their clauses and others to be trivially satisfied, we consider only clauses that are not trivially satisfied in their instance. Similar for clauses containing both literals x_i and $\overline{x_i}$, for some variable x_i.

Function f

We show progressively how the function builds $GS_\sigma(A, B)$ and defines genes and their sequences in chromosomes of A and B. For each variable x_i occurring three times, let Cx_i^1, Cx_i^2, and Cx_i^3 be *aliases* for the clauses where x_i occurs (notice that a clause composed of two literals has two aliases). We define a *variable component* $\mathcal{C}_i$ adding vertices (genes) x_i^1, x_i^2, and x_i^3 to $\mathcal{A}$, vertices (genes) Cx_i^1, Cx_i^2, and Cx_i^3 to $\mathcal{B}$, and edges $ex_i^j = (Cx_i^j, x_i^j)$ and $e\overline{x_i}^j = (Cx_i^j, x_i^k)$ for $j \in \{1, 2, 3\}$ and $k = (j+1) \bmod 3+1$ in $GS_\sigma(A, B)$. An edge ex_i^j ($e\overline{x_i}^j$) has weight 1 if the literal x_i ($\overline{x_i}$) belongs to the clause Cx_i^j, otherwise it has weight 0. Edges in the variable component $\mathcal{C}_i$ form a cycle of length 6 (Fig. 4.5) in $GS_\sigma(A, B)$. Variable components for variables occurring two times are defined similarly and form cycles of length 4. Genomes are $A = \{(x_i^j)$ for each occurrence j of each variable $x_i \in X\}$ and $B = \{(Cx_i^j) : Cx_i^j$ is an alias to a clause in ϕ with only one literal$\} \cup \{(Cx_i^j\ Cx_{i'}^{j'}) : Cx_i^j$ and $Cx_{i'}^{j'}$ are aliases to the same clause in $\phi\}$. (A and B have only circular chromosomes with 1 or 2 genes.)

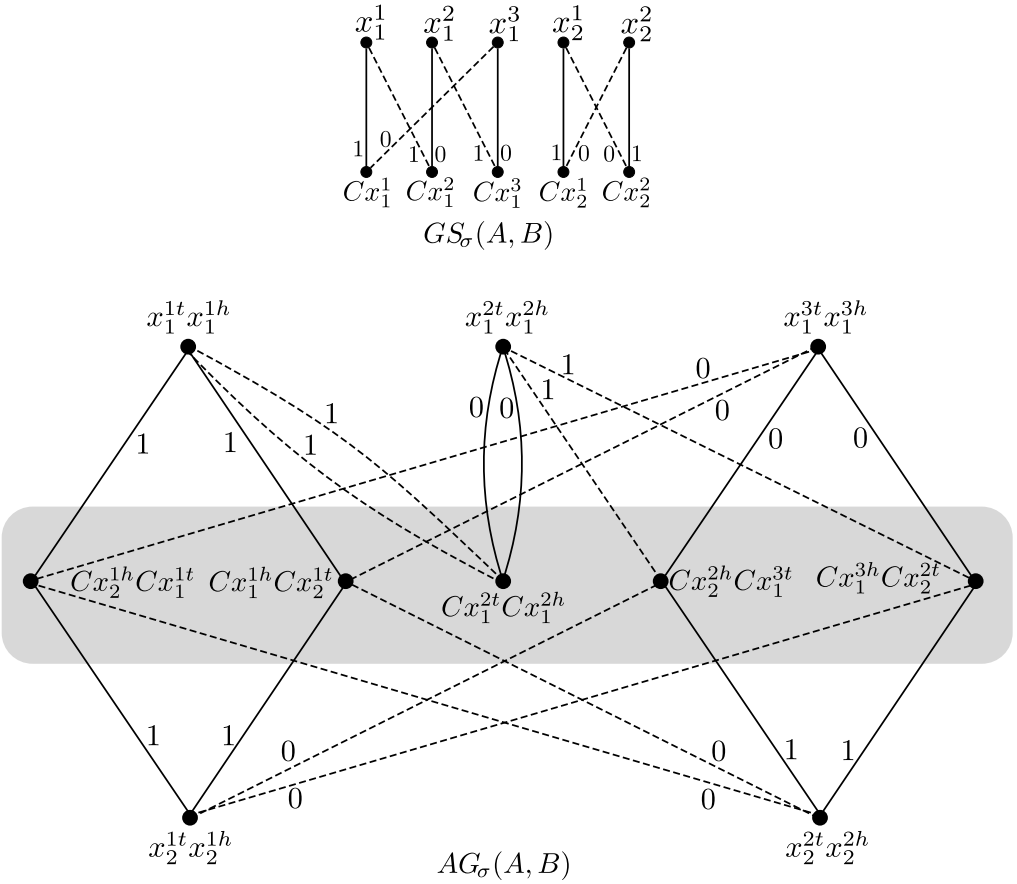


Figure 4.5: **Preliminary graphs of MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY.** $GS_\sigma(A, B)$ and $AG_\sigma(A, B)$ for genomes $A = \{(x_1^1), (x_1^2), (x_1^3), (x_2^1), (x_2^2)\}$ and $B = \{(Cx_1^1\ Cx_2^1), (Cx_1^2), (Cx_1^3\ Cx_2^2)\}$ given by function f applied to 2SAT3 clauses $C_1 = (x_1 \vee x_2)$, $C_2 = (\overline{x_1})$, and $C_3 = (\overline{x_1} \vee x_2)$. In $GS_\sigma(A, B)$, solid edges correspond to ex_i^j and dashed edges correspond to $e\overline{x_i}^j$. In $AG_\sigma(A, B)$, shaded region corresponds to genes of genome B and solid (dashed) edges correspond to solid (dashed) edges of $GS_\sigma(A, B)$.

The function f as defined here maps an instance ϕ of MAX-2SAT3 (a 2-CNF formula) to an instance $f(\phi)$ of FFDCJ-SIMILARITY (genomes A and B, and $GS_\sigma(A,B)$) and is clearly polynomial. Besides, since all chromosomes are circular, the corresponding weighted adjacency graph $AG_\sigma(A,B)$ (or $AG_\sigma(A^M,B^M)$ for some matching M) is a collection of cycles.

Now, notice that any maximal matching in $GS_\sigma(A,B)$ covers all genes in both $\mathcal{A}$ and $\mathcal{B}$, inducing in $AG_\sigma(A,B)$ only cycles of length 2, composed by (genes in) chromosomes (x_i^j) and $(Cx_i^{j'})$, or cycles of length 4, composed by chromosomes (x_i^j), (x_k^l), and $(Cx_i^{j'}\ Cx_k^{l'})$.

Recall that the normalized weight for a cycle C is $\widehat{w}(C) = \frac{w(C)}{|C|}$. In this transformation, each cycle C is such that $\widehat{w}(C) = 0, 0.5$, or 1. A cycle C such that $\widehat{w}(C) > 0$ is a *helpful cycle* and represents a clause satisfied by one or two literals ($\widehat{w}(C) = 0.5$ or $\widehat{w}(C) = 1$, respectively). See an example in Fig. 4.6.

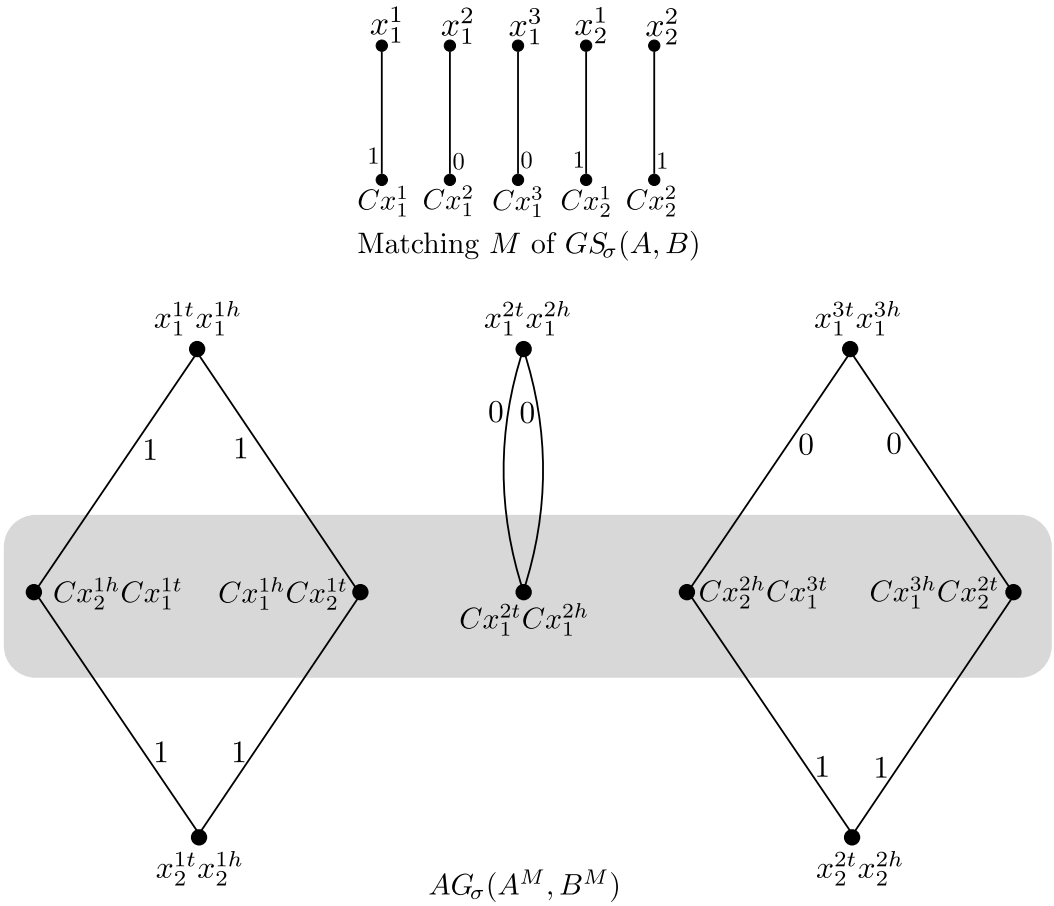
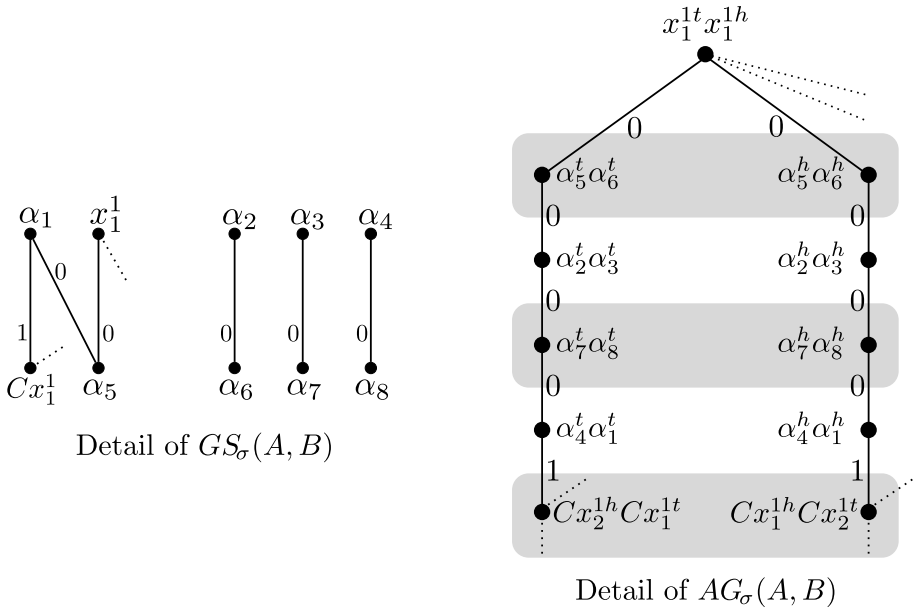


Figure 4.6: **Example of the relation between a solution of FFDCJ-SIMILARITY using preliminary graphs and a solution of MAX-2SAT3 in MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY.** A matching M of $GS_\sigma(A,B)$ and cycles induced by M in $AG_\sigma(A^M,B^M)$ for genomes of Fig. 4.5. This solution of FFDCJ-SIMILARITY represents clauses C_1 and C_3 of MAX-2SAT3 satisfied.

In this scenario, however, a solution of FFDCJ-SIMILARITY with performance ratio r could lead to a solution of MAX-2SAT3 with ratio $2r$, since the total normalized weight

for two cycles C_1 and C_2 with $\widehat{w}(C_1) = \widehat{w}(C_2) = 0.5$ (two clauses satisfied by one literal each) is the same for one cycle C with $\widehat{w}(C) = 1.0$ (one clause satisfied by two literals). Therefore, achieving the desired ratio requires some modifications in f. It is not possible to make these two types of cycles have the same weight, but it suffices to get close enough.

We introduce special genes into the genomes called *extenders*. For some p even, for each edge $ex_i^j = (Cx_i^j, x_i^j)$ of weight 1 in $GS_\sigma(A, B)$ we introduce p extenders $\alpha_1, \ldots, \alpha_p$ into A (as a consequence, they are also introduced into $\mathcal{A}$) and p extenders $\alpha_{p+1}, \ldots, \alpha_{2p}$ into B. Each ex_i^j of weight 1 has its own set of extenders, and the same process is done for each $e\overline{x_i}^j$ of weight 1. Edge ex_i^j is replaced by edges (Cx_i^j, α_1) with weight 1 (which we consider equivalent to ex_i^j) and (α_{p+1}, x_i^j) with weight 0, and edges (α_k, α_{p+k}) with weight 0 are added to $GS_\sigma(A, B)$ for each k, $1 \leq k \leq p$ (extenders α_1 and α_{p+1} are now part of the variable component $\mathcal{C}_i$). Regarding new chromosomes in genomes A and B, A is updated to $A \cup \{(\alpha_1\ -\alpha_p)\} \cup \{(\alpha_k\ -\alpha_{k+1}) : k \in \{2, 4, \ldots, p-2\}\}$ and B to $B \cup \{(\alpha_k\ -\alpha_{k+1}) : k \in \{p+1, p+3, \ldots, 2p-1\}\}$. By this construction, which is still polynomial, the path from x_i^{jt} to Cx_i^{jt} in $AG_\sigma(A, B)$ is extended from 1 to $1+p$ edges, from $\{(x_i^{jt}, Cx_i^{jt})\}$ to $\{(x_i^{jt}, \alpha_p^t), (\alpha_{p+1}^t, \alpha_2^t), (\alpha_3^t, \alpha_{p+2}^t), (\alpha_{p+3}^t, \alpha_4^t), \ldots, (\alpha_1^t, Cx_i^{jt})\}$. The same occurs for the path from x_i^{jh} to Cx_i^{jh} (see Fig. 4.7). Now, cycles in $AG_\sigma(A, B)$ induced by edges of weight 0 in $GS_\sigma(A, B)$ have normalized weight 0, cycles previously with normalized weight 1 are extended and have normalized weight $\frac{1}{1+p}$, and cycles previously with normalized weight 0.5 are extended and have normalized weight $\frac{1}{2+p}$. Notice that, for a sufficiently large p, $\frac{1}{1+p}$ is quite close to $\frac{1}{2+p}$, hence the problem of finding the maximum similarity in this graph is very similar to finding the maximum number of helpful cycles.



Detail of $GS_\sigma(A, B)$

Detail of $AG_\sigma(A, B)$

Figure 4.7: **Final graphs of MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY including extender genes.** Detail of graphs $GS_\sigma(A, B)$ and $AG_\sigma(A, B)$ for genomes of Fig. 4.5 including extenders for edge (x_1^1, Cx_1^1) for $p = 4$. Shaded regions correspond to genes of genome B. Extending all edges of weight 1 and selecting the matching of Fig. 4.6, this helpful cycle (only half of it is in this figure) would have normalized weight $\frac{4}{4(p+1)} = \frac{1}{p+1} = \frac{1}{5} = 0.2$.

Function g

By the structure of variable components in $GS_\sigma(A, B)$, and since solutions of FFDCJ-SIMILARITY are restricted to maximal matchings only, any solution γ for $f(\phi)$ is a matching that covers only edges ex_i^j or $e\overline{x_i}^j$ for each variable component $\mathcal{C}_i$. For a $\mathcal{C}_i$, if edges ex_i^j ($e\overline{x_i}^j$) are in the solution then the variable x_i is assigned to true (false), inducing in polynomial time an assignment for each $x_i \in X$ and therefore a solution $g(f(\phi), \gamma)$ to MAX-2SAT3. A clause is satisfied if vertices (or the only vertex) corresponding to its aliases are in a helpful cycle.

Ratio and final steps

The next step to obtain the strict reduction is showing that $R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) \leq R_{\text{FFDCJ-SIMILARITY}}(f(\phi), \gamma)$. In order to do so, we must set some parameters to specific values and establish some properties and relations between MAX-2SAT3 and FFDCJ-SIMILARITY.

Given $f(\phi)$ and a feasible solution γ of FFDCJ-SIMILARITY with the maximum number of helpful cycles, denote by c' the number of helpful cycles in γ. Notice that c' is also the maximum number of satisfied clauses of MAX-2SAT3, that is, the value of an optimal solution for MAX-2SAT3 for any instance ϕ, denoted here by $opt_{\text{2SAT3}}(\phi)$. Thus, $c' = opt_{\text{2SAT3}}(\phi)$.

Let $n := |A| = |B|$ before extenders are added. We choose for p (the number of extenders added for each edge of weight 1 in $GS_\sigma(A, B)$) the value $2n$ and define

$$\omega = \frac{1}{2+p} = \frac{1}{2+2n}$$

and

$$\varepsilon = \frac{1}{1+p} - \frac{1}{2+p} = \frac{1}{4n^2+6n+2}\,,$$

which implies that $\omega + \varepsilon = \frac{1}{p+1}$. Thus, it is easy to see that $n\varepsilon < \omega$, i.e.,

$$\varepsilon < \frac{\omega}{n} < 1\,. \tag{4.4}$$

If $opt_{\text{SIM}}(f(\phi))$ denotes the value of an optimal solution for FFDCJ-SIMILARITY with instance $f(\phi)$ and c^* denotes the number of helpful cycles in an optimal solution of FFDCJ-SIMILARITY, then we have immediately that

$$\frac{opt_{\text{SIM}}(f(\phi))}{\omega+\varepsilon} \leq c^* \leq \frac{opt_{\text{SIM}}(f(\phi))}{\omega}\,. \tag{4.5}$$

Besides that

$$0 \leq c^* \leq n\,, \tag{4.6}$$

and

$$c^*\omega \leq opt_{\text{SIM}}(f(\phi)) \leq c^*(\omega+\varepsilon)\,. \tag{4.7}$$

Thus, we have

$$\begin{aligned} c^*(\omega+\varepsilon) &= c^*\omega + c^*\varepsilon \\ &< c^*\omega + \frac{c^*\omega}{n} && (4.8)\\ &\leq c^*\omega + 1\cdot\omega && (4.9)\\ &= c^*\omega + \omega\,, && (4.10)\end{aligned}$$

where (4.8) comes from (4.4) and (4.9) is valid due to (4.6). Given the definitions above, we are able to demonstrate some propositions related to the ratios of the reduction.

Proposition 4.1 *Let c' be the number of helpful cycles in a feasible solution of* FFDCJ-SIMILARITY *with the greatest number of helpful cycles possible. Let c^* be the number of helpful cycles in an optimal solution of* FFDCJ-SIMILARITY*. Then,*

$$c' = c^*\,.$$

Proof. Since c' is the greatest number of helpful cycles possible, it is immediate that $c^* \leq c'$.

Let us now show that $c^* \geq c'$. Suppose for a moment that $c^* < c'$. Since c^* and c' are integers, this implies that $c^* + 1 \leq c'$, i.e.,

$$c^* \leq c' - 1\,. \tag{4.11}$$

Let $\mathcal{C}'$ be the set of cycles with c' cycles, i.e., with the maximum number of helpful cycles possible. Let $\widehat{w}(\mathcal{C}') := \sum_{C\in\mathcal{C}'} \widehat{w}(C) = \sum_{C\in\mathcal{C}'} w(C)/|C|$. Then

$$\begin{aligned} \widehat{w}(\mathcal{C}') \geq c'\omega &= (c'-1)\omega + \omega \\ &\geq c^*\omega + \omega && (4.12)\\ &> c^*(\omega+\varepsilon) && (4.13)\\ &\geq opt_{\text{SIM}}(f(\phi))\,, && (4.14)\end{aligned}$$

where (4.12) follows from (4.11), (4.13) comes from (4.10), and (4.14) is valid due to (4.7). It means that $\widehat{w}(\mathcal{C}') > opt_{\text{SIM}}(f(\phi))$, which is a contradiction.

Therefore, $c' = c^*$. □

Proposition 4.2 *Let c^r be the number of helpful cycles given by an approximate solution for* FFDCJ-SIMILARITY *with approximation ratio r. Let c' be the same as defined in Proposition 4.1. Then,*

$$c^r \geq \frac{c'}{r}\,.$$

Proof. Given an instance $f(\phi)$ of FFDCJ-SIMILARITY, let γ^r be an approximate solution of $f(\phi)$ with performance ratio r, i.e., $val(f(\phi), \gamma^r) \geq \frac{opt_{\text{SIM}}(f(\phi))}{r}$. Let c^r be the number of

helpful cycles of γ^r. Then

$$\begin{aligned} c^r &\geq \frac{\left(\frac{opt_{\text{SIM}}(f(\phi))}{r}\right)}{\omega+\epsilon} \\ &> \frac{opt_{\text{SIM}}(f(\phi))}{r(\omega+\omega/n)} && (4.15) \\ &= \frac{opt_{\text{SIM}}(f(\phi))}{r\omega} \cdot \frac{n}{n+1} \\ &\geq \frac{c'\omega}{r\omega} \cdot \frac{n}{n+1} && (4.16) \\ &= \frac{c'}{r} \cdot \left(1 - \frac{1}{n+1}\right) \\ &= \frac{c'}{r} - \frac{c'}{r(n+1)} \\ &\geq \frac{c'}{r} - 1\,, && (4.17) \end{aligned}$$

where (4.15) follows from (4.4), (4.16) is valid from (4.7) and Proposition 4.1, and (4.17) is true because $r \geq 1$ and $c' = c^* \leq n$. Then, from (4.17) we know that $c^r > \frac{c'}{r} - 1$ and, since c^r is an integer number, the result follows. □

From the results above, it is possibile to establish the desired ratio for the strict reduction.

Lemma 4.3 *Let c^r be the number of helpful cycles given by an approximate solution for the* FFDCJ-SIMILARITY *with approximation ratio r. Then, $R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) \leq r$.*

Proof. Directly from the result of Proposition 4.2,

$$R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) = \frac{opt_{\text{2SAT3}}(\phi)}{c^r} = \frac{c'}{c^r} \leq r\,,$$

□

We now present the main result of this section.

Theorem 4.4 FFDCJ-SIMILARITY *is APX-hard.*

Proof. The statement follows immediately from the strict reduction MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY given by the functions (f, g), presented previously, and the relation of ratios $R_{\text{MAX-2SAT3}}(\phi, g(f(\phi), \gamma)) \leq R_{\text{FFDCJ-SIMILARITY}}(f(\phi), \gamma)$, given by Lemma 4.3. □

This theorem also leads us directly to the first inapproximability result.

Corollary 4.5 FFDCJ-SIMILARITY *cannot be approximated by a ratio of* $2012/2011 - \varepsilon$ *for any $\varepsilon > 0$, unless $P = NP$.*

Proof. First, notice that if a problem is APX-hard, the existence of a PTAS for it implies P = NP. Since a strict reduction preserves membership in the class PTAS, finding a PTAS for FFDCJ-SIMILARITY implies a PTAS for every APX-hard problem and P = NP. A PTAS for FFDCJ-SIMILARITY would also imply an approximation ratio better than $2012/2011 = 1.0005\ldots$, unless P = NP. This follows immediately from the reduction in

Theorem 4.4 with $R_{\text{MAX-2SAT3}} = R_{\text{FFDCJ-SIMILARITY}}$ and the fact that MAX-2SAT3 is shown [19] to be NP-hard to approximate within a ratio of $2012/2011 - \varepsilon$ for any $\varepsilon > 0$. □

However, there is a slightly stronger result.

Corollary 4.6 FFDCJ-SIMILARITY *cannot be approximated with approximation ratio better than* $22/21 = 1.0476\ldots$*, unless* $P = NP$*.*

Proof. Notice particularly that the reduction MAX-2SAT3 $\leq_{\text{strict}}$ FFDCJ-SIMILARITY can be trivially extended to MAX-2SAT $\leq_{\text{strict}}$ FFDCJ-SIMILARITY by extending variable components to arbitrary sizes. This increases the lower bound to $22/21 = 1.0476\ldots$ [69]. □

4.3 An Exact Algorithm

In order to exactly compute the family-free DCJ similarity between two given genomes, we propose an integer linear program (ILP) formulation that is similar to the one for the family-free DCJ distance [89]. It adopts the same notation and also uses an approach to solve the maximum cycle decomposition problem defined by Shao, Lin and Moret [115].

Let A and B be two genomes, let $G = GS_\sigma(A, B)$ be their gene similarity graph, and let X_A and X_B be the extremity sets (including null extremities) with respect to A and B for the capped adjacency graph $AG_\sigma(A, B)$, respectively. The weight $w(e)$ of an edge e in G is also denoted by w_e. For the ILP formulation, an extension $H = (V_H, E_H)$ of the capped weighted adjacency graph $AG_\sigma(A, B)$ is defined such that $V_H = X_A \cup X_B$, and $E_H = E_m \cup E_a \cup E_s$ has three types of edges: (*i*) *matching edges* that connect two extremities in different extremity sets, one in X_A and the other in X_B, if they are null extremities or there exists an edge connecting these genes in G; the set of matching edges is denoted by E_m; (*ii*) *adjacency edges* that connect two extremities in the same extremity set if they form an adjacency; the set of adjacency edges is denoted by E_a; and (*iii*) *self edges* that connect two extremities of the same gene in an extremity set; the set of self edges is denoted by E_s. Matching edges have weights defined by the normalized gene similarity σ, all adjacency and self edges have weight 0. Notice that any edge in G corresponds to two matching edges in H.

The description of the ILP follows. For each edge e in H, we create a binary variable x_e to indicate whether e will be in the final solution. We require first that each adjacency edge be chosen:

$$x_e = 1, \qquad \forall\ e \in E_a.$$

Now we rename each vertex in H such that $V_H = \{v_1, v_2, \ldots, v_k\}$ with $k = |V_H|$. We require that each of these vertices be adjacent to exactly one matching or self edge:

$$\sum_{e=v_r v_t \in E_m \cup E_s} x_e = 1, \forall\ v_r \in X_A, \quad \text{and} \quad \sum_{e=v_r v_t \in E_m \cup E_s} x_e = 1, \forall\ v_t \in X_B.$$

Then, we require that the final solution be valid, meaning that if one extremity of a gene in A is assigned to an extremity of a gene in B, then the other extremities of these two genes have to be assigned as well:

$$x_{a^h b^h} = x_{a^t b^t}, \qquad \forall\ ab \in E_G.$$

We also require that the matching be maximal. This can easily be ensured if we guarantee that at least one of the vertices connected by an edge in the gene similarity graph be chosen, which is equivalent to not allowing both of the corresponding self edges in the weighted adjacency graph be chosen:

$$x_{a^h a^t} + x_{b^h b^t} \leq 1, \qquad \forall\ ab \in E_G.$$

To count the number of cycles, we use the same strategy as described by Shao, Lin and Moret [115]. For each vertex v_i we define a variable y_i that labels v_i such that

$$0 \leq y_i \leq i, \qquad 1 \leq i \leq k.$$

We also require that adjacent vertices have the same label, forcing all vertices in the same cycle to have the same label:

$$y_i \leq y_j + i \cdot (1 - x_e),\ \forall\ e = v_i v_j \in E_H,$$
$$y_j \leq y_i + j \cdot (1 - x_e),\ \forall\ e = v_i v_j \in E_H.$$

We create a binary variable z_i, for each vertex v_i, to verify whether y_i is equal to its upper bound i:

$$i \cdot z_i \leq y_i, \qquad 1 \leq i \leq k.$$

Since all variables y_i in the same cycle have the same label but a different upper bound, only one of the y_i can be equal to its upper bound i. This means that z_i is 1 if the cycle with vertex i as representative is used in a solution.

Now, let $L = \{2j : j = 1, \ldots, n\}$ be the set of possible cycle lengths in H, where $n := \min(|A|, |B|)$. We create the binary variable x_{ei} to indicate whether e is in i, for each $e \in E_H$ and each cycle i. We also create the binary variable x_{ei}^{ℓ} to indicate whether e belongs to i and the length of cycle i is ℓ, for each $e \in E_H$, each cycle i, and each $\ell \in L$.

We require that if an edge e belongs to a cycle i, then it can be true for only one length $\ell \in L$. Thus,

$$\sum_{\ell \in L} x_{ei}^{\ell} \leq x_{ei}, \qquad \forall\ e \in E_H \text{ and } 1 \leq i \leq k. \tag{4.18}$$

We create another binary variable z_i^{ℓ} to indicate whether cycle i has length ℓ. Then $\ell \cdot z_i^{\ell}$ is an upper bound for the total number of edges in cycle i of length ℓ:

$$\sum_{e \in E_M} x_{ei}^{\ell} \leq \ell \cdot z_i^{\ell}, \qquad \forall\ \ell \in L \text{ and } 1 \leq i \leq k.$$

The length of a cycle i is given by $\ell \cdot z_i^{\ell}$, for $i = 1, \ldots, k$ and $\ell \in L$. On the other hand, it is the total amount of matching edges e in cycle i. That is,

$$\sum_{\ell \in L} \ell \cdot z_i^{\ell} = \sum_{e \in E_m} x_{ei}, \qquad 1 \leq i \leq k.$$

We have to ensure that each cycle i must have just one length:

$$\sum_{\ell \in L} z_i^{\ell} = z_i, \qquad 1 \leq i \leq k.$$

Now we create the binary variable y_{ri} to indicate whether the vertex v_r is in cycle i. Thus, if $x_{ei} = 1$, i.e., if the edge $e = v_r v_t$ in H is chosen in cycle i, then $y_{ri} = 1 = y_{ti}$ (and $x_e = 1$ as well). Hence,

$$\left.\begin{array}{rcl} x_{ei} & \leq & x_e, \\ x_{ei} & \leq & y_{ri}, \\ x_{ei} & \leq & y_{ti}, \\ x_{ei} & \geq & x_e + y_{ri} + y_{ti} - 2, \end{array}\right\} \quad \forall\ e = v_r v_t \in E_H \text{ and } 1 \leq i \leq k. \tag{4.19}$$

Since y_r is an integer variable, we associate y_r to the corresponding binary variable y_{ri}, for any vertex v_r belonging to cycle i:

$$y_r = \sum_{i=1}^{r} i \cdot y_{ri}, \qquad \forall\ v_r \in V_H.$$

Furthermore, we must ensure that each vertex v_r may belong to at most one cycle:

$$\sum_{i=1}^{r} y_{ri} \leq 1, \qquad \forall\ v_r \in V_H.$$

Finally, we set the objective function as follows:

$$\text{maximize} \quad \sum_{i=1}^{k} \sum_{\ell \in L} \sum_{e \in E_m} \frac{w_e x_{ei}^{\ell}}{\ell}.$$

Note that, with this formulation, we do not have any path as a component. Therefore, the objective function above is exactly the family-free DCJ similarity $\mathrm{s}_{\mathrm{FFDCJ}}(A, B)$ as defined in Equations (4.2) and (4.3).

Notice that the ILP formulation has $O(N^4)$ variables and $O(N^3)$ constraints, where $N = |A| + |B|$. The number of variables is proportional to the number of variables x_{ei}^{ℓ}, and the number of constraints is upper bounded by (4.18) and (4.19).

4.4 Heuristics

We now propose four heuristic algorithms to compute the family-free DCJ similarity of two given genomes: one which is directly derived from a maximum matching of the gene similarity graph GS_σ and three greedy-like heuristics that, according to different criteria, select cycles from the weighted adjacency graph AG_σ, such that the cycles selected by each heuristic induce a matching in GS_σ.

Maximum Matching

In the first heuristic, shown in Algorithm 4.1 (Maximum-Matching), we find a maximum weighted bipartite matching M in GS_σ by the Hungarian Method, also known as Kuhn-Munkres Algorithm [49, 92, 127]. Given the matching M, it is straightforward to obtain the reduced genomes A^M and B^M and return the similarity value $s_\sigma(A^M, B^M)$.

Algorithm 4.1 MAXIMUM-MATCHING(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: Build the gene similarity graph $GS_\sigma(A, B)$.
2: Obtain a maximum weighted matching M in $GS_\sigma(A, B)$ defining reduced genomes A^M and B^M.
3: Build the capped weighted adjacency graph $AG_\sigma(A^M, B^M)$ of the reduced genomes.
4: Let $\mathcal{C}$ be the set of all cycles in $AG_\sigma(A^M, B^M)$.
5: Return $s_\sigma(A^M, B^M) = \sum_{C \in \mathcal{C}} \widehat{w}(C)$.

For the implementantion of this heuristic we cast similarity values (floating point edge weights in $[0, 1]$) in $GS_\sigma(A, B)$ to integers by multiplying them by some power of ten, depending on the precision of similarity values. Given real or general simulated instances, and for a power of ten large enough, this operation has little impact on the optimality of the weighted matching M for the original weights in $GS_\sigma(A, B)$ obtained from the Kuhn-Munkres algorithm, i.e., the weight of M for the original weights in $GS_\sigma(A, B)$ is optimal or near-optimal since only less significant digits are not considered.

Greedy Heuristics

Before describing the greedy heuristics, we need to introduce the following concepts. We say that two edges in $AG_\sigma(A, B)$ are *compatible* if one connects the head and the other connects the tail of the same pair of genes, or if they connect extremities of distinct genes in both genomes. Otherwise they are *incompatible*. A set of edges, in particular a cycle, is *consistent* if it has no pair of incompatible edges. A set of cycles is consistent if the union of all of their edges is consistent. Observe that a consistent set of cycles in $AG_\sigma(A, B)$ induces a matching in $GS_\sigma(A, B)$.

Each one of the three greedy algorithms selects disjoint and consistent cycles in the capped $AG_\sigma(A, B)$. Consistent cycles are selected from the set of all cycles of $AG_\sigma(A, B)$, that is obtained in Step 4 of each heuristic (see Algorithms 4.2, 4.3 and 4.4 below), using a cycle enumeration algorithm by Hawick and James [70], which is based on Johnson's algorithm [80]. For this reason, the running time of our heuristics is potentially exponential in the number of vertices of $AG_\sigma(A, B)$.

In the three heuristics, after completing the cycle selection by iterating over the set of all cycles of $AG_\sigma(A, B)$, the induced matching M in $GS_\sigma(A, B)$ could still be non-maximal. Whenever this occurs, among the genes that are unsaturated by M, we can identify *disposable genes* by one of the two following conditions:

1. Any unsaturated gene in $GS_\sigma(A, B)$ that is connected only to saturated genes, is a disposable gene;

2. For a given set of vertices $S \subseteq \mathcal{A}$ (or $S \subseteq \mathcal{B}$) in $GS_\sigma(A, B)$ such that, for the set of connected genes $\mathrm{N}(S)$, we have $|S| > |\mathrm{N}(S)|$ (Hall's theorem), then any subset of size $|S| - |\mathrm{N}(S)|$ of unsaturated genes of S can be set as disposable genes. In our implementation we choose those $|S| - |\mathrm{N}(S)|$ unsaturated genes with the smallest labels. Such $S \subseteq \mathcal{A}$ can be found as follows. Let V be the set of vertices saturated

by M, and let M' be a maximum cardinality matching in $GS_\sigma(A,B) \setminus V$. Consider the sets $\mathcal{A}' = \mathcal{A} \setminus V$ and $\mathcal{B}' = \mathcal{B} \setminus V$. Now let $GS'_\sigma(A,B)$ be a directed bipartite graph on the vertex set $\mathcal{A}' \cup \mathcal{B}'$, which includes the edges of M' oriented from $\mathcal{B}'$ to $\mathcal{A}'$ and the remaining edges of $GS_\sigma(A,B) \setminus V$ oriented from $\mathcal{A}'$ to $\mathcal{B}'$, and let $U \subseteq \mathcal{A}'$ be the set of vertices of $\mathcal{A}'$ unsaturated by M'. $S \subseteq \mathcal{A}$ is the corresponding set of vertices reachable from U in $GS'_\sigma(A,B)$, if any. $S \subseteq \mathcal{B}$ can be found analogously.

If there is no consistent cycle to be selected and the matching M is still non-maximal, new consistent cycles appear in $AG_\sigma(A,B)$ after the deletion of all identified disposable genes (see Fig. 4.8). In order to delete a disposable gene g, we need to remove from $AG_\sigma(A,B)$ the edges corresponding to extremities g^t or g^h and "merge" the two vertices that represent these extremities. Every time disposable genes are deleted from $AG_\sigma(A,B)$, a new iteration of the algorithms starts from Step 4 (see again Algorithms 4.2, 4.3 and 4.4). This procedure assures that, in each one of the three algorithms, the final set of selected cycles defines a maximal matching M, such that $AG_\sigma(A^M, B^M)$ is exactly the union of those selected cycles.

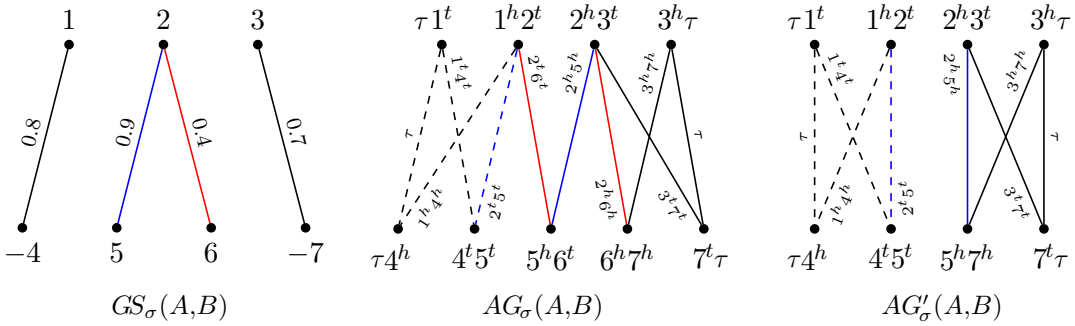


Figure 4.8: **Example of disposable genes.** Consider genomes $A = \{(\circ\ 1\ 2\ 3\ \circ)\}$ and $B = \{(\circ\ -4\ 5\ 6\ -7\ \circ)\}$ and their gene similarity graph $GS_\sigma(A,B)$. The selection of the dashed cycle in $AG_\sigma(A,B)$ adds the edges connecting gene 1 to gene 4 and gene 2 to gene 5 to the matching M in $GS_\sigma(A,B)$. After this selection, although the matching M is not yet maximal, there are no more consistent cycles in $AG_\sigma(A,B)$. Observe that gene 6 in $GS_\sigma(A,B)$ is unsaturated and its single neighbor—gene 2—is already saturated. Since gene 6 can no longer be saturated by M, it is a disposable gene and is deleted from $AG_\sigma(A,B)$, resulting in $AG'_\sigma(A,B)$, where a new consistent cycle appears. The selection of this new cycle adds to the matching M the edge connecting gene 3 to gene 7. Both $AG_\sigma(A,B)$ and $AG'_\sigma(A,B)$ have a simplified representation, in which the edge weights, as well as two of the four null edges of the capping, are omitted. Furthermore, for the sake of clarity, in this simplified representation each edge has a label describing the extremities connected by it.

Best Density

The best density heuristic is shown in Algorithm 4.2 (Greedy-Density). The *density* of a cycle C is given by $\frac{w(C)}{|C|^2}$ (its weight divided by the square of its length). The cycles of $AG_\sigma(A,B)$ are arranged in decreasing order of their densities, and consistent cycles are selected following this order.

Algorithm 4.2 GREEDY-DENSITY(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: **for** $\ell = 10, 20, \ldots,$ maximum cycle length possible **do**
5: List all cycles of $AG_\sigma(A, B)$ of length at most ℓ in decreasing order of their densities.
6: While it is possible, select the best density consistent cycle C that is also consistent with all cycles in $\mathcal{C}$ and add it to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \setminus C$, update M by adding the new gene connections induced by C.
7: If M is not a maximal matching of $GS_\sigma(A, B)$, find and delete disposable genes from $AG_\sigma(A, B)$ and go back to Step 4.
8: Return $\sum_{C \in \mathcal{C}} \widehat{w}(C)$.

Since the number of cycles of any length may be exponential in the size of the input graph, in our implementation we add a heuristic in which initially the search is restricted to cycles of length up to ten. Then, as long as the obtained matching is not maximal, Steps 4 to 7 are repeated, while gradually increasing the allowed maximum cycle length in steps of ten.

Best Length

The best length heuristic is shown in Algorithm 4.3 (GREEDY-LENGTH). The cycles of $AG_\sigma(A, B)$ are found in increasing order of their lengths, and ties are broken by the decreasing order of their weights. Here we first find and select cycles of length 2, then of length 4, and so on, for each fixed length iterating over the set of all cycles in decreasing order of their weights. Consistent cycles are selected following this procedure.

Algorithm 4.3 GREEDY-LENGTH(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: **for** $\ell = 2, 4, \ldots,$ maximum cycle length possible **do**
5: List all cycles of $AG_\sigma(A, B)$ of length ℓ in decreasing order of their weights.
6: While it is possible, select the best weight consistent cycle C that is also consistent with all cycles in $\mathcal{C}$ and add it to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \setminus C$, update M by adding the new gene connections induced by C.
7: If M is not a maximal matching of $GS_\sigma(A, B)$, find and delete disposable genes from $AG_\sigma(A, B)$ and go back to Step 4.
8: Return $\sum_{C \in \mathcal{C}} \widehat{w}(C)$.

Best Length with Weighted Maximum Independent Set

The best length heuristic with WMIS is shown in Algorithm 4.4 (GREEDY-WMIS) and is a variation of GREEDY-LENGTH. Instead of selecting cycles of greater weights for a fixed length, this algorithm selects the greatest amount of cycles for a fixed length by a WMIS algorithm. The heuristic builds a *cycle graph* where each vertex is a cycle of $AG_\sigma(A, B)$, the weight of a vertex is the weight of the cycle it represents and two vertices are adjacent if the cycles they represent are inconsistent. The heuristic tries to find next an independent set with the greatest weight in the cycle graph. Since this graph is not d-claw-free for any fixed d, the WMIS algorithm [17, 18] does not guarantee any fixed approximation ratio.

Algorithm 4.4 GREEDY-WMIS(A, B, σ)

Input: genomes A and B, gene similarity function σ
Output: a family-free DCJ similarity between A and B

1: $M := \emptyset$; $\mathcal{C} := \emptyset$.
2: Build the gene similarity graph $GS_\sigma(A, B)$.
3: Build the capped weighted adjacency graph $AG_\sigma(A, B)$.
4: **for** $\ell = 2, 4, \ldots,$ maximum cycle length possible **do**
5: List all cycles of $AG_\sigma(A, B)$ of length ℓ.
6: Select a set $\mathcal{C}'$ of consistent cycles trying to maximize the sum of weights by a WMIS algorithm and add $\mathcal{C}'$ to $\mathcal{C}$, let $AG_\sigma(A, B) := AG_\sigma(A, B) \setminus \mathcal{C}'$, update M by adding the new gene connections induced by $\mathcal{C}'$.
7: If M is not a maximal matching of $GS_\sigma(A, B)$, find and delete disposable genes from $AG_\sigma(A, B)$ and go back to Step 4.
8: Return $\sum_{C \in \mathcal{C}} \widehat{w}(C)$.

4.5 Experimental Results

Experiments for the ILP and our heuristics were conducted on an Intel i7-4770 3.40GHz machine with 16 GB of memory. In order to do so, we produced simulated datasets by the Artificial Life Simulator (ALF) [41] and obtained real genome data from NCBI, using the FFGC tool[1] to obtain similarity scores between genomes. Gurobi Optimizer 7.0 was set to solve ILP instances with default parameters, time limit of 1800 seconds and 4 threads, and the heuristics were implemented in C++.

Simulated Data

We generated datasets with 10 genome samples each, running pairwise comparisons between all genomes in the same dataset. Each dataset has genomes of sizes around 25, 50 or 1000 (the latter used only for running the heuristics), generated based on a sample from the tree of life with 10 leaf species and PAM distance of 100 from the root to the deepest leaf. Gamma distribution with parameters $k = 3$ and $\theta = 133$ was used for gene length distribution. For amino acid evolution we used the WAG substitution model with default parameters and the preset of Zipfian indels with rate 0.00005. Regarding genome level events, we allowed gene duplications and gene losses with rate 0.002, and reversals and

[1] https://bibiserv2.cebitec.uni-bielefeld.de/ffgc

transpositions (which ALF refers to as translocations) with rate 0.0025, with at most 3 genes involved in each event. To test different proportions of genome level events, we generated simulated datasets with 2- and 5-fold increase for reversal and transpositions rates.

Results are summarized in Table 4.1. Each dataset is composed of 10 genomes, totaling 45 comparisons of pairs per dataset. Rate $r = 1$ means the default parameter set for genome level events, while $r = 2$ and $r = 5$ mean the 2- and 5-fold increase of rates, respectively. For the ILP the table shows the average time for instances for which an optimal solution was found, the number of instances for which the optimizer did not find an optimal solution within the given time limit and, for the latter class of instances, the average relative gap between the best solution found and the upper bound found by the solver, calculated by $(\frac{\text{upper bound}}{\text{best solution}} - 1) \times 100$. For our heuristics, the running time for all instances of sizes 25 or 50 was negligible, therefore the table shows only the average relative gap between the solution found and the upper bound given by the ILP solver (if any).

Table 4.1: **Results of experiments for simulated genomes.**

	ILP			Maximum-Matching	Greedy-Density	Greedy-Length	Greedy-wmis
	Time (s)	Not finished	Gap (%)	Gap (%)	Gap (%)	Gap (%)	Gap (%)
25 genes, $r = 1$	19.50	0	–	16.26	5.03	5.84	5.97
25 genes, $r = 2$	84.60	2	69.21	58.69	30.77	43.57	43.00
25 genes, $r = 5$	49.72	0	–	81.39	43.83	55.38	55.38
50 genes, $r = 1$	265.23	12	23.26	63.02	24.76	27.86	26.94
50 genes, $r = 2$	463.50	29	38.12	123.71	65.41	66.52	64.78
50 genes, $r = 5$	330.88	29	259.72	281.70	177.58	206.60	206.31

Results clearly show the average relative gap of heuristics increases proportionally to the rate of reversals and transpositions. This is expected, as higher mutation rates often result in higher normalized weights on longer cycles, thus the association of genes with greater gene similarity scores will be subject to the selection of longer cycles. Interestingly, for some larger instances the relative gap for heuristics is very close to the values obtained by the ILP solver, suggesting the use of heuristics may be a good alternative for some classes of instances or could help the solver finding lower bounds quickly. It is worth noting that the Greedy-Density heuristic found solutions with gap smaller than 1% for 38% of the instances with 25 genes.

In a single instance (25 genes, $r = 2$), the gap between the best solution found and the upper bound was much higher for the ILP solver and for the heuristics. This instance in particular is precisely the one with the largest number of edges in $GS_\sigma(A, B)$ in the dataset. This may indicate that a moderate increase in degree of vertices (1.3 on average to 1.8 in this case) may result in much harder instances for the solver and the heuristics, as after half of the time limit the solver attained no significant improvement on solutions found, and the heuristics returned solutions with a gap even higher.

We also simulated 10 genomes of sizes around 50, with PAM distance of 15 from the root to the deepest leaf, therefore evolutionarily "closer" to each other and for which higher similarity values are expected. For these genomes the default rates were multiplied by ten (10-fold) for Zipfian indels, gene duplications, gene losses, reversals and transpositions, otherwise there would be no significative difference between them. The exact ILP algo-

rithm found an optimal solution for only 4 of the 45 instances, taking 840.59 seconds on average. For the remaining instances, where the ILP did not finish within the time limit, the average gap is 329.53%. Regarding the heuristics (Table 4.2), that all run in negligible time, Greedy-Density outperforms the others, with an average gap of 163% compared to the best upper bound found by the ILP solver. Surprisingly, values returned by greedy heuristics are better than values obtained by the ILP for these instances. Results again suggest that the ILP could benefit greatly from heuristics by using their results as initial lower bounds. Moreover, for some groups of instances even heuristics alone can obtain excellent results.

Table 4.2: **Results of experiments for 10 simulated genomes (45 pairwise comparisons) with smaller PAM distance.**

	ILP			Maximum-Matching	Greedy-Density	Greedy-Length	Greedy-wmis
	Time (s)	Not finished	Gap (%)	Gap (%)	Gap (%)	Gap (%)	Gap (%)
50 genes, $r = 10$	840.59	41	329.53	415.57	163.00	172.02	168.58

Although we have no upper bounds for comparing the results of our heuristics for genome sizes around 1000, they are still very fast. For these genomes we analyze the Maximum-Matching algorithm separately afterwards, taking into account for now only the other three heuristics. The average running times are 0.30 s, 15.11 s and 12.16 s for Greedy-Density, Greedy-Length and Greedy-wmis, respectively, showing nevertheless little difference on results.

However, in 25% of the instances with $r = 5$, the difference from the best to the worst solutions provided by these heuristics varied between 10% and 24%, the best of which were given by Greedy-Density. That is probably because, instead of prioritizing shorter cycles, Greedy-Density attempts to balance both normalized weight and length of the selected cycles. The average running times for the instances with $r = 5$ are 1.84 s, 76.02 s and 80.67 s for Greedy-Density, Greedy-Length and Greedy-wmis, respectively.

Still for genomes of size around 1000 and $r = 5$, the Maximum-Matching heuristic is the fastest, with an average running time of 1.70 s. Despite being the best heuristic for a few cases, the similarity value given by this heuristic is merely 27% of the value given by the best heuristic, on average. While the Maximum-Matching heuristic is clearly not useful for calculating similarity values, these results show how significant it is to choose cycles with the best normalized weights versus prioritizing edges with best weights in the gene similarity graph for the ffdcj-similarity problem. Since this property of the Maximum-Matching somehow reflects the strategy of family-based comparative genomics, this observation indicates an advantage of family-free analysis compared to family-based analysis.

To better understand how cycles scale, we generated 5-fold instances with 100, 500, 1000, 5000, and 10000 genes, running the Greedy-Density heuristic for these instances and counting different cycle lengths. The running time was 0.008s, 0.667s, 1.98s, 508s, and 2896s, respectively, on average. Results (Fig. 4.9) show that most of the cycles found are of short lengths compared to the genome sizes, providing some insight on why heuristics are fast despite having to enumerate a number of cycles that could be exponential. Besides, even the maximum number of longer cycles found for any instance is reasonably small.

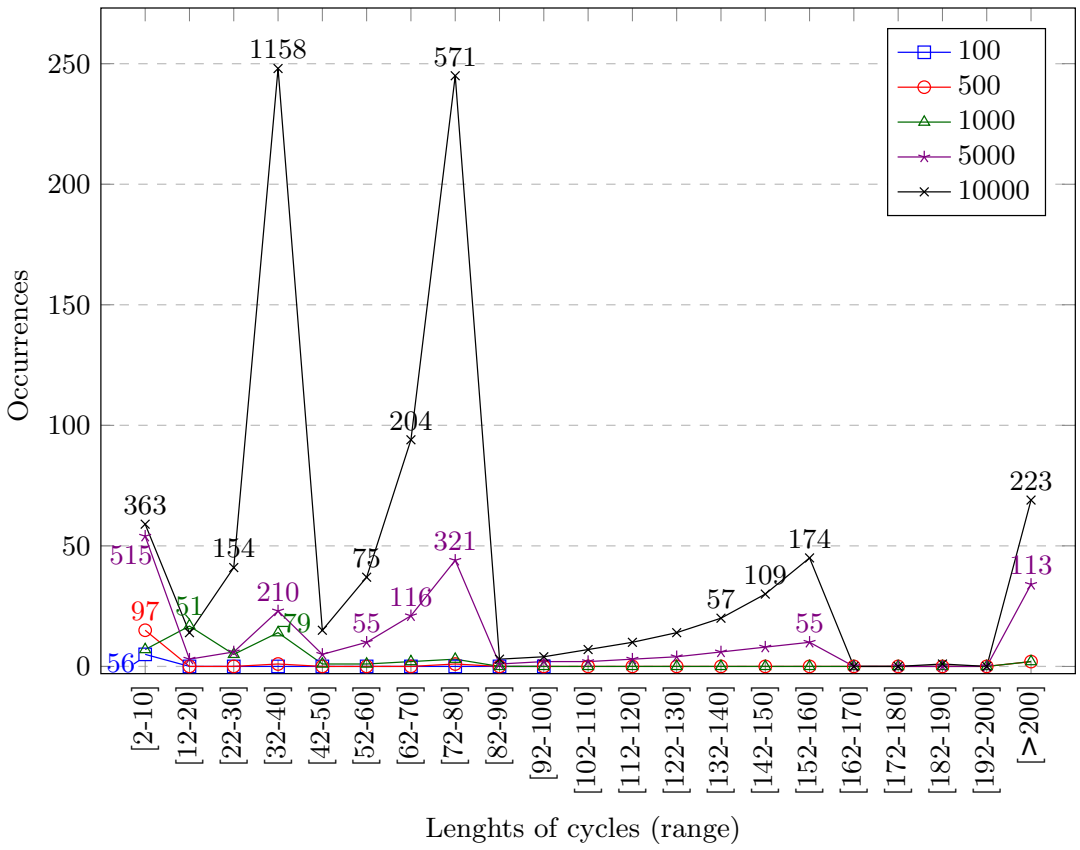


Figure 4.9: **Average count of cycles found by lengths in the Greedy-Density heuristic.** Instances have $r = 5$ and genome sizes of 100, 500, 1000, 5000, and 10000 genes. Numbers above marks denote the maximum number of cycles for a pair of genomes in an instance (only for values greater than 50). Recall that this heuristic finds and selects cycles of lengths for smaller ranges first.

Finally, as expected, experiments for genomes simulated with different parameters indicate the FFDCJ similarity decreases as the PAM distance or the rates of genome level events increases (data not shown).

Real Genome Data

To show the applicability of our methods to real data, we obtained from NCBI protein-coding genes of X chromosomes of human (*Homo-sapiens*, assembly GRCh38.p7), house mouse (*Mus musculus*, assembly GRCm38.p4 C57BL/6J), and Norway rat (*Rattus norvegicus*, assembly Rnor_6.0). In mammals, the set of genes on the X chromosome has been reasonably conserved throughout the last several million years [94], having however their order disrupted many times.

Since protein sequences are used to obtain the similarity scores (with the help of the BLASTp tool) instead of nucleotide sequences, 76 genes from the rat genome were excluded because no protein sequence was available. Besides, when a gene has multiple isoforms, the longest is kept. The number of genes in the resulting genomes were 822, 953 and 863

for human, mouse and rat, respectively, some of them removed from the pairwise genome comparison due to the pruning process of FFGC.

Table 4.3 shows, as expected, that the two rodent X chromosomes have a higher similarity than any of them to the human X chromosome. The values returned by the greedy heuristics are very similar, where Greedy-Length is the fastest. Maximum-Matching results are less than 5% distant from the results of the greedy heuristics, which indicates the choice of cycles has some influence but does not dominate the similarity values obtained for these instances. Matching sizes are similar for all heuristics, showing that about 8% of the genes of the smaller genomes could not be matched to some gene of the other genome and had to be removed, that is, they are disposable genes.

Table 4.3: **Results for heuristics on real genomes.** *Smaller genome* column shows for each pair of genomes the number of genes in the smaller one, an upper bound for the matching size. Heuristics are represented by their initials (e.g. Greedy-Length = GL).

	Smaller genome	Matching size				Time (s)				Similarity			
		MM	GD	GL	GW	MM	GD	GL	GW	MM	GD	GL	GW
human/mouse	696	643	643	643	643	0.07	19.6	0.1	8.6	404.56	420.64	421.48	420.72
human/rat	672	613	611	611	612	0.05	11.6	0.04	3.3	358.36	374.17	374.27	373.82
mouse/rat	746	690	689	689	689	0.17	0.18	0.13	0.18	481.53	500.59	500.57	500.36

4.6 Concluding remarks

In this chapter we studied the (NP-hard) problem of computing the family-free DCJ similarity. After presenting it formally, we showed that the problem is also APX-hard and cannot be approximated with approximation ratio better than $22/21 = 1.0476\ldots$, unless P = NP. Then, we proposed an exact ILP algorithm to solve it. Following, we developed four heuristic algorithms and could show that they perform well while having reasonable running times also for realistic-size genomes. Further, heuristics could improve the ILP running time and results.

Initial experiment on real data can be considered a proof of concept. In general, the computational results of this work can be used to more systematically study the applicability of the DCJ similarity measure in various contexts. One important point to be investigated is whether, differently from parsimonious distance measures that usually only hold for closely related genomes, a genomic similarity would allow to perform good comparisons of more distant genomes as well. Fine-tuning of both the data preparation and objective function may be necessary, though.

For example, one drawback of the function s_{FFDCJ} as defined in Equation (4.3) is that distinct pairs of genomes might give family-free DCJ similarity values that cannot be compared easily, because the value of s_{FFDCJ} varies between 0 and $|M|$, where M is the matching giving rise to s_{FFDCJ}. Therefore some kind of normalization would be desirable. A simple approach could be to divide s_{FFDCJ} by the size of the smaller genome (a trivial upper bound for $|M|$). On the other hand, dividing by the size of the larger would decrease the maximum similarity to values bellow 1, depending on the proportion of sizes of compared genomes. Moreover, such simple approaches be applied as a simple postprocessing step, keeping all theoretical results of this work valid. A better normalization, however, might be to divide by $|M|$ itself. An analytical treatment here seems more difficult, though.

Appendix

4.A How to get data

This final subsection presents a brief and high level description of steps and tools used to obtain simulated or real data in a suitable input format for the implementation used in the experiments. The complete description can be found in https://git.facom.ufms.br/diego/ffdcj-sim.

The file format is very simple and defines two unichromosomal genomes. Each line is composed by 4 fields:

1. Gene number in genome A;
2. Gene number in genome B;
3. Relative orientation between genes; and
4. Similarity between genes.

For instance:

```
1  1  1  0.1
2  2  1  0.2
2  1 -1  0.3
3  2  1  0.4
```

where relative orientation 1 means a direct correspondence and -1 a reverse correspondence. Notice, however, that this file format does not define explicitly the order of genes in genomes. This is handled by the implementation, but in short, for the input:

```
1  1  1  0.64753765
2  1 -1  0.76287652
```

a compatible gene order would be $A = \{(\circ\ 1\ -2\ \circ)\}$, $B = \{(\circ\ 1\ \circ)\}$ and another one would be $A = \{(\circ\ -1\ 2\ \circ)\}$, $B = \{(\circ\ -1\ \circ)\}$.

To obtain these input files from simulated or real data, FFGC[2,3] can be used to automate some steps. This is a workflow that provides many functionalities related to family-free analysis of genes.

[2] https://bibiserv2.cebitec.uni-bielefeld.de/ffgc
[3] https://git.danieldoerr.de/projects/FFGC

For simulated data, after obtaining the files generated by ALF[4] (online or standalone versions), the script `convert_alf_seq_ids.py` (by Daniel Doerr, included with the source files of the implementation of the heuristics) must be executed taking as input the FASTA[5] files provided by ALF to prepare them for use with FFGC. This script just makes few adjustments in the FASTA files. After that, the command `create_project` of FFGC can create a project which automates the process of obtaining input files in a format that can be used with our implementation.

For real data, GenBank files of species from NCBI[6] database are required. The following step is using the `create_project` as above. For example, to download the GenBank file for Homo Sapiens:

1. Look for Homo Sapiens in genome database at NCBI (e.g. https://www.ncbi.nlm.nih.gov/genome/?term=homo+sapiens);
2. In the X chromosome row, click at "RefSeq" column (ex. NC_000023.11); and
3. Download the GenBank file by clicking at "Send to" with "Complete Record", "File" (in "Choose Destination") and "GenBank (full)" (at "Format") options selected.

Lastly, given input files with genomes as specified above, the same implementation also generates input files containing restrictions and the objective function for the ILP solver.

[4] http://alfsim.org
[5] https://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml
[6] https://www.ncbi.nlm.nih.gov

Chapter 5

Family-based local DCJ similarity

As the last part of this doctoral study, we introduce the concept of (family-based) local DCJ similarity, a local genome rearrangement measure analog to local sequence alignment [105]. As this thesis is directed towards rearrangement measures, the first sections of this chapter focuses on presenting preliminaries and this novel measure.

Then, in the following sections, we present ANGORA, an improved workflow for ancestral genome reconstruction from highly diverged genomes such as those of plants, including how the local DCJ similarity allowed reconstructed ancestral regions with higher resolution [105]. Such a workflow relies on an established workflow in the reconstruction of ancestral plants [107]. However, it is important to stress that the enhancement allowed by the local DCJ similarity is only one among other improvements of our workflow. In addition, instead of using gene annotations for inferring the genome content of the ancestral sequence, we identify genomic markers through a process called *genome segmentation*. This enables us to reconstruct the ancestral genome from hundreds of thousands of markers rather than the tens of thousands of annotated genes.

With the enhanced workflow at hand, we reconstruct the ancestral genome of eudicots, a major sub-clade of flowering plants, using whole genome sequences of five modern plants. Our reconstructed genome is highly detailed, yet its layout agrees well with that reported in Badouin *et al.* [10], which was reconstructed using the original workflow. Using local genome rearrangement, not only the marker-based, but also the gene-based reconstruction of the eudicot ancestor exhibited increased genome content, evidencing the power of this novel concept. We review our ancestral reconstruction results in relation to those of Badouin *et al.* in the last part of the chapter. At the very end, technical details related to availability of tools and data and parameters used in tools are provided in an appendix.

Another important point to highlight is that, for the different ancestral genome reconstruction approaches considered here, it does matter how genomic markers are defined. Therefore, in this chapter we consider chromosomes as sequences of "genomic markers" and we use "annotated genes" to refer to one specific method for obtaining such markers [2, 135].

5.1 Preliminaries

Here, for ease of reading, we recall a couple of previously described definitions before presenting few formal definitions necessary in the following sections.

Genomic sequences

Given a sequence S over $n =: |S|$ markers, $S[i]$ denotes the marker at the position i and $\mathcal{G}(S) := \bigcup_{i=1}^{n}\{|S[i]|\}$ is the *(genome) content* of S. Further, we define the *multiplicity* of a marker g in sequence S as $m_S(g) := |\{i : 1 \leq i \leq n \text{ and } |S[i]| = g\}|$. A sequence S is *duplicated* if any of its markers has multiplicity larger than one. Such markers are *duplicate* markers. Further, two sequences S and T are *balanced* if $\mathcal{G}(S) = \mathcal{G}(T) =: \mathcal{G}$ and each marker $g \in \mathcal{G}$ has the same multiplicity in both genomes, i.e., $m_S(g) = m_T(g)$. The concepts of multiplicity, duplication, and balance naturally propagate to collections of marker sequences and thus apply equally to genomes.

The *interval* $[i, j]$ in sequence S gives rise to the substring $S[i, j] = S[i]S[i+1]\cdots S[j]$, with $1 \leq i \leq j \leq |S|$. An interval $[i, j]$ of sequence S is called *maximal* if it cannot be extended to its left or right without changing the genome content, i.e., either $i = 1$ or $\mathcal{G}(S[i-1, j]) \neq \mathcal{G}(S[i, j])$ and either $j = n$ or $\mathcal{G}(S[i, j+1]) \neq \mathcal{G}(S[i, j])$. Given two sequences S and T, a pair of intervals $[i, j]$ and $[k, l]$ of S are *common intervals* if $\mathcal{G}(S[i, j]) = \mathcal{G}(T[k, l])$. A sequence T is a *subsequence* of S if $T = S[i_1]\ S[i_2]\cdots S[i_k]$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq |S|$.

Adjacency graph for genomes without duplicate markers

A genome without duplicate markers A can be represented by its *set of adjacencies* $\text{adj}(A)$, where each genomic marker g of its chromosomes is represented by a pair of its head and tail extremity g^h and g^t, respectively, i.e., by pair (g^t, g^h) if marker g lies in reading direction of the chromosome, otherwise by (g^h, g^t). Then $\text{adj}(A)$ is given by the set of incident extremities of consecutive markers, where the first and last adjacencies of linear chromosomes correspond to the outermost extremities of the first and last markers, called *telomeric* adjacencies.

Given two balanced genomes A and B without duplicate markers, the adjacency graph $AG(A, B)$ is a bipartite multigraph (U, V, E), with vertex sets $U = \text{adj}(A)$ and $V = \text{adj}(B)$ and edge multiset $E = \{(u, v)\ \textit{with multiplicity}\ |u \cap v| : u \in U, v \in V\ \textit{and}\ u \cap v \neq \emptyset\}$. For such A and B, there is one single decomposition of $AG(A, B)$ in paths and cycles, which can be found easily. Informally, short cycles and paths in $AG(A, B)$ reflect conserved structures between A and B, whereas long cycles and paths reflect the opposite. For instance, it is a classic result of the field that the *DCJ distance* between A and B, that is, the mininum number of DCJ operations necessary to transform A into B, can be computed in linear time by counting the number of cycles and odd paths in $AG(A, B)$ [15, 134]. The adjacency graph is described in detail in Chapter 3.

5.2 The local DCJ similarity

Similar to local sequence alignment, local genome rearrangement aims at identifying highly conserved pairs of substrings of two given marker sequences. For the same reason that the

edit distance cannot be used for computing local alignments, the DCJ distance cannot be used to compute local rearrangements: Both would favor pairs of substrings that minimize the number of edit operations independent of their length, thereby giving pairs of small substrings—in particular the pair of empty strings—a dishonest advantage. Clearly, the method of choice are similarity measures that, rather than solely penalizing *dissimilarity*, quantify *similarity*. Conversely, global measures of DCJ similarity that only maximize the (weighted) number of cycles and paths in the adjacency graph [89, 104, 106], are unsuitable as well: In search of locally similar sequences, it is not sufficient to reward only similarity (then, a best local solution would always correspond to a global solution), but it is necessary to also penalize dissimilarity.

With the goal of studying highly diverged genomes, we designed a procedure able to tolerate all kinds of differences caused by insertion, deletion, or duplication of one or several genomic markers. To this end, we first discover referenced-based approximate common intervals in the studied genomes. Each discovered set of intervals gives rise to a set of pairs of substrings between the reference and the remaining genomes for which local rearrangement scores are calculated.

Let S and T be two substrings associated with one of these pairs of approximate common intervals. Our method relies in two steps that are illustrated by an example in Figure 5.1. First, pairs of sequences S', T' are identified such that (i) S' is a subsequence of S, and T' of T, (ii) S' and T' are balanced, and (iii) for each marker $g \in \mathcal{G}(S')$ holds true that $m_{S'}(g) = \min(m_S(g), m_T(g))$. The last constraint ensures maximality of the balanced subsequences.

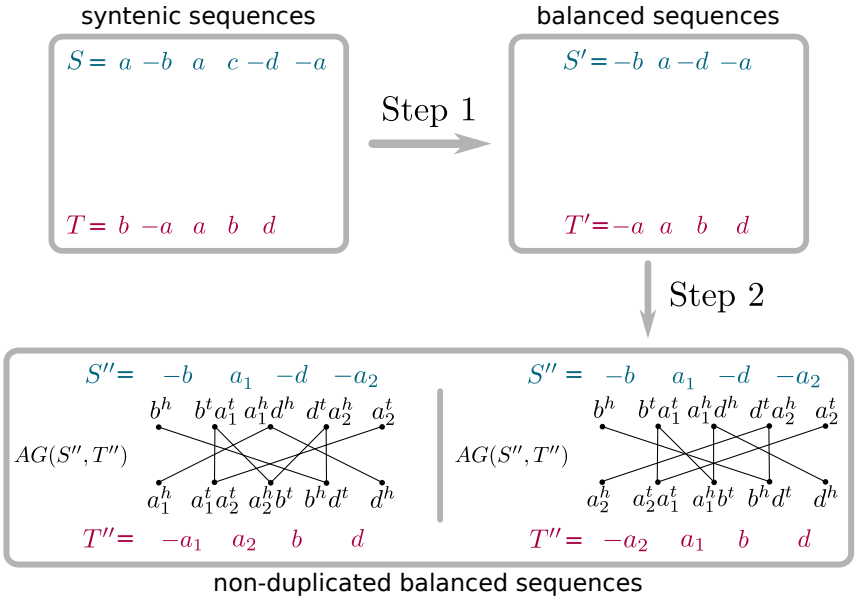


Figure 5.1: **Example of local DCJ similarity.** The figure illustrates the procedure for computing local DCJ similarity scores from a pair of syntenic marker sequences.

Sequences S' and T' are then subjected to a second procedure that finds one-to-one assignments between all markers of the two sequences, thus further refining them to non-duplicated balanced sequences S'' and T''. This allows us to define formally the local DCJ similarity problem:

Problem LOCAL-DCJ-SIMILARITY(S, T, f): Given two substrings S and T associated with a pair of approximate common intervals, and a function $f : 2\mathbb{N} \to \mathbb{R}$ that scores each cycle and path proportional to its length, identify non-duplicated balanced sequences S'' and T'' that maximize the following formula:

$$s_{\text{DCJ}}(S'', T'') = \sum_{C \in \mathcal{C}} f(|C|) + \frac{1}{2}\left(\sum_{O \in \mathcal{O}} f(|O|+1) + \sum_{E \in \mathcal{E}} f(|E|+2)\right) - d \cdot p\,,$$

where $\mathcal{C}$, $\mathcal{O}$ and $\mathcal{E}$ are the sets of cycles, odd paths, and even paths in the constructed adjacency graph of S'' and T'', $d := |S| + |T| - (|S''| + |T''|)$ is the number of deleted markers and p is the deletion penalty.

Notice that, as in earlier literature [23, 95], the lengths of paths are corrected in $s_{\text{DCJ}}(S'', T'')$ so that structures with the same sorting distance have the same "length".

Because short cycles and paths are indicators of similarity, whereas long cycles and paths suggest the opposite, we found a simple realization of f that has been working well in our experiments:

$$f(l) = \frac{2 - l}{L - 2} + 1\,. \tag{5.1}$$

Our function f makes use of a constant L, a length threshold that demarcates short from long cycles and paths.

5.3 An improved ancestral genome reconstruction workflow

In an attempt to combine the two complementary strategies presented in Section 1.3, that is, model-based and synteny-based reconstruction methods, we developed a rearrangement-aware synteny-based reconstruction method that extends an established pipeline for ancestral genome reconstruction in plants [97,107] used in multiple studies [10,93,132]. In addition to other important improvements, the local DCJ similarity make it possible for our method to refine the genome content of syntenic blocks—conserved neighborhoods of individual pairs of markers—prior to deriving contiguous ancestral sequences.

We present in this section ANGORA (ANcestral reconstruction by local GenOme Rearrangement Analysis), a workflow for inferring ancestral genome sequences with unlike higher degree of detail than obtained by currently available approaches. Later in this chapter we further report on our ongoing progress in refining the resolution of the ancestral genome sequence of eudicots based on the genome sequences of five modern plants. We achieve a high degree of detail by improving several steps in the ancestral reconstruction process: First, our method identifies genomic markers, enabling the reconstruction of the ancestral genome from hundreds of thousands of markers rather than the tens of thousands of genes that have been annotated in the five eudicot genomes as of today. That way, our method does not need to rely on the quality of the gene annotation. But more importantly, our method can lead to a more comprehensive reconstruction of the ancestral genome content, as it is not restricted to those blocks of DNA attributed to protein-coding genes, and reveal new conserved blocks of yet unknown function [121]. Second, it infers syntenic blocks across all extant genomes by tolerating inserted, deleted,

and duplicated markers. Third, our method takes into account the internal structure of syntenic blocks for the reconstruction of contiguous ancestral regions by means of the local DCJ similarity measure proposed in this chapter.

In the following we describe in detail ANGORA. Our method is based on previous work by Salse [97, 107], but, in short, it additionally includes a preceding step to identify genomic markers. Subsequently, syntenic blocks are identified, families are refined with the help of the local DCJ similarity, and finally syntenic blocks are used to derive contiguous ancestral regions.

Identification of genomic markers

We obtain genomic markers by a heuristic for the genome segmentation problem (GSP) [25]. Informally, the objective of genome segmentation is, given a DNA sequence and pairwise alignments of the DNA sequence onto itself, to decompose the DNA sequence into families of non-overlapping similar segments, called *atoms*. Figure 5.2 shows an example of a segmentation of two DNA sequences.

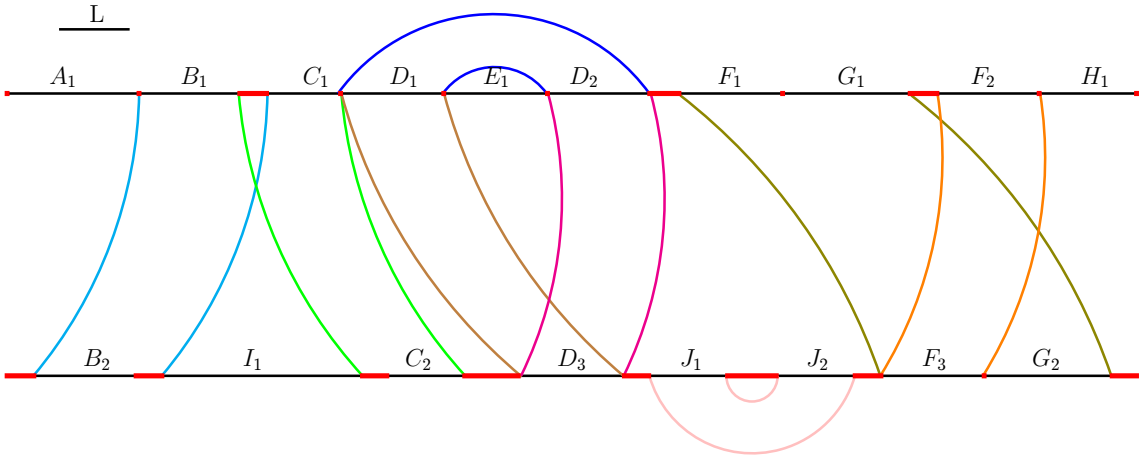


Figure 5.2: **Example of a segmentation of two DNA sequences.** The minimum segment length L is indicated by the line in the upper left corner, waste regions not covered by any atom are marked by thick red lines, input pairwise alignment boundaries by colored arcs. Capital letters above atoms indicate their family membership. For ease of visualization, the two sequences are not depicted as a single concatenated DNA sequence.

In order to define a consistent set of families, genome segmentation requires that no two atoms overlap and no alignment boundary lies within any of the created atoms. In addition, whenever a segment s defining an atom is aligned to another segment t (so they are put in the same family), t must comprise another single atom (see Fig. 5.3)

A trivial segmentation would establish every single character of the input sequence into an atom of its own, thus satisfying the stated criteria. To avoid such a meaningless segmentation, a minimal length requirement is also imposed on the constructed atoms. Any nucleotide that is not covered by an atom resides in a *waste region*. Note that the genome segmentation problem for multiple DNA sequences is simply defined as the segmentation problem of the concatenated DNA sequences. Given the observations above, we can define the problem GSP formally.

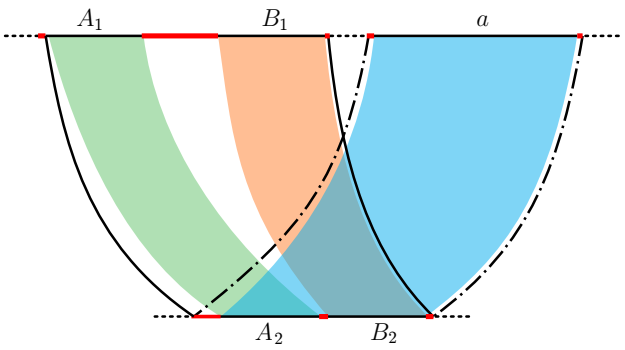


Figure 5.3: **Example of an inconsistent segmentation.** Waste regions not covered by any atom are marked by thick red lines and two input pairwise alignment boundaries are shown by solid and dash-dotted arcs. Atoms are shown in solid black lines, capital letters near atoms indicate their family membership, and colored shaded regions represent the alignments that define family membership of atoms. This situation does not define a proper segmentation, because the segment defining the atom a is aligned to another segment comprising two atoms A_2 and B_2, thus there is now way to define a's membership.

> **Problem** GSP(S, α, L): Given a sequence S, a set of alignments α of S onto itself and a length L, find a set of atoms $\mathcal{A}$ that minimizes the total number of nucleotides located in waste regions and for which the following conditions hold:
>
> 1. No two atoms overlap;
> 2. Each atom has length at least L;
> 3. There is no alignment boundary inside any atom;
> 4. Whenever a segment s defining an atom is aligned to another segment t to define membership, t must comprise another single atom.

In 2013, Visnovská and colleagues have proven the intractability of GSP and devised a heuristic called IMP for its solution [128].

Discovery of syntenic blocks

Syntenic blocks are blocks of two or more extant genome sequences that are *homologous*, i.e., they originate from the same block of a common ancestral sequence. The identification of syntenic blocks in highly diverged genomes, such as the five eudicots subject to our study, is challenging. That is because on the one side, the notion of synteny is highly flexible, simultaneously allowing an entire chromosome to be classified into a single syntenic block, as well as individual segments thereof [59]. On the other side, multiple rounds of mutations such as insertions, deletions, duplications, and rearrangements can scramble and decompose syntenic blocks into barely recognizable units. Methods to identify syntenic blocks under such conditions must be equally flexible: they must tolerate comprehensive changes in the order and multiplicity of genomic markers, but at the same time pick up the signal of synteny on all levels of granularity, ranging from chromosome level down to synteny of individual pairs of genomic markers.

One such method that is particularly fast (speed is another important concern of this step in the ancestral reconstruction workflow) is Gecko3 [131], which identifies syntenic blocks by discovering approximate common intervals in marker sequences. These are sets of intervals with associated genome content $\mathcal{G}$ such that the symmetric difference between each interval and $\mathcal{G}$ is bounded by δ^{sum} and, more specifically, the number of excessive (i.e., *inserted*) markers is bounded by δ^{add}, and the number of missing markers by δ^{loss}. Gecko3 identifies the genome content of a set of intervals by a referenced-based approach. In doing so, a designated genome (the "reference") is taken as scaffold for the discovery of approximate common intervals in the other genomes. Any interval in the reference defines the genome content $\mathcal{G}$ of an interval set. Gecko3 can find approximate common intervals with multiple occurrences within a single sequence and also provides a *quorum* parameter q by which reference-based approximate common intervals can be discovered that are conserved only in a subset of genomes of size at least q.

Family refinement using local DCJ similarity

In reconstructing ancestral genomes, we use non-duplicated balanced sequences S'' and T'' identified by the local DCJ similarity optimization procedure to refine the genomic marker families across the entire genomic dataset. Because each family can contribute with at most one marker to the ancestral reconstruction, this enables substantial improvement in determining the ancestral genome content, as detailed in the next section. To this end, we implemented a procedure that takes unambiguous one-to-one assignments across overlapping syntenic blocks to decompose their marker families into disjoint subsets. Further, if non-overlapping sets of syntenic blocks share markers from the same family, this family is also decomposed into disjoint subsets corresponding to the syntenic block affiliation of its members. The refinement process is depicted in Figure 5.4 and described here in detail.

Two sets of syntenic blocks $B = \{b_1, b_2, \ldots\}$ and $B' = \{b'_1, b'_2, \ldots\}$ *overlap* if any block of B shares a genomic marker with a block of B'. Given a collection $\mathcal{B}$ of sets of syntenic blocks, we define a graph $G = (V, E)$ with vertex set $V = \mathcal{B}$ and edge set $E = \{(B, B') : \{B, B'\} \subseteq \mathcal{B}, B \text{ and } B' \text{ overlap}\}$. Each connected component of G induces a (maximal) *overlapping set* of sets of syntenic blocks. For each such overlapping set O, new (sub-) families are created according to the following two rules:

(i) Let F be a family of markers and $F_O \subset F$ be the subset of markers embedded in any set of syntenic blocks of O. For each such family F for which $F_O \neq \emptyset$, a new family F_O is created;

(ii) Let $m_1 \in F_O$ be a marker of the reference sequence S_1 and let $S_2, \ldots, S_k$ be the $k-1$ genomic sequences other than the reference such that each S_i, $2 \leq i \leq k$, has a marker assigned to m_1 in at least one local DCJ similarity computation. If, for all $2 \leq i \leq k$, m_1 is assigned to the same marker m_i of S_i in every local DCJ similarity computation between the reference and S_i, then the set of markers $\{m_1, m_2, \ldots, m_k\}$ induces a new family. This rule further refines new families created by rule (i).

Reconstruction of CARs

The last step of the workflow is conducted with ANGES and is the same as in the original workflow of Salse [107]. ANGES takes as input syntenic blocks or identifies them by

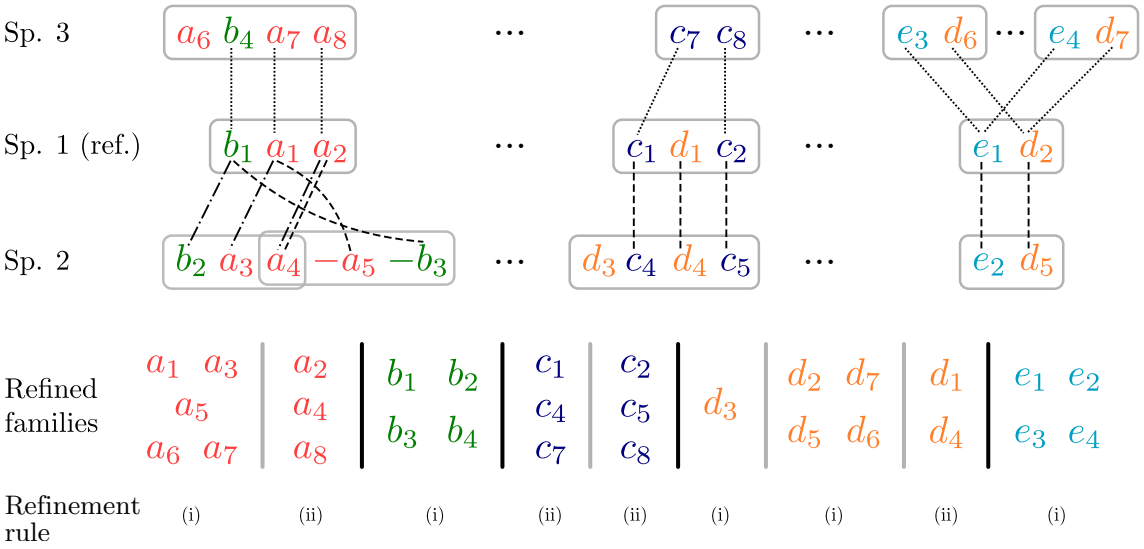


Figure 5.4: **Example of family refinement for hypothetical species 1 (reference), 2 and 3.** For each species, occurrences of exemplary syntenic blocks are shown, indicated by gray boxes. Families a, b, c, d and e are refined based on one-to-one assignments (indicated by dotted, dashed, and dotted-dashed lines) of local DCJ similarity calculations between the reference and the other two sequences. Subscript indices are used to distinguish markers of the same family.

discovering maximal common intervals (or constrained variants thereof). The identified intervals are then either weighted by user-provided data, or according to the occurrences in the extant genomes and subsequently used to construct and output a PQ-tree. A *PQ-tree* is a hierarchical data structure capable for the lossy representation of all common intervals of two or more permutations. To this end, PQ-trees make use of two kinds of nodes: P-nodes, which do not impose any order of their child nodes, and Q-nodes, which indicate a linear order of their children.

5.4 Results

As a test scenario for our workflow, we have conducted the ancestral genome reconstruction of eudicots, a major sub-clade of flowering plants, using whole genome sequences of five modern plants. This is the same reconstruction made by Badouin *et al.* [10] using the original workflow of Salse [107]. A brief overview of eudicots, the selected species and the challanges for ancestral reconstruction of plant genomes can be found bellow.

In the following few subsections we describe the implementation and evaluate each major step of the ancestral reconstruction of eudicots made by our workflow, comparing our results with those reported by Badouin *et al.* [10].

Eudicots

Flowering plants, with *eudicots* being their largest sub-clade, are an important subject of paleogenomic studies, not only because of their ecological significance and relevance for the crop industry, but also because the reconstruction of ancestral plant genomes is considered

the most challenging endeavor of the field [119,122]. The reconstruction of ancestral plant genomes is hard for multiple reasons: above all, plant genomes are often repetitive, as a result of one or more rounds of whole genome multiplication events that often occurred in their evolutionary past. Each round of polyploidization is followed by a period of dramatic genomic turnover in which the numbers of chromosomes and genes are reduced close to the order of magnitude prior to polyploidization. In doing so, chromosomes sustain large-scale rearrangements. Redundant genes and other functional units are randomly lost, leading to a fractionated layout of the genome when compared to its pre-polyploidization state. Furthermore, plant genomes are large, often exhibiting extensive intra- and inter-genic regions which themselves host repetitive elements such as transposons and long terminal repeats [107].

We study the eudicot phylogeny composed of grape, a representative of the rosids, and four asterids—artichoke, coffee, lettuce, and sunflower. Polyploidization is a major source of genomic innovation in plants and the studied eudicots are no exception to this rule [10, 107]. After the speciation of the eudicots and monocots around 140 to 150 millions years ago, the eudicot ancestor underwent a *whole genome triplication* (WGT), further denoted as γ, common to all known eudicots of today. Further polyploidizations occurred on subbranches, such as the WGT in the ancestor of the Asterids II group, to which sunflower, artichoke, and lettuce belong. The sunflower lineage underwent another *whole genome duplication* (WGD) event. Figure 5.5 gives an overview of the eudicot phylogeny and the described polyploidizations. The genome architecture of grape is closest to the post-γ ancestor, with only one chromosome fission and three chromosome fusions separating the two genomes. Therefore, in this chapter, the karyotypic architecture of the grape genome serves as proxy for reconstructing the genome of the post-γ ancestor.

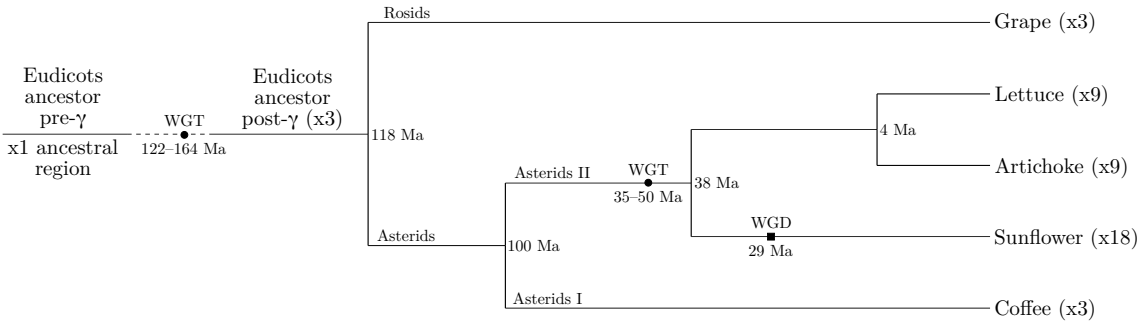
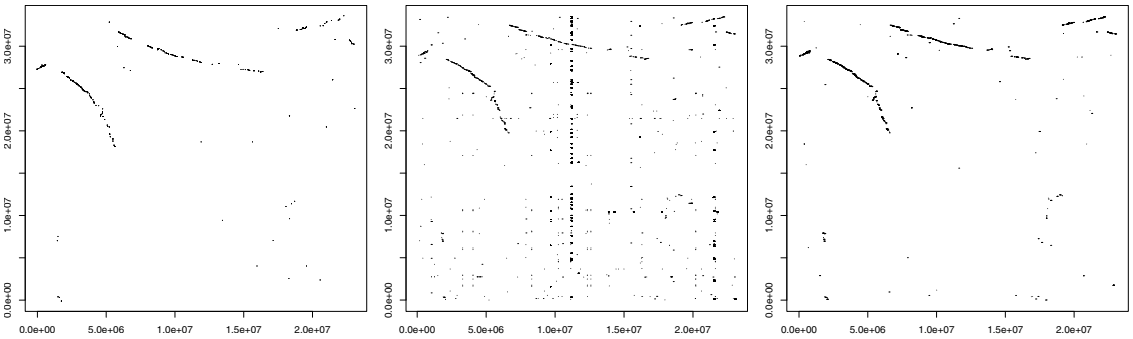


Figure 5.5: **Eudicot phylogeny including grape and four asterids [10].** Circles and squares mark WGT and WGD events, respectively. Time is expressed in millions of years (Ma).

Genome segmentation

To enable the processing of large genomic datasets such as the one at hand, we have re-implemented the heuristic IMP [128] in `C++` and adapted it for parallel computation. Our software, named GEESE (GEnomE SEgmentation), is included in the ANGORA workflow, but can also be obtained separately. (For details, see Section 5.A.) Following the approach by Visnovská, Vinař and Brejová [128], we used LASTZ [67] to compute local sequence alignments between all pairs of genomic sequences from the five eudicots. In doing so, we chose alignment parameters (see Section 5.C) that improved the clarity and

detail of dot-plots of inter-species chromosome pairs, such as those shown in Figure 5.6. We further compared our dot-plots with those generated by CoGe [88], a popular platform for comparative genomics analyses, under default parameter settings. Based on the DAGChainer [62] algorithm, CoGe provides functionality to identify genomic markers in pairs of genomes. Despite CoGe's method for identifying genomic markers being unrelated to ours, the dot-plots are similar, suggesting that the constructed genome segmentation is robust and unbiased.



(a) (b) (c)

Figure 5.6: **Dot-plot for chromosomes 1 of grape and 11 of coffee.** The plots are given by (a) CoGe [88], (b) our computed LASTZ alignments, and (c) after genome segmentation.

Based on 246 million pairwise local alignments reported by LASTZ, IMP derived 640 thousand atoms of minimum length 100bp which are associated with families occurring in two or more genomes. In comparison, the total amount of annotated genes in the five eudicots is around 140 thousand [10]. Table 5.1 shows for the same five eudicots the number of genes that have been used in ancestral reconstruction by Badouin *et al.* [10] and information on the annotated genes and genomic markers obtained from latest genome databases. We subsequently removed those markers/genes from their genomic sequences that were associated to families not shared by at least two genomes. That way we obtained 9,374 families from the set of annotated genes, with average size 6.5 and occurring in 4.1 genomes on average. For genomic markers, 123,218 families were derived, with average size 5.7 and occurring in 2.9 genomes on average.

Table 5.1: **Genes, markers and families in each of the five eudicots.** For each species, shared genes (markers) represents the amount of genes (markers) occurring in at least one other genome. Average family occurrences shows, for families occurring in at least two genomes, how many times each family occur on average in each genome.

	Badouin *et al.* (2017)	Annotated genes				Genomic markers			
	total genes	total genes	shared genes	families	avg. fam. occ.	total markers	shared markers	families	avg. fam. occ.
Grape	26,346	23,180	10,514	7,675	1.4	145,152	50,103	31,533	1.6
Coffee	25,574	21,971	13,267	9,374	1.4	97,735	34,125	23,598	1.4
Artichoke	27,121	23,394	11,124	7,034	1.6	396,323	153,448	92,401	1.7
Lettuce	12,841	37,829	11,249	7,032	1.6	860,023	178,217	83,806	2.1
Sunflower	52,243	58,022	14,604	7,300	2.0	1,364,948	223,500	93,526	2.4

Syntenic blocks

We extended Gecko3 by our method for computing local DCJ similarity scores, thereby quantifying structural similarities within approximate common intervals, which the original Gecko3 does not take into account. We have used in Gecko3 a *default* and a *relaxed* table (see Table 5.6 in the Section 5.C) to set indel thresholds depending on the size of the shared genome content of compared intervals. Using grape as reference genome, we ran Gecko3 with varying quorum, and default and relaxed indel thresholds. A list of results for each of those parameter settings is shown in Table 5.2. For the calculation of the local DCJ similarity scores of reported syntenic blocks, we set the deletion cost to $p = 0.25$ and the length threshold of function f (see Eq. (5.2)) to $L = 8$. Gecko3 reported 48,877 syntenic blocks for our final choice of parameter settings (see Table 5.2, run 2). Each such block occurred on average 1.0, 1.1, 1.6, 1.7, and 2.1 times in grape, coffee, artichoke, lettuce and sunflower, respectively. These values are compatible with the ancestral polyploidization events of their phylogeny.

Contiguous ancestral regions

Our reconstructed genome of the eudicot ancestor is composed of 32,788 markers distributed across 3,153 CARs, with the largest CARs comprising between 50 and 100 markers. This ancestral genome is in remarkably high agreement with that constructed by Badouin and colleagues [10], despite the fact that quite different sets of genomic markers have been used: By comparing the proportions of genomic markers attributed to each ancestral chromosome with the proportions derived from Badouin *et al.*'s gene-based reconstruction, the two ancestral genomes differ only 3.2% on average in absolute terms, with standard deviation of 3.7%. Figure 5.7 shows the comparison of ancestral genome content w.r.t. coffee and grape chromosomes of this analysis. The genome architecture of grape is closest to the post-γ ancestor, therefore the layout of the grape genome serves as proxy for reconstructing the genome of the post-γ ancestor in this chapter.

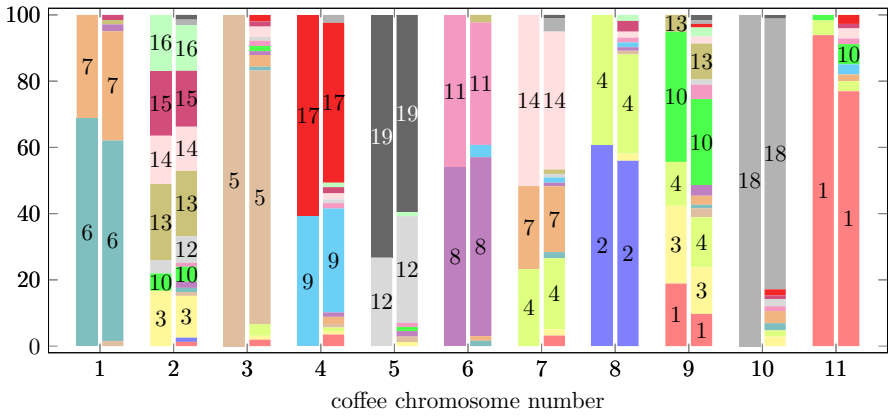


Figure 5.7: **Shared content between coffee and grape genomes in the reconstructed ancestor.** For each coffee chromosome (x-axis), each pair of bars shows the proportion shared with grape chromosomes (indicated by the color and chromosome number inside each bar segment) by the ancestral genome of Badouin *et al.* [10] (left) and ours (right), respectively. For better visualization, proportions of ancestral genome contents below 1% are not shown.

The method used for calculating the proportions of shared content between the grape genome and one of the other genomes in the reconstructed ancestor is as follows: Given a chromosome C of the other species, let S be the set of syntenic blocks occurring in C that are part of some CAR. We compute the percentage of blocks in S that also occur in each of the chromosomes in grape genome. As an example, for the chromosome 1 of coffee genome, the total number of syntenic blocks that are part of some CAR is 385 and, from this total, 233 blocks (60.52%) occur in chromosome 6 and 127 (32.99%) in chromosome 7 of grape genome. Based on these calculated proportions, the difference of the layouts given by our ancestral reconstruction and the ancestral reconstruction by Badouin *et al.* is used as a measure of how much the two reconstructions differ. Our measure is simply the average absolute difference over all chromosomes between our calculated proportions and those reported by Badouin *et al.* More specifically, we calculate the average of $|P - P_{\mathrm{B}}|$ for each proportion P_{B} reported by them and the corresponding proportion P by our method.

We investigated whether our family refinement approach using local genome rearrangement improved the ancestral reconstruction. We followed three different paths: First, we quantified the impact that the family refinement procedure has on the ancestral genome content. Second, to untangle the effects of this refinement procedure from marker-based vs. gene-based reconstruction, we re-ran our reconstruction pipeline, this time using the latest gene annotations of the five eudicot genomes. To this end, we constructed gene families as described by Salse [107] by binning genes with *cumulative identify percentage* (CIP) of 60% and *cumulative alignment length percentage* (CALP) of 70% [109]. Third, we quantified the *fixation* in ancestral marker order by measuring the average number of children of Q nodes in the PQ-tree constructed by ANGES.

The results obtained with these modified workflows make us believe that the family refinement indeed has a non-negligible positive effect: First, when skipping the local rearrangement-based family refinement procedure, the number of markers in the reconstructed ancestor amounted to 27,798. In other words, the family refinement led to an increase of 18% in ancestral genome content. Second, in the gene-based reconstruction, we observed similar results: Whereas local rearrangement-based family refinement led to 6,961 ancestral genes, without refinement their number decreased to 5,945. Third, the average number of children of Q nodes increased through rearrangement-based family refinement from 4.0 to 4.6 in marker-based reconstruction. Again, we observed the same trend in the gene-based reconstruction (4.2 without and 5.0 with refinement).

In addition, we studied the parameter space of our pipeline by conducting multiple runs listed in Table 5.2. By far the biggest impact w.r.t. the size of the ancestral genome content had the parameter settings of Gecko3, i.e., the choice of δ table, and the quorum parameter q (cf. runs 1, 2, and 7). ANGES weights syntenic blocks to guide the choice of discarding some of them in cases of conflict. We provided our local DCJ similarity scores as weights, but also ran ANGES on its internally computed weights, observing only minor differences, although surprisingly in favor of ANGES' weights (cf. runs 5 and 6). Furthermore, ANGES provides two different algorithms for reconstructing the PQ-tree: a heuristic (H) and a branch-and-bound (B) algorithm. Although the latter can recruit more markers into the ancestral genome content (cf. runs 3 and 4), it has a much higher running time, that only allowed us to compute ancestral reconstructions when we dramatically reduced the number of provided syntenic blocks. We limited then the number of overlapping syntenic blocks to 30 and chose (heuristically) promising subsets whenever this limit was exceeded.

Table 5.2: **Overview of ancestral reconstructions under varying parameters of our pipeline.** Our final choice of parameters is highlighted in gray.

run	Gecko3			family refinement	ANGES			ancestor	
	δ table	q	syntenic blocks		DCJ sim. weights	overlap limit.	alg.	markers	PQ-tree fixation
1	default	3	35,708	y	y	-	H	29,746	3.78
2	relaxed	3	48,877	y	y	-	H	32,350	4.62
3	relaxed	3	48,877	y	y	22,831	H	29,871	3.90
4	relaxed	3	48,877	y	y	22,831	B	30,212	3.81
5	relaxed	3	48,877	n	n	-	H	27,914	4.01
6	relaxed	3	48,877	n	y	-	H	27,798	4.03
7	relaxed	4	37,298	y	y	-	H	28,607	4.09

The number of syntenic blocks after this filtering step dropped to 22,831, reducing the number of markers and the fixation of the ancestral PQ-tree.

5.5 Concluding remarks

Recently, Badouin and colleagues reconstructed the eudicot ancestor from the gene annotations of grape, coffee, artichoke, lettuce and sunflower and arrived at an ancestral genome comprising 6,525 genes [10]. In this chapter, we followed the same workflow for ancestral reconstruction, but made multiple improvements: First, instead of using annotated genes, we identify genomic markers and use them as building blocks of the ancestral sequence, allowing us to reconstruct both intra- and intergenic blocks of DNA. Second, instead of using CloseUp [63], a statistical method for discovering syntenic blocks in pairs of genomic sequences, we use Gecko3 [131], which computes exact solutions under a principled definition of synteny [59, 77] in multiple sequences. Third, based on the local DCJ similarity introduced in this chapter, we score syntenic blocks and refine the family assignment of their contained genomic markers. Our improvements lead to a reconstruction of the ancestral eudicot genome that is composed of 32,788 markers distributed across 3,153 CARs. Remarkably, the layout of our ancestral genome differs on average only in 3.2% from that Badouin *et al.* [10]. Our method is also applicable to gene-based reconstruction, where it increased the genome content of the eudicot ancestor to 6,961 reconstructed genes while differing on average only in 4.6% from Badouin *et al.*'s reconstruction.

There is, however, room for immediate improvement on how is calculated this average difference (of proportions of shared content between coffee and other species chromosomes). Currently, we calculate the average of $|P - P_{\mathrm{B}}|$ for each proportion P_{B} reported by Badouin *et al.* [10] and the corresponding proportion P given by our method. Because we calculate these differences in absolute terms, whenever there are many residual small proportions they tend to dominate the average making it smaller, neglecting large shared regions between chromosomes. An alternative would be calculating the average of differences in relative terms with respect to Badouin *et al.* proportions, that is, taking their values as reference. For instance, the average of $|P - P_{\mathrm{B}}|/P_{\mathrm{B}} \times 100$. In this case however, differences in small proportions end up overly increasing the average (for instance, 2% differ in 100% from 1%). A better alternative could be using some kind of weighted average, nevertheless no best approach is clear so far.

Appendix

5.A Workflow, tools and data availability

The workflow implementation, named ANGORA, is publicly available at https://gitlab.ub.uni-bielefeld.de/gi/angora. All steps necessary to download and configure the workflow, its dependencies, and how it can be run are described in the provided README.md. The workflow package includes two small sample datasets, one having 3 unichromosomal genomes of simulated species, and one having 3 multichromosomal genomes of real species (*Ostreococcus* green algae).

Experimental data obtained in this ancestral genome reconstruction study is available at http://doi.org/10.4119/unibi/2936848.

Gecko3-DCJ is available at https://gitlab.ub.uni-bielefeld.de/gi/gecko-dcj.

GEESE is available at https://gitlab.ub.uni-bielefeld.de/gi/geese.

5.B Genome databases

This section describes datasets used in this chapter, including where they can be obtained and how they were prepared to be used with our workflow.

Data repositories

Following are the genome assemblies used in this study and where they can be obtained:

Grape *Vitis vinifera* (wine grape), assembly GCA_000003745.2 12X, NCBI, https://www.ncbi.nlm.nih.gov/genome/401?genome_assembly_id=214125

Coffee *Coffea canephora*, assembly v1.0, Coffee Genome Hub, http://coffee-genome.org/coffeacanephora

Artichoke Cynara cardunculus var. scolymus, assembly GCA_001531365.1 CcrdV1, NCBI, https://www.ncbi.nlm.nih.gov/genome/11286?genome_assembly_id=372115

Lettuce *Lactuca sativa* var Salinas, assembly L. sativa cv Salinas V8, Lettuce Genome Resource, http://lgr.genomecenter.ucdavis.edu/Private/Downloads/BulkDownload.php

Sunflower *Helianthus annuus* (common sunflower), assembly GCA_002127325.1 HanXRQr1.0, NCBI, https://www.ncbi.nlm.nih.gov/genome/351?genome_assembly_id=317475

Data preparation

Our workflow takes as input genome data in the form of GenBank files. However, the genome data obtained from different databases (e.g. JGI, NCBI, Ensembl) are in different formats. Conversion of these formats was necessary to standardize the dataset. Further, scaffolds not associated to any chromosome but which were also present in the data files have been filtered out. In detail, we have:

Sunflower: Removed scaffolds and added `locus_tag` to all `CDS` entries.

Grape: Removed scaffolds and added `locus_tag` to all `CDS` entries.

Artichoke: Removed scaffolds and added `locus_tag` to all `CDS` entries.

Coffee: A GenBank file was built from Fasta and GFF files by using the auxiliary script `fa+gff2gbk.py` included in the workflow under the `data/scripts` folder. 12,996 unmapped scaffolds (totaling 204 Mb) were removed.

Lettuce: A GenBank file was built from Fasta and GFF files by using the auxiliary script `fa+gff2gbk.py` included in the workflow under the `data/scripts` folder. Chromosome names had to be shortened, since they were too long and were causing errors in BioPython's GenBank writer.

5.C Parameters for tools used in this study

Here we describe the parameters for the ANGORA's family filtering step and for the integrated tools that are used in this ancestral genome reconstruction study. We also provide the locations where third party tools can be obtained.

LASTZ: local sequence alignment

The tool used for aligning DNA sequences is LASTZ, which can be obtained at https://github.com/lastz/lastz. However, LASTZ has a bug that, depending on the parameters choice, outputs inconsistent data. By the time this study was conducted, there was no public release correcting the bug. Therefore, our workflow contains a custom LASTZ hotfix version that can be found at https://gitlab.ub.uni-bielefeld.de/gi/lastz-hotfix. The complete list of LASTZ parameters can be found at https://lastz.github.io/lastz/. Parameters and their settings used in our eudicot study are:

```
--notransition
--step=10
--gapped
--hspthresh=6000
--nochain
--gfextend
--ambiguous=iupac
```

```
--masking=5
--filter=identity:70
```

Values under 6000 for `hspthresh` resulted in excessive noise for the eudicots dataset. Table 5.3 shows how to configure LASTZ parameters in the `config.yaml` file of our workflow.

Table 5.3: **Mapping of parameters for sequence alignment.**

LASTZ parameter	→	Workflow configuration (`config.yaml`) entry
any parameter `--parameter`		`lastz_params: --parameter ...`
any parameter `--parameter=<value>`		`lastz_params: --parameter=<value> ...`

GEESE: genome segmentation

For genome segmentation we have used GEESE, an efficient parallel implementation of the IMP algorithm [128] written in `C++`, which can be found at https://gitlab.ub.uni-bielefeld.de/gi/geese. The parameters used are:

- Minimum atom length (`--minLength`): 100;
- Minimum percent identity (`--minIdent`): 30;
- Minimum alignment block size (`--minAlnLength`): 13;
- Maximum gap length inside of an alignment (`--maxGap`): 100.

Table 5.4 shows how to configure GEESE parameters in the `config.yaml` file of our workflow.

Table 5.4: **Mapping of parameters for segmentation.**

GEESE parameter	→	Workflow configuration (`config.yaml`) entry
`--minLength <value>`		`marker_min_length: <value>`
`--minIdent <value>`		`sgmtn_alignment_ident: <value>`
`--minAlnLength <value>`		`sgmtn_alignment_minlen: <value>`
`--maxGap <value>`		`sgmtn_alignment_maxgap: <value>`

Filtering families

After the genome segmentation, the resulting families pass through a filtering step made by the script that post-processes the genome segmentation output (`atoms2cog.py`). In this filtering step, families that are too large or occur only once can be removed. We have filtered out families according to the following rules:

- Families larger than 98% of all the families (`--percent 98`);
- Families that have a single representative (`--ignore0`).

Table 5.5 shows how to configure family filtering parameters in the `config.yaml` file of our workflow.

Table 5.5: **Mapping of parameters for family filtering.**

`atoms2cog.py` parameter	→	Workflow configuration (`config.yaml`) entry
any parameter `--parameter`		`cog_params: --parameter ...`
any parameter `--parameter <value>`		`cog_params: --parameter <value> ...`

Gecko3-DCJ: discovering syntenic blocks

Syntenic blocks in our workflow are found by Gecko3-DCJ, which finds (referenced-based) approximate common intervals and quantifies their structural similarity by means of the local DCJ similarity score. A collection of intervals associated with genome content $\mathcal{G}$ is approximate common if the symmetric difference between the genome content of each interval and $\mathcal{G}$ is bounded by δ^{sum} and, more specifically, the number of excessive (i.e., *inserted*) markers is bounded by δ^{add}, and the number of missing markers by δ^{loss}. The two δ tables (default and relaxed) used in the eudicots study are shown in Table 5.6 (`-dT <table>` parameter). The quorum parameter q was set to 3 using the `-q 3` option. All Gecko3-DCJ options used in command line can also be set using its graphical interface.

Table 5.6: δ **tables used by Gecko3.**

	Default				Relaxed		
Size	δ^{add}	δ^{loss}	δ^{sum}	Size	δ^{add}	δ^{loss}	δ^{sum}
2	0	0	0	2	0	0	0
3	0	0	0	3	1	0	1
4	1	0	1	4	1	1	1
5	2	1	2	5	2	1	2
6	3	2	3	6	3	2	3
7	4	2	4	7	4	2	4
8	5	3	5	8	6	3	7
9	6	3	6	9	8	4	8

The local DCJ similarity formula as defined in the main text requires a function $f : 2\mathbb{N} \to \mathbb{R}$ that scores each cycle and path proportional to its length. In this study we have used the function

$$f(l) = \frac{2-l}{L-2} + 1\,, \tag{5.2}$$

where l is the length of the cycle or path and L is a length threshold that demarcates short from long cycles and paths (called *borderline cycle length* in Gecko3-DCJ). In the calculation of the local DCJ similarity, enabled in Gecko3-DCJ by the `--dcj` option, the value $L = 8$ was used (`--dcjBorderline 8`). See Figure 5.8 for an example of the function f for $L = 8$. We have also allowed the use of heuristics to compute the local DCJ similarity when gene families are too large (`--dcjUseHeuristics`). As for the deletion cost d, we have used the value 0.25 (`--dcjPenality 0.25`).

Table 5.7 shows how to configure Gecko3-DCJ parameters in the `config.yaml` file of our workflow.

ANGES: Ancestral genome reconstruction

In this last step of the pipeline, ANGESpy3 (a port of ANGES to Python 3) was used with default parameters. In the main experiments made for this work we provided to

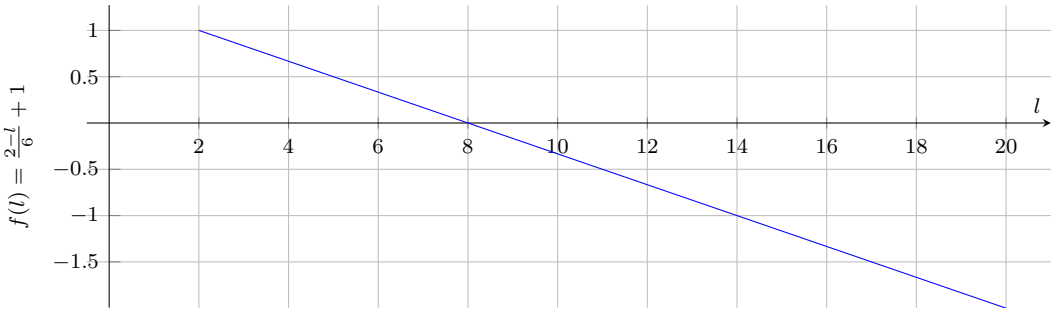


Figure 5.8: **Plot of function f as defined in Equation 5.2 for $L = 8$.** The horizontal axis represents the cycle or path length l and the vertical axis represents the score $f(l)$.

Table 5.7: **Mapping of parameters for discovery of syntenic blocks.**

Gecko3-DCJ parameter	→	Workflow configuration (`config.yaml`) entry
any parameter `--parameter`		`gecko_params: --parameter ...`
any parameter `--parameter <value>`		`gecko_params: --parameter <value> ...`

ANGESpy3 syntenic block scores calculated by the average local DCJ similarity between the block occurrence in the reference genome (grape) and all block occurrences in other species. Besides, the heuristic algorithm was used to reconstruct the PQ-tree. ANGESpy3 can be downloaded from https://gitlab.ub.uni-bielefeld.de/gi/angespy3.

Table 5.8 shows how to configure ANGESpy3 behavior in the `config.yaml` file of our workflow.

Table 5.8: **Mapping of parameters for ancestral reconstruction.**

ANGESpy3 behavior	→	Workflow configuration (`config.yaml`) entry
Use local DCJ similarity scores for blocks		`anges_use_sim_weight: True`
Compute scores for blocks (do not use the local DCJ simimarity score)		`anges_use_sim_weight: False`
Use heuristics + branch-and-bound		`anges_run_bab: True`
Use heuristics only		`anges_run_bab: False`

Chapter 6

Conclusion

After presenting some background on the subject of this work and preliminary definitions, in Chapter 3 we have studied the family-based DCJ distance for genomes with duplicate genes. This problem is known to be NP-hard [115], despite having no formal proof. We have proposed an $O(k)$-approximation algorithm for the restrict case of balanced unichromosomal genomes [102, 103]. This limitation is due to an intermediate step that approximates the minimum common string partition problem [60, 83], which guarantees the approximation ratio of our algorithm. The approximation algorithm has linear running time in the size of the genomes. A natural extension to this algorithm would be approximating the problem for unbalanced genomes, multichromosomal genomes, or both. Another use for the approximation could be providing an initial lower bound to the ILP solver. In addition, the hardness of this restricted case (balanced unichromosomal genomes) yet has to be investigated.

Next, in Chapter 4, we have presented formally the (NP-hard) problem of computing the family-free DCJ similarity, showing its APX-hardness and a lower bound for approximation ratios (unless P = NP) [104, 106]. Following, we have proposed an exact ILP algorithm and four combinatorial heuristics to solve it, with computational experiments comparing the results obtained by the heuristics and by the ILP solver [104, 106]. While the ILP is fast and accurate for smaller instances, it cannot solve larger instances due to the number of restrictions, which is cubic in the size of the input genomes. On the other hand, the heuristics obtained good results for a number of instances, some of them even better than the results returned by the ILP solver after reaching the time limit. Moreover, the ILP could benefit greatly from heuristics by using their outputs as initial lower bounds. One drawback of the similarity function as defined in this work is that distinct pairs of genomes might give family-free DCJ similarity values that cannot be compared easily, because the similarity value varies between 0 and $|M|$, where M is the matching giving rise to the similarity value. Therefore some kind of normalization would be desirable. A simple approach could be to divide the similarity value obtained by the size of the smaller genome, because this is a trivial upper bound for $|M|$. Moreover, it can be applied as a simple postprocessing step, keeping all theoretical results of this work valid. A better normalization, however, might be to divide by $|M|$ itself. An analytical treatment here seems more difficult, though.

Lastly, in Chapter 5 we have proposed the local DCJ similarity, a local genome rearrangement based measure [105]. Then, we have shown how to use the local DCJ similarity to improve a popular ancestral genome reconstruction workflow, among other modifications. Results for the eudicot ancestor reconstruction from grape, coffee, artichoke, lettuce and sunflower show that our reconstructed genome is highly detailed, yet its layout agrees well with that reported in Badouin *et al.* [10]. Using our local genome rearrangement measure in both marker-based and gene-based reconstructions of the eudicot ancestor exhibited increased genome content, evidencing the power of this novel concept. Further investigation of theoretical aspects of the local DCJ similarity (e.g. hardness, exact algorithms, approximations) and other useful applications for this measure are subjects of future work related to the content of that chapter.

List of publications derived from this doctoral study

D. P. Rubert, P. Feijão, M. D. V. Braga, J. Stoye, and F. V. Martinez. A linear time approximation algorithm for the DCJ distance for genomes with bounded number of duplicates. In *Proc. of the 16th International Workshop on Algorithms in Bioinformatics (WABI 2016)*, pages 293–306, 2016. [103]

D. P. Rubert, P. Feijão, M. D. V. Braga, J. Stoye, and F. H. V. Martinez. Approximating the DCJ distance of balanced genomes in linear time. *Algorithms for Molecular Biology*, 12(1):3, 2017. [102]

D. P. Rubert, G. L. Medeiros, E. A. Hoshino, M. D. V. Braga, J. Stoye, and F. V. Martinez. Algorithms for computing the family-free genomic similarity under DCJ. In *Proc. of the 15th RECOMB Comparative Genomics Satellite International Workshop (RECOMB-CG 2017)*, pages 76–100, 2017. [106]

D. P. Rubert, E. A. Hoshino, M. D. V. Braga, J. Stoye, and F. V. Martinez. Computing the family-free DCJ similarity. *BMC Bioinformatics*, 19(6):152, 2018. [104]

D. P. Rubert, F. H. V. Martinez, J. Stoye, and D. Doerr. Analysis of local genome rearrangement improves resolution of ancestral genomic maps in plants. *BMC Genomics*, 2019, to appear (Proc. of the 17th RECOMB Comparative Genomics Satellite International Workshop RECOMB-CG 2019). [105]

Bibliography

[1] M. Abrouk, F. Murat, C. Pont, J. Messing, S. Jackson, T. Faraut, E. Tannier, C. Plomion, R. Cooke, C. Feuillet, et al. Palaeogenomics of plants: synteny-based modelling of extinct ancestors. *Trends in Plant Science*, 15(9):479–487, 2010.

[2] B. L. Aken, S. Ayling, D. Barrell, L. Clarke, V. Curwen, S. Fairley, J. Fernandez Banet, K. Billis, C. García Girón, T. Hourlier, K. Howe, A. Kähäri, F. Kokocinski, F. J. Martin, D. N. Murphy, R. Nag, M. Ruffier, M. Schuster, Y. A. Tang, J.-H. Vogel, S. White, A. Zadissa, P. Flicek, and S. M. J. Searle. The ensembl gene annotation system. *Database*, 2016, 2016.

[3] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. Efficient tools for computing the number of breakpoints and the number of adjacencies between two genomes with duplicate genes. *Journal of Computational Biology*, 15(8):1093–1115, 2008.

[4] S. Angibaud, G. Fertin, I. Rusu, A. Thévenin, and S. Vialette. On the approximability of comparing genomes with duplicates. *Journal of Graph Algorithms and Applications*, 13(1):19–53, 2009.

[5] S. Angibaud, G. Fertin, I. Rusu, and S. Vialette. A pseudo-boolean framework for computing rearrangement distances between genomes with duplicates. *Journal of Computational Biology*, 14(4):379–393, 2007.

[6] Y. Anselmetti, N. Luhmann, S. Bérard, E. Tannier, and C. Chauve. Comparative methods for reconstructing ancient genome organization. In J. C. Setubal, J. Stoye, and P. F. Stadler, editors, *Comparative Genomics: Methods and Protocols*, Methods in Molecular Biology, pages 343–62. Humana Press, 2017.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[8] P. Avdeyev, S. Jiang, S. Aganezov, F. Hu, and M. A. Alekseyev. Reconstruction of ancestral genomes in presence of gene gain and loss. *Journal of Computational Biology*, 23(3):150–64, Mar 2016.

[9] M. Bader and E. Ohlebusch. Sorting by weighted reversals, transpositions, and inverted transpositions. *Journal of Computational Biology*, 14(5):615–636, 2007.

[10] H. Badouin, J. Gouzy, C. J. Grassa, F. Murat, S. E. Staton, L. Cottret, C. Lelandais-Brière, G. L. Owens, S. Carrère, B. Mayjonade, L. Legrand, N. Gill, N. C. Kane, J. E.

Bowers, S. Hubner, A. Bellec, A. Bérard, H. Bergès, N. Blanchet, M.-C. Boniface, D. Brunel, O. Catrice, N. Chaidir, C. Claudel, C. Donnadieu, T. Faraut, G. Fievet, N. Helmstetter, M. King, S. J. Knapp, Z. Lai, M.-C. Le Paslier, Y. Lippi, L. Lorenzon, J. R. Mandel, G. Marage, G. Marchand, E. Marquand, E. Bret-Mestries, E. Morien, S. Nambeesan, T. Nguyen, P. Pegot-Espagnet, N. Pouilly, F. Raftis, E. Sallet, T. Schiex, J. Thomas, C. Vandecasteele, D. Varès, F. Vear, S. Vautrin, M. Crespi, B. Mangin, J. M. Burke, J. Salse, S. Muños, P. Vincourt, L. H. Rieseberg, and N. B. Langlade. The sunflower genome provides insights into oil metabolism, flowering and Asterid evolution. *Nature*, 546(7656):148–52, Jun 2017.

[11] V. Bafna and P. Pevzner. Genome rearrangements and sorting by reversals. In *Proc. of the 34th Annual Symposium on Foundations of Computer Science (FOCS 1993)*, pages 148–157, 1993.

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

[13] A. Bergeron, S. Corteel, and M. Raffinot. The algorithmic of gene teams. In *Proc. of the 2nd International Workshop on Algorithms in Bioinformatics (WABI 2002)*, pages 464–476, 2002.

[14] A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. In S. Miyano, J. Mesirov, S. Kasif, S. Istrail, P. A. Pevzner, and M. Waterman, editors, *Proc. of the 9th International Conference on Research in Computational Molecular Biology (RECOMB 2005)*, pages 615–629, 2005.

[15] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. In *Proc. of the 6th International Workshop on Algorithms in Bioinformatics (WABI 2006)*, pages 163–173, 2006.

[16] A. Bergeron, J. Mixtacki, and J. Stoye. HP distance via double cut and join distance. In P. Ferragina and G. M. Landau, editors, *Proc. of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, pages 56–68, 2008.

[17] P. Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. In *Proc. of the 7th Scandinavian Workshop on Algorithm Theory (SWAT 2000)*, pages 214–219, 2000.

[18] P. Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. *Nordic Journal of Computing*, 7(3):178–184, 2000.

[19] P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proc. of the 26th International Colloquium on Automata, Languages, and Programming (ICALP'99)*, pages 200–209, 1999.

[20] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness of short symmetric instances of MAX-3SAT. *Electronic Colloquium on Computational Complexity (ECCC)*, 2003. Report TR03-049.

[21] G. Blin, G. Fertin, and C. Chauve. The breakpoint distance for signed sequences. In *Proc. of the 1st Conference on Algorithms and Computational Methods for biochemical and Evolutionary Networks (CompBioNets' 04)*, volume 3, pages 3–16, 2004.

[22] M. D. V. Braga, C. Chauve, D. Dörr, K. Jahn, J. Stoye, A. Thévenin, and R. Wittler. The potential of family-free genome comparison. In C. Chauve, N. El-Mabrouk, and E. Tannier, editors, *Models and Algorithms for Genome Evolution*, chapter 13, pages 287–307. Springer, 2013.

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

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

[25] B. Brejová, M. Burger, and T. Vinař. Automated segmentation of DNA sequences with complex evolutionary histories. In *Proc. of the 11th International Workshop on Algorithms in Bioinformatics (WABI 2011)*, pages 1–13, 2011.

[26] D. Bryant. The complexity of the breakpoint median problem. Technical Report CRM-2579, Centre de Recherches Mathématiques, Université de Montréal, 1998.

[27] D. Bryant. The complexity of calculating exemplar distances. In D. Sankoff and J. H. Nadeau, editors, *Comparative Genomics*, pages 207–211. Kluwer Academic Publishers, 2000.

[28] L. Bulteau, G. Fertin, and I. Rusu. Sorting by transpositions is difficult. *SIAM Journal on Discrete Mathematics*, 26(3):1148–1180, 2012.

[29] L. Bulteau and M. Jiang. Inapproximability of (1,2)-exemplar distance. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(6):1384–1390, 2013.

[30] A. Caprara. Sorting by reversals is difficult. In *Proc. of the 1st International Conference on Research in Computational Molecular Biology (RECOMB '97)*, pages 75–83. ACM, 1997.

[31] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[32] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[33] Z. Chen, B. Fu, J. Xu, B. Yang, Z. Zhao, and B. Zhu. Non-breaking similarity of genomes with gene repetitions. In *Proc. of the 18th Annual Symposium on Combinatorial Pattern Matching (CPM 2007)*, pages 137–143, 2007.

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

[35] D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, 1998.

[36] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Transactions on Algorithms*, 3(1):2:1–2:19, 2007.

[37] P. Crescenzi. A short guide to approximation preserving reductions. In *Proc. of the 12th Annual IEEE Conference Computational Complexity (CCC'97)*, pages 262–273, 1997.

[38] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Information and Computation*, 93(2):241–262, 1991.

[39] M. Csurös. Count: evolutionary analysis of phylogenetic profiles with parsimony and likelihood. *Bioinformatics*, 26(15):1910–1912, 2010.

[40] L. F. I. Cunha, L. A. B. Kowada, R. d. A. Hausen, and C. M. H. de Figueiredo. A faster 1.375-approximation algorithm for sorting by transpositions. *Journal of Computational Biology*, 22(11):1044–1056, 2015.

[41] D. A. Dalquen, M. Anisimova, G. H. Gonnet, and C. Dessimoz. Alf – a simulation framework for genome evolution. *Molecular Biology and Evolution*, 29(4):1115, 2012.

[42] Z. Dias and J. Meidanis. Genome rearrangements distance by fusion, fission, and transposition is easy. In *Proc. of the 8th Symposium on String Processing and Information Retrieval (SPIRE 2001)*, pages 250–253, 2001.

[43] D. Doerr, L. A. B. Kowada, E. Araujo, S. Deshpande, S. Dantas, B. M. E. Moret, and J. Stoye. New genome similarity measures based on conserved gene adjacencies. *Journal of Computational Biology*, 2017.

[44] D. Doerr, J. Stoye, S. Bocker, and K. Jahn. Identifying gene clusters by discovering common intervals in indeterminate strings. *BMC Genomics*, 15(Suppl 6):S2, 2014.

[45] D. Doerr, A. Thévenin, and J. Stoye. Gene family assignment-free comparative genomics. *BMC Bioinformatics*, 13(Suppl 19):S3, 2012.

[46] W. Duchemin, Y. Anselmetti, M. Patterson, Y. Ponty, S. Bérard, C. Chauve, C. Scornavacca, V. Daubin, and E. Tannier. DeCoSTAR: Reconstructing the ancestral organization of genes or genomes using reconciled phylogenies. *Genome Biology and Evolution*, 9(5):1312–19, May 2017.

[47] D. Earl, N. Nguyen, G. Hickey, R. S. Harris, S. Fitzgerald, K. Beal, I. Seledtsov, V. Molodtsov, B. J. Raney, H. Clawson, et al. Alignathon: a competitive assessment of whole-genome alignment methods. *Genome Research*, 24(12):2077–2089, 2014.

[48] 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 *Proc. of the 4th International Workshop on Algorithms in Bioinformatics (WABI 2004)*, pages 1–13, 2004.

[49] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[50] I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(4):369–379, 2006.

[51] N. Eriksen. $(1+\varepsilon)$-approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517 – 529, 2002.

[52] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. of the 38th IEEE Annual Symposium on Foundations of Computer Science (FOCS 1997)*, pages 137–143, 1997.

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

[54] C. E. Ferreira, C. Fernandes, F. Miyazawa, J. Soares, J. Pina Jr, K. Guimarães, M. Carvalho, M. Cerioli, P. Feofiloff, R. Dahab, and Y. Wakabayashi. Uma introdução sucinta a algoritmos de aproximação. *Colóquio Brasileiro de Matemática-IMPA, Rio de Janeiro-RJ*, 2001.

[55] G. Fertin, A. Labarre, I. Rusu, É. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. MIT press, 2009.

[56] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183 – 198, 1983.

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

[58] M. B. Gerstein, C. Bruce, J. S. Rozowsky, D. Zheng, J. Du, J. O. Korbel, O. Emanuelsson, Z. D. Zhang, S. Weissman, and M. Snyder. What is a gene, post-encode? history and updated definition. *Genome Research*, 17(6):669–681, 2007.

[59] C. G. Ghiurcuta and B. M. E. Moret. Evaluating synteny for improved comparative studies. *Bioinformatics*, 30(12):i9–18, Jun 2014.

[60] A. Goldstein, P. Kolman, and J. Zheng. Minimum common string partition problem: Hardness and approximations. *Eletronic Journal of Combinatorics*, 12(R50), 2005.

[61] Q.-P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327 – 339, 1999.

[62] B. J. Haas, A. L. Delcher, J. R. Wortman, and S. L. Salzberg. DAGchainer: a tool for mining segmental genome duplications and synteny. *Bioinformatics*, 20(18):3643–6, 2004.

[63] S. E. Hampson, B. S. Gaut, and P. Baldi. Statistical detection of chromosomal homology using shared-gene density alone. *Bioinformatics*, 8(21):1339–48, 2005.

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

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

[66] 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.

[67] R. S. Harris. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, Pennsylvania State University, 2007.

[68] 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.

[69] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[70] K. A. Hawick and H. A. James. Enumerating circuits and loops in graphs with self-arcs and multiple-arcs. Technical Report CSTN-013, Massey University, 2008.

[71] X. He and M. H. Goldwasser. Identifying conserved gene clusters in the presence of homology families. *Journal of Computational Biology*, 12(6):638–656, 2005.

[72] S. Heber, R. Mayr, and J. Stoye. Common intervals of multiple permutations. *Algorithmica*, 60(2):175–206, 2011.

[73] S. Heber and J. Stoye. Algorithms for finding gene clusters. In *Proc. of the 1st International Workshop on Algorithms in Bioinformatics (WABI 2001)*, pages 252–263, 2001.

[74] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.

[75] F. Hu, J. Zhou, L. Zhou, and J. Tang. Probabilistic reconstruction of ancestral gene orders with insertions and deletions. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 11(4):667–72, 2014.

[76] Y.-L. Huang, C.-C. Huang, C. Y. Tang, and C. L. Lu. An improved algorithm for sorting by block-interchanges based on permutation groups. *Information Processing Letters*, 110(8):345 – 350, 2010.

[77] K. Jahn. Efficient computation of approximate gene clusters based on reference occurrences. *Journal of Computational Biology*, 18(9):1255–1274, 2011.

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

[79] H. Jiang, C. Zheng, D. Sankoff, and B. Zhu. Scaffold filling under the breakpoint distance. In *Proc. of the 8th RECOMB Comparative Genomics Satellite International Workshop (RECOMB-CG 2010)*, volume 6398 of *Lecture Notes on Bioinformatics*, pages 83–92, 2010.

[80] D. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[81] B. R. Jones, A. Rajaraman, E. Tannier, and C. Chauve. ANGES: Reconstructing ANcestral GEnomeS maps. *Bioinformatics*, 28(18):2388–90, 2012.

[82] H. Kaplan and N. Shafrir. The greedy algorithm for edit distance with moves. *Information Processing Letters*, 97(1):23–27, 2006.

[83] P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. *The Electronic Journal of Combinatorics*, 14(1):R50, 2007.

[84] B. Larget, J. B. Kadane, and D. L. Simon. A bayesian approach to the estimation of ancestral genome arrangements. *Molecular Phylogenetics and Evolution*, 36(2):214–23, Aug. 2005.

[85] G.-H. Lin and G. Xue. Signed genome rearrangement by reversals and transpositions: models and approximations. *Theoretical Computer Science*, 259(1):513 – 531, 2001.

[86] 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.

[87] C. 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(1):295, 2006.

[88] E. Lyons and M. Freeling. How to usefully compare homologous plant genes and chromosomes as DNA sequences. *The Plant Journal*, 53(4):661–73, Jan 2008.

[89] F. V. Martinez, P. Feijão, M. D. V. Braga, and J. Stoye. On the family-free DCJ distance and similarity. *Algorithms for Molecular Biology*, 10:13, 2015.

[90] J. Meidanis, M. E. M. T. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. Technical report, University of Campinas and University of Brasília, Campinas, Brazil, 2000.

[91] C. Mira and J. Meidanis. Sorting by block-interchanges and signed reversals. In *Proc. of the 4th International Conference on Information Technology (ITNG'07)*, pages 670–676, 2007.

[92] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the SIAM*, 5(1):32–28, 1957.

[93] F. Murat, R. Zhang, S. Guizard, H. Gavranović, R. Flores, D. Steinbach, H. Quesneville, E. Tannier, and J. Salse. Karyotype and gene order evolution from reconstructed extinct ancestors highlight contrasts in genome plasticity of modern rosid crops. *Genome Biology and Evolution*, 7(3):735–49, 2015.

[94] S. Ohno. *Sex chromosomes and sex-linked genes*, volume 1 of *Monographs on Endocrinology*. Springer, 2013.

[95] A. Ouangraoua and A. Bergeron. Combinatorial structure of genome rearrangements scenarios. *Journal of Computational Biology*, 17(9):1129–1144, 2010.

[96] M. Ozery-Flato and R. Shamir. Two notes on genome rearrangement. *Journal of Bioinformatics and Computational Biology*, 1(01):71–94, 2003.

[97] C. Pont, S. Wagner, A. Kremer, L. Orlando, C. Plomion, and J. Salse. Paleogenomics: reconstruction of plant evolutionary trajectories from modern and ancient DNA. *Genome Biology*, 20(1):29, Feb 2019.

[98] P. Portin and A. Wilkins. The evolving definition of the term "gene". *Genetics*, 205(4):1353–1364, 2017.

[99] A. J. Radcliffe, A. D. Scott, and E. L. Wilmer. Reversals and transpositions over finite alphabets. *SIAM Journal on Discrete Mathematics*, 19(1):224–244, 2005.

[100] S. Rahmann and G. W. Klau. Integer linear programs for discovering approximate gene clusters. In *Proc. of the 6th International Workshop on Algorithms in Bioinformatics (WABI 2006)*, pages 298–309, 2006.

[101] V. Raman, B. Ravikumar, and S. S. Rao. A simplified NP-complete MAXSAT problem. *Information Processing Letters*, 65(1):1 – 6, 1998.

[102] D. P. Rubert, P. Feijão, M. D. V. Braga, J. Stoye, and F. H. V. Martinez. Approximating the DCJ distance of balanced genomes in linear time. *Algorithms for Molecular Biology*, 12(1):3, 2017.

[103] D. P. Rubert, P. Feijão, M. D. V. Braga, J. Stoye, and F. V. Martinez. A linear time approximation algorithm for the DCJ distance for genomes with bounded number of duplicates. In *Proc. of the 16th International Workshop on Algorithms in Bioinformatics (WABI 2016)*, pages 293–306, 2016.

[104] D. P. Rubert, E. A. Hoshino, M. D. V. Braga, J. Stoye, and F. V. Martinez. Computing the family-free DCJ similarity. *BMC Bioinformatics*, 19(6):152, 2018.

[105] D. P. Rubert, F. H. V. Martinez, J. Stoye, and D. Doerr. Analysis of local genome rearrangement improves resolution of ancestral genomic maps in plants. *BMC Genomics*, 2019. to appear (Proc. of the 17th RECOMB Comparative Genomics Satellite International Workshop RECOMB-CG 2019).

[106] D. P. Rubert, G. L. Medeiros, E. A. Hoshino, M. D. V. Braga, J. Stoye, and F. V. Martinez. Algorithms for computing the family-free genomic similarity under DCJ. In *Proc. of the 15th RECOMB Comparative Genomics Satellite International Workshop (RECOMB-CG 2017)*, pages 76–100, 2017.

[107] J. Salse. Ancestors of modern plant crops. *Current Opinion in Plant Biology*, 30:134–42, 2016.

[108] J. Salse, M. Abrouk, F. Murat, U. M. Quraishi, and C. Feuillet. Improved criteria and comparative genomics tool provide new insights into grass paleogenomics. *Briefings in Bioinformatics*, 10(6):619–30, 2009.

[109] J. Salse, S. Bolot, M. Throude, V. Jouffe, B. Piegu, U. M. Quraishi, T. Calcagno, R. Cooke, M. Delseny, and C. Feuillet. Identification and characterization of shared duplications between rice and wheat provide new insight into grass genome evolution. *The Plant Cell*, 20(1):11–24, 2008.

[110] D. Sankoff. Edit distance for genome comparison based on non-local operations. In *Proc. of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM 1992)*, pages 121–135, 1992.

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

[112] D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. In *Proc. of the 3rd Annual International Conference on Computing and Combinatorics (COCOON 1997)*, pages 251–263, 1997.

[113] T. Schmidt and J. Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Proc. of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004)*, pages 347–358, 2004.

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

[115] M. Shao, Y. Lin, and B. Moret. An exact algorithm to compute the double-cut-and-join distance for genomes with duplicate genes. *Journal of Computational Biology*, 22(5):425–435, 2015.

[116] D. Shapira and J. A. Storer. Edit distance with move operations. In A. Apostolico and M. Takeda, editors, *Proc. of the 13th Annual Symposium Combinatorial Pattern Matching (CPM 2002)*, pages 85–98, 2002.

[117] D. Shapira and J. A. Storer. Edit distance with move operations. *Journal of Discrete Algorithms*, 5(2):380 – 392, 2007.

[118] B. Shapiro, A. Rambaut, and A. J. Drummond. Choosing appropriate substitution models for the phylogenetic analysis of protein-coding sequences. *Molecular Biology and Evolution*, 23(1):7–9, 2006.

[119] D. E. Soltis, V. A. Albert, J. L. Mack, C. D. Bell, A. H. Paterson, C. Zheng, D. Sankoff, C. W. de Pamphilis, P. K. Wall, and P. S. Soltis. Polyploidy and angiosperm diversification. *American Journal of Botany*, 96(1):336–48, Jan. 2009.

[120] K. Swenson, M. Marron, K. Earnest-DeYong, and B. M. E. Moret. Approximating the true evolutionary distance between two genomes. In *Proc. of the 7th Workshop on Algorithm Engineering and Experiments and the 2nd Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALCO 2005)*, pages 121–129, 2005.

[121] R. J. Taft, M. Pheasant, and J. S. Mattick. The relationship between non-protein-coding DNA and eukaryotic complexity. *BioEssays*, 29(3):288–299, 2007.

[122] H. Tang, J. E. Bowers, X. Wang, R. Ming, M. Alam, and A. H. Paterson. Synteny and collinearity in plant genomes. *Science*, 320(5875):486–8, Apr. 2008.

[123] J. Tang, B. M. E. Moret, L. Cui, and C. W. dePamphilis. Phylogenetic reconstruction from arbitrary gene-order data. In *Proc. of the 4th IEEE Symposium on Bioinformatics and Bioengineering (BIBE 2004)*, pages 592–599. IEEE Press, 2004.

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

[125] E. Tannier, C. Zheng, and D. Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinformatics*, 10(1):120, 2009.

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

[127] N. Tomizawa. On some techniques useful for solution of transportation network problems. *Networks*, 1(2):173–194, 1971.

[128] M. Visnovská, T. Vinař, and B. Brejová. DNA sequence segmentation based on local similarity. In *Proc. of the 13th Conference on Information Technologies - Applications and Theory (ITAT 2013)*, pages 36–43, 2013.

[129] M. E. M. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *Proc. of String Processing and Information Retrieval: A South American Symposium*, pages 96–102. IEEE, 1998.

[130] E. Willing, S. Zaccaria, M. D. V. Braga, and J. Stoye. On the inversion-indel distance. *BMC Bioinformatics*, 14(S-15):S3, 2013.

[131] S. Winter, K. Jahn, S. Wehner, L. Kuchenbecker, M. Marz, J. Stoye, and S. Böcker. Finding approximate gene clusters with GECKO 3. *Nucleic Acids Research*, 44(20):9600–10, 2016.

[132] G. A. Wu, S. Prochnik, J. Jenkins, J. Salse, U. Hellsten, F. Murat, X. Perrier, M. Ruiz, S. Scalabrin, J. Terol, M. A. Takita, K. Labadie, J. Poulain, A. Couloux, K. Jabbari, F. Cattonaro, C. Del Fabbro, S. Pinosio, A. Zuccolo, J. Chapman, J. Grimwood, F. R. Tadeo, L. H. Estornell, J. V. Muñoz-Sanz, V. Ibanez, A. Herrero-Ortega, P. Aleza, J. Pérez-Pérez, D. Ramón, D. Brunel, F. Luro, C. Chen, W. G. Farmerie, B. Desany, C. Kodira, M. Mohiuddin, T. Harkins, K. Fredrikson, P. Burns, A. Lomsadze, M. Borodovsky, G. Reforgiato, J. Freitas-Astúa, F. Quetier, L. Navarro, M. Roose, P. Wincker, J. Schmutz, M. Morgante, M. A. Machado, M. Talón, O. Jaillon, P. Ollitrault, F. Gmitter, and D. Rokhsar. Sequencing of diverse mandarin, pummelo and orange genomes reveals complex history of admixture during citrus domestication. *Nature Biotechnology*, 32(7):656–62, Jun 2014.

[133] A. W. Xu and B. M. E. Moret. GASTS: Parsimony scoring under rearrangements. In *Proc. of the 11th International Workshop on Algorithms in Bioinformatics (WABI 2011)*, pages 351–63, 2011.

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

[135] M. Q. Zhang. Computational prediction of eukaryotic protein-coding genes. *Nature Reviews Genetics*, 3(9):698–709, 2002.

[136] C. Zheng, Y. Jeong, M. G. Turcotte, and D. Sankoff. Resolution effects in reconstructing ancestral genomes. *BMC Genomics*, 19(Suppl 2):100, May 2018.