Gene Family-free Genome Comparison

Ph. D. Thesis

submitted to the
Faculty of Technology,
Bielefeld University, Germany
for the degree of Dr. rer. nat.

by

Daniel Dörr

March, 2015

Supervisor:
Prof. Dr. Jens Stoye

Referees:
Prof. Dr. Jens Stoye
Prof. Dr. David Sankoff
Prof. Dr. Cedric Chauve

Gedruckt auf alterungsbeständigem Papier nach DIN-ISO 9706.
Printed on non-aging paper according to DIN-ISO 9706.

Zusammenfassung

Die *rechnergestützte vergleichende Genomik* gewährt wertvolle Einsichten in die gemeinsame und individuelle evolutionäre Historie von lebenden und ausgestorbenen Spezies. Genome zu vergleichen bedeutet deren Unterschiede zu bestimmen, die durch Mutationen in ihrer evolutionären Vergangenheit entstanden sind.

Im Bereich der Genomevolution differenziert man zwischen *Punktmutationen*, *Genomumordnungen* und Änderungen des *Gengehalts* von Genomen. Punktmutationen verändern ein oder wenige aufeinanderfolgende Nukleotide in der DNA-Sequenz. Genomumordnungen ändern die Reihenfolge und Aufteilung von Genen in chromosomale Sequenzen. Der Gengehalt wird durch die Evolution von Genfamilien beeinflusst, welche zu Genduplikationen oder dem Verlust von Genen führt.

Studien zur Erforschung von Genomumordnungen zwischen Genomen setzen die Kenntnis der evolutionären Verhältnisse zwischen deren Genen voraus. Mittels des biologischen Konzepts der *Homologie* kann die Menge aller Gene in Genfamilien unterteilt werden: Alle Gene in einer Genfamilie sind paarweise homolog zueinander, was bedeutet, dass sie von einer gemeinsamen Ursequenz abstammen. Homologien zwischen Genen sind in der Regel unbekannt und werden daher häufig mit rechnergestützten Methoden vorhergesagt. Dazu werden Sequenzähnlichkeiten zwischen Genen oder andere Ähnlichkeiten in den Eigenschaften ihrer Genprodukte quantifiziert. Allerdings ist die Vorhersage von Homologien häufig unzuverlässig, was zu Fehlern in einer anschließenden Studie von Genomumordnungen führt.

Diese Doktorarbeit verfolgt einen recht jungen Forschungszweig mit der Zielsetzung, Fehler durch falsche oder unvollständige Vorhersagen von Genfamilien in der Untersuchung von Genomumordnungen zu vermeiden. Dazu werden neue rechnergestützte Methoden zur Erforschung von Genomumordnungen entwickelt, die die Kenntnis von Genfamilien nicht voraussetzen. Dieser Ansatz, auch genannt *genfamilienfreier Genomvergleich*, ist innovativ, da Unterschiede zwischen Genen, welche durch Punktmutationen entstanden sind, in der Untersuchung von Genomumordnungen berücksichtigt werden. Anstelle von vorhergesagten Genfamilien greift der vorgestellte Ansatz direkt auf Genähnlichkeiten zurück, welche üblicherweise zur Vorhersage von Genfamilien verwendet werden.

Die Endpunkte zweier Gene, die auf einer chromosomalen Sequenz nebeneinander liegen, bilden eine *Nachbarschaft*. Die Anzahl *konservierter Nachbarschaften*, das heißt Nachbarschaften, welche zwei untersuchten Genomen gemein sind, können als Maß zur Quantifizierung ihrer Ähnlichkeit verwendet werden. Wenn der Gengehalt beider Genome identisch ist, dann ist die Anzahl konservierter Nachbarschaften das duale Maß zur *Breakpoint-Distanz*. Diese Doktorarbeit untersucht das Problem zur Berechnung der Anzahl konservierter Nachbarschaften im Rahmen des genfamilienfreien Genomvergleichs. Dazu wird die Berechnungskomplexität analysiert und es werden exakte und heuristische Verfahren entwickelt, die den paarweisen Genomvergleich ermöglichen.

Des Weiteren wird die Problematik der Rekonstruktion von Ursequenzen im Rahmen des genfamilienfreien Genomvergleichs betrachtet. Die vorliegende Arbeit untersucht das Problem, ein viertes Genom, *Median* genannt, anhand drei gegebener Genome zu konstruieren, was die Anzahl paarweiser konservierter Nachbarschaften maximiert. Hierbei wird das Modell des *gemischten multichromosomalen Breakpoint-Medians* verallgemeinert. Anschließend wird die NP-Schwere des Problems bewiesen und ein exakter Algorithmus zur Berechnung einer Lösung vorgestellt.

Mit der Länge des evolutionären Zeitraums steigt die Zahl der Genomumordnungen, welche die Genordnung zunehmend durcheinanderbringen. Aus diesem Grund sind Studien über evolutionär weit entfernte Genome, die konservierte Nachbarschaften identifizieren, nicht aufschlussreich. Dennoch können verallgemeinerte Definitionen konservierter Genordnung ein schwächeres, aber dennoch vorhandenes Signal gemeinsamer Genordnung auffangen. Dies ist Gegenstand eines Forschungszweigs, welcher sich mit der Identifikation *syntenischer Bereiche* beschäftigt. Wenn Genfamilien bekannt sind, dann lässt sich eine chromosomale Sequenz als Zeichenfolge (String) über dem Alphabet von Genfamilienbezeichnungen darstellen. Ein Paar von Intervallen in zwei Strings wird *Common Intervals* genannt, wenn ihre Zeichenmenge identisch ist. Auf dieser Definition beruht ein Modell zur Bestimmung syntenischer Bereiche in zwei oder mehr Genomen. In dieser Doktorarbeit wird die Definition von *Common Intervals* auf *Indeterminate Strings* erweitert. *Indeterminate Strings* sind Sequenzen, in denen jede Position aus einer nicht-leeren Zeichenmenge besteht. In der vorliegenden Arbeit werden mehrere Modelle von *Common Intervals* für *Indeterminate Strings* vorgestellt und effiziente Algorithmen für das Auffinden entsprechender Intervallpaare in zwei *Indeterminate Strings* vorgestellt. Die neu entwickelten Algorithmen werden anschließend dazu verwendet, syntenische Bereiche im Rahmen des genfamilienfreien Genomvergleichs zu bestimmen.

Alle vorgestellten Modelle und Algorithmen werden an simulierten oder biologischen Datensätzen evaluiert und ihre Eignung für den genfamilienfreien Genomvergleich untersucht.

Abstract

Computational comparative genomics offers valuable insights into the shared and individual evolutionary histories of living and extinct species and expands our understanding of cellular processes in living cells. Comparing genomes means identifying differences that originated from mutational modifications in their evolutionary past.

In studying genome evolution, one differentiates between *point mutations*, *genome rearrangements*, and *content modifications*. Point mutations affect one or few consecutive nucleotide bases in the DNA sequence, whereas genome rearrangements operate on larger genomic regions, thereby altering the order and composition of genes in chromosomal sequences. Lastly, content modifications are a result of gene family evolution that causes gene duplications and losses.

Genome rearrangement studies commonly assume that evolutionary relationships between all pairs of genes are resolved. Based on the biological concept of *homology*, the set of genes can be partitioned into *gene families*. All genes in a gene family are homologous, i.e., they evolved from the same ancestral sequence. Homology information is generally not given, hence gene families are commonly predicted computationally on the basis of sequence similarity or higher order features of their gene products. These predictions are often unreliable, leading to errors in subsequent genome rearrangement studies.

In an attempt to avoid errors resulting from incorrect or incomplete gene family assignments, we develop new methods for genome rearrangement studies that do not require prior knowledge of gene family assignments of genes. Our approach, called *gene family-free genome comparison*, is innovative in that we account for differences between genes caused by point mutations while studying their order and composition in chromosomes. In lieu of gene family assignments, our proposed methods rely on pairwise similarities between genes. In practice, we obtain gene similarities from the conservation of their protein sequences.

Two genes that are located next to each other on a chromosome are said to be adjacent, their adjoining extremities form an *adjacency*. The number of *conserved adjacencies*, i.e., those adjacencies that are common to two genomes, gives rise to a measure for gene order-based genome similarity. If the gene content of both

genomes is identical, the number of conserved adjacencies is the dual measure of the well-known *breakpoint distance*. We study the problem of computing the number of conserved adjacencies in a family-free setting, which relies on pairwise similarities between genes. We analyze its computational complexity and develop exact and heuristic algorithms for its solution in pairwise comparisons.

We then advance to the problem of reconstructing ancestral sequences. Given three genomes, we study the problem of constructing a fourth genome, called the *median*, which maximizes a family-free, pairwise measure of conserved adjacencies between the median and each of the three given genomes. Our model is a family-free generalization of the well-studied *mixed multichromosomal breakpoint median*. We show that this problem is NP-hard and devise an exact algorithm for its solution.

Gene orders become increasingly scrambled over longer evolutionary periods of time. In distant genomes, gene order analyses based on identifying pairs of conserved adjacencies might no longer be informative. Yet, relaxed constraints of gene order conservation are still able to capture weaker, but nonetheless existing remnants of common ancestral gene order, which leads to the problem of identifying syntenic blocks in two or more genomes. Knowing the evolutionary relationships between genes, one can assign a unique character to each gene family and represent a chromosome by a string drawn from the alphabet of gene family characters. Two intervals from two strings are called *common intervals* if the sets of characters within these intervals are identical. We extend this concept to *indeterminate strings*, which are a class of strings that have at every position a non-empty set of characters. We propose several models of common intervals in indeterminate strings and devise efficient algorithms for their corresponding discovery problems. Subsequently, we use the concept of common intervals in indeterminate strings to identify syntenic regions in a gene family-free setting.

We evaluate all our proposed models and algorithms on simulated or biological datasets and assess their performance and applicability in gene family-free genome analyses.

Acknowledgements

For the past four years of PhD student life, I was supported by many people who I now like to express my sincere gratitude.

First of all, I am genuinely thankful to my advisor, Jens Stoye, who introduced me to this fascinating project. He taught me how to do research and generously shared his deep knowledge and expertise that continues to inspire me also in personal life. His endless support made the successful completion of this work possible. I was indeed fortunate to have him as my advisor.

I am deeply thankful to Cedric Chauve, who hosted me for three months at the Simon Fraser University in spring 2014 and committed precious time and energy to my project and to me. His critical insights guided me through major parts of this thesis.

My thanks to James H. Kaufman, in who's lab I spent the summer of 2012 at the IBM Almaden Research Center in San Jose. James introduced me to the field of healthcare informatics and taught me a different approach to research, both of which I am very thankful for.

I would like to thank current and former members of the Genome Informatics group, in particular Pedro Feijão for many valuable discussions, especially in the recent months, Roland Wittler for reviewing, Annelyse Thévenin and Katharina Jahn for being helpful co-authors and inspiring mentors.

I am indebted to many people that influenced me on the long path which shaped my decision to study bioinformatics. In particular, I would like to thank my father, who raised my interest in computers at a young age, Simon Pamiés for introducing me to computer programming, and my inspiring biology teacher Martin Heilen — if it were not for him, I certainly would not have studied anything related to biology.

My special thanks go to 'Die WG': Anne Reh, Christian Munier, David Ries, Florian Sprengel, and Julian Mayland. The three years of living with you in a shared flat have been fantastic. Without you, being a PhD student would not have been as enjoyable as it truly was! I am indebted to my parents who supported me in all those years. I especially thank my wonderful girlfriend Vera Surall for her endless love, care, and encouragement.

Finally, I would like to acknowledge funding from the CLIB-Graduate Cluster Industrial Biotechnology that granted me a three year scholarship and supplied me with generous travel funds. My visit in Cedric Chauve's lab was funded by the German Academic Exchange Service (DAAD), and for the past ten months I was funded by a scholarship of the Genome Informatics group.

Contents

Chapter 1

Introduction

With today's abundance of electronically available genomic sequences, the field of *computational comparative genomics* continuously contributes decisive insights to the understanding of genome evolution of living and extinct species. Introduced by David Sankoff [92, 96] and Joseph H. Nadeau and Benjamin A. Taylor [78], the field gives rise to a broad variety of *in silico* methods to study the structural organization of genomes. Such studies do not only lead to improved knowledge of the species' phylogeny, but also hint at interactions within and between sets of genes by means of their involvement in metabolic and regulatory networks.

Genomes evolve through various types of mutations: point mutations affect one or few nucleotide bases; genome rearrangements alter the order and partition of genes into chromosomes; genes become duplicated or lost as a result of gene family evolution; lastly, whole genome effects such as duplications of chromosomes or whole genomes dramatically alter the gene content of the organism. Due to the varying mechanisms of these mutations and their characteristic impact on genomic sequences, individual lines of research formed over the past decades, each specializing in the study of a certain type of mutation.

Initial approaches to study genome rearrangements considered pairwise comparisons with well identified one-to-one relationships between orthologous genes [92], for many of which polynomial time algorithms for computing distances and evolutionary scenarios could be designed [15, 16, 47, 94, 115]. David Sankoff initiated formulations and algorithms for genome rearrangement problems with duplicated genes originating from unrestricted homology assignments [93], quickly followed by the outline of a general approach that would consider both gene orders and gene trees as input to genome rearrangement problems [97]. Since then, genome rearrangement with unequal gene content, where genomes are represented by signed sequences, has been intensively explored; for reviews see [30, 41].

Extending pairwise genome rearrangement studies to three or more genomes leads to the problem of reconstructing ancestral gene orders. Thereby, one differen-

tiates between the *small parsimony problem*, if a phylogeny is given along which one aims to construct the most parsimonious gene orders of branching ancestral states, and the *big parsimony problem*, where the phylogeny itself is subject to optimization, too. Yet small parsimony problems under most rearrangement distances are already hard problems, even for the simplest cases that ask for the construction of a median genome from three known gene orders, given one-to-one orthology assignments [17, 25, 29, 83, 94, 114]. Notable exceptions include the *mixed multichromosomal breakpoint median* [106] and the median under the *single cut or join* distance [40]. Despite the computational challenges, software tools such as MGR [25], MGRA [2], and GRAPPA [76] have been developed and enable the reconstruction of ancestral gene orders within acceptable running times in practice. Sankoff *et al.* initiated in [95] the study of ancestral gene order reconstruction under a more general genome model that allows genomes to exhibit unequal gene content, including duplicated genes. However, this branch of research remains largely unexplored up to a few exceptions [97, 106, 116].

Another line of research studies broader notions of conserved gene order by identifying *syntenic blocks*, which are pairs of genomic segments of similar gene content, whereby no or only weak constraints are imposed on the order of the contained genes. If, in addition, these genes share functional relationships, syntenic blocks are also called *gene clusters*. Initial efforts to discover syntenic blocks required the identification of one-to-one relationships between orthologous genes [14, 49, 50], similar to early rearrangement studies. Most of these approaches were subsequently adapted to a more general genome model permitting gene duplications and losses [35, 48, 99].

All of the above methods, that we call *gene family-based*, require prior gene family assignments. However, biological gene families are difficult to assess; commonly, they are predicted computationally. The outcome of such efforts depends heavily on parameter choices in sequence comparison, similarity quantification and clustering. These parameters are user-controlled and influence the size and granularity of computed gene families. In particular, when genes within biological gene families are largely diverged, computational means may not be able to resolve gene family assignments accurately [42]. Consequently, errors are introduced into the primary dataset which deteriorate subsequent analyses.

In an attempt to avoid these errors, we propose new methods for genome comparison that do not require prior gene family assignments.

Thesis overview

In this thesis, we study three different rearrangement problems and introduce their family-free counterparts.

After a short recapitulation of biological concepts and the introduction of basic notations in Chapter 2, we study in Chapter 3 the problem of maximizing family-free adjacencies in pairwise genome comparisons. Two genes that are located next to each other on a chromosome are said to be adjacent, their adjoining extremities form an *adjacency*. The number of *conserved adjacencies*, i.e., those adjacencies that are common to two genomes, gives rise to a measure for gene order-based genome similarity. If the gene content of both genomes is identical, the number of conserved adjacencies is the dual measure of the well-known *breakpoint distance* [92]. We study the problem of computing the number of conserved adjacencies in a family-free setting, which relies on pairwise similarities between genes. We analyze its computational complexity and develop exact and heuristic algorithms for its solution in pairwise comparisons. Subsequent experiments on simulated and biological datasets demonstrate the practical applicability of our family-free model and our algorithms.

In Chapter 4, we introduce the problem of constructing a mixed multichromosomal breakpoint median of three genomes in a family-free setting. We then prove its NP-hardness and develop an exact algorithm for its solution. We further present a heuristic method that shows improved resistance against perturbances in the gene order caused by events of gene duplication and loss. We subsequently evaluate our family-free model and the developed algorithms in experiments on simulated and biological datasets.

In Chapter 5, we present models and algorithms for the family-free identification of *syntenic blocks*. To this end, we make use of a recent definition of syntenic blocks suggested by Ghiurcuta and Moret [46]. We formulate the problem of identifying syntenic blocks as a problem of discovering *common intervals in indeterminate strings*. We then present several common interval-based models on indeterminate strings and devise fast polynomial-time algorithms for their corresponding discovery problems. The chapter closes with experiments on a biological dataset comprising 93 bacterial genomes.

Lastly, we give a short conclusion and present future directions of this work in Chapter 6.

Several parts of this thesis have been published in advance: The general concepts and the analysis of the biological dataset of Chapter 3 were published in [38], [26], and [65]. Further, Chapter 5 appeared in [37] and [26].

Chapter 2

Background

In this chapter, we review basic biological concepts required for the understanding of subsequently presented family-free models of genome comparison. To this end, we discuss the organization of genetic information on DNA sequences located inside each living cell. We then describe the processes and effects of evolution on DNA sequences, which make genome comparison a challenging subject of research.

Our studies are based on a simple abstract genome model, which is subsequently introduced in this chapter. It then follows a discussion of the family-free principle in which we introduce the *gene similarity graph* that represents another fundamental data structure of this work.

2.1 Genetic information

Life on earth is classified into three major domains: *bacteria*, *archaea*, and *eukaryotes*, whereby *viruses* reside on the borderline of what is consentaneously considered as *life*. A *species* consists of a population of organisms whose genetic information that adheres to a common gene pool. The genetic information of an organism is encoded in its *genome*, which is physically represented by DNA sequences located inside each of its cells (with exception to certain differentiated cells of eukaryotic organisms). Whereas the genome of a prokaryotic cell is enclosed in a nucleus, *bacteria*, *archaea*, and *viruses* lack cellular compartmentalization. Eukaryotic cells contain additional genomic sequences in *organelles*, such as *mitochondria*.

The genetic information of the cell is organized in a highly complex manner. Figure 2.1 depicts the four main levels of genetic information, which can be hierarchically arranged into *sequence*, *gene order*, *gene family*, and *epigenetics/spatial structure*. Starting at the lowest level, nucleotide bases are the building blocks of DNA sequences, each composed of two anti-parallel strands. These sequences are often condensed through *histones* or histone-like proteins into highly compact structures forming linear or circular chromosomes. The chromosomal DNA accommodates

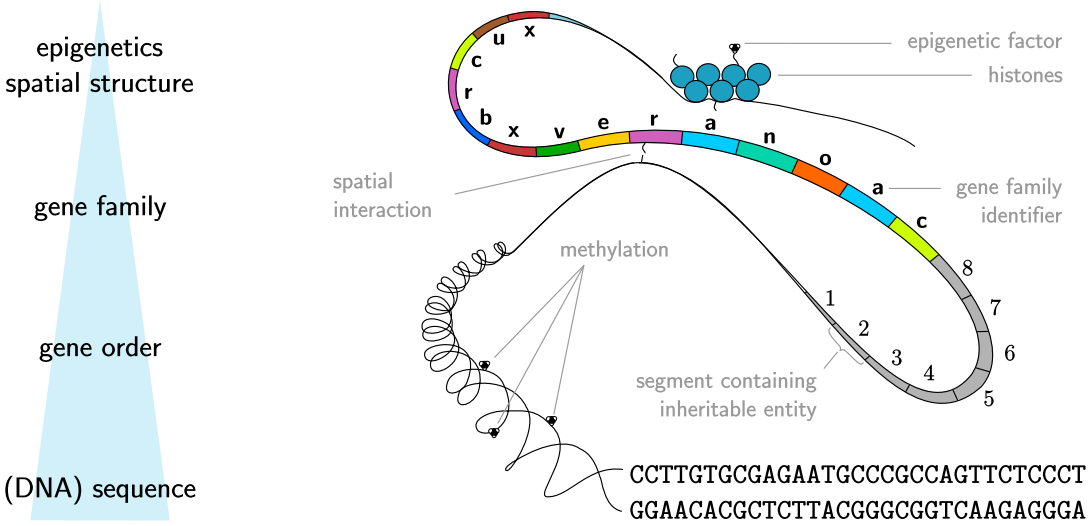


Figure 2.1: The four main levels of genetic information.

inheritable entities that contribute to a certain function in the cell and evolve under natural selection. In this work, we call a segment that is associated with an inheritable entity a *gene*. The succession of genes and their partition into chromosomal sequences comprises a higher level of genetic information known as *gene order*. Genes may overlap or are interspersed by segments that are under no selective pressure. Moreover, they are usually not independent of each other. Their order on the chromosome has a crucial influence on the organism's cellular processes. Genes are imperfectly passed on from one generation to the next. Their degree of conservation over many generations varies largely, depending on their influence on the organism's viability and fitness. Two or more genes originating from the same ancestral sequence are called *homologous* [60]. A set of homologous genes is called a *gene family*. Gene family information presupposes knowledge of segments corresponding to inheritable entities. It establishes a higher order of genetic information, which consists of evolutionary relationships between genes. Lastly, spatial structure resulting from the condensed conformation of the chromosome, as well as epigenetic factors, such as DNA or histone methylation, contribute to the genetic information of the organism.

This work studies the interaction of the lower two levels of genetic information, namely sequence and gene order information, thereby resolving either necessary basic or implicit gene family information.

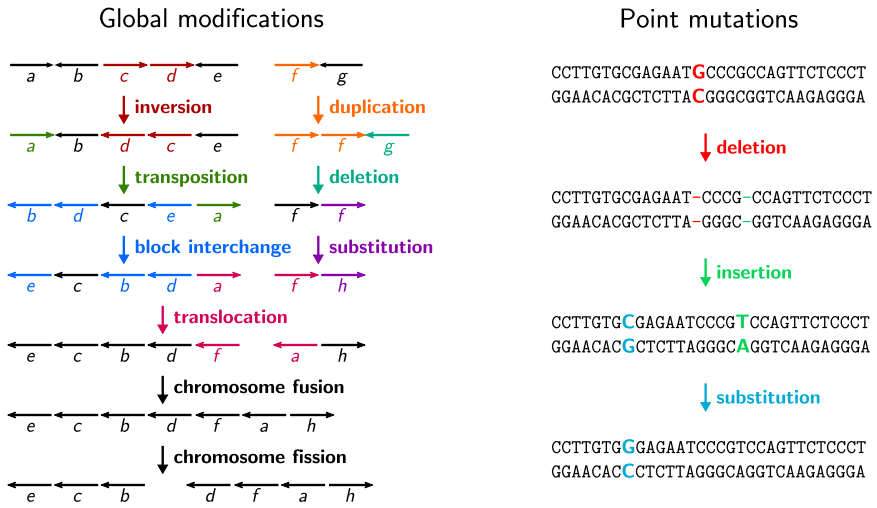


Figure 2.2: Mutational changes in genomes can be classified into global modifications that act on one ore more genes, and local modifications, called point mutations that affect on one or few nucleotides.

2.2 Evolutionary modifications

Over an evolutionary period of time, genomic DNA sequences accumulate *mutations*, that are generally classified into *global modifications* and *point mutations* (see Figure 2.2). The latter comprise mutations that affect one or few consecutive nucleotide bases in the DNA sequence through substitution, deletion, or insertion. Global modifications operate on larger genomic regions, thereby affecting one or more genes. The associated mutational operations can be differentiated in *genome rearrangements* such as inversions, transpositions, block interchanges, translocations, chromosome fusions and fissions, chromosome circularizations and linearizations, and *content modifications* such as substitutions, duplications, insertions, and deletions of one or more genes. The study of global modifications is subject to *gene order analysis*. In this work we perform gene order analyses in which we do not explicitly model genome rearrangements or content modifications, but merely study their impact on the gene order. That is, we identify *syntenic regions*, which are regions of conserved gene order in two or more genomes of related species. In doing so, we often assume that global modifications happen *parsimoniously*, i.e., every *occurred* mutation can also be unanimously *observed* in the DNA sequence. In other words, we assume that the smallest number of global mutational changes that transforms one DNA sequence into another can explain their true evolutionary history. This is not generally true in applied gene order analyses of real biological sequences. Phenomena such as *breakpoint reuse* [11, 98], or *parallel* and *convergent evolution* [53, 73, 77, 91] violate the parsimony principle. Nevertheless, the evolutionary rate of global modifications is

slow, therefore such effects are considerably rare in close and intermediate evolutionary histories [74, 100].

2.3 Evolutionary relationships

Establishing relationships between gene orders of different species presumes knowledge of relationships between their genes. In evolutionary biology [60], a set of genes is called *homologous* if and only if they evolved from the same ancestral sequence. The specific evolutionary relationships of homologous genes can be described by a *gene tree*. Therein, each gene is represented by a leaf label. Starting at the common ancestral sequence of all genes in the set, which corresponds to the root node, genes branch off along the evolutionary path to the leaves. Each internal node of the gene tree corresponds to an evolutionary event, by which the common ancestry of two or more genes diverges into separate paths. These events can be generally classified into *speciation*, *duplication*, and *horizontal gene transfer*. In doing so, two genes are *orthologous* if their lowest common ancestor in the gene tree corresponds to a speciation event, and *paralogous*, if it represents a duplication event. Lastly, genes are *xenologous* if their common evolutionary path forked through horizontal gene transfer.

Example 1 *Figure 2.3 (a) exemplifies duplication, speciation, and horizontal gene transfer in a gene tree of five genes, $A1$, $A2$, $B1$, $B2$, and $C2$, of species A, B, and C. Gene pairs $(A1, A2)$, $(B1, B2)$, $(A1, B2)$, $(A1, C2)$, $(A2, B1)$, and $(B1, C2)$ are paralogs, whereas gene pairs $(A1, B1)$ and $(A2, B2)$ are orthologs. Gene pairs $(A2, C2)$ and $(B2, C2)$ are xenologs. Figure 2.3 (b) visualizes the gene tree embedded in its corresponding species tree.*

Knowledge of exact evolutionary relationships between genes is extremely valuable in gene order analysis. Such information is generally not given, but is commonly predicted. Yet, genome-wide gene tree reconstruction is computationally expensive. Most commonly, predicted evolutionary relationships are provided in the shape of a *gene family assignment*:

Definition 1 (gene family assignment) *Let Σ be the universe of genes, a* gene family assignment *$\mathcal{H} \subseteq \Sigma \times \Sigma$ is an* equivalence relation *between genes. The* equivalence class *of $x \in \Sigma$, i.e., the set of genes belonging to the same family as gene x under gene family assignment $\mathcal{H}$, is denoted as*

$$[x]_{\mathcal{H}} \equiv \{y \in \Sigma \mid (x, y) \in \mathcal{H}\}. \tag{2.1}$$

Recall that an equivalence relation is *reflexive*, i.e., $(x, x) \in \mathcal{H}$ for all x in Σ, (2) *symmetric*, i.e., for every (x, y) in $\mathcal{H}$ there exists also (y, x) in $\mathcal{H}$, and (3) *transitive*, i.e., for every $\{(x, y), (y, z)\} \subseteq \mathcal{H}$ there also exists $(x, z) \in \mathcal{H}$. In other words, a gene

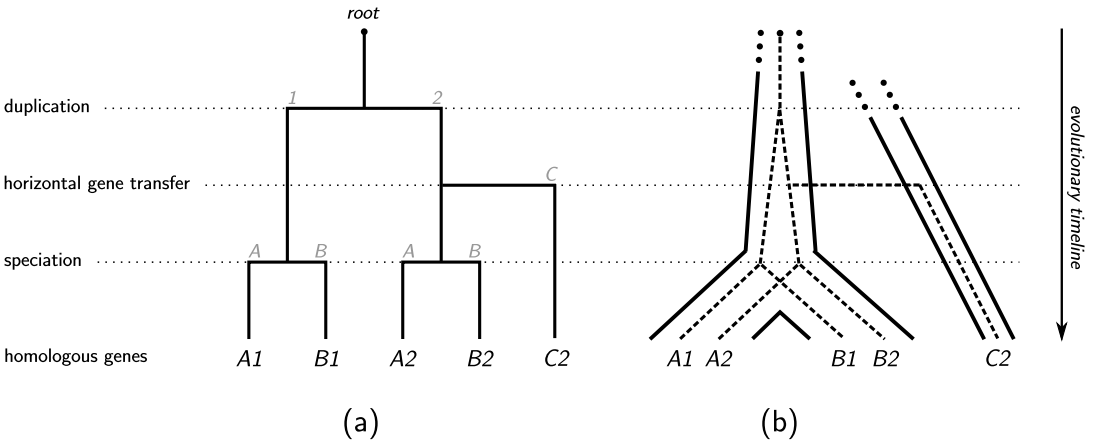


Figure 2.3: (a) Gene tree of five genes comprising three species; (b) diagram of the gene tree (dashed lines) embedded in its species tree (outer continuous lines).

family assignment $\mathcal{H}$ is a partition of genes (into non-intersecting subsets). It is obvious that a set of homologous genes always forms a gene family, hence these terms are often used synonymously. However, it is worthwhile differentiating between homologies and gene families, as there exist several different notions of the term *gene family* in the literature. For instance, molecular evolutionists such as Tomoko Ohta [79] differentiate between *multi-gene families* and *super families*, whereby in the former, genes are not only homologous, but also have related biological functions. Further, protein coding genes often comprise multiple domains, which can be shared between multi-gene families.

Other uses of the term gene family are practically motivated: It is common to call genes that are clustered into groups according to their sequence similarity, *co-orthologs* or *gene families*. These often correspond to subtrees of larger gene families. Various databases exist, e.g. COG [107], eggNOG [85], OrthoDB [111] — only to name a few — that offer information of precomputed gene families of protein coding genes. Since gene duplication and sub- or neofunctionalization occurs frequently in evolution, the number of homologous genes in a genome that are pooled into the same gene family grows the higher one ascends in the evolutionary tree. With increasing number of diverse genomes, these gene families become less useful for gene order analyses, if only a close subset of taxa is of interest. The blemish of missing gene trees needed for truly resolving evolutionary relationships between genes is often covered in databases by offering varying levels of granularity. This means that for some subsets of species in a database (but generally not for all), gene families are re-computed with tighter parameters. Moreover, the computed sequence-based similarity estimates are rarely based on models of sequence evolution as these involve considerably higher computational costs. Subsequently, dif-

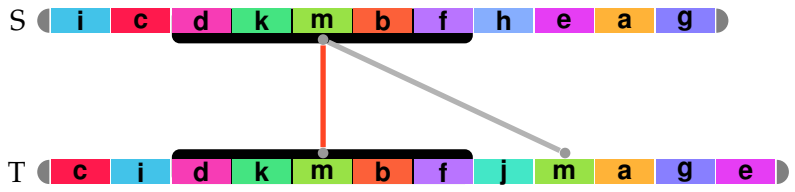


Figure 2.4: Gene **m** in gene order sequence S has two orthologous counterparts in sequence T. The pair of positional homologs is connected by an orange line. The syntenic region around the pair is highlighted in black.

ferential evolutionary rates are disregarded, amplifying the dilemma of predicting gene families.

Tree-based databases such as TreeFam [67] and OrthologID [31] may provide more accurate information desired for gene order studies. They also tend to be more often manually curated than their sequence similarity-based counterparts. In return, the provided gene family information is often sparse and covers not all genes of a genome. Moreover, such databases usually comprise only a handful of species. As a result, they are of limited use in gene order studies.

Several software tools like OrthoMCL [68], InParanoid [80], or MultiMSOAR [101] are freely available and allow for direct computation of gene family assignments in a dataset of interest. Typically, these approaches assume that gene families naturally cluster into densely connected subgraphs in the gene similarity network. However, this is not always the case: Low-quality sequences obtained by next generation sequencing may artificially reduce edge weights, and even prevent edges from appearing in the network at all. Moreover, protein coding genes that comprise multiple domains can have strong ties not only to their own family but also to other families they share a domain with. Some of these genes may not be at all traceable back to a single gene family. While some recent approaches can deal with the ambiguities caused by multi-domain proteins [58, 103], it is still a major challenge to define cut-offs in the clustering process that at the same time eliminate spurious granularity [42, 70].

The varying notations and the practical difficulties of gene family prediction compel to differentiate between *gene family assignment* and *homology assignment*:

Definition 2 (homology assignment) *Let Σ be the universe of genes, a* homology assignment *$\mathcal{H} \subseteq \Sigma \times \Sigma$ is a* reflexive *and* symmetric *mapping between genes. The set of genes belonging to the same gene family as gene x under homology assignment $\mathcal{H}$, is given by its* transitive closure,

$$[x]^+_{\mathcal{H}} \equiv \{x\} \cup \bigcup_{\substack{y \in \Sigma \setminus \{x\} \\ (x,y) \in \mathcal{H}}} [y]^+_{\mathcal{H}} \,. \tag{2.2}$$

On the one side, a homology assignment can be seen as an incomplete gene family assignment, because the constraint of transitivity is lifted; on the other side, it can be used to additionally link certain members of a super family belonging to different gene families, such as *positional homologs* [28], or *main orthologs/exemplars* [93]. Recently, detailed evolutionary relationships became less a prerequisite, but a result of gene order analyses [19, 43, 117]. This development is founded on the concept of *positional homology* [28, 34]. In a duplication event, a gene gets copied into a new location of the genome. That is to say, one of the duplicates retains its position on the chromosome, whereas the other integrates elsewhere. When comparing the duplicates with a third genome containing the gene, the pair of orthologs within the syntenic region is called a pair of *positional homologs* [28] (see Figure 2.4). Positional homologs are more likely to be functionally related than not positionally conserved pairs of orthologs [28]. They establish one-to-one relations between genes of two or more genomes that indicate ancestral gene orders. Hence, in the following, we mainly focus on a restricted homology assignment:

Definition 3 (one-to-one homology/gene family assignment) *A* one-to-one homology assignment $\mathcal{H}_1$ *is a homology assignment such that for each gene* $g \in \Sigma$*, there exists no other gene* g' *in* $[g]^+_{\mathcal{H}_1}$ *belonging to the same genome as* g*. If in addition* $\mathcal{H}_1$ *is transitive, it is a* one-to-one gene family assignment*.*

A one-to-one homology assignment $\mathcal{H}_1$ does not take the conservation of gene order into account.

2.4 Genome model

In the following, we introduce a genome model that facilitates the study of gene orders on an abstract level. In doing so, we do not impose any requirements on the type and resolution of inheritable entities that are considered genes in practical studies, except that they must be non-overlapping segments with a defined start and end position on the DNA sequence. Several software tools and methods for gene prediction are available that identify genomic features suitable for the genome model described in the following [59, 71, 90, 104].

2.4.1 Genomes, chromosomes, and genes

In this work, a genome G is entirely represented by a tuple $G \equiv (\mathcal{C}, \mathcal{A})$, where $\mathcal{C}$ denotes a non-empty set of unique genes, and $\mathcal{A}$ is a set of *adjacencies*, which represent the immediate gene neighborhoods. In doing so, we assume that genes are non-overlapping, although this is not generally true for genes of real biological sequences. Genes are represented by their *extremities*, i.e., a gene $g \equiv (g^{\mathrm{t}}, g^{\mathrm{h}})$, $g \in \mathcal{C}$, consists of a *head* g^{h} and a *tail* g^{t}, where h and t are called *terminals*. The

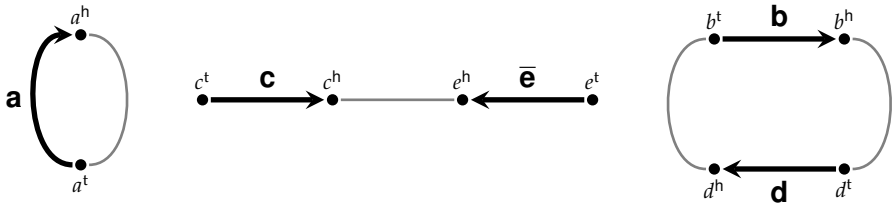


Figure 2.5: Graph representation of a genome, in which vertices correspond to gene extremities, thick black directed edges represent genes, and thin gray undirected edges depict adjacencies. Edge labels of genes indicate the canonical reading direction of each chromosome.

set of adjacencies $\mathcal{A}$ consists of non-intersecting unordered pairs of gene extremities: $\{x^a, y^b\} \subseteq \bigcup \mathcal{C}$, with $a, b \in \{\mathrm{h}, \mathrm{t}\}$, (where $\bigcup \mathcal{C}$ is a short form of writing $\bigcup_{(g^{\mathrm{t}}, g^{\mathrm{h}}) \in \mathcal{C}} \{g^{\mathrm{t}}, g^{\mathrm{h}}\}$). In the following, we will conveniently use the notation $\mathcal{C}(G)$ and $\mathcal{A}(G)$ to denote the set of genes and the set of adjacencies of genome G, respectively. Further, we define the *size of a genome* as the number of its genes $|G| \equiv |\mathcal{C}(G)|$.

Genomes can be represented by a multigraph, in which $\bigcup \mathcal{C}$ represents the set of vertices and $\mathcal{C} \cup \mathcal{A}$ the set of edges. In doing so, edges of set $\mathcal{C}$, which connect extremities *within* genes, are directed, pointing to the genes' heads. Edge set $\mathcal{A}$, which links extremities *between* genes, is undirected.

Example 2 *$G = (\mathcal{C}, \mathcal{A})$ with $\mathcal{C} = \{a, b, c, d, e\}$ and $\mathcal{A} = \{\{a^{\mathrm{t}}, a^{\mathrm{h}}\}, \{c^{\mathrm{h}}, e^{\mathrm{h}}\}, \{d^{\mathrm{t}}, b^{\mathrm{h}}\}, \{b^{\mathrm{t}}, d^{\mathrm{h}}\}\}$ is a genome. $\mathcal{C}$ can be more explicitly written in terms of gene extremities as $\mathcal{C} = \{(a^{\mathrm{t}}, a^{\mathrm{h}}), (b^{\mathrm{t}}, b^{\mathrm{h}}), (c^{\mathrm{t}}, c^{\mathrm{h}}), (d^{\mathrm{t}}, d^{\mathrm{h}}), (e^{\mathrm{t}}, e^{\mathrm{h}})\}$. The graph representation of genome G is shown in Figure 2.5.*

Further, we introduce the notation of *subgenomes*. A *subgenome* G' of genome G corresponds to a set of non-intersecting *subsequences* of G:

Definition 4 (subgenome) *Genome G' is a* subgenome *of genome G, denoted by $G' \subseteq G$, iff $\mathcal{C}(G') \subseteq \mathcal{C}(G)$ and for each $\{x^a, y^b\}$ in $\mathcal{A}(G')$, with $a, b \in \{\mathrm{h}, \mathrm{t}\}$, either $\{x^a, y^b\}$ is contained in $A(G)$ or there exists a sequence of adjacencies $\{\{x^a, z_1^{a_1}\}, \{z_1^{b_1}, z_2^{a_2}\}, \ldots, \{z_n^{b_n}, y^b\}\} \subseteq \mathcal{A}(G)$ such that $\{z_1, \ldots z_n\} \cap \mathcal{C}(G') = \emptyset$, whereby $\{a_i, b_i\} = \{\mathrm{h}, \mathrm{t}\}$ for each $i = 1, \ldots, n$.*

Chromosomes represent a special class of subgenomes that unambiguously decompose a genome G into a set $\mathcal{X}(G) = \{X_1, \ldots, X_n\}$, so that the set of genes of G and the set of adjacencies are the union of non-intersecting subsets $\mathcal{C}(G) = \mathcal{C}(X_1) \cup \cdots \cup \mathcal{C}(X_n)$ and $\mathcal{A}(G) = \mathcal{A}(X_1) \cup \cdots \cup \mathcal{A}(X_n)$, respectively. A *chromosome* X is a genome for which holds either

1. $\bigcup \mathcal{C}(X) = \bigcup \mathcal{A}(X)$ or

2. $\bigcup \mathcal{A}(X) \subset \bigcup \mathcal{C}(X)$ and $|\bigcup \mathcal{C}(X) \setminus \bigcup \mathcal{A}(X)| = 2$,

and there exists no smaller genome $X' \subset X$ satisfying any of the two conditions. In the former case, X is called *circular*, whereas in the latter, it is called *linear*. Chromosomes are connected components in the graph representation of the genomes, and form a path if they are linear, and a simple cycle if they are circular.

Example 2 (continued) *G has three chromosomes $X_1 = (\{a\}, \{\{a^{\mathrm{t}}, a^{\mathrm{h}}\}\})$, $X_2 = (\{c, e\}, \{\{c^{\mathrm{h}}, e^{\mathrm{h}}\}\})$, $X_3 = (\{b, d\}, \{\{b^{\mathrm{t}}, d^{\mathrm{h}}\}, \{b^{\mathrm{t}}, d^{\mathrm{h}}\}\})$, where X_2 is linear, and X_1, X_3 are circular.*

A chromosome can be read in two directions. W.l.o.g. each chromosome is assigned a canonical *reading direction*, resulting in a natural left to right reading order of its corresponding genes and adjacencies. That is, a gene g facing against the reading direction is denoted by an overline $\overline{g}$. In doing so, a genome can be represented by a collection of words, in which each character corresponds to a gene. Each circular chromosome is arbitrarily cut between any two genes and its circularity is indicated by surrounding brackets. Words that represent the same chromosome up to reversal fall into the same equivalence class.

Example 2 (continued) *Genome G is entirely represented by the words (a), $c\ \overline{e}$, $(b\ d)$. Further, word (a) is equivalent to $(\overline{a})$, word $c\ \overline{e}$ is equivalent to $e\ \overline{c}$, and $(b\ d)$ is equivalent to words $(d\ b)$, $(\overline{b}\ \overline{d})$, and $(\overline{d}\ \overline{b})$.*

2.4.2 Telomeres

In some genome models the extremities of linear genomes, called *telomeres*, are modeled explicitly. We will identify telomeres by symbol $\circ$ and a subscript number to differentiate between them. A telomere is modeled as a special gene, meaning that it is part of the universe of genes Σ, but has only one extremity, which we define as *head* $\circ^{\mathrm{h}}$. To this end,

$$\mathcal{T}(X) \equiv \begin{cases} \{\circ_1, \circ_2\} & \text{if } X \text{ is linear} \\ \emptyset & \text{otherwise} \end{cases}$$

denotes the set of telomeres of chromosome X. We further define $\mathcal{C}_\circ \equiv \mathcal{C}(X) \cup \mathcal{T}(X)$ as the set of genes and telomeres of chromosome X. Similarly, the adjacency set $\mathcal{A}_\circ$ of chromosome X contains, next to adjacencies $\mathcal{A}(X)$, also adjacencies between telomeres and the outermost gene extremities:

$$\mathcal{A}_\circ(X) \equiv \begin{cases} \mathcal{A}(X) \cup \{\{\circ_1, x\}, \{\circ_2, y\} \mid \{x, y\} = \bigcup \mathcal{C}(X) \setminus \bigcup \mathcal{A}(X)\} & \text{if } X \text{ is linear} \\ \mathcal{A}(X) & \text{otherwise,} \end{cases}$$

where extremities x and y are arbitrarily assigned to telomeres $\circ_1$ and $\circ_2$. Finally, we extend these functions to genomes, i.e., $\mathcal{T}(G) \equiv \bigcup_{X \in \mathcal{X}(G)} \mathcal{T}(X)$, $\mathcal{C}_\circ(G) \equiv \bigcup_{X \in \mathcal{X}(G)} \mathcal{C}_\circ(X)$, and $\mathcal{A}_\circ(G) \equiv \bigcup_{X \in \mathcal{X}(G)} \mathcal{A}_\circ(X)$.

2.5 The Family-free Principle

The family-free principle embodies the idea to perform gene order analysis without the use of gene family or homology assignments. Instead, we are given *gene similarities* on the basis of a *similarity measure* $\sigma : \Sigma \times \Sigma \to \mathbb{R}_{\geq 0}$ over the universe of genes Σ. Gene similarities establish relationships between genes that enable the execution of gene order studies without the prerequisite of predicted gene families. Unlike homology assignments, gene similarities are not a biological concept, but a general framework that includes many other measures of gene relationships such as measures of sequence and functional similarity, and similarity between folding structures of protein coding genes. Independent of the particular similarity measure σ, we consider relations between genes imposed by σ as candidates for homology assignments. We require similarity measure σ to be *symmetric*, i.e., $\sigma(x, y) = \sigma(y, x)$ for any two genes $x, y \in \Sigma$ and that each gene is most similar to itself, i.e., $\sigma(x, x) = \sup_{y \in \Sigma} \sigma(x, y)$. Note that gene similarities can represent a homology assignment $\mathcal{H}$:

$$\sigma_{\mathcal{H}}(x, y) = \begin{cases} 1 & \text{if } (x, y) \in \mathcal{H} \\ 0 & \text{otherwise}. \end{cases} \tag{2.3}$$

In genome models with telomeres, any two telomeres, $\circ_i, \circ_j \in \Sigma$ have fixed similarity $\sigma(\circ_i, \circ_j) = s_\circ$ (although individual weights may be set in case of contig ends in unassembled genomes) and have zero similarity to any genes.

In subsequent genome comparisons, we will make use of the following graph representation, which features similarity relationships between genes of different genomes and ignores similarities between genes within the same genome:

Definition 5 (gene similarity graph [26, 38]) *Given k genomes $G_1, \ldots, G_k$ and similarity measure σ, the* gene similarity graph *is a weighted, undirected, k-partite graph $B = (V_1, V_2, \ldots, V_k, E)$, where each vertex set V_i, $1 \leq i \leq k$, represents genes of the i^{th} genome, i.e., $V_i = \mathcal{C}(G_i)$, and edge set $E = \{\{g, h\} \mid g \in \mathcal{C}(G_i),\ h \in \mathcal{C}(G_j), 1 \leq i < j \leq k : \sigma(g, h) > 0\}$ denotes the set of edges between genes belonging to two distinct genomes. Edges are weighted, with each edge $\{g, h\} \in E$ having edge weight $w(\{g, h\}) \equiv \sigma(g, h)$. The* gene similarity graph with telomeres $B_\circ$ *is analogously defined for vertex sets $V_i = \mathcal{C}_\circ(G_i)$, with $1 \leq i \leq k$. Telomeres between distinct genomes are always connected in $B_\circ$ with edges of fixed weight $s_\circ$.*

For convenience, we define $E(B)$ as set of edges of gene similarity graph $B = (V_1, \ldots, V_k, E)$, and $E(B_\circ)$ in case of gene similarity graph with telomeres $B_\circ$, respectively. Further, we call a gene *singleton* if its corresponding vertex in B is not incident to any edge in $E(B)$.

Example 3 *An example of a gene similarity graph with telomeres $B_\circ$ for genomes $G = g_1\ g_2\ \overline{g_3},\ (\overline{g_4}\ \overline{g_5}\ g_6\ g_7)$ and $H = (\overline{h_1}\ \overline{h_2}\ h_3\ \overline{h_4}),\ \overline{h_5}\ h_6,\ (h_7)$ is shown in Figure 2.6.*

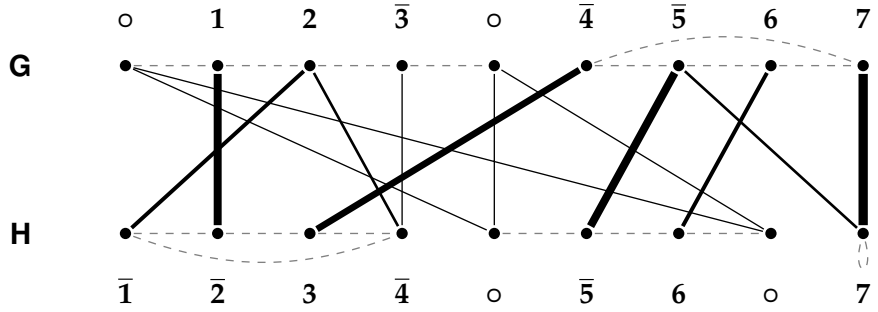


Figure 2.6: Gene similarity graph $B_\circ$ of genomes G and H. Each vertex represents a gene or a telomere. For simplicity, genes are labeled according to their indices. Telomeres are simply labeled by $\circ$, omitting their chromosome affiliations and numbering. Black edges correspond to similarities between genes of G and H. Their thickness indicates their edge weight. Dashed gray edges indicate adjacencies of genomes G and H, but are not explicitly modeled in the gene similarity graph.

Gene similarity measure σ is not explicitly specified, but similarity values are indicated by the thickness of the black edges in the graph.

Chapter 3

Family-free adjacencies

In this chapter, we propose a gene family-free approach to calculate the number of conserved adjacencies. The subsequently presented results have in part been published in [38], [26], and [65]. First, we briefly review the *breakpoint distance*, which is the dual measure of the number of conserved adjacencies in gene family-based analysis when genomes have equal gene content. Thereafter, we introduce the problem of computing gene family-free adjacencies, followed by an analysis of its computational complexity. In doing so, we relate the posed problem to previous works of Bryant [27], Blin *et al.* [20], and Angibaud *et al.* [6]. We then proceed to outline an *Integer Linear Program* (ILP) that solves our problem exactly. We discuss preprocessing algorithms that can reduce the candidate space of optimal solutions in practice. Further, we present a heuristic method that is based on previous work of Angibaud *et al.* [6]. Lastly, we evaluate our model and algorithms on simulated and biological datasets. To this end, we compare our results to those of Angibaud *et al.* [6] which were obtained through a comparable gene family-based approach on a dataset comprising twelve γ-proteobacterial genomes.

3.1 Breakpoint distance

The *breakpoint distance* is an early and still popular measure of gene order *dissimilarity*, which was initially proposed by Watterson *et al.* [112]. It is used to calculate the evolutionary distance between two genomes on the basis of quantifying global modifications in their gene orders, without explicitly drawing on rearrangement operations. The breakpoint distance is defined on two genomes with equal gene content as the number of adjacencies that are not common to both. Recall that in this work we assume genes to be unique, i.e., they occur once in exactly one genome that is associated with a specific organism. Therefore we will phrase the breakpoint distance in terms of a one-to-one homology assignment.

Definition 6 (breakpoint distance) *Given two genomes G and H of equal size $|G| = |H|$ and a one-to-one homology assignment $\mathcal{H}_1$ such that for every gene g in $\mathcal{C}(G)$ there exists exactly one assignment $\{g, h\} \in \mathcal{H}_1$ with $h \in \mathcal{C}(H)$ and* vice versa. *The projection of adjacencies of genome G onto genome H is*

$$\mathcal{A}^H(G) \equiv \left\{ \{h_1^a, h_2^b\} \mid \{\{g_1, h_1\}, \{g_2, h_2\}\} \subseteq \mathcal{H}_1 : \{g_1^a, g_2^b\} \in \mathcal{A}(G) \right\},$$

and $\mathcal{A}_\circ^G(H)$ defined analogously. The breakpoint distance *between G and H is*

$$d_{BP}(G, H) = |H| - |\mathcal{A}^H(G) \cap \mathcal{A}(H)| - \frac{|(A_\circ^H(G) \setminus A^H(G)) \cap (A_\circ(H) \setminus A(H))|}{2}.$$

Note that the size $|H|$ of genome H is defined as the number of its genes, excluding telomeres. Further, $\mathcal{A}^H(G)$ and $\mathcal{A}_\circ^H(G)$ are symmetric, therefore it holds that $|\mathcal{A}^H(G)| = |\mathcal{A}^G(H)|$ and $|\mathcal{A}_\circ^H(G)| = |\mathcal{A}_\circ^G(H)|$ for any two genomes G and H with equal gene content. We say that two genomes G and H are *equivalent*, iff $d_{BP}(G, H) = 0$ under one-to-one homology assignment $\mathcal{H}_1$. Thus, breakpoint distance as defined above is a *metric* [20, 112].

3.2 Pairwise family-free adjacencies

In the following, we present a model for identifying shared adjacencies between two genomes in a gene family-free setting. This model is subsequently extended for comparisons of two or more genomes in the next section. For now, we focus on two genomes G and H under a given similarity measure σ. In this study, telomeres in G and H are modeled explicitly. We then relate between adjacencies of two different genomes as follows:

Definition 7 (conserved adjacency) *Given two genomes G and H and gene similarity measure σ, two adjacencies, $\{g_1^a, g_2^b\} \in \mathcal{A}_\circ(G)$ and $\{h_1^a, h_2^b\} \in \mathcal{A}_\circ(H)$ with $a, b \in \{\mathrm{h}, \mathrm{t}\}$ are* conserved *iff $\sigma(g_1, h_1) > 0$ and $\sigma(g_2, h_2) > 0$.*

We subsequently define the *adjacency score* of any four extremities g^a, h^b, i^c, j^d, where $a, b, c, d \in \{\mathrm{h}, \mathrm{t}\}$ and $g, h, i, j \in \Sigma$ as the geometric mean of their corresponding gene similarities:

$$s(g^a, h^b, i^c, j^d) \equiv \begin{cases} \frac{1}{2}\sqrt{\sigma(g, h) \cdot \sigma(i, j)} & \text{if any of } g, h, i, j \text{ is a telomere,} \\ \sqrt{\sigma(g, h) \cdot \sigma(i, j)} & \text{otherwise.} \end{cases} \tag{3.1}$$

Note that the adjacency score itself does not rely on adjacency constraints. Rather, these constraints are imposed by Definition 7. The convex nature of the geometric mean of two gene similarity scores rewards conserved adjacencies between highly similar genes the most, whereas combinations of highly and remotely similar genes, or two remotely similar genes are decreasingly scored.

We now aim to establish a one-to-one homology assignment between genes and telomeres of genomes G and H, which maximizes the sum of adjacency scores of conserved adjacencies. In doing so, we circumvent the identification of gene duplication events by matching those duplicates that either have high similarity to each other, or are contained in a conserved adjacency with high adjacency score. Our one-to-one homology assignment takes into account gene similarities (which are treated as indicator for homology), adjacency scores, and insertions and deletions of one or few genes. To this end, we compute a *matching* $\mathcal{M} \subseteq E$ in gene similarity graph $B_\circ = (U, V, E)$ of genomes G and H. A matching $\mathcal{M}$ in $B_\circ$ induces subgenomes $G_\mathcal{M} \subseteq G$ and $H_\mathcal{M} \subseteq H$, with gene sets $\mathcal{C}(G_\mathcal{M})$ and $\mathcal{C}(H_\mathcal{M})$ corresponding to the set of connected vertices of the matching subgraph $B_\circ^\mathcal{M} = (U, V, \mathcal{M})$. We quantify gene similarities and scores of conserved adjacencies between matching-induced subgenomes $G_\mathcal{M}$ and $H_\mathcal{M}$ by means of the following measures:

$$adj_{GH}(\mathcal{M}) = \sum_{\substack{\{\{g_1,h_1\},\{g_2,h_2\}\}\subseteq\mathcal{M},\\ \{g_1^a,g_2^b\}\in\mathcal{A}_\circ(G_\mathcal{M}),\\ \{h_1^a,h_2^b\}\in\mathcal{A}_\circ(H_\mathcal{M})}} s(g_1^a, g_2^b, h_1^a, h_2^b) \tag{3.2}$$

$$edg(\mathcal{M}) = \sum_{e\in\mathcal{M}} w(e) \tag{3.3}$$

Observe that gene pairs (g_1, h_1) and (g_2, h_2) in conserved adjacency measure adj_{GH} share the same terminals a and b, respectively, where $a, b \in \{\mathrm{h}, \mathrm{t}\}$. This leads to the following optimization problem, in which we aim to find a solution that maximizes a linear combination of both quantities adj_{GH} and edg:

Problem 1 (FF-Adjacencies) *Given two genomes G, H, and some $\alpha \in [0, 1]$, find a matching $\mathcal{M}$ in gene similarity graph $B_\circ$ of G and H such that the following formula is maximized:*

$$\mathcal{F}_\alpha(\mathcal{M}) = \alpha \cdot adj_{GH}(\mathcal{M}) \; + \; (1-\alpha) \cdot edg(\mathcal{M}). \tag{3.4}$$

Zhang and Leong [117] study this problem with a similar combination of sequence similarity and synteny score. Therein, similarities between genes are restricted to reciprocal best BLAST hits, and each gene is "*adjacent*" to three genes to its left and right. Further, problem FF-Adjacencies is related to the works of Tang and Moret [105] and Angibaud *et al.* [6], which study the *breakpoint distance* under gene family assignments with duplicated genes. Problem FF-Adjacencies can be easily conformed to gene family constraints by employing gene similarity measure $\sigma_\mathcal{H}$ of Equation (2.3) given some gene family assignment $\mathcal{H}$. However, Tang and Moret [105], and Angibaud *et al.* [6] impose an additional constraint, requiring that at least one representative of each gene family in genomes G and H must also be contained in $\mathcal{M}$-induced subgenomes $G_\mathcal{M}$ and $H_\mathcal{M}$. While such constraint is reasonable in gene family studies, where gene family assignments act as filter in reducing

false positive associations between genes, the gene similarity graph can include also small weakly connected components that most likely lead to false positive homology assignments, depending on the particular similarity function. Nevertheless, this constraint was maintained in our initial gene family-free approach [38]. That is because it facilitates the reduction of the solution space and allows to identify anchors in the gene similarity graph, similar to the work of Angibaud *et al.* [6]. In the work at hand, we present an original approach, described in Section 3.7, to limit the candidate space of optimal solutions when this constraint is not imposed. Note that for $\alpha = 0$, problem FF-Adjacencies, independent of the mentioned constraint, coincides with the maximum weighted bipartite matching problem.

3.3 Family-free adjacencies for more than two genomes

In the previous section we introduced problem FF-Adjacencies for pairwise comparisons. In this section, we advance toward a more general model applicable for the simultaneous study of several genomes. Conserved adjacencies obtained by this approach can further benefit ancestral genome reconstruction, as it will be explained in Section 4.5 of the next chapter. Given $k \geq 2$ genomes, we aim to find a one-to-one homology assignment between genes, as previously in the pairwise case. One way is to find all completely connected subgraphs of size k in the gene similarity graph and then perform a k-dimensional matching (also known as k-matching). Yet, this approach neglects many connected components that do not form complete cliques or only spread over a smaller subset of genomes. Consequently, with increasing number of genomes in the dataset, the matching size will decrease until only few fully connected genes remain. Thus, we use a *partial k-matching* instead, which allows for missing genes and edges:

Definition 8 (partial k-matching) *Given a gene similarity graph $B = (G_1, \ldots, G_k, E)$, a* partial k-matching $\mathcal{M} \subseteq E$ *is a subset of edges such that for each connected component C in $B_{\mathcal{M}} \equiv (G_1, \ldots, G_k, \mathcal{M})$ no two genes in C belong to the same genome.*

We subsequently extend problem FF-Adjacencies to the simultaneous comparison of several genomes:

Problem 2 (FF-Adjacencies for $k > 2$ genomes) *Given a gene similarity graph $B = (G_1, \ldots, G_k, E)$ and some $\alpha \in [0, 1]$, find a partial k-matching $\mathcal{M}$ that maximizes the following formula:*

$$\mathcal{F}_\alpha(\mathcal{M}) = \alpha \cdot \sum_{1 \leq x < y \leq k} adj_{G_x G_y}(\mathcal{M}) \; + \; (1 - \alpha) \cdot edg(\mathcal{M}). \tag{3.5}$$

3.4 Computational complexity of pairwise family-free adjacencies

In this section we study the computational complexity of problem FF-Adjacencies in pairwise comparisons. For $\alpha = 0$, problem FF-Adjacencies coincides with the maximum weighted bipartite matching problem, which can be solved in polynomial time [63]. Yet, this does not answer the question of the computational complexity of problem FF-Adjacencies for $\alpha > 0$. In the following, we show first equivalence between instances of the NP-hard *exemplar breakpoint distance* problem and certain instances of problem FF-Adjacencies. Then, in Section 3.4.2, we elucidate further on the relationship between maximum matchings and solutions to problem FF-Adjacencies. This sets our work in context of previous results of Blin *et al.* [20] and Angibaud *et al.* [6].

Before we can turn to the NP-hardness proof of problem FF-Adjacencies, we need to show the following lemma that is used in both subsequent subsections:

Lemma 1 *Given two genomes G and H, and gene similarity measure $\sigma_1 : \Sigma \times \Sigma \to \{0, 1\}$, every solution to problem FF-Adjacencies for $\alpha < \frac{1}{3}$ corresponds a maximal matching in gene similarity graph $B_\circ$ of G and H.*

Proof: Assume we are given an optimal solution $\mathcal{M} \subseteq E$ to problem FF-Adjacencies for genomes G and H with $\alpha < \frac{1}{3}$. Further, assume for contradiction there exist two vertices g and h, for which there is an edge $\{g, h\} \subseteq E$, and g, h are not contained in $\mathcal{M}$*-induced subgraph* $B_\circ^{\mathcal{M}} \equiv (U, V, \mathcal{M})$, where $U \subseteq \mathcal{C}_\circ(G)$ and $V \subseteq \mathcal{C}_\circ(H)$.

Since g, h are disconnected in $B_\circ^{\mathcal{M}}$, it is obvious that edge set $\mathcal{M}' = \mathcal{M} \cup \{\{g, h\}\}$ is a valid matching and therefore also a feasible solution to problem FF-Adjacencies. To prove its optimality, we differentiate between two cases: (i) The $\mathcal{M}'$-induced subgenomes $G_{\mathcal{M}'}$ and $H_{\mathcal{M}'}$ share the same conserved adjacencies as genomes $G_{\mathcal{M}}$ and $H_{\mathcal{M}}$ and (ii) $G_{\mathcal{M}'}$ and $H_{\mathcal{M}'}$ have less conserved adjacencies than $G_{\mathcal{M}}$ and $H_{\mathcal{M}}$.

Case (i). If $\mathcal{M}'$-induced subgenomes $G_{\mathcal{M}'}$ or $H_{\mathcal{M}'}$ share the same conserved adjacencies as $\mathcal{M}$, then

$$\mathcal{F}_\alpha(\mathcal{M}) < \mathcal{F}_\alpha(\mathcal{M}') = \mathcal{F}_\alpha(\mathcal{M}) + (1 - \alpha) \cdot w(\{g, h\}) = \mathcal{F}_\alpha(\mathcal{M}) + (1 - \alpha)\,,$$

which is true because we presume $\alpha < \frac{1}{3}$, thus leading to the first contradiction of our claim.

Case (ii). We now study the worst case when edge $\{g, h\}$ disrupts as many conserved adjacencies as possible between genomes $G_{\mathcal{M}}$ and $H_{\mathcal{M}}$, while not contributing to any new. The number of disrupted adjacencies cannot be larger than two. That is because the integration of gene g into $G_{\mathcal{M}'}$ leads to a change in adjacencies $\mathcal{A}_\circ(G_{\mathcal{M}'}) = \mathcal{A}_\circ(G_{\mathcal{M}}) \setminus \{g_1^a, g_2^b\} \cup \{\{g_1^a, g^{\mathrm{h}}\}, \{g^{\mathrm{t}}, g_2^b\}\}$, thereby affecting only adjacency $\{g_1^a, g_2^b\}$ of genome $G_{\mathcal{M}}$. The analogous case holds for gene h in $H_{\mathcal{M}'}$. We now assume that the two adjacencies of $G_{\mathcal{M}}$ and $H_{\mathcal{M}}$, that are removed in $G_{\mathcal{M}'}$ and $H_{\mathcal{M}'}$, are associated with two different conserved adjacencies, denoted

by $\psi = \{\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\}\}$ and $\omega = \{\{g_3^c, g_4^d\}, \{h_3^c, h_4^d\}\}$. In a worst case scenario, g_i and h_i, $i = 1, \ldots, 4$, are genes, thus giving rise to maximal adjacency scores $s(\psi) = s(\omega) = 1$. But then inequality

$$\begin{aligned} \mathcal{F}_\alpha(\mathcal{M}') &= \mathcal{F}_\alpha(\mathcal{M}) - \alpha \cdot (s(\psi) + s(\omega)) + (1-\alpha) \cdot w(\{g,h\}) \\ &= \mathcal{F}_\alpha(\mathcal{M}) - 2\alpha + (1-\alpha) = \mathcal{F}_\alpha(\mathcal{M}) + (1-3\alpha) \\ &> \mathcal{F}_\alpha(\mathcal{M}), \end{aligned}$$

also leads to a contradiction of the claim that genes g, h sharing edge $\{g,h\} \in E(B_\circ)$ remain disconnected in an optimal solution $\mathcal{M}$ to problem FF-Adjacencies for $\alpha < \frac{1}{3}$. □

3.4.1 Reduction from exemplar breakpoint distance problem

The exemplar breakpoint distance relates to the *exemplar matching* in a bipartite graph, which is a matching containing exactly one edge (the "*exemplar*") per connected component. The problem that one aims to solve in computing the exemplar breakpoint distance can be stated as follows: Given two genomes G and H and gene family assignment $\mathcal{H}$, compute the minimum breakpoint distance $d_{BP}(G', H')$ over all subgenomes $G' \subseteq G$ and $H' \subseteq H$ such that $\mathcal{C}(G')$ and $\mathcal{C}(H')$ contain exactly one representative of each gene family with members in both $\mathcal{C}(G)$ and $\mathcal{C}(H)$. The exemplar breakpoint distance is NP-hard even for simple instances, where each gene family occurs exactly once in one genome, and in the other genome at most twice:

Problem 3 ((1,2)-EBD [27]) *Given two genomes G and H and gene family assignment $\mathcal{H}$ such that for each gene or telomere $g \in \mathcal{C}_\circ(G)$ holds that $|[g]_\mathcal{H} \cap \mathcal{C}_\circ(G)| = 1$ and $|[g]_\mathcal{H} \cap \mathcal{C}_\circ(H)| \in \{1,2\}$, find a subgenome $H' \subseteq H$ with $|H'| = |G|$ such that the distance $d_{BP}(G, H')$ is minimal.*

Problem 3 is NP-hard [27]. We proceed proving the following theorem:

Theorem 1 *Problem FF-Adjacencies is NP-hard for $0 < \alpha < \frac{1}{3}$.*

Proof: First, we reduce problem $(1,2)$-EBD to instances of problem FF-Adjacencies ($\Leftarrow$) and then show that a solution to $(1,2)$-EBD also leads to a solution to problem FF-Adjacencies ($\Rightarrow$).

($\Leftarrow$) Given genomes G, H, gene similarity measure $\sigma_1 : \Sigma \times \Sigma \to \{0,1\}$, and $\alpha \in]0, \frac{1}{3}[$. Further, let any gene or telomere $g \in \mathcal{C}_\circ(G)$ be connected to at most two genes h_1 and h_2 such that $\sigma(g,h_1) = \sigma(g,h_2) = 1$ and there exists no gene $g' \neq g$ for which $\sigma(g',h_1) = 1$ or $\sigma(g',h_2) = 1$. Clearly, such an instance of problem FF-Adjacencies corresponds to valid input of problem $(1,2)$-EBD. Now, since $\alpha < \frac{1}{3}$, a solution $\mathcal{M}$ to problem FF-Adjacencies for genomes G and H corresponds

to a maximal matching according to Lemma 1. In this particular case, every maximal matching is also maximum because exactly one edge incident to each gene $g \in \mathcal{C}_\circ(G)$ must be part of $\mathcal{M}$ according to Lemma 1. Since gene similarity measure σ_1 gives equal weight to edges and thus equal (non-zero) score to conserved adjacencies, solution $\mathcal{M}$ to problem FF-Adjacencies maximizes the number of conserved adjacencies among all possible maximum matchings in gene similarity graph $B_\circ$ of genomes G and H. A solution $\mathcal{M}$, which maximizes the number of conserved adjacencies between genomes $G = G_\mathcal{M}$ and $H_\mathcal{M} \subseteq H$, minimizes the breakpoint distance $d_{BP}(G, H_\mathcal{M})$ between genomes G and $H_\mathcal{M}$. We conclude that a solution to problem FF-Adjacencies for the described genomes G and H and $\alpha \in]0, \frac{1}{3}[$ is a valid solution to problem $(1,2)$-EBD.

($\Rightarrow$) Let genomes G and H and gene family assignment $\mathcal{H}$ be a valid input of problem $(1,2)$-EBD. Then G and H are also a valid input for problem FF-Adjacencies with gene similarity measure $\sigma_\mathcal{H}$, and a solution $\mathcal{M}$ to problem FF-Adjacencies for genomes G and H and some $\alpha \in]0, \frac{1}{3}[$ is also a solution to problem $(1,2)$-EBD for G and H.

□

We further believe strongly that the following conjecture holds true:

Conjecture 1 *Problem FF-Adjacencies is NP-hard for any $\alpha > 0$.*[1]

3.4.2 Maximum matchings in solutions to problem FF-Adjacencies

According to Lemma 1, whenever a gene similarity measure puts equal weight on all edges, the solution to problem FF-Adjacencies for $\alpha < \frac{1}{3}$ is a maximal matching. Furthermore, we previously mentioned that for $\alpha = 0$, problem FF-Adjacencies coincides with the maximum matching problem (and is equivalent to the maximum weighted matching problem under arbitrary σ). It is therefore obvious to ask at which point, as α is increased from 0 to $\frac{1}{3}$, a solution of problem FF-Adjacencies is no longer maximum, when gene similarity measure $\sigma_1 : \Sigma \times \Sigma \to \{0, 1\}$ is employed. The answer to this question relates problem FF-Adjacencies to previous works of Blin *et al.* [20] and Angibaud *et al.* [6], who studied the problem of computing the breakpoint distance under the maximum matching model.

Assume we are given a maximum matching $\mathcal{M}_{MM}$ in gene similarity graph $B_\circ$ of two genomes G and H and gene similarity measure σ_1 such that $\mathcal{M}_{MM}$ is a solution for problem FF-Adjacencies for small enough $\alpha > 0$. Further, let $n > 0$ be the number of edges in $\mathcal{M}_{MM}$. If we increase α, at which point does a maximal matching $\mathcal{M}_\mathcal{A}$ with $n - 1$ edges give a higher score under objective function $\mathcal{F}_\alpha$ than maximum matching $\mathcal{M}_{MM}$, because the former includes more conserved adjacencies? In an extremal scenario, $\mathcal{M}_{MM}$ has no conserved adjacencies at all, and its score under the objective function $\mathcal{F}_\alpha$ is $(1 - \alpha)n$. On the other side, the maximal matching can

[1] The conjecture was proven by Kowada *et al.* [62]

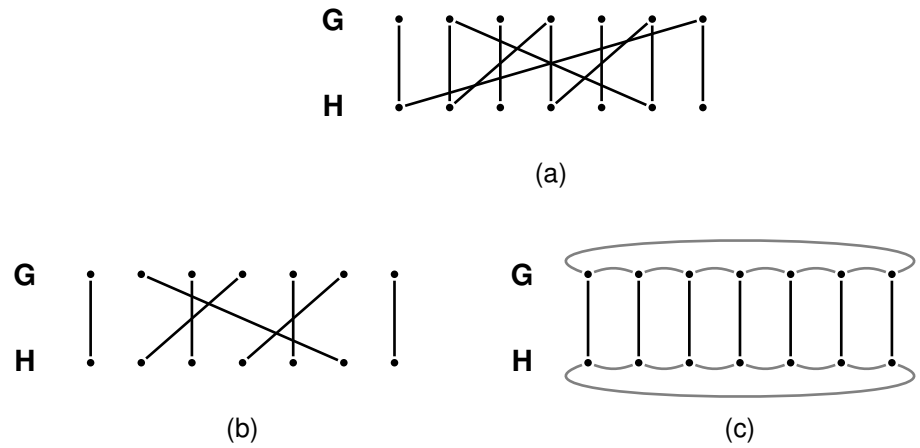


Figure 3.1: (a) Extremal scenario between (b) a maximum matching containing no conserved adjacencies and (c) a maximal matching containing as many conserved adjacencies (indicated by gray arcs) as possible.

include as many conserved adjacencies as possible, i.e., n conserved adjacencies assuming that genomes G and H contain only circular chromosomes. These two cases are schematically shown in Figure 3.1 (a-c). However, for circular chromosomes the score of $\mathcal{M}_{\mathcal{A}}$ under objective function $\mathcal{F}_\alpha$ is $\alpha(n-1)+(1-\alpha)(n-1)=(n-1)$. We solve the following equation for α:

$$\begin{aligned} & (n-1) < (1-\alpha)n \\ \iff \quad & n-1 < n-\alpha n \\ \iff \quad & \alpha < \frac{1}{n}\,. \end{aligned}$$

We conclude that every solution to problem FF-Adjacencies for any genomes G and H under gene similarity measure σ_1 is a maximum matching if $\alpha < \frac{1}{n}$ where $n = \min(|\mathcal{C}_\circ(G)|, |\mathcal{C}_\circ(H)|)$ and $\frac{1}{n} \ll \frac{1}{3}$ in practical applications. Further, when gene family constraints are imposed and $0 < \alpha < \frac{1}{n}$, problem FF-Adjacencies is equivalent to computing the breakpoint distance under the maximum matching model as described by Blin *et al.* [20] and Angibaud *et al.* [6].

3.5 Bounds

In the following we establish upper and lower bounds on the score $\mathcal{F}_\alpha(\mathcal{M})$ of a solution $\mathcal{M}$ to problem FF-Adjacencies in any gene similarity graph $B_\circ$ of two genomes G and H. A lower bound is simply constructed through any maximum weight matching $\mathcal{M}_{MWM}$ in $B_\circ$, which gives rise to lower bound $\mathcal{F}_\alpha(M_{MWM})$ for any matching $\mathcal{M}$ that is a solution to problem FF-Adjacencies in $B_\circ$.

We now study the upper bound of objective function $\mathcal{F}_\alpha$ which is a convex combination of the two measures *edg* and *adj*. Observe that the number of conserved adjacencies of any matching $\mathcal{M}$ in $B_\circ$ is at most its size $n = |\mathcal{M}|$. This case occurs if the gene order between G and H is perfectly conserved and both contain only circular chromosomes. In order to establish an upper bound of adjacency measure $adj_{GH}(\mathcal{M})$ for any solution $\mathcal{M}$, we assume that all n edges $e_i \in \mathcal{M}$, $1 \leq i \leq n$, are contained in two conserved adjacencies, respectively. W.l.o.g. genomes G and H comprise each a single circular chromosome. We follow the natural ordering of these edges, such that every pair of consecutive edges is contained in a conserved adjacency. Then the upper bound of adjacency measure $adj_{GH}(\mathcal{M})$ is given by

$$adj_{GH}(\mathcal{M}) \leq \sum_{i=1}^{\lceil \frac{1}{2}n \rceil} \sqrt{w(e_{2i-1}) \cdot w(e_{2i})} + \sum_{i=1}^{\lfloor \frac{1}{2}n \rfloor} \sqrt{w(e_{2i}) \cdot w(e_{2i+1})}\,, \tag{3.6}$$

where we define the $(n+1)^{\text{st}}$ edge to be equivalent to the first, i.e., $e_{n+1} \equiv e_1$, which allows us to sum over the weights of all *possible* conserved adjacencies in a matching $\mathcal{M}$. Applying the Arithmetic Mean-Geometric Mean Inequality we obtain

$$\sum_{i=1}^{n} w(e_i) \geq \sum_{i=1}^{\lceil \frac{1}{2}n \rceil} \sqrt{w(e_{2i-1}) \cdot w(e_{2i})} + \sum_{i=1}^{\lfloor \frac{1}{2}n \rfloor} \sqrt{w(e_{2i}) \cdot w(e_{2i+1})} \tag{3.7}$$

$$\Leftrightarrow \quad edg(\mathcal{M}) \geq adj_{GH}(\mathcal{M})\,, \tag{3.8}$$

which holds true for any matching $\mathcal{M}$ in any gene similarity graph $B_\circ$. Further, because the objective function $\mathcal{F}_\alpha$ is a convex combination of *edg* and *adj*, it holds that $\mathcal{F}_\alpha(\mathcal{M}) \leq edg(\mathcal{M})$ for any $\mathcal{M}$, independent of α. This leads to the following lemma:

Lemma 2 *Given two genomes G and H and gene similarity measure σ, any solution $\mathcal{M}_{MWM}$ to the maximum weighted matching problem in gene similarity graph $B_\circ$ of G and H, is a $\frac{1}{1-\alpha}$ approximation of problem FF-Adjacencies for $\alpha < 1$. Furthermore, for any matching $\mathcal{M}$ that is a solution to problem FF-Adjacencies,*

$$(1-\alpha) \cdot edg(\mathcal{M}_{MWM}) \leq \mathcal{F}_\alpha(\mathcal{M}) \leq edg(\mathcal{M}) \leq edg(\mathcal{M}_{MWM})\,.$$

3.6 An exact solution to problem FF-Adjacencies

In the following, we describe program `FFAdj-2G` that solves problem FF-Adjacencies exactly in pairwise comparisons. The presented results are based on previous work of Angibaud *et al.* [6]. The idea is to translate the problem FF-Adjacencies into a 0-1 linear program. That means we define a set of constraints, which are solely linear inequalities, whose variables are binary (i.e., they take on values 0 or 1) and an objective function (maximization or minimization of a linear formula). 0-1 linear

Algorithm 1 Program `FFAdj-2G` is an ILP for finding a solution to problem FF-Adjacencies (Problem 1) for two genomes G and H.

Objective:

`Maximize`

$$\alpha \cdot \sum_{\substack{\{g_1^a, g_2^b\} \in \mathcal{A}^\star(G), \\ \{h_1^a, h_2^b\} \in \mathcal{A}^\star(H)}} s(g_1^a, g_2^b, h_1^a, h_2^b) \cdot \mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) + (1-\alpha) \cdot \sum_{\substack{g \in \mathcal{C}_\circ(G), \\ h \in \mathcal{C}_\circ(H)}} \sigma(g,h) \cdot \mathbf{a}(g,h)$$

Constraints:

(`C.01`) for all $g \in \mathcal{C}_\circ(G)$, $\sum_{h \in \mathcal{C}_\circ(H)} \mathbf{a}(g,h) = \mathbf{b}(g)$

for all $h \in \mathcal{C}_\circ(H)$, $\sum_{g \in \mathcal{C}_\circ(G)} \mathbf{a}(g,h) = \mathbf{b}(h)$

(`C.02`) for all $\{g_1^a, g_2^b\} \in \mathcal{A}^\star(G)$ and for all $\{h_1^a, h_2^b\} \in \mathcal{A}^\star(H)$ with $a, b \in \{\text{h}, \text{t}\}$

$\mathbf{a}(g_1^a, h_1^b) + \mathbf{a}(g_2^a, h_2^b) - 2 \cdot \mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) \geq 0$,

if $\{g_1^a, g_2^b\} \notin \mathcal{A}_\circ(G)$, then for all g in $\{g_3, \ldots, g_n\} \subseteq \mathcal{C}(G)$

such that $\{\{g_1^a, g_3^{a_3}\}, \{g_3^{b_3}, g_4^{a_4}\}, \ldots, \{g_n^{b_n}, g_2^b\}\} \subseteq \mathcal{A}_\circ(G)$,

$\mathbf{b}(g) + \mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) \leq 1$,

if $\{h_1^a, h_2^b\} \notin \mathcal{A}_\circ(H)$, then for all $h \in \{h_3, \ldots, h_n\} \subseteq \mathcal{C}(H)$

such that $\{\{h_1^a, h_3^{a_3}\}, \{h_3^{b_3}, h_4^{a_4}\}, \ldots, \{h_n^{b_n}, h_2^b\}\} \subseteq \mathcal{A}_\circ(H)$,

$\mathbf{b}(h) + \mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) \leq 1$

Domains:

(`D.01`) for all $g \in \mathcal{C}_\circ(G)$ and for all $h \in \mathcal{C}_\circ(H)$, $\mathbf{a}(g,h) \in \{0,1\}$

(`D.02`) for all $g \in \mathcal{C}_\circ(G)$, $\mathbf{b}(g) \in \{0,1\}$,

for all $h \in \mathcal{C}_\circ(H)$, $\mathbf{b}(h) \in \{0,1\}$

(`D.03`) for all $\{g_1^a, g_2^b\} \in \mathcal{A}^\star(G)$ and for all $\{h_1^a, h_2^b\} \in \mathcal{A}^\star(H)$ with $a, b \in \{\text{h}, \text{t}\}$

$\mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) \in \{0,1\}$

programs are a special form of *Integer Linear Programs* (ILPs). We subsequently use a solver to assign a value for each variable such that the constraints are verified and the objective is optimized.

Program `FFAdj-2G` requires as input two genomes G and H, a number $\alpha \in [0,1]$, and a gene similarity measure σ. The objective, the variables, and the constraints are defined in Algorithm 1. Therein, we explore the set of conserved adjacencies of

all possible subgenomes of G and H, by means of the sets of *candidate adjacencies* of G and H.

Definition 9 (candidate adjacencies set) *For a given genome G,*

$$\mathcal{A}^{\star}(G) \equiv \bigcup_{G' \subseteq G} \mathcal{A}_{\circ}(G')$$

denotes the set of candidate adjacencies *over all subgenomes $G' \subseteq G$.*

We will now discuss program `FFAdj-2G`, as presented in Algorithm 1, in detail, thereby making use of gene similarity graph $B_{\circ}$ of genomes G and H, which is implicitly used in our program in order to obtain a matching $\mathcal{M}$ between genes of genomes G and H.

`FFAdj-2G` contains three types of 0-1 variables, which are listed in domain specifications (`D.01`), (`D.02`), and (`D.03`) in Algorithm 1. Variables $\mathbf{a}(g,h)$ indicate if edge $\{g,h\}$ in gene similarity graph $B_{\circ}$ is part of the anticipated matching $\mathcal{M}$. That is, iff variable $\mathbf{a}(g,h)$ is assigned value 1, then $\{g,h\} \in \mathcal{M}$. Similarly $\mathbf{b}(g)$, $g \in \mathcal{C}_{\circ}(G)$, and $\mathbf{b}(h)$, $h \in \mathcal{C}_{\circ}(H)$, encode if the corresponding vertices of genes or telomeres g and h in gene similarity graph $B_{\circ}$ are incident to an edge in $\mathcal{M}$. Lastly, variables $\mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b)$ indicate if gene extremities g_1^a, g_2^b of subgenome $G_{\mathcal{M}}$ and h_1^a, h_2^b of subgenome $H_{\mathcal{M}}$, with $a,b \in \{\mathrm{h}, \mathrm{t}\}$, induced by matching $\mathcal{M}$ can possibly form conserved adjacencies, i.e., $\{g_1^a, g_2^b\} \in \mathcal{A}_{\circ}(G_{\mathcal{M}})$ and $\{h_1^a, h_2^b\} \in \mathcal{A}_{\circ}(H_{\mathcal{M}})$.

Program `FFAdj-2G` contains two types of constraints, where constraint (`C.01`) represents matching constraints, and constraint (`C.02`) defines the rules of forming valid conserved adjacencies. For the former, it is straightforward to see that if the sum of edges that are part of the matching and that are incident to a single vertex equals one, then the resulting outcome will be a valid matching $\mathcal{M}$. To this end, we set the sum of variables $\mathbf{a}(g,h)$ for a given gene g of genome G over all genes h of genome H to be equal to $\mathbf{b}(g)$, and do so analogously for each gene h and its corresponding variable $\mathbf{b}(h)$. Since $\mathbf{a}$ and $\mathbf{b}$ are 0-1 variables, constraint (`C.01`) fulfills two purposes: Next to ensuring a valid matching, it acts as switch for variables $\mathbf{b}$, that indicate if a gene g is incident to an edge in solution $\mathcal{M}$. These variables are used in the second constraint, (`C.02`), to ensure that there are no genes $g \in \mathcal{C}(G_{\mathcal{M}})$ and $h \in \mathcal{C}(G_{\mathcal{M}})$ that are located within a conserved adjacency pair $(\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$. In the trivial case, no gene is located within adjacency $\{g_1^a, g_2^b\}$ in genome G in the first place, i.e., $\{g_1^a, g_2^b\} \in \mathcal{A}_{\circ}(G)$, and nothing needs to be done. Yet, whenever, subgenome $G_{\mathcal{M}}$ contains an adjacency $\{g_1^a, g_2^b\}$, where genes g_1 and g_2 are further apart in G, i.e., there exists a sequence of adjacencies $\{\{g_1^a, g_3^{a_3}\}, \{g_3^{b_3}, g_3^{a_4}\}, \ldots, \{g_n^{b_n}, g_2^b\}\} \subseteq \mathcal{A}_{\circ}(G)$, where $\{a_i, b_i\} = \{\mathrm{h}, \mathrm{t}\}$ with $3 \leq i \leq n$, containing genes $\{g_3, \ldots, g_n\} \in \mathcal{C}(G)$, then each of these genes $g_3, \ldots, g_n$ must not be contained in $\mathcal{M}$-induced subgenome $G_{\mathcal{M}}$. Constraint (`C.02`) ensures this by (i) either allowing conserved adjacency pair $(\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$, or (ii) any genes of

$\{g_3, \dots g_n\}$ to be part of $G_{\mathcal{M}}$, or (iii) neither of them to be true. This is achieved by constraint $\mathbf{b}(g) + \mathbf{c}(g_1^a, g_2^b, h_1^a, h_2^b) \leq 1$ for each gene $g \in \{g_3, \dots, g_n\}$, and analogously for adjacency $\{h_1^a, h_2^b\}$ of subgenome $H_{\mathcal{M}}$.

3.7 Speeding up computations

We now aim to reduce the number of variables and constraints in program `FFAdj-2G` that are irrelevant for obtaining an exact solution to problem FF-Adjacencies for two given genomes G and H, gene similarity measure σ, and a fixed $\alpha \in [0, 1]$. For instance, it is obvious that for any pair of genes or telomeres (g, h), $g \in \mathcal{C}_\circ(G)$ and $h \in \mathcal{C}_\circ(H)$, for which $\sigma(g, h) = 0$, variable $\mathbf{a}(g, h)$ and any related constraint can be safely discarded, since the value of $\mathbf{a}(g, h)$ has no effect on the solution.

We call a pair of candidate adjacencies $(\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$, where $\{g_1^a, g_2^b\} \in \mathcal{A}_\circ^\star(G)$, $\{h_1^a, h_2^b\} \in \mathcal{A}_\circ^\star(H)$, a *conserved candidate adjacency*, if $\sigma(g_1, h_1) > 0$ and $\sigma(g_2, h_2) > 0$. Whenever $s(g_1^a, g_2^b, h_1^a, h_2^b) = 0$ for any conserved candidate adjacency $c = (\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$, then variable $\mathbf{c}(h_1^a, h_2^b, g_1^a, g_2^b)$ and its related constraints can be omitted in the ILP.

With only these trivial omissions, the number of suboptimal solutions that program `FFAdj-2G` must evaluate remains still high. Hence the resulting ILP is excessively large and contains many combinations that must be evaluated by the solver, making the program too slow for practical applications.

The combinatorial complexity in solving problem FF-Adjacencies is dominated by the number of conserved candidate adjacencies in a matching. For example, problem FF-Adjacencies allows solutions in which the first and last genes of two linear chromosomes can participate in a conserved adjacency, whereas all their other genes are not contained in the matching. In most cases, such a scenario is suboptimal, yet this — as well as many other scenarios leading to suboptimal solutions — must be considered by program `FFAdj-2G`. In the remainder of this section, we present a naïve preprocessing algorithm to reduce the number of candidate conserved adjacencies that must be considered in identifying a solution to problem FF-Adjacencies. To this end, we give two solutions to the following loosely-stated problem:

Problem 4 *For any conserved candidate adjacency $c = (\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$, determine (i) whether every solution to problem FF-Adjacencies must contain c (ii) or every matching $\mathcal{M}$ in which c is a conserved adjacency pair between $G_{\mathcal{M}}$ and $H_{\mathcal{M}}$, leads to a suboptimal solution to problem FF-Adjacencies.*

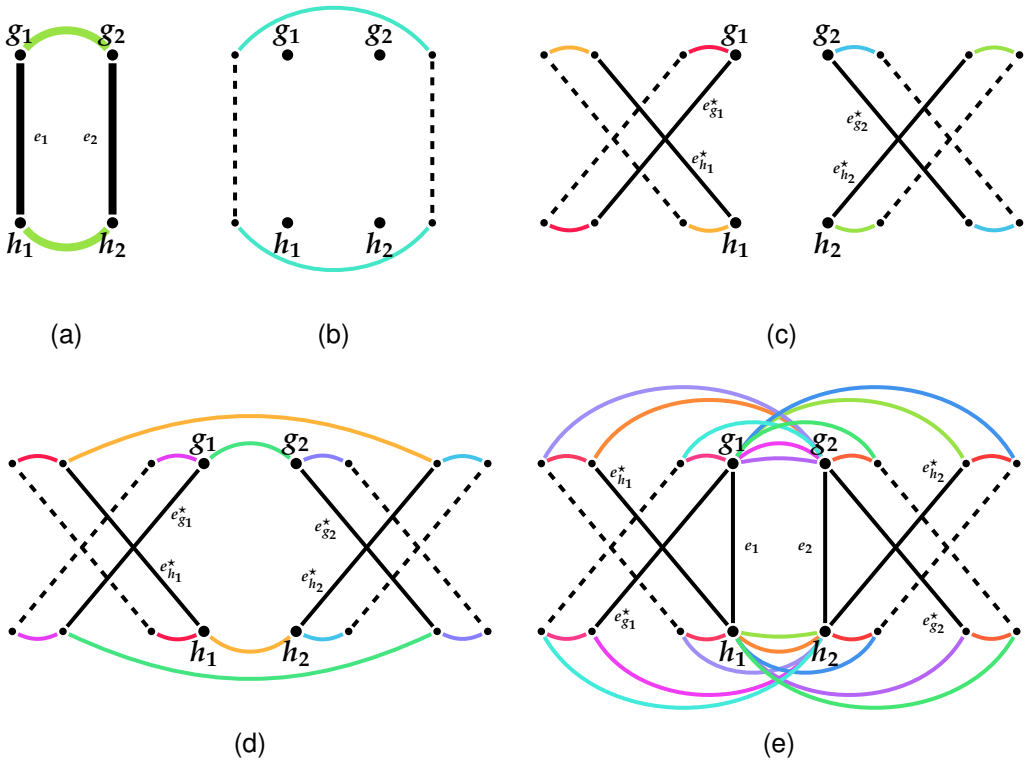


Figure 3.2: Parts (a)–(e) show all possible conserved candidate adjacencies, in which genes g_1, g_2, h_1 and h_2 can be involved. Black edges identify edges that are in the gene similarity graph $B_\circ$, pairs of arcs of same color correspond to conserved candidate adjacencies of their incident genes. Dashed black edges denote edges that are assumed to be present in $B_\circ$, but remain unobserved in the described analytic framework.

3.7.1 Identifying anchors in the gene similarity graph

In this section, we will address the part (i) of Problem 4. That is, we aim to identify pairs of candidate adjacencies of two genomes G and H and gene similarity measure σ, which must be part of an optimal solution to problem FF-Adjacencies.

The number of conserved candidate adjacencies that must be considered in finding an optimal solution $\mathcal{M}$ to problem FF-Adjacencies for genomes G and H can be greatly reduced if one knows that certain edges must be part of $\mathcal{M}$. Then any conserved candidate adjacency that crosses these edges can no longer be realized in optimal solution $\mathcal{M}$ and henceforth can be safely discarded prior to executing program `FFAdj-2G`. We call edges *anchors*, when they are proven to be contained in every optimal solution before any solution to problem FF-Adjacencies is calculated.

To identify anchors, we look at very simple local structures in the gene similarity graph $B_\circ$ of genomes G and H and calculate the net gain of including these structures into a matching. Let us consider a conserved candidate adjacency pair $c = (\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\})$, where $\{g_1^a, g_2^b\} \in \mathcal{A}_\circ(G)$, $\{h_1^a, h_2^b\} \in \mathcal{A}_\circ(H)$. That is, genes g_1, g_2 and h_1, h_2 are immediate adjacencies in genomes G and H, as visualized in the graph of Figure 3.2 (a), where edges $\{g_1, h_1\}$, $\{g_2, h_2\}$ are black, and the colored arcs indicate the conserved candidate adjacency between the two pairs of extremities (not explicitly shown) of the four genes. We denote by $e_1 = \{g_1, h_1\}$, $e_2 = \{g_2, h_2\}$ the edges between genes g_1, h_1 and g_2, h_2 in gene similarity graph $B_\circ$ of G and H and by $w_1 = \sigma(g_1, h_1)$, $w_2 = \sigma(g_2, h_2)$ their edge weights, respectively. Similarly,

$$w^\star_{g_1} = \max_{\substack{h \in \mathcal{C}(H) \\ h \neq h_1}} \sigma(g_1, h)$$

is the highest weight of any edge $e^\star_{g_1} \neq e_1$ incident to vertex g_1 in $B_\circ$. Edges $e^\star_{g_2}, e^\star_{h_1}$, and $e^\star_{h_2}$ and their corresponding weights $w^\star_{g_2}, w^\star_{h_1}, w^\star_{h_2}$ are defined analogously. Since our goal is to find anchors, let e_1 and e_2 be the highest weighted edges incident to vertices g_1, g_2, h_1, and h_2.

We assume for contradiction that we are given a solution $\mathcal{M}$ to problem FF-Adjacencies of genomes G and H that does not contain conserved candidate adjacency c. What is the minimal gain of including c into a matching $\mathcal{M}' \subseteq \mathcal{M} \cup \{\{g_1, h_1\}, \{g_2, h_2\}\}$? If the answer to this question is strictly positive, then c is an anchor and we discard all conserved candidate adjacencies that are in conflict with c.

There are four structurally different worst case scenarios that must be considered to answer this question analytically. We derive a general formula for computing the minimal gain of constructing matching $\mathcal{M}'$ from any matching $\mathcal{M}$ such that $\mathcal{M}'$ contains conserved candidate adjacency c. To allow for a computationally fast detection of anchors, we want our formula to depend only on edges incident to genes g_1, g_2, h_1, and h_2. Thus, each edge beyond these four edges is considered to have highest similarity

$$w^\star = \max_{\substack{g \in \mathcal{C}(G) \\ h \in \mathcal{C}(H)}} \sigma(g, h)\,,$$

corresponding to the worst case of losing edges or adjacencies due to the inclusion of conserved candidate adjacency c. In the first scenario, no edge incident to g_1, g_2, h_1, and h_2 is contained in $\mathcal{M}$, as shown in Figure 3.2 (b). Instead, the four genes are spanned by a conserved adjacency. We can calculate for a given α if the net gain of

including conserved adjacency c is higher than losing a "*perfect conserved adjacency*" with adjacency score $w^\star$:

$$\alpha\sqrt{(w^\star)^2} < \alpha\sqrt{w_1 w_2} + (1-\alpha)(w_1 + w_2) \tag{3.9}$$

$$\iff \qquad \alpha < \frac{w_1 + w_2}{w_1 + w_2 + w^\star - \sqrt{w_1 w_2}} \tag{3.10}$$

Note that the denominators of Inequality (3.10) and of successive inequalities are strictly positive under the proposed assumptions of edge weights (and therefore, these inequalities hold true).

In the second worst case scenario, vertices g_1, g_2, h_1, and h_2 are incident to edges $e^\star_{g_1}$, $e^\star_{g_2}$, $e^\star_{h_1}$, and $e^\star_{h_2}$, respectively. All four edges can be involved in remote conserved adjacencies, as shown by Figure 3.2 (c). Since we do not inspect edges that are not incident to vertices g_1, g_2, h_1, and h_2, we assume that these remote conserved adjacencies are formed with edges of highest weight $w^\star$. Note that in contrast to conserved adjacency c, they must not necessarily be part of $\mathcal{A}_\circ(G)$ and $\mathcal{A}_\circ(H)$ but can be part of any set of conserved adjacencies between any pair of subgenomes $G' \subseteq G$ and $H' \subseteq H$ that is associated with a solution to problem FF-Adjacencies. As such, remote conserved adjacencies are almost impossible to predict, justifying our narrow approach of focusing only on edges incident to g_1, g_2, h_1, and h_2. The worst case corresponding to the described scenario is captured by inequalities

$$\alpha\left(\sqrt{w^\star_{g_1} w^\star} + \sqrt{w^\star_{g_2} w^\star} + \sqrt{w^\star_{h_1} w^\star} + \sqrt{w^\star_{h_2} w^\star}\right) + (1-\alpha)(w^\star_{g_1} + w^\star_{g_2} + w^\star_{h_1} + w^\star_{h_2}) < \alpha\sqrt{w_1 w_2} + (1-\alpha)(w_1 + w_2) \tag{3.11}$$

$$\iff \quad \alpha < \frac{w_1 + w_2 - w^\star_{g_1} - w^\star_{g_2} - w^\star_{h_1} - w^\star_{h_2}}{w_1 + w_2 - \sqrt{w_1 w_2} + \sqrt{w^\star_{g_1} w^\star} + \sqrt{w^\star_{g_2} w^\star} + \sqrt{w^\star_{h_1} w^\star} + \sqrt{w^\star_{h_2} w^\star} - w^\star_{g_1} - w^\star_{g_2} - w^\star_{h_1} - w^\star_{h_2}}\,. \tag{3.12}$$

Similarly, edge $e^\star_{g_1}$ could form a conserved adjacency with $e^\star_{g_2}$, which could result in an overall matching score superior to that of a matching which includes conserved adjacency c, if $e^\star_{g_1}$ and $e^\star_{g_2}$ participate in two further adjacencies. This, of course, holds analogously for edges $e^\star_{h_1}$ and $e^\star_{h_2}$, and both scenarios can simultaneously occur in a valid matching. Our considerations are summarized in inequalities

$$\begin{aligned}\alpha\left(\sqrt{w^\star_{g_1} w^\star_{g_2}} + \sqrt{w^\star_{g_1} w^\star} + \sqrt{w^\star_{g_2} w^\star} + \sqrt{w^\star_{h_1} w^\star_{h_2}} + \sqrt{w^\star_{h_1} w^\star} + \sqrt{w^\star_{h_2} w^\star}\right)& \\ + (1-\alpha)(w^\star_{g_1} + w^\star_{g_2} + w^\star_{h_1} + w^\star_{h_2}) < \alpha\sqrt{w_1 w_2} + (1-\alpha)(w_1 + w_2)& \end{aligned} \tag{3.13}$$

$$\iff \quad \alpha < \frac{w_1 + w_2 - w^\star_{g_1} - w^\star_{g_2} - w^\star_{h_1} - w^\star_{h_2}}{w_1 + w_2 - w^\star_{g_1} - w^\star_{g_2} - w^\star_{h_1} - w^\star_{h_2} - \sqrt{w_1 w_2} + \sqrt{w^\star_{g_1} w^\star_{g_2}} + \sqrt{w^\star_{g_1} w^\star} + \sqrt{w^\star_{g_2} w^\star} + \sqrt{w^\star_{h_1} w^\star_{h_2}} + \sqrt{w^\star_{h_1} w^\star} + \sqrt{w^\star_{h_2} w^\star}}\,. \tag{3.14}$$

Figure 3.2 (d) visualizes both described scenarios. At last, either edges e_1 or e_2 can participate in other conserved adjacencies. For instance, edge e_1 can form a conserved adjacency with $e^\star_{g_2}$, $e^\star_{h_2}$, or another edge beyond vertices g_2 and h_2 of weight $w^\star$. If e_1 forms a conserved adjacency with any of the former two, the overall

matching score can be higher than choosing conserved adjacency c, if $e^\star_{g_2}$ or $e^\star_{h_2}$ are involved in a further conserved adjacency of high weight. Note that conserved adjacency c and edges $e^\star_{g_2}$, $e^\star_{h_2}$ cannot both be part of the same matching.

$$\max \begin{pmatrix} \alpha\left(\sqrt{w_1 w^\star_{g_2}} + \sqrt{w^\star_{g_2} w^\star}\right) + (1-\alpha) w^\star_{g_2}\,, \\ \alpha\left(\sqrt{w_1 w^\star_{h_2}} + \sqrt{w^\star_{h_2} w^\star}\right) + (1-\alpha) w^\star_{h_2}\,, \\ \alpha\sqrt{w_1 w^\star} \end{pmatrix} < \alpha\sqrt{w_1 w_2} + (1-\alpha) w_2 \tag{3.15}$$

$$\iff \quad \alpha < \min \begin{pmatrix} \frac{w_2 - w^\star_{g_2}}{w_2 - w^\star_{g_2} + \sqrt{w_1 w^\star_{g_2}} + \sqrt{w^\star_{g_2} w^\star} - \sqrt{w_1 w_2}}, \\ \frac{w_2 - w^\star_{h_2}}{w_2 - w^\star_{h_2} + \sqrt{w_1 w^\star_{h_2}} + \sqrt{w^\star_{h_2} w^\star} - \sqrt{w_1 w_2}}, \\ \frac{w_2}{w_2 + \sqrt{w_1 w^\star} - \sqrt{w_1 w_2}} \end{pmatrix}. \tag{3.16}$$

These equations hold for edge e_2 analogously, although only one of the edge pairs $(e_1, e^\star_{g_2})$, $(e_1, e^\star_{h_2})$, $(e_2, e^\star_{g_1})$ and $(e_2, e^\star_{h_1})$ can be part of a valid matching. Figure 3.2 (e) visualizes the possibilities described by Inequalities (3.15) and (3.16) for both edges e_1 and e_2. We give a general simplified formula to compute the gain of including conserved adjacency c over all other possibilities as described by Inequalities (3.9), (3.11), (3.13), and (3.15), where $\overline{w} = w^\star_{g_1} = w^\star_{g_2} = w^\star_{h_1} = w^\star_{h_2}$:

$$\alpha\sqrt{w_1 w_2} + (1-\alpha)(w_1 + w_2) > \max \begin{cases} \alpha w^\star\,, \\ 4\alpha\sqrt{\overline{w} w^\star} + (1-\alpha) 4\overline{w}\,, \\ 2\alpha\left(\overline{w} + 2\sqrt{\overline{w} w^\star}\right) + (1-\alpha) 4\overline{w}\,, \\ \alpha\sqrt{w_1 \overline{w}} + (1-\alpha)(w_1 + \overline{w})\,, \\ \alpha\sqrt{w_2 \overline{w}} + (1-\alpha)(w_2 + \overline{w})\,, \\ \alpha\sqrt{w_1 w^\star} + (1-\alpha) w_1\,, \\ \alpha\sqrt{w_2 w^\star} + (1-\alpha) w_2 \end{cases} \tag{3.17}$$

The inequality above allows us to visualize the gain of establishing adjacency c on an analytic level, as pictured in Figure 3.3 (a) for three different parameter choices. Obviously, in practical applications the exact edge weights and the previously described inequalities are used. Similarly, Inequalities (3.10), (3.12), (3.14) and (3.16) can be integrated into the following:

$$\alpha < \min \begin{cases} \frac{w_1+w_2}{w_1+w_2+w^\star-\sqrt{w_1 w_2}}, \\ \frac{w_1+w_2-4\overline{w}}{4(\sqrt{\overline{w}w^\star}-\overline{w})+w_1+w_2-\sqrt{w_1 w_2}}, \\ \frac{w_1+w_2-4\overline{w}}{w_1+w_2+2w^\star-\sqrt{w_1 w_2}}, \\ \frac{w_2-\overline{w}}{w_2-\overline{w}+\sqrt{w_1\overline{w}}+\sqrt{\overline{w}w^\star}-\sqrt{w_1 w_2}}, \\ \frac{w_1-\overline{w}}{w_1-\overline{w}+\sqrt{w_2\overline{w}}+\sqrt{\overline{w}w^\star}-\sqrt{w_1 w_2}}, \\ \frac{w_2}{w_2+\sqrt{w_1 w^\star}-\sqrt{w_1 w_2}} \\ \frac{w_1}{w_1+\sqrt{w_1 w^\star}-\sqrt{w_1 w_2}} \end{cases} \tag{3.18}$$

Figure 3.3 (b) visualizes the above inequality for three parameter choices. Observe that the inequality fails if $\overline{w} \geq \frac{1}{2}w^\star$. Inequality (3.11) and (3.12) are the limiting factors in the gain of choosing conserved adjacency c and, thus, are the driving functions in the plots of Figure 3.3.

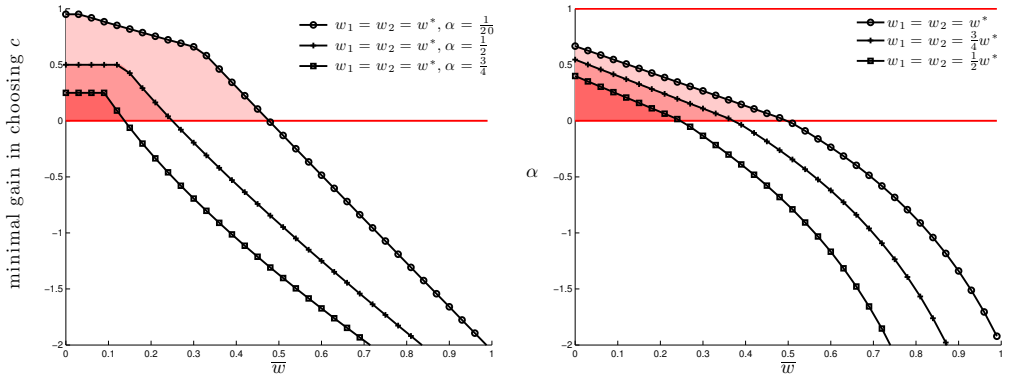
Figures 3.3 (a) exhibits reasonably large areas of positive minimal gain in all three parameter choices. This raises the hope that identifying anchors is an efficient method to reduce the solution space of problem FF-Adjacencies in practice. The plot shows that the minimal gain decreases with increasing α — what seems surprising at first sight, but is a result of the increasingly higher scored conserved adjacencies in alternative matchings that conflict with conserved adjacency c. The minimal gain is generally limited by $\overline{w} < \frac{1}{2}w_1 = \frac{1}{2}w_2$. That is, if edges e_1 and e_2 are adjacent (in the graph theoretic meaning of the word) to edges with half their weight or higher, the minimal conserved adjacency c is negative and c cannot be an anchor.

Figure 3.3 (b) indicates that few or no anchors could be found for $\alpha > 0.7$. Note that the analytic evaluation of anchors includes all possible worst case scenarios of alternative matchings that could give a higher score than conserved adjacency c. However, some of these scenarios can be ruled out when identifying anchors in practice. For example, if edges e_1 and e_2 are not adjacent to any other edges, then only the scenario described by Inequality (3.9) needs to be considered.

3.7.2 Remaining subgraph test

In the following, we give a solution to part (ii) of Problem 4, that is, we identify conserved candidate adjacencies that cannot be part of an optimal solution to problem FF-Adjacencies.

In solving problem FF-Adjacencies, a conserved adjacency that is established between two pairs of distantly located genes in two genomes removes many edges from the gene similarity graph that can no longer be part of a matching. We now proceed to define a test that evaluates if the remaining subgraph, i.e., the gene similarity graph without edges removed by establishing a given conserved adjacency,



(a) (b)

Figure 3.3: Part (a) shows the gain of establishing conserved adjacency c against all other possibilities as described by Inequality (3.17), for $w_1 = w_2 = w^\star$ and three different values of α, as a function of $\overline{w}$. The areas, corresponding to parameter choices for which Inequality (3.17) holds true, are highlighted in red colors and bounded by a red horizontal line. Similarly, Part (b) visualizes Inequality (3.18) for three different values of w_1 and w_2.

can (still) lead to an optimal solution of problem FF-Adjacencies. If the test fails, the reverse conclusion leads to the omission of the tested conserved adjacency from the set of conserved candidate adjacencies.

More formally, given a gene similarity graph $B_\circ$ of two genomes G and H, let $c = \{\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\}\}$ be the tested conserved candidate adjacency, where $\{g_1^a, g_2^b\} \in \mathcal{A}^\star(G)$ and $\{h_1^a, h_2^b\} \in \mathcal{A}^\star(H)$ and $a, b \in \{\mathrm{h}, \mathrm{t}\}$. In any matching in which $c = \{\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\}\}$ is a conserved adjacency pair, the (possibly empty) set of genes $\{g_3, \ldots, g_n\}$ such that $\{\{g_1^a, g_3^{a_3}\}, \{g_3^{b_3}, g_4^{a_4}\}, \ldots, \{g_n^{b_n}, g_2^b\}\} \subseteq \mathcal{A}_\circ(G)$, where $\{a_i, b_i\} = \{\mathrm{h}, \mathrm{t}\}$ with $3 \leq i \leq n$, and the (possibly empty) set of genes $\{h_3, \ldots, h_m\}$ such that $\{\{h_1^a, h_3^{c_3}\}, \{h_3^{d_3}, h_4^{c_4}\}, \ldots, \{h_m^{d_m}, h_2^b\}\} \subseteq \mathcal{A}_\circ(H)$, where $\{c_j, d_j\} = \{\mathrm{h}, \mathrm{t}\}$ with $3 \leq j \leq m$, can no longer be part of a solution to problem FF-Adjacencies for genomes G and H. If any of the two gene sets are non-empty, they induce a partition in gene similarity graph $B_\circ$ of G and H, into the set of *deleted edges* $E_D = \{\{g', h'\} \mid \{g', h'\} \in E(B_\circ) : g' \in \{g_3, \ldots, g_n\}$ or $h' \in \{h_3, \ldots, h_m\}\}$ and the *remaining edges* $E_R = E(B_\circ) \setminus E_D$.

We now apply upper and lower bounds of objective function $\mathcal{F}_\alpha$ of problem FF-Adjacencies that were established in Section 3.5 on the remaining subgraph E_R. This leads to the formulation of the *remaining subgraph test* captured by the following lemma:

Lemma 3 *Let $\mathcal{M}_{MWM} \subseteq E(B_\circ)$ and $\mathcal{M}^R_{MWM} \subseteq E_R$ be maximum weighted matchings in bipartite graphs $B_\circ$ and $(U, V, E_R) \subset B_\circ$, respectively, where $U \subset \mathcal{C}_\circ(G)$ and $V \subset \mathcal{C}_\circ(H)$. Further, let $\mathcal{M}_{heur} \subseteq E(B_\circ)$ be a heuristic solution, which is any matching in $B_\circ$ such*

that $\mathcal{F}_\alpha(\mathcal{M}_{heur}) \geq \mathcal{F}_\alpha(\mathcal{M}_{MWM})$. *Then conserved candidate adjacency pair* $c = \{\{g_1^a, g_2^b\}, \{h_1^a, h_2^b\}\}$ *can be discarded if*

$$\mathcal{F}_\alpha(\mathcal{M}_{heur}) > edg(\mathcal{M}_{MWM}^R) \tag{3.19}$$

without losing optimality in solving problem FF-Adjacencies for gene similarity graph $B_\circ$.

Proof: Recall that according to Lemma 2, the score $F_\alpha(\mathcal{M})$ of any solution $\mathcal{M} \subseteq E$ to problem FF-Adjacencies cannot exceed $edg(\mathcal{M}_{MWM})$, i.e., the sum of edge weights of a maximum weighted matching solution $\mathcal{M}_{MWM} \subseteq E$. Now, assume for contradiction that there exists an optimal solution $\mathcal{M}'$ to problem FF-Adjacencies for G and H, where conserved candidate adjacency pair c is established between $G_{\mathcal{M}'}$ and $H_{\mathcal{M}'}$. Since any edge in E_D cannot be part of $\mathcal{M}'$, it holds that

$$\mathcal{F}_\alpha(\mathcal{M}') \leq edg(\mathcal{M}_{MWM}^R) < \mathcal{F}_\alpha(\mathcal{M}_{\text{heur}}),$$

contradicting the premise of Equation (3.19).

□

Computing a maximum weight matching solution in gene similarity graph $B_\circ$ of genomes G and H is achievable within $\mathcal{O}(|E(B_\circ)|\sqrt{|G| + |H|})$ time [39], however an implementation of Duan and Su's algorithm was not publicly available at the time of writing. The number of possible conserved candidate adjacencies of $B_\circ$ is in order $\mathcal{O}(2^{|E(B_\circ)|})$, rendering the *remaining subgraph test* inefficient when applied in an exhaustive procedure. Nevertheless, if edges of conserved candidate adjacency c have maximal weight and the inclusion of c leads to a suboptimal solution to problem FF-Adjacencies, then any conserved candidate adjacency spanning c will also lead to a suboptimal solution. Thus, it is often not necessary to apply the remaining subgraph test for all conserved candidate adjacencies. We propose a preprocessing algorithm that executes the remaining subgraph test only for reasonably large deleted subgraphs, where the edges of the tested conserved candidate adjacency c are artificially set to maximal edge weight. If the test then concludes that the establishment of c leads to a suboptimal solution, then all spanning conserved candidate adjacencies can also be omitted from the solution space.

We will see in Section 3.9.2 that the utilized implementation of a maximum weight matching algorithm was too slow to apply the remaining subgraph test in practice.

3.8 A heuristic solution to problem FF-Adjacencies

With increasing genome size, it may become infeasible in practice to solve problem FF-Adjacencies exactly. Moreover, even when genome sizes are small, but the studied genomes accumulated many genome rearrangements so that no or only an insufficient amount of anchors can be found, the running time and size of program

Algorithm 2 Algorithm `FFAdj-MCS`

Input: Two genomes G, H, their corresponding gene similarity graph $B_\circ = (U, V, E)$, and $\alpha \in [0, 1]$
Output: Matching $\mathcal{M} \subseteq E$

1: Unseen $\leftarrow E$
2: Initialize empty list L
3: **while** Unseen $\neq \emptyset$ **do**
4: Find an MCS $S \subseteq$ Unseen with highest score $\mathcal{F}_\alpha(S)$
5: Unseen $\leftarrow$ Unseen $\setminus S$
6: Remove all edges incident to edges of S in Unseen.
7: Remove all singleton vertices of U and V
8: Elongate MCSs in L and possibly remove further edges from Unseen and further vertices from U and V, respectively
9: Append S to L
10: **end while**
11: $\mathcal{M} = \bigcup_{S \in L} S$

`FFAdj-2G` inflate to a point where exact solutions can no longer be obtained by today's computational means. In this section, we present heuristic `FFAdj-MCS` as described by Algorithm 2. `FFAdj-MCS` is an adaptation of the heuristic `IILCS` of Angibaud *et al.* [6].

`IILCS` allows to compute the number of adjacencies between two genomes when gene families are known, under an exemplar, intermediate, or maximum matching model. It is a greedy algorithm based on the idea of solving the *longest common substring* (LCS) problem: Given two strings S and T, find a longest string X that is a substring of both S and T. Solving a gene family-based problem, `IILCS` iteratively identifies LCSs in strings drawn from the alphabet of genes family identifiers representing chromosomal sequences and subsequently matches their corresponding genes until a satisfying matching is constructed.

This strategy can no longer be applied for problem FF-Adjacencies: Connected components in the gene similarity graph of two genomes G and H do not form gene families in general. Hence, their genes cannot be represented by a single gene family identifier in a string representation of genomes G and H, without losing information. However, in case edges are unweighted, chromosomes can be represented as *indeterminate strings*. Indeterminate strings, which will be further discussed in Chapter 5, are a class of strings that have one or more characters per position. Yet, we aim to address the general case with arbitrary edge weights, therefore we cannot make use of existing algorithms for the identification of LCSs in indeterminate strings such as those described in [56]. Given the gene similarity graph $B_\circ$ of two genomes G and H, our heuristic `FFAdj-MCS` matches in each iteration a maximal sequence of consecutive conserved candidate adjacencies that *locally* maximizes the objective function $\mathcal{F}_\alpha$, i.e., the convex combination of weights of conserved adjacencies and edges. We call this *maximal common substring* (MCS) of two chromosomes

an *MCS of highest score*. Note that we allow maximal common substrings of size 1, i.e. consisting only of single edges.

In each iteration, edges of the corresponding pairs of genes in identified MCSs of highest score are matched. That is, adjacent edges that can no longer be part of the matching are removed from gene similarity graph $B_\circ$. Furthermore, if in the process of removing edges a gene becomes a singleton, it will be also removed from the graph, facilitating the formation of a new immediate adjacency of its neighboring genes. We then aim to elongate previously processed MCSs by extending their extremities along newly created adjacencies. In the next iteration, the highest scoring MCS that has not been processed is identified and its edges are fixated in the matching. Algorithm `FFAdj-MCS` stops if all edges of the gene similarity graph $B_\circ$ are matched.

The runtime of the heuristic can be summarized as follows: Given gene similarity graph $B_\circ = (U, V, E)$ of two genomes, a maximal common substring of highest weight can be found in $\mathcal{O}(|E|)$ time. In each iteration, at least one maximal common substring is found. The update of the list of previously computed MCSs is achieved in time linear to their size. Since the number of possible matched edges is bounded by $n = \min(|U|, |V|)$, the algorithm terminates after $\mathcal{O}(n)$ iterations. The algorithm therefore requires overall $\mathcal{O}(|E| \cdot n)$ time and $\mathcal{O}(|E| + |U| + |V|)$ space.

3.9 Experimental results and discussion

We used the CPLEX[2] solver to run program `FFAdj-2G`. The 0-1 linear program itself is generated by scripts written in the Python programming language and we used the *NetworkX* library to run graph-based algorithms. Likewise, algorithm `FFAdj-MCS` is implemented in Python. In support of simplified code but at the expense of accuracy, our implemented algorithms do not allow a chromosome to be circular, even though this is permitted by our presented model. However, the maximal error made by this inaccuracy in comparing two circular chromosomes is at most two conserved adjacencies, which is negligible in subsequent analyses.

We evaluate our algorithms on simulated datasets obtained by ALF [33] and on a set of twelve γ-proteobacterial genomes from the dataset of Lerat *et al.* [66]. The latter is used in several other works and is to some degree considered as a standard reference dataset in comparative genomics [6, 21]. In both datasets, gene similarities were obtained with the *relative reciprocal BLAST score* [84]. To this end, genes were compared on the basis of their encoding protein sequence using BLASTP [3] with an e-value threshold of 10^{-5}, disabling the default query sequence filtering step.

Parameter name	Value	
sequence evolution		
substitution model	WAG (amino acid substitution model)	
insertion and deletion	Zipfian distribution	exponent $c = 1.8214$
	insertion rate	0.0003
	maximum insertion length	50
rate variation among sites	Γ-distribution	shape parameter $a = 1$
	number of classes	5
	rate of invariable sites	0.01
genome rearrangement		
inversion	rate	0.002
	maximum inversion length	300 (25)
transposition	rate	0.001
	maximum transposition length	300 (25)
	rate of inverted transposition	0.1
gene family evolution		
gene duplication	rate	0.005
	max. no. of genes involved in one dupl.	5
	probability of transposition after dupl.	0.5
	fission/fusion after duplication	0.1
	probability of rate change	0.2
	rate change factor	0.9
	probability of temporary rate change (duplicate)	0.5
	temporary rate change factor (duplicate)	1.5
	life of rate change (duplicate)	10 PAM
	probability of temporary rate change (orig+duplicate)	0.3
	temporary rate change factor (orig+duplicate)	1.2
	life of rate change (orig+duplicate)	10 PAM
gene loss	rate	0.005
	maximum length of gene loss	5
gene fission/fusion	rate	0.001
	maximum number of fused genes	2

Table 3.1: Parameter settings for simulations generated by ALF [33].

3.9.1 Simulated genome evolution

ALF [33] is a popular framework to simulate genome evolution. The simulator covers many aspects of genome evolution from point mutations to global modifications. The latter includes two types of genome rearrangements, as well as various options to customize the process of gene family evolution. In our simulations, we mainly use standard parameters suggested by the authors of ALF. In doing so, we focus on three parameters that primarily influence the outcome of gene family-free genome analysis: (i) the rate of sequence evolution, (ii) the rate of genome rearrangements, and (iii) the rate of gene duplications and losses. We keep all three rates constant, only varying the evolutionary distance between the generated extant genomes. Aiming to assess the performance of our algorithms under extremal conditions, we set the rate of rearrangements and gene duplications/losses considerably high, while

[2]http://www.ibm.com/software/integration/optimization/cplex-optimizer/

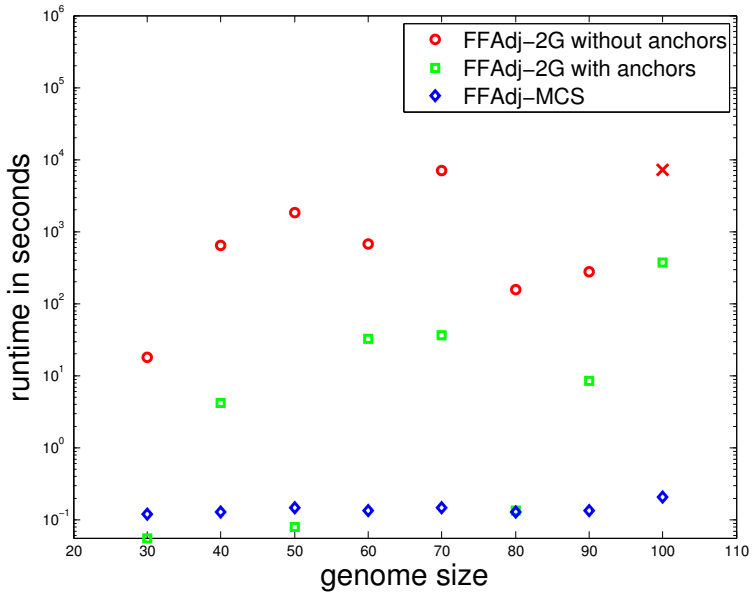


Figure 3.4: Semilog plot of runtimes of program `FFAdj-2G` with and without anchors and of program `FFAdj-MCS` as a function of genome size. The red "x" symbol on the upper right indicates that the runtime of program `FFAdj-2G` without anchors for genome size 100 exceeded two hours of computation, at which point the program was terminated.

keeping sequence evolution in a moderate range. Inversions tend to occur more frequently than transpositions in bacterial genomes. In fact, experimental results of Blanchette *et al.* [18] indicate that inversions may occur 2 to 2.5 times as frequent. Consequently, we set the ratio of inversions to transpositions to 2. We confine our simulations to protein coding genes. Therefore, we chose the *Whelan and Goldman* (WAG) model of amino acid evolution [113]. A comprehensive list of parameter settings used in our simulations is shown in Table 3.1.

In all our simulations we generate two genomes that evolve along separate evolutionary paths of variable lengths that emanate from a common ancestor. We use the genome sequence of an *Escherichia coli* K-12 strain (NCBI accession number `NC_000913`) as common ancestral genome, from which the two extant genomes diverge. The genome sequence of the *E. coli* strain comprises 4320 protein coding genes.

3.9.2 Runtime

We first assess the practical runtime of our algorithms. In particular, we are interested in the speed-up gained by *identifying anchors* and *testing remaining subgraphs* as described in Sections 3.7.1 and 3.7.2, respectively. The *NetworkX* library provides

an implementation of Zvi Galil's maximum weight matching algorithm [44], which we used in the remaining subgraph test. However, the time of a single maximum weight matching in our dataset took more than five minutes to compute. This renders the remaining subgraph test infeasible for our purposes, which require to compute several hundred maximum weight matchings for each instance of our generated integer linear programs. We leave to future work further evaluation of other implemented algorithms, in particular those that address the specific problem of maximum weight matchings in bipartite graphs.

The identification of anchors turned out to perform exceedingly well. Whereas running program `FFAdj-2G` on genomes beyond hundred genes became computationally infeasible, the reduction of the search space using anchors allows us to compute exact solutions in pairwise comparisons between genomes larger than 4000 genes in less than two hours of computation. We assessed the runtime performance of our programs as a function of genome size. To this end, we truncated the genome sequence of the *E. coli* strain to lengths ranging from 30 to 100 genes. We further set the maximum insertion and transposition length in genomes generated by ALF to 25 genes and generated genome pairs with an evolutionary distance of 125 *percent accepted mutations* (PAM). We executed all our programs on a single CPU, allocating 2GB of working memory. For the sake of benchmarking the running times of `FFAdj-2G` and `FFAdj-MCS`, we set α to 0.5, which is a considerably high value as indicated by the analytical results visualized in Figure 3.3 of Section 3.7.1. Figure 3.4 visualizes the outcome of our evaluation. The runtime of program `FFAdj-2G` shows large variations, but considerably decreases when anchors are supplied. The variations in runtime are an artifact of CPLEX's internal heuristics that can solve certain optimization problems very efficiently, whereas solving others of equal or lower complexity require considerably longer running times. The runtime of program `FFAdj-2G` without anchors for genome size 100 exceeded two hours of computation, at which point we terminated the program. Our heuristic `FFAdj-MCS` exhibits short running times as anticipated.

3.9.3 Quality of orthology assignments

We generated seven datasets of genome pairs with evolutionary distances ranging from 10 to 130 PAM. In doing so, we used the entire genome sequence of the aforementioned *E. coli* strain as root genome. Also, we set the maximum insertion and transposition length in these simulations to 300 genes. We further allowed CPLEX to occupy 8 CPU cores and allocated 8GB of working memory. Table 3.2 summarizes the main characteristics of the generated datasets. Note that the numbers of transpositions shown in the table also include transpositions of genes after gene duplication. In accord with our parameter settings, the number of ordinary transpositions (i.e., those not related to gene duplication events) is on average half the number of inversions. Also, we computed the *average node degree* of gene similar-

PAM	Inversions	Transpositions	Duplications	Losses	Avg. node degree
10	80	80	235	230	3.96
30	228	262	753	750	3.07
50	439	511	1383	1201	2.49
70	560	674	1685	1647	1.99
90	743	801	2307	2329	1.79
110	929	986	2626	2651	1.73
130	1114	1070	3055	3251	1.64

Table 3.2: Benchmark data comprising seven genome pairs generated by ALF [33].

ity graphs of the generated genome pairs by omitting singleton genes, which were removed prior to analysis.

Table 3.3 lists counts of identified anchors for program `FFAdj-2G`. The large number of identified anchors even in distant genome pairs which underwent many genome rearrangements and events of gene duplication and loss, allowed us to obtain exact results for almost all datasets. However, for $\alpha = 0.7$, we were unable to obtain exact results for datasets with evolutionary distance 10, 90 and 130 PAM. Instead, we instructed CPLEX to terminate the computation after two hours and to return a best feasible, possibly suboptimal, solution from its current solution pool. Moreover, our attempts failed to obtain exact results for higher values of α. The choice of α in the performance of heuristic `FFAdj-MCS` was only marginal, hence we chose $\alpha = 0.1$ by default.

Simulating genome evolution enables detailed knowledge of the occurring mutational changes in generated genomes. ALF provides a two-dimensional array that captures true orthology assignment between genes of generated genomes. We use this information to evaluate the quality of solutions returned by our algorithms. To this end, we counted *true positive* (TP), *false positive* (FP), *true negative* (TN) and *false negative* (FN) orthology assignments returned by our algorithms `FFAdj-2G` and `FFAdj-MCS`. Subsequently we computed their *precision* $(TP/(TP + FP))$, *re-*

PAM	FFADJ-2G: Number of conserved adjacencies / number of identified anchors										FFAdj-MCS
	$\alpha = 0$		$\alpha = 0.1$		$\alpha = 0.3$		$\alpha = 0.5$		$\alpha = 0.7$		$\alpha = 0.1$
10	3426	-	3588	3027	3606	2832	3620	2494	3622*	2181	3599
30	2324	-	2581	1952	2628	1820	2661	1608	2685	1410	2606
50	1416	-	1700	1148	1781	1048	1821	915	1847	777	1739
70	941	-	1170	759	1276	704	1322	629	1346	516	1254
90	606	-	790	455	886	405	929	347	770*	75	879
110	406	-	520	314	602	277	639	236	661	110	596
130	276	-	367	221	443	198	473	161	250*	19	436

Table 3.3: Number of conserved adjacencies and identified anchors in matchings obtained by programs `FFAdj-2G` and `FFAdj-MCS`. The numbers marked by * indicate that the result was obtained from a possibly suboptimal solution returned by a prematurely terminated instance of `FFAdj-2G`.

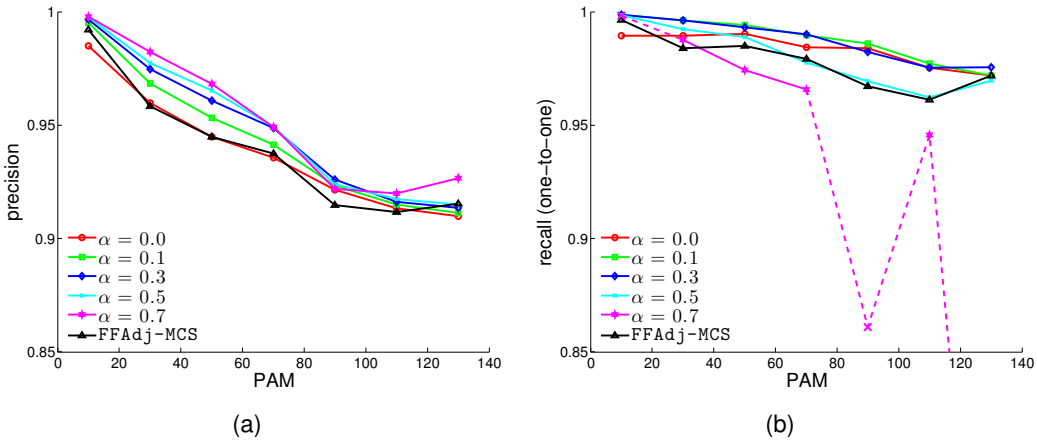


Figure 3.5: Precision and recall of orthology assignments obtained by `FFAdj-2G` and `FFAdj-MCS` for various choices of parameter α. The dashed lines with x marks indicate results obtained from a possibly suboptimal solution of program `FFAdj-2G` with $\alpha = 0.7$.

call $(TP/(TP+FN))$, and *accuracy* $((TP+TN)/(TP+TN+FP+FN))$. Yet, since our algorithms aim to establish one-to-one homology assignments, it is a fact that multiple true orthology assignments per gene lead to an undesired increase of *false negatives*. High counts of false negatives could lead to the wrong conclusion that the method is performing badly, where instead it might perform optimally as a method which returns only one-to-one orthology assignments. Thus, unlike typical benchmarkings of tools for prediction of orthology assignments, we count as *false negative* a gene that is not incident to any true positive orthology assignment. While this slight change does not alter the general trend inherent to our statistical measures, it does adjust their total quantities. In doing so, *one minus recall* and *one minus accuracy* indicate the remaining potential for improved methods that restrict themselves to the identification of one-to-one orthology assignments.

In our experiments, the number of true negatives was very high, which is typical of tools for orthology detection. As a result, the accuracy in all experiments was very close to 1. However, precision and recall of results obtained by our programs varied with the evolutionary distance and parameter choices for α, as visualized in Figure 3.5.

Recall, when $\alpha = 0$, problem FF-Adjacencies coincides with the maximum weighted matching problem. In other words, (when α is set to zero) `FFAdj-2G` obtains maximum weight matchings without using any synteny information. Still, in terms of precision and recall of orthology assignments, maximum weight matchings perform exceptionally well. However, the maximum weight matchings contained much fewer conserved adjacencies than matchings returned by `FFAdj-2G` for $\alpha > 0$, as shown in Table 3.3. The gain of conserved adjacencies in orthology assignments of

Species/strain name	Short name	Accession no.	Size (bp)	#Genes
Buchnera aphidicola APS	BAPHI	NC_002528	640,681	564
Escherichia coli K12	ECOLI	NC_000913	4,639,675	4320
Haemophilus influenzae Rd	HAEIN	NC_000907	1,830,138	1657
Pseudomonas aeruginosa PA01	PAERU	NC_002516	6,264,404	5571
Pasteurella multocida Pm70	PMULT	NC_002663	2,257,487	2012
Salmonella typhimurium LT2	SALTY	NC_003197	4,857,432	4423
Wigglesworthia glossinidia brevipalpis	WGLOS	NC_004344	697,724	611
Xanthomonas axonopodis pv. citri 306	XAXON	NC_003919	5,175,554	4312
Xanthomonas campestris	XCAMP	NC_003902	5,076,188	4179
Xylella fastidiosa 9a5c	XFAST	NC_002488	2,679,306	2766
Yersinia pestis CO_92	YPEST-CO92	NC_003143	4,653,728	3885
Yersinia pestis KIM5 P12	YPEST-KIM	NC_004088	4,600,755	4048

Table 3.4: The genomic dataset of our analysis comprises 12 γ-proteobacteria from Lerat *et al.* [66].

`FFAdj-2G` w.r.t. those identified by maximum weight matchings (i.e., `FFAdj-2G` with $\alpha = 0$) ranged from 4.7% for PAM 10 and $\alpha = 0.1$ to 71.4% for PAM 130 and $\alpha = 0.5$.

3.9.4 Experimental results on a biological dataset

We evaluated our programs on a set of 12 γ-proteobacterial genomes from the dataset of Lerat *et al.* that was also used by Angibaud *et al.* [6] who computed the breakpoint distance between genomes with duplicates, thereby employing various matching models. The genomic data including gene annotations have been obtained from NCBI under the accession numbers listed in Table 3.4. All genomes comprise a single, circular chromosome. The genomes were "linearized" in the order inherent to the NCBI data, and telomeres were added at the beginning and at the end of the resulting chromosomal sequences.

Table 3.5 lists in detail the main results of our analysis. In all 66 pairwise comparisons, exact results with program `FFAdj-2G` could be obtained for $\alpha = 0.1$. However, for $\alpha = 0.5$ the computation of 16 pairwise comparisons exceeded two hours of computation. Just as in previous experiments, we terminated the computation and reported a feasible, possibly suboptimal, solution identified by CPLEX.

In order to assess the gain of the family-free analysis, we compare the results of `FFAdj-2G` and `FFAdj-MCS` with adjacencies obtained by program `Adjacencies-Intermediate-Matching` of Angibaud *et al.* [6]. The linear program `Adjacencies-Intermediate-Matching` maximizes the number of adjacencies under the intermediate model between two genomes with gene family constraints. To compare the number of adjacencies (a common measure of these three programs) correctly, we must take into account two facts. First, the number of genes of the studied genomes differ. In [6], the authors used gene annotations and gene families that

		Angibaud *et al.* [6]		FFADJ-2G $\alpha = 0.1$		FFADJ-2G $\alpha = 0.5$		FFAdj-MCS $\alpha = 0.1$	
Genome 1	**Genome 2**	**Size**	**#Adj**	**Size**	**#Adj**	**Size**	**#Adj**	**Size**	**#Adj**
BAPHI	ECOLI	534	377	559	393	548	408	559	394
BAPHI	HAEIN	432	161	469	164	464	170	469	164
BAPHI	PAERU	470	229	529	245	521	260	529	249
BAPHI	PMULT	448	188	490	193	487	200	490	193
BAPHI	SALTY	535	376	559	393	548	407	559	393
BAPHI	WGLOS	374	203	421	221	417	225	419	221
BAPHI	XAXON	415	188	499	202	494	209	499	202
BAPHI	XCAMP	415	188	496	201	490	208	495	201
BAPHI	XFAST	399	162	467	172	463	178	467	172
BAPHI	YPEST-CO92	532	361	558	377	549	387	558	377
BAPHI	YPEST-KIM	525	348	553	372	543	382	553	372
ECOLI	HAEIN	1216	550	1370	549	1350	577	1370	554
ECOLI	PAERU	1734	651	2656	704	1670*	499*	2623	736
ECOLI	PMULT	1366	662	1641	685	1600	732	1631	685
ECOLI	SALTY	3152	2874	3413	2827	3275	2953	3411	2831
ECOLI	WGLOS	573	380	607	410	603	412	607	409
ECOLI	XAXON	1268	425	2121	500	1528*	385*	2084	528
ECOLI	XCAMP	1266	420	2102	497	2037*	478*	2073	518
ECOLI	XFAST	889	324	1280	368	1254	427	1267	388
ECOLI	YPEST-CO92	2341	1744	2666	1732	2568	1858	2667	1736
ECOLI	YPEST-KIM	2355	1747	2682	1747	2578	1869	2678	1751
HAEIN	PAERU	949	333	1250	365	1230	405	1246	376
HAEIN	PMULT	1375	849	1412	838	1397	856	1414	838
HAEIN	SALTY	1227	550	1387	555	1365	582	1385	557
HAEIN	WGLOS	443	165	488	174	488	180	488	179
HAEIN	XAXON	775	241	1092	260	1074	282	1087	265
HAEIN	XCAMP	769	238	1079	261	1058	280	1070	262
HAEIN	XFAST	674	205	892	229	884	240	888	230
HAEIN	YPEST-CO92	1172	522	1340	543	1314	574	1337	546
HAEIN	YPEST-KIM	1171	517	1336	536	1309	569	1334	540
PAERU	PMULT	1055	373	1483	417	1447*	467*	1472	435
PAERU	SALTY	1736	644	2700	713	1673*	502*	2664	740
PAERU	WGLOS	511	251	572	278	567	291	573	281
PAERU	XAXON	1626	609	2721	734	1745*	470*	2671	770
PAERU	XCAMP	1609	596	2684	704	1731*	459*	2644	731
PAERU	XFAST	978	405	1407	487	1373*	538*	1392	495
PAERU	YPEST-CO92	1662	671	2399	747	1633*	538*	2370	784
PAERU	YPEST-KIM	1667	662	2409	744	1625*	520*	2381	775
PMULT	SALTY	1375	670	1650	705	1597	753	1643	701
PMULT	WGLOS	468	197	520	211	516	220	520	213
PMULT	XAXON	832	274	1271	312	1251	342	1250	317
PMULT	XCAMP	823	267	1248	299	1227	329	1237	303
PMULT	XFAST	716	234	974	266	960	285	969	268
PMULT	YPEST-CO92	1320	648	1593	680	1561	729	1594	684
PMULT	YPEST-KIM	1316	639	1590	676	1556	726	1588	681
SALTY	WGLOS	572	381	606	408	601	411	606	408
SALTY	XAXON	1289	434	2115	497	1510*	371*	2082	510
SALTY	XCAMP	1282	427	2101	495	1456*	368*	2075	511
SALTY	XFAST	895	325	1303	392	1276	439	1291	399
SALTY	YPEST-CO92	2350	1758	2691	1724	2580	184	2691	1714
SALTY	YPEST-KIM	2368	1761	2708	1729	2583	1866	2707	1722
WGLOS	XAXON	464	194	537	213	526	224	537	213
WGLOS	XCAMP	463	194	539	217	530	227	539	216
WGLOS	XFAST	436	163	490	175	483	183	490	175
WGLOS	YPEST-CO92	570	377	604	404	601	408	604	407
WGLOS	YPEST-KIM	561	364	598	396	594	400	598	399
XAXON	XCAMP	3438	3256	3635	3223	3554	3320	3645	3221
XAXON	XFAST	1476	1075	1604	1083	1576	1105	1604	1078
XAXON	YPEST-CO92	1181	420	1916	482	1820*	470*	1887	499
XAXON	YPEST-KIM	1183	422	1935	475	1460*	368*	1906	490
XCAMP	XFAST	1465	1060	1601	1078	1578	1104	1600	1073
XCAMP	YPEST-CO92	1168	412	1892	466	1831*	456*	1860	490
XCAMP	YPEST-KIM	1162	412	1915	457	1863*	514*	1880	483
XFAST	YPEST-CO92	866	323	1234	379	1211	413	1225	386
XFAST	YPEST-KIM	861	315	1230	371	1208	402	1223	379
YPEST-CO92	YPEST-KIM	3387	3327	3629	3558	3622	3570	3632	3552

Table 3.5: Matching size (Size) and number of conserved adjacencies (#Adj) of solutions obtained by an earlier study of Angibaud *et al.* [6], programs `FFAdj-2G`, and `FFAdj-MCS`. The numbers marked by * indicate that the result was obtained from a possibly suboptimal solution returned by a prematurely terminated instance of `FFAdj-2G`.

are reported in [21] whereas in our current study we employed gene annotations from NCBI. Nevertheless, the difference in number of genes is on average 0.02% per genome. Secondly, the genes for `Adjacencies-Intermediate-Matching` are unsigned, which artificially increases the number of adjacencies. Taking into account the differences in gene number, we observed more adjacencies than in the matchings of Angibaud *et al.* [6]. Thereby, `FFAdj-2G` estimated on average 8.4% for $\alpha = 0.1$ and 8.2% for $\alpha = 0.5$, `FFAdj-MCS` even 10.0% more conserved adjacencies. Furthermore, the matching produced by `FFAdj-2G` was on average 28% ($\alpha = 0.1$), respectively 18.2% ($\alpha = 0.5$), and by `FFAdj-MCS` on average 27.2% larger than those produced by Angibaud *et al.* [6].

3.9.5 Discussion

In this chapter, we introduced a model for the identification of conserved adjacencies between two or more genomes that does not require prior knowledge of gene family assignments. To this end, we formulated problem FF-Adjacencies for pairwise comparisons which was subsequently extended for the case of more than two genomes. Problem FF-Adjacencies is parameterized by $\alpha \in [0, 1]$, which allows to balance the impact of synteny (i.e., conserved adjacencies) and similarities between genes on its solutions. We showed that for $0 < \alpha < \frac{1}{3}$, problem FF-Adjacencies in pairwise comparisons is NP-hard.

We then presented exact program `FFAdj-2G` and heuristic `FFAdj-MCS`. We further explored the space of feasible solutions to problem FF-Adjacencies analytically in context of the 0-1 linear program `FFAdj-2G`. This allowed us to remove suboptimal solutions from the solution space. In doing so, we identified simple conserved candidate adjacencies which were used as anchors in constructing an optimal solution to problem FF-Adjacencies between two genomes. By that, the number of variables and constraints in practical applications of program `FFAdj-2G` was reduced, resulting in a tremendous speedup, as shown in subsequent experiments. This allows us to compute exact solutions to problem FF-Adjacencies for genomes with more than 4000 genes.

The choice of α in problem FF-Adjacencies does not only influence the number of conserved adjacencies and the quality of orthology assignments in its solutions. It also has a high impact on the number of anchors that we are able to identify in the gene similarity graph of two genomes. That is to say, with higher α, the number of identified anchors decreases. This was predicted by our analytic framework, and subsequently confirmed in experiments on simulated and biological datasets. For $\alpha > 0.7$, the number of identified anchors was too low to compute exact solutions for problem FF-Adjacencies with program `FFAdj-2G`.

Experiments on simulated datasets showed high precision, recall, and accuracy of one-to-one orthology assignments derived from exact and heuristic solutions obtained by programs `FFAdj-2G` and `FFAdj-MCS`. Yet, maximum weight match-

ings (i.e, solutions to problem FF-Adjacencies for $\alpha = 0$) performed equally well w.r.t. to the quality of orthology assignments, while exhibiting significantly fewer conserved adjacencies. This result emphasizes the underlying goal in solving problem FF-Adjacencies and in performing gene family-free analysis in general: To find one-to-one homology assignments between two genomes that maximize a measure of gene order similarity. That being said, in general there exist many other synteny-independent true homology assignments between genes of two genomes that underwent a sufficient amount of gene duplication and loss. Our results show that the use of synteny improved the precision of orthology assignments only by 3% at most and made little to no improvements on the methods' recall. This result is backed by benchmarks of the orthology detection tool `PoFF` in [65], which uses our heuristic program `FFAdj-MCS` to marginally improve orthology assignments.

Lastly, we computed the number of conserved adjacencies in pairwise comparisons of twelve γ-proteobacterial genomes and compared our results to that of Angibaud *et al.* [6]. In doing so, we showed that our model establishes considerably larger matchings (leading to more one-to-one orthology assignments) between genes of the bacterial genomes. Moreover, our matchings contain on average more conserved adjacencies, too.

Chapter 4

Family-free median

In the previous chapter we mainly focused on a measure of gene family-free genome comparison for two genomes. Here, we go beyond pairwise comparisons and discuss a gene family-free model for the reconstruction of a possible candidate for the common ancestor of three genomes. In doing so, we extend the gene family-based problem of computing the *mixed multichromosomal breakpoint median* to a gene family-free setting. The present chapter is similarly structured as the previous: After a short review of the gene family-based problem in the subsequent section, we propose a gene family-free generalization. We then discuss its computational complexity by proving that the presented problem is MAX SNP-hard. Further, we formulate a 0-1 linear program that allows us to compute exact solutions. Whereas our model for computing family-free adjacencies between two genomes tolerated events of gene duplication and loss, the herein presented model is susceptible to gene losses and resolves gene duplications only to a limited extent. We discuss the effects of gene family evolution in our presented model and proceed to present a 0-1 linear program for computing gene family-free adjacencies between three genomes, thereby extending results of the previous chapter. Our algorithm gives rise to a heuristic approach to construct a median of three genomes in a family-free setting. We then compare both methods in simulated datasets. Lastly, we use our heuristic method to reconstruct the genome sequence of the *black death* again from genome sequences of three *Yersinia pestis* strains. We compare our results to those of Rajaraman *et al.* [87].

4.1 Gene family-based median of three

Ancestral gene order reconstruction represents a prominent path of research in genome analysis which has been studied intensely in the past decades. Hereby, the *median of three* constitutes a fundamental problem corresponding to the simplest unrooted phylogeny that contains an internal node. The median of three asks for the reconstruction of a fourth, ancestral genome, called a *median*, from three extant

genomes. To this end, one aims at minimizing the sum of pairwise distances between a median and the extant genomes, given a gene order distance measure of choice. Often, karyotypic constraints are raised, such as permitting only linear or circular chromosomes, or limiting the number of chromosomes in a median. Median problems for most distance measures are NP-hard, with the notable exception of the SCJ median [40] and the mixed multichromosomal breakpoint median [106]. For the case of three extant genomes, the latter can be stated as follows:

Problem 5 (Mixed multichromosomal breakpoint median of three) *Given three genomes G, H, and I of equal size $|G| = |H| = |I|$ and a one-to-one homology assignment $\mathcal{H}_1$ such that for every gene g in $\mathcal{C}(G)$ there exist exactly two genes $\{h, i\} \subset [g]_{\mathcal{H}_1}$ with $h \in \mathcal{C}(H)$ and $i \in \mathcal{C}(I)$. Find a genome M that minimizes the sum of distances*

$$d_{BP}(M, G) + d_{BP}(M, H) + d_{BP}(M, I)\,,$$

where $|M| = |G|$ and for each gene $g \in \mathcal{C}(G)$ there exists a unique gene $m \in \mathcal{C}(M)$ with $m \in [g]_{\mathcal{H}_1}$. Genome M is a mixed multichromosomal breakpoint median *of genomes G, H, and I.*

Tannier *et al.* presented in [106] a maximum weighted matching-based approach that solves Problem 5 in polynomial time. To this end, each gene family is assigned a unique identifier. Subsequently, genes are represented by their gene family label. Then, a graph Γ is constructed, in which each gene extremity x^a of a gene labeled with gene family x and terminal $a \in \{\mathrm{h}, \mathrm{t}\}$ corresponds to a vertex in Γ. Edges are drawn between all pairs of gene extremities, corresponding to possible adjacencies in a median M. Each edge is weighted $0, 1, 2$, or 3 according to its number of occurrences in extant genomes G, H, and I. Lastly, each vertex x^a gives rise to an additional, *telomeric* vertex t_{x^a} in Γ that is connected to x^a by an edge weighted $0, \frac{1}{2}, 1$, or $\frac{3}{2}$ according to *half the number* of occurrences of telomeric adjacency $\{\circ, x^a\}$ in extant genomes G, H, and I. A solution to the maximum weighted matching problem in graph Γ is a solution to Problem 5 [106]. Recently, Kováč proposed in [61] a similar approach on the basis of solving the computationally less expensive maximum cardinality matching problem.

4.2 A family-free generalization

Just like Problem 5, the family-free median problem asks for a fourth genome M that maximizes the sum of pairwise adjacency scores to three given extant genomes G, H, and I. However, the gene content of the requested median M must first be defined: each gene $m \in \mathcal{C}(M)$ must be unambiguously associated with a triple of extant genes (g, h, i), $g \in \mathcal{C}(G)$, $h \in \mathcal{C}(H)$, and $i \in \mathcal{C}(I)$. Moreover, the computation of adjacency scores of conserved adjacencies between the median and its extant genomes demands knowledge of the gene similarities between each triple of extant genes (g, h, i) and its corresponding, presumably extinct, gene m:

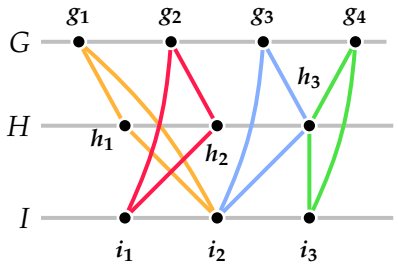
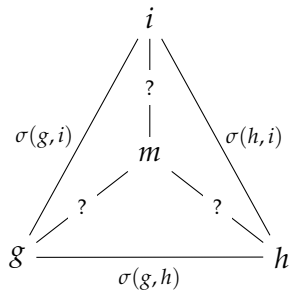


Figure 4.1: Gene similarity graph of three genomes G, H, and I of Example 4. The colored components give rise to candidate median genes, some of which are conflicting.



Since gene similarities to median genes are generally not given, we derive them from gene similarities between their corresponding extant genes. Following our scoring scheme of adjacency scores, we define the similarity between genes g, h, and i to its counterpart m as the geometric mean of their pairwise similarities:

$$\sigma(g,m) = \sigma(h,m) = \sigma(i,m) \equiv \sqrt[3]{\sigma(g,h) \cdot \sigma(g,i) \cdot \sigma(h,i)} \tag{4.1}$$

In the following we make use of mapping $\pi_G(m) \equiv g$, $\pi_H(m) \equiv h$, and $\pi_I(m) \equiv i$ to relate gene m with its extant counterparts. Two candidate median genes or telomeres m_1 and m_2 are *conflicting* if $m_1 \neq m_2$ and the intersection between associated gene sets $\{\pi_G(m_1), \pi_H(m_1), \pi_I(m_1)\}$ and $\{\pi_G(m_2), \pi_H(m_2), \pi_I(m_2)\}$ is non-empty. A median M is called *conflict-free* if no two genes or telomeres $m_1, m_2 \subseteq \mathcal{C}(M)$ are conflicting.

Example 4 *Genomes G, H, and I visualized in Figure 4.1 give rise to the following four candidate median genes: $m_1 = (g_1, h_1, i_2)$ represented by the yellow component, $m_2 = (g_2, h_2, i_1)$ represented by the red component, $m_3 = (g_3, h_3, i_2)$ represented by the blue component, and $m_4 = (g_4, h_3, i_3)$ represented by the green component. Further, candidate median gene pairs (m_1, m_3) and (m_3, m_4) are conflicting, respectively.*

We now introduce the problem of obtaining a median in a gene family-free setting:

Problem 6 (FF-Median) *Given three genomes G, H, and I, and gene similarity measure σ, find a conflict-free median M, which maximizes the following formula:*

$$\mathcal{F}_{\curlywedge}(M) = \sum_{\{m_1^a, m_2^b\} \in \mathcal{A}(M)} \sum_{\substack{X \in \{G,H,I\}, \\ \{\pi_X(m_1)^a, \pi_X(m_2)^b\} \in \mathcal{A}(X)}} s(m_1^a, m_2^b, \pi_X(m_1)^a, \pi_X(m_2)^b), \quad (4.2)$$

where $a, b \in \{h, t\}$ and $s(\cdot)$ is the adjacency score as defined by Equation (3.1).

The adjacency score of adjacency pair $\{m_1^a, m_2^b\}, \{\pi_X(m_1)^a, \pi_X(m_2)^b\}$, where $\{m_1^a, m_2^b\} \in \mathcal{A}(M)$ and $X \in \{G, H, I\}$, can be entirely expressed in terms of pairwise similarities between genes of extant genomes, G, H, and I, using Equation (4.1):

$$s(m_1^a, \pi_X(m_1)^a, m_2^b, \pi_X(m_2)^b) = \sqrt[6]{\prod_{\{Y,Z\} \subset \{G,H,I\}} \sigma(\pi_Y(m_1), \pi_Z(m_1)) \cdot \sigma(\pi_Y(m_2), \pi_Z(m_2))}$$

Note that the right side of the last equation is independent of genome X. From Equation (4.2) it becomes apparent that an adjacency in median M has only an impact in a solution to problem FF-Median if it participates in a conserved adjacency with genes from at least one extant genome. To indicate the presence of an adjacency $\{x_1^a, x_2^b\}$ in an extant genome X, we use the following function

$$\mathbb{I}_X(x_1^a, x_2^b) = \begin{cases} 1 & \text{if } \{x_1^a, x_2^b\} \in \mathcal{A}(X) \\ 0 & \text{otherwise.} \end{cases} \quad (4.3)$$

Moreover, a median gene can only give rise to a conserved adjacency with non-zero adjacency score, if all pairwise similarities of its corresponding extant genes g, h, i are non-zero. Thus, the search for M can be limited to those triplet sets of extant genes, with non-zero pairwise similarities. Henceforth, we assume that each gene in all extant genomes participates in at least one such triplet, otherwise it will be omitted in its corresponding genomic sequence.

In the following, a median gene m and its extant counterparts (g, h, i) are treated as equivalent. We denote the set of all *candidate median genes* by

$$\Sigma_{\curlywedge} = \{(g, h, i) \mid g \in \mathcal{C}(G), h \in \mathcal{C}(H), i \in \mathcal{C}(I) : \sigma(g, h) \cdot \sigma(g, i) \cdot \sigma(h, i) > 0\}. \quad (4.4)$$

Each pair of median genes $(g_1, h_1, i_1), (g_2, h_2, i_2) \in \Sigma_{\curlywedge}$ and terminals $a, b \in \{\mathrm{h}, \mathrm{t}\}$ give rise to a *candidate median adjacency* $\{(g_1^a, h_1^a, i_1^a), (g_2^b, h_2^b, i_2^b)\}$ if $(g_1^a, h_1^a, i_1^a) \neq (g_2^b, h_2^b, i_2^b)$, and (g_1^a, h_1^a, i_1^a) and (g_2^b, h_2^b, i_2^b) are non-conflicting. We denote the set of all candidate median adjacencies and the set of all *conserved* candidate median adjacencies by $\mathcal{A}_{\curlywedge} = \{\{m_1^a, m_2^a\} \mid m_1, m_2 \in \Sigma_{\curlywedge},\ a, b \in \{\mathrm{h}, \mathrm{t}\}\}$ and $\mathcal{A}_{\curlywedge}^C = \{\ \{m_1^a, m_2^b\} \in \mathcal{A}_{\curlywedge} \mid \sum_{X \in \{G,H,I\}} \mathbb{I}_X(\pi_X(m_1)^a, \pi_X(m_2)^b) \geq 1\}$, respectively.

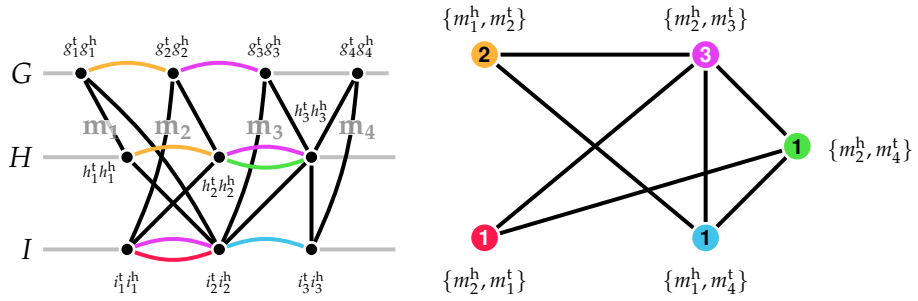


Figure 4.2: (a) Gene similarity graph of three genomes G, H and I of Example 4 and (b) its corresponding MAX WIS instance Λ with the weights of vertices written inside the circles.

4.3 Complexity of problem FF-Median

In this section, we discuss an inherent relation between problem FF-Median and the *maximum weighted independent set problem*. Given a graph $\Lambda = (V, E)$, the *maximum independent set problem* (MAX IS) asks for a maximum cardinality subset $\mathcal{I} \subseteq V$ such that no two vertices in $\mathcal{I}$ are connected by an edge of E. Any subset $\mathcal{I} \subseteq V$, whose induced subgraph $(\mathcal{I}, E)$ is entirely disconnected (i.e., contains no edges), is called an *independent set*. The *maximum weighted independent set problem* (MAX WIS) is a variant, in which each vertex v in V is assigned weight $w(v)$. MAX WIS asks for an independent set $\mathcal{I} \subseteq V$ of maximum weight. Both MAX IS and MAX WIS are NP-hard problems [45].

It is straightforward to see that problem FF-Median can be phrased as a MAX WIS problem: Construct graph $\Lambda = (V, E)$ with vertex set $V = \mathcal{A}_\chi^C$. Each vertex $\{m_1^a, m_2^b\} \in V$ is weighted according to the cumulative sum of adjacency scores of conserved adjacencies with genes in extant genomes:

$$w(\{m_1^a, m_2^b\}) = \sqrt[6]{\prod_{\{X,Y\}\subset\{G,H,I\}} \sigma(\pi_X(m_1), \pi_Y(m_1))\sigma(\pi_X(m_2), \pi_Y(m_2))} \cdot \sum_{X\in\{G,H,I\}} \mathbb{I}_X(\pi_X(m_1)^a, \pi_X(m_2)^b) \tag{4.5}$$

Further, any two pair of vertices $\{m_1^a, m_2^b\}, \{m_3^c, m_4^d\} \in V$, $a, b, c, d \in \{\mathrm{h}, \mathrm{t}\}$, are connected by an edge of E if (1) any two of the candidate median genes m_1, m_2, m_3, m_4 are conflicting, or (2) any two of the extremities $m_1^a, m_2^b, m_3^c, m_4^d$ are identical.

Example 4 (continued) *Figure 4.2 (a) depicts the gene similarity graph of the same three genomes G, H, and I as in Figure 4.1, this time arcs are colored and correspond to conserved adjacencies between pairs of non-conflicting candidate median genes. In doing so, the*

orientation of all genes in the diagram is always left (tail) to right (head). Figure 4.2 (b) visualizes the constructed MAX WIS instance Λ. Let all edges corresponding to pairwise similarities between genes have uniform weight, then the following solutions with score 3 are co-optimal for problem FF-Median and MAX WIS: (i) $\mathcal{A}(M) = \{\{m_2^h, m_3^t\}\}$ (purple vertex in Λ), (ii) $\mathcal{A}(M') = \{\{m_1^h, m_2^t\}, \{m_2^h, m_1^t\}\}$ (yellow and red vertices in Λ), (iii), $\mathcal{A}(M'') = \{\{m_1^h, m_2^t\}, \{m_2^h, m_4^t\}\}$ (yellow and green vertices in Λ).

Theorem 2 *Problem FF-Median is MAX SNP-hard.*

4.3.1 Reduction

The *maximum independent set problem for graphs bounded by node degree* 3, denoted as MAX IS-3 is MAX SNP-hard [81]. The corresponding decision problem can be informally stated as follows: Given a graph Λ bounded by degree 3 and some number $l \geq 1$, does there exists a set of vertices $V' \subseteq V$ of size $|V'| = l$ whose induced subgraph is unconnected? In the following, we present a transformation scheme $\mathbf{R}$ to phrase Λ as FF-median instance $\mathbf{R}(\Lambda) = (G, H, I, \sigma)$ such that the value $\mathcal{F}_\lambda(M)$ of a median M of $\mathbf{R}(\Lambda)$ is limited by $\mathcal{F}_\lambda(M) \leq 2 \cdot l + 3$. In doing so, we associate vertices of V with genes of extant genomes G, H and I. In order to keep track of associated genes, we denote by function $\xi(x)$ the list of vertices associated with gene x. We further introduce two types of unassociated genes, "$\emptyset$" and "$*$", whose members are identified by subscript notation.

Transformation $\mathbf{R}$:

1. Construct genome G such that for each vertex $v \in V$ there exist two associated genes $g_v, \bar{g}_v \in \mathcal{C}(G)$, i.e. $\xi(g_v) = \xi(\bar{g}_v) = v$. Further, let each gene pair $g_v, \bar{g}_v$ form a circular chromosome, giving rise to adjacency set $\mathcal{A}(G) = \{\{\bar{g}_v^{\mathrm{h}}, g_v^{\mathrm{t}}\}, \{g_v^{\mathrm{h}}, \bar{g}_v^{\mathrm{t}}\} \mid v \in V,\ g_v, \bar{g}_v \in \mathcal{C}(G)\}$.

2. For each edge $(u, v) \in E$ construct a circular chromosome $\mathcal{X}_{uv}$ hosting two genes $x_{uv}, x_\emptyset \in \mathcal{C}(\mathcal{X}_{uv})$, with gene x_{uv} being associated with both vertices u and v and gene $x_\emptyset$ being unassociated. Further, let both genes form a circular chromosome, giving rise to adjacency set $\mathcal{A}(\mathcal{X}_{uv}) = \{\{x_{uv}^{\mathrm{h}}, x_\emptyset^{\mathrm{t}}\}, \{x_\emptyset^{\mathrm{h}}, x_{uv}^{\mathrm{t}}\}\}$.

3. Assign each chromosome constructed in the previous step either to genome H or to genome I such that each vertex $v \in V$ is associated with at most two genes per genome.

4. Complete genomes H and I with additional circular chromosomes $\mathcal{X}_v$ where $\mathcal{C}(\mathcal{X}_v) = \{x_v, x_\emptyset\}$ and $\mathcal{A}(\mathcal{X}_v) = \{\{x_v^{\mathrm{h}}, x_\emptyset^{\mathrm{t}}\}, \{x_\emptyset^{\mathrm{h}}, x_v^{\mathrm{t}}\}\}$ such that each vertex in V is associated with exactly two genes per genome.

5. For each vertex $v \in V$, let $g, \bar{g} \in \mathcal{C}(G)$, $h, \bar{h} \in \mathcal{C}(H)$, and $i, \bar{i} \in \mathcal{C}(I)$ be the pairs of genes associated with v, i.e. $\xi(g) = \xi(\bar{g}) = \xi(h) \cap \xi(i) = \xi(\bar{h}) \cap \xi(\bar{i}) = v$.

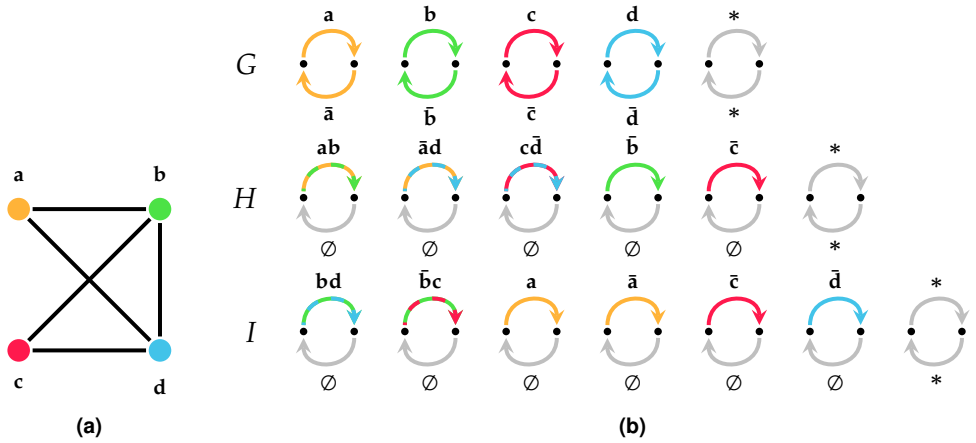


Figure 4.3: (a) A simple graph bounded by degree three and (b) a corresponding FF-Median instance constructed with transformation scheme **R**.

Assign gene similarities $\sigma(g,h) = \sigma(g,i) = \sigma(h,i) = 1$ and $\sigma(\bar{g},\bar{h}) = \sigma(\bar{g},\bar{i}) = \sigma(\bar{h},\bar{i}) = 1$.

6. Add a copy of circular chromosome $\mathcal{X}_*$ to each genome G, H, and I, where $\mathcal{C}(\mathcal{X}_*) = \{x_*, \bar{x}_*\}$ and $\mathcal{A}(\mathcal{X}_*) = \{\{x_*^{\mathrm{h}}, \bar{x}_*^{\mathrm{t}}\}, \{\bar{x}_*^{\mathrm{h}}, x_*^{\mathrm{t}}\}\}$. Let $g_*, \bar{g}_* \in \mathcal{C}(G), h_*, \bar{h}_* \in \mathcal{C}(H)$, and $i_*, \bar{i}_* \in \mathcal{C}(I)$, set the gene similarity score between all pairs of genes in $\{g_*, h_*, i_*\}$ and $\{\bar{g}_*, \bar{h}_*, \bar{i}_*\}$ respectively, to 1. Lastly, set the gene similarity score of all pairs of unassociated genes of type "$\emptyset$" including genes $g_*, \bar{g}_*$ to $\frac{1}{4}$.

Except for step 3, none of the instructions of transformation scheme **R** are computationally challenging. Note that in step 3 the demanded partitioning of chromosomes into genomes H and I is always possible as consequence of Vizing's Theorem [110], by which every graph with maximum node degree d is edge-colorable using at most d or $d+1$ colors. Hence, using colors $\chi_1, \chi_2, \chi_3, \chi_4$ with $\chi_1 = \chi_2 \equiv I$, $\chi_3 = \chi_4 \equiv H$ and Misra and Gries' algorithm [75], edges of graph $\Lambda = (E, V)$ can be partitioned into two groups in $\mathcal{O}(|E||V|)$ time implying an assignment to genomes H and I.

Example 5 *Figure 4.3 (b) shows a FF-Median instance constructed with transformation scheme **R** from the simple graph depicted in Figure 4.3 (a). Gene similarities between genes are not shown, but can be derived from the genes' labeling.*

We structure our proof that the presented transformation is in fact a valid mapping of an MAX IS-3 instance to an instance of FF-Median into three different lemmas:

Lemma 4 *Given a median M of FF-Median instance $\mathbf{R}(\Lambda) = (G, H, I, \sigma)$, (1) for each median gene $(g, h, i) \in \mathcal{C}(M)$ where g, h, or i are associated with vertices in $V(\Lambda)$ holds $\xi(g) = \xi(h) \cap \xi(i) = v$, $v \in V(\Lambda)$; (2) there exist at most two median genes whose corresponding extant genes are not associated to any vertex in $V(\Lambda)$.*

Proof: Assume for contradiction that claim (1) does not hold. Then either $\xi(g) \neq \xi(h) \cap \xi(i)$, or $\xi(h) \cap \xi(i) = \emptyset$, both of which violate the constraint of establishing gene similarities between associated genes that is given in step 5. For claim (2), observe that the only unassociated genes in genome G are genes g_* and $\bar{g}_*$ introduced in step 6, limiting the overall number of unassociated genes in any median M. □

Lemma 5 *The conserved adjacency set of any median M of FF-Median instance $\mathbf{R}(\Lambda) = (G, H, I, \sigma)$ is of the form $\mathcal{A}(M) \cap \mathcal{A}_\lambda^{\mathsf{C}} = \mathcal{A}_\lambda^{G}(M) \cup \{\{m_*^{\mathrm{h}}, \overline{m}_*^{\mathrm{t}}\}, \{\overline{m}_*^{\mathrm{h}}, m_*^{\mathrm{t}}\}\}$, where the extant genes corresponding to m_* and $\overline{m}_*$ are all unassociated genes of type "$*$" and $\mathcal{A}(M)_\lambda^{G} \subseteq \{\{m_1^{\mathrm{h}}, m_2^{\mathrm{t}}\} \in A_\lambda^{\mathsf{C}} \mid \xi(\pi_G(m_1)) = \xi(\pi_G(m_2))\}$.*

Proof: Observe that both candidate median adjacencies $a_* = \{m_*^{\mathrm{h}}, \overline{m}_*^{\mathrm{t}}\}$ and $\bar{a}_* = \{\overline{m}_*^{\mathrm{h}}, m_*^{\mathrm{t}}\}$ are conserved in all three genomes, whereas all other conserved candidate adjacencies between associated and unassociated genes can be at most conserved in H and I. Establishing adjacencies $a_*, \bar{a}_*$ gives rise to a cumulative adjacency score of 6. Conversely, up to 4 non-conflicting adjacencies between associated and unassociated genes can be established that are conserved in both genomes H and I. However, since such adjacencies are only conserved between unassociated genes of type "$\emptyset$" whose gene similarities are set to $\frac{1}{4}$, the best cumulative adjacency score can not exceed 4. Thus, adjacencies $a_*, \bar{a}_*$ must be contained in any median. Further, because of this and the fact that in both genomes H and I, each gene associated with vertices of $V(\Lambda)$ is only adjacent to an unassociated gene, M cannot contain adjacencies that are conserved in extant genomes other than genome G, which are the adjacencies of each gene pair $(g_v, \bar{g}_v)$ associated with the same vertex $v \in V(\Lambda)$. □

Lemma 6 *Given FF-median instance $\mathbf{R}(\Lambda) = (G, H, I, \sigma)$, let m_u, m_v be any pair of candidate median adjacencies of $\mathcal{A}_\lambda$ whose corresponding extant genes are associated to vertices $u, v \in V(\Lambda)$, then m_u, m_v are conflicting if and only if $(u, v) \in E$.*

Proof: By construction in step 5 of transformation scheme $\mathbf{R}$, each vertex $v \in V$ is associated with exactly two candidate median genes $m_v = (g, h, i), \overline{m}_v = (\bar{g}, \bar{h}, \bar{i})$, $m_v, \overline{m}_v \in \Sigma_\lambda$, such that $\xi(g) = \xi(h) \cap \xi(i) = v$ and $\xi(\bar{g}) = \xi(\bar{h}) \cap \xi(\bar{i}) = v$. Further, let u be another vertex of $V(\Lambda)$, such that $(u, v) \in E(\Lambda)$, and $m_u, \overline{m}_u$ are its two corresponding candidate median genes. Then, by construction in step 2, there exists exactly one extant gene x with $\xi(x) = uv$ (which, by assignment in step 3, is either contained in genome H or in genome I). Consequently, either m_u is in conflict with m_v, or $\overline{m}_u$ with $\overline{m}_v$, or $\overline{m}_u$ with m_v, or m_u with $\overline{m}_v$. Recall that by construction in step

2 in $\mathbf{R}$ and by Lemma 5, $m_u, \overline{m}_u$ and $m_v, \overline{m}_v$ form conserved candidate adjacencies $\{m_u^{\mathrm{h}}, \overline{m}_u^{\mathrm{t}}\}$, $\{\overline{m}_u^{\mathrm{h}}, m_u^{\mathrm{t}}\}$ and $\{m_v^{\mathrm{h}}, \overline{m}_v^{\mathrm{t}}\}$, $\{\overline{m}_v^{\mathrm{h}}, m_v^{\mathrm{t}}\}$, respectively. Clearly, independent of which of the candidate median gene pairs of u and v are in conflict, both pairs of candidate median adjacencies are in conflict with each other.

Now, let u, v be two vertices of $V(\Lambda)$ such that edge $(u, v) \notin E(\Lambda)$, then there exists no gene x in extant genomes H and I with $\xi(x) = uv$. Even more, due to Lemma 4, there cannot exist a candidate median gene (g, h, i) with $\{u, v\} \subseteq \xi(g) \cup \xi(h) \cup \xi(i)$. Thus, the candidate median genes of u and v are not conflicting and neither are their corresponding candidate median adjacencies. □

We proceed to show that the given transformation scheme gives rise to an approximation preserving reduction known as *L-reduction*. An L-reduction reduces a problem P to a problem Q by means of two polynomial-time computable transformation functions: A function $f : P \to Q' \subseteq Q$ that maps each instance of P onto an instance of Q, herein represented by transformation scheme $\mathbf{R}$, and a function $g : Q' \to P$ to transform any feasible solution of an instance in Q' to a feasible solution of an instance of P. Here, a *feasible* solution means any – not necessarily *optimal* – solution that obeys the problem's constraints. A feasible solution of FF-Median instance (G, H, I, σ) is an *ancestral genome* X where $\mathcal{C}(X) \subseteq \Sigma_\lambda$ and $\mathcal{A}(X) \subseteq \mathcal{A}_\lambda$ such that $\mathcal{A}(X)$ is conflict-free. We give the following transformation scheme to map ancestral genomes of an FF-Median instance to solutions of a MAX IS-3 instance:

Transformation $\mathbf{S}$: Given any ancestral genome X of $\mathbf{R}(\Lambda)$, return $\{\xi(\pi_G(m_1)) \mid \{m_1^a, m_2^b\} \in \mathcal{A}(X) : \mathbb{I}_G(\pi_G(m_1)^a, \pi_G(m_2)^b) = 1 \text{ and } \xi(\pi_G(m_1)) \neq \emptyset\}$.

We define score function $s_\lambda(X) \equiv \frac{1}{2}\mathcal{F}_\lambda(X) - 3$ of an ancestral genome X. For $(\mathbf{R}, \mathbf{S})$ to be an L-reduction the following two properties must hold for any given MAX IS-3 instance (Λ, l): (1) There is some constant α such that for any median M of the transformed FF-Median instance $\mathbf{R}(\Lambda)$ holds $s_\lambda(M) \leq \alpha \cdot l$; (2) There is some constant β such that for any ancestral genome X of $\mathbf{R}(\Lambda)$ holds $l - |\mathbf{S}(X)| \leq \beta \cdot |s_\lambda(M) - s_\lambda(X)|$. We proceed to prove the following lemma:

Lemma 7 $(\mathbf{R}, \mathbf{S})$ *is an L-reduction of problem MAX IS-3 to problem FF-Median with* $\alpha = \beta = 1$.

Proof: For any median M of FF-Median instance $\mathbf{R}(\Lambda)$, the number of conserved median adjacencies with correspondence to the same vertex of Λ is two, giving rise a cumulative adjacency score of two. From Lemmata 5 and 6 immediately follows that any ancestral genome of $\mathbf{R}(\Lambda)$ that maximizes the number of conserved adjacencies also maximizes the number of independent vertices in Λ. Recall that the two conserved adjacencies between unassociated genes of type "$*$" (which are part of all medians) give rise to a cumulative adjacency score of 6, we conclude that $|\mathcal{A}(M) \cap \mathcal{A}_\lambda^{\mathrm{C}}| - 2 = \frac{1}{2}\mathcal{F}_\lambda(M) - 3 = s_\lambda(M) = l$, thus $\alpha = 1$.

Because $l = s_\lambda(M)$, it remains to show that $l - |S(X)| \leq \beta|l - s_\curlywedge(X)|$. In a *sub-optimal* ancestral genome of $\mathbf{R}(\Lambda)$, median genes with no association to vertices of Λ can also contain extant genes of type "$\emptyset$". These unassociated median genes can form "mixed" conserved adjacencies with genes that are associated with vertices of Λ. Such mixed conserved adjacencies have no correspondence to vertices in Λ and do not contribute to the transformed solution $\mathbf{S}(X)$ of an ancestral genome X. Yet, as mentioned earlier, the cumulative adjacency score of all mixed conserved adjacencies can not not exceed 4. Therefore it holds that $|S(X)| \geq s_\curlywedge(X)$ and we conclude $\beta = 1$. □

4.4 An exact solution to problem FF-Median

Being a well-studied problem, there exist various exact and approximation algorithms for MAX-WIS [12, 54, 82]. Problem FF-Median, formulated as MAX-WIS, gives rise to $\mathcal{O}(n^5)$ vertices and $\mathcal{O}(n^9)$ edges in graph Λ of three genomes G, H, and I, where $n = \max(|G|, |H|, |I|)$. That is because there are $\mathcal{O}(n)$ adjacencies in each extant genome, whose contained genes can participate in $\mathcal{O}(n^4)$ candidate median adjacencies. Each of the $\mathcal{O}(n^5)$ candidate median adjacencies can be in conflict with $\mathcal{O}(n^4)$ other candidate median adjacencies, contributing to a total number of $\mathcal{O}(n^9)$ edges in Λ. In this section, we present program `FF-Median`, described by Algorithm 3, that exploits the specific properties of problem FF-Median, thereby using only $\mathcal{O}(n^5)$ variables and statements. It is an adaptation of Tannier *et al.*'s algorithm described in Section 4.1.

Program `FF-Median` makes use of two types of binary variables **a** and **b** as declared in domain specifications (`D.01`) and (`D.02`). The former variable type indicates the presence or absence of candidate genes in an optimal median M. The latter, variable type **b**, specifies if an adjacency between two gene extremities or telomeres is established in M. Recall that a solution to problem FF-Median is a conflict-free median M whose cumulative adjacency score of conserved adjacencies between genes of M and genes of extant genomes is maximized. Hence, the objective function of program `FF-Median` sums over the adjacency score of conserved candidate median adjacencies multiplied by variable **b**. In doing so, the indicator function of Equation (4.3) determines the multiplicity (i.e., $0, 1, 2$, or 3) of conserved adjacencies between median genes and extant counterparts. Constraint (`C.01`) ensures that M is conflict-free, by demanding that each extant gene (or telomere) can be associated with at most one median gene (or telomere). Further, constraint (`C.02`) dictates that a median adjacency can only be established between genes that both are part of the median. Lastly, constraint (`C.03`) guarantees that each gene extremity and telomere of the median participates in at most one adjacency.

Just like independent sets that are solutions to MAX-WIS do not necessarily lead to a valid genome M in the proof of Theorem 2, variable assignments returned

Algorithm 3 Program `FF-Median` is an ILP for finding an an optimal solution to FF-Median (Problem 6) for 3 genomes.

Objective:

`Maximize`

$$\sum_{\substack{(g_1,h_1,i_1),(g_2,h_2,i_2)\in\Sigma_\curlywedge,\\ a,b\in\{\mathrm{h},\,\mathrm{t}\}}} \mathbf{b}(g_1^a,g_2^b,h_1^a,h_2^b,i_1^a,i_2^b)\ \sqrt[6]{\sigma(g_1,h_1,i_1)\sigma(g_2,h_2,i_2)}\ (\mathbb{I}_G(g_1^a,g_2^b)+\mathbb{I}_H(h_1^a,h_2^b)+\mathbb{I}_I(i_1^a,i_2^b))$$

Constraints:

(`C.01`) $\forall\, g' \in \mathcal{C}_\circ(G)$: $\sum_{\substack{(g,h,i)\in\Sigma_\curlywedge,\\ g=g'}} \mathbf{a}(g,h,i) \le 1$

$\forall\, h' \in \mathcal{C}_\circ(H)$: $\sum_{\substack{(g,h,i)\in\Sigma_\curlywedge,\\ h=h'}} \mathbf{a}(g,h,i) \le 1$

$\forall\, i' \in \mathcal{C}_\circ(I)$: $\sum_{\substack{(g,h,i)\in\Sigma_\curlywedge,\\ i=i'}} \mathbf{a}(g,h,i) \le 1$

(`C.02`) $\forall\, (g_1,h_1,i_1),(g_2,h_2,i_2) \in \Sigma_\curlywedge$ and $\forall\, a,b \in \{\mathrm{h},\,\mathrm{t}\}$:

$$2\cdot\mathbf{b}(g_1^a,g_2^b,h_1^a,h_2^b,i_1^a,i_2^b) \le \mathbf{a}(g_1,h_1,i_1)+\mathbf{a}(g_2,h_2,i_2)$$

(`C.03`) $\forall\, (g_1,h_1,i_1) \in \Sigma_\curlywedge$ and $\forall\, a \in \{\mathrm{h},\,\mathrm{t}\}$:

$$\sum_{\substack{(g_2,h_2,i_2)\in\Sigma_\curlywedge,\\ b\in\{\mathrm{h},\,\mathrm{t}\}}} \mathbf{b}(g_1^a,g_2^b,h_1^a,h_2^b,i_1^a,i_2^b) \le 1$$

Domains:

(`D.01`) $\forall\, (g,h,i) \in \Sigma_\curlywedge$: $\mathbf{a}(g,h,i) \in \{0,1\}$

(`D.02`) $\forall\, (g_1,h_1,i_1),(g_2,h_2,i_2) \in \Sigma_\curlywedge$ and $\forall\, a,b \in \{\mathrm{h},\,\mathrm{t}\}$:

$$\mathbf{b}(g_1^a,g_2^b,h_1^a,h_2^b,i_1^a,i_2^b) \in \{0,1\}$$

by program `FF-Median` result in an incomplete adjacency set $\mathcal{A}_\circ(M)$, lacking unconserved adjacencies. Certainly, additional variables and constraints can be introduced that prohibit this shortcoming. Yet, in view of the already large number of constraints that are imposed by program `FF-Median`, we propose the following naïve post-processing scheme of its output, in order to construct a valid median M that is a solution to problem FF-Median:

1. Create empty genome M;

2. For each variable $\mathbf{a}(g, h, i)$ with value 1, create a unique gene $m = (g, h, i)$; add m to the set of genes and telomeres $\mathcal{C}_\circ(M)$;
3. For each variable $\mathbf{b}(g_1^a, g_2^b, h_1^a, h_2^b, i_1^a, i_2^b)$ with value 1, add median adjacency $\{(g_1^a, h_1^a, i_1^a), (g_2^b, h_2^b, i_2^b)\}$ to $\mathcal{A}_\circ(M)$
4. Arbitrarily create adjacencies between any two gene extremities m_1^a and m_2^b, that do not participate in any adjacencies of $\mathcal{A}_\circ(M)$.

Several straightforward simplifications will greatly reduce the number of variables and constraints. First and foremost, we implement Kováč's approach [61] of instantiating only variables for candidate median adjacencies that are supported by at least one extant adjacency, thereby reducing the number of variables $\mathbf{b}$ and their related constraints from $\mathcal{O}(n^6)$ to $\mathcal{O}(n^5)$. Moreover, we omit variables $\mathbf{b}$ between pairs of conflicting candidate median genes, because the representing adjacencies can never be part of a conflict-free median. At last, we add every candidate gene m to median M that is not conflicting with any other candidate median gene. Note that even if gene m does not participate in any conserved adjacency, its presence in M does not deteriorate score $\mathcal{F}_\lambda(M)$. Thus, we can already fix the value of its corresponding variable $\mathbf{a}(g, h, i)$ to 1.

4.5 The effect of gene family evolution on family-free medians

Gene families evolve by duplication, speciation, and loss, thereby altering the order of genes in genomes. Gene family-based approaches handle gene family evolution either prior to gene order analysis by pruning genomes into equal gene content, or allow duplicate genes in gene order comparisons. In case of the latter, resulting ambiguities in the ancestral (conserved) gene order are resolved by obtaining an optimal gene order over all possible matchings between gene duplicates. To this end, different types of matchings are employed, such as exemplar, intermediate, or maximum matching [6, 22].

When inferring homology assignments, family-free analysis inevitably entails the necessity of taking the effects of gene family evolution into account. Problem FF-Adjacencies, as described in Section 3.2, handles differences in gene content caused by gene family evolution by facilitating the establishment of conserved adjacencies in subgenomes. This implicates the omission of genes intermitting conserved adjacencies in the final matching.

Problem FF-Median can tolerate differences in gene content only to a limited extent. A positive example is shown on the left side of Figure 4.4. It visualizes the outcome of a duplication event of a gene belonging to gene family a, which occurred along the evolutionary path leading to genome I after speciation from the common ancestor of G, H, and I, i.e., median M. The true ancestral gene order "$a\ b\ c$" will be recovered as long as the cumulative score of the adjacency between a and b (yellow

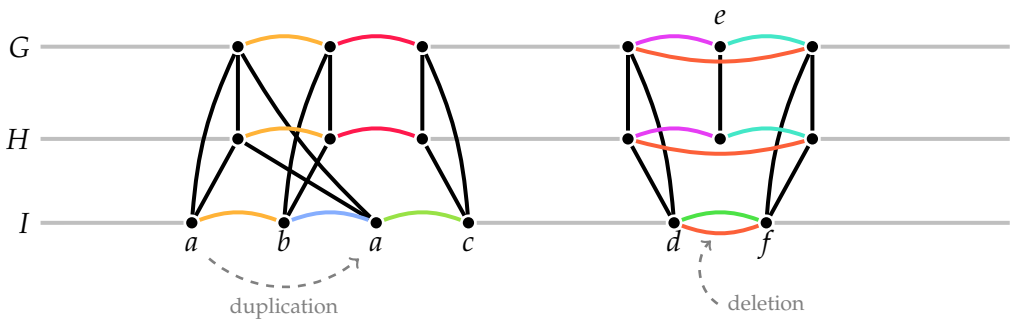


Figure 4.4: The effect of duplication and deletion of a single gene in problem FF-Median. Colored arcs correspond to potential median adjacencies.

arcs), which is conserved in all three extant genomes, plus the score of the twofold conserved adjacency between b and c (red arcs) is larger than the cumulative score of the onefold conserved adjacencies b, a (blue arc) and a, c (green arc) of genome I. Note this is true in particular when genes within gene families exhibit uniform gene similarities. Yet, in other cases, where immediate neighborhoods of true positional homologs are less conserved, problem FF-Median likely fails to obtain the correct ancestral gene order. Even worse, it is generally affected by gene deletion events, such as the one shown in the example on the right side of Figure 4.4. Thereby, similar to the previous example, the gene belonging to gene family e was lost along the evolutionary path from median M to genome I. Since problem FF-Median requires a 3-matching between genes of all extant genomes, the remaining gene family members in genomes G and H can never become part of the reconstructed median M. Whereas the inability to reconstruct the true gene content of median M is thereby concluded, the broader gene order of genes belonging to families d and f can still be recovered. Their common adjacency, conserved in genome I (green arc), can be strengthened through a preprocessing method which removes genes from gene orders of G, H, and I, that are singletons or only connected to genes belonging to two extant genomes. In the example, such an approach would result in a threefold conserved adjacency between gene families d and f (orange arcs). Yet, whenever the remaining gene family members, that are homologous to the lost gene, share gene similarities with further genes of the third genome, the preprocessing method is doomed to failure and it becomes less likely that even the broader gene order is recovered.

Problem FF-Median can be extended to account for gene insertions and deletions, as well as perturbations in the assessment of gene similarities by (i) considering conserved adjacencies in subgenomes of genomes G, H, and I, and (ii) relaxing the 3-matching to a partial 3-matching. In doing so, the set of candidate genes of median M is extended to include all sets of extant genes that form subcomponents

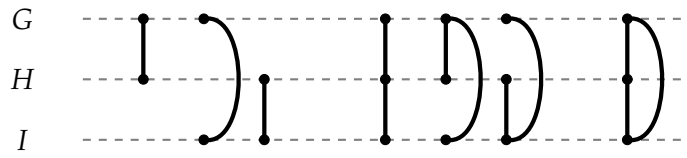


Figure 4.5: The seven valid types of components of a partial 3-matching.

of cliques of size three in the gene similarity graph of extant genomes G, H, and I. Figure 4.5 visualizes the seven possible subcomponents permitted by a partial 3-matching. The resulting gene content of family-free median genomes corresponds to genes conserved in at least two extant genomes, i.e., genes that share at least one edge in the gene similarity graph.

However, the improved capability of resolving events of gene family evolution comes at the expense of a dramatically increased search space of optimal family-free medians. This makes the computation of such medians extremely challenging. We henceforth propose an alternative, practically motivated approach, by solving problem FF-Adjacencies for three genomes G, H, and I, and subsequently applying Tannier *et al.*'s algorithm [106] (Section 4.1) on its outcome to obtain a median genome.

4.6 Solving problem FF-Adjacencies for three genomes

We now describe program `FFAdj-3G`, as shown in Algorithm 4. It is an adaptation of program `FFAdj-2G` that was discussed in Section 3.6. Program `FFAdj-3G` returns an exact solution to problem FF-Adjacencies for three genomes G, H, and I (see Problem 2 in Section 3.3), given their gene similarity graph $B_\circ = (G, H, I, E)$. To this end, the program makes use of the same three types of binary variables a, b, and c (see domains (`D.01`) - (`D.03`)) previously described in Section 3.6.

Constraints (`C.01`) and (`C.03`) ensure that the resulting matching $\mathcal{M}$ forms a valid partial 3-matching. That is, no two genes of a connected component in the $\mathcal{M}$-induced subgraph $B_\circ^{\mathcal{M}}$ belong to the same genome (see Definition 8). In doing so, (`C.01`) establishes pairwise matching constraints, i.e., it guarantees that in the matching-induced subgraph, each gene is connected to at most one gene per genome. Note that — unlike in program `FFAdj-2G` — variables **b** are assigned 1 for each gene that is incident to *at least one* edge of partial 3-matching $\mathcal{M}$. That is, the value of a variable **b** can be 1 even though its corresponding gene is not incident to an edge of $\mathcal{M}$. But then, program `FFAdj-3G` permits a gene to be incident to several edges of $\mathcal{M}$, if each of these edges is incident to genes of different genomes. Additional constraints are enforced by (`C.03`) on every pair of edges that share a common gene in one genome, but are incident to genes of different genomes. Let

Algorithm 4 Program `FFAdj-3G` is an ILP for finding an an optimal solution for FF-Adjacencies (Problem 2) for three genomes.

Objective:

`Maximize`

$$\sum_{\{X,Y\}\subset\{G,H,I\}} \alpha \cdot \sum_{\substack{\{x_1^a,x_2^b\}\in\mathcal{A}^\star(X),\\ \{y_1^a,y_2^b\}\in\mathcal{A}^\star(Y)}} s(x_1^a,x_2^b,y_1^a,y_2^b)\cdot \mathbf{c}(x_1^a,x_2^b,y_1^a,y_2^b) + (1-\alpha)\cdot \sum_{\substack{x\in\mathcal{C}_\circ(X),\\ y\in\mathcal{C}_\circ(Y)}} \sigma(x,y)\cdot \mathbf{a}(x,y)$$

Constraints:

(`C.01`) for all $\{X,Y\}\subset\{G,H,I\}$, for all $x\in\mathcal{C}_\circ(X)$,

$$\sum_{y\in\mathcal{C}_\circ(Y)} \mathbf{a}(x,y) \le \mathbf{b}(x)$$

(`C.02`) for all $\{X,Y\}\subset\{G,H,I\}$,

for all $\{x_1^a,x_2^b\}\in\mathcal{A}^\star(X)$, for all $\{y_1^a,y_2^b\}\in\mathcal{A}^\star(Y)$

$\mathbf{a}(x_1^a,y_1^b)+\mathbf{a}(x_2^a,y_2^b)-2\cdot\mathbf{c}(x_1^a,x_2^b,y_1^a,y_2^b)\ge 0$,

if $\{x_1^a,x_2^b\}\notin\mathcal{A}_\circ(X)$, then for all x in $\{x_3,\ldots,x_n\}\subseteq\mathcal{C}(X)$

such that $\{\{x_1^a,x_3^{a_3}\},\{x_3^{b_3},x_4^{a_4}\},\ldots,\{x_n^{b_n},x_2^b\}\}\subseteq\mathcal{A}_\circ(X)$,

$\mathbf{b}(x)+\mathbf{c}(x_1^a,x_2^b,y_1^a,y_2^b)\le 1$,

if $\{y_1^a,y_2^b\}\notin\mathcal{A}_\circ(Y)$, then for all $y\in\{y_3,\ldots,y_n\}\subseteq\mathcal{C}(Y)$

such that $\{\{y_1^a,y_3^{a_3}\},\{y_3^{b_3},y_4^{a_4}\},\ldots,\{y_n^{b_n},y_2^b\}\}\subseteq\mathcal{A}_\circ(Y)$,

$\mathbf{b}(y)+\mathbf{c}(x_1^a,x_2^b,y_1^a,y_2^b)\le 1$

(`C.03`) $\forall\, g\in\mathcal{C}_\circ(G),\ \forall\, h\in\mathcal{C}_\circ(H),\ \forall\, i\in\mathcal{C}_\circ(I)$, if $\sigma(g,h)>0$ and $\sigma(g,i)>0$,

$$\sum_{\substack{h'\in\mathcal{C}_\circ(H)\\ h'\neq h}} \mathbf{a}(h',i)+\mathbf{a}(g,h)+\mathbf{a}(g,i)\le 2,\ \sum_{\substack{i'\in\mathcal{C}_\circ(I)\\ i'\neq i}} \mathbf{a}(h,i')+\mathbf{a}(g,h)+\mathbf{a}(g,i)\le 2$$

if $\sigma(g,h)>0$ and $\sigma(h,i)>0$,

$$\sum_{\substack{g'\in\mathcal{C}_\circ(G)\\ g'\neq g}} \mathbf{a}(g',i)+\mathbf{a}(g,h)+\mathbf{a}(h,i)\le 2,\ \sum_{\substack{i'\in\mathcal{C}_\circ(I)\\ i'\neq i}} \mathbf{a}(g,i')+\mathbf{a}(g,h)+\mathbf{a}(h,i)\le 2,$$

if $\sigma(g,i)>0$ and $\sigma(h,i)>0$,

$$\sum_{\substack{g'\in\mathcal{C}_\circ(G)\\ g'\neq g}} \mathbf{a}(g',h)+\mathbf{a}(g,i)+\mathbf{a}(h,i)\le 2,\ \sum_{\substack{h'\in\mathcal{C}_\circ(H)\\ h'\neq h}} \mathbf{a}(g,h')+\mathbf{a}(g,i)+\mathbf{a}(h,i)\le 2$$

Domains:

(`D.01`) for all $g\in\mathcal{C}_\circ(G)$ and for all $h\in\mathcal{C}_\circ(H)$, $\mathbf{a}(g,h)\in\{0,1\}$

(`D.02`) for all $g\in\mathcal{C}_\circ(G)$, $\mathbf{b}(g)\in\{0,1\}$,

for all $h\in\mathcal{C}_\circ(H)$, $\mathbf{b}(h)\in\{0,1\}$

(`D.03`) for all $\{g_1^a,g_2^b\}\in\mathcal{A}^\star(G)$ and for all $\{h_1^a,h_2^b\}\in\mathcal{A}^\star(H)$,

$\mathbf{c}(g_1^a,g_2^b,h_1^a,h_2^b)\in\{0,1\}$

us consider three genes $g \in G, h \in H$, and $i \in I$, which are connected by two edges $\{g,h\},\{g,i\} \in E$. This scenario is represented in Figure 4.6, where the two edges $\{g,h\}$ and $\{g,i\}$ are colored green. The figure schematizes all 16 combinations in which edges in the neighborhood of $\{g,h\}$ and $\{g,i\}$ (including $\{g,h\}$ and $\{g,i\}$) can participate in a matching only constrained by (C.01), where saturated edges are indicated by thick continuous lines and unsaturated edges by dashed lines. For instance, Figure 4.6(a) represents the case in which no edge in the neighborhood of $\{g,h\}$ and $\{g,i\}$ is saturated. When applying Constraint (C.03) on the visualized combinations, it is ensured that (i) the sum of saturated edges that are red or green is less than or equal to two, and (ii) that the sum of saturated edges that are blue or green is less than or equal to two. Combinations that violate any of the two sum constraints, shown in Figures 4.6(h), 4.6(l), and 4.6(p), are exactly those that violate the partial 3-matching property. The black line between genes h and i indicates that edge $\{h,i\}$ can be saturated or unsaturated, but is not considered by the two sum constraints. In case edge $\{h,i\}$ is saturated, conflicts with additionally saturated blue and red edges, resulting in violations of the partial 3-matching constraint, are already prohibited by the pairwise matching constraints of (C.01).

Lastly, Constraint (C.02) covers the rules of forming conserved adjacencies: (i) it ensures that a variable **c**, which indicates a conserved adjacency for two edges, is set to 1 only if the edges are saturated; (ii) using variables **b**, it prohibits that no gene (and thus no incident edge) within a conserved adjacency is part of the matching.

Approaches presented in Section 3.7 to identify and remove suboptimal solutions from the solution space in pairwise comparisons cannot be applied in the comparison of three genomes. Consequently, it is impossible to calculate exact solutions with program FFAdj-3G for average-sized bacterial genomes in reasonable runtime. Hence, we suggest a heuristic variant of program FFAdj-3G, which we call FFAdj-3G-H in the following. Given a gene similarity graph $B_\circ = (G,H,I,E)$ of three genomes G, H, and I, FFAdj-3G-H solves problem FF-Adjacencies for all three genomes pairs (G,H), (G,I), (H,I) exactly, using program FFAdj-2G. Then, the heuristic integrates the resulting matchings $\mathcal{M}_{GH}$, $\mathcal{M}_{GI}$, $\mathcal{M}_{HI}$ into a gene similarity graph $B'_\circ = (G,H,I,\mathcal{M}_{GH} \cup \mathcal{M}_{GI} \cup \mathcal{M}_{HI}) \subseteq B_\circ$. Connected components of $B'_\circ$, that are not valid components of a partial 3-matching, are subsequently *enriched* by adding incident edges of their corresponding genes in $B_\circ$. In other words, let C be a set of genes corresponding to a component in $B'_\circ$ that is not a valid component of a partial 3-matching, then the set of edges $\{\{u,v\} \in E \mid v \in C\}$ is added to gene similarity graph $B'_\circ$. Heuristic FFAdj-3G-H then applies program FFAdj-3G on $B'_\circ$ to obtain a feasible (possibly suboptimal) solution to problem FF-Adjacencies of genomes G, H, and I. Lastly, the homology assignment derived from such a solution is then used as input to Tannier *et al.*'s maximum weighted matching-based approach [106] (Section 4.1) to obtain a valid median genome of genomes G, H, and I.

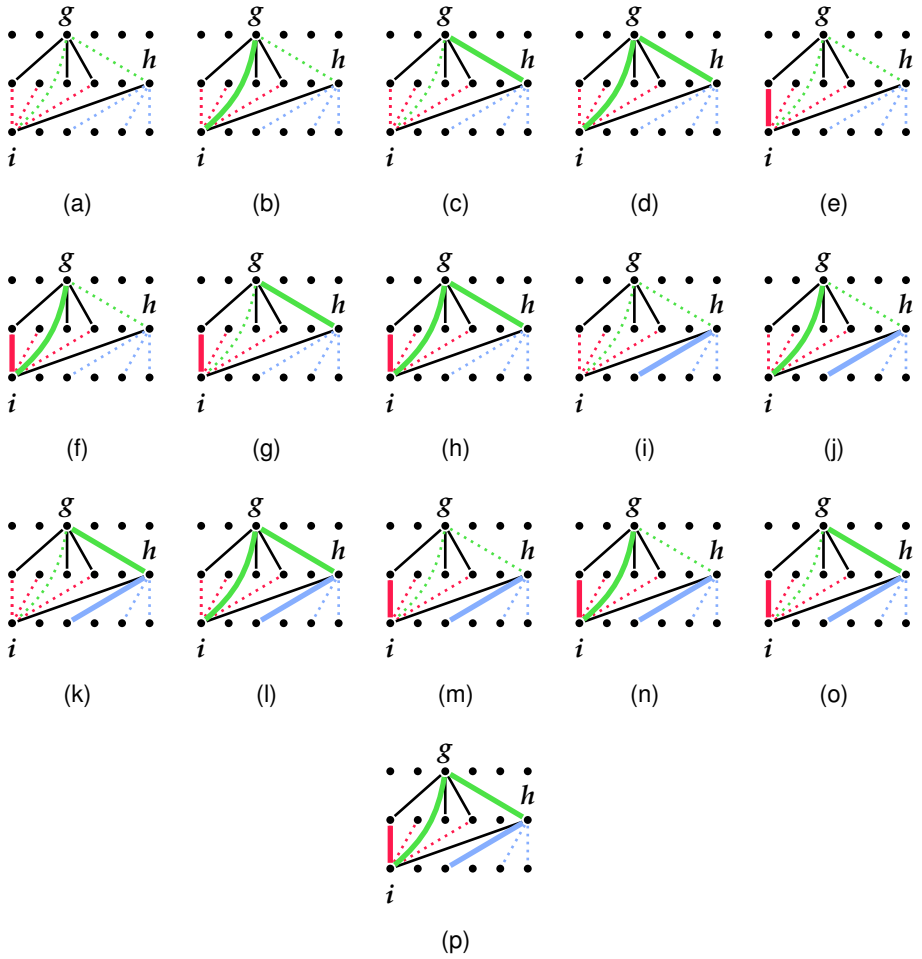


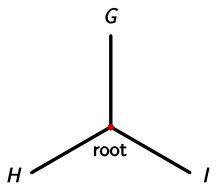
Figure 4.6: The effect of Constraint (C.03) on combinations of saturated edges. Parts (a) – (p) visualize all 16 possibilities that are valid under Constraint (C.01). The parts show how edges incident to genes i and h are effected by the first case of Constraint (C.03) that acts on edges $\{g,h\}$ and $\{g,i\}$ (green lines). Saturated edges are indicated by thick continuous lines, unsaturated edges by dashed lines. Continuous black lines are not considered by the constraint and can be either saturated or unsaturated. Only combinations shown in Parts (h), (l), and (p) violate constraint (C.02).

threshold of 10^{-5} and disabled query sequence filtering and used CPLEX to solve programs `FF-Median` and `FFAdj-3G`.

We allowed CPLEX to use up to 16 CPU cores and 16 GB of working memory. If the computation exceeded two hours, the calculations were stopped and a best feasible suboptimal solution of CPLEX's current solution pool was reported. Note that the computation of exact solutions to problem FF-Adjacencies for three genomes using `FFAdj-3G` is computationally infeasible even in small genomic datasets, hence we did not include program `FFAdj-3G` in subsequent analysis.

4.7.1 Simulations

We simulated genome evolution of three genomes diverging from a common ancestral sequence using the simulation framework ALF [33] with the same parameters (shown in Table 3.1) as in the simulations discussed in Section 3.9.1. Thereby, we used the following simple phylogeny as template to generate extant sequences from a supplied root genome:



In doing so, the genomic sequence of an *E. coli* K-12 strain, truncated to its first 1000 protein coding genes, was used as root genome. We then generated seven genomic datasets, thereby stretching the (equidistant) pairwise distances between extant genomes of the phylogeny from 10 to 130 PAM. Details about the evolutionary modifications in the generated datasets are shown in Table 4.1.

We subsequently calculated the statistical quantities described in Section 3.9.3 for results obtained by programs `FF-Median` and `FFAdj-3G-H` in all seven simulated datasets. However, this time we considered an ortholog assignment as true positive only if the two orthologous genes were positional homologs. Recall that in a duplication event, one gene gets copied into a new location, whereas the original copy remains unchanged. Two genes are positional homologs if both evolved directly from a common ancestral gene and did not emerge as duplicates in any duplication event along their connecting evolutionary paths. Identifying positional homologs is a crucial first step in reconstructing true medians through family-free analysis. Figures 4.7 (a) and (b) visualize precision and recall of positional ortholog assignments in solutions of programs `FF-Median` and `FFAdj-3G-H` in the seven simulated datasets. Due to the naturally high number of true negatives, the accuracy of both methods in all datasets was either 1 or very close to 1.

PAM	Genome	Inversions	Transpositions	Duplications	Losses
	G	10	8	41	30
10	*H*	8	11	33	27
	I	8	13	44	38
	G	28	19	73	102
30	*H*	25	25	59	102
	I	33	34	87	72
	G	44	56	162	128
50	*H*	47	41	128	161
	I	45	44	143	115
	G	79	85	225	181
70	*H*	74	75	223	178
	I	48	76	251	185
	G	87	79	267	321
90	*H*	85	95	278	226
	I	98	99	264	254
	G	108	124	317	356
110	*H*	110	112	335	313
	I	101	109	347	360
	G	103	144	422	376
130	*H*	131	142	417	398
	I	127	135	379	396

Table 4.1: Benchmark data comprising seven genomic datasets generated by ALF [33].

We further analyzed the performance of both programs by counting the number of recovered adjacencies in their reconstructed medians, i.e., the number of true positive adjacencies in comparison to the root genome. The outcome of this analysis is visualized in Figure 4.7 (c). For datasets with evolutionary distances 10 and 30 PAM, it was impossible for program `FF-Median` to obtain exact solutions within two hours of computation. In those cases we used suboptimal solutions from CPLEX's solution pool at the time of termination for calculating precision, recall, and number of recovered adjacencies shown in the diagrams of Figure 4.7.

Our experiments on the seven simulated datasets reveal a superior performance of program `FFAdj-3G-H`. In particular, the program outperformed program `FF-Median` in the number of true adjacencies recovered by its reconstructed medians. We point out again that suboptimal solutions were used in the evaluation of datasets with pairwise distances 10 and 30 PAM, hence their results should be treated with caution, as the true performance of program `FF-Median` (measured on exact solutions) could be much better here. Nevertheless, the superior performance of program `FFAdj-3G-H` — especially in datasets with medium large evolu-

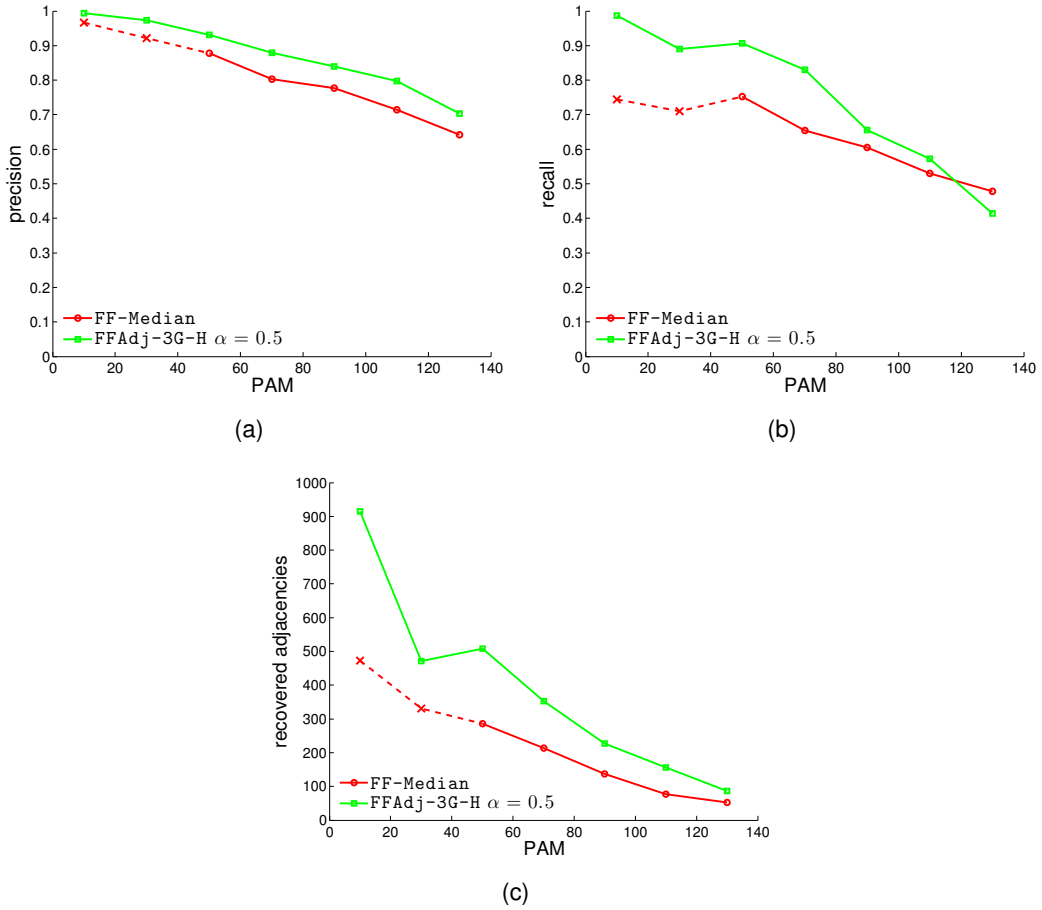


Figure 4.7: (a) Precision and (b) recall of positional orthologs obtained by `FF-Median` and `FFAdj-3G-H` with $\alpha = 0.5$. Part (c) visualizes number of recovered adjacencies in medians obtained by the two programs in the seven simulated datasets, out of 1000 true ancestral adjacencies. The red dashed lines indicate results obtained from a suboptimal solution of program `FF-Median`.

tionary distances — can be explained by the method's ability to tolerate events of gene duplication and loss. Yet, this advantage diminishes in phylogenies with large evolutionary distances, where large numbers of rearrangements and gene duplications/losses degrade the conserved gene order of extant genomes. In that case, our experiments show that the recall rate of identifying positional homologs eventually drops below that of program `FF-Median`.

4.7.2 Experiments on a biological dataset

Recently, an over 650 years old *Yersinia pestis* strain has been sequenced by Bos *et al.* [24]. The ancient γ-proteobacterial strain, known as *black death agent* of the mid-

Species/strain name	Accession No.	Size (bp)	#Genes
Yersinia pestis CO92	AL590842.1	4,653,728	4264
Yersinia pestis KIM 10	AE009952.1	4,600,755	4090
Yersinia pestis biovar Microtus str 91001	AE017042.1	4,595,065	3944

Table 4.2: The genomic dataset of three *Yersinia pestis* strains used in our reconstruction of the genome order of the black death.

dle ages, is considered to be the common ancestor of previously sequenced *Yersinia pestis* strains. The genome assembly of the ancient strain proposed by Bos *et al.* has been refined by Rajaraman *et al.* [87] using additional information of the genome structure of 11 extant *Yersinia pestis* and *Yersinia pseudotuberculosis* strains. In doing so, Rajaraman *et al.* aligned the 11 extant genomes against the proposed genome sequence of the ancient *Yersinia pestis* strain, henceforth denoted as *black death*. Based on the whole genome alignment, all genomes were partitioned into 9055 sequences of non-overlapping homologous markers, occurring in one or several copies (duplicates) in the ancient genome sequence. These homologous markers where then subject to a gene order study that gave rise to an improved genome assembly of the black death.

In the following, we aim to reconstruct the sequence of protein coding genes in the main chromosome of the black death by means of the order of protein coding genes in the main chromosomes of three extant *Yersinia pestis* strains. To this end, we will reconstruct a median genome using program `FFAdj-3G-H` and subsequently compare the results to those of Rajaraman *et al.* [87]. We chose the genome sequences of *Yersinia pestis* KIM 10, *Yersinia pestis* CO92, and *Yersinia pestis* biovar Microtus str. 91001, whose common ancestor is the *black death*. An overview on the genomic dataset is shown in Table 4.2. The phylogeny of the three *Yersinia pestis* strains in context of other currently sequenced strains is depicted in Figure 4.8.

We then computed pairwise similarities between genes using RRBS, applying the same procedure as previously used for simulated genome sequences. The median obtained by program `FFAdj-3G-H` corresponds to a single circular chromosome. All except for three adjacencies were supported by one, two, or three extant genomes.

The dataset provided by Rajaraman *et al.* [87] includes a mapping between markers from the ancient genomic sequence of the black death agent and locations in genomic sequences of extant *Yersinia pestis* strains. The lengths of markers range from 110bp to nearly 5000bp, thereby most ($> 80\%$) of the markers have lengths smaller than 500bp. The length distribution of markers is visualized in Figure 4.9 (a). We used the mapping of Rajaraman *et al.* to relate genes from genomes of *Yersinia pestis* C092, *Yersinia pestis* KIM 10, and *Yersinia pestis* biovar Microtus str. 91001 to markers. However, we found markers spreading over multiple genes as well as genes that cover several markers, as shown by the two distributions visualized in

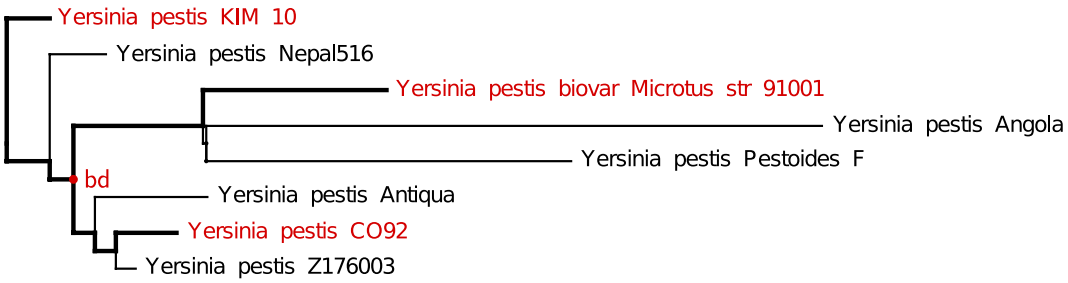


Figure 4.8: Phylogeny of sequenced *Yersinia pestis* strains. The highlighted subtree is subject of the herein described analysis (bd = black death).

Figures 4.9 (b) and (c), respectively. To complicate matters even further, in more than 750 cases markers were assigned unequal numbers of genes in extant genomes.

We subsequently assigned each marker to one or more genes, which resulted in 8532 out of 9055 markers assigned to at least one gene. We then evaluated the gene family assignment derived from the median obtained by `FFAdj-3G-H`. To this end, a marker was identified as conflicting with the gene family assignment, if the gene families of extant genes associated with the same marker were neither a superset nor a subset of each other. Note that this rule accounts for the over 750 cases in which markers were assigned unequal numbers of genes in extant genomes. The analysis resulted in the detection of 67 conflicts, i.e., 67 out of 9055 markers contained misassigned gene families in the median obtained by program `FFAdj-3G-H`.

We then studied the agreement of adjacencies in the median reconstructed by program `FFAdj-3G-H` with adjacencies in the assembled genome of Rajaraman *et al.* [87]. In doing so, we discarded 88 out of 3957 median adjacencies that either contained misassigned gene families or were not assigned to any markers. We then defined an adjacency of the reconstructed median to be *supported by* an adjacency of markers, if either (i) its corresponding genes were assigned to the same marker or (ii) all markers assigned to its corresponding genes were consecutive in the genome assembled by Rajaraman *et al.*. The evaluation of 3869 adjacencies in the reconstructed median identified 3850 supported adjacencies.

4.7.3 Discussion

In this chapter, we studied models and algorithms to construct a family-free median from three extant genomes. We introduced problem FF-Median, which is a family-free generalization of the well-known mixed multichromosomal breakpoint median of three genomes. We then studied the complexity of problem FF-Median. In doing so, we reduced instances of the weighted independent set problem to instances of problem FF-Median, thereby proving NP-hardness of the latter. We then discussed a 0-1 linear program for its exact solution.

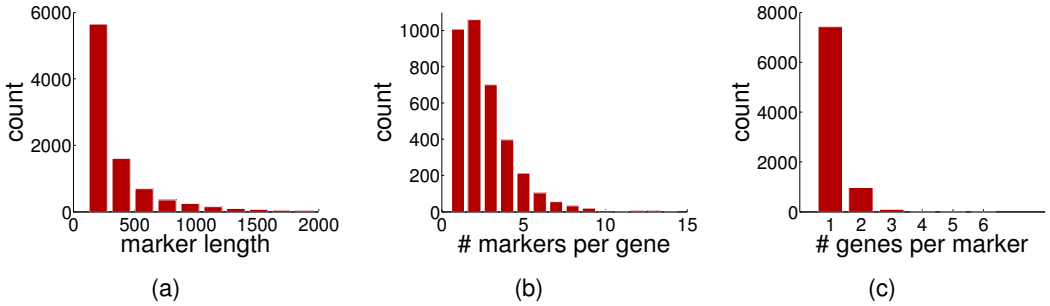


Figure 4.9: Part (a) visualizes a histogram of the distribution of marker lengths in the assembled genome sequence of Rajaraman *et al.* [87]. The diagrams to the right show histograms of (b) markers associated to the same gene and of (c) genes associated with the same marker, respectively.

Whereas our model of family-free adjacencies, presented in the previous chapter, can tolerate effects of gene family evolution in the chromosomal gene order, our family-free median model can only resolve certain cases of gene duplication. It is generally susceptible to gene losses that occurred along the evolutionary paths between the three extant genomes that are subject to analysis and their common ancestor. However, there is no straightforward definition of a family-free median model that tolerates events of gene family evolution, yet at the same time facilitates the calculation of exact solutions within reasonable time. Therefore, we devised with algorithm `FFAdj-3G-H` a heuristic approach to obtain family-free medians that is able to tolerate the effects of gene family evolution. Our method is based on problem FF-Adjacencies for three genomes, which was introduced in the previous chapter. Further, algorithm `FFAdj-3G-H` relies on Tannier *et al.*'s algorithm [106] to obtain a median gene order.

The importance of accounting for events of gene duplication and loss in family-free analysis were shown in subsequently performed experiments on simulated datasets: `FFAdj-3G-H` performed considerably better than `FF-Median` in identifying positional orthologs and in reconstructing the true gene order of the median.

Lastly, we demonstrate the applicability of algorithm `FFAdj-3G-H` on biological datasets by reconstructing the gene order of protein coding genes of the black death from genomes of three extant *Yersinia pestis* strains. The four genomes are separated by only 650 years of evolution. We compare our results to those of Rajaraman *et al.* [87]. The outcome of the analysis is encouraging: The median reconstructed by `FFAdj-3G-H` shows reasonable similarity to the genome structure proposed by Rajaraman *et al.*, although the latter used genomic markers for reconstruction, which were directly obtained from paleogenomic sequences of the black death.

Chapter 5

Family-free synteny

In the previous two chapters we described family-free models based on *adjacencies*. An adjacency is a simple proximity relation between two genes belonging to the same chromosomal sequence. However, gene orders become increasingly scrambled over longer evolutionary periods of time. When comparing two genomes that are distantly related, gene order analysis based on identifying pairs of conserved adjacencies may no longer be feasible. Yet, relaxed constraints of gene order conservation are still able to capture weaker, but nonetheless existing remnants of common ancestral gene order. In this chapter, we will study a relaxed proximity relation that allows us to identify conserved regions in two genomes based on the concept of *common intervals*. We present a practical approach that does not reconstruct one-to-one orthology assignments between genes. This simplification allows us to obtain fast, exact algorithms with polynomial running times. We subsequently evaluate our models and algorithms on a dataset of 93 bacterial genomes and compare its performance with that of a gene family-based method developed by Jahn [57]. The herein presented work is published in [26] and [37].

5.1 Generalized adjacencies

In Section 3.2 we presented problem FF-Adjacencies, which does not allow genes located in-between conserved adjacencies. We now describe a parameterized model of generalized adjacencies that does not only tolerate orthology assignments of genes in-between conserved adjacencies, but goes one step further, by also allowing crossing pairs of "conserved adjacencies", which we call *conserved θ-adjacencies*: Two gene extremities g_1^a and g_2^b in a genome G form a θ-adjacency if at most $\theta - 1$ genes lie between them. Two pairs of θ-adjacencies $\{g_1^a, g_2^b\}$ in a genome G and $\{h_1^a, h_2^b\}$ in a genome H form a *conserved θ-adjacency* if their corresponding genes are similar, i.e., $\sigma(g_1, h_1) > 0$ and $\sigma(g_2, h_2) > 0$. This leads to the following optimization problem:

Problem 7 (θ-Adjacencies) *Given two genomes G, H, $\alpha \in [0,1]$, and $\theta \in \mathbb{N}_{>0}$, find a matching $\mathcal{M}$ in gene similarity graph $B_\circ$ of G and H such that the following formula is maximized:*

$$\mathcal{F}_\alpha^\theta(\mathcal{M}) = \alpha \cdot adj^\theta(\mathcal{M}) \; + \; (1-\alpha) \cdot edg(\mathcal{M}) \,, \tag{5.1}$$

where

$$adj^\theta(\mathcal{M}) = \sum_{\substack{\{\{g_1,h_1\},\{g_2,h_2\}\} \subseteq \mathcal{M}, \\ \{g_1^a, g_2^b\} \in \mathcal{A}^\theta(G_\mathcal{M}), \\ \{h_1^a, h_2^b\} \in \mathcal{A}^\theta(H_\mathcal{M})}} s(g_1^a, g_2^b, h_1^a, h_2^b) \,,$$

and $\mathcal{A}^\theta(X)$ denotes the set of θ-adjacencies *of genome X, for which holds that in any adjacency $\{x_1^a, x_2^b\} \in \mathcal{A}^\theta(X)$, no more than $\theta - 1$ genes lie between x_1 and x_2 in genome X.*

θ-Adjacencies have been described previously in literature for gene family-based analysis [118] and are particularly useful for identifying *gene clusters*, which are small sets of genes that share an associated function and therefore remain locally preserved over longer periods of evolutionary time. Algorithm 1 can be easily adapted to find optimal solutions for Problem 7. Nevertheless, exact approaches become quickly computationally infeasible even for small θ. Moreover, the model itself can be criticized for its implicit handling of gene insertions and deletions, as well as its inability to account for unequal numbers of gene duplicates. Therefore, we will now study a broader definition of synteny and derive a family-free model that does not exhibit the described disadvantages of θ-adjacencies.

5.2 Synteny and gene clusters

Broader notions of conserved gene order ignore gene orientation and tolerate local genome rearrangements. In the subsequent analysis, we make use of a genome model in which telomeres are not modeled explicitly. Recently, Ghiurcuta and Moret concluded in [46] the following definition of syntenic blocks, which is compatible with many other commonly used definitions in literature. We call a set of genes $A \subseteq \mathcal{C}(G)$ of genome G a *block*, if all genes in A are contiguously connected to each other by adjacency set $\{\{g_1^a, g_2^b\} \in \mathcal{A}(G) \mid \{g_1, g_2\} \subseteq A\}$:

Definition 10 (syntenic blocks [46]) *Given two genomes G and H, and homology assignment $\mathcal{H}$, two sets of genes $A \subseteq \mathcal{C}(G)$ and $B \subseteq \mathcal{C}(G)$ are called* syntenic *if and only if A and B are blocks, and for each gene $g \in A$ there exists a homology assignment $\{g, h\} \in \mathcal{H}$ such that $h \in B$, and for each gene $h' \in B$ there exists a homology assignment $\{g', h'\}$ such that $g' \in A$.*

We note that Ghiurcuta and Moret assume that each gene in $\mathcal{C}(G)$ and $\mathcal{C}(H)$ has at least one homology statement in $\mathcal{H}$. This condition can be established in a straight forward manner by obtaining subgenomes $G' \subseteq G$ and $H' \subseteq H$ that only

contain genes with homology statements, on which the analysis of syntenic blocks is subsequently performed.

If $\mathcal{H}$ is in addition a gene family assignment, the definition of syntenic blocks is equivalent to that of *common intervals in strings*: Assigning a unique identifier to each gene family, a chromosome can be represented by a string drawn from the set of gene family identifiers. Two intervals from two strings are called *common intervals* if the sets of characters of their corresponding substrings are identical [4, 35]. In case these strings represent chromosomes, two intervals are common intervals if and only if their corresponding gene sets are syntenic blocks.

Gene clusters are sets of genes on a chromosome that stay in close neighborhood to each other over a longer period of evolutionary time due to mutual functional relationships. Most prominent gene clusters are *operons*, which are gene sets that are co-transcribed, and occur most prevalently in prokaryotes and fungi. From a theoretical and technical point of view, there is little difference between detecting remnants of ancestral gene order and identifying gene clusters. In fact, common intervals are frequently used in comparative gene cluster studies [37, 57, 86, 99]. Moreover, many known gene clusters that occur in multiple species are syntenic blocks according to Definition 10. Yet, state-of-the-art gene cluster detection algorithms tolerate limited numbers of inserted and deleted genes in blocks corresponding to gene cluster occurrences [23, 57, 86]. A pair of intervals in two strings, whose character sets are not identical, yet intersect substantially, are called *approximate common intervals* [5, 23]. Such interval pairs are also useful in detecting true synteny, considering that Definition 10 is a theoretical construct, i.e., one may have reason to claim two blocks syntenic despite few genes exhibit no homologous counterparts within the blocks, but to other genes outside. Furthermore, in case the homology assignment is an incomplete gene family assignment, false negatives can give rise to synteny violations in otherwise syntenic blocks. In the following we discuss family-free models and algorithms to identify syntenic blocks and gene clusters based on the concepts of common intervals and approximate common intervals.

5.3 Family-free syntenic blocks

5.3.1 A naïve approach

In [26] we suggested a naïve family-free approach based on problem FF-Adjacencies to detect syntenic blocks for two genomes G and H. Any pair of blocks (A, B), $A \subseteq \mathcal{C}(G)$ and $B \subseteq \mathcal{C}(H)$ can be syntenic. Therefore we build for each (A, B) a maximum weighted bipartite matching $\mathcal{M}$ between the gene sets of A and B. This is equivalent to solving problem FF-Adjacencies with $\alpha = 0$ for subgenomes corresponding to A and B.

An unmatched gene in A and B is either a duplicate occurrence if it is incident to an unchosen edge within the interval pair, or an inserted gene, if there are no

incident edges or all of them point to genes outside the interval pair. The obtained matching score $\mathcal{F}_0(\mathcal{M})$ needs to be corrected for the number of genes occurring in the intervals. Otherwise, the largest score is obtained for (G, H), the interval pair defined by the complete genomes. Simply normalizing $\mathcal{F}_0(\mathcal{M})$ by the length of A and B is also not advisable, as it causes the best-scoring syntenic blocks to be of length one, i.e., the best-scoring pair of genes. Instead, a trade-off between matching score and interval length needs to be defined. The corrected score can then be used to decide whether an interval pair should pass for a conserved segment or not.

Recall that a single maximal weight matching $\mathcal{M}$ can be computed in $\mathcal{O}(|A| \cdot |B| \cdot \sqrt{|A| + |B|})$ time [39]. However, already for two genomes there are $\mathcal{O}(|G|^2 \cdot |H|^2)$ interval combinations that need to be tested. One order of magnitude is saved if neither duplicate genes nor gene insertions and deletions are allowed. In this case, only intervals of the same size need to be paired. The maximum weight matching can be accelerated by performing incremental updates of previously computed solutions, which can be done in $\mathcal{O}(|A| \cdot |B|)$ time [108]. Nevertheless, the computational complexity of this problem remains of high polynomial order, rendering it computationally impractical.

5.3.2 A practical approach

The remainder of this chapter is largely based on [37]. In an effort to develop an approach that can be used in practice, we propose a family-free synteny model that avoids the costly computation of matchings. To this end, we make the bold simplifying assumption that every non-zero similarity, be it ever so small, between genes of two genomes can give rise to a true homology. In doing so, we reduce the problem of family-free synteny detection to that of finding syntenic blocks as stated in Definition 10. We construct a tentative homology assignment $\mathcal{H}_\sigma$ in which each pair of genes with non-zero similarity corresponds to a homology relation, i.e., $\mathcal{H}_\sigma \equiv \{(g, h) \in \Sigma \mid \sigma(g, h) > 0\}$. We call syntenic blocks obtained under homology assignment $\mathcal{H}_\sigma$ *family-free syntenic blocks*.

In contrast to previously presented family-free approaches, homology assignments are not explicitly determined. Yet, this model presumes there exists a true homology assignment which renders a pair of blocks, identified through subsequent analysis, syntenic. This, of course, is only true if the employed similarity measure produces no or a negligible amount of false positives. Nevertheless, there are several reasons why we believe this model is able to obtain good results in practice (and we demonstrate this in our subsequent evaluation): first and foremost, the synteny constraint itself acts as filter for false positives that are out of bounds of family-free syntenic blocks; second, one can always remove weak edges in a preprocessing step prior to a synteny analysis; lastly, we suggest a scoring scheme for family-free syntenic blocks that incorporates similarities between genes. The subsequent evaluation

of our model described in Section 5.9 shows that family-free syntenic blocks with high score are more likely to give rise to true synteny than those of low score.

Family-free syntenic blocks differ in size and, most substantially, in similarities between genes. Therefore, we introduce a simple scoring scheme for gene sets, by which all family-free syntenic blocks of two genomes can be ranked. The scoring scheme takes into account the number and the similarities of the contained genes. We define a score function μ_{XY} over a gene x in genome X and a gene set $C \subseteq \mathcal{C}(Y)$ of genome Y as

$$\mu_{XY}(x, C) = \begin{cases} \dfrac{\max\limits_{y' \in C} \sigma(x, y')}{\max\limits_{y \in \mathcal{C}(Y)} \sigma(x, y)} & \text{if } \max\limits_{y \in \mathcal{C}(Y)} \sigma(x, y) > 0 \\ 0 & \text{otherwise} \end{cases}$$

so that μ_{XY} takes values between 0 and 1, being 1 if a gene with highest similarity to i is contained in gene set C. The overall score of two gene sets (A, B), $A \subseteq \mathcal{C}(G)$ and $B \subseteq \mathcal{C}(H)$ of genomes G and H is then

$$syn_{GH}((A, B)) = \sum_{g \in \mathcal{C}(A)} \mu_{GH}(g, B) + \sum_{h \in \mathcal{C}(B)} \mu_{HG}(h, A)\,. \tag{5.2}$$

In Section 5.2 we mentioned that syntenic blocks are equivalent to common intervals under a gene family assignment. For general homology assignments, where transitivity is not guaranteed, syntenic blocks are equivalent to *weak common intervals in indeterminate strings*, which will be discussed in detail in the subsequent section. Indeterminate strings, also known as degenerate strings, are a class of strings that have at every position a non-empty set of characters [51]. Assigning each homology pair in $\mathcal{H}$ a unique identifier, we assume in the subsequent analysis that indeterminate strings drawn from the alphabet of homology pair identifiers correspond to chromosomes.

We propose a further use case of our model: if one or several gene family prediction methods return gene family assignments $\mathcal{H}_1, \ldots, \mathcal{H}_n$ that are in conflict with each other, our model can be used to predict syntenic blocks in the union of all $\mathcal{H}_1 \cup \cdots \cup \mathcal{H}_n$. The identified syntenic blocks can be subsequently used to remove homology pairs that are not supported by synteny.

5.4 Common intervals in indeterminate strings

Common intervals were initially introduced on permutations [109] and subsequently extended to strings [4, 35]. In the following we generalize the concept of common intervals to indeterminate strings. Since it is applicable in a much broader context, we will formally state and discuss common intervals in indeterminate strings by temporarily omitting the context of family-free synteny detection.

For an indeterminate string S with n index positions must hold that for every i, $1 \leq i \leq n$, $S[i] \subseteq \Sigma$ and $S[i] \neq \emptyset$, where $S[i]$ denotes the character set associated with the i-th position in S. In the special case where every position of indeterminate string S holds a singleton set, S is equivalent to an ordinary string. We denote the *length* of an indeterminate string S with n index positions by $|S| \equiv n$ and its *cardinality*, i.e., the number of *all* elements in S, by $\|S\| \equiv \sum_{i=1}^{n} |S[i]|$. Two positions a and b, $1 \leq a \leq b \leq |S|$, induce the indeterminate *substring* $S[a,b] \equiv S[a]\, S[a+1] \ldots S[b]$. To distinguish intervals in different indeterminate strings, we indicate the affiliation of an interval $[i,j]$ to indeterminate string S by the subscript notation $[i,j]_S$.

Example 6 *$S = \{a,d,g\}\,\{c\}\,\{a,d\}\,\{e,f\}\,\{b\}\,\{c,g\}$ is an indeterminate string of length $|S| = 6$ and cardinality $\|S\| = 11$ over alphabet $\Sigma = \{a,b,c,d,e,f,g\}$. The third element of S is given by character set $S[3] = \{a,d\}$. Interval $[2,4]$ induces the substring $S[2,4] = \{c\}\,\{a,d\}\,\{e,f\}$.*

Obviously, indeterminate strings are reduced forms of *regular expressions*. That is, every indeterminate string S corresponds to a regular expression in which every set $S[i]$, $1 \leq i \leq n$ is represented by an alternation of its contained characters.

Example 6 (continued) *Indeterminate string $S = \{a,d,g\}\,\{c\}\,\{a,d\}\,\{e,f\}\,\{b\}\,\{c,g\}$ gives rise to the regular expression $(a|d|g)c(a|d)(e|f)b(c|g)$.*

In the past years, many classic string problems have been transferred to indeterminate strings, such as pattern matching [52], finding repetitive structures [8], computing the cover [7], and longest common supersequences [55, 56].

The idea behind common intervals is to compare strings, or rather substrings, based on their character sets. The character set of an ordinary string S is defined as $\mathcal{C}(S) \equiv \{S[i] \mid 1 \leq i \leq |S|\}$. The equivalent concept on indeterminate strings is the following:

Definition 11 (character set) *The* character set *of an indeterminate string S of length n is denoted by $\mathcal{C}(S) \equiv \bigcup_{i=1}^{n} S[i]$.*

Note that the character sets $\mathcal{C}(S)$ and $\mathcal{C}(T)$ of two indeterminate strings S and T can be identical, yet no two positions between S and T may share the same character set. In two ordinary strings S and T over a finite alphabet Σ, two intervals, $[i,j]$ in S and $[k,l]$ in T, are called *common intervals* if $\mathcal{C}(S[i,j]) = \mathcal{C}(T[k,l])$. The analogon for indeterminate strings is:

Definition 12 (strict common intervals) *Given two indeterminate strings S and T, two intervals, $[i,j]$ in S and $[k,l]$ in T, are* strict common intervals *if and only if their character sets $\mathcal{C}(S[i,j])$ and $\mathcal{C}(T[k,l])$ are equal.*

A weaker definition based on the intersection relation between character sets is:

Definition 13 (weak common intervals) *Given two indeterminate strings S and T, two intervals, $[i,j]$ in S and $[k,l]$ in T, are* weak common intervals *with* common character set $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$ *if for each x, $i \leq x \leq j$, it holds that $C \cap S[x] \neq \emptyset$ and for each y, $k \leq y \leq l$, it holds that $C \cap T[y] \neq \emptyset$.*

In all our use cases, in particular for the discovery of family-free syntenic blocks, the concept of weak common intervals appears to be more appropriate. Nevertheless, since the generalization of common intervals to indeterminate strings is original, we will also outline an algorithm for the discovery of strict common intervals in Section 5.6.

Continuing a previous line of research initially proposed by Schmidt and Stoye in [99], we further extend strict and weak common intervals by allowing a limited number of insertions and deletions:

Definition 14 (approximate strict common intervals) *Given two indeterminate strings S and T and a threshold $\delta \in \mathbb{N}_0$, two intervals, $[i,j]$ in S and $[k,l]$ in T, are* approximate strict common intervals *with* common character set $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$ *if and only if*

$$\bigcup_{\substack{i \leq x \leq j, \\ S[x] \subseteq C}} S[x] = \bigcup_{\substack{k \leq y \leq l, \\ T[y] \subseteq C}} T[y]\,,$$

and the number of positions *that are not subset of C is limited by δ, i.e., $|\{x \mid i \leq x \leq j : S[x] \not\subseteq C\}| + |\{y \mid k \leq y \leq l : T[y] \not\subseteq C\}| \leq \delta$. These positions are called* indels.

Definition 15 (approximate weak common intervals) *Given two indeterminate strings S and T and a threshold $\delta \in \mathbb{N}_0$, two intervals, $[i,j]$ in S and $[k,l]$ in T, are* approximate weak common intervals *with* common character set $C = \mathcal{C}(S[i,j]) \cap \mathcal{C}(T[k,l])$ *if the number of* positions *with no intersection with C is limited by δ, i.e., $|\{x \mid i \leq x \leq j : S[x] \cap C = \emptyset\}| + |\{y \mid k \leq y \leq l : T[y] \cap C = \emptyset\}| \leq \delta$. These positions are called* indels.

Generally, algorithms for discovering common intervals of ordinary strings only report pairs of intervals that both are *maximal*, i.e., cannot be extended to the left or right without increasing their character set. The equivalent condition of *maximality* in indeterminate strings is as follows:

Definition 16 (maximal) *An interval $[i,j]$ in S is called* maximal *if (i) $i = 1$ or $S[i-1] \not\subseteq \mathcal{C}(S[i,j])$, and (ii) $j = |S|$ or $S[j+1] \not\subseteq \mathcal{C}(S[i,j])$.*

Observe that maximality cannot be combined with the weak common intervals property without omitting expedient interval pairs, as the subsequent example shows:

Example 7 *Given a weak common intervals pair $([i,j]_S, [k,l]_T)$ of indeterminate strings S and T. Let $S[i-1]$ be a subset of $\mathcal{C}(S[i,j])$, but not intersecting with $\mathcal{C}(T[k,l])$. Then $[i,j]_S$ is not maximal according to Definition 16, yet its extended interval $[i-1,j]_S$ is not a common intervals pair with $[k,l]_T$.*

In terms of weak common intervals pairs, one may consider to restrict the search space spanned over two indeterminate strings S and T to those pairs that cannot be extended with respect to their common character set. We introduce the following property derived from [57]:

Definition 17 (C-closed) *Given an indeterminate string S, an interval $[i,j]$, and a character set $C \subseteq \Sigma$, interval $[i,j]$ is* C-closed *if $S[i], S[j] \cap C \neq \emptyset$, and if $i = 1$ or $S[i-1] \cap C = \emptyset$, and if $j = |S|$ or $S[j+1] \cap C = \emptyset$.*

However, the number of interval pairs that are closed w.r.t. their common character set remains absurdly high even for simple structures. We give the following example:

Example 8 *Given two indeterminate strings $S = \{a\}\{b,c\}\{d,e\}\{f\}$ and $T = \{c\}\{a,b\}\{d,f\}\{e\}$. Then the following weak common intervals with interval lengths 2 or larger are closed w.r.t. their common character set:* $([1,2]_S, [1,2]_T)$, $([1,3]_S, [1,3]_T)$, $([1,3]_S, [1,4]_T)$, $([1,3]_S, [2,3]_T)$, $([1,3]_S, [2,4]_T)$, $([1,4]_S, [1,3]_T)$, $([1,4]_S, [1,4]_T)$, $([1,4]_S, [2,3]_T)$, $([1,4]_S, [2,4]_T)$, $([2,3]_S, [1,3]_T$, $([2,3]_S, [1,4]_T)$, $([2,3]_S, [2,3]_T)$, $([2,3]_S, [2,4]_T)$, $([2,4]_S, [1,3]_T)$, $([2,4]_S, [1,4]_T)$, $([2,4]_S, [2,3]_T)$, $([2,4]_S, [2,4]_T)$, $([3,4]_S, [3,4]_T)$.

A reasonable balance between omitting dispensable and including expedient weak common intervals is found by the subset of those that are *mutually-closed*, as defined as follows:

Definition 18 (mutually-closed) *Given a pair of intervals $([i,j]_S, [k,l]_T)$ of indeterminate strings S and T, $[i,j]_S$ and $[k,l]_T$ are* mutually-closed *if $[i,j]_S$ is $\mathcal{C}(T[k,l])$-closed and $[k,l]_T$ is $\mathcal{C}(S[i,j])$-closed.*

The number of mutually-closed weak common intervals in our example is low as expected:

Example 8 (continued) *The mutually-closed weak common intervals of lengths 2 or larger of indeterminate strings $S = \{a\}\{b,c\}\{d,e\}\{f\}$ and $T = \{c\}\{a,b\}\{d,f\}\{e\}$ are:* $([1,2]_S, [1,2]_T), ([1,4]_S, [1,4]_T), ([3,4]_S, [3,4]_T)$.

We consequently restrict the enumeration of weak common intervals and approximate weak common intervals to those that are mutually-closed. The maximal number of mutually-closed weak common intervals of two indeterminate strings S and

T of lengths n and m, respectively, is bounded by nm. This result follows from the fact that the number of intervals $[k,l]$ in T that are mutually-closed weak common intervals with any interval of fixed left bound i in S is bounded by m. Likewise, the maximal number of mutually-closed approximate weak common intervals of indeterminate strings S and T is bounded by $(\delta+1)^2nm$. Note that mutually-closed weak common intervals are a special subclass of mutually-closed approximate weak common intervals for $\delta = 0$.

We now describe algorithms to compute all mutually-closed weak common intervals, all maximal strict common intervals and all mutually-closed approximate weak common intervals of two indeterminate strings S of length n and T of length m. The efficient enumeration of approximate strict common intervals of two indeterminate strings remains as an open problem.

5.5 Discovering weak common intervals

We now describe the algorithm *Weak Common Intervals on Indeterminate Strings* (WCII). It solves the following problem:

Problem 8 *Given two indeterminate strings S and T, discover all mutually-closed weak common intervals of S and T.*

To tackle this problem we make use of the following constructs:

Definition 19 (index string) *Given an indeterminate string S of length n, $I_S \equiv \{1\}\{2\}\cdots\{n\}$ denotes the* index string *of S.*

Definition 20 (index mapping) *Given two indeterminate strings S and T of lengths n and m respectively, the* index mapping *of S onto T is given by $(T_S[y])_{y=1,\ldots,m}$, where*

$$T_S[y] = \begin{cases} \{x \mid x = 1,\ldots,n : S[x] \cap T[y] \neq \emptyset\} & \text{if } T[y] \cap \mathcal{C}(S) \neq \emptyset \\ \{\infty\} & \text{otherwise.} \end{cases}$$

Example 9 *Figure 5.1(a) visualizes the intersection of character sets of positions between indeterminate strings $S = \{g\}\{b,p\}\{x\}\{n,p\}\{d,o,s\}\{a,z\}\{e,n,w\}\{f\}\{l,v\}\{h,u,z\}\{j,r\}\{k\}\{m,q,y\}$ and $T = \{d\}\{g,b\}\{a\}\{p,s\}\{n\}\{a,b\}\{f,m,w\}\{e,w\}\{i\}\{q,y\}\{h\}\{c,r\}\{z\}$. Their corresponding index string I_S and index mapping T_S are shown in Figure 5.1 (b).*

Note that index strings and index mappings are again indeterminate strings, with indices representing characters drawn from the alphabet of natural numbers. The key idea of WCII arises from the following observation:

Observation 1 *Given two indeterminate strings S and T with index string I_S and index mapping T_S, two intervals $[i,j]$ in S and $[k,l]$ in T are weak common intervals if and only if $[i,j]_{I_S}$ and $[k,l]_{T_S}$ are weak common intervals.*

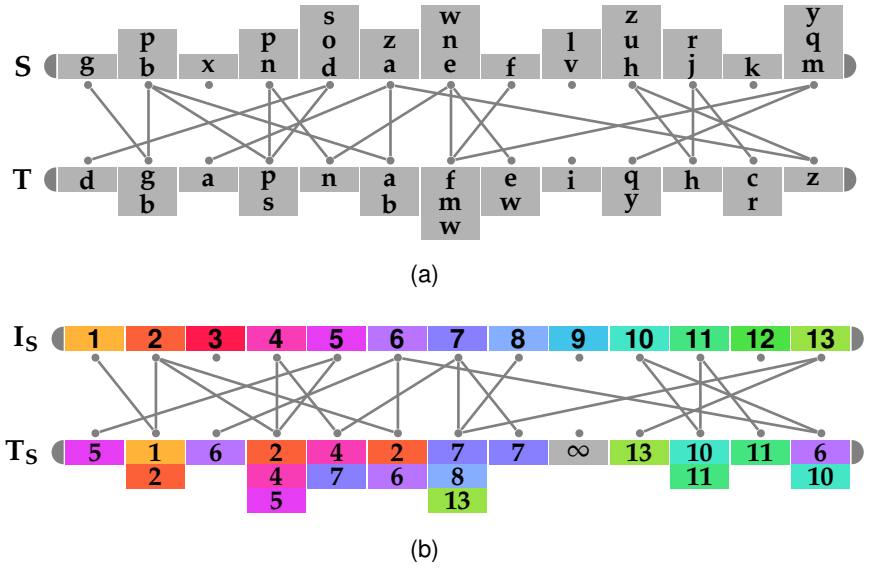


Figure 5.1: (a) Visualization of indeterminate strings S and T of Example 9 and (b) their corresponding index string I_S and index mapping T_S.

This equivalence holds because any two positions, x in S and y in T, intersect if and only if $I_S[x]$ and $T_S[y]$ intersect. Since it holds that $I_S[x] = \{x\}$ for all $x = 1, \ldots, n$, we simplify the notation of single character set $I_S[x]$ to just x and character set $\mathcal{C}(I_S[i,j])$ to just $[i,j]$. Note that character $c \in \mathcal{C}(I_S[i,j]) \subseteq \{1, \ldots, n\}$ serves subsequently both as character $c \in [i,j]$ as well as index in I_S.

WCII is an adaptation of Didier's Algorithm [35] of enumerating maximal common intervals in ordinary strings. Didier's strategy can be described as follows: The algorithm iterates over all positions i in I_S as possible left interval bounds. In each iteration all mutually-closed weak common interval pairs are reported that share the same left bound i in I_S. For each possible right bound $j \geq i$, the algorithm iterates through the set of positions in T_S that contain j in their character set. To this end, a table Pos is used, where $\mathrm{Pos}[j]$, $1 \leq j \leq n$, is a sorted list of positions in T_S containing character j. Each position $y \in \mathrm{Pos}[j]$ is associated with an interval $[k,l]_{T_S}$, $k \leq y \leq l$, called the *min-rank interval* of character j for position y. It is the largest interval around y for which every position in $[k,l]_{T_S}$ contains at least one character in $[i,j]$. Obviously, $[k,l]_{T_S}$ is $[i,j]$-closed. It remains to be tested if $[i,j]_{I_S}$ is closed w.r.t. $\mathcal{C}(T_S[k,l])$ and that every position in $[i,j]_{I_S}$ and $[k,l]_{T_S}$ contains a character from $C = \mathcal{C}(I_S[i,j]) \cap \mathcal{C}(T_S[k,l])$. To show the latter, it is sufficient to show that $[i,j] \subseteq \mathcal{C}(T_S[k,l])$, because the character set of each position in I_S corresponds to the single element set of its index. The details of both tests are explained after relevant data structures are introduced. If both conditions are satisfied, a mutually-closed weak common intervals pair is found and subsequently reported.

We follow two strategies of Didier's Algorithm, that improve the performance: precomputing *min-rank intervals* and following paths of *rank-nearest successors*.

In order to identify min-rank intervals, it is sufficient to observe the smallest character $c \geq i$ in each position of T_S. To this end, we make use of the following construct:

Definition 21 (*i*-reduced string) *Given index mapping T_S, $(T_S^i[y])_{y=1,\ldots,m}$ is the* i-reduced string of T_S of the ith iteration, *where* $T_S^i[y] = \min(\{c \mid c \in T_S[y] \cup \{\infty\} : c \geq i\})$.

Min-rank intervals in T_S^i are identical to *rank intervals* as initially defined by Didier *et al.* [35]. Interestingly, rank intervals in T_S^i correspond directly to min-rank intervals in T_S:

Lemma 8 *The set of min-rank intervals in T_S is identical to the set of rank intervals in T_S^i.*

Proof: Didier *et al.* [35] show that rank intervals in a string are nested and that their number is bounded by the length of the string.

Observe that for any position y in T_S^i the rank interval of character $j = T_S^i[y]$ is identical to the min-rank interval of j at position y in T_S. Let y be a position in T_S and $j \in T_S[y]$ such that $j > T_S^i[y]$. Further, let $[k,l]_{T_S}$ be the min-rank interval of j at $T_S[y]$, $j' = \max(\{c \mid c \in \mathcal{C}(T_S^i[k,l]) : c \leq j\})$, and $[k',l']_{T_S}$ be the min-rank interval of j' at its corresponding position in T_S. Because $j' \leq j$, it consequently holds that $[k',l']_{T_S} \subseteq [k,l]_{T_S}$. Now, according to the definition of min-rank intervals, $T_S^i[k'-1] > j'$, if such position exists. Since j' is the largest character in $T_S^i[k,l]$ that is smaller than or equal to j, it must also hold that $T_S^i[k'-1] > j$. The same argument holds for $T_S^i[l'+1]$ if such position exists, therefore $[k,l]_{T_S} = [k',l']_{T_S}$ is the min-rank interval of both characters j' and j. We conclude that all min-rank intervals for any character in T_S at iteration i are contained in the set of rank intervals of T_S^i. □

Consequently, all min-rank intervals in T_S in the ith iteration (i.e., for a fixed left bound i in I_S) can be precomputed in $\mathcal{O}(m)$ time using the algorithm given by Didier *et al.* [35]. They are stored in table Int. For a currently processed character j at position y in T_S, Int$[y]$ contains its corresponding min-rank interval. Unlike Didier's Algorithm, Int must be updated after each iteration such that all positions in Int accessed in the following $(j+1)$th iteration contain the corresponding min-rank intervals of character $j+1$. Details of the update step can be found in Section 5.5.1.

The second strategy of Didier's Algorithm for speeding up the computation is to continuously increase the right bound j in I_S while walking through positions and characters of T_S. In doing so, the algorithm jumps from a current position y that contains character j to its *rank-nearest successor*, which is the position y' containing character $j+1$ with the smallest *min-rank distance* to y as defined as follows:

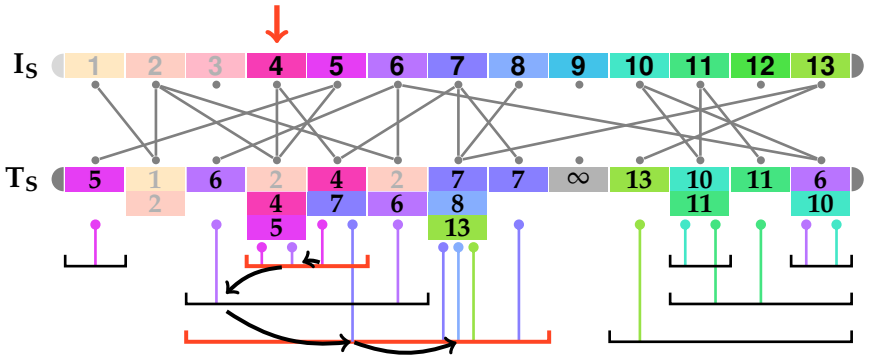


Figure 5.2: Visualization of all min-rank intervals and an exemplary path of rank-nearest successors for iteration $i = 4$ of Algorithm WCII applied on indeterminate strings of Example 9. On this path of rank-nearest successors, the algorithm finds mutually-closed weak common intervals pairs $([4,5]_S, [4,5]_T)$ and $([4,8]_S, [3,8]_T)$, which are highlighted in orange color.

Definition 22 (min-rank distance) *The* min-rank distance *of any two positions k and l in indeterminate string T_S for the ith iteration is given by:*

$$d^i_{T_S}(k,l) \equiv \max(\{T^i_S[p] \mid k \leq p \leq l\}) .$$

If several co-optimal positions are available, the tie is broken by choosing the leftmost one as rank-nearest successor. In case no position with character $j+1$ exists, or the smallest min-rank distance is infinite, j has no successor. For the ith iteration, all rank-nearest successors are precomputed and stored in table Succ, which is explained in more detail in Section 5.5.2.

Example 9 (continued) *Figure 5.2 visualizes for iteration $i = 4$ all min-rank intervals of T_S and one exemplary path of rank-nearest successors that leads to mutually-closed weak common intervals pairs $([4,5]_S, [4,5]_T)$ and $([4,8]_S, [3,8]_T)$.*

We now give an overview of Algorithm 5. To this end, apprehend that connecting characters larger than or equal to i at their corresponding positions in T_S with their rank-nearest successors through directed edges results in a forest of rooted trees. Nodes (across all trees) sharing the same character are said to reside on the same *level*. In lines 8-28 of Algorithm 5, the algorithm traverses along paths through this forest in a bottom-up procedure, from one level to the next, starting at those leaves with character i. Besides the currently visited nodes of the level, the algorithm keeps track of the *path bounds*, which are the outermost positions in T_S a path has visited thus far. The currently visited nodes of the paths and their corresponding path bounds are stored in a list labeled List. Only after all nodes of the same level j are processed, the algorithm follows all current paths to nodes of the next level $j+1$, thereby ensuring that each character in T_S is processed at most once. To this end, for

Algorithm 5 Algorithm WCII adapts the search strategy of Didier's Algorithm [35] for common intervals in strings to the computation of weak common intervals in indeterminate strings.

Input: Two indeterminate strings S and T
Output: All mutually-closed weak common intervals of S and T

1: Construct index mapping T_S and table Pos
2: **for** $i \leftarrow 1$ **to** $|S|$ **do**
// initialize List *with positions in* T_S *that contain character* i
3: Initialize empty list List
4: **for each** element y in Pos$[i]$ **do**
// tuple storing current position y *in path of rank-nearest successors, and current path bounds*
5: add $(y, [y, y])$ to List
6: **end for**
// main loop
7: $j \leftarrow i$
8: **while** List is not empty **do**
9: previous $\leftarrow [\infty, \infty]$
10: Initialize empty list List$'$
11: **for each** element $(y, [p_k, p_l])$ in List **do**
12: $[k, l] \leftarrow$ min-rank interval of character j around position y in T_S.
13: **if** $i - 1 \notin \mathcal{C}(T_S[k, l])$ **then**
// test if there are interval pairs to output
14: $y' \leftarrow$ rank-nearest successor of y if such exists, otherwise ∞
15: **if** $[p_k, p_l] \subseteq [k, l]$ **and** $[k, l] \neq$ previous **and** $y' \notin [k, l]$ **then**
16: **output** $([i, j], [k, l])$
17: previous $\leftarrow [k, l]$
18: **end if**
// compute the next level
19: **if** $y' \neq \infty$ **and** y is the leftmost index with shortest min-rank distance to y' among positions of the list having the same rank-nearest successor y' **then**
// update path bounds
20: $p'_k \leftarrow \min(p_k, y')$
21: $p'_l \leftarrow \max(p_l, y')$
22: add $(y', [p'_k, p'_l])$ to List$'$
23: **end if**
24: **end if**
25: **end for**
26: List $\leftarrow$ List$'$
// Increase right bound j
27: $j \leftarrow j + 1$
28: **end while**
29: **end for**

all positions containing character j that have the same rank-nearest successor y', the algorithm discontinues the paths of all but the leftmost one with shortest min-rank distance to y' (line 19). Traversing along paths of rank-nearest successors in WCII differs from Didier's Algorithm by the fact that a position in T_S may be visited by the same path several times on different levels.

For any given min-rank interval $[k,l]_{T_S}$ there cannot be more than one weak common intervals partner in I_S starting at position i. Therefore it is sufficient to track at least one path in each min-rank interval to find all mutually-maximal intervals of I_S and T_S. Positions in Pos are sorted, thus paths leading to the same weak common intervals pair appear adjacent to each other in List and the pair is reported only for the first (lines 15-17).

For each node in List, associated with character j and position y, the algorithm checks if the min-rank interval $[k,l]_{T_S}$ of j encloses the path bounds up to position y (see condition in line 15). If validated, a weak common intervals pair has been found, given by $([i,j]_{I_S},[k,l]_{T_S})$. To ensure mutual closedness, the pair is only reported if $i-1$ is not contained in the character set $\mathcal{C}(T_S[k,l])$ and the successor of y is not within the current bounds of its path (see conditions in lines 13 and 15). Checking for the former can be achieved in $\mathcal{O}(1)$ time after $\mathcal{O}(m)$ time preprocessing by performing a range minimum query on an array of size $\mathcal{O}(m)$ where each position containing character $i-1$ is assigned 0 and 1 otherwise.

The overall complexity of the algorithm can be summarized as follows: Table Pos can be constructed within $\mathcal{O}(\|T_S\|)$ time and requires $\mathcal{O}(\|T_S\|)$ space. Each position in I_S is regarded exactly once as left bound i for all weak common intervals that are reported in one iteration. Once T_S^i is computed for $i=1$ it can be updated using table Pos, taking overall $\mathcal{O}(\|T_S\|)$ time for all left bounds $i=1,\ldots,n$. Further, for each left bound the algorithm performs $\mathcal{O}(m)$ steps to precompute all min-rank intervals and $\mathcal{O}(\|T_S\|)$ steps to precompute all rank-nearest successors. The subsequent bottom-up procedure and the reporting of weak common intervals requires again $\mathcal{O}(\|T_S\|)$ time. Therefore we have:

Theorem 3 *Given two indeterminate strings S and T, Algorithm WCII finds all pairs of mutually-closed weak common intervals of S and T in $\mathcal{O}(|S|\cdot\|T_S\|)$ time.*

The following two subsections describe the prodedures of updating table Int and computing table Succ in detail.

5.5.1 Updating table Int

Didier *et al.* [35] give a linear time algorithm to identify all rank intervals in a string, that we use to precompute all (min-) rank intervals for characters in T_S^i. Their bounds are stored in table Int such that entry Int$[y]$ stores the bounds of the min-rank interval for character $T_S^i[y]$. Table Int must be updated when proceeding from one level to the next, so that Int$[y]$, always points to the min-rank interval of the currently processed character in $T_S[y]$.

Algorithm 6 updates table Int for the next level. In the first part of the algorithm the min-rank interval of each position y containing the currently processed character j is copied to the position of its rank-nearest successor y' if such exists and if

Algorithm 6 Update algorithm for table INT

Input: Tables INT, POS, and SUCC, min-rank string T_S^i, and current level j
Output: Table INT for level $j+1$

```
    // push rank intervals of j to successor positions
 1: for each element y in Pos[j] do
 2:     y' ← rank-nearest successor of y if exists, otherwise ∞
 3:     if y' ≠ ∞ and Int[y'] ⊂ Int[y] then
 4:         Int[y'] ← Int[y]
 5:     end if
 6: end for
    // Update rank intervals in the neighborhood of positions y for which T_S^i[y] = j+1
 7: Initialize empty queue Queue
 8: Previous ← −∞
 9: for each element y in Pos[j+1] do
10:     if T_S^i[y] ≠ j+1 then
11:         if Previous ≠ −∞ and d^i_{T_S}(y, Previous) ≤ j+1 then
12:             Int[y] ← Int[Previous]
13:         else
14:             Push y onto Queue
15:         end if
16:     else
17:         while Queue is not empty do
18:             Pop y' from Queue
19:             if d^i_{T_S}(y, y') ≤ j+1 then
20:                 Int[y'] ← Int[y]
21:             end if
22:         end while
23:         Previous ← y
24:     end if
25: end for
```

$\text{INT}[y'] \subset \text{INT}[y]$. Now, in the proof of Lemma 8 we showed that for any character j at any position y in T_S for which $T_S^i[y] \neq j$, there exists some $j' \leq j$ at some position y' such that $T_S^i[y'] = j'$ and the min-rank intervals of j and j' are identical. Let us for now assume that $j' < j$. Then j' is visited prior to j, therefore the min-rank interval of j' is passed from j' to a position containing $j'+1$, to $j'+2$, etc., eventually reaching character j on position y. Note that such a path must exist if $([i,j]_{I_S}, [k,l]_{T_S})$ are weak common intervals.

The case where $j' = j$ is treated in lines 7 onwards. To this end we iterate once through all positions $\text{Pos}[j+1]$ of the character $j+1$ that will be processed next. Whenever the algorithm finds a position y such that $T_S^i[y] = j+1$, previous and subsequent positions in $\text{Pos}[j+1]$ that reside in the corresponding min-rank interval are updated to $\text{INT}[y]$.

5.5.2 Computing table Succ

In order to identify weak common intervals pair $([i,j]_{I_S}, [k,l]_{T_S})$, the algorithm follows a path of rank-nearest successors from i to j within the bounds of min-rank interval $[k,l]_{T_S}$ of character j at a position y, $k \leq y \leq l$. By doing so, the algorithm avoids the costly computation of determining the character set in an interval, which would be otherwise necessary in verifying that all characters in $i, \ldots, j$ are contained in a min-rank interval $[k,l]_{T_S}$.

We can adopt Remark 3 from Didier *et al.* [35]:

Observation 2 *For each min-rank interval $[k,l]_{T_S}$ holds that the min-rank distance of any two positions inside the interval is strictly smaller than any min-rank distance of a position inside $[k,l]_{T_S}$ to a position outside the interval.*

Observation 2 ensures us that a path of rank-nearest successors from i to j along positions in T_S will always stay within bounds of min-rank interval $[k,l]_{T_S}$ of j for position y as long as $([i,j]_{I_S}, [k,l]_{T_S})$ are weak common intervals.

Let D_{T_S} be the table that stores indeterminate string T_S. In each iteration, the rank-nearest successors of all characters in T_S are precomputed and stored in a table Succ, which is an analogous table to D_{T_S}. That is, the rank-nearest successor of character c at position y in T_S is stored at the same index position in Succ as character c in D_{T_S}. Thus, the size of Succ is $\mathcal{O}(\|T_S\|)$.

Since Succ is traversed parallel to D_{T_S}, the look-up of the rank-nearest successor of a character c at position y in T_S can be performed in $\mathcal{O}(1)$ time if one keeps track of the index position of c.

Constructing Succ is achieved by sweeping through D_{T_S} once from left to right and once from right to left. Thereby most recent positions of observed characters $i, \ldots, n$ are tracked in a table Occ. Let p be the index position of a character $c \geq i$ at a position y in D_{T_S}, Succ$[p]$ is set to Occ$[c+1]$ if

1. character $c+1$ has been observed in a previous position (including y) and
2. Succ$[p]$ is either undefined or $d^i_{T_S}(y, \text{Occ}[c+1]) < d^i_{T_S}(y, \text{Succ}[p])$ or $d^i_{T_S}(y, \text{Occ}[c+1]) = d^i_{T_S}(y, \text{Succ}[p])$ and Occ$[c+1] <$ Succ$[p]$.

Min-rank distances can be computed in $\mathcal{O}(1)$ time after $\mathcal{O}(m)$ preprocessing by performing *range maximum queries* on T^i_S [13].

5.6 Discovering strict common intervals

In this section we aim to solve the following problem:

Problem 9 *Given indeterminate strings S and T, discover all maximal strict common intervals of S and T.*

Similar to Algorithm WCII, the herein proposed algorithm is an adaptation of Didier's Algorithm [35]. Unlike the approach discussed in the previous section, for strict common intervals it is no longer sufficient to determine whether or not two positions of indeterminate strings S and T intersect. Instead, one must know the exact character set of the intersection, because every character of an interval of S must be contained in an interval of T in order to render both a strict common intervals pair. Henceforth, the algorithm operates not on index string I_S and index mapping T_S, but directly on indeterminate strings S and T.

An indeterminate string X gives rise to a family of ordinary strings of the form $\overline{X} = \overline{X}_1\overline{X}_2\cdots\overline{X}_{|X|}$, where each $\overline{X}_i$, $1 \leq i \leq |X|$, is a concatenation of the elements of $X[i]$ in arbitrary order. We call this family the *flat strings* of X. For our purpose it is sufficient to consider just one arbitrarily selected representative of them, denoted as $\overline{X}$ in the following. Note that one cannot naïvely run Didier's Algorithm on flat strings of S and T to obtain all maximal strict common intervals of S and T, because the method fails to find correct rank-nearest successors, resulting in paths that exceed the bounds of corresponding intervals in indeterminate string T as shown by this example:

Example 10 *Given two indeterminate strings* $S = \{1,2\}\,\{3,4\}\,\{5,6\}$ *and* $T = \{5,2\}\,\{1,3\}\,\{2,4\}\,\{6\}$ *and a maximal strict common intervals pair* $([1,2]_S, [2,3]_T)$. *For the sake of simplicity, let each character correspond to its actual rank. Consider the flat string* $\overline{T} = 5\,2\,1\,3\,2\,4\,6$ *of* T. *Candidates for the rank-nearest successor of character* 1 *at the* 3*rd position in* $\overline{T}$ *are positions* 2 *and* 5, *both accommodating character* 2. *The rank-nearest successor under rank distance (as defined in [35]) among the two is position* 2 *with rank distance* 2, *while position* 5 *has rank distance* 3. *Yet, position* 2 *in* $\overline{T}$ *corresponds to position* 1 *in* T, *which is outside of the interval* $[2,3]_T$ *corresponding to the strict common intervals pair.*

Following Didier *et al.* [35], we define by $\textsc{Rank}_X$ the *character ranking table w.r.t. a given ordinary string* X: For each character $c \in \Sigma$,

$$\textsc{Rank}_X[c] = \min_{\substack{1\leq i\leq |X|,\\ c=X[i]}} |\mathcal{C}(X[1,i])|$$

denotes its *rank*, i.e., the number of *distinct* characters up to, and including, the first occurrence of c in string X if such exists, or '∞' otherwise. If it is clear for which string X table $\textsc{Rank}_X$ is computed, we denote it simply by $\textsc{Rank}$.

In the ith iteration, the algorithm computes for each fixed left bound i in indeterminate string S table $\textsc{Rank}$ of flat string $\overline{S[i,n]}$. Choosing any representative of the flat string family of $S[i,n]$ results in an arbitrary assignment of ranks in character sets of positions in $S[i,n]$ that have more than one character. We will see that the choice of the flat string representative neither confines the correctness, nor affects

the runtime of our proposed algorithm, since every character in a character set must be "seen" by the algorithm, independent of the order determined by their ranks.

Similar to Didier *et al.*'s approach, our adaptation follows for each iteration i the paths of rank-nearest successors through indeterminate string T, starting at positions that contain the character corresponding to rank 1. Hereby the rank-nearest successor is determined w.r.t. the following distance:

Definition 23 (max-rank distance) *Given rank table* Rank*, the* max-rank distance *of any two positions k and l in indeterminate string T for the ith iteration is given by:*

$$d^i_{\uparrow}(k,l) \equiv \max\{\textsc{Rank}[c] \mid c \in \mathcal{C}(T[k,l])\}\,.$$

For each position y in T along the path of rank-nearest successors, we identify the largest interval $k \leq y \leq l$ around y for which holds that $\mathcal{C}(T[k,l])$ contains no character of higher rank than the current. We denote these intervals *max-rank intervals*. If $T[y]$ itself contains characters of higher rank, the max-rank interval remains undefined. If however the max-rank interval is defined, the algorithm tests if it completely surrounds the path of rank-nearest successors. In case the current rank is the highest rank in the character set of its corresponding position j in indeterminate string S, a pair of strict common intervals $([i,j]_S, [k,l]_T)$ is found. Whereas max-rank interval $[k,l]_T$ is maximal by definition, it remains to be tested if interval $[i,j]_S$ is maximal, too. To this end, the algorithm checks if both character sets of positions $S[i-1]$ and $S[j+1]$ (if such exist) contain a character with higher rank than the current. If true, $[i,j]_S$ is maximal and the interval pair $([i,j]_S, [k,l]_T)$ is reported.

Analogous to the approach in the previous section, max-rank intervals are precomputed and stored in table Int. However, unlike for min-rank intervals, for each position y in T there exists exactly one max-rank interval that is a valid candidate to participate in a pair of strict common intervals. Thus, once table Int is computed, it does not need to be updated within the ith iteration. We use Didier *et al.*'s original approach for computing rank intervals by applying it on the *max-rank string* $T^i_{\uparrow}$ of T, as defined as follows:

Definition 24 (max-rank string) *Given indeterminate string X and table* Rank*, $(X^i_{\uparrow}[p])_{p=1,\ldots,|X|}$ is the* max-rank string of X for the ith iteration*, where* $X^i_{\uparrow}[p] = \operatorname{argmax}_{c \in X[p]} \textsc{Rank}[c]$.

The rank interval of a position y in $T^i_{\uparrow}$ is identical to the only well-defined max-rank interval of $T[y]$. Lastly, rank-nearest successors can be precomputed and stored in table Succ similar to the approach described in Section 5.5.2.

For each of the n positions in S, our adaptation of Didier's algorithm uses $\mathcal{O}(\|T\|)$ time to compute tables Rank and Succ, and $\mathcal{O}(m)$ time to compute Int, consuming $\mathcal{O}(\|T\|)$ space. Traversing through paths of rank-nearest successors in Succ requires $\mathcal{O}(\|T\|)$ time, since every entry in table Succ is looked up a constant number of

Algorithm 7 Algorithm AWCII is a search algorithm for approximate weak common intervals in indeterminate strings. It is an adaptation of RGC [57], an algorithm for computing approximate common intervals in strings.

Input: Indeterminate strings S, T, indel threshold δ.
Output: All mutually-closed approximate weak common intervals of S and T with at most δ indels

Construct index mapping T_S and table Pos
for $i \leftarrow 1$ **to** $|S|$ **do**
  for $j \leftarrow i$ **to** $|S|$ **do**
    PREVIOUS $\leftarrow -\infty$
    for each element y in Pos$[j]$ **do**
      $d_k \leftarrow \delta$
      while $d_k \geq 0$ **do**
        $k \leftarrow$ leftmost position in T_S s.t. $|\{p \mid k \leq p \leq y : T_S[p] \cap [i,j] = \emptyset\}| \leq d_k$
        // d'_k corresponds to the observed number of indels in $[k,y]_{T_S}$
        $d'_k \leftarrow |\{p \mid k \leq p \leq y : T_S[p] \cap [i,j] = \emptyset\}|$
        if PREVIOUS $\geq k$ **then**
          break
        end if
        $d_l \leftarrow \delta - d_k$
        while $d_l \geq 0$ **do**
          $l \leftarrow$ rightmost position in T_S s.t. $|\{p \mid y \leq p \leq l : T_S[y] \cap [i,j] = \emptyset\}| \leq d_l$
          $d_S \leftarrow j - i - |\mathcal{C}(T_S[k,l]) \cap [i,j]| + 1$
          if $d_S + d_k + d_l \leq \delta$ **and** $\{i-1, j+1\} \cap \mathcal{C}(T_S[k,l]) = \emptyset$ **and** $i \in \mathcal{C}(T_S[k,l])$ **then**
            output $([i,j],[k,l])$
          end if
          $d_l \leftarrow |\{p \mid y \leq p \leq l : T_S[y] \cap [i,j] = \emptyset\}| - 1$
        end while
        $d_k \leftarrow d'_k - 1$
      end while
      PREVIOUS $\leftarrow y$
    end for
  end for
end for

times. Testing interval bounds in INT takes constant time. The maximality test for interval $[i,j]_S$ can also be achieved in constant time if at the beginning of the ith iteration the max-rank string of $S[\max(1, i-1), n]$ is determined, which takes $\mathcal{O}(\|S\|)$ time and $\mathcal{O}(n)$ space. We conclude with the following theorem:

Theorem 4 *Given two indeterminate strings S and T, all pairs of maximal strict common intervals of S and T are computable in $\mathcal{O}(|S| \cdot (\|S\| + \|T\|))$ time and $\mathcal{O}(\|S\| + \|T\|)$ space.*

5.7 Discovering approximate weak common intervals

We now present the algorithm *Approximate Weak Common Intervals on Indeterminate Strings* (AWCII) as presented in Algorithm 7, thus line numbers mentioned in this subsection refer to Algorithm 7. AWCII solves the following problem:

Problem 10 *Given two indeterminate strings S and T and indel threshold $\delta \in \mathbb{N}_0$, discover all mutually-closed approximate weak common intervals of S and T with no more than δ indels.*

Following a strategy similar to WCII, AWCII solves Problem 10 for index mappings I_S and T_S, instead of S and T. As before, in each iteration the algorithm maintains a fixed left bound i in I_S. For each character $j \in [i, n]$ all positions y in T_S are processed that contain character j (lines 5-25). Thereby character j at position y in T_S can be associated with several different intervals around y that are candidates of mutually-closed approximate weak common interval partners for interval $[i, j]_{I_S}$. Only intervals surrounding one (or several) positions y can be mutually-closed. However, for an interval $[k, l]_{T_S}$ containing indels, it no longer holds that the min-rank distance of any two positions within the interval is always smaller than the min-rank distance from any position inside to any position outside the interval. As a result, neither precomputed min-rank intervals nor following paths of rank-nearest successors can be used for improving the algorithm's performance. Instead we pursue a different approach, thereby making AWCII an adaptation of the RGC algorithm of Jahn [57].

For each $d_k = 1, \ldots, \delta$ (lines 7-23) AWCII identifies the leftmost position k in T_S such that at most d_k indels are contained in interval $[k, y]_{T_S}$ and $T_S[k] \cap [i, j] \neq \emptyset$. Let $d'_k \leq d_k$ be the observed number of indels in $[k, l]_{T_S}$ (see line 9), the algorithm then finds for each $d_l = 1, \ldots, \delta - d'_k$ (lines 14-21) the rightmost position l such that again $T_S[l] \cap [i, j] \neq \emptyset$ and the number of indels in $[y, l]_{T_S}$ does not exceed d_l. Each (k, l) of the at most $(\delta + 1)^2$ combinations of leftmost and rightmost positions gives rise to a candidate pair of mutually-closed approximate weak common intervals $([i, j]_{I_S}, [k, l]_{T_S})$. For each candidate pair it is checked that the number of characters in $[i, j]$ not contained in $\mathcal{C}(T_S[k, l])$ plus the already consumed number of indels in $[k, l]_{T_S}$ does not exceed δ. Finally, it is tested if $[i, j]_{I_S}$ is $\mathcal{C}(T_S[k, l])$-closed. If both conditions are satisfied, a mutually-closed approximate weak common intervals pair is found and is subsequently reported (line 18).

Runtime improvements are achieved by precomputing right and left bounds of candidate intervals $[k, l]_{T_S}$ for each character j in T_S. These bounds are computed within $\mathcal{O}((\delta + 1)\|T_S\|)$ time for a fixed left bound i in I_S and stored in tables L and R respectively. Further, for each such candidate interval $[k, l]_{T_S}$ the number of characters that are within $[i, j]$ can also be precomputed. This number is used to determine δ_S in line 16. The construction of the corresponding table, called RangeContent, is achieved within $\mathcal{O}((\delta + 1)^2 n\|T_S\|)$ time for a fixed left bound i. Note that RangeContent differs significantly from the data structure Num used in RGC [57] to count characters in intervals.

The first speedup in the algorithm is accomplished by precomputing left and right bounds of all candidate intervals in T_S of the ith iteration, which are subsequently stored in tables L and R. To this end, we sweep for each $j \in [i, n]$ through all characters in T_S from one side to the other. Specifically, when sweeping from left to right, table R is constructed, table L in reverse direction. During each sweep we keep track of up to $\delta + 1$ outermost positions in T_S that contain any character in $[i, j]$ and are not more than $0, 1, \ldots \delta$ indels away from the current position. Whenever we come across character j at any position y, all tracked outermost positions and their corresponding number of indels are stored in L, respectively R. Precomputation of L and R in the ith iteration takes $\mathcal{O}((\delta+1) \cdot n\|T_S\|)$ time and requires $\mathcal{O}((\delta+1) \cdot \|T_S\|)$ space.

The next improvement in performance of the algorithm is achieved by precomputing table RangeContent, which stores for each candidate interval $[k, l]_{T_S}$ corresponding to character j its character content shared with $[i, j]_{I_S}$, i.e., $C = \mathcal{C}(T_S[k, l]) \cap [i, j]$. The size of C is required in line 16 in Algorithm AWCII to compute δ_S. Consequently we store for each interval corresponding to character j at position y in T_S and defined by bounds in L and R, the number of characters that are within $[i, j]$. In other words, RangeContent stores at most $(\delta + 1)^2$ entries associated with candidate intervals of character j at position y in T_S.

Computing RangeContent can be achieved by sweeping for each $j \in [i, n]$ once from left to right and once from right to left through all characters in T_S. At first, table RangeContent is initialized with zeros. Every entry in RangeContent serves as counter for the number of characters that are within $[i, j]$ of a candidate interval in T_S. In each sweep we keep track of the most recent position p_j in which character j has been observed. We process every character c of each position in T_S for which $i \leq c \leq j$. Going through all combinations (k, l) of interval bounds in L and R for character c, we increase the counter of $[k, l]_{T_S}$ as long as $p_j \in [k, l]_{T_S}$ and the counter has not yet been increased for character j. The latter condition prevents that a character is counted twice, once during a sweep from left to right and then again when sweeping from right to left.

In terms of overall runtime, for each fixed bound i in I_S the algorithm spends $\mathcal{O}((\delta + 1)^2 n\|T_S\|)$ time on computation of the above mentioned auxiliary tables. Thereafter, AWCII requires $\mathcal{O}((\delta + 1)^2\|T_S\|)$ time to iterate through all combinations of candidate intervals in L and R and to identify approximate weak common intervals.

Testing for $\mathcal{C}(T_S[k, l])$-closedness of interval $[i, j]_{I_S}$ can be achieved in $\mathcal{O}(1)$ time by precomputing a table for all candidate intervals in T_S of the ith iteration, where each entry indicates if a character $i - 1$ or $j + 1$ is contained in the corresponding candidate interval. Such a table can be constructed within $\mathcal{O}((\delta + 1) \cdot \|T_S\|)$ time using a simple sweep algorithm. We conclude with the following theorem:

Algorithm 8 Algorithm ACSI is a runtime heuristic that computes all approximate weak common intervals in indeterminate strings.

Input: Indeterminate strings S, T, indel threshold δ.
Output: All mutually-closed approximate weak common intervals of S and T with at most δ indels

```
1: Construct index mapping T_S and table Pos
2: for i ← 1 to |S| do
3:     for each element y in Pos[i] do
4:         for j ← i to |S| do
5:             EXTEND(i, j, y, y, δ)
6:         end for
7:     end for
8: end for

9: procedure EXTEND(i, j, k, l, d)
10:    Increase interval [k, l]_{T_S} to both sides until [i, j] ∩ T_S[k − 1] = [i, j] ∩ T_S[l + 1] = ∅
11:    if i − 1 ∈ C(T_S[k, l]) or j + 1 ∈ C(T_S[k, l]) then
12:        return
13:    end if
       // test if there are interval pairs to output
14:    if j − i + 1 − |C(T_S[k, l]) ∩ [i, j])| ≤ d and i ∈ C(T_S[k, l]) then
15:        output ([i, j]_S, [k, l]_T)
16:    end if
       // extend to both directions until all indels d are consumed
17:    k' ← highest index strictly smaller than k − 1 such that C(T_S[k', k − 1]) ∩ [i, j] ≠ ∅
18:    if k − k' − 1 ≤ d and {i − 1, j + 1} ∩ C(T_S[k', k − 1]) = ∅ then
19:        EXTEND(i, j, k', l, d + k' + 1 − k)
20:    end if
21:    l' ← lowest index strictly larger than l + 1 such that C(T_S[l + 1, l']) ∩ [i, j] ≠ ∅
22:    if l' − l − 1 ≤ d and {i − 1, j + 1} ∩ C(T_S[l + 1, l']) = ∅ then
23:        EXTEND(i, j, k, l', d + l + 1 − l')
24:    end if
25: end procedure
```

Theorem 5 *Given two indeterminate strings S and T and indel threshold $\delta \in \mathbb{N}_0$, algorithm AWCII computes all pairs of mutually-closed approximate weak common intervals of S and T in $\mathcal{O}((\delta+1)^2 \cdot |S|^2 \|T_S\|)$ time.*

5.8 A runtime heuristic for discovering approximate weak common intervals

Algorithm ACSI (see Algorithm 8) represents a runtime heuristic that solves Problem 10 exactly and in practice outperforms both AWCII and, when $\delta = 0$, even WCII for the non-approximate problem variant (Problem 8) by orders of magnitude.

Just as algorithm WCII and AWCII, ACSI operates on index string I_S and index mapping T_S instead of indeterminate strings S and T directly. For every fixed interval $[i, j]$ in I_S, ACSI identifies mutually-closed approximate weak common in-

terval partners $[k,l]$ in T_S. To this end, it iterates through elements of $\text{Pos}[i]$, i.e., positions in T_S that contain character i (lines 3-7 of Algorithm 8). For each such position $y \in \text{Pos}[i]$ the algorithm calls a recursive procedure, denoted EXTEND (line 5). This recursive procedure requires 5 parameters, corresponding to fixed bounds $[i,j]_{I_S}$, the currently processed interval $[k,l]$ in T_S, and the current number of allowed indels, d. In the initial call, interval $[k,l]_{T_S}$ is set to $[y,y]_{T_S}$ and $d = \delta$. EXTEND then increases interval $[k,l]_{T_S}$ to both sides until $[i,j] \cap T_S[k-1] = \emptyset$ and $[i,j] \cap T_S[l+1] = \emptyset$ (line 10). If in this process the algorithm observes characters $i-1$ or $j+1$ in $\mathcal{C}(T_S[k,l])$, EXTEND returns to the previous call (lines 11-13). Otherwise, it verifies if $([i,j]_{I_S}, [k,l]_{T_S})$ is a mutually-closed approximate weak common intervals pair by testing if the number of characters in $[i,j]$ that are missing in $\mathcal{C}(T_S[k,l])$ is less than or equal to the current d and if $i \in \mathcal{C}(T_S[k,l])$ (line 14). The interval pair is reported if both conditions are validated. EXTEND then increases $[k,l]_{T_S}$ to the left, thereby consuming indel positions as long as their overall number remains less than or equal to the current d (line 17). If a position $k' < k-1$ has been found such that $[i,j] \cap T[k'] \neq \emptyset$, EXTEND is called recursively with parameter values $[i,j]_{I_S}$, $[k',l]_{T_S}$, and the remaining number of allowed indels, given by $d + k' + 1 - k$ (lines 18-20). This step is also symmetrically executed for the right side of $[k,l]_{T_S}$ (lines 21-24).

The actual heuristic speed-up of the algorithm is achieved by calling procedure EXTEND in line 5 not for all intervals $[i,j]$ in I_S but only for those that have chances of success for being a weak common intervals pair with an interval $[k,l]$ around a position $y \in \text{Pos}[i]$. Thus, the neighborhood around position y is scanned for suitable values of j prior to the execution of EXTEND. Instead of iterating through all $j \in [i,n]$ in lines 4-6, where $n = |S|$, a heuristic improvement is achieved by choosing only those values $\mathcal{J} \subseteq [i,n]$ that are reachable within at most δ indels around position y in T_S. To this end, the algorithm finds the leftmost position $k^\star \leq y$ in T_S such that $|\{p \mid k^\star \leq p \leq y : T_S[p] \cap [i,n] = \emptyset\}| \leq \delta$. Likewise it identifies the rightmost position $l^\star \geq y$ such that $|\{p \mid y \leq p \leq l^\star : T_S[p] \cap [i,n] = \emptyset\}| \leq \delta$. We can now set $\mathcal{J} = \mathcal{C}(T_S[k^\star, l^\star]) \cap [i,n]$. In practice $|\mathcal{J}| \ll n - i + 1$. Identifying $k^\star$, $l^\star$ and $\mathcal{J}$ is achievable within $\mathcal{O}(n+m)$ time and $\mathcal{O}(n)$ space. The algorithm further improves in runtime in practice by identifying the largest possible rightmost bound $j^\star$ in S such that $|\mathcal{J} \cap [i,j^\star]| \leq \delta$. Subsequently, characters larger than $j^\star$ in T_S are omitted in further iterations. These characters may lead to further indels in T_S, thus reducing the largest possible interval around y that is reachable within at most δ indels. The process of identifying the leftmost position $k^\star$ and rightmost position $l^\star$ in T is re-iterated, resulting in a possibly reduced character set $\mathcal{J} = \mathcal{C}(T_S[k^\star, l^\star]) \cap [i,j^\star]$. The iteration stops if the value of $j^\star$ no longer changes.

Example 11 *Given indeterminate strings* $S = \{g\}\{b,p\}\{x\}\{n,p\}\{d,o,s\}\{a,z\}\{e,n,w\}\{f\}\{l,v\}\{h,u,z\}\{j,r\}\{k\}\{m,q,y\}$ *and* $T = \{d\}\{g,b\}\{a\}\{p,s\}\{n\}\{a,b\}\{f,m,w\}\{e,w\}\{i\}\{q,y\}\{h\}\{c,r\}\{z\}$ *and indel threshold* $\delta = 1$, *Figure 5.3 (a) shows for*

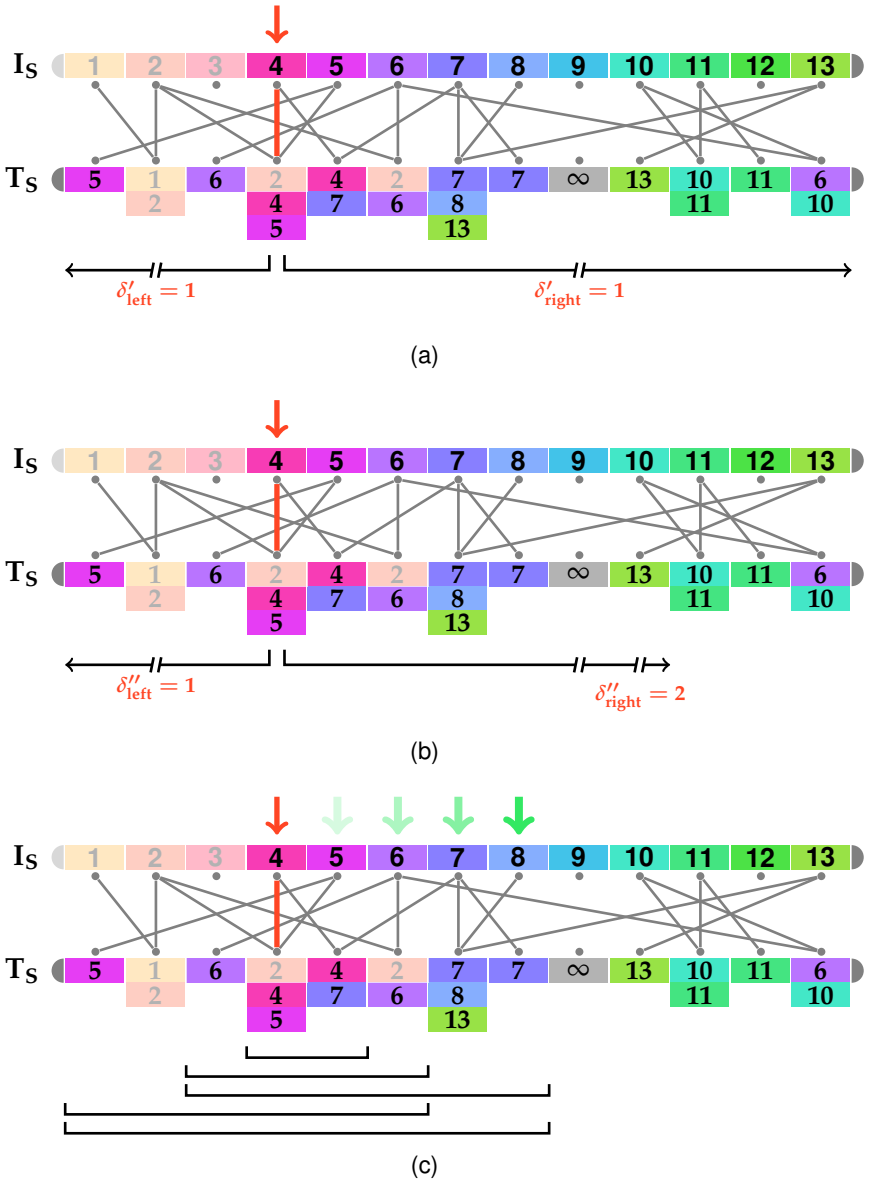


Figure 5.3: Visualization of heuristic runtime improvement for Example 11. The initial step, shown in Figure (a) leads to identification of index set $\mathcal{J} = \{4,5,6,7,8,10,11,13\}$. In the second iteration shown in Figure (b), the index set is reduced to $\mathcal{J} = \{4,5,6,7,8\}$. $\mathcal{J}$ represents the set of all right-most interval bounds of S that are candidates for approximate common intervals. By calling procedure EXTEND for each $j \in \mathcal{J}$, ACSI identifies candidate intervals, shown in Figure (c), for mutually-closed approximate weak common intervals pairs.

left bound $i = 4$ in S the search strategy of the heuristic improvement in its initial step, which determines $\mathcal{J} = \{4,5,6,7,8,10,11,13\} = \{4,\ldots,13\} \setminus \{9,12\}$. Since indel threshold is set to $\delta = 1$, the largest rightmost bound in S is $j^\star = 11$. In the second iteration shown in Figure 5.3 (b), $\mathcal{J} = \{4,5,6,7,8\}$ and $j^\star = 8$. The process terminates after the third iteration, since the largest rightmost bound $j^\star$ does not change. ACSI then calls for each $j \in J = \{4,5,6,7,8\}$ procedure EXTEND *to identify candidate intervals, shown in Figure 5.3 (c), for mutually closed approximate weak common intervals pairs.*

In terms of complexity, table Pos can be constructed within $\mathcal{O}(\|T_S\|)$ time and requires $\mathcal{O}(\|T_S\|)$ space as mentioned in Section 5.5. Algorithm ACSI calls EXTEND $\mathcal{O}(|S| \cdot \|T_S\|)$ times. EXTEND itself requires $\mathcal{O}(\delta^2\|T_S\|)$ time and $\mathcal{O}(\delta^2|S|)$ space, which can be improved to $\mathcal{O}(\delta|S|)$ space if the interval extension to the left side is always performed before extending to the right. The heuristic speed-up deteriorates the asymptotic runtime by a factor of $\mathcal{O}(|S|^2 + |S| \cdot |T|)$ time. Altogether, ACSI requires $\mathcal{O}(|S| \cdot \|T_S\| \cdot (|S|^2 + |S| \cdot |T| + \delta^2\|T_S\|)$ time and $\mathcal{O}(\|T_S\| + \delta|S|)$ space. However in practice Algorithm ACSI outperforms both WCII and AWCII as shown in the next section.

5.9 Results and Discussion

In the following, we highlight the benefit of our family-free synteny model in comparison with present family-based approaches. To this end, we chose a genomic dataset of bacterial genomes that has been used in a prior synteny study [69] and was originally obtained from [32]. The dataset features 133 chromosomal sequences, of which we removed all sequences originating from plasmids.

5.9.1 Gene family-based dataset

Genes in this dataset are annotated with COG (Clusters of Orthologous Groups) identifiers [107] which are used to establish a homology assignment between genes. The set of genes in the dataset was revised by the latest available gene information under the accession numbers of the respective genomes at NCBI. To this end, genes that are meanwhile marked as pseudo-genes were removed from the dataset. No genes were added, since COG annotations of new genes are not available. We further omitted all genomes from subsequent analyses of which more than 10 pseudo-genes were removed in this process. 93 genomes remained, comprising on average 2726 genes (minimum/maximum number of genes: 784/8317).

5.9.2 Gene family-free dataset

Pairwise similarities between genes in the dataset were obtained using the relative reciprocal BLAST score [84]. Genes were compared on the basis of their encoding

Stringency f	Avg. #disconnected genes	Avg. pairwise similarities	Avg. node degree
unpruned	3262	6499	1.93
0.1	3271	6242	1.85
0.2	3312	5349	1.60
0.3	3364	4137	1.25
0.4	3420	3095	0.96
0.5	3486	2336	0.74
0.6	3565	1784	0.58
0.7	3661	1384	0.46
0.8	3775	1088	0.38
0.9	3901	877	0.31

Table 5.1: Average number of disconnected genes, average number of pairwise similarities, and average node degree of genes in pairwise gene similarity graphs as a function of stringency in our dataset. The average number of genes per pairwise comparison in the dataset is 5444. Disconnected genes are genes that are not incident to an edge in pairwise gene similarity graphs.

protein sequence using BLASTP [3] with an e-value threshold of 0.1 and disabled composition-based score adjustments.

For evaluation purposes, we produced different degrees of pruned pairwise gene similarity graphs by filtering spurious gene similarities. To this end, we employed an undirected variant of the stringency criterion parameterized by $f \in [0,1]$ as proposed in [64]. Given two genomes G and H, the stringency filter takes into account gene similarities of genes $g \in \mathcal{C}(G)$ and $h \in \mathcal{C}(H)$, when calculating a local threshold value for gene similarity $\sigma_{GH}(f,g,h)$. The maximal similarity of gene g is given by $\sigma_{GH}^{\max}(g) = \max_{h' \in \mathcal{C}(H)}(\sigma_{GH}(g,h'))$. The maximal similarity of gene h to any gene g in genome G is analogously defined. The *stringency* parameter $f \in [0,1]$ allows to restrict the set of gene similarities to those with a certain degree of locally minimal weight:

$$\sigma_{GH}(f,g,h) = \begin{cases} \sigma_{GH}(g,h) & \text{if } \sigma(g,h) \geq f \cdot \max\{\sigma_{GH}^{\max}(g), \sigma_{HG}^{\max}(h)\} \\ 0 \quad \text{otherwise.} \end{cases}$$

The effects in the choice of parameter values for f captured on three different properties of our dataset in pairwise comparisons are shown in Table 5.1.

To evaluate our family-free model, we ran an implementation of ACSI for $\delta = 0$ on all unpruned pairwise gene similarity graphs in our dataset and compared the obtained $4,015,841$ family-free syntenic blocks with respect to their contained COG identifiers. We discarded all family-free syntenic blocks for which at least one interval contained less than two genes with a COG identifier. In the remaining $1,194,036$ family-free syntenic blocks, we observed that the similarity between their respective sets of COG identifiers depends strongly on the family-free synteny score proposed in Equation (5.2), as shown in Table 5.2. Among the family-free syntenic

	family-free synteny score										
overlap in %	< 1	$[1-2[$	$[2-3$	$[3-4[$	$[4-5[$	$[5-6[$	$[6-7[$	$[7-8[$	$[8-9[$	$[9-10[$	≥ 10
100	28.1	22.0	46.7	78.4	90.2	75.6	84.6	63.2	86.5	78.4	95.0
$[80-100[$	0.0	0.0	0.1	0.2	0.4	1.8	2.0	10.4	8.2	18.5	4.9
$[60-80[$	1.7	1.7	2.7	2.1	2.7	13.6	8.4	17.4	4.0	2.6	0.2
$[40-60[$	12.0	14.7	18.5	9.9	2.4	2.5	2.1	5.0	0.7	0.3	0.0
$[20-40[$	0.1	0.1	0.3	0.7	1.1	3.4	1.5	1.8	0.1	0.2	0.0
$[0-20[$	58.1	61.4	31.8	8.8	3.2	3.1	1.4	2.7	0.6	0.2	0.0
total	30002	239077	289450	253643	199372	49254	58889	17952	23603	4568	28226

Table 5.2: Statistics of overlaps between the COG identifier sets in family-free syntenic blocks. Columns stand for bins of family-free synteny scores, rows for bins of overlap sizes of their corresponding COG identifier sets. Values are given in percent with respect to the total number of blocks per score bin given in the last row.

blocks with a score greater than or equal to 10, 95% have the same set of identifiers in both intervals. While this number decreases for smaller scores, still a quarter of the family-free syntenic blocks with a score lower than 1 do not differ in their COG identifiers. This shows that our approach is able to detect syntenic blocks that would also be detected with well-established gene family-based approaches. This is not a surprise, as weak common intervals are in fact a generalization of the classic common intervals model: A run of ACSI on a dataset where similarity scores are only set between members with the same COG identifiers finds the exact same set of syntenic blocks as the common intervals-based approach.

To evaluate the predictive power of our approach, we compare the output of our program to syntenic blocks predicted by the *reference gene cluster algorithm* (RGC) [57]. While this algorithm is capable of multiple genome comparison and the detection of faint conservation patterns, we use it in this context for pairwise genome comparison to detect interval pairs (I_1,I_2) whose gene sets have a symmetric set distance of at most 2. It has been previously observed that the generalization to approximate conservation underlying the RGC method is not only a way to find imperfectly conserved syntenic blocks, but also a means to add robustness against errors in gene family assignment. For example, an interval pair may appear to have a set distance of two because besides the shared genes, there is one gene unique to I_1 and one gene unique to I_2. However a closer inspection of the genes reveals that these genes are in fact homologs that were not recognized in the preceeding partitioning of genes into homology families. We ran RGC on all pairs of the 93 genomes with gene family assignments obtained from the COG database, thereby setting parameters $\delta = 2$ (maximal tolerated symmetric set distance) and $s = 3$ (minimal interval length). The program returned among others $192,900$ "single-mismatch blocks", i.e., syntenic blocks that contain exactly one gene family in each interval that is not shared between the two. In $47,453$ (24.60%) of the single-mismatch blocks, we observe a similarity score between the two extra genes in our BLAST dataset. ACSI found 89.84% of the single-mismatch blocks and 75.24% of the extra gene pairs turned out to be pairwise best hits. Moreover we observe that in $18,143$ among

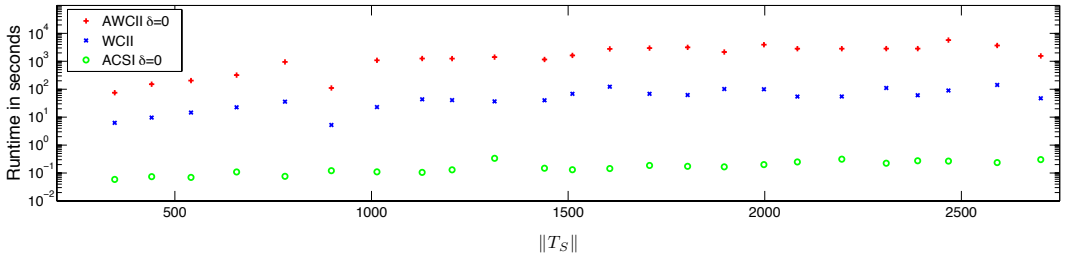


Figure 5.4: Running times of algorithms ACSI and AWCII with $\delta = 0$ and WCII, measured in a sample of 24 pairwise comparisons of genomes that are contained in the studied dataset. All algorithms produced identical output (as expected). Running times are plotted against the number of pairwise gene similarities (equivalent to the size of $\|T_S\|$) contained in the pairwise comparison.

the single-mismatch blocks predicted by RGC the two extra genes have exactly the same annotation string. (Annotations containing the word "hypothetical" were ignored.) ACSI finds 90.19% of these single-mismatch blocks. Surprisingly, 4.59% of the blocks in which the two extra genes had best hits to each other were not found by ACSI. This is because for one or more of the other genes in the syntenic blocks our BLAST results did not return any similarity score to a gene in the other interval. Apparently, genes in syntenic blocks can be very faintly related in terms of sequence similarity.

In practice ACSI outperforms both WCII and AWCII as shown by Figure 5.4. Thus, in all subsequent results, we used ACSI to compute mutually-closed (approximate) weak common intervals.

5.9.3 Comparison with RegulonDB

We now evaluate the ability of our family-free model to identify gene clusters in bacterial genomes and compare our results with those obtained by RGC with gene families obtained from the COG database.

Among other information about transcriptional regulation, RegulonDB [89] provides a list of operon locations in *Escherichia coli* K12. While the majority of operons in RegulonDB are computationally predicted, some are also experimentally confirmed. From 2649 operons reported in RegulonDB, 846 span two or more genes. We mapped these operons to the annotation of the *E. coli* K12 genome in our data set. However, 104 operons contain genes that are not annotated in our dataset and thus were omitted from subsequent analysis. The remaining 742 operons span between 2 and 16 genes, 71.83% of which span 2 or 3 genes. The number of detected operons depends strongly on the degree of evolutionary relatedness between the *E. coli* K12 genome and other genomes in the dataset. While ACSI and RGC predicted many

Unique to...	RGC $\delta = 0$	RGC $\delta = 2$	ACSI $\delta = 0$, $f = 0.0$	ACSI $\delta = 0$, $f = 0.9$	ACSI $\delta = 2$, $f = 0.0$	ACSI $\delta = 2$, $f = 0.9$
RGC $\delta = 0$	-	118	133	119	190	175
RGC $\delta = 2$	0	-	56	49	80	72
ACSI $\delta = 0$, $f = 0.0$	4	45	-	0	61	52
ACSI $\delta = 0$, $f = 0.9$	11	59	21	-	82	62
ACSI $\delta = 2$, $f = 0.0$	0	8	0	0	-	0
ACSI $\delta = 2$, $f = 0.9$	5	20	11	0	20	-

Table 5.3: Unique findings (with 100% overlap) of operons by ACSI and RGC with minimum interval length $s = 2$ and varying parameters. Each column shows the number of unique findings of an algorithm and parameter setting indicated by the column heading in comparison to algorithms and parameter settings specified in the rows.

occurrences in other closely related γ-proteobacteria in our dataset, for the majority of genomes only few occurrences of operons were reported. The sets of reported operons found by ACSI and RGC are not entirely overlapping. Instead, ACSI finds operons which RGC does not find and vice versa. A complete overview of unique findings for algorithms and parameter settings is shown in Table 5.3.

Figure 5.5 gives an overview of the number of found operons in the dataset, in which genomes are ordered according to the cumulative sum of operon findings by ACSI. We differentiated between findings where the gene set of a family-free syntenic block was a superset of genes of an operon (labeled "100% overlap"), more than 50%, or at most 50% of genes overlapped. In cases where multiple family-free syntenic blocks in different locations of the other genomes contained operon genes, we chose the block with the highest overlap. The graph visualizing the sum over all 100%, $> 50\%$, and $\leq 50\%$ overlaps is labeled as "cumulative". Out of the 742 operons, 633 are entirely overlapped by family-free syntenic blocks reported by ACSI in all genome comparisons against *E. coli* K12 in the unpruned graph and 612 operons for a pruned graph with $f = 0.9$. In comparison, RGC with $\delta = 0$ and $s = 2$ finds 504 operons, whereas 622 operons are found with parameter settings $\delta = 2$ and $s = 3$.

5.9.4 Discussion

In this chapter, we introduced a family-free model for synteny detection that is based on the study of (approximate) weak common intervals in indeterminate strings. Our family-free model relies on a scoring scheme of family-free syntenic blocks which rates both interval size and the degree of similarity between genes contained in the two corresponding intervals.

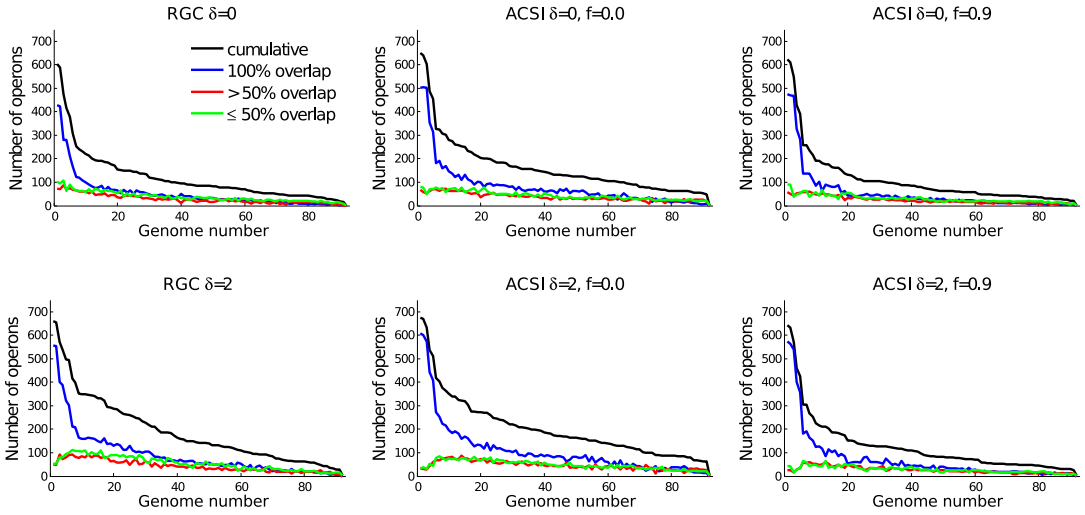


Figure 5.5: Number of operons in genomes reported by RGC with parameter settings $\delta = 0$, $s = 2$ (top left) and $\delta = 2$, $s = 3$ (bottom left) and ACSI in unpruned graphs (middle) and pruned graphs with $f = 0.9$ (right) for $\delta = 0$ (top) and $\delta = 2$ (bottom). Genomes are ordered according to the cumulative sum of reported operons.

We use our family-free model to predict syntenic blocks and gene clusters between pairs of genomes. This approach is evaluated in comparison with the common intervals-based reference gene cluster model of Jahn [57]. To this end, we analyzed genomic data of 93 bacterial genomes annotated with COG identifiers and studied operons reported in RegulonDB [89]. The results show that the correlation of COG annotations in family-free syntenic blocks depends strongly on the family-free synteny score: For family-free syntenic blocks with scores 10 or higher, 95% or more have identical COG identifiers in both corresponding intervals. Moreover, in an analysis of syntenic blocks with faint conservation pattern we observed that our approach is able to predict conserved regions which would be missed by family-based methods due to deficient gene family assignments. Further results indicate that sequence similarities are not always sufficient to establish homologies. Therefore, it may be worthwhile to enrich the gene similarity graph with other information such as COG data.

The analysis of operons contained in RegulonDB shows that the number of operons occurring in family-free syntenic blocks obtained by ACSI is higher than in their counterparts reported by RGC. However, we note that the presented analyses do not capture the specificity of the studied algorithms in finding operons. That is, family-free syntenic blocks in which operon locations are reported may span several additional genes. This may be due to the fact that the genomic region surrounding the operon exhibits conserved gene content, or due to the deficiency of the model

in differentiating between conserved and unconserved gene content, as a result of an unreliable measure of gene similarity.

Chapter 6

Conclusion and outlook

In this thesis, we explored a recent line of research in the field of computational comparative genomics, whose goal is to devise new methods for rearrangement studies that do not require prior gene family assignments. We established family-free models in two major applications of the field: We present two different approaches to detect conserved structures in gene orders. Furthermore, we introduce a family-free model in ancestral genome reconstruction that addresses the problem of constructing the median of three genomes.

In Chapter 3, we presented a family-free model to identify conserved adjacencies between two or more genomes. To this end, we formulated problem FF-Adjacencies, which asks for a matching in the gene similarity graph of two genomes that maximizes a linear combination of a measure of conserved adjacencies and edge weights, parameterized by $\alpha \in [0, 1]$. We subsequently extended problem FF-Adjacencies to the study of two or more genomes. Focusing on the pairwise case, we proved problem FF-Adjacencies NP-hard for $0 < \alpha < \frac{1}{3}$, which was later shown to be true for $\alpha > 0$ by Kowada *et al.* [62].

We subsequently described 0-1 linear program `FFAdj-2G` to compute exact solutions of problem FF-Adjacencies for two genomes. We then devised two methods to identify usually very good, but not generally optimal solutions in the search space of problem FF-Adjacencies. One of the methods could be applied in practice and showed outstanding performance in reducing the number of variables and constraints of program `FFAdj-2G`. With that, we are able to compute exact solutions between genomes with more than 4000 genes. Furthermore, we present heuristic `FFAdj-MCS` that is based on the idea of identifying maximal common substrings between two gene orders. Both, `FFAdj-2G` and `FFAdj-MCS` are based on previous works of Angibaud *et al.* [6].

Our new algorithms were evaluated on a simulated dataset. To this end, we utilized ALF [33], a popular framework for genome simulations. Parameters of

genome evolution in ALF were adjusted to high rates of genome rearrangements and gene duplications/losses, while keeping sequence evolution at a moderate rate. This results in a particularly difficult scenario for family-free models. Nevertheless, one-to-one ortholog assignments derived from solutions of our algorithms exhibited high precision, recall and accuracy. Yet, maximum weight matchings (that do not make use of synteny information) in our dataset showed comparable performance in one-to-one orthology prediction. However, the numbers of conserved adjacencies in solutions of program `FFAdj-2G` and `FFAdj-MCS` were considerably larger. This result is not surprising because of the many gene duplications that led to numerous synteny-independent orthologies between genes of two genomes. Consequently, the identification of positional orthologs is a difficult task, to whose realization gene family-free methods can tremendously contribute.

Lastly, both algorithms were applied on a dataset of twelve γ-proteobacterial genomes. We compared our results to that of Angibaud *et al.* [6], showing that our matchings are considerably larger and contain on average more conserved adjacencies.

We then proceeded to study a simple family-free model for ancestral reconstruction in Chapter 4. In doing so, we proposed problem FF-Median, which asks for the construction of a median of three given genomes that maximizes a pairwise measure of conserved adjacencies. We showed that this problem is inherently related to the weighted independent set problem. We proved by reduction that problem FF-Median is MAX SNP-hard. We then described 0-1 linear program `FF-Median` for its solution.

The model underlying problem FF-Median is prone to events of gene family evolution: its has limited ability to tolerate gene duplications and is generally unable cope with gene losses that occurred on the evolutionary paths between the common ancestor and the three genomes subject to analysis. Unfortunately, preprocessing methods that are commonly used in gene family-based analysis to treat differences in gene content prior to ancestral reconstruction fail in family-free analysis. An attempt to extend problem FF-Median toward a model with improved resistance against perturbations in the gene order caused by events of gene family evolution leads to a problem that is computationally infeasible for practical applications. We therefore proposed heuristic `FFAdj-3G-H` that integrates exact solutions to problem FF-Adjacencies between two genomes into a (possibly suboptimal) solution to problem FF-Adjacencies for three genomes. The heuristic employs Tannier *et al.*'s method [106] to construct a valid median genome.

Experiments on simulated datasets reveal superior performance of heuristic `FFAdj-3G-H` in identifying positional orthologs and in reconstructing the ancestral gene order of the common ancestor of three genomes. We then demonstrate applicability of heuristic `FFAdj-3G-H` in biological datasets by reconstructing the gene order of protein coding genes of the black death, using three extant *Yersinia*

pestis strains. The gene order and the one-to-one orthology assignment constructed by `FFAdj-3G-H` is mostly in agreement with results of Rajaraman *et al.* [87]. This outcome is expected, since the three *Yersinia pestis* strains descended only 650 years ago from the black death. Consequently, the number of evolutionary modifications along their individual evolutionary paths is very small.

In Chapter 5 we presented family-free models and algorithms to discover syntenic blocks of two genomes in a family-free setting. However, unlike our previous family-free models, the detection of family-free syntenic blocks does not lead to one-to-one orthology assignments between genes. This simplification allowed us to design fast polynomial time algorithms to discover all family-free syntenic blocks of two genomes by means of a pairwise gene similarity measure. In doing so, we phrased the detection of family-free syntenic blocks as a problem of discovering common intervals in indeterminate strings. Whereas many other traditional string problems have also been studied in indeterminate strings, the problem of detecting common intervals remained unaddressed. We explored three common interval models in indeterminate strings: Weak common intervals, strict common intervals, and approximate weak common intervals. We then devised efficient polynomial time algorithms for their corresponding discovery problems.

We evaluated our family-free synteny model by discovering family-free syntenic blocks in 93 bacterial genomes. The identified pairs of syntenic blocks show high overlap in COG identifiers. Moreover, we tested the capability of our family-free method in detecting gene clusters, by identifying operons in an *Escherichia coli* K 12 strain that are reported in RegulonDB [89], a database collecting information about for transcriptional regulation. We compared our results to those obtained by a family-based algorithm of Jahn [57] on the basis of COG annotations. The evaluation showed that our family-free approach is able to find considerably more operons.

Outlook

With program `FFAdj-2G`, we presented a first exact algorithm to study family-free adjacencies. Yet, there is much room for improving its practical running time through further studies that explore the solution space of problem FF-Adjacencies. Such studies may lead to the identification of other suboptimal solutions that could then be discarded prior to running program `FFAdj-2G`. In doing so, the remaining subgraph test described in Section 3.7.2 should be put into practice, by evaluating fast algorithms to compute maximum weighted matchings (or approximations thereof).

In all our family-free analyses, we employed the *relative reciprocal BLAST score* [84] to measure similarities between genes. While sequence similarity is an obvious and reasonable measure in constructing the gene similarity graph, similarity scores can

also integrate additional information such as functional similarity. Such information can be obtained from various databases, most notably, from the Gene Ontology database [9]. Family-free genome comparisons of this kind may give further insights into the functional organization of the genome. Moreover, family-free genome comparison can also be performed based on distances between genes, instead of similarities. The use of distance measures could lead to the formulation of family-free distances that are suitable for distance-based phylogenetic reconstruction. A principal study was performed by Martinez *et al.* [72], who proposed a family-free variant of the well-known *double cut and join* (DCJ) distance. However, their distance lacks metric properties, a not surprising consequence of the unconstrained similarity measure. Even worse, phylogenetic reconstruction requires tree-additive metrics, yet the DCJ distance as well as most other gene order distances rely on the principle of parsimony, which makes them generally not tree-additive [1, 102]. Nevertheless, it is well-known that many tree reconstruction algorithms, such as the prominent neighbor-joining method [88], are to a certain extent robust against deviations from tree-additivity, which lead to the study of near-additive metrics [10]. A promising avenue of further research is to combine family-free distances with substitution rate functions of DNA substitution models. Whereas the latter provide anticipated properties of tree-additivity, family-free distances could increase accuracy in reconstruction, as the rate of genome rearrangements is generally much lower than the rates of point mutations. Such combined distances can be studied in a similar framework as proposed in [36] by means of affine-additive distance mappings.

Future work in the study of family-free medians should certainly include a study of the solution space of problem FF-Median, which may lead to the exclusion of suboptimal solutions or the fixation of optimal components prior to analysis. Such results are already known for the family-based problem: Kováč showed in [61] that in the mixed multichromosomal breakpoint median of three all adjacencies occurring in two or three genomes are part of an optimal solution. Further efforts should be directed toward the formulation of a family-free median problem that tolerates gene duplications and losses.

Furthermore, our model of family-free synteny could benefit from an extension to the simultaneous discovery of syntenic blocks in more than two genomic sequences. This requires an extension of the definition of closedness from pairs to sets of intervals in two or more indeterminate strings. We suggest the following definition:

Definition 25 *A set of intervals C in two or more indeterminate strings is* closed *if there is no interval c in C that has a right or left neighboring position whose character set intersects with the character sets of all other intervals $c' \neq c$ of C.*

Note that closed intervals and mutually-closed intervals are equivalent in the pairwise case. However, the definition allows that subsets of a set of closed intervals may not be closed. This makes the enumeration of closed weak common intervals

in more than two indeterminate strings particularly challenging and a worthwhile subject of future work.

In this thesis, we presented three different studies of family-free genome comparison. Yet, the still young branch of family-free genome comparison offers many directions in which presented studies can be extended. Most evidently, the principle of family-free genome comparison can be applied to numerous other existing family-based studies. More interestingly, the family-free principle could even be integrated into a methodology to study sequence, synteny, and gene family evolution simultaneously. Such methods will benefit many kinds of applications, ranging from methods for species tree/gene tree reconciliation and gene tree reconstruction to the study of whole genome duplications and the reconstruction of ancestral genomes. The idea of integrative methods to study genome evolution as a whole is not new: It was already exploited in a seminal paper by Sankoff and El-Mabrouk in 2000 [97] and its relevance was recently emphasized by Chauve *et al.* [30]. This work contributes to the cause by reducing the gap between studies of sequence evolution and genome rearrangements.

Bibliography

[1] S. Aganezov and M. A. Alekseyev. On pairwise distances and median score of three genomes under DCJ. *BMC Bioinformatics*, 13 Suppl 19:S1, 2012.

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

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol*, 215(3):403–410, 1990.

[4] A. Amir, A. Apostolico, G. M. Landau, and G. Satta. Efficient text fingerprinting via Parikh mapping. *J. Discrete Algorithms*, 1(5–6):409–421, 2003.

[5] A. Amir, L. Gasieniec, and R. Shalom. Improved approximate common interval. *Inform. Process. Lett.*, 103(4):142–149, 2006.

[6] 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. *J. Comp. Biol.*, 15(8):1093–1115, 2008.

[7] P. Antoniou, M. Crochemore, C. S. Iliopoulos, I. Jayasekera, and G. M. Landau. Conservative string covering of indeterminate strings. In *Proc. of PSC 2008*, 108–115. 2008.

[8] P. Antoniou, C. S. Iliopoulos, I. Jayasekera, and W. Rytter. Computing repetitive structures in indeterminate strings. In *Proc. of PRIB 2008*, 108–115. 2008.

[9] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, *et al.* Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nat. Genet.*, 25(1):25–29, 2000.

[10] K. Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25(2–3):251–278, 1999.

[11] O. Attie, A. E. Darling, and S. Yancopoulos. The rise and fall of breakpoint reuse depending on genome resolution. *BMC Bioinformatics*, 12 Suppl 9:S1, 2011.

[12] R. Bar-Yehuda and S. Moran. On approximation problems related to the independent set and vertex cover problems. *Discrete Appl. Math.*, 9(1):1–10, 1984.

[13] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proc. of LATIN 2000*, vol. 1776 of *LNCS*, 88–94. 2000.

[14] A. Bergeron, S. Corteel, and M. Raffinot. The algorithmic of gene teams. In *Proc. of WABI 2002*, vol. 2452 of *LNCS*, 464–476. 2002.

[15] A. Bergeron, J. Mixtacki, and J. Stoye. On sorting by translocations. *J. Comp. Biol.*, 13(2):567–578, 2006.

[16] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. *J. Comp. Biol.*, 13(7):1340–1354, 2006.

[17] M. Bernt, D. Merkle, and M. Middendorf. Solving the preserving reversal median problem. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 5:332–347, 2008.

[18] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172(1):GC 11–GC 17, 1996.

[19] G. Blin, A. Chateau, C. Chauve, and Y. Gingras. Inferring positional homologs with common intervals of sequences. In *Proc. of RECOMB 2006*, vol. 4205 of *LNCS*, 24–38. 2006.

[20] G. Blin, C. Chauve, and G. Fertin. The breakpoint distance for signed sequences. In *Proc. of CompBioNets 2004*, vol. 3 of *Texts in Algorithmics*, 3–16. 2004.

[21] G. Blin, C. Chauve, and G. Fertin. Genes order and phylogenetic reconstruction: Application to γ-proteobacteria. In *Proc. of RECOMB-CG 2005*, vol. 3678 of *LNCS*, 11–20. 2005.

[22] G. Blin, C. Chauve, G. Fertin, R. Rizzi, and S. Vialette. Comparing genomes with duplications: A computational complexity point of view. *IEEE/ACM Trans. Comput. Biology Bioinf.*, 4(4):523–534, 2007.

[23] S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye. Computation of median gene clusters. *J. Comput. Biol.*, 16(8):1085–1099, 2009.

[24] K. I. Bos, V. J. Schuenemann, G. B. Golding, H. A. Burbano, N. Waglechner, *et al.* A draft genome of yersinia pestis from victims of the black death. *Nature*, 478(7370):506–510, 2011.

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

[26] M. D. V. Braga, C. Chauve, D. Doerr, K. Jahn, J. Stoye, A. Thévenin, and R. Wittler. The potential of family-free genome comparison. In *Models and Algorithms for Genome Evolution*, vol. 19 of *Comp. Biol.*, chap. 13, 287–323. Springer London, 2013.

[27] D. Bryant. The complexity of calculating exemplar distances. *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, 1:207–212, 2004.

[28] I. J. Burgetz, S. Shariff, A. Pang, and E. R. M. Tillier. Positional homology in bacterial genomes. *Evol. Bioinform. Online*, 2:77–90, 2006.

[29] A. Caprara. The reversal median problem. *Informs J. Computing*, 15(1):93–113, 2003.

[30] C. Chauve, N. El-Mabrouk, L. Guéguen, M. Semeria, and E. Tannier. Duplication, rearrangement and reconciliation: A follow-up 13 years later. In *Models and Algorithms for Genome Evolution*, vol. 19 of *Comp. Biol.*, 47–62. Springer London, 2013.

[31] J. C. Chiu, E. K. Lee, M. G. Egan, I. N. Sarkar, G. M. Coruzzi, and R. DeSalle. OrthologID: Automation of genome-scale ortholog identification within a parsimony framework. *Bioinformatics*, 22(6):699–707, 2006.

[32] F. D. Ciccarelli, T. Doerks, C. von Mering, C. J. Creevey, B. Snel, and P. Bork. Toward automatic reconstruction of a highly resolved tree of life. *Science*, 311(5765):1283–1287, 2006.

[33] D. A. Dalquen, M. Anisimova, G. H. Gonnet, and C. Dessimoz. Alf – a simulation framework for genome evolution. *Mol. Biol. Evol.*, 29(4):1115–1123, 2012.

[34] C. N. Dewey. Positional orthology: Putting genomic evolutionary relationships into context. *Brief. Bioinformatics*, 12(5):401–412, 2011.

[35] G. Didier, T. Schmidt, J. Stoye, and D. Tsur. Character sets of strings. *J. Discr. Alg.*, 5(2):330–340, 2007.

[36] D. Doerr, I. Gronau, S. Moran, and I. Yavneh. Stochastic errors vs. modeling errors in distance based phylogenetic reconstructions. *Algorithm. Mol. Biol.*, 7(1):22, 2012.

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

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

[39] R. Duan and H.-H. Su. A scaling algorithm for maximum weight matching in bipartite graphs. In *Proc. of SODA 2012*, 1413–1424. 2012.

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

[41] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. MIT Press, 2009.

[42] C. Frech and N. Chen. Genome-wide comparative gene family classification. *PLoS ONE*, 5(10):e13 409, 2010.

[43] Z. Fu, X. Chen, V. Vacic, P. Nan, Y. Zhong, and T. Jiang. MSOAR: A high-throughput ortholog assignment system based on genome rearrangement. *J. Comput. Biol.*, 14(9):1160–1175, 2007.

[44] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

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

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

[48] X. He and M. H. Goldwasser. Identifying conserved gene clusters in the presence of homology families. *J. Comp. Biol.*, 12(6):638–656, 2005.

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

[50] S. Heber and J. Stoye. Algorithms for finding gene clusters. In *Proc. of WABI 2001*, vol. 2149 of *LNCS*, 252–263. 2001.

[51] J. Holub and W. Smyth. Algorithms on indeterminate strings. In *Proc. of AWOCA 2003*, 36–45. 2003.

[52] J. Holub, W. F. Smyth, and S. Wang. Fast pattern-matching on indeterminate strings. *J. Discr. Alg.*, 6(1):37–50, 2008.

[53] S. B. Hoot and J. D. Palmer. Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. *J. Mol. Evol.*, 38(3):274–281, 1994.

[54] D. J. Houck and R. R. Vemuganti. An algorithm for the vertex packing problem. *Operations Research*, 25(5):773–787, 1977.

[55] C. Iliopoulos, M. S. Rahman, M. Voracek, and L. Vagner. Finite automata based algorithms on subsequences and supersequences of degenerate strings. *J. Discr. Alg.*, 8(2):117 – 130, 2010.

[56] C. S. Iliopoulos, M. S. Rahman, and W. Rytter. Algorithms for two versions of LCS problem for indeterminate strings. *J. Comb. Math. Comb. Comp.*, 71:155–172, 2009.

[57] K. Jahn. Efficient computation of approximate gene clusters based on reference occurrences. *J. Comput. Biol.*, 18(9):1255–1274, 2011.

[58] J. M. Joseph and D. Durand. Family classification without domain chaining. *Bioinformatics*, 25(12):i45–i53, 2009.

[59] O. Keller, M. Kollmar, M. Stanke, and S. Waack. A novel hybrid gene prediction method employing protein multiple sequence alignments. *Bioinformatics*, 27(6):757–763, 2011.

[60] E. V. Koonin. Orthologs, paralogs, and evolutionary genomics. *Annu. Rev. Genet.*, 39:309–338, 2005.

[61] J. Kováč. On the complexity of rearrangement problems under the breakpoint distance. *J. Comput. Biol.*, 21(1):1–15, 2013.

[62] L. A. B. Kowada, D. Doerr, S. Dantas, and J. Stoye. New genome similarity measures based on conserved gene adjacencies. In *Proc. of RECOMB 2016, to appear*, LNBI. Springer Verlag, Berlin, 2016.

[63] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.

[64] M. Lechner, S. Findeiß, L. Steiner, M. Marz, P. F. Stadler, and S. J. Prohaska. Proteinortho: Detection of (co-)orthologs in large-scale analysis. *BMC Bioinformatics*, 12:124, 2011.

[65] M. Lechner, M. Hernandez-Rosales, D. Doerr, N. Wieseke, A. Thévenin, J. Stoye, R. K. Hartmann, S. J. Prohaska, and P. F. Stadler. Orthology detection combining clustering and synteny for very large datasets. *PLoS ONE*, 9(8):e105 015, 2014.

[66] E. Lerat, V. Daubin, and N. A. Moran. From gene trees to organismal phylogeny in prokaryotes: The case of the γ-proteobacteria. *PLoS Biol.*, 1(1):e19, 2003.

[67] H. Li. TreeFam: A curated database of phylogenetic trees of animal gene families. *Nucleic Acids Res.*, 34(90001):D572–D580, 2006.

[68] L. Li, C. J. Stoeckert, and D. S. Roos. OrthoMCL: Identification of ortholog groups for eukaryotic genomes. *Genome Res.*, 13(9):2178–2189, 2003.

[69] X. Ling, X. He, and D. Xin. Detecting gene clusters under evolutionary constraint in a large number of genomes. *Bioinformatics*, 25(5):571, 2009.

[70] J. Liu and B. Rost. Domains, motifs and clusters in the protein universe. *Curr. Opin. Chem. Biol.*, 7(1):5–11, 2003.

[71] A. Lomsadze, P. D. Burns, and M. Borodovsky. Integration of mapped RNA-Seq reads into automatic training of eukaryotic gene finding algorithm. *Nucleic Acids Res.*, 42(15):e119, 2014.

[72] F. V. Martinez, P. Feijão, M. D. V. Braga, and J. Stoye. On the family-free DCJ distance. In *Proc. of WABI 2014*, vol. 8701 of *LNCS*, 174–186. 2014.

[73] D. P. Mindell, M. D. Sorenson, and D. E. Dimcheff. Multiple independent origins of mitochondrial gene order in birds. *Proc. Natl. Acad. Sci. U.S.A.*, 95(18):10 693–10 697, 1998.

[74] A. Mira, L. Klasson, and S. G. E. Andersson. Microbial genome evolution: sources of variability. *Curr. Opin. Microbiol.*, 5(5):506–512, 2002.

[75] J. Misra and D. Gries. A constructive proof of Vizing's theorem. *Inform. Process. Lett.*, 41(3):131–133, 1992.

[76] B. M. Moret, L. S. Wang, T. Warnow, and S. K. Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17 Suppl 1:S165–73, 2001.

[77] C. L. Morrison, A. W. Harvey, S. Lavery, K. Tieu, Y. Huang, and C. W. Cunningham. Mitochondrial gene rearrangements confirm the parallel evolution of the crab-like form. *Proc. Biol. Sci.*, 269(1489):345–350, 2002.

[78] J. H. Nadeau and B. A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proc. Natl. Acad. Sci. U.S.A.*, 81(3):814–818, 1984.

[79] T. Ohta. *Gene Families: Multigene Families and Superfamilies*. John Wiley & Sons, Ltd, 2001.

[80] G. Ostlund, T. Schmitt, K. Forslund, T. Köstler, D. N. Messina, S. Roopra, O. Frings, and E. L. L. Sonnhammer. InParanoid 7: New algorithms and tools for eukaryotic orthology analysis. *Nucleic Acids Res.*, 38(Database issue):D196–D203, 2010.

[81] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comp. Sys. Sci.*, 43(3):425–440, 1991.

[82] P. M. Pardalos and N. Desai. An algorithm for finding a maximum weighted independent set in an arbitrary graph. *Int. J. Comput. Math.*, 38(3–4):163–175, 1991.

[83] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Elec. Colloq. on Comput. Complexity*, 71:5, 1998.

[84] C. Pesquita, D. Faria, H. Bastos, A. E. Ferreira, A. O. Falcão, and F. M. Couto. Metrics for GO based protein semantic similarity: a systematic evaluation. *BMC Bioinformatics*, 9(Suppl 5):S4, 2008.

[85] S. Powell, D. Szklarczyk, K. Trachana, A. Roth, M. Kuhn, *et al.* eggNOG v3.0: Orthologous groups covering 1133 organisms at 41 different taxonomic ranges. *Nucleic Acids Res.*, 40(D1):D284–D289, 2011.

[86] S. Rahmann and G. W. Klau. Integer linear programs for discovering approximate gene clusters. In *Proc. of WABI 2006*, vol. 4175 of *LNBI*, 298–309. 2006.

[87] A. Rajaraman, E. Tannier, and C. Chauve. FPSAC: Fast phylogenetic scaffolding of ancient contigs. *Bioinformatics*, 29(23):2987–2994, 2013.

[88] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4(4):406–425, 1987.

[89] H. Salgado, M. Peralta-Gil, S. Gama-Castro, A. Santos-Zavaleta, L. Muñiz-Rascado, *et al.* RegulonDB v8.0: Omics data sets, evolutionary conservation, regulatory phrases, cross-validated gold standards and more. *Nucleic Acids Res.*, 41(Database issue):D203–D213, 2013.

[90] S. L. Salzberg, A. L. Delcher, S. Kasif, and O. White. Microbial gene identification using interpolated Markov models. *Nucleic Acids Res.*, 26(2):544–548, 1998.

[91] D. San Mauro, D. J. Gower, R. Zardoya, and M. Wilkinson. A hotspot of gene order rearrangement by tandem duplication and random loss in the vertebrate mitochondrial genome. *Mol. Biol. Evol.*, 23(1):227–234, 2006.

[92] D. Sankoff. Edit distances for genome comparisons based on non-local operations. In *Proc. of CPM 1992*, vol. 644 of *LNCS*, 121–135. 1992.

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

[94] D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *J. Comp. Biol.*, 5:555–570, 1998.

[95] D. Sankoff, D. Bryant, M. Deneault, B. F. Lang, and G. Burger. Early eukaryote evolution based on mitochondrial gene order breakpoints. *J. Comput. Biol.*, 7(3–4):521–535, 2000.

[96] D. Sankoff, R. Cedergren, and Y. Abel. Genomic divergence through gene rearrangement. vol. 183 of *Meth. Enzymol.*, 428 – 438. Academic Press, 1990.

[97] D. Sankoff and N. El-Mabrouk. Duplication, rearrangement and reconciliation. In *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map alignment and the Evolution of Gene Families*, vol. 1 of *Comput. Biol. Series*, 537–550. Springer Netherlands, 2000.

[98] D. Sankoff and P. Trinh. Chromosomal breakpoint reuse in genome sequence rearrangement. *J. Comput. Biol.*, 12(6):812–821, 2005.

[99] T. Schmidt and J. Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Proc. of CPM 2004*, vol. 3109 of *LNCS*, 347–358. 2004.

[100] R. Shao, M. Dowton, A. Murrell, and S. C. Barker. Rates of gene rearrangement and nucleotide substitution are correlated in the mitochondrial genomes of insects. *Mol. Biol. Evol.*, 20(10):1612–1619, 2003.

[101] G. Shi, M.-C. Peng, and T. Jiang. MultiMSOAR 2.0: An accurate tool to identify ortholog groups among multiple genomes. *PLoS ONE*, 6(6):e20 892, 2011.

[102] J. S. J. Shi and J. T. J. Tang. An experimental evaluation of corrected inversion and DCJ distance metric through simulations. *Int. Conf. Bioinform. Biomed. Eng.*, 1–4, 2010.

[103] N. Song, R. D. Sedgewick, and D. Durand. Domain architecture comparison for multidomain homology identification. *J. Comput. Biol.*, 14(4):496–516, 2007.

[104] M. Stanke and S. Waack. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*, 19 Suppl 2:ii215–25, 2003.

[105] J. Tang and B. M. E. Moret. Phylogenetic reconstruction from gene-rearrangement data with unequal gene content. In *Proc. of WADS 2003*, vol. 2748 of *LNCS*, 37–46. 2003.

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

[107] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, *et al.* The COG database: An updated version includes eukaryotes. *BMC Bioinformatics*, 4:41, 2003.

[108] I. H. Toroslu and G. Üçoluk. Incremental assignment problem. *Inform. Sciences*, 177(6):1523–1529, 2006.

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

[110] V. G. Vizing. On an estimate of the chromatic class of a *p*-graph. *Diskret. Analiz No.*, 3:25–30, 1964.

[111] R. M. Waterhouse, E. M. Zdobnov, F. Tegenfeldt, J. Li, and E. V. Kriventseva. OrthoDB: The hierarchical catalog of eukaryotic orthologs in 2011. *Nucleic Acids Res.*, 39(Database issue):D283–8, 2011.

[112] G. Watterson, W. Ewens, T. Hall, and A. Morgan. The chromosome inversion problem. *J. Theor. Biol.*, 99(1):1–7, 1982.

[113] S. Whelan and N. Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Mol. Biol. Evol.*, 18(5):691–699, 2001.

[114] A. W. Xu and B. M. E. Moret. GASTS: Parsimony scoring under rearrangements. In *Proc. of WABI 2011*, vol. 6833 of *LNBI*, 351–363. 2011.

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

[116] Z. Yin, J. Tang, S. Schaeffer, and D. Bader. A Lin-Kernighan heuristic for the DCJ median problem of genomes with unequal contents. In *Proc. of COCOON 2014*, vol. 8591 of *LNCS*, 227–238. 2014.

[117] M. Zhang and H. W. Leong. Identifying positional homologs as bidirectional best hits of sequence and gene context similarity. In *Proc. of ISB 2011*, 117–122. 2011.

[118] Q. Zhu, Z. Adam, V. Choi, and D. Sankoff. Generalized gene adjacencies, graph bandwidth, and clusters in yeast evolution. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 6(2):213–220, 2009.