Universität Bielefeld Technische Fakultät

Phylogeny-based Analysis of Gene Clusters

Ph. D. Thesis

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

by

Roland Wittler

January 11, 2010

Supervisor:	Prof. Dr. Jens Stoye
Referees:	Prof. Dr. Robert Giegerich Prof. Dr. Jens Stoye

Sparsamkeit ist eine Zier, doch weiter kommst du ohne ihr.

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

Zusammenfassung

Durch die Erforschung von verwandtschaftlichen Beziehungen, der Phylogenie, verschiedener Spezies erlangen wir umfassende Informationen über deren Evolution. Ein Ansatz für die Rekonstruktion phylogenetischer Szenarien oder zum Vergleich von Genomen ist die Untersuchung des Erbguts auf Ebene der DNA-, RNA- oder Proteinsequenzen. Eine weitere Herangehensweise ist das Studium der Struktur der Genome. Auf dieser höheren Ebene betrachtet man oft die Reihenfolge der Gene innerhalb des Genoms.

Um verschiedene Genome auf die Anordnung der enthaltenen Gene zu abstrahieren, werden diese zunächst miteinander verglichen, um zu bestimmen, welche Gene einander entsprechen. Anschließend wird jeder dieser Genfamilien ein eindeutiger Bezeichner, z.B. eine Nummer, zugeordnet, so dass schließlich jedes Genom durch eine Abfolge von Genbezeichnern repräsentiert werden kann. Nimmt man an, dass die zu untersuchenden Genome genau dieselben Gene enthalten und jedes davon genau einmal, so lässt sich die Genfolge formal recht einfach als Permutation modellieren. Diese Einschränkung ermöglicht zwar mathematisch elegante und effiziente Analyseverfahren, gibt jedoch die biologische Realität nur selten angemessen wieder. Hingegen führen realistischere Modelle, welche allgemeinere Sequenzen von Genen zulassen, oft zu komplexen Methoden mit inpraktikablen Laufzeiten. Desweiteren berücksichtigen Varianten dieser Modelle auch die Orientierung der Gene, also die Information auf welchem der beiden DNA-Stränge ein Gen liegt.

Evolutionäre Prozesse führen zu strukturellen Veränderungen im Genom und somit zu einer Umordnung der Genfolgen. Die Genzusammenstellung in einigen Bereichen bleibt dabei jedoch erhalten und findet sich in mehreren, verwandten Genomen. Diese Abschnitte mit den darin enthaltenen Genen werden als *Gencluster* bezeichnet. Studien konnten zeigen, dass solche konservierten Regionen oftmals Gene enthalten, deren Funktionen einem gemeinsamen Kontext zuzuordnen sind oder die evolutionär zusammenhängen. So kann das Auffinden und der Vergleich von Genclustern wertvolle Hinweise auf die Funktion oder die evolutionäre Entwicklung von Genen geben.

In der Literatur werden zahlreiche formale Definitionen für Gencluster, basierend auf verschiedenen Modellen für Genordnungen, diskutiert und analysiert. In dieser Forschungsarbeit wird ein allgemeines Konzept von Genclustern eingeführt, welches einerseits so flexibel ist, dass es eine Vielzahl von Gencluster-Modellen umfasst, aber andererseits konkret genug ist, um Folgedefinitionen, theoretische Betrachtungen und algorithmische Vorgehensweisen darauf aufzubauen.

Wenn die Genome mehrerer Spezies denselben Gencluster aufweisen, wurde dieser vermutlich von einem gemeinsamen Vorfahren geerbt. Ein Kernpunkt dieser Arbeit ist die Entwicklung und Untersuchung einer Methode, welche aus den Genordnungen heute lebender Spezies auf Basis ihrer Phylogenie Gencluster ihrer Vorfahren rekonstruiert. Die Betrachtung von Genclustern in einem evolutionären Zusammenhang gibt Aufschlüsse über die strukturelle Entwicklung von Genomen, die Evolution von Genclustern im Allgemeinen, aber auch die Entstehung und Entwicklung von bestimmten Genclustern.

Ein Kriterium, das für die Rekonstruktion zugrunde gelegt wird, ist das sog. *Parsimony*-Prinzip (engl. für Sparsamkeit), welches besonders im Kontext der Phylogenie oft Verwendung findet. Bei diesem Verfahren wird ein Szenario gesucht, welches die beobachteten Daten mit möglichst wenigen evolutionären Ereignissen erklärt. In diesem Fall sollen den Vorfahrspezies Gencluster so zugeordnet werden, dass die Anzahl an Neuentstehungen und Verlusten von Clustern, die sich duch das Szenario ergeben würden, minimal ist. Eine solche Zuordnung lässt sich mit Hilfe bekannter Methoden, z.B. des *Fitch-Hartigam*-Algorithmus, effizient bestimmen.

Abhängig vom verwendeten Modell für die Genordnung und Gencluster, können die auf diese Weise für einen Vorfahren bestimmten Gencluster Widersprüche aufweisen. Das heißt, es gäbe keine in diesem Modell gültige Anordnung von Genen, die alle rekonstruierten Cluster enthält. Dies legt ein weiteres Kriterium nahe: die Forderung nach *Konsistenz*. Die Überprüfung dieser Eigenschaft ist jedoch eine komplexe Aufgabe. Im Rahmen dieser Arbeit wurde für verschiedene Modelle überprüft, wie bzw. ob dieses Problem effizient gelöst werden kann. Die Resultate reichen von Algorithmen, welche die Eigenschaft der Konsistenz für das Modell der *konservierten Nachbarschaften* mit linearem Zeitaufwand prüfen, bis zum Nachweis der NP-Vollständigkeit für komplexere Modelle, die beweisen, dass die Überprüfung der Konsistenz zu einer Klasse von sehr schwer lösbaren Problemen gehört.

Als Zielsetzung der Rekonstruktion wird eine Kombination aus beiden Eigenschaften definiert. Unter allen konsistenten Zuordnungen wird eine solche gesucht, welche die Anzahl an Entstehungen und Verlusten von Genclustern minimiert. Es konnte ein wichtiger Zusammenhang zwischen beiden Kriterien gezeigt werden, auf welchem der entwickelte Rekonstruktionsalgorithmus aufbaut. Dieser berechnet exakte Ergebnisse in praktikablen Laufzeiten. Eine umfassende Evaluation auf Basis simulierter Daten zeigte, dass unabhängig vom Modell nahezu alle rekonstruierten Gencluster korrekt waren, und je nach simulierter evolutionärer Distanz der Spezies hinreichend viele der theoretisch vorhandenen Cluster gefunden wurden.

Die Anwendung der Methode auf Genomdaten von Coryne-Bakterien unter der Verwendung von verschiedenen Genclustermodellen zeigte unter anderem, dass die Ergebnisse nur sehr wenig vom verwendeten Modell abhängen, da die Reihenfolge der Gene in konservierten Bereichen meist vollständig erhalten war. In Zusammenarbeit mit Biologen konnte bestätigt werden, dass die rekonstruierten, evolutionären Szenarien sinnvoll sind. Darüber hinaus konnte exemplarisch gezeigt werden, dass die erhaltenen Ergebnisse neue Fragestellungen aufwerfen, auf Fehler in den Eingabedaten hinweisen und für weiterführende Analysen verwendet werden können.

Abstract

The exploration of the ancestral history, the phylogeny, of different species can raise valuable information about their evolution. One approach for the reconstruction of phylogenetic scenarios or for the comparison of genomes is the examination of the genetic material on the level of the DNA, RNA or protein sequences. Another possibility is to study the genomic structure. On this higher level, one commonly considers the order of the genes within the genome.

To abstract different genomes to the arrangement of their contained genes, these are first compared to determine which genes correspond to each other. Then, a unique identifier, for example a number, is assigned to each of the obtained gene families such that, finally, every genome can be represented by a sequence of gene identifiers. When we assume that all considered genomes contain exactly the same genes and each gene exactly once, gene orders can be formalized simply by permutations. This restriction offers mathematically elegant and efficient analytical methods. However, biological reality can rarely be depicted adequately. On the contrary, more realistic models which allow more general sequences of genes often involve complex methods with impractical running times. Apart from that, variants of these models also incorporate gene orientation, i.e. the information on which of the two DNA strands a gene is located.

Evolutionary processes can modify a genome structurally and rearrange its gene order. The gene composition of some regions, however, is preserved and can be found in several related genomes. These segments and the contained genes are denoted as *gene clusters*. Studies revealed that such conserved regions often comprise functionally or phylogenetically related genes. The identification and comparison of gene clusters can therefore give valuable hints on their function or the evolution of genes.

Various formal definitions of gene clusters based on different models of gene order are discussed and analyzed in the literature. In this thesis, we introduce a general concept of gene clusters which, on the one hand, is flexible enough to cover a variety of gene clusters models and, on the other hand, is concrete enough to build the basis for follow-up definitions, theoretical considerations and algorithmical approaches.

If the genomes of several species comprise the same gene cluster, it was probably inherited from a common ancestor. A major aim of this project is the development and evaluation of a method that reconstructs ancestral gene clusters from the gene order of contemporary species based on their phylogeny. The analysis of gene clusters in an evolutionary context offers clues about the structural development of genomes, the evolution of gene clusters in general, but also about the origin and development of specific clusters.

One criterion the reconstruction is based on is the so-called *parsimony principle*, which is commonly used, especially in the field of phylogenetics. The objective of this approach is a scenario which explains the observed data by assuming as few evolutionary events as possible. Here, we want to assign gene clusters to ancestral species in such a way that the number of gains and losses of clusters implied by the scenario is minimized. Such an assignment can be determined efficiently by well known algorithms, for instance the method by Fitch and Hartigan.

Depending on the underlying model of gene order and gene clusters, the clusters obtained for one ancestor in this manner can be conflicting. That means, in the considered model, there is no putative order of genes that contains all reconstructed clusters. Thus, a further criterion should be considered: the requirement of *consistency*. However, the verification of this property is a complex task. Within the scope of this thesis, we discuss whether and how this problem can be solved efficiently for different models. Our results range from algorithms which verify consistency for the model of *conserved adjacencies* in linear time to the confirmation of NP-completeness for more complex models proving that the verification of consistency belongs to a class of very hard problems.

We define the objective of the reconstruction as a combination of both properties. Among all consistent assignments, we seek for those which minimize the number of gains and losses of gene clusters. We revealed an important connection between both criteria. Based on this, we developed an algorithm that finds exact solutions in practical running times. A broad evaluation on simulated data showed that, independent of the considered model, almost all reconstructed clusters were correct and, depending on the simulated evolutionary distances of the species, adequately many of the theoretically present clusters were found.

We applied the method on genomic data of Corynebacteria using different gene cluster models. This study indicates that reconstruction results hardly depend on the utilized gene cluster model since the order of genes in conserved segments were preserved exactly in most cases. In cooperation with biologists, we confirmed evolutionary scenarios reconstructed by our method to be reasonable. Furthermore, we exemplified how our results can raise new open questions, indicate errors in the input data, and can be used for further analyses.

Contents

Chapter 1

Introduction

The genome of an organism comprises all inheritable information for its development and the formation of specific properties. It is encoded in DNA molecules – two parallel chains composed of *nucleotides*, where opposite nucleotides build complementary base pairs: *adenine* always pairs with *thymine*, and *cytosine* with *guanine*. These long double strands form *chromosomes*, which can be circular or linear. The entire genome of an organism is composed of one or more such chromosomes and is contained in all of its cells. There, *genes* and other coding regions of the DNA are read (transcribed), and used to produce proteins or functional RNAs, which in turn are responsible for the structure and function of the cell and hence the whole organism.

All cellular life forms can be classified into three domains: *Bacteria*, *Archaea*, and *Eucariota*. Organisms of the first two domains are *prokaryotic*. That means, their genome is contained in the so-called *nucleoid* of the cell. A typical prokaryotic genome consists of one circular chromosome of length one to twelve million base pairs covering about one to twelve thousand genes. Extra-chromosomal DNA molecules, called *plasmids*, can contain further genes. In contrast, the structure of *eukaryotic* genomes is more complex. Here, the DNA is embedded into the *nucleus*, which is enclosed within a membrane. It is composed of several circular or linear chromosomes. Additionally, other compartments of the cell, like *mitochondria* or *chloroplasts*, can contain separate, small genomes. The size of eukaryotic genomes ranges from about twelve million base pairs in yeast to 14 billion base pairs in broad bean comprising about six thousand to 40 thousand genes, respectively. For comparison, the human genome is of length three billion base pairs, containing about 30 thousand genes. Further characteristics are a complex partitioning into noncoding and coding subsegments of the genes, the *intron-exon* structure, and multiple (*diploid* or *polyploid*) copies of chromosomes.

During evolution, a genome is changed by mutations in a random way. On the one hand, substitutions, deletions or insertions of single nucleotides can modify the encoding of a gene. On the other hand, parts of the genome can be rearranged by large scale operations, like for instance *inversions*, where a segment of a chromosome is reversed, *block interchanges*, where a segment is cut out and inserted at another position, the exchange of chromosome ends, called *translocations*, or *fusions* and *fissions* of chromosomes.

Both small scale and large scale events change, evoke or abolish properties of the affected organism. Thus, due to natural selection, some of these genetic and genomic modifications get fixed or lost in a population over time. It is known for a long time that small mutations coincide with certain diseases, like diabetes or heart attack. Recent work also reveals that genome rearrangements play an important role in cancer development [5, 124, 127, 139, 141, 146].

With more and more genomes being completely sequenced (see the GOLD database [116] for an overview of sequencing projects), there is an increasing need for computational, efficient methods to analyze genomic sequences. In the young field of *comparative genomics*, one tries to identify differences and similarities among several genomes to understand their function and evolutionary processes.

One approach is to study the genomes on the small scale level of DNA, RNA or protein sequences. For this purpose, parts of the genomes are compared, for example, to reconstruct mutation events, to estimate evolutionary distances and relationships, or to classify characteristics for ‘healthy’ and ‘ill’ individuals based on single nucleotide polymorphisms (SNPs) or repeat copy number analyses.

A second approach, on a large scale level, is whole-genome comparison based on the order of the genes within the genomes. In the next section, we will discuss methods to obtain gene ordering information and different ways to model genomes correspondingly. Then, in the subsequent section, we will give a short overview on how this abstract representation can be used for different further analyses, like *genome rearrangement* studies or the detection of *gene clusters* to gain knowledge about evolutionary processes or functional coherence of genes.

1.1 Modeling Gene Order

When we want to analyze a genome by means of its genes, we firstly have to identify the genes in the genomic sequence. Since protein-coding regions mostly show certain characteristics, probabilistic approaches can be utilized to find these genes. Prevalent gene prediction methods are based on hidden Markov models. Popular tools are GLIMMER [53, 142] and GeneMark [23, 32] for procaryotic genomes, and GENSCAN [36] and geneid [26, 131] for eukaryotic genomes. Beside these *ab initio* approaches, one can also search for genomic segments that are similar to sequences already known to be coding in other organisms. To this end, databases, such as Ensembl [95] and RefSeq [136], are set up to store gene sequences. Systems like GenDB [121] even combine a variety of different tools to predict and annotate genes in newly sequenced genomes. Eventually, biological experiments are needed to finally verify computationally detected genes.

To compare genomes on the basis of the order of their genes, usually a *homology assignment* is applied. This means, one tries to identify which genes originated from a common ancestral gene and therefore belong to the same gene family. After that, a unique identifier, mostly a number, is assigned to each of these families, which allows to represent each genome as a sequence of the corresponding identifiers. The overall abstraction procedure is illustrated in Figure 1.1.

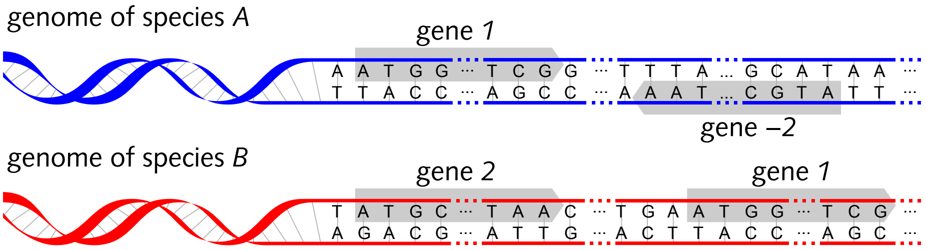

genome of species *A*
gene 1
A A T G G ··· T C G G ··· T T T A ... G C A T A A ···
T T A C C ··· A G C C ··· A A A T ... C G T A T T ···
gene −2
genome of species *B*
gene 2
gene 1
T A T G C ··· T A A C ··· T G A A T G G ··· T C G ···
A G A C G ··· A T T G ··· A C T T A C C ··· A G C ···

Figure 1.1: Illustration of gene-order-based genome modeling. The genome of the species A and B contain, beside others, homologous genes of the families 1 and 2 whereas the order and orientation of the genes differ. Genome A is represented as $(\ldots, 1, \ldots, -2, \ldots)$ and genome B as $(\ldots, 2, \ldots, 1, \ldots)$.

The concept of *homology* is further subdivided. When two genes originate from a common ancestor by gene duplication, these duplicates are called *paralogous* genes or, in short, *paralogs*. Two genes in different species which do not derive from gene duplication but originate from a single gene in an ancestral species are called *orthologous* genes or *orthologs*, respectively. Note that, even though paralogs originally occurred within one genome, they can later be spread across different species. Hence, homologous genes in different species are not necessarily orthologs. Apart from this original distinction, one commonly calls two homologous genes *paralogs* if they occur in the same genome and *orthologs* otherwise. To simplify matters, we will stick to the latter, informal yet practical definition.

1.1.1 Homology Assignment

To estimate homology relations among a set of genes, their sequences – either in terms of DNA or amino acids – are compared. For each pair of genes, some similarity (or distance) value is determined. Here, most methods use the tool BLAST [2] or one of its variants to compute pairwise local alignments, which in turn imply similarity measures like the BLAST E-value.

Based on these values, families of putatively homologous genes are constructed in such a way that the similarity between genes within the same family is high whereas the similarity across different families is low. For this purpose, various approaches have been proposed. Many of these assign candidates to one family if they meet a certain similarity threshold and then refine the obtained clustering by merging or splitting families, or iteratively inserting or removing single genes [46, 65, 133, 150].

Some methods firstly seek for specific patterns of similarity to initially assign the genes to clusters. When the gene most similar to a is b and, *vice versa*, the gene most similar to b is a, this is called a *reciprocal best hit*. The methods underlying the well-known COG database [154, 155] and the recent eggNOG database [100, 122] follow the idea that triangles of reciprocal best hits across distantly related genomes correspond to orthologs. Certain triples are then merged to larger families.

Other approaches use hierarchical clustering methods, like single-linkage clustering, as a first step to create a tree that represents the distances of all the gene sequences. Then, this complete tree is separated into several subtrees that correspond to the gene families [107, 135, 163].

There are many further methods, for instance based on Markov clustering [45, 64], random walks [128], transitive graph clustering [161] or tree reconciliation [95, 138].

The increasing amount of tools for homology assignment indicates, on the one hand, the need for such tools. On the other hand, it shows that this task is not solved sufficiently yet. A major drawback of most of the aforementioned approaches is that they have been used to compute gene families for a specific set of genomes. Although some of these databases provide the functionality to compare and represent a 'foreign' genome in terms of gene families, this is always limited to the families found in the underlying genomes.

Apart from this practical disadvantage, there is also a more fundamental problem. Most homology assignments are based on some threshold on the similarity or other parameters. As exemplified in Chapter 6, the setting of such parameters can influence the gene families we obtain, and thus the results of subsequent studies highly depend on the preprocessing. More general, since all discussed methods can only *predict* homology, we should never rely on the correctness of their results and bear this in mind when drawing conclusions built on predicted gene families.

1.1.2 Basic Models

After assigning each gene in all genomes under consideration to a gene family, we can abstract a genome to its gene order by simply representing it as an ordered list of gene families – strictly speaking, as a *sequence* over the set of gene families. Please note that, throughout this thesis, the term *set* is generally understood to refer to a *finite set*.

Definition 1. (Sequence) *Let O be a set. Then, a* sequence *over O is any concatenation of elements from O. The* length *of a sequence s is the number of elements in s, denoted by $|s|$. A sequence s is written as $(s_1, s_2, \dots, s_{|s|})$ with $s_i \in O$ for all $i = 1, \dots, |s|$. Any sequence $s' = (s_i, s_{i+1}, \dots, s_j)$ with $1 \le i \le j \le |s|$ is a* substring *of s.*

Conventionally, gene families are represented by some identifier, for instance a number. Hence, we model genomes as sequences over $\{1, \dots, N\}$, where N is the number of different gene families found in all genomes. We assume homologous genes to perform the same biological function and thus consider them to be 'equal'. Therefore, to simplify matters, we shortly use the term 'gene a', instead of 'gene assigned to gene family a'. Further, throughout this thesis, the terms *genome* and *gene order* are used interchangeably unless otherwise noted.

In the course of evolution, some genes may get lost in a lineage. Since a sequence over O does not have to contain all elements of O, such gene deletions are covered when we model genomes as sequences. Moreover, paralogs are included as well, because a sequence is allowed to possess the same element several times. However, this flexibility of the genome model often involves impractical runtime complexities of approaches based on this model.

In contrast, the following simple mathematical structure allows for the development of elegant and efficient algorithms for many purposes. It is therefore often used in gene-order-based comparative genomics.

Definition 2. (Permutation) *Let $O = \{o_1, \ldots, o_N\}$ be a set of size N. Then, a* permutation *π of O is a bijection $O \longrightarrow O$ written as $\big(\pi(o_1), \pi(o_2), \ldots, \pi(o_N)\big)$. Any sequence $s' = \big(\pi(o_i), \pi(o_{i+1}), \ldots, \pi(o_j)\big)$ with $1 \leq i \leq j \leq N$ is a* substring *of π.*

When we model genomes as permutations, this corresponds to gene orders in which each gene occurs exactly once. Although, for instance the genome structures of viruses or organelles, like chloroplasts or mitochondria, often conform to permutations, genomic data in general disconfirms these strong restrictions. Paralogous genes and unequal gene content among different organisms preclude modeling gene order as permutations directly.

There are several possibilities to overcome this obstacle and to obtain permutations from given data. Instead of using the whole genome, one could only consider parts of the genomes for which the restrictions hold. Or, instead of the actual order of genes, one can try to find other, short genome segments which can be used as appropriate markers. Pevzner and Tesler [134] suggested a formal procedure to divide the whole genomic sequence into blocks such that the genomes can finally be represented as permutations of these blocks. Another approach is to compute the so-called *core genome*, the set of genes that are possessed in all given genomes, and to use only those genes of each genome whose sequences are most similar to the corresponding representative. This can be an artificial consensus sequence or the sequence of the gene as it occurs in a chosen reference genome.

A further method to cope with multiple gene occurrences, which is more robust against the choice of a similarity measure or a reference genome, was proposed by Sankoff [144]. In each genome, one occurrence of each gene is chosen as the so-called *exemplar* and all other occurrences are discarded. For the resulting permutations, the problem – in Sankoff's study the minimization of a rearrangement distance – is solved. Among all possible choices of exemplars, we search for a combination that yields the best overall result. The number of possible exemplar choices is exponentially growing with the number of genomes, which indicates that this method is infeasible for many concerns. Indeed, this minimization problem was shown to be NP-hard for a specific optimization objective (the inversion distance computation), where only two genomes are under investigation [35].

Instead of keeping only one gene copy per paralogous gene family in each genome, Tang and Moret [153] suggested to consider all copies in a *matching* framework. To obtain permutations, a unique identifier is assigned to each copy whereas all possible assignments are enumerated. For instance, if two genomes g and h contain two copies of gene a each, we compare genome $g = (\ldots, a_1, \ldots, a_2, \ldots)$ and h under two matchings: $h = (\ldots, a_1, \ldots, a_2, \ldots)$ and $h = (\ldots, a_2, \ldots, a_1, \ldots)$. Since the number of matchings grows exponentially with the copy number of the paralogs and the number of genomes under consideration, this approach is severely limited. Nevertheless, it has been successfully applied on real data, for example to analyze the gene order of γ-proteobacteria [27].

In general, we lose information by applying any of the above procedures to map complex gene order data onto simple permutation by neglecting either genes or paralogy information. Furthermore, removing gene copies can also imply new, false information about co-location of genes. However, as we will see in Chapter 5, some computations would not be feasible without such simplifying preprocessing.

1.1.3 Model Variants

The double-stranded genome can be transcribed in two directions, and a gene can be either encoded in one reading direction on the *forward* strand or in the opposite direction on the *reverse* strand of the genome. A convenient way to also include this orientation of the gene within the genome is to use *signed sequences* or *signed permutations*, respectively. Here, we assign either a plus or minus sign to each element of a sequence (or permutation, respectively), depending on the genome strand the corresponding gene lies on.

Definition 3. (Signed Sequence) *Let O be a set. Then, a* signed sequence *over O is any concatenation of elements from $\{+o, -o \mid o \in O\}$.*

Definition 4. (Signed Permutation) *Let $O = \{o_1, \ldots, o_N\}$ be a set of size N. Then, $\big(\sigma_1\pi(o_1), \sigma_2\pi(o_2), \ldots, \sigma_N\pi(o_N)\big)$ is a* signed permutation *if $\pi = \big(\pi(o_1), \pi(o_2), \ldots, \pi(o_N)\big)$ is a permutation over O and $\sigma_i \in \{+, -\}$ for $1 \leq i \leq N$.*

Conventionally, the plus sign is omitted when writing a signed sequence or signed permutation. The notions of *length* and *substring* apply analogously to the unsigned case.

Prokaryotic genomes are often composed of circular chromosomes. These can be modeled by circular sequences (or permutations, respectively), which can be written and read starting from any position:

$$\begin{aligned} &(s_1, s_2, \ldots, s_{i-1}, s_i, \ldots, s_{|s|}) \\ &= (s_i, \ldots, s_{|s|}, s_1, s_2, \ldots, s_{i-1}) \text{ for any } 1 \leq i \leq |s|\,. \end{aligned}$$

If a genome is composed of multiple chromosomes, these can be concatenated to one single sequence. When we add unique symbols which represent the chromosomes' extremities, the *telomeres*, we separate the sequence into substrings corresponding to the individual chromosomes. If necessary, empty chromosomes can be added to ensure the same number of telomere symbols, as shown in the following example. Note that throughout this thesis, the end of an example is indicated by the symbol ◇.

Example 1. Let A, B and C be three genomes composed of several chromosomes:

$$\begin{aligned} A &= \boxed{\quad a_1 \quad}\ \boxed{\ a_2\ }\ \boxed{\ a_3\ } \\ B &= \boxed{\ b_1\ }\ \boxed{\ b_2\ }\ \boxed{b_3} \\ C &= \boxed{\quad c_1 \quad}\ \boxed{\ c_2\ }\,. \end{aligned}$$

Then, we construct the following sequences:

$$\begin{aligned} S_A &= \Big(t_1, \boxed{\quad a_1 \quad}, t_2, \boxed{\ a_2\ }, t_3, \boxed{\ a_3\ }, t_4\Big) \\ S_B &= \Big(t_1, \boxed{\ b_1\ }, t_2, \boxed{\ b_2\ }, t_3, \boxed{b_3}, t_4\Big) \\ S_C &= \Big(t_1, \boxed{\quad c_1 \quad}, t_2, \boxed{\ c_2\ }, t_3, t_4\Big)\,. \end{aligned}$$

◇

If, in each genome, all chromosomes together contain each gene exactly once, the above concatenation yields permutations. When a distinction between the telomeres is not needed, they are all denoted by the same symbol, conventionally $\circ$.

As we have seen, there are various ways to model genomes by the order of their genes where we have to trade off biological realism against mathematical simplicity. When genomes are analyzed with respect to their gene order, different models of gene orders are chosen, depending on the actual objective and the data under consideration. Some problem statements and approaches will be addressed in the following section.

1.2 Comparative Genomics

One of the first gene-order-based analyses in the field of whole-genome comparison was proposed by David Sankoff in 1992 [143]. He introduced the concept of modeling genomes as permutations to study their gene order in the context of *genome rearrangements*. Here, the evolution of genome structure is simulated by large-scale genome rearrangement operations. Following the concept of parsimony, the general aim is to transform one given genome into another, using as few operations as possible. The length of such a minimal sequence of transformations is used as an evolutionary distance measure.

Different studies vary in the considered rearrangement operations and whether the actual shortest transformation sequence or solely its length is computed. For further details on genome rearrangements, see the recent book by Fertin *et al.* [75], or the reviews by Bourque and Zhang [34] or Li *et al.* [114]. Due to a typically high problem complexity, gene-order-based rearrangement studies mostly build on permutations. For a discussion on different methods to handle duplicated genes in this context, see the book chapter by El-Mabrouk [63]. Recent approaches utilize genome sequence information rather than gene order and are not restricted to permutations [92, 117].

Another approach in whole-genome comparison based on mathematical modeling of gene order is the search for *gene clusters*. Unlike genome rearrangements, this concept does not depend on specific models for evolutionary events and is therefore also called *model free*. Nevertheless, rearrangement is still assumed in general. During evolution, rearrangement processes change the order of genes. But comparison of contemporary genomes shows that in some segments the gene order is conserved among different species.

The presence of these gene clusters may be explained by two phenomena: The evolutionary distance was too small to randomize the gene orders completely, or selective pressure prevented these segments from being broken. The first factor can again be used to infer evolutionary distances: The more clusters are found, the closer are related the genomes [152]. But, particularly, the second explanation is of great interest: In fact, it has been shown that conserved regions often contain functionally or evolutionary associated genes [29, 48, 51, 86, 110, 111, 126, 152]. Hence, the analysis of gene clusters can give clues about the function of genes and valuable insights in evolutionary processes like rearrangement processes or lateral gene transfer.

In this framework, gene order may also be modeled as permutations, like for genome rearrangement. But there are also approaches in which gene orders are represented as sequences of genes.

Based on these different representations for genomes, several formal definitions of gene clusters have been developed. A very basic and well-established example are *common intervals*. In this model, a cluster is a set of genes which occur contiguously in several genomes, where the ordering and orientation of the genes may differ. For an overview on respective problem statements and its solutions, see the review of Bergeron *et al.* [13]. Variants of this model, restricting the inner structure of the clusters, are for example *nested common intervals* [28, 91] or *framed common intervals* [19].

The above-mentioned models require the occurrences of the gene clusters in the genomes to be exact – neither appearance of foreign genes nor missing genes are permitted. A more general cluster model, *max-gap clusters*, also known as *gene teams*, allows that the cluster occurrences are interrupted by genes not belonging to the cluster to a certain degree [14, 86]. Recent studies on *approximate clusters* even incorporate missing genes and are thus more realistic [30]. For a discussion of different error-tolerant cluster models, possible representations and their properties, see the review by Hoberman and Durand [91] or the report by Chauve *et al.* [41].

1.3 Phylogeny

In phylogenetics, the ancestral history of species is studied. One major aim is the reconstruction of the evolutionary relationships of contemporary species, their *phylogeny*. Following Darwin [52], evolution proceeds in a tree-like fashion since all considered species descended from a common ancestor by an iterative process of *speciation*, which means separation into subpopulations which evolve differently, and eventually form new species by natural selection.

Mathematically, trees are formalized in a graph theoretic context. A *graph* G is a pair (V, E) of a set V containing *vertices*, also called *nodes*, and a set E of *edges*. In *undirected graphs*, edges are defined as two-element subsets of V, and an edge $\{u, v\}$ is interpreted to connect the vertices u and v. In *directed graphs*, edges are defined as (ordered) pairs over $V \times V$, and an edge (u, v) is interpreted to connect the two vertices by pointing from u to v. In both models, an edge containing a node v is said to be *incident* to v and two vertices connected by an edge are called *adjacent*. The *degree* of a vertex v is the number of edges incident to v. A *path* is a sequence of vertices $(v_1, v_2, \dots, v_l)$ where any two consecutive vertices v_i and v_{i+1} are connected by an edge. An undirected graph G is *connected* if, for all vertices $u, v \in V$, G contains a path starting in u and ending in v. A path is *simple* if no edge is used more than once. In directed graphs, all edges of a path have to point in the same direction.

Based on this concept, trees are defined as graphs with certain properties.

Definition 5. (Tree) *A* tree *is a graph in which any two nodes are connected by a unique simple path. A* leaf *is a node with degree one. All other nodes are* internal. *It is called* binary *if all internal nodes have degree three, and* multifurcating *otherwise.*

When we model the phylogeny of a set of species as a tree, the leaves correspond to the contemporary species, internal nodes represent ancestral species, and the edges connect ancestors with their descendants. Additionally, weights can be assigned to all edges, reflecting the evolutionary distance between the species. Usually, phylogenetic trees are binary. If a reconstructed phylogeny is multifurcating, this can have two reasons. Firstly, due to a lack of information, we cannot unravel all speciation events properly. Secondly, a population might have branched into more than two subpopulations at the same time, which is commonly assessed to be unlikely.

Since, in our context, we assume ancestor-descendant relationships, phylogenetic trees can be defined on the basis of directed graphs. The node that corresponds to the common ancestor of all species is called *root*.

Definition 6. (Rooted Tree) *A* rooted tree *is a tree that is a directed graph where one node is elected the* root*, and any other node is connected to the root by a unique simple path in which all edges point away from the root. It is called* binary *if the root has degree two and all other internal nodes have degree three, and* multifurcating *otherwise. For an edge* (u, v)*, node* u *is the* parent node *of* v *and* v *is a* child node *of* u*. If a node* u *lies on the path from the root to* v*, then* u *is an* ancestral node *of* v *and* v *is a* descendant *of* u*.*

Methods to reconstruct a phylogeny for a given set of species can be classified into two groups. *Distance-based* approaches build on precomputed distances between all pairs of species. The general objective is to find a weighted tree such that the distances between the leaves in this tree are preferably close to the given distances of the corresponding species. Commonly, the distances of the species are determined by comparing segments of the genomic sequences of the species. For instance, sequence alignment methods are used to estimate edit distances.

Species can also be compared in terms of *characters*, which are certain inheritable attributes of the species, for example the number of extremities or the nucleotide at a specific position of the genomic sequence. An actual quantity or quality of such a character in a species, for instance ‘4’ or ‘cytosine’, is called a *character state*. The aim of character-based reconstruction methods is to find a reasonable evolutionary scenario, which includes a tree and an assignment of character states to all internal nodes. Following the idea of *parsimony*, such a scenario should explain the character states observed in the contemporary species with a minimum amount of state changes along the edges of the phylogenetic tree. In contrast, *maximum likelihood* methods assume certain probabilities for a character to change from one state to another and seek for a most likely scenario.

Additionally to reconstructing the topology of a phylogenetic tree, character-based approaches also infer states for the ancestral species, which can be analyzed to study correlations or the evolution of different characters.

For a more detailed introduction to phylogenetics, we recommend the book “Inferring Phylogenies” by Felsenstein [74]. The concept of parsimony will be introduced in more detail in Section 2.2.2.

1.4 Overview of the Thesis

The aim of this thesis is to study gene clusters in a phylogenetic context.

In general, a basic goal of gene cluster analyses is to detect gene clusters that are conserved in two genomes. But usually, one wants to compare more than two genomes. Co-location of a set of genes in the genomes of two related species might occur just by chance. Therefore, more sophisticated methods seek for clusters which are contained in a set of genomes. However, certain clusters may not be conserved among all genomes under consideration but only in a subset. To cover such gene clusters as well, a quorum parameter q is introduced to specify gene clusters as co-location of genes in at least q genomes.

The approach taken in this thesis goes one step further and takes the phylogeny of the species into account. Given the evolutionary tree and the gene order of several contemporary species, we want to reconstruct putative gene clusters of the ancestral species. Beside the pure identification of gene clusters, we also obtain scenarios for the origin and development of the clusters. Studying the evolution of gene clusters can give valuable information about underlying evolutionary processes, the ancestral history of the species, and functional and evolutionary relations of genes.

As elucidated in the preceding sections, there are various definitions of gene clusters based on different models of gene orders discussed in the literature. An essential claim of our procedure is not being constrained to one certain formalization. To this end, we introduce a unifying, generalized model of gene clusters which generally includes all specific definitions. Further theoretical considerations and algorithmical approaches are solely based on this abstract gene cluster model. It builds an interface between our reconstruction framework and a variety of concrete gene cluster definitions.

We want to assign gene clusters to the ancestral nodes in the phylogenetic tree, which we call a *labeling*. Instead of following a methodological reconstruction approach, we define an objective to specify *optimal* labelings. We formalize a rating of labelings to ensure to obtain results which meet certain criteria – independently of the underlying concrete gene cluster model or method. Our definition of optimality combines two criteria.

Firstly, without regarding its actual shape, we consider any gene cluster as a binary character, specifying whether it is contained in a certain gene order or not. Then, a change of a character state corresponds to the gain or loss of a cluster. Following the concept of parsimony, a reconstructed assignment should involve as few character state changes as possible. That means, we desire a labeling that explains the clusters observed in the contemporary species with a minimal amount of gains and losses of gene clusters.

Secondly, depending on the underlying model of gene order and gene clusters, putative ancestral gene clusters derived from different contemporary species can imply contradicting ordering information, which we call a *conflict*. This would exclude the existence of any potential, ancient gene order that contains all these clusters. Hence, a natural aim for a reconstructed set of clusters is to rule out that it contains any conflicts, i.e. we want to ensure *consistency*.

Finally, under all consistent labelings those with a minimal amount of cluster gains and losses are defined to be optimal.

The goal of reconstructing consistent labelings was set by Bergeron *et al.* [10] who present an algorithm that reconstructs sets of framed common intervals which are consistent. But their approach does not optimize any objective function. Adam *et al.* [1] require parsimony as an objective function, but their algorithm does not always find an exact solution. In addition, a second heuristical step is added to reach consistency. Moreover, other methodological approaches have been developed and successfully applied to analyze certain datasets [24, 44, 58, 90, 99, 118, 119, 164]. However, on the one hand, they are based on specific cluster definitions, mostly adjacencies of genes, or assume certain rearrangement scenarios. On the other hand, many of these methods do either not solve an objective function exactly or do not follow any objective at all.

As a first result, we elucidate an important relation between the two optimization criteria: Any most parsimonious labeling can be modified to reach consistency and therefore optimality by deleting clusters which are involved in conflicts. To identify these in a given set of gene clusters, we introduce a first, simple algorithm to find all conflicting subsets. This is a general, central issue in the context of consistency. Based on this, as one of our main results, we present a method that always finds an optimal solution. It utilizes the Fitch-Hartigan approach [77, 85] to find a most parsimonious labeling of the tree. Then, a bounded search is used to transform this into an optimal solution. Since the search space can grow exponentially with respect to the genome lengths and we want to provide an exact result, this search strategy has an exponential worst-case time complexity. Two different search strategies are combined to increase the sensitivity.

The above findings and algorithms are based on the abstract gene cluster model. To integrate a concrete cluster definition into the framework while keeping the total runtime complexity within acceptable limits, we have to provide an efficient method to determine whether a given set of clusters is consistent or not.

We address various models and discuss if an integration is practically possible. In particular, we consider the most prominent concepts of gene clusters, namely *adjacencies* and *common intervals*, and two variants of the latter model, *framed* and *nested common intervals*. We study these models defined on permutations before we reconsider them defined on sequences. For each particular case, we first define it as an instance of the abstract model and then analyze if consistency can be verified efficiently. Our results range from simple, linear time solutions for adjacencies to NP-completeness for more general models.

We implemented the algorithmical framework including all models for which consistency can be verified in polynomial time. To certify the effectiveness of our method and to compare different cluster models, we perform a broad evaluation on simulated data. Due to the exactness of the algorithm, these results also reveal characteristics of our definition of optimality.

Finally, we exemplarily apply the reconstruction approach on genomic data of Corynebacteria to confirm the reconstructed evolutionary scenarios to be reasonable and informative.

Structure of the thesis. In Chapter 2, we introduce the general objective, including the abstract gene cluster model, the optimization criteria and their correlation. Based on this, we introduce our algorithmical framework to find optimal solutions, in Chapter 3. Then, we discuss several concrete gene cluster definitions: In Chapter 4, we introduce, integrate and evaluate permutation-based cluster models, and in Chapter 5, we reconsider these models defined on sequences. Then, we present results on real data in Chapter 6 before we finally conclude our study in Chapter 7.

Parts of this thesis, including the results of Chapters 2 to 4, have been published in advance [151]. An implementation of the developed method is available from the web site: `http://bibiserv.techfak.uni-bielefeld.de/rococo/`.

1.5 Acknowledgements

After studying genomic evolution on an abstract level for a long time, I would now also like to review my time as a PhD student from a wider perspective. I thank all the people who made this thesis possible and turned its development process into an invaluable period of my life.

First of all, I am very grateful for the extensive support of my adviser Jens Stoye. From him, I have learned a lot not only in research, science and academics but also for my personal life. Further, I thank Robert Giegerich for his support within and beyond the scope of the 'Graduiertenkolleg Bioinformatik'. Thanks go also to Andreas Tauch and Frank-Jörg Vorhölter for motivation and guidance from the biological side, and to Cedric Chauve for inspiration and fruitful discussions on the mathematical side.

Many special thanks go to all the former and current members of the Genome Informatics group who provided such a great atmosphere – in particular to our secretary Heike Samuel for handling many administrative affairs; to my good friend and office mate Peter Husemann for his encouragement, our discussions and all the fun we had; and to Inke Herms and Katharina Jahn for helping with words and deeds, and being good friends. I thank my family and especially my wife Sarah for taking great care of me during all the time.

Finally, I would like to acknowledge the 'DFG Graduiertenkolleg Bioinformatik' for three years of scholarship and the 'NRW Graduate School in Bioinformatics and Genome Research' for financial support. Besides funding, both institutions offered valuable scientific exchange during retreats and in seminars.

Chapter 2

Labeling a Phylogenetic Tree with Sets of Gene Clusters

We start this chapter with the theoretical basis for our approach to the reconstruction of ancestral gene clusters. Given a phylogenetic tree and the gene orders of contemporary species, we want to label the inner nodes of the tree with sets of gene clusters. Our optimization criterion in this context combines two properties: *consistency*, i.e. for each ancestral node, there must exist at least one potential gene order that contains all the reconstructed clusters, and *parsimony*, i.e. the number of gains and losses of gene clusters during evolution has to be minimal.

A central concept of our method is a general definition of gene clusters which unifies a variety of different gene cluster models discussed in the current literature. This abstract model is used as a black box such that the optimization approach is not restricted to any specific type of gene clusters.

The structure of this chapter is as follows. First, we introduce an abstract *gene cluster model* in Section 2.1. Based on this concept, we define *labelings* and the optimization criteria *consistency* and *parsimony* in Section 2.2. Finally, in Section 2.3, we formally state the labeling problem by combining the two criteria and show an important relation of the objectives, which is used to tackle the problem.

2.1 Abstract Gene Cluster Model

In genome comparison, conserved gene clusters are usually defined on sets of genomes. They specify patterns in the gene order that are found in all considered genomes or at least in some of them. However, for the moment, we define the presence of a cluster for just one genome. Later on, we will compare different genomes with respect to these occurrences in order to identify common structures and to analyze their ancestral history.

As stated in the Introduction, depending on the goal and the given data, or for computational reasons, many different definitions of gene clusters based on various genome models have been studied. Here, we define an abstract *gene cluster model* that is both general enough to allow several definitions of different models of gene clusters, and concrete enough to be used in the algorithm we will present. To fully describe any model of gene clusters formally, we essentially have to answer the following questions:

What is a genome? Since gene clusters reflect certain substructures of a gene order, we first have to specify the structure of the gene order itself to establish a basis for the actual gene cluster definition.

How does a gene cluster look like? Before we can search for clusters in a genome or reconstruct ancient clusters, we have to formally define the structure of a cluster.

Which cluster is contained in which genome? We have to frame rules to decide which clusters are contained in a given genome, or which genomes contain a given cluster.

These considerations are formalized in the following definition. Note that it does not restrict the genomes or the gene clusters in any way.

Definition 7. (Gene Cluster Model) *A* gene cluster model *is a triple* $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$*, where*

- $\mathcal{U}$ *is a set, called the universal set, i.e. the set of all genomes;*
- $\mathcal{C}$ *is the set of all gene clusters; and*
- $\sqsubseteq \subseteq \mathcal{C} \times \mathcal{U}$ *is a binary relation that determines whether a gene cluster* $c \in \mathcal{C}$ *is contained in a genome* $g \in \mathcal{U}$*.*

We would like to highlight that all following theoretical considerations, the problem definition, and the solution presented in Chapter 3 are based on this abstract definition. That means, the abstract model can be seen as an interface between our framework and a variety of concrete gene cluster definitions: We can incorporate any cluster model by defining $\mathcal{U}$, $\mathcal{C}$ and $\sqsubseteq$ accordingly. This concept is also known as *oracle-based*. In this case, the oracle is the abstract gene cluster model, which is used as a black box in algorithms to answer questions like *"Is this gene cluster contained in this genome?"* The actual answer and therefore the outcome of the algorithm depends on the concrete instantiation of the oracle.

The following definition of a simple gene cluster model illustrates how the oracle can be instantiated. Further models will be discussed in Chapters 4 and 5.

Model. (Common Intervals on Permutations) *Let N be an arbitrary but fixed length of the gene orders and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. Then the gene cluster model of* common intervals on permutations *is defined as* $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$*, where*

- $\mathcal{U}_c$ *is the set of all permutations over* $\mathcal{G}_N$*;*
- $\mathcal{C}_c$ *is the set of all subsets* $c \subsetneq \mathcal{G}_N$ *with* $|c| > 1$*; and*
- $c \sqsubseteq_c g :\Leftrightarrow$ *all genes in* c *occur contiguously in genome* g*.*

Note that trivial clusters like singletons and clusters of size N would not provide any deeper insight and are therefore excluded from the definition.

In the subsequent example, we enumerate the genomes and clusters explicitly and thus have to fix the parameter N. For the sake of presentation, we choose a very small value for the genome length.

Example 2. For $N = 4$, the above model is defined by the following sets:

$$\begin{aligned}\mathcal{U}_c = \big\{&(1,2,3,4),(1,2,4,3),(1,3,2,4),(1,3,4,2),(1,4,2,3),(1,4,3,2),\\&(2,1,3,4),(2,1,4,3),(2,3,1,4),(2,3,4,1),(2,4,1,3),(2,4,3,1),\\&(3,1,2,4),(3,1,4,2),(3,2,1,4),(3,2,4,1),(3,4,1,2),(3,4,2,1),\\&(4,1,2,3),(4,1,3,2),(4,2,1,3),(4,2,3,1),(4,3,1,2),(4,3,2,1)\big\}\end{aligned}$$

$$\begin{aligned}\mathcal{C}_c = \big\{&\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},\\&\{1,2,3\},\{1,2,4\},\{1,3,4\},\{2,3,4\}\big\}\end{aligned}$$

The adjacencies and permutations are related as follows, where a check-mark means that the gene cluster in the corresponding line is contained in the genome in the corresponding column.

$\sqsubseteq_c$	$(1,2,3,4)$	$(1,2,4,3)$	$(1,3,2,4)$	$(1,3,4,2)$	$(1,4,2,3)$	$(1,4,3,2)$	$(2,1,3,4)$	$(2,1,4,3)$	$(2,3,1,4)$	$(2,3,4,1)$	…
$\{1,2\}$	✓	✓					✓	✓			
$\{1,3\}$			✓	✓			✓		✓		
$\{1,4\}$					✓	✓		✓	✓	✓	
$\{2,3\}$	✓		✓		✓	✓			✓	✓	
$\{2,4\}$		✓	✓	✓	✓						
$\{3,4\}$	✓	✓		✓		✓	✓	✓		✓	
$\{1,2,3\}$	✓		✓				✓		✓		
$\{1,2,4\}$		✓			✓			✓			
$\{1,3,4\}$				✓		✓	✓	✓	✓	✓	
$\{2,3,4\}$	✓	✓	✓	✓	✓	✓				✓	

◇

So far, we specified gene clusters to be embodied by single, independent gene orders. Certainly, the existence of such a substructure in a genome is not meaningful before it is viewed in a broader context. Thus, in the following sections, we establish properties of sets of gene clusters, in particular from a phylogenetic perspective.

2.2 Labelings and Their Properties

Given the gene order (and therefore the gene clusters) of contemporary species, we want to infer conserved gene clusters for the ancestral genomes as will be shown in Figure 2.1(a) on page 24. The actual result of the gene cluster reconstruction method is a set of putative clusters for each ancestral node in the phylogenetic tree. Independently of further properties of such an assignment, we formalize this general scheme in the following definition.

Definition 8. (Labeling) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model and $T = (V, E)$ be a tree. A* labeling *of T is a function that assigns to each node $v \in V$ a set of gene clusters $C \subseteq \mathcal{C}$:*

$$\lambda : V \longrightarrow \mathcal{P}(\mathcal{C}) \,,$$

where $\mathcal{P}(\mathcal{C})$ is the power set, i.e. the set of all subsets, of $\mathcal{C}$.

On various occasions, we preferably consider an individual gene cluster separately. For this purpose, we want to know whether a specific cluster is assigned to a node of the tree or not. Formally, we decompose a labeling λ into binary labelings λ_c for all $c \in \mathcal{C}$:

$$\lambda_c(v) := \begin{cases} 1 & \text{if } c \in \lambda(v), \\ 0 & \text{otherwise.} \end{cases}$$

If we assume that a rooted phylogenetic tree is given and all of the descendants of an ancestral node v share a gene cluster, it is obvious that v should be labeled with this cluster as well. Similarly, if none of the descendants have a specific cluster, the node should not be labeled with that cluster. But what if only a subset of the descendants shows a common structure? And what if different descendants suggest contradictory clusters for the node v?

Since we do not know what exactly has happened during evolution, many answers can be motivated without ever being verified. Beside others, methodological approaches have been proposed in previous studies [10, 44]. Even though such reconstruction algorithms sound plausible, a more natural way to ensure explanatory power of the results may be the definition of an objective function that rates a given labeling. This way, we can specify properties of a *good* result and verify whether an algorithm finds one of the *optimal* solutions. Next, we characterize two key properties of labelings, which will finally be combined to the overall objective in Section 2.3.

2.2.1 Consistency

Before we actually state the first requirement for a labeling to be a reasonable reconstruction result, we introduce the following concept as a motivation. Assume a labeling assigns a set of putative gene clusters to an ancestral node in the tree. These ancient clusters can in turn be used to represent a set of putative ancient genomes: all genomes of the universal set that contain all the given clusters.

Definition 9. (Genome Set) *Let* $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ *be a gene cluster model and* $C \subseteq \mathcal{C}$ *a set of gene clusters. The* genome set *of* C*, denoted by* $\mathrm{GS}(C)$*, is the subset of genomes in* $\mathcal{U}$ *that contain all gene clusters in* C*:*

$$\mathrm{GS}(C) := \{g \in \mathcal{U} \mid c \sqsubseteq g \textit{ for all } c \in C\}\,.$$

Example 3. Consider the gene cluster model $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$ of common intervals on unsigned permutations with $N = 4$ as detailed in Example 2.

Let $C = \big\{\{1,4\}, \{1,3,4\}, \{2,3,4\}\big\}$.

Then the genome set of C is: $\mathrm{GS}(C) = \big\{(1,4,3,2), (2,3,4,1)\big\}$. ◇

A naive algorithm to compute $\mathrm{GS}(C)$ follows the definition and proceeds in two steps. It enumerates all possible genomes, and then tests for each of them whether it contains all clusters in C or not. In general, the cardinality of the universal set $\mathcal{U}$ can increase exponentially with respect to the length of the genomes. Hence, processing all genomes is infeasible for practical instances. In Chapters 4 and 5, more sophisticated methods to handle genome sets are developed. For some concrete gene cluster models, we circumvent the explicit, genome-wise checking. However, we will also show that this is not an easy task in general.

Continuing the discussion of the implications of a labeling, a natural question that arises is if the genome set can be empty. Depending on the gene cluster model used, this can happen if some of the clusters derived from different contemporary species are in contradiction with others. The following notion, introduced by Bergeron *et al.* [10], conceptualizes this extreme case.

Definition 10. (Conflicting Set of Gene Clusters) *Let* $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ *be a gene cluster model and* $C \subseteq \mathcal{C}$ *a set of gene clusters.* C *is* conflicting *if* $\mathrm{GS}(C) = \emptyset$*.* C *is* minimal conflicting *if it is conflicting and there is no conflicting subset* $C' \subsetneq C$*.*

Observation 1. *Let* $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ *be a gene cluster model and* $C \subseteq \mathcal{C}$ *a conflicting set of gene clusters.* C *is* minimal conflicting *if and only if for each* $c \in C$*:* $\mathrm{GS}(C \backslash \{c\}) \neq \emptyset$*.*

Note that every conflicting set is either minimal conflicting or it contains minimal conflicting subsets. For a set of gene clusters C, let $\mathrm{Conf}(C)$ be the set of all minimal conflicting subsets of C.

Example 4. Again, consider the gene cluster model $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$ of common intervals on permutations with $N = 4$ as detailed in Example 2.

Let $C = \big\{\{1,4\}, \{1,2,3\}, \{1,3,4\}, \{2,3,4\}\big\}$ be a given set of common intervals. C is conflicting, since $\mathrm{GS}(C) = \emptyset$, and it contains exactly two minimal conflicting subsets:

$$\begin{aligned} C_1 &= \big\{\{1,4\}, \{1,2,3\}, \{2,3,4\}\big\}, \text{and} \\ C_2 &= \big\{\{1,2,3\}, \{1,3,4\}, \{2,3,4\}\big\}. \end{aligned}$$

Following Observation 1, this can be seen, for instance for C_1:

$$\begin{aligned}
\mathrm{GS}(C_1) &= \mathrm{GS}\left(\{\{1,4\},\{1,2,3\},\{2,3,4\}\}\right) = \emptyset \\
\mathrm{GS}\left(C_1 \backslash \{\{1,4\}\}\right) &= \mathrm{GS}\left(\{\{1,2,3\},\{2,3,4\}\}\right) \\
&= \left\{(1,2,3,4),(1,3,2,4),\dots\right\} \neq \emptyset \\
\mathrm{GS}\left(C_1 \backslash \{\{1,2,3\}\}\right) &= \mathrm{GS}\left(\{\{1,4\},\{2,3,4\}\}\right) \\
&= \left\{(1,4,2,3),(1,4,3,2),\dots\right\} \neq \emptyset \\
\mathrm{GS}\left(C_1 \backslash \{\{2,3,4\}\}\right) &= \mathrm{GS}\left(\{\{1,4\},\{1,2,3\}\}\right) \\
&= \left\{(2,3,1,4),(3,2,1,4),\dots\right\} \neq \emptyset
\end{aligned}$$

Since C_1 and C_2 are the only minimal conflicting subsets of C, $\mathrm{Conf}(C)=\{C_1,C_2\}$. ◇

Assigning a conflicting set of gene clusters to a node in the tree would exclude any valid gene order and thus the existence of a potential genome for the corresponding ancestral species. We therefore rule out that a reconstructed labeling contains any conflicts by the following criterion.

Definition 11. (Consistency) *Let* $(\mathcal{U},\mathcal{C},\sqsubseteq)$ *be a gene cluster model and* $T=(V,E)$ *be a tree. A labeling* λ *is* consistent *if for all* $v \in V$ *the labeling contains no conflicts,* $\mathrm{Conf}(\lambda(v))=\emptyset$.

Corresponding methods and complexity issues concerning specific cluster models will be addressed in Chapters 4 and 5.

2.2.2 Parsimony

One way of explaining the emergence of contemporary, observed data without further knowledge about the ancient or intermediate states is to apply the *parsimony principle*, also known as *Occam's razor*. This approach is widely used in phylogeny since it was proposed in the 1960s by Wagner [159], and Edwards and Cavalli-Sforza [60, 61]. But beyond biological systematics, this idea is also used in other scientific fields, like philosophy, mathematics or physics. It even goes back to Aristotle:

> "We may assume the superiority *ceteris paribus* [other things being equal] of the demonstration which derives from fewer postulates or hypotheses." [4]

Generally spoken, this means that from different explanations we choose the simplest. In the context of phylogenetics, it can be formulated as: Among all scenarios which result in the observed data, we want to find those with a minimal amount of evolutionary changes.

Thereby, the data can consist of different types of attributes of the species, called *characters*, for which different qualities or quantities, called *character states*, can be observed. Various kinds of characters can be considered, for example eye color, number of extremities, or the amino acid at a certain position of a specific gene. For phylogenetic inferences, suitable characters must represent inheritable features. The state of a character can either be passed on unchanged from an ancestor to an descendant or it can change due to evolutionary processes.

Each single change of a character state is called an *evolutionary event*, or in short just *event*. Eventually, all events combined cause the varying states observed at today living species.

Our aim is to infer the history of gene clusters. For each cluster, we create a binary character whose state can either be 1 or 0, representing whether the species possesses the corresponding cluster or not. An event can be either a change of a state from 0 to 1, which is the gain of a gene cluster, or from 1 to 0, which is the loss of a cluster.

Instead of simply counting the number of events to determine the amount of evolutionary changes, we can also distinguish between different types of events. To this end, we define a function which assigns a penalty, called *weight*, to each event and minimize the total weight of all events in the scenario. These weights can depend on the edge in the phylogenetic tree the event occurs at, the affected character (resp. cluster), and the actual event that happens (resp. gain, loss or no change). The only restrictions one usually adds are:

1. The weight for a change of a state is higher than for keeping a state.
2. The weight for an event on one edge for one character is independent of an event on any other edge or for any other character.

The first assumption is natural and is more a mathematical rather than a practical restriction. There is no sense in trying to minimize evolutionary events while penalizing stability. On the contrary, the latter specification is debatable. In general, it is not natural that characters evolve independently. For instance, the change of an amino acid at one position of a gene may involve the change of a second amino acid to preserve a certain protein structure. For our purposes, it is infeasible to include such interdependencies of the evolution of different gene clusters for two reasons: On the one hand, our aim is to gain new insights in the evolution of gene clusters. Beside the fact that to our knowledge, there are no general patterns of dependency found so far, assuming such prior knowledge could bias the outcome of the study. On the other hand, structural interconnections, like overlaps or nestedness of gene clusters, depend on the actual definition of the cluster model, and functional dependencies are specific to certain data sets. The scope of our survey, however, is to develop a method which is both model-independent and not adjusted to any certain data set. In conclusion, we follow Aristotle's advice to avoid further hypotheses and weight evolutionary events independently. Nevertheless, we are aware of this inaccuracy and bear it in mind for the evaluation of our results later on.

The following definition formalizes the above characteristics for our purposes. It includes the above mentioned restrictions that allow us to deduce some key properties which build a basis for the reconstruction method presented in Chapter 3.

Definition 12. (Parsimony Weight Function) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, $T = (V, E)$ be a rooted tree and*

$$w : \mathcal{C} \times E \times \{0,1\} \times \{0,1\} \longrightarrow \mathbb{R}$$

be a function that assigns the weight $w(c, e, x, y)$ to the event of cluster c on edge e changing from state x to y. It is called a parsimony weight function *if for all $c \in \mathcal{C}, e \in E$:*

$$w(c, e, x_1, y_1) \leq w(c, e, x_2, y_2) \quad \text{for all } x_1, y_1, x_2, y_2 \in \{0,1\}$$
$$\text{with } x_1 = y_1 \text{ and } x_2 \neq y_2.$$

Based on this, we can now formulate parsimony as a second property of a labeling in the gene cluster reconstruction framework.

Definition 13. (Parsimony) *Let $(\mathcal{U}, \mathcal{C}, \sqsupseteq)$ be a gene cluster model, $T = (V, E)$ be a rooted tree with each leaf $l \in V$ labeled with a set of gene clusters C_l, and w be a parsimony weight function. A labeling $\lambda : V \to \mathcal{P}(\mathcal{C})$ is* most parsimonious *if $\lambda(l) = C_l$ for all leaves and the total weight*

$$W(T, \lambda) \; := \sum_{(u,v) \in E} \sum_{c \in \mathcal{C}} w\big(c, (u, v)\,, \lambda_c(u), \lambda_c(v)\big)$$

is minimal.

Not only the concept of parsimony itself, but a considerable amount of variants of weighting schemes for different applications has been studied and discussed in the past. Traditionally, the following properties are used to characterize a parsimony weight function w. It is called

cluster-independent if $w(c_i, e, x, y) = w(c_j, e, x, y)$ for all $c_i, c_j \in \mathcal{C}$. In some cases, the gene cluster model under consideration may allow us to judge evolutionary events for specific clusters differently. Then, we can use a cluster-dependent weighting scheme to incorporate this into our objective.

edge-independent if $w(c, e_i, x, y) = w(c, e_j, x, y)$ for all $e_i, e_j \in E$. If the evolutionary distances in a tree are known, e.g. in terms of time and/or mutation rate, this knowledge can be used to define weights for events on each edge separately.

symmetric if $w(c, e, x, y) = w(c, e, y, x)$ for all $x, y \in \{0, 1\}$. We might assume that the formation of a gene cluster is less probable that the loss of a cluster. This can easily be modeled by assigning a higher value to $w(c, e, 0, 1)$ than to $w(c, e, 1, 0)$.

In the following, we want to review some classical, concrete weighting schemes shortly. For a broader and more detailed discussion, see for example the review by Felsenstein [74]. Originally, these objectives were not formulated based on weighting functions for binary states on gene clusters. Here, we show how they can be rephrased according to our purposes.

Camin and Sokal [39] introduced a model that assumes an ordering of the states. A character can only change its state stepwise in the given direction: from “primitive to derived”. In the context of binary characters, this scheme only allows a state to change from 0 to 1 and prohibits any reversal, i.e. a change from 1 to 0. This can be formalized by simply assigning a weight of infinity for the latter event.

$$w_{\textit{Camin-Sokal}}(c, e, x, y) := \begin{cases} 1 & \text{if } x = 0 \text{ and } y = 1, \\ \infty & \text{if } x = 1 \text{ and } y = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Another long-standing maximum parsimony model is the so-called *Dollo parsimony*. It assumes that the state 1 corresponds to a complex characteristic. Following Dollo's principle, the formation of such a property is very rare. Le Quesne [112, 113] suggested and Farris [70, 71] formalized this idea, by allowing the state 1 to arise only once and minimizing only the number of changes from 1 to 0. This means that *homoplasy*, the formation of the same trait in different lineages, is precluded. We penalize the gain of a cluster by $\#leaves$, the number of leaves in the tree. This way, we ensure that it is always 'cheaper' to introduce the state 1 only once and to lose it again (at most $\#leaves$ times) than to introduce it a second time.

$$w_{Dollo}(c, e, x, y) := \begin{cases} \#leaves & \text{if } x = 0 \text{ and } y = 1, \\ 1 & \text{if } x = 1 \text{ and } y = 0, \\ 0 & \text{otherwise.} \end{cases}$$

However, generally, there are several counter-examples for homoplasy known by now, for example the parallel development of wings of birds and bats. Since we cannot exclude this phenomenon in the development of gene clusters, we should not rule it out in principle but rather explicitly allow homoplasy to take place. This ensures to be able to detect and examine this interesting feature, which might elucidate the evolution or the function of certain gene clusters.

In spite of further possible good reasons for defining a particularly detailed, specialized weighting scheme, we want to stick to a preferably simple function due to two reasons: Firstly, a weight function that is cluster-dependent cannot be directly transferred from one cluster model to another. A major aim of this work is to use different cluster models for the reconstruction process and to compare the results. Using different, specific weighting schemes would correspond to applying different optimization criteria and therefore producing results which are not comparable. Secondly, instead of adjusting a weight function to a fixed cluster model or a specific data set, we want to evaluate the general objective – independent of distance measures in the tree, probabilities of evolutionary events or any other assumptions.

To this end, the following most basic but also commonly used function is chosen in our context. This model and generalizations have been introduced, used and discussed by various authors [59, 61, 69, 104]. In his review, Felsenstein defines it as *Wagner parsimony*. It is also known under the name of Fitch, who developed a widely known algorithm to find a most parsimonious labeling under Wagner parsimony [77]. This method will be discussed in Section 3.1.

Definition 14. (Wagner Parsimony Weight Function) *Let* $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ *be a gene cluster model,* $T = (V, E)$ *be a rooted tree. Then the* Wagner parsimony weight function w_{Wagner} *is a symmetric, edge- and cluster-independent parsimony weight function, defined for all* $c \in \mathcal{C}$ *and* $e \in E$ *as follows:*

$$w_{Wagner}(c, e, x, y) := \begin{cases} 1 & \textit{if } x \neq y, \\ 0 & \textit{otherwise.} \end{cases}$$

For this weighting scheme, the total parsimony weight of the binary labelings is equivalent to the sum of the *symmetric set differences* of the node labels as originally formalized in Definition 8 on page 16. In general, this set theoretic distance is a commonly used, mathematically simple metric, which is in particular utilized in phylogenetics [81, 83, 115].

$$\begin{aligned} W(T,\lambda) &= \sum_{(u,v)\in E} \sum_{c\in\mathcal{C}} w_{\mathit{Wagner}}\big(c, (u,v)\,, \lambda(u), \lambda(v)\big) \\ &= \sum_{(u,v)\in E} \sum_{c\in\mathcal{C}} \left\{ \begin{array}{ll} 1 & \text{if } \lambda_c(u) \neq \lambda_c(v) \\ 0 & \text{otherwise} \end{array} \right\} \\ &= \sum_{(u,v)\in E} \Big| \big\{ \; c \in \mathcal{C} \mid \lambda_c(u) \neq \lambda_c(v) \; \big\} \Big| \\ &= \sum_{(u,v)\in E} \Big| \big(\; \lambda(u) \cup \lambda(v) \; \big) \backslash \big(\; \lambda(u) \cap \lambda(v) \; \big) \Big| \end{aligned}$$

We would like to emphasize that the general concept studied in this work, including the objective *parsimony*, its relation to the criterion of *consistency* and the overall method, is based on the general definition of parsimony weight functions and not fixed to the Wagner parsimony function, which we chose for this study. Hence, specialization on certain gene cluster models or data sets is possible without difficulty.

Small and large parsimony. The task of finding a most parsimonious evolutionary scenario can be further subdivided into two variants – one being a subproblem of the other. In general, the character states of the contemporary species under consideration are given. If in addition the topology of the corresponding phylogenetic tree is known, we only seek for a most parsimonious assignment of character states to the ancestral species. This is known as the *small parsimony problem.* In contrast, if the phylogeny is not given and we also have to find such a tree that affords a most parsimonious assignment, this is called the *large parsimony problem.* The small parsimony problem can be solved efficiently in polynomial time and space. Adequate methods will be discussed in Section 3.1. By comparison, the large parsimony problem is known to be NP-hard and can in practice only be solved heuristically [82]. Furthermore, some concerns have been raised about the correctness of the inferred tree topologies, as for instance the problem of *long branch attraction* [72, 89]. However, there are other, more reliable methods to determine phylogenies. Throughout this thesis, we assume the phylogenetic tree to be given and our overall optimization criterion will only be based on the small parsimony problem. But, since this can be seen as a subproblem of the large parsimony problem, one could think of integrating it into corresponding heuristics solving the large problem.

In the context of the small parsimony problem, an advantage of symmetric weight functions, like the Wagner parsimony, is that no directed ancestor-descendant relation is assumed. Thus, a given phylogenetic tree does not necessarily have to be rooted. Any node can be chosen as a root without altering the result.

Revisiting our notion of consistency, the parsimony criterion generally still allows conflicts. For instance, Figures 2.1(b) and 2.1(c) on the following page show most parsimonious labelings which are at the same time inconsistent. A corresponding, more extensive example will be discussed later. In the following section, we will combine parsimony with the criterion of consistency and elucidate important relations between them.

2.3 The Labeling Problem

In the last sections, two criteria for labeling a phylogenetic tree with sets of gene clusters have been motivated. We consider the problem of finding a labeling that is both parsimonious and consistent.

Definition 15. (Labeling Problem) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, $T = (V, E)$ a rooted tree with each leaf $l \in V$ labeled with a set of gene clusters C_l, and w be a parsimony weight function. The* labeling problem *is to find, under all consistent labelings of T, a labeling λ of minimal total weight $W(T, \lambda)$. Such a labeling is called* optimal.

The following example illustrates the different criteria of the above definition.

Example 5. Consider the Wagner parsimony weight function and the gene cluster model of common intervals on permutations with genome length $N = 3$:

$$\mathcal{U}_c = \{(1,2,3),(1,3,2),(2,1,3),(2,3,1),(3,1,2),(3,2,1)\}$$
$$\mathcal{C}_c = \{\{1,2\},\{1,3\},\{2,3\}\}$$
$$\{1,2\} \sqsubseteq (1,2,3), \{1,2\} \not\sqsubseteq (1,3,2), \{1,2\} \sqsubseteq (2,1,3), \ldots$$

In this case, $\{\{1,2\},\{1,3\},\{2,3\}\}$ is a (minimal) conflicting set. In Figure 2.1(b) on the next page, a concrete problem instance of the labeling problem is given, and Figures 2.1(b) to 2.1(f) show different labelings with the following properties:

	λ_1	λ_2	λ_3	λ_4	λ_5
$W(\lambda_i)$	3	3	6	8	4
parsimonious	✓	✓			
consistent				✓	✓
optimal					✓

◇

In the following, we show important interconnections between the definition of optimality and its two subcriteria. Later on, these relations are used for the computation of optimal labelings.

If a labeling λ of T has minimal weight but is inconsistent, we suppose that it contains too many clusters, and we therefore reach consistency by deleting some conflicting clusters from the labeling of some nodes. The following Theorem 1 verifies that we find an optimal labeling using this strategy. It excludes the existence of any labeling that contains clusters which the parsimonious labeling λ does not include, but nevertheless has a smaller weight. Moreover, Theorem 2 guarantees that *all* optima can be found this way.

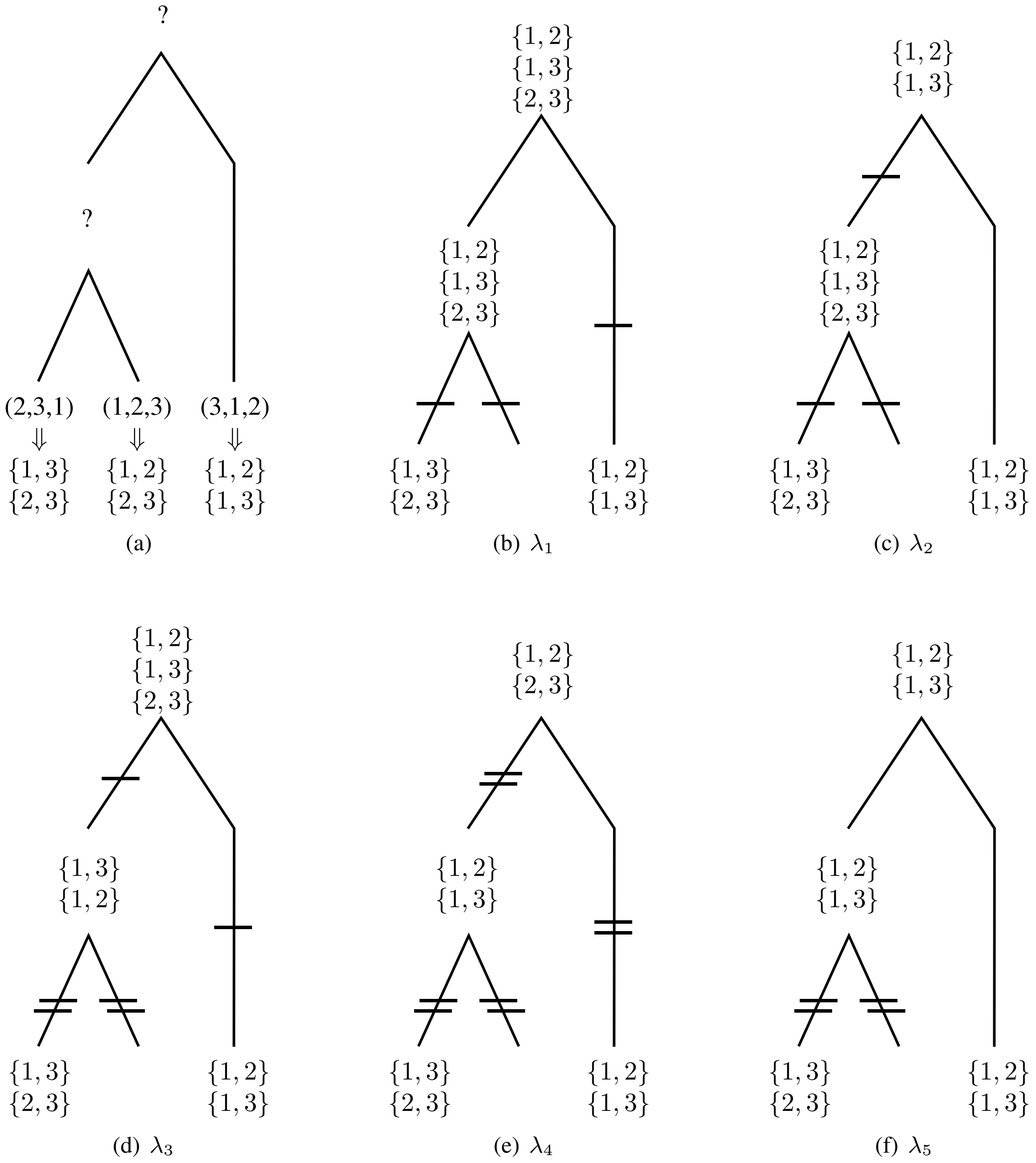


Figure 2.1: Example of different labelings for a given instance of the labeling problem for the gene cluster model of common intervals on permutations with genome length $N = 3$. Evolutionary events and thus the weight of the labelings are indicated by horizontal bars on the edges. See Example 5 on the previous page for further details. (a) The leaves of the given tree are labeled with gene orders comprising the shown clusters. (b–f) Different labelings, as discussed in the example, are illustrated.

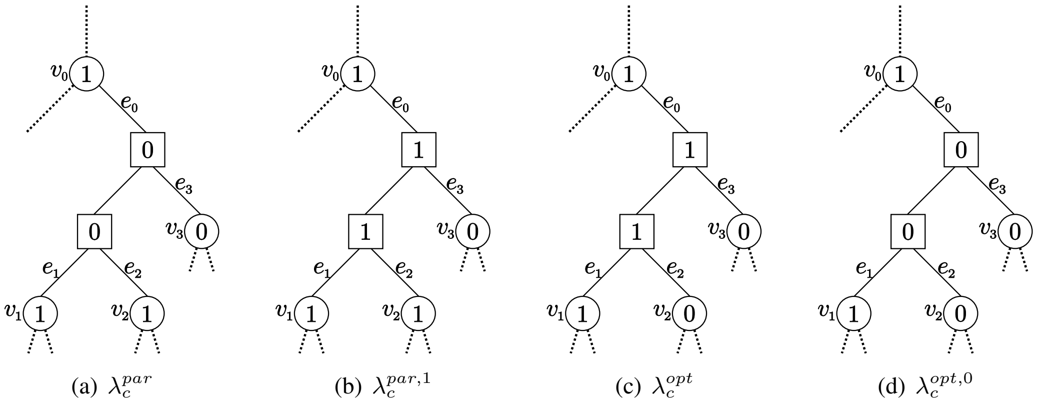
Theorem 1. *Let $\lambda^{par}\colon V \longrightarrow \mathcal{P}(\mathcal{C})$ be a most parsimonious labeling for a rooted tree $T = (V, E)$. Then there exists an optimal labeling λ^{opt} with $\lambda^{opt}(v) \subseteq \lambda^{par}(v)$ for all $v \in V$.*

Theorem 2. *Let $\lambda^{opt}\colon V \longrightarrow \mathcal{P}(\mathcal{C})$ be an optimal labeling for a rooted tree $T = (V, E)$. Then there exists a most parsimonious labeling λ^{par} with $\lambda^{opt}(v) \subseteq \lambda^{par}(v)$ for all $v \in V$.*

Proof of Theorem 1. For contradiction, assume that for each optimal labeling λ^{opt} there is at least one cluster c satisfying the condition

$$\lambda_c^{opt}(v) = 1 \text{ and } \lambda_c^{par}(v) = 0 \tag{2.1}$$

for some node $v \in V$. For such a cluster, consider a maximal connected component of vertices in T which satisfy this condition and let $v_1, \ldots, v_k$ be the neighboring nodes that are direct descendants of the component connected by the edges $e_1, \ldots, e_k$ and let v_0 be the direct ancestor of the component connected by e_0 if there is any such node. Note that $v_0, \ldots, v_k$ do not satisfy condition 2.1 by definition. See Fig. 2.2(a) and 2.2(c) for an example.



(a) λ_c^{par} (b) $\lambda_c^{par,1}$ (c) λ_c^{opt} (d) $\lambda_c^{opt,0}$

Figure 2.2: An example for the connected component used in the proofs of Theorems 1 and 2. The square nodes are the inner nodes, and the round nodes are the surrounding v_i. See Example 6 on the next page for further details.

For each of these nodes v_i, $0 \leq i \leq k$, one of the following cases must hold:

(i) $\lambda_c^{par}(v_i) = 1$ and $\lambda_c^{opt}(v_i) = 1$,

(ii) $\lambda_c^{par}(v_i) = 1$ and $\lambda_c^{opt}(v_i) = 0$, or

(iii) $\lambda_c^{par}(v_i) = 0$ and $\lambda_c^{opt}(v_i) = 0$.

We partition the surrounding edges $e_1, \ldots, e_k$ into non-overlapping sets corresponding to the above three cases. We collect all edges whose corresponding nodes satisfy (i) in E_{11}, and the other edges in E_{10} and E_{00} respectively, and construct the following labelings:

- Let $\lambda^{par,1}$ be the labeling we get by substituting the inner zeros of λ^{par} by ones as shown in Figure 2.2(b).
- Let $\lambda^{opt,0}$ be the labeling we get by substituting the inner ones of λ^{opt} by zeros as shown in Figure 2.2(d).

Now, we compare the weights of the edges for the labelings $\lambda^{par,1}$ and λ^{par}, and $\lambda^{opt,0}$ and λ^{opt}. An edge $e \in E_{00}$ has weight $w(c, e, 1, 0)$ in $\lambda^{par,1}$ and weight $w(c, e, 0, 0)$ in λ^{par}. The same edge has weight $w(c, e, 0, 0)$ in $\lambda^{opt,0}$ and weight $w(c, e, 1, 0)$ in λ^{opt}, which yields the same difference, but with inverse sign. Hence, if all edges in E_{00} induce a total weight difference of W_{00} between $\lambda^{par,1}$ and λ^{par}, we have a difference of $-W_{00}$ between $\lambda^{opt,0}$ and λ^{opt}. Analogously, all edges in E_{11} induce a total weight difference of $-W_{11}$ between $\lambda^{par,1}$ and λ^{par}, and a total difference of W_{11} between $\lambda^{opt,0}$ and λ^{opt}. Furthermore, let W_{10}^{par} and W_{10}^{opt} be the total weight differences of edges in E_{10} between $\lambda^{par,1}$ and λ^{par} and $\lambda^{opt,0}$ and λ^{opt} respectively, and let w_0^{par} and w_0^{opt} be the corresponding weight differences of the edge e_0 if it exists and zero otherwise.

Altogether, this leads to the following equations:

$$W(T, \lambda^{par,1}) = W(T, \lambda^{par}) + W_{00} - W_{11} + W_{10}^{par} + w_0^{par} \tag{2.2}$$

$$W(T, \lambda^{opt,0}) = W(T, \lambda^{opt}) - W_{00} + W_{11} + W_{10}^{opt} + w_0^{opt} \tag{2.3}$$

$$\begin{aligned}
\text{with}\quad W_{00} &= \sum_{e \in E_{00}} w(c, e, 1, 0) - w(c, e, 0, 0) \geq 0 \\
W_{11} &= \sum_{e \in E_{11}} w(c, e, 0, 1) - w(c, e, 1, 1) \geq 0 \\
W_{10}^{par} &= \sum_{e \in E_{10}} w(c, e, 1, 1) - w(c, e, 0, 1) \leq 0 \\
W_{10}^{opt} &= \sum_{e \in E_{10}} w(c, e, 0, 0) - w(c, e, 1, 0) \leq 0 \\
w_0^{par} &= w(c, e_0, \lambda^{par,1}(v_0), 1) - w(c, e_0, \lambda^{par}(v_0), 0) \quad \text{(or 0 if } v_0 \text{ undef.)} \\
w_0^{opt} &= w(c, e_0, \lambda^{opt,0}(v_0), 0) - w(c, e_0, \lambda^{opt}(v_0), 1) \quad \text{(or 0 if } v_0 \text{ undef.)}
\end{aligned}$$

According to Definition 12, the weight for state changes (0 → 1 or 1 → 0) cannot be lower than for keeping a state for any edge. Hence, the above defined weight differences W_{00} and W_{11} cannot be negative, and W_{10}^{par} and W_{10}^{opt} cannot be positive.

Example 6. Figure 2.2 on page 25 shows a connected component as used in this proof. The same subtree is depicted with the different labelings. For this instance, the weighting scheme $w := w_{\mathit{Wagner}}$ results in:

$$\begin{aligned} W_{00} &= w(c, e_3, 1, 0) - w(c, e_3, 0, 0) = 1 \\ W_{11} &= w(c, e_1, 0, 1) - w(c, e_1, 1, 1) = 1 \\ W_{10}^{par} &= w(c, e_2, 1, 1) - w(c, e_2, 0, 1) = -1 \\ W_{10}^{opt} &= w(c, e_2, 0, 0) - w(c, e_2, 1, 0) = -1 \\ w_0^{par} &= w(c, e_0, 1, 1) - w(c, e_0, 1, 0) = -1 \\ w_0^{opt} &= w(c, e_0, 1, 0) - w(c, e_0, 1, 1) = 1 \end{aligned}$$

$$\text{such that} \quad \begin{aligned} W(T, \lambda^{par,1}) &= W(T, \lambda^{par}) - 2 \\ W(T, \lambda^{opt,0}) &= W(T, \lambda^{opt}) \end{aligned}$$

◇

The following case analysis will reveal that, in any case, we obtain one of two relations. In the end, we will show that these relations contradict the assumptions, which finally proves the theorem.

Case (1): v_0 undefined or $\lambda^{par}(v_0) \neq \lambda^{opt}(v_0)$

If v_0 is not defined, we have $w_0^{par} = w_0^{opt} = 0$. Otherwise, due to the definition of the connected component, we have to have $\lambda^{par}(v_0) = 1$ and $\lambda^{opt}(v_0) = 0$. This results in the following weight differences for e_0:

$$\begin{aligned} w_0^{par} &= w(c, e_0, \lambda^{par,1}(v_0), 1) - w(c, e_0, \lambda^{par}(v_0), 0) \\ &= w(c, e_0, 1, 1) - w(c, e_0, 1, 0) \\ &\leq 0, \text{ and} \\ w_0^{opt} &= w(c, e_0, \lambda^{opt,0}(v_0), 0) - w(c, e_0, \lambda^{opt}(v_0), 1) \\ &= w(c, e_0, 0, 0) - w(c, e_0, 0, 1) \\ &\leq 0. \end{aligned}$$

We distinguish between the following subcases:

Case (1a): $W_{00} < W_{11}$

Since neither W_{10}^{par} nor w_0^{par} can be positive, Equation 2.2 yields $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$.

Case (1b): $W_{00} \geq W_{11}$

Since neither W_{10}^{opt} nor w_0^{opt} can be positive, Equation 2.3 yields $W(T, \lambda^{opt,0}) \leq W(T, \lambda^{opt})$.

Case (2): $\lambda^{par}(v_0) = \lambda^{opt}(v_0)$

In this case, we get the following weight differences for e_0:

$$\begin{aligned} w_0^{par} &= w(c, e_0, \lambda^{par,1}(v_0), 1) - w(c, e_0, \lambda^{par}(v_0), 0) \\ &= w(c, e_0, \lambda^{opt}(v_0), 1) - w(c, e_0, \lambda^{opt,0}(v_0), 0) \\ &= -\left(w(c, e_0, \lambda^{opt,0}(v_0), 0) - w(c, e_0, \lambda^{opt}(v_0), 1)\right) \\ &= -w_0^{opt}. \end{aligned} \tag{2.4}$$

Again, we look at two sub cases:

Case (2a): $W_{00} - W_{11} + w_0^{par} < 0$

Since W_{10}^{par} can not be positive, Equation 2.2 yields $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$.

Case (2b): $W_{00} - W_{11} + w_0^{par} \geq 0$

Since due to Equation 2.4 $w_0^{par} = -w_0^{opt}$, we have:

$$\begin{aligned} & W_{00} - W_{11} + w_0^{par} \geq 0 \\ \Leftrightarrow\quad & W_{00} - W_{11} - w_0^{opt} \geq 0 \\ \Leftrightarrow\quad & -W_{00} + W_{11} + w_0^{opt} \leq 0 \end{aligned}$$

Since W_{10}^{opt} can not be positive, Equation 2.3 yields $W(T, \lambda^{opt,0}) \leq W(T, \lambda^{opt})$.

In any case, we get either $W(T, \lambda^{par,1}) < W(T, \lambda^{par})$, which contradicts the parsimony of λ^{par}, or we get $W(T, \lambda^{opt,0}) \leq W(T, \lambda^{opt})$, which in case of inequality simply contradicts the optimality of λ^{opt} since $\lambda^{opt,0}$ is consistent as well but has a lower weight. If we have equality, $\lambda^{opt,0}$ is optimal. If there is no other node v with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par,1}(v) = 0$, this contradicts the assumption that there is no such optimal labeling. If there is another such node, we continue the whole argumentation recursively with $\lambda^{opt,0}$ until we eventually end in one of the above contradictions. □

Proof of Theorem 2. The proof is analog to the proof of Theorem 1 with switched roles of λ^{par} and λ^{opt}.

Let λ^{opt} be an optimal labeling and assume that for each parsimonious labeling λ^{par} there is at least one cluster c with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par}(v) = 0$ for some node $v \in V$. Given the same definitions of $\lambda^{opt,0}$ and $\lambda^{par,1}$ as in the proof of Theorem 1, (2.2) and (2.3) hold and we follow the same case analysis except for small changes in the sub cases, where we now differentiate by $\leq$ (and $>$) instead of $<$ (and $\geq$).

Case (1a): $W_{00} \leq W_{11}$

Equation 2.2 yields $W(T, \lambda^{par,1}) \leq W(T, \lambda^{par})$.

Case (1b): $W_{00} > W_{11}$

Equation 2.3 yields $W(T, \lambda^{opt,0}) < W(T, \lambda^{opt})$.

Case (2a): $W_{00} - W_{11} + w_0^{par} \leq 0$
Equation 2.2 yields $W(T, \lambda^{par,1}) \leq W(T, \lambda^{par})$.

Case (2b): $W_{00} - W_{11} + w_0^{par} > 0$
Equation 2.3 yields $W(T, \lambda^{opt,0}) < W(T, \lambda^{opt})$.

In any case, we analogously reach one of the following contradictions.

We either get $W(T, \lambda^{opt,0}) < W(T, \lambda^{opt})$, which contradicts the optimality of λ^{opt}, or we get $W(T, \lambda^{par,1}) \leq W(T, \lambda^{par})$, which in case of inequality simply contradicts the parsimony of λ^{par}. If we have equality, $\lambda^{par,1}$ is parsimonious. If there is no other node v with $\lambda_c^{opt}(v) = 1$ and $\lambda_c^{par,1}(v) = 0$, this contradicts the assumption that there is no such parsimonious labeling. If there is another such node, we continue the whole argumentation recursively with $\lambda^{par,1}$ until we eventually end in one of the above contradictions. □

Note that, if a symmetric parsimony weight function is assumed, both theorems hold for unrooted trees as well. In this case, in the beginning of the above proofs, we do not distinguish between an ancestral node v_0 and descendants $v_1, \ldots, v_k$. Then, the remainder of the argumentation applies analogously.

Figure 2.3 illustrates the relationships between the different classes of labelings. It also depicts and characterizes the search space for the labeling problem and already gives hints on how an optimum can be found. First, we choose any most parsimonious labeling as a starting point. Then, we have to refine the labeling to reach consistency by removing conflicting gene clusters from some assignments. How these respective steps are realized and further subproblems can be solved, will be explained in the next chapter.

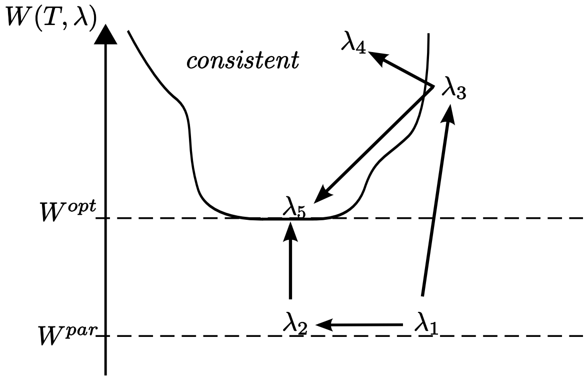


Figure 2.3: Example for the different classes of labelings and their relations. The space of all possible labelings for the problem instance discussed in Example 5 on page 23 is shown, including the labelings λ_1 to λ_5 explicitly. The labelings are arranged according to their weight $W(T, \lambda)$ and whether they are consistent (λ_4 and λ_5) or not (λ_1, λ_2, and λ_3). Within the set of consistent labelings, the ones with minimal weight W^{opt} are optimal (λ_5). It is always possible to get an optimal labeling by deleting some gene clusters from a most parsimonious labeling of weight W^{par} (λ_1 or λ_2). A deletion of a cluster is depicted by an arrow.

Chapter 3

An Algorithmical Framework to Find an Optimal Labeling

In the previous chapter, we introduced the problem of labeling a given phylogenetic tree with sets of gene clusters. Theorem 1 on page 25 displays an important dependency between the underlying criteria and motivates a strategy to find an optimal labeling. Here, we enlarge upon the individual steps of this approach, where we conceptionally proceed similar to the structure of the last chapter: We first discuss algorithms for parsimony and consistency independently, which will then be combined to solve the overall labeling problem. To this end, we compute a most parsimonious labeling. If this is also consistent, it is optimal by definition. Otherwise, following the theorem, there is an optimum which can be reached by excluding gene clusters from some node labelings. All methods we introduce are based on the general definition of a gene cluster model.

The chapter is organized in the following way. A most parsimonious labeling minimizing the Wagner parsimony score function can be computed efficiently by the linear time algorithm of Fitch and Hartigan [77, 85]. We present a variant of this method that is tailored to binary state characters in Section 3.1. In Section 3.2, we introduce our algorithm to identify all minimal conflicting subsets of a given set of gene clusters. Then, we combine both methods in a sophisticated branch-and-bound approach for finding an optimal labeling in Section 3.3. In Section 3.4, we describe how to reconstruct an optimal labeling that preferably contains many clusters. Finally, we comment on the overall complexity of the presented methods in Section 3.5.

3.1 Fitch and Hartigan

As mentioned in Section 2.2.2 on page 18, for the reconstruction of a phylogenetic scenario, evolutionary events can be penalized in different ways. Depending on the considered weighting scheme, specific algorithms ensure an efficient computation of most parsimonious labelings. Both the assumption proposed by Camin and Sokal, and the weight function based on Dollo's parsimony are very restrictive. Thus, a solution for the small parsimony problem can be obtained easily. But also for more general models, efficient algorithms were developed.

A common method for finding a most parsimonious labeling under the Wagner parsimony weighting scheme, which is symmetric, and edge- and character-independent, was first published by Fitch [77] and is thus well-known as the *Fitch algorithm*. Concurrently, Hartigan [85] developed a similar, but more powerful framework, published two years later. In contrast to the Fitch algorithm, his method is not restricted to binary trees and can be used to find all co-optimal solutions. Since he proved its correctness, he was also the first to show the correctness of Fitch's algorithm, depicting it as a special case of his general method.

Shortly after, Sankoff and Rousseau [147, 145] described a generalization for metric, edge-independent weights which may vary for certain events. Erdős and Szély [66], in turn, proposed a similar approach in a further generalized framework for edge-dependent weights. Even for asymmetric models, a linear-time algorithm was recently introduced by Csűrös [49].

As discussed in Section 2.2.2 on page 18, we choose the Wagner parsimony weight function for the purpose of this basic study. In the following, we present the algorithm as proposed by Hartigan, since it is more powerful than Fitch's formulation and not unnecessarily complex. More precisely, we introduce a variant for binary state characters.

The assumption that characters, in our case gene clusters, evolve independently, allows us to perform the construction of a parsimonious labeling character-wise. That means, we compute binary labelings for all gene clusters separately, each minimizing the parsimony weight. Finally, the combination yields an overall labeling that satisfies the parsimony criterion:

$$\begin{aligned}\min_{\lambda}(W(T,\lambda)) &= \min_{\lambda}\Big(\sum_{(u,v)\in E}\sum_{c\in\mathcal{C}} w\big(c,(u,v),\lambda(u),\lambda(v)\big)\Big)\\ &= \sum_{c\in\mathcal{C}}\min_{\lambda}\Big(\sum_{(u,v)\in E} w\big(c,(u,v),\lambda(u),\lambda(v)\big)\Big)\,.\end{aligned}$$

Here, we only give a motivation for how and why the algorithm works. For a more detailed discussion and the proof of correctness, we refer to the original publication of Hartigan [85]. Algorithm 1 works in a dynamic programming fashion. For each gene cluster, it performs two main steps: A bottom-up phase, followed by a top-down refinement.

Bottom-up phase. During a traversal beginning at the leaves and going up to the root, we collect candidate states for the labeling of each node, where putative states are stored in two sets: A primary set P is used for states which definitely yield a most parsimonious labeling, and in a secondary set S, we deposit states which alternatively could also result in a correct solution.

```
Algorithm 1: $\mathrm{FitchBin}(T) \longrightarrow \lambda$        Fitch-Hartigan for Zero-One Instances
Input: A rooted tree $T = (V, E)$ with each leaf $l \in V$ labeled with $\lambda(l) \subseteq \mathcal{C}$.
Output: A labeling $\lambda$ with minimal Wagner parsimony weight $W(T, \lambda)$.
1  foreach $c \in \mathcal{C}$ do
       /* Bottom-up phase */
2      foreach leaf $l$ do
3          $\mathrm{P}(l) \longleftarrow \{\lambda_c(l)\}$
4          $\mathrm{S}(l) \longleftarrow \emptyset$
5      foreach unlabeled node $u$ whose children $v_1, \ldots, v_{\ell(u)}$ are labeled do
6          for $b \in \{0, 1\}$ do $k(b) \longleftarrow |\{v_i \mid b \in \mathrm{P}(v_i)\}|$
7          $K \longleftarrow \max\{k(0), k(1)\}$
8          $\mathrm{P}(u) \longleftarrow \{b \mid k(b) = K\}$
9          $\mathrm{S}(u) \longleftarrow \{b \mid k(b) = K - 1\}$
       /* Top-down refinement */
10     assign any $b \in \mathrm{P}(r)$ to the root node $r$: $\lambda_c(r) := b$
11     foreach unrefined node $v$ whose parent node $u$ is already refined to $\lambda_c(u) = a$ do
12         refine the labeling of $v$ to $\lambda_c(v) \longleftarrow b$, with any $b \in B(v, a)$:

               $$B(v,a) := \begin{cases} \{a\} & \text{if } a \in \mathrm{P}(v), \\ \{a\} \cup \mathrm{P}(v) & \text{if } a \in \mathrm{S}(v), \\ \mathrm{P}(v) & \text{otherwise.} \end{cases}$$

13 return $\lambda$
```

For a leaf l, the state is given and therefore fixed. Hence, the primary candidate set is $\mathrm{P}(l) := \{\lambda_c(l)\}$, and there is no alternative labeling, $\mathrm{S}(l) := \emptyset$ (lines 2 to 4).

To determine the candidate sets for an inner node u, we first count the occurrences of each state in the descendant nodes. On the one hand, the selection of the dominant state implies a minimal number of events and thus gives a valid solution – even if finally the label of its parent node differs from this state, which adds an unavoidable extra weight of one. But on the other hand, in the latter case, choosing the other state would not increase the weight at the parent edge since then the labels are equal. If, additionally, this other state occurs exactly once less in the child nodes than the maximum, this would also yield a total increase by one. To provide the opportunity to find *all* solutions, we store the dominant state (or both states in the case of a tie) in $\mathrm{P}(u)$, and the putative alternative in $\mathrm{S}(u)$ (lines 5 to 9).

Top-down refinement. In this second phase, we have to decide for each node, which of its candidate states we select to obtain a most parsimonious labeling.

For the root node r, selecting any of the states from the primary set $\mathrm{P}(r)$ will finally yield a proper solution if the descendants are labeled correspondingly (line 10).

Then, in a top-down traversal, we assign a labeling to each node v depending on the labeling of its parent node u (lines 11 and 12). If the ancestral state a is also a candidate in the primary set $\mathrm{P}(v)$, only this state can minimize the number of events. But if, instead, a is contained in the secondary set $\mathrm{S}(v)$, both a and the state in $\mathrm{P}(v)$ imply exactly one unavoidable state change: Either one extra change on one of the descending edges, or a change on the edge (u, v). Otherwise, if the ancestral state a is not contained in any of the two candidate sets, selecting a for v would imply at least two changes among v and its descendants more than the state in $\mathrm{P}(v)$. Hence, in this case, the most parsimonious choice is the state in $\mathrm{P}(v)$, even though this causes a change from $\lambda_c(u)$ to $\lambda_c(v)$.

Finally, after all clusters have been processed this way, a most parsimonious overall labeling λ is returned. Since a traversal takes $\mathcal{O}(|V|)$ steps and the computation for each node can be done in constant time, the two phases can be performed in $\mathcal{O}(|V|)$ time for each of the $|\mathcal{C}|$ gene clusters. This gives a linear total time complexity of $\mathcal{O}(|V||\mathcal{C}|)$.

In lines 10 and 12 of Algorithm 1, there may be an arbitrary choice of labeling a node v either with $\lambda_c(v) = 0$ or $\lambda_c(v) = 1$. Preferring $\lambda_c(v) = 1$ yields a labeling containing more clusters. However, the more clusters are reconstructed, the more conflicts can occur. We suggest two strategies with different aims. Later, in Section 3.4, we will describe how both can be combined in a two-phase approach. The two strategies are:

Sparse variant. To avoid conflicts in the first place, we choose $\lambda_c(v) = 0$ whenever possible.

Dense variant. To reconstruct as many clusters as possible, we choose $\lambda_c(v) = 1$ whenever possible.

As already discussed in the previous chapter, parsimony is not the only criterion for a reasonable labeling since a most parsimonious labeling may be inconsistent. Even the sparse approach cannot preclude conflicts completely. In the following section, we introduce an algorithm to identify conflicts.

3.2 Finding Conflicts

Let us assume we have given a set of gene clusters, for instance a set of putative ancient clusters that has been reconstructed for an ancestral species. Here, we consider the problem of identifying all minimal conflicting subsets. For this purpose, we first generalize the term *genome set* specified in Definition 9 on page 17 by introducing the concept of *filtering*.

Definition 16. (Filtered Set of Genomes) *Let $(\mathcal{U}, \mathcal{C}, \sqin)$ be a gene cluster model, $c \in \mathcal{C}$ be a gene cluster and $G \subseteq \mathcal{U}$ be a set of genomes. Then G* filtered by *c is the set of all genomes in G which contain gene cluster c:*

$$G \veedot c := \{g \in G \mid c \sqin g\} .$$

It can be seen easily that successive filtering operations are order-independent. That means, for any gene clusters c_1 and c_2, and any set of genomes G, the following equation holds:

$$(G \mathbin{\dot\vee} c_1) \mathbin{\dot\vee} c_2 = (G \mathbin{\dot\vee} c_2) \mathbin{\dot\vee} c_1 \,.$$

This property allows us to merge a chain of single filtering steps to one new operation.

Definition 17. (Multiple-Filtered Set of Genomes) *Let* $(\mathcal{U}, \mathcal{C}, \mathrel{\exists})$ *be a gene cluster model,* $C = \{c_1, c_2, \ldots, c_k\} \subseteq \mathcal{C}$ *be a set of gene clusters and* $G \subseteq \mathcal{U}$ *be a set of genomes. Then,* G filtered by C *is the set of all genomes in G which contain all gene clusters in* C*:*

$$\begin{aligned} G \mathbin{\mathring{\vee}} C &:= (\cdots((G \mathbin{\dot\vee} c_1) \mathbin{\dot\vee} c_2) \mathbin{\dot\vee} \cdots) \mathbin{\dot\vee} c_k \\ &= \{g \in G \,|\, c \mathrel{\exists} g \;\; \textit{for all } c \in C\} \,. \end{aligned}$$

Finally, we can reformulate the original definition of a genome set as a filtration of the universal set:

$$\mathrm{GS}(C) = \mathcal{U} \mathbin{\mathring{\vee}} C \,.$$

In this framework, the computation of a genome set can methodically be treated as a stepwise filtering process, which means that the result is constructed in a so-called *on-line* fashion. This provides the opportunity to access and reuse intermediate results, or to decide on an early abort of the computation. These features will be utilized in the algorithm described later on.

Given a set of gene clusters C, a naive way to compute $\mathrm{Conf}(C)$, the set of all minimal conflicting subsets of C, would be exhaustive search: Enumerating all subsets of C and testing for each of them whether it is minimal conflicting. The notion of minimality motivates a more sophisticated approach, though. Algorithm 2 on the next page finds all minimal conflicting subsets of a given set of clusters C in a recursive manner. The main method creates an initially empty result list $\mathrm{ConfList}$ and invokes the initial call of the recursive subprocedure. When all recursions are finished, $\mathrm{ConfList}$ contains all minimal conflicting subsets and is returned (lines 1 to 3).

In each pass of the subprocedure $\mathrm{SearchConfRec}$, one minimal conflicting subset is determined and added to $\mathrm{ConfList}$, before further recursions are called. This is done in three steps:

1. First, we identify a conflicting subset $M \subseteq C$. To this end, we initialize a set of genomes G with the universal set and iteratively filter G with more and more clusters from C until we have filtered with all given clusters or the filtered set is empty. After that, if G is not empty, we have $\mathcal{U} \mathbin{\mathring{\vee}} C = \mathrm{GS}(C) \neq \emptyset$ and thus C does not contain any conflicting subset. Otherwise, we define a set M containing all involved clusters we filtered with. Then, M is a conflicting subset of C but not necessarily minimal conflicting (lines 4 to 10).
2. Now, following Observation 1 on page 17, we successively discard clusters c from the set M for which $M \backslash \{c\}$ still contains a conflicting subset. After this step, we have refined M such that $\mathrm{GS}(M) = \emptyset$ and for all $c \in M : \mathrm{GS}(M \backslash \{c\}) \neq \emptyset$. Hence, M is a minimal conflicting set and is inserted into the result list (lines 11 to 13).

```
Algorithm 2: SearchConf($C$) $\longrightarrow$ Conf($C$)        Search for Conflicting Subsets
   Input: A set of gene clusters $C = \{c_1, c_2, \ldots, c_{|C|}\} \subseteq \mathcal{C}$.
   Output: Conf($C$), the set of minimal conflicting subsets of $C$.
 1 ConfList $\longleftarrow$ empty list
 2 SearchConfRec($C$, ConfList)
 3 return ConfList

   Procedure: SearchConfRec($C$, ConfList)
   Input: A set of gene clusters $C = \{c_1, c_2, \ldots, c_{|C|}\} \subseteq \mathcal{C}$, and
          ConfList, a list of solutions found so far.
 4 $G \longleftarrow \mathcal{U}$
 5 $i \longleftarrow 1$
 6 while $G \neq \emptyset$ and $i \leq |C|$ do
 7     $G \longleftarrow G \mathbin{\dot\vee} c_i$
 8     $i \longleftarrow i + 1$
 9 if $G = \emptyset$ then
10     $M \longleftarrow \{c_1, c_2, \ldots, c_i\}$
11     foreach $c \in M$ do
12         if GS($M \backslash \{c\}$) $= \emptyset$ then $M \longleftarrow M \backslash \{c\}$
13     if $M \notin$ ConfList then append $M$ to ConfList
       /* recursion */
14     foreach $c \in M$ do SearchConfRec($C \backslash \{c\}$, ConfList)
```

3. Finally, we have to find all remaining minimal conflicting subsets of C. We start a recursive search for each $c \in M$, searching for minimal conflicting subsets of $C \backslash \{c\}$. This way, we exclude the possibility that the same conflicting set might be found again in the recursive calls (line 14). Furthermore, since minimal conflicting sets cannot be subsets of each other by definition, we do not miss any minimal conflicting set.

Note that the order in which the sets are processed does not alter the overall result since, in any case, the above argumentation holds and thus the algorithm is correct. Nevertheless, in Section 3.2.1, we will discuss how this order can influence the running time in practice.

The efficiency of Algorithm 2 depends on the actual gene cluster model. Hence, in this general context, we analyze the running time in terms of the number of filtrations, which we denote by $\mathrm{F}(|C|, m, \bar{c})$, where C is a set of gene clusters of size $|C|$ and m is the number of minimal conflicting subsets of C with a maximal cardinality of $\bar{c}$. If C does not contain any conflict, which is especially the case if only one cluster is left, we only perform $|C|$ filter operations in lines 6 to 8. Otherwise, additionally, in the second loop, GS is computed

$\mathcal{O}(|C|)$ times by $\mathcal{O}(|C|)$ filter operations each. In the last loop there are $\mathcal{O}(\bar{c})$ recursive calls with both m and $|C|$ decreased by one:

$$\begin{aligned}
\mathrm{F}(|C|, 0, \bar{c}) &= \mathcal{O}(|C|) \\
\mathrm{F}(1, m, \bar{c}) &= \mathcal{O}(1) \\
\mathrm{F}(|C|, m, \bar{c}) &= \mathcal{O}\left(|C|^2 + \bar{c}\,\mathrm{F}(|C|-1, m-1)\right) \\
&= \mathcal{O}\left(|C|^2 \bar{c}^{\min(|C|-1, m)}\right) .
\end{aligned}$$

The space requirement comprises the recursion stack of size $\mathcal{O}(m|C|)$ times some space to store a set of clusters (for example a list of indices), and for the representation of a genome set which depends on the actual model and its implementation.

Further research might explore the existence of a more efficient method to compute all minimal conflicting subsets. Even so, the following example concerning common intervals shows that the number of minimal conflicting subsets can be exponential with respect to the genome length and to the size of C. Similar examples can be constructed for all gene cluster models discussed in this thesis and for all other models including clusters which represent the adjacency of two genes. Hence, in general, the time and space requirements for the computation of the minimal conflicting sets are exponential.

Example 7. Consider the gene cluster model of common intervals on permutations with genome length $N = 2n + 1$ for any $n > 0$ and the following set of gene clusters:

$$\begin{aligned}
C = &\left\{\{g, g+1\} \mid g = 1, \ldots, N-1\right\} \\
&\cup \left\{\{g, g+2\} \mid g = 1, 3, \ldots, N-2\right\} \cup \{1, N\} .
\end{aligned}$$

We can force each pair of genes g and $g + 2$ (for all $g = 1, 3, 5, \ldots, N - 2$) to be connected in the genome either directly by using the cluster $\{g, g + 2\}$, or via $g + 1$ using the clusters $\{g, g + 1\}$ and $\{g + 1, g + 2\}$. This way, we have 2^n possibilities to build a chain from 1 to N. If we add the common interval $\{1, N\}$, this would yield a cycle in the genome. Because cycles are not allowed in the model, this is a conflict. Furthermore, if we take out any of the clusters, we break the cycle as well as the conflict. Hence, all considered subsets of C are minimal conflicting.

As a result, C contains at least $2^{\lfloor N/2 \rfloor}$ minimal conflicting subsets. This number is also exponential in the size of C since $|C|$ is linear in N. ◇

The above example also shows that, sometimes, one gene cluster can cause multiple conflicts. As we will see later, for the search of an optimal labeling, those clusters which are involved in most of the conflicts are of special interest. For this reason, we define the *conflict index* to specify their number.

Definition 18. (Conflict Index) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, $C \subseteq \mathcal{C}$ a set of gene clusters and $c \in C$. The* conflict index *of c with respect to C is defined as:*

$$\mathrm{CI}(C, c) := |\{C' \in \mathrm{Conf}(C) \mid c \in C'\}| .$$

Example 8. (Example 4 continued) As detailed in Example 4 on page 17, for the gene cluster model of common intervals on permutations with $N = 4$ and the set of clusters $C = \big\{\{1,4\}, \{1,2,3\}, \{1,3,4\}, \{2,3,4\}\big\}$, we have the following conflicting subsets:

$$C_1 = \big\{\{1,4\},\{1,2,3\},\{2,3,4\}\big\}, \text{and}$$
$$C_2 = \big\{\{1,2,3\},\{1,3,4\},\{2,3,4\}\big\}.$$

The clusters $\{1,2,3\}$ and $\{2,3,4\}$ are contained in both minimal conflicting subsets C_1 and C_2. Therefore, their conflict index is $\mathrm{CI}(C,\{1,2,3\}) = \mathrm{CI}(C,\{2,3,4\}) = 2$. The other two clusters appear just once; hence, $\mathrm{CI}(C,\{1,4\}) = \mathrm{CI}(C,\{1,3,4\}) = 1$. ◇

The concept of counting conflicts in a set of gene clusters is not only useful for our particular reconstruction method. Recently, Chauve *et al.* [43] studied the combinatorics of minimal conflicting sets of common intervals on permutations. They give an algorithm that determines if a cluster is conflicting in polynomial time when the size of the common intervals is restricted, and they show that computing the conflict index of common intervals is #P-hard. Furthermore, they propose that, for a reconstructed set of common intervals, instead of identifying all minimal conflicting subsets, maximal consistent subsets should be considered as well. A preliminary evaluation on simulated data indicated that this information is more suitable to distinguish true from false positives. But this finding turned out to be the wrong way around due to an experimental error [42]. In fact, the conflict index can be used to discriminate true from false positives effectively – at least for common intervals on permutations in the considered evaluation setup.

However, our definition of an optimal labeling is independent of this notion. Although we will utilize it to speed up our reconstruction method, the obtained true and false positives are not based on the conflict index explicitely.

3.2.1 Practical Enhancements

Here, we introduce and evaluate different approaches to decrease the running time of the search algorithm. Although these techniques do not improve the worst-case runtime complexity, they yield a noteworthy speed-up in practice. Before we discuss the specific approaches and their performance, we first describe the general configuration for the evaluation.

Evaluation Setup

We compared the different approaches described in this section based on simulated data. Its generation approximately corresponds to the situation in which we later search for conflicts. The procedure consists of three steps.

1. First, we generate a random dataset as it would be appropriate as input for the gene cluster reconstruction method. To this end, we assign a gene order of length 25 to the root node of a balanced, rooted, binary tree with eight leaves. Then, in a top-down manner, we perform five *reversals* along each edge, where each time a segment of the gene order is inverted. To this end, the end points of the reversals are chosen randomly. Eventually, we have a tree with shuffled but still related gene orders at its leaves.

2. Then, a most parsimonious labeling for the inner nodes of this tree is computed, by applying the dense variant of the Fitch-Hartigan algorithm (Algorithm 1 on page 33). We used the gene cluster model of common intervals on permutations as defined on page 15, where we only considered clusters with a maximal size of 15.

3. Finally, the labeling for the root node $C := \lambda(r)$ is used to analyze the performance of the variants of the search algorithm. For each variant, we record the running time to compute $\mathrm{Conf}(C)$.

These steps were repeated 1000 times, where runs were aborted if any of the variants did not finish within 100 seconds. From the remaining 945 instances, 209 labelings were consistent whereas 736 contained conflicts. Figure 3.1 shows the distribution of the number of minimal conflicting subsets in C. The frequency of instances C is rapidly decreasing with the *problem size*, measured by $|\,\mathrm{Conf}(C)|$.

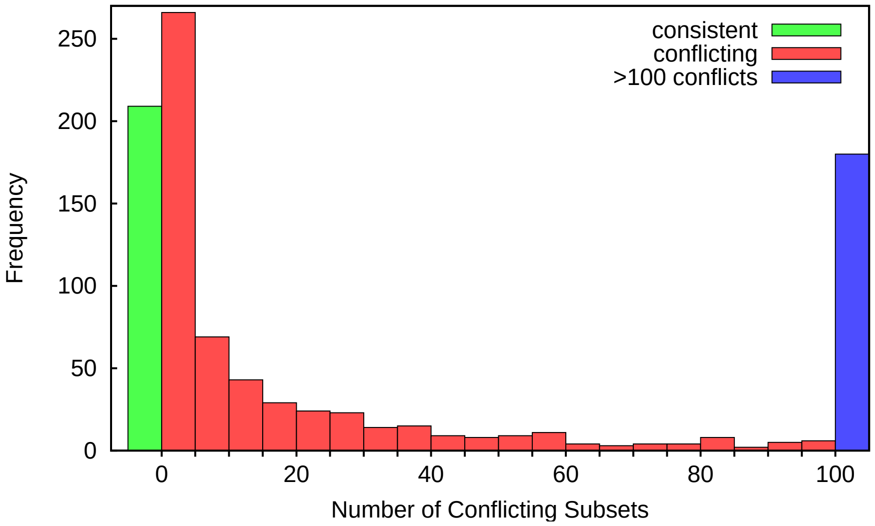


Figure 3.1: The distribution of $|\,\mathrm{Conf}(C)|$, the number of minimal conflicting subsets contained in the simulated set of clusters C. 209 of 945 instances contained no conflict, i.e. they were consistent (leftmost bar), and 180 sets contained more than 100 conflicts (rightmost bar). The remainder is partitioned into buckets of size five: For instance, there were 266 instances with $0 < |\,\mathrm{Conf}(C)| \leq 5$, and 69 instances with $5 < |\,\mathrm{Conf}(C)| \leq 10$.

Randomization

As already mentioned, in Algorithm 2 in lines 6 to 8, the order in which the gene clusters are taken from the set C does not affect the overall result since, in any case, the algorithm finds all conflicting subsets. It neither has an impact on the worst-case runtime complexity. However, here, we discuss different approaches to enumerate the elements from C and show how these can reduce the running time in practice.

In general, the set of all gene clusters $\mathcal{C}$ is represented as an ordered list and a specific set of gene clusters C is in turn represented as a list of elements (or indices) of $\mathcal{C}$. In the following, we consider three techniques to iterate over C.

Simple fixed order. In this approach, we arrange the elements in a set of clusters by their order in the list representing $\mathcal{C}$. We retain this sorting for all sets and subsets we use for the search for conflicts.

Shuffled fixed order. Here, we randomly shuffle the aforementioned order of the elements once before the search for conflicting subsets is performed. We generate a permutation of C uniformly at random in $\mathcal{O}(|C|)$ time using the so-called *Knuth shuffle* [57, 105]. After that, we keep this ordering for subsets used in the recursions.

Randomized order. Instead of shuffling the list only once and then enumerating the list sequentially, we iteratively select each element randomly. This way, we run through C and its subsets in each recursion in a different order.

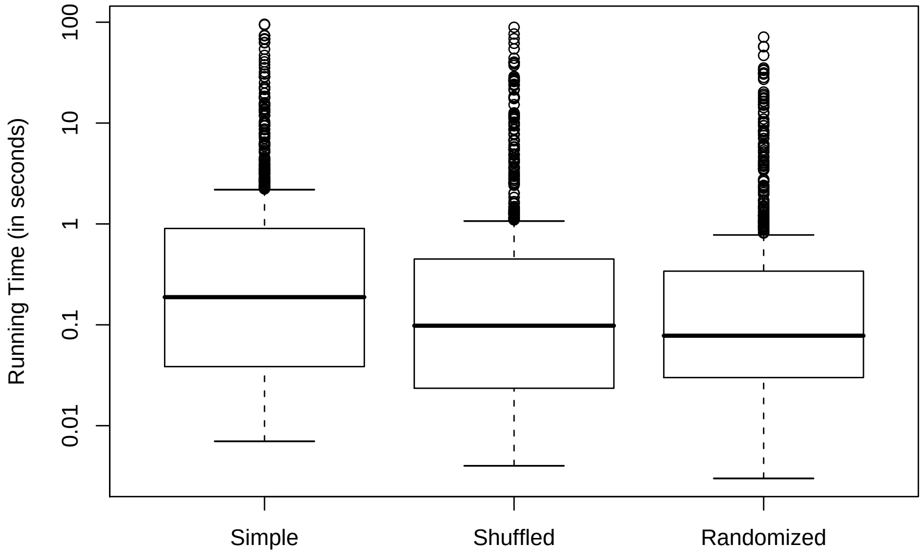
Figure 3.2 contrasts the running times of the simple fixed-order approach with the two randomized variants. In Figure 3.2(a), we see that compared to the simple approach, on average, a shuffled ordering improves the running times, and using the randomized order even gives a further, slight enhancement. This can be explained as follows. In the simple fixed-order approach, we arrange the elements in a set of clusters by their order in $\mathcal{C}$, which in turn corresponds to the order of the clusters as they are read from the input. The gene orders at the leaves of the phylogenetic tree are scanned in a sliding-window approach and new clusters are appended to the list. But clusters which occur consecutively in one input gene order cannot cause a conflict independently of clusters contained in other genomes. Thus, considering clusters genome-wise is disadvantageous, and any shuffled order performs better on average. By choosing a completely random arrangement throughout, we generally avoid adhering to a bad case.

The more conflicts occur, the stronger is this effect. Hence, the obtained improvement correlates with the size of the problem instances. To analyze the runtime improvement in more detail, we define the term *speed-up* by an approach A with respect to an approach B with associated running times t_A and t_B to refer to the ratio t_B/t_A. The comparison of the relative computation times for varying cardinality of $\mathrm{Conf}(C)$ in Figure 3.2(b) shows that the shuffled-order approach outperforms the randomized variant for smaller problem sizes, because shuffling the set only once is faster. In contrast, for bigger instances, full randomization pays off and surpasses shuffling. Since randomization yields the best overall speed-up, and in particular decreases the running times for large and thus costly instances, we select this approach for our further study.

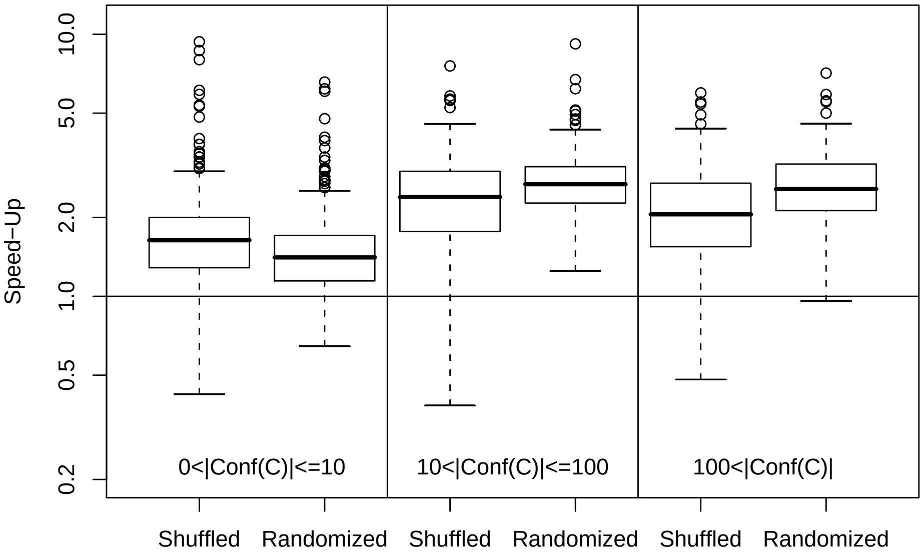
Utilize Overlaps

The following technique also modifies the order in which elements from a set of gene clusters are enumerated and can be combined with all approaches discussed above.

We say that two gene clusters *overlap* if they have at least one gene in common. Obviously, each cluster in a minimal conflicting set has to overlap with other clusters in the set. Otherwise, it would be independent and therefore not essential for the conflict. This observation



(a)



(b)

Figure 3.2: Evaluation of the running times for different approaches to enumerate elements of a set of gene clusters in lines 6 to 8 in Algorithm 2 on page 36. The calculations were performed on a 2600 Mhz Sparc processor with 32 GB of main memory. (a) The absolute running times of the three variants are compared in a box-and-whisker diagram with logarithmic time scale. (b) The speed-up by the shuffled-order and the randomized-order approach are shown with respect to the simple fixed-order approach. For different instance sizes, the ratios are plotted in a box-and-whisker diagram with logarithmic ordinate.

motivates a simple, yet effective strategy. As before, we iterate over the set of gene clusters C in one of the above mentioned ways. But instead of iteratively filtering with each cluster, we only consider those which overlap with one of the preceding clusters and skip all others. Whenever we reach the end of the list and still no conflicting set is identified, we start a new iteration over the remaining clusters until no overlapping clusters can be found.

With respect to the randomized-order approach, applying this criterion additionally results in an average speed-up of 1.77. The runtime improvement slightly decreases for larger problem sizes, as indicated in Figure 3.3.

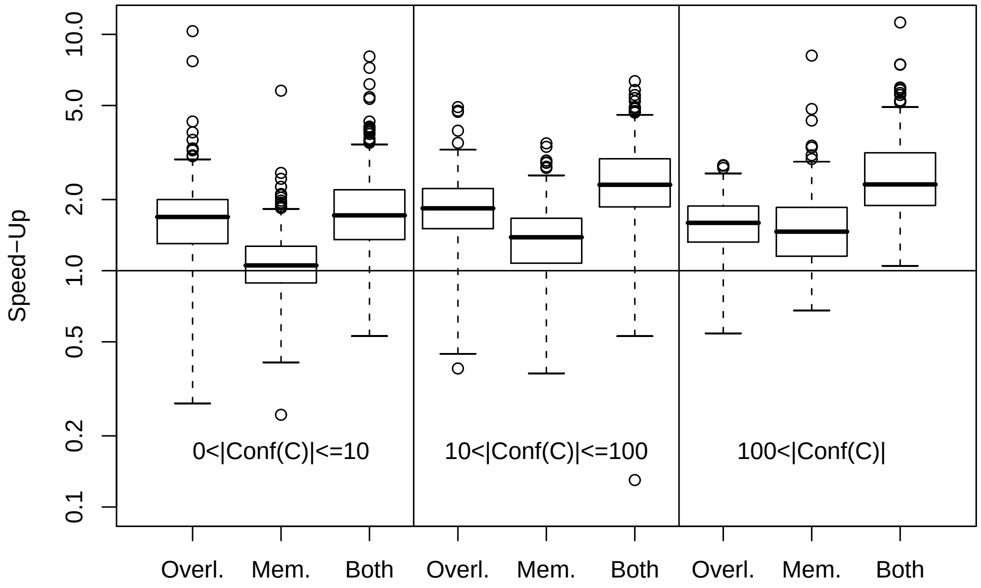


Figure 3.3: Improvement of running times for two techniques applied on the randomized variant of Algorithm 2 on page 36. The speed-up achieved by the overlap criterion (overl.), memoization (mem.), and the combination of both with respect to applying none is compared in box-and-whisker diagrams (with logarithmic ordinate) for different problem sizes.

Memoization

The last approach we discuss to enhance the search for minimal conflicting sets is *memoization*. This is a commonly known method to speed up recursive methods by avoiding function calls of already processed inputs. For instance, assume that the set C of gene clusters contains amongst others the minimal conflicting subsets $A = \{a_1, a_2 \ldots, a_{|A|}\}$. Once we have identified A, we start recursive calls to search for conflicting subsets of $C \backslash \{a_1\}$, $C \backslash \{a_2\}$ etc. If the first recursion would yield a conflict containing a_2 and the second recursion a conflicting subset containing a_1, both recursions would initiate a search for $C \backslash \{a_1, a_2\}$, which in turn would separately call identical, further recursions.

Such redundant calculations can be ruled out by providing a concurrent data structure for all recursions to record for which part of the search space a recursion has already been started. That means, we store the set of all processed subsets of $\mathcal{C}$. For this purpose, we implemented a tree data structure similar to a *trie*, which provides operations for looking up or inserting a list over $\mathcal{C}$ in $\mathcal{O}\left(|\mathcal{C}|\right)$ runtime using $\mathcal{O}\left(|\mathcal{C}|^2\right)$ space [78]. Although checking and updating this data structure before each recursive call takes additional time, this strategy altogether saves computation time.

Since memoization can be applied independently of any ordering of the gene cluster sets, it can be combined with any of the other techniques – both randomization and the overlap criterion. Figure 3.3 shows that the speed-up we gain by memoization correlates with the problem complexity. The more conflicts exist, the larger the search space is, and the more redundant computations can be saved.

All in all, the fully randomized variant combined with the overlap criterion and memoization performs best in this evaluation. The speed-up compared to the simple fixed-order approach without any additional techniques is 5.07 on average, and 6.18 if weighting with the problem size $|\operatorname{Conf}(C)|$.

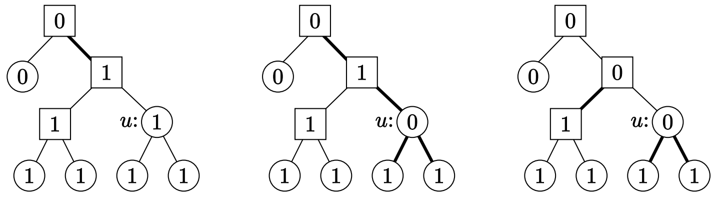
Algorithm 2 and any of the discussed variants can be used to identify all minimal conflicting subsets contained in the labeling of a node. This and the notion of a *conflict index* are essential parts of the method to find an optimal labeling, which is described in the following section.

3.3 Branch-And-Bound Search

The following method to solve the labeling problem, defined in Chapter 2, pursues the approach motivated by Theorem 1 on page 25. We can use any most parsimonious labeling λ_{par}, for example computed as described in Section 3.1, as a starting point. Then, there are two cases. Either, in the simple case, this labeling does not contain any conflicts, which means it is consistent and thus optimal by definition. Or, otherwise, it is inconsistent. Then, the labeling for some nodes in the tree contain conflicts. In this case, we explore the search space to find an optimal labeling λ_{opt} starting from λ_{par} as described next.

Given an inconsistent labeling, consistency can be reached by deleting some clusters from some nodes. For this purpose, we have to find a sequence of deletions that increases the total weight $W(T, \lambda)$ as little as possible. In general, we consider all such sequences using a recursive approach: If after a deletion, a consistent labeling is obtained, no further deletions are needed and the current labeling is stored as a potential optimum. Otherwise, further deletions are performed in a recursive branch-and-bound manner: For each cluster $c \in \mathcal{C}$ and for each node v, we make a copy of the labeling, delete cluster c from the labeling of node v, and start a new recursion with the new labeling. In the end, from all potential optima, one with lowest weight is reported as an actual optimum.

Obviously, for a most parsimonious labeling, a deletion cannot decrease the weight. But in general, during the exploration of the search space, a deletion of a cluster c from the labeling of a node v could create a labeling whose weight can be decreased by further deletions of the same cluster c in nodes neighboring v. See Figure 3.4 on the following page for an example. Since we are interested in labelings with lowest possible weight, we directly perform these further deletions in a reoptimization step. To this end, we compute a new most parsimonious



(a) Before deletion: $w(T, \lambda_c) = 1$.

(b) After deletion: $w(T, \lambda_c') = 4$.

(c) After reoptimization: $w(T, \lambda_c'') = 3$.

Figure 3.4: Example for the reoptimization after deleting a cluster. A tree T and different labelings for a specific cluster c are shown. Fixed values are depicted in round nodes. Thin edges add 0 to the weight $w(T, \lambda_c)$ and bold edges add 1. (a) T is labeled with a most parsimonious labeling λ_c with weight $w(T, \lambda_c) = 1$. (b) Deleting c from node u increases the weight to $w(T, \lambda_c') = 4$. (c) The reoptimization decreases the weight to $w(T, \lambda_c'') = 3$.

labeling for the current cluster with a fixed value for the actual node: $\lambda_c(v) = 0$. In general, this can be done by splitting the tree as shown in Figure 3.5 and performing the usual algorithm to solve the parsimony problem for these subtrees independently. If more than one labeling is fixed, we can further split the trees recursively. A recombination of the resulting parsimonious labelings will finally yield an optimal overall result, which can be shown by a simple contradiction argument.

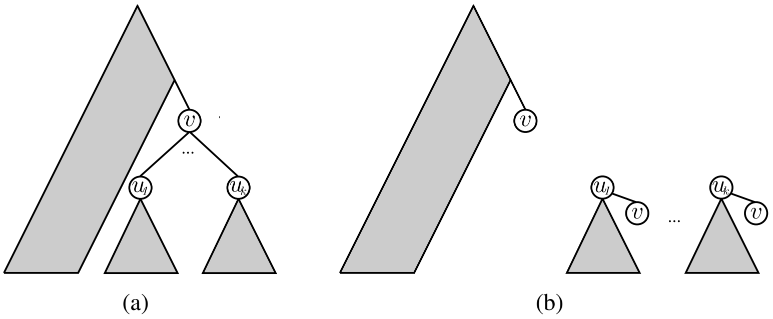


Figure 3.5: Splitting a given tree for the computation of a most parsimonious labeling for a cluster c for a fixed value at node v. Parts of the tree are depicted abstractly as polygons. (a) The complete tree. (b) In all resulting subtrees, the former internal node v is now treated as a leaf whereby its labeling gets fixed.

However, for specific parsimony weighting schemes, there exist more efficient reoptimization procedures. For example, for the Wagner parsimony function, we do not split the tree and use variants of the two phases of the Fitch-Hartigan algorithm to update those parts of the tree that are affected by the deletion. For this, the tree does not have to be traversed completely, but just as long as any changes occur as described in Algorithm 3. The following Lemma guarantees parsimony of the reoptimized labeling with respect to the fixed values for deleted clusters.

```
Algorithm 3: $\mathrm{ReOpt}(T, u)$                    Reoptimization after Deletion
Input: A rooted tree $T = (V, E)$ with each node $v \in V' \subseteq V$, including all leaves,
       labeled with a value $b_v$, and a node $u \in V'$ in which the labeling changed.
fix the new value of $u$
/* Bottom-up phase */
foreach node $v$ in a bottom-up fashion, starting with $u$ do
    if $v \in V'$ then $\mathrm{P}(v) := \{b_v\}$ and $\mathrm{S}(v) := \emptyset$
        else compute $\mathrm{P}(v)$ and $\mathrm{S}(v)$ as defined in Algorithm 1
    if $\mathrm{P}(v)$ and $\mathrm{S}(v)$ remain unchanged then do not process any further parents of $v$
/* Top-down phase */
foreach node $v$ where
    - $\mathrm{P}(v)$ or $\mathrm{S}(v)$ changed during the bottom-up phase, or
    - $\lambda_c(p)$ for parent $p$ of $v$ changed during the top-down phase
do
    if $v \in V'$ then $\lambda_c(v) := b_v$
        else compute $\lambda_c(v)$ as defined in Algorithm 1
```

Lemma 1. *Let $T = (V, E)$ be a rooted tree with each node $v \in V' \subseteq V$, including all leaves, labeled with a value b_v. Then, Algorithm 3 yields a most parsimonious labeling under the Wagner parsimony scheme with respect to the given constraints, i.e. $\lambda_c(v) = b_v$ for all $v \in V'$.*

Proof. A node with a fixed value can be seen like a leaf. Hence, the labeling of the subtree with this node as a leaf is parsimonious. The subtree rooted at this node is parsimonious, since the calculation of $B(u, a)$ and the following choice of $\lambda_c(u)$ guarantee parsimony for both possible values of $\lambda_c(v) = a$, independently of $\mathrm{P}(v)$ and $\mathrm{S}(v)$ [85]. Hence, the entire tree is most parsimonious under the given constraint of the fixed values. □

As in the computation of a parsimonious labeling (Algorithm 1 on page 33), the result of the reoptimization is not necessarily unique. For example, we could analogously follow the sparse or the dense approach. Theorem 1 also holds for instances that include values fixed to zero, since a fixation for a node v simply yields both $\lambda_c^{par}(v) = 0$ and $\lambda_c^{opt}(v) = 0$, without impairing the argumentation (cf. the corresponding proof on pages 25 and following). Hence, all possible reoptimization results allow to find an optimum by further deletions.

We perform the reoptimization after each deletion, before starting the recursion. This guarantees that the weight cannot be decreased in any recursive call. Therefore, the current recursion is stopped once the current weight exceeds the weight of the best solution found so far, as shown in Figure 3.6 on the following page. Obviously, the weight of a (consistent or inconsistent) most parsimonious labeling or a known optimum (if we are looking for further optima) can be used as a lower bound. Once a consistent labeling of such a minimal weight is found, the search can be terminated completely.

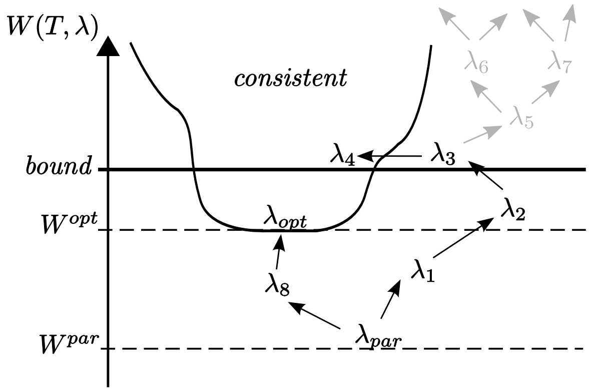


Figure 3.6: Example for the bounding. The space of all possible labelings for a given problem instance is shown including some explicit labelings. The labelings are arranged according to their weight $W(T, \lambda)$ and whether they are consistent (λ_4, λ_{opt}) or not (all others). Once a consistent labeling is found (e.g. λ_4), all labelings with higher weight are discarded (shown in gray).

Consistent labelings with low weight correspond to tight upper bounds. To lower the bound preferably quickly, we start the recursions in a promising order. To annihilate as many conflicts as possible by one deletion, we first compute the conflict index for each node for each cluster and select the candidates in descending order of their conflict index. Moreover, the reoptimization step often causes deletions of the same cluster for neighboring nodes. This again can annihilate more conflicts without a further increase of the weight. Hence, we consider the number of conflicts a cluster causes in the overall tree.

Since the correctness of the algorithm does not depend on the order of the recursions, the desired order does not have to be determined exactly. To save time in this sorting step, we just estimate the conflict index instead of computing it properly. A simple bounding of the recursion depth during the search for conflicting subsets (Algorithm 2 on page 36) already gives a suitable speed-up. A limitation to depth two results in a quadratic time complexity for each bounded search. Experiments showed that this yields the best tradeoff for estimation quality and running time.

The overall strategy is laid out in Algorithm 4. Note that, for the sake of clarity, the following technical details are omitted in the pseudo code.

The search for minimal conflicting subsets of the node labelings is not stated explicitly. Either, this search is only performed once in the very beginning, and all conflicts are identified and stored for each node separately. Then, we can simply update the corresponding data structure whenever we perform a deletion, and when no conflict is left, the labeling is consistent. Or, if we use the bounded search for conflicts and not all of them are identified, we have to repeat the search whenever all known conflicts are abolished by our deletions. In any case, since the estimation of the conflict index can be done for each node independently, we can easily parallelize this step.

Furthermore, also not mentioned in the code, *memoization* is again used to reduce the running time. The labeling resulting from a series of deletions and the corresponding reoptimiza-

```
Algorithm 4: $\mathrm{FindOpt}(T, \lambda_{par}) \longrightarrow \lambda_{opt}$    Recursive Search for an Optimal Labeling
Input: A rooted tree $T$ and a most parsimonious labeling $\lambda_{par}$.
Output: An optimal labeling $\lambda_{opt}$.
$\lambda_{opt} \leftarrow$ null, $W(T, \lambda_{opt}) \leftarrow \infty$
$\mathrm{FindOptRec}(T, \lambda_{par}, \lambda_{opt})$
return $\lambda_{opt}$

Procedure: $\mathrm{FindOptRec}(T, \lambda, \lambda_{opt})$
Input: A rooted tree $T = (V, E)$, a labeling $\lambda$, and $\lambda_{opt}$, the consistent labeling with the
       lowest weight found so far.
if $\lambda$ is consistent then $\lambda_{opt} \leftarrow \lambda$
else /* branch */
    foreach conflicting cluster $c \in \mathcal{C}$ in descending order of $\sum_{v \in V} \mathrm{CI}(\lambda(v), c)$ do
        foreach node $v \in V$ in descending order of $\mathrm{CI}(\lambda(v), c)$ do
            $\lambda'(v) \leftarrow \lambda(v) \backslash \{c\}$
            $\mathrm{ReOpt}(T, v)$ (Algorithm 3)
            /* bound or search recursively */
            if $W(T, \lambda') < W(T, \lambda_{opt})$ then $\mathrm{FindOptRec}(T, \lambda', \lambda_{opt})$
end
```

tions is independent of the order in which they are performed. Thus, during the exploration of the search space, some intermediate results are reached several times in different branches of the recursion tree. To avoid redundant recursive calls, we store and check for combinations of deletions that have already been processed.

Theorem 3. *Algorithm 4 solves the labeling problem exactly.*

Proof. The claim follows from Theorem 1, Lemma 1 and the above description. □

Although Theorem 1 guarantees that, in any case, we find a solution that is optimal, this result is not unique in general. In the following section, we show how we can take advantage of this ambiguity.

3.4 Two-Phase Approach

As already mentioned, neither the initial computation of a most parsimonious labeling, nor the reoptimization step do always yield a unique result. Depending on the decisions made, the recursive search described in the previous section may give different labelings, all of which are optimal.

We already introduced the two extremes, the sparse and the dense variant. On the one hand, the dense variant increases the sensitivity of the reconstruction by preferring to assign a cluster to a node in case of ambiguity. On the other hand, more clusters accompany more conflicts, which slows down the branch-and-bound search for an optimum. Even though these approaches seem to be converse at first sight, we now introduce a method that combines the advantages of both:

In the first phase, the sparse variant is used to get a parsimonious labeling containing as few conflicts as possible, which can then be solved very quickly by the branch-and-bound search. In the second phase, the dense variant is used to reconstruct as many clusters as possible. This tends to result in many conflicts, which have to be solved by Algorithm 4. But the branch-and-bound search can be accelerated by the result of the first phase:

- The recursion can be bounded by the known optimal weight (line 10).
- The search can be stopped completely, once a consistent labeling with the optimal weight is found (line 4).

The overall procedure is summarized in Algorithm 5. Note that each phase separately finds an optimum.

```
Algorithm 5: $\mathrm{TwoPhase}(T) \longrightarrow \lambda^{dense}_{opt}$                Two-Phase Approach
  Input: A rooted tree $T{=}(V,E)$ with each leaf $l{\in}V$ labeled with a set of gene clusters $C_l$.
  Output: An optimal labeling $\lambda^{dense}_{opt}$.

  /* Phase 1 */
  $\lambda^{sparse} \longleftarrow$ sparse variant of $\mathrm{FitchBin}(T)$
  $\lambda^{sparse}_{opt} \longleftarrow \mathrm{FindOpt}(T, \lambda^{sparse})$

  /* Phase 2 */
  $\lambda^{dense} \longleftarrow$ dense variant of $\mathrm{FitchBin}(T)$
  $\lambda^{dense}_{opt} \longleftarrow \mathrm{FindOpt}(T, \lambda^{sparse})$ bounded by $W(T, \lambda^{sparse}_{opt})$
```

Theorem 2 guarantees that for all optimal labelings there is a parsimonious labeling that can be used to find it. Furthermore, the Fitch-Hartigan algorithm finds all most parsimonious labelings if all solutions are reported. Hence, the above procedure can easily be adopted to find *all* optima. Certainly, it would not be practical to actually compute all optima. Nevertheless, since no optimum is excluded *a priori*, it would be possible to search for some particular optima that obey certain additional criteria.

3.5 Complexity

To complete the discussion of our method to solve the labeling problem exactly, we now want to examine its overall complexity.

As explained in Section 3.1, the running time of the Fitch-Hartigan algorithm to compute a most parsimonious labeling is linear in the number of nodes in the tree times the cardinality of the cluster set: $\mathcal{O}\left(|V||\mathcal{C}|\right)$. The latter parameter, in turn, depends on the gene cluster model in use. The number of clusters contained in a gene order, with respect to its length, ranges from linear for simple cluster models to exponential for very complex models. For instance, concerning the model of common intervals on permutations introduced on page 15, a genome of length N contains $N-k+1$ common intervals of size k, and thus $N(N-1)/2-1$ clusters in total. Further cluster models will be discussed in the following chapters.

Beside *parsimony*, we also have to account for consistency issues, which in contrast contribute exponential components to the overall problem even for simple gene cluster models. As already mentioned in Section 3.2, identifying all minimal conflicting subsets takes an exponential number of filtration operations, which we try to avoid by substituting a complete calculation by a repeated, bounded estimate of the conflict index.

However, finally, the overall running time is dominated by the branch-and-bound search. In the worst case, we assume a parsimonious labeling for which all clusters assigned to each node belong to some conflict. We further assume that we have to delete D of $|\mathcal{C}|$ clusters to reach consistency for one node. This yields $\binom{|\mathcal{C}|}{d}$ different combinations of d deletion operations for each $0 \leq d \leq D$, which is already exponential in $|\mathcal{C}|$. Furthermore, since we have to ensure consistency for all nodes, we have to raise this number to the power of $|V|$.

In the best case, even the dense, most parsimonious labeling is consistent *per se*. Then, consistency is verified with $\mathcal{O}(|\mathcal{C}|)$ filtrations, and no further recursions have to be performed since we know that the weight is minimal and the labeling is optimal.

In contrast to this artificial extreme cases, we observe the following pattern in practice. In the first step of the two-phase algorithm (Algorithm 5), the labeling λ^{sparse} computed by the sparse variant of the Fitch-Hartigan algorithm rarely contains any conflicts. Hence, in most cases, we get a first optimal labeling without any branch-and-bound search: $\lambda^{sparse} = \lambda^{sparse}_{opt}$. Additionally, since the weight of the optimum is equal to the weight of the most parsimonious labeling, the bound for the second phase is very low. In fact, only deletions that do not increase the weight are followed up and the recursion tree is shallow. If not too many conflicts have to be resolved, this allows to find the first consistent labeling λ^{dense}_{opt} quickly, and in the considered case this labeling is optimal. However, for some instances the dense labeling λ^{dense} in the second phase comprises an infeasible amount of conflicts. In this case, we can either reduce the problem complexity by only considering gene clusters within a certain range of size, or we have to be content to exclusively calculate the sparse optimum.

In any case, despite the theoretical exponential runtime complexity, we can compute suitable results even for big, complex instances. This can be seen in both the evaluation on simulated data as part of Chapters 4 and 5, and in the discussion of a real biological dataset in Chapter 6.

Chapter 4

Permutation-based Gene Cluster Models

In Chapter 2, we introduced an abstract gene cluster model that comprises a variety of concrete cluster definitions which are studied in the literature. Since both the formalization of the problem and the corresponding methods presented in Chapter 3 are based on the abstract model, they are theoretically applicable to any of these different definitions of gene clusters. We now discuss some concrete cluster models, elaborate on how they can be integrated in our framework, investigate whether and how this can be done efficiently, and finally evaluate the effectiveness and efficiency on simulated data.

Different definitions of gene clusters can be roughly classified by their underlying genome model, i.e. the mathematical formalization of gene order. Either, all genomes under consideration contain the same genes and each of these exactly once – in other words, all gene orders are permutations of each other. Or, without such restrictions, gene order is modeled as an arbitrary sequence of genes, such that a genome may contain a gene once, multiple times, or not at all.

In this chapter, we confine ourselves to gene clusters that are defined on permutations before we discuss sequence-based models in the next chapter.

This chapter is mainly divided into two parts. Firstly, we introduce and elaborate on four different, permutation-based gene cluster definitions in the context of our abstract framework in Sections 4.1 to 4.4. Then, in Section 4.5, we evaluate the method and compare the considered models. Beside others, we analyze sensitivity, precision and running time obtained on simulated data.

4.1 Adjacencies on Permutations

Probably the simplest formalization of co-localization of genes is the concept of *adjacencies*, i.e. two directly neighboring genes. This elementary pattern of gene order conservation, also known as *gene pairs* or *neighboring genes*, has been widely used in whole genome comparison. In the following, we only give examples for its application. The popularity of this model can be explained by its simple combinatorial structure, which enables to design space-saving data structures and fast algorithms, allowing to process also large data sets.

Dandekar *et al.* [51] found that “proteins encoded by conserved gene pairs appear to interact physically”. In line with that, Overbeek *et al.* [126] studied runs of adjacent genes and confirmed functional coupling of respective proteins. Keogh *et al.* [102] considered conserved adjacencies to analyze the evolution of gene order, and especially to predict a whole genome duplication event in the lineage of several fungi. Won *et al.* [162] used broken adjacencies as an indicator for genome rearrangement to finally reconstruct the rearrangement history of yeast genomes. The notion of broken adjacencies leads us to the idea of counting genes which are adjacent in one genome, but not in a second, as a measure of divergence of the respective genomes, known as the *breakpoint distance*. Related problems, such as finding a gene order that minimizes this distance to three given gene orders [25], known as the median problem, are well-studied. A corresponding data structure, the so-called *breakpoint graph*, finds its application in different rearrangement problems, for instance the inversion distance problem [17].

Since adjacencies are particularly suitable to efficiently represent genomes as sets of binary state characters as described in Section 2.2.2, various existing methods for phylogenetic analysis have been used [148]. Especially in the field of gene order reconstruction, this model is one of the most prevalent concepts [24, 44, 119, 123].

We now demonstrate how adjacencies can be easily integrated into our framework by defining a concrete instantiation of the abstract gene cluster model introduced in Section 2.1 on page 14.

Model 1. (Unsigned Adjacencies on Permutations) *Let N be an arbitrary but fixed length of the gene orders and $\mathcal{G}_N := \{1, \dots, N\}$ the set of all genes. Then, the gene cluster model of* unsigned adjacencies on permutations *is defined as $(\mathcal{U}_u, \mathcal{C}_u, \sqsubseteq_u)$, where*

- *$\mathcal{U}_u$ is the set of all permutations over $\mathcal{G}_N$;*
- *$\mathcal{C}_u$ is the set of all pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N, a \neq b$; and*
- *$\{a, b\} \sqsubseteq_u g :\Leftrightarrow a$ and b are adjacent in g.*

This definition fully describes the labeling problem on adjacencies formally and therefore allows us to solve it using the methodological framework introduced in the previous chapter. However, to substantially reduce the complexity of the most central operation, the filtering, we show how it can be realized efficiently for this specific gene cluster model. Instead of naively checking for each genome in $\mathcal{U}$ if it contains a given adjacency, we use the following, graph theoretic approach.

4.1.1 Efficient Filtering

During the calculation of all minimal conflicting subsets as laid out in Algorithm 2 on page 36, we begin all filtering procedures with the universal set and iteratively select further clusters. Thus, we only have to consider calculations of $G \mathbin{\dot{\vee}} c$ for such subsets $G \subsetneq \mathcal{U}$ that are results of preceding filter operations.

We construct an undirected graph $G = (V, E)$ where each gene $i \in \mathcal{G}_N$ is uniquely represented by one vertex $v_i \in V$. At the very beginning, when the universal set has not yet been filtered with any adjacency, the edge set E is empty. For each adjacency $\{a, b\}$ we filter with, an edge $\{v_a, v_b\}$ is added to E connecting the corresponding vertices. A path in the graph represents the order of the contained genes as they have to appear in all genomes of the resulting genome set. If we could connect all edges and nodes in the graph to one acyclic path by adding further edges, such a path would correspond to a genome containing all required adjacencies. To keep track whether such a valid gene order exists or the resulting genome set is empty, we test two properties of the graph whenever we include a new adjacency:

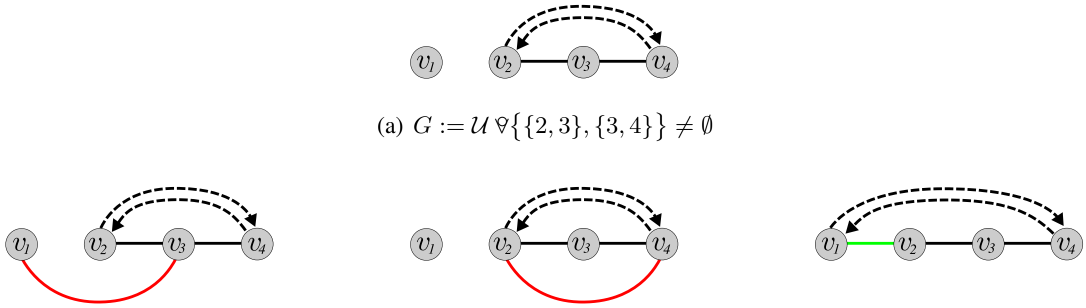
1. To ensure that the graph contains no furcations, we count how many edges are incident to each node. When the degree of any node exceeds two, no valid path and thus no valid gene order can be built. Since each time, only the degree of two nodes has to be updated and tested, this takes constant time per adjacency.
2. Moreover, we have to assure that the graph contains no cycles. This can also be done in constant time: We include one pointer from the first to the last node of each maximal path and another one pointing in the opposite direction. Recall that due to the maximal degree of two, each node can only be part of at most one maximal path. If a newly added edge would connect two nodes pointing at each other, this would create a cycle. Otherwise, the edge is added and the pointers can be updated easily.

If induced by one of the two cases, we return an empty genome set. Else, the graph represents all valid gene orders and can be used for further filtering steps. See Figure 4.1 on the following page for an illustration.

The graph can be stored in $\mathcal{O}(N)$ space and, including its initialization, we can perform k filter operations in $\mathcal{O}(N + k)$ time. A permutation of length N contains $N - 1$ adjacencies. Hence, after at most N filtrations, we get the empty set and stop filtering. This yields an overall time complexity of $\mathcal{O}(N)$. If we assume k to be much smaller than N, we could save the $\mathcal{O}(N)$ time for the initialization of the nodes. Instead, we create new nodes online whenever an edge containing this node is added to the graph. Using a hash table of size $\mathcal{O}(k)$, we can add and access nodes in constant time, which results in an overall time and space complexity of $\mathcal{O}(k)$.

4.1.2 Model Variants

A slightly more sophisticated variant of the adjacency model is motivated by the observation that the orientation of genes can play a role in co-expression and also in gene order conservation. As expected, a comparison of gene pairs between the prokaryotes *H. influenzae* and *E. coli* showed that genes encoded in the same direction (➡ ➡) are more often conserved than



(a) $G := \mathcal{U} \forall \{\{2,3\},\{3,4\}\} \neq \emptyset$

(b) $G \forall \{1,3\} = \emptyset$

(c) $G \forall \{2,4\} = \emptyset$

(d) $G \forall \{1,2\} \neq \emptyset$

Figure 4.1: Example for the graph representation of a genome set under the unsigned adjacency model for $N = 4$. Edges are depicted as solid lines whereas pointers are illustrated as dashed arrows. (a) The result of filtering the universal set with the two adjacencies $\{2,3\}$ and $\{3,4\}$. (b) Filtering the graph in (a) with $\{1,3\}$ would imply three neighbors for gene 3, which violates the constraints of a permutation and thus yields an empty genome set. (c) Filtering the graph in (a) with $\{2,4\}$ would imply a cycle, which is also not consistent with the genome model and yields an empty genome set. (d) In contrast, filtering with $\{1,2\}$ extends the path, the pointers are updated, and the resulting graph represents the genome set $G \forall \{1,2\} = \{(1,2,3,4),(4,3,2,1)\}$.

other gene pairs. In contrast, for eukaryotes a different pattern was observed. Surprisingly, in *C. albicans* and *S. cervisiae*, genes encoded convergently (➡ ⬅) are as often conserved as genes encoded in the same direction. For further details of these studies, see the review of Huynen *et al.* [96].

Independently of the actual pattern of orientation, these studies suggest that it is reasonable to consider the relative orientation of genes when we model gene order conservation.

Model 2. (Signed Adjacencies on Signed Permutations) *Again, let N be the length of the genomes and $\mathcal{G}_N := \{1,\dots,N\}$ the set of all genes. Then, the gene cluster model for* signed adjacencies on signed permutations *is defined as $(\mathcal{U}_s, \mathcal{C}_s, \sqsubseteq_s)$, where*

- *$\mathcal{U}_s$ is the set of all signed permutations over $\mathcal{G}_N$;*
- *$\mathcal{C}_s$ is the set of all pairs $\{a,b\}$ with $a,b \in \{i,-i \mid i \in \mathcal{G}_N\}, a \neq b$; and*
- *$\{a,b\} \sqsubseteq_s g :\Leftrightarrow$ in g, a is directly followed by $-b$, or b by $-a$.*

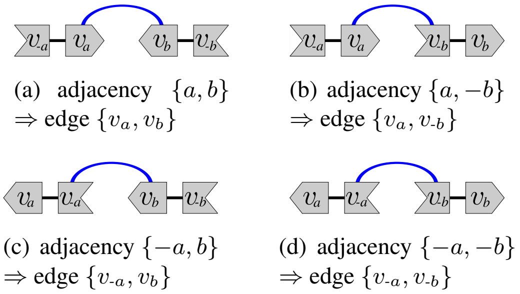
Note that the representation of a signed adjacency as an unordered pair is accurate since the definition of $\sqsubseteq_s$ does not depend on the actual assignment of a and b.

Example 9. In Model 2 for $N = 4$, the genome $(1,-2,-3,4)$ contains exactly the adjacencies:

$$\{1,2\},\{-2,3\},\{-3,-4\}\,.$$

$\diamond$

The graph theoretic representation introduced for the filtering in the unsigned model can be extended to the signed case as follows. For each gene i, we create two nodes: v_i for the 'head' and $v_{\text{-}i}$ for the 'tail' of the gene, and connect them by an edge. When we now include a signed adjacency $\{a, b\}$ into the graph, the sides of the genes $|a|$ and $|b|$ we have to connect correspond to the vertices v_a and v_b, as illustrated in Figure 4.2. Apart from that, the filtering method as well as its space and time complexities remain unchanged.



(a) adjacency $\{a, b\}$ ⇒ edge $\{v_a, v_b\}$

(b) adjacency $\{a, -b\}$ ⇒ edge $\{v_a, v_{\text{-}b}\}$

(c) adjacency $\{-a, b\}$ ⇒ edge $\{v_{\text{-}a}, v_b\}$

(d) adjacency $\{-a, -b\}$ ⇒ edge $\{v_{\text{-}a}, v_{\text{-}b}\}$

Figure 4.2: Including a signed adjacency of the genes a and b ($a, b > 0$) into the graph representation of the signed adjacency model. (a–d) The four different cases of relative orientation of the genes and the corresponding new edge are shown.

Recall that, in our framework, the filter operation is used to find all minimal conflicting subsets of a given set of gene clusters. For the basic gene cluster models discussed in this section, we could also bypass the filtering-based Algorithm 2 and instead use the graph representation introduced above to identify the conflicts directly.

To this end, we proceed as explained but do not stop adding edges when the genome set is empty. We rather include *all* adjacencies. After that, each minimal cycle in the graph and each triple of edges incident to one node corresponds to a minimal conflicting set of adjacencies. As already mentioned, if there is no such substructure in the graph, valid gene orders exist. Hence, the identification of these two classes of conflicts is sufficient. Enumerating all t conflicting triples can be done in time linear in t, whereas all c minimal cycles can be determined in $\mathcal{O}\left((N+k)(c+1)\right)$ time using $\mathcal{O}\left(N+k\right)$ space, where N is the number of genes and k the number of adjacencies [101]. This results in an overall runtime of $\mathcal{O}\left(t+(N+k)(c+1)\right)$ to identify all minimal conflicting subsets of a given set of adjacencies.

As mentioned in the Introduction, genomes can be composed of one or multiple circular chromosomes. Hence, we might also want to model adjacencies on circular permutations and thus modify the mentioned methods correspondingly. This can easily be done by relaxing the cycle constraint to allow for one big cycle in the graph or even generally skip this restriction at all to model multiple circular chromosomes.

As we will see in the following Sections 4.2, 4.3 and 5.1.2, the model of adjacencies on permutations can be handled as a special case of other, more general models – both the unsigned and the signed case. Even though for adjacencies on permutations, the methods discussed here are more efficient, in practice, we refrain from a separate implementation and resort to the generalized methods.

4.2 Common Intervals on Permutations

To find larger conserved regions, we now address a model for gene clusters that, in contrast to adjacencies, generally span more than two genes. In 1998, Dandekar *et al.* [51] found that in operons, even though their gene content may be conserved across species, the order of genes is not necessarily preserved. Such observations motivate the notion of *common intervals*, segments of the genome containing the same set of genes in an arbitrary order but not interrupted by other genes.

The term *interval* stems from the original, mathematical problem statement. There, a common interval is defined on a set of permutations which is, without loss of generality, assumed to include the identity $(1, \dots, N)$ (otherwise all genes can be renamed appropriately). In this case, a set of genes contiguous in all genomes has to appear contiguously in the identity permutation as well and thus be of the form $\{i, i+1, \dots, i+l\}$, which corresponds to an interval $[i, i+l]$.

The basic problem of enumerating all common intervals of two permutations and respective algorithms were proposed in 2000 by Uno and Yagiura [157]. In 2001, Heber and Stoye [87, 88] presented an efficient extension for finding common intervals in multiple permutations as well as in signed, circular and multichromosomal gene orders. Their algorithm is based on an effective representation of a set of common intervals by a generating subset of so-called *irreducible* common intervals. Bergeron *et al.* [12] introduced the theory of *strong* common intervals as another basis of representation and developed corresponding algorithms. For details, we refer to the recent review of Bergeron *et al.* [13].

Common intervals are widely used in the context of genome rearrangement. Beside utilizing their combinatorial structure for a heuristic approach to find sorting scenarios [22], they have been integrated into other problems. For example, the general sorting problem – finding a shortest series of operations that rearrange one gene order into the other – is refined to *perfect sorting* where only scenarios are considered that do not break any common intervals [7, 15, 38, 55, 76]. Analogously, the general median problem can be restricted to the *preserving* median problem [21].

Also in gene order reconstruction, common intervals are used. Adam *et al.* [1] proposed a method similar to our approach. First, a most parsimonious labeling, minimizing the symmetric set distance, is computed before conflicts are removed. In contrast to the framework presented here, their algorithm is restricted with respect to the gene cluster model and the parsimony weight function. Furthermore, no overall objective is defined and both steps of the methods are heuristic.

Another approach to reconstruct ancestral gene orders has been developed recently by Chauve and Tannier [44]. Their methodological framework is based on adjacencies and common intervals to compute conflict-free contiguous regions of one specific ancestral genome.

We incorporate common intervals into our optimization scheme as follows. Note that this model has already been introduced exemplary in Section 2.1 on page 15. We now repeat this definition.

Model 3. (Common Intervals on Permutations) *Let N be an arbitrary but fixed length of the gene orders and $\mathcal{G}_N := \{1, \dots, N\}$ the set of all genes. Then, the gene cluster model of* common intervals on permutations *is defined as* $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$*, where*

- *$\mathcal{U}_c$ is the set of all permutations over $\mathcal{G}_N$;*
- *$\mathcal{C}_c$ is the set of all subsets $c \subsetneq \mathcal{G}_N$ with $|c| > 1$; and*
- *$c \sqsubseteq_c g :\Leftrightarrow$ all genes in c occur contiguously in genome g.*

As mentioned above, trivial clusters like singletons and clusters of size N would not provide any deeper insight and are therefore excluded from the definition. Note that a common interval of size two is equivalent to an unsigned adjacency. A concrete instance for a specific N is given in Example 2 on page 15.

4.2.1 Efficient Filtering with PQ Trees

Already in 1976, Booth and Lueker [31] developed *PQ trees* to efficiently represent and operate on sets of permutations. This data structure has been introduced to solve the *consecutive arrangement problem*: Given a set of objects O and a set S of subsets $s \subseteq O$, find a permutation of O in which each subset s occurs as a substring. Using PQ trees, this task can be solved in linear time – more precisely in time $\mathcal{O}\left(|O| + |S| + \sum_{s \in S} |s|\right)$. The consecutive arrangement problem is for example related to physical mapping and graph planarity and is also well suitable in the context of common intervals, as proposed by Landau *et al.* [109]. In fact, a PQ tree can be used to represent a set of gene orders.

Definition 19. (PQ Tree) *A* PQ tree *over a set O is a rooted, ordered, labeled tree with three types of nodes: Leaves,* P-nodes *and* Q-nodes*. The leaves are labeled with elements of O such that there is a bijection from the set of leaves to the set O. Any tree consisting of only one P-node with all elements of O as its children is a* universal tree*, usually denoted by T_U. The* empty tree *T_ϵ is a P-node with no children.*

Definition 20. (Properties of PQ Trees) *Let T and T' be PQ trees over O. They are* equivalent*, denoted by $T_1 \equiv T_2$, if and only if one can be transformed into the other by a sequence of the following operations:*

- *Arbitrarily permute the child nodes of a P-node.*
- *Reverse the order of the child nodes of a Q-node.*

The frontier *$F(T)$ is the permutation of O obtained by reading the leaves of T from left to right. The* permutation set *of T is defined as the set of frontiers of any tree equivalent to T:*

$$\mathrm{Perm}(T) := \left\{F(T') \mid T' \equiv T\right\} .$$

By definition, all universal trees over a set O are equivalent. The empty tree has no frontier and its permutation set is empty.

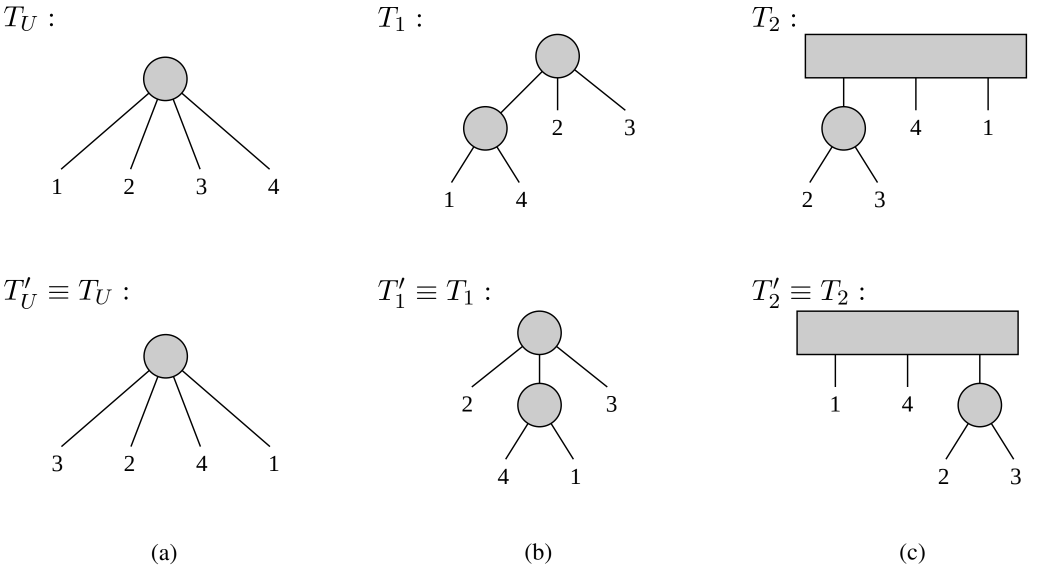


Figure 4.3: Illustration of different PQ trees over the set $O := \{1, 2, 3, 4\}$. As conventional, P-nodes are depicted as circles whereas Q-nodes are depicted as rectangles. (a) Two (equivalent) universal trees T_U and T'_U are shown, with the frontiers $F(T_U) = (1, 2, 3, 4)$ and $F(T'_U) = (3, 2, 4, 1)$, respectively. The permutation set $\mathrm{Perm}(T_U) = \mathrm{Perm}(T'_U)$ corresponds to the set of all permutations over O. (b) and (c) Two equivalent PQ trees are shown each. The set $\mathrm{Perm}(T_1) = \mathrm{Perm}(T'_1)$ consists of twelve permutations, and $\mathrm{Perm}(T_2) = \mathrm{Perm}(T'_2)$ consists of four permutations.

In Figure 4.3, different trees are visualized to exemplify the definitions given above. They can be directly translated for our purpose to model common intervals on permutations. A tree over the set of genes $\mathcal{G}_N$ is used to represent a subset of genomes as follows. The permutation set of a tree corresponds to a set of gene orders, where the permutation set of a universal tree corresponds to the set of all genomes, the universal set $\mathcal{U}_c$.

Furthermore, in our context, the consecutive arrangement problem is equivalent to the computation of a genome set where S is a set of common intervals $C \subsetneq \mathcal{C}_c$. Then, the proper orderings correspond to the genome set $\mathrm{GS}(C)$. As already mentioned, the consecutive arrangement problem can be efficiently solved using PQ trees. To this end, a *reduce* function, which in turn relates to our definition of *filtering*, is iteratively applied starting with a universal tree.

Definition 21. (Reduce Function) *Let T be a PQ tree over O and $s \subsetneq O$. Then,* $\mathrm{Reduce}(T, s)$ *computes a PQ tree T' such that*

$$\mathrm{Perm}(T') = \{f \in \mathrm{Perm}(T) \mid \textit{the elements of } s \textit{ appear as a substring of } f\}\,.$$

Finally, reducing the universal tree is equivalent to filtering the universal set, as it is needed for the identification of conflicting sets.

Corollary 1. *Let T_U be a universal PQ tree over $\mathcal{G}_N$, and $C = \{c_1, \ldots, c_{|C|}\} \subsetneq \mathcal{C}_c$ a set of common intervals. Further, let T_k be a k^{th} ($k \leq |C|$) reduction of T_U:*

$$T_k := \mathrm{Reduce}(\cdots \mathrm{Reduce}(\mathrm{Reduce}(T_U, c_1), c_2) \cdots, c_k)$$

Then, the following equation holds:

$$\mathrm{Perm}(T_k) = \mathcal{U}_c \forall \{c_1, \ldots, c_k\}, \textit{ and in particular}$$
$$\mathrm{Perm}(T_{|C|}) = \mathcal{U}_c \forall C\,.$$

Example 10. We model common intervals on permutations with $N = 4$, as detailed in Example 2 on page 15, in the PQ tree framework. For this purpose, we define the set of genes as the object set $O := \mathcal{G}_N = \{1, 2, 3, 4\}$.

In Figure 4.3(a), two equivalent universal trees over O corresponding to $\mathcal{U}_c$ are depicted. Let $c_1 = \{1, 4\}$, $c_2 = \{1, 3, 4\}$ and $c_3 = \{1, 2, 3\}$ be subsets of O (or common intervals, respectively). For $C := \{c_1, c_2, c_3\}$, we get:

$$\mathrm{Reduce}(T_U, c_1) \longrightarrow T_1 \text{ (cf. Figure 4.3(b))}$$
$$\mathrm{Reduce}(T_1, c_2) \longrightarrow T_2 \text{ (cf. Figure 4.3(c))}$$
$$\mathrm{Reduce}(T_2, c_3) \longrightarrow T_\epsilon\,.$$

According to Corollary 1, the permutation sets correspond to the respective filtered genome sets:

$$\begin{aligned}
\mathrm{Perm}(T_U) &= \mathcal{U}_c \\
\mathrm{Perm}(T_1) &= \mathcal{U}_c \dot{\forall}\, c_1 \\
\mathrm{Perm}(T_2) &= \mathcal{U}_c \forall \{c_1, c_2\} \\
\mathrm{Perm}(T_\epsilon) &= \mathcal{U}_c \forall C = \mathrm{GS}(C) = \emptyset\,.
\end{aligned}$$

◇

For a formal discussion and further relations between PQ trees and common intervals, we refer to the work of Landau *et al.* [109].

Beside the efficient implementation of the filtering operation, PQ trees also provide a compact representation of a genome set, and thus of a node labeling in the phylogenetic tree. Instead of actually annotating a node with the set of reconstructed common intervals, we can use the corresponding PQ tree which represents all putative gene orders.

A similar data structure, called *PC tree*, was introduced by Hsu and McConnell [93] to model the *circular* consecutive arrangement problem. Here, the order of the leaves of an unrooted tree represents a circular permutation. Analog to PQ trees, the ordering of subtrees incident to a P-node can be shuffled arbitrarily. In contrast, a subtree rooted at a *C-node* can be cut out of the overall tree at any of its edges to reverse the order of the remaining edges and finally reconnect it by the formerly disconnected edge. The frontiers of PC trees which are equal by means of these transformations again define a set of permutations, and the reduce function can be defined analogously, too. Hsu and McConnell gave a linear time algorithm to compute the reduce function, which allows us to efficiently model common intervals on circular permutations as well. In fact, the latter algorithm is less complex than the method for reducing PQ trees proposed by Booth and Lueker. Further, PC trees can be easily adopted to model linear permutations by just including a sentinel, i.e. an object which is not contained in the set O. For further details on PC trees, the reduce algorithm, and their connection to PQ trees and graph theory, we suggest the "Handbook of Data Structures and Applications" [94, chapter 32].

4.2.2 Generalized Models

The gene cluster model of common intervals is popular in gene-order-based genome comparison. Apart from its original, basic definition discussed in this section, meanwhile, modified and generalized models have been developed. Two permutation-based variants will be discussed in the subsequent sections, and common intervals on sequences will be introduced in the following chapter. Further well-known models, which can be defined on permutations, are generalized models like

r-window clusters, where a region of given size has to contain at least a specified set of genes,

max-gap clusters, where the distance between the regions containing a specified set of genes is restricted by a given gap size (cf. *gene teams* [14]), or

approximate common intervals, where the exact gene content of a cluster may vary among the genomes to a certain extent [30, 41]. Although this model is originally defined on sequences, it could also be applied to permutations.

Among others, the first two models are recapitulated by Hoberman and Durand [91], who also survey general desiderata and properties of several gene cluster definitions.

Our abstract gene cluster model allows to incorporate the mentioned models into our reconstruction framework. Nevertheless, computing an optimal labeling by such a straightforward integration would be infeasible due to the manifold combinatorial structure of these cluster definitions. But other, practical methods to solve the labeling problem for a specific cluster model might be found.

4.3 Framed Common Intervals on Permutations

As the term *framed common interval* already implies, this concept is an extension of common intervals. It is also related to signed adjacencies, because we consider common intervals framed by two genes whose orientations have to be conserved.

Since this combinatorial structure has been defined for different purposes, it is known under different terms. In the context of gene clusters, it has first been introduced in 2003 by Bergeron and Stoye [18, 19] as *conserved intervals*. They analyzed combinatorial properties, gave an efficient algorithm to find all intervals of two permutations, and revealed links to genome rearrangement.

In fact, already in 1996, Hannenhalli [84] defined the same structure under the name *subpermutations*. He showed that the number of framed common intervals contained in two permutations is linked to their translocation distance, and, based on this relation, proposed polynomial time algorithms to compute the distance as well as a minimal sorting sequence, which were later corrected by Bergeron *et al.* [16]. A similar connection of framed common intervals and genome rearrangements has been shown and utilized in the context of sorting by reversals [8, 9] and sorting by transpositions [50].

Analogously to common intervals, also framed common intervals have been used to refine the general rearrangement median problem to finding a median under the restriction that no interval is destroyed in the rearrangement scenario [20].

In gene order reconstruction, framed common intervals have been the first model to formally state the problem of finding putative ancestral sets of gene clusters preserving consistency. This framework was set by Bergeron *et al.* [10, 11]. Their method, in some sense, follows the set theoretic approach of Fitch and Hartigan to find a most parsimonious labeling of a tree. However, it does not aim at optimizing an objective function.

With the following model definition, we assimilate framed common intervals into our unified reconstruction framework.

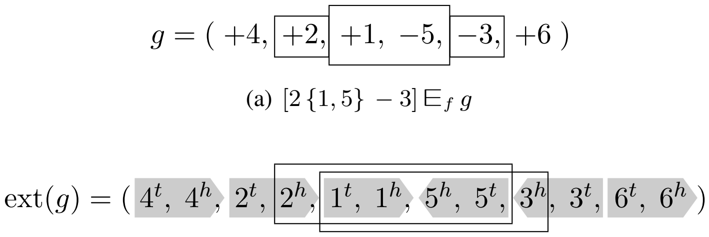
Model 4. (Framed Common Intervals on Signed Permutations) *Let N be an arbitrary but fixed length of the gene orders and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. A* framed common interval *$[a\, I\, b]$, defined on signed permutations, consists of two* extremities *a and b with $|a| \neq |b|$ and $|a|, |b| \in \mathcal{G}_N$, and a set of* inner elements *$I \subseteq \mathcal{G}_N \backslash \{|a|, |b|\}$. We say that $[a\, I\, b]$ is* contained *in a signed permutation π, if and only if in π, a is followed by b or $-b$ by $-a$, and in-between these extremities, exactly the elements of I occur in arbitrary order and with arbitrary signs. Then, the gene cluster model of* framed common intervals on signed permutations *is defined as $(\mathcal{U}_f, \mathcal{C}_f, \sqsubseteq_f)$, where*

- *$\mathcal{U}_f$ is the set of all signed permutations over $\mathcal{G}_N$;*
- *$\mathcal{C}_f$ is the set of all framed common intervals over $\mathcal{G}_N$; and*
- *$[a\, I\, b] \sqsubseteq_f g :\Leftrightarrow [a\, I\, b]$ is contained in g.*

For a short example including a graphical representation of a framed common interval, see Figure 4.4(a) on the following page. Note that a framed common interval of size two $[a\, \{\}\, b]$ is equivalent to a signed adjacency of its extremities $\{a, -b\}$.

Although framed common intervals are defined on signed permutations whereas common intervals are defined on unsigned permutations, their conspicuous relation extends to the efficient computation of the filter operation. A technique similar to those for common intervals as explained above, directly defined on PQ trees, was developed by Bergeron *et al.* [10]. In contrast, our explanations will be more abstract and, in particular, not restricted to one specific data structure.

We map framed common intervals on signed permutations over $\{1, \ldots, N\}$ to a variant of common intervals on unsigned permutations over the set $\{1^h, 1^t, \ldots, N^h, N^t\}$. In this formulation, each gene is represented by two elements – its ‘head’ and its ‘tail’. The appearance of


$g = (+4, +2, +1, -5, -3, +6)$
$\mathrm{ext}(g) = (4^t, 4^h, 2^t, 2^h, 1^t, 1^h, 5^h, 5^t, 3^h, 3^t, 6^t, 6^h)$

(a) $[2\,\{1,5\}\,-3] \sqsubseteq_f g$

(b) $c_a \sqsubseteq_c \mathrm{ext}(g)$ and $c_b \sqsubseteq_c \mathrm{ext}(g)$

Figure 4.4: Example for the mapping of framed common intervals on signed permutations to common intervals on unsigned permutations over the gene set $\{1, \ldots, 6\}$. (a) A signed permutation g is given, and the framed common interval $[2\,\{1,5\}\,-3]$, contained in g, is illustrated by a *box diagram*. The inner elements are enclosed by a big rectangle whereas the framing elements are enclosed by small rectangles. (b) The corresponding extended unsigned permutation $\mathrm{ext}(g)$ is shown. Each pairs of elements representing a signed gene is highlighted by a gray arrow. The common intervals $c_a = \{3^h, 1^t, 1^h, 5^t, 5^h\}$ and $c_b = \{2^h, 1^t, 1^h, 5^t, 5^h\}$ as defined in Observation 2 are depicted as rectangles.

$(\ldots, +a, \ldots)$ in a signed permutation g corresponds to $(\ldots, a^t, a^h, \ldots)$ in an unsigned permutation and $(\ldots, -a, \ldots)$ to $(\ldots, a^h, a^t, \ldots)$. In the following, let $\mathrm{ext}(g)$ be this extended, unsigned representation of g.

Observation 2. *Let $(\mathcal{U}_f, \mathcal{C}_f, \sqsubseteq_f)$ be the gene cluster model of framed common intervals on signed permutations and $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$ be the model of common intervals on unsigned permutations over $\{1^h, 1^t, \ldots, N^h, N^t\}$. Further, let $[a\,I\,b] \in \mathcal{C}_f$ be any framed common interval, $g \in \mathcal{U}_f$ a signed permutation and* $\mathrm{ext}(g)$ *the corresponding unsigned permutation. Then, the following relation holds:*

$$[a\,I\,b] \sqsubseteq_f g \;\Leftrightarrow\; c_a \sqsubseteq_c \mathrm{ext}(g) \;\wedge\; c_b \sqsubseteq_c \mathrm{ext}(g)\,,$$

$$\textit{with } c_a := I \cup \begin{cases} \{a^h\} & \textit{if } a > 0, \\ \{a^t\} & \textit{otherwise;} \end{cases}$$

$$\textit{and } c_b := I \cup \begin{cases} \{b^t\} & \textit{if } b > 0, \\ \{b^h\} & \textit{otherwise.} \end{cases}$$

This enables us to reduce filtering with one framed common interval to filtering with two common intervals. But before we actually redefine filtering with any interval, we have to ensure that both elements, head and tail, corresponding to one gene in a signed permutation occur consecutively in any unsigned permutation. This can easily be done by filtering the universal set of unsigned permutations: $\mathcal{U}'_c := \mathcal{U}_c \forall \big\{\{a^h, a^t\} \mid a \in \mathcal{G}_N\big\}$. Then, any filtering $\mathcal{U}_f \forall \cdots \forall [a\,I\,b]$ can be mapped to the filter operations $\mathcal{U}'_c \forall \cdots \forall c_a \forall c_b$ with c_a and c_b as defined in the observation. The relation is illustrated in Figure 4.4.

The above technique allows to efficiently map framed common intervals to common intervals. The length of the permutations and the number of filtrations are doubled, and in the beginning, we have to pre-filter the universal set N times. Thus, using the PQ tree framework, we still have a linear time complexity. Furthermore, a replacement of PQ trees by another suitable data structure, for instance PC trees for modeling circular permutations, would not interfere with this mapping.

4.4 Nested Common Intervals on Permutations

Another extension to common intervals are *nested common intervals*. In our context, the term *nestedness* describes a specific pattern of gene clusters: A cluster of size $|c|$ is called nested if it contains a nested cluster of size $|c| - 1$.

Such a recursive structure has been observed in real data already in 1996 [108], but until recently, no formal gene cluster model was defined including this property as a requirement explicitly. In some studies, gene clusters are detected by greedily extending small clusters, which implicitly presupposes a nested structure. A prominent example is the methodological definition of *über-operons* [110]. First, Hoberman and Durand [91] discussed nestedness as a desired property of gene clusters and proposed a first algorithm to identify respective clusters. Recently, Blin and Stoye [28] formally defined and studied *nested common intervals*, and gave efficient algorithms to detect them in genomes modeled both as permutations and as sequences.

Here, we first discuss the integration of nested common intervals defined on permutations into the optimization scheme.

Model 5. (Nested Common Intervals on Permutations) *Let N be an arbitrary but fixed length of the gene orders and $\mathcal{G}_N := \{1, \ldots, N\}$ the set of all genes. A* nested common interval*, defined on permutations, is either an unordered pair of genes $\{a, b\}$, or an ordered pair (c, a) of a nested common interval c and a gene a not contained in c. The gene cluster model of* nested common intervals on permutations *is defined as $(\mathcal{U}_n, \mathcal{C}_n, \sqsubseteq_n)$, where*

- *$\mathcal{U}_n$ is the set of all permutations over $\mathcal{G}_N$;*
- *$\mathcal{C}_n$ is the set of all nested common intervals over $\mathcal{G}_N$;*
- *$\{a, b\} \sqsubseteq_n g :\Leftrightarrow$ genes a and b are adjacent in genome g; and*
  $(c, a) \sqsubseteq_n g :\Leftrightarrow$ in genome g, gene a directly precedes or succeeds the genes in c and $c \sqsubseteq_n g$.

The following toy example illustrates the recursive composition of a nested common interval and how it can be identified in a permutation.

Example 11. Consider Model 5 for $N = 5$. The nested cluster $c := ((\{1, 2\}, 3), 4)$ is for instance contained in the following genomes, where the nested structure is indicated by underlining the (sub-) clusters:

$$(\underline{\underline{\underline{1,2},3},4},5), (\underline{\underline{\underline{2,1},3},4},5), (\underline{\underline{3,\underline{1,2}},4},5), (\underline{\underline{3,\underline{2,1}},4},5),$$

$$(\underline{4,\underline{\underline{1,2},3}},5), (\underline{4,\underline{\underline{2,1},3}},5), (\underline{4,\underline{3,\underline{1,2}}},5), (\underline{4,\underline{3,\underline{2,1}}},5), \ldots$$

In contrast, although in genome $g := (2, 3, 1, 4, 5)$, genes 1 to 4 are adjacent, we have:

$$\{1,2\} \not\sqsubseteq_n g \;\Rightarrow\; (\{1,2\},3) \not\sqsubseteq_n g \;\Rightarrow\; c \not\sqsubseteq_n g\,.$$

◇

Similar to framed common intervals, also nested common intervals can be mapped to basic common intervals. Indeed, in this case the transformation is simpler since both models are defined on unsigned permutations, and thus, splitting genes into head and tail is not necessary. Instead, we just model one nested common interval as a series of common intervals corresponding to all its subclusters.

Observation 3. *Let $(\mathcal{U}_n, \mathcal{C}_n, \sqsubseteq_n)$ be the gene cluster model of nested common intervals on permutations and $(\mathcal{U}_c, \mathcal{C}_c, \sqsubseteq_c)$ be the model of common intervals on permutations, each over the set of genes $\mathcal{G}_N := \{1, \ldots, N\}$. By definition, we have $\mathcal{U}_n = \mathcal{U}_c$. Further, let $(\cdots(\{a_1, a_2\}, a_3)\cdots, a_K) \in \mathcal{C}_n$ be any nested common interval and $g \in \mathcal{U}_n$ a permutation. Then, the following relation holds:*

$$(\cdots(\{a_1,a_2\},a_3)\cdots,a_K) \sqsubseteq_n g \;\Leftrightarrow\; \bigcup_{i=1}^{k} a_i \sqsubseteq_c g \quad \textit{for all } k = 2,\ldots,K\,.$$

Hence, any filtering $G \dot{\vee} (\cdots(\{a_1, a_2\}, a_3)\cdots, a_K)$ of a set of genomes G can easily be mapped to the filter operations $G \dot{\vee} \{a_1, a_2\} \dot{\vee} \cdots \dot{\vee} \{a_1, a_2, \ldots, a_K\}$. This, in turn, allows us to utilize the efficient data structures and algorithms described for common intervals in Section 4.2.

However, due to the structure of nested clusters, we have to accept an exponential problem complexity as shown next.

Observation 4. *A permutation of length N comprises $N - l + 1$ substrings of length l. Each such substring contains 2^{l-2} different nested common intervals of size l. Thus, the permutation possesses $(N - l + 1)2^{l-2}$ clusters of size l. In total, we get the following number of nested common intervals of size at least min and at most max in a permutation of length N:*

$$\begin{aligned} & \sum_{l=min}^{max} (N-l+1)2^{l-2} \\ = \;& (N-max+2)2^{max-1} - (N-min+3)2^{min-2} \\ \textit{i.e.} \;& 2^N - (N+1) \quad \textit{for } min = 2 \textit{ and } max = N\,. \end{aligned}$$

Surely, for the process of labeling the tree, we do not have to keep all clusters of all permutations. We can rather use a modified preprocessing as described next.

4.4.1 Modified Preprocessing

This approach drastically decreases the cost for the pre-processing and minimizes the number of nested clusters under consideration. Nevertheless, this number generally remains exponential since in the worst case – and also in practice – parts of the gene orders are perfectly conserved among several genomes, which yield an exponential number of reconstructed clusters each. Even if we would handle such a substructure explicitly as one object rather than many nested common intervals, we would still have to reconsider all contained clusters separately during the optimization phase.

We first scan each permutation in a sliding window approach to detect all nested clusters of size two. For each such pair of genes, we check for occurrences of the same pair in all other permutations. This search can be speeded up by generating the inverse permutations beforehand. Then, we compute a most parsimonious labeling for this pair. If it is not reconstructed for any inner node, neither this pair nor any larger cluster containing this pair has to be considered in the remainder of the reconstruction method.

After processing all pairs, we iteratively enlarge each cluster found so far in both directions and proceed analogously. Using this bottom-up strategy, we omit unnecessary processing of futile candidates. We end up with a most parsimoniously labeled tree and continue as described in Algorithm 4 on page 47.

A cluster has to be considered only if it is contained in the most parsimonious labeling of any inner node. But in fact, there may be various such labelings. For instance, compared to the sparse variant, applying the dense variant of the Fitch-Hartigan algorithm tends to yield much more candidates and fewer unnecessary clusters can be disregarded. In both cases, we only take a certain set of clusters into account. Thus, the parsimony weights for the computed most parsimonious and optimal labelings (W_{par}^{sparse}, W_{par}^{dense}, W_{opt}^{sparse} and W_{opt}^{dense}) underestimate the total parsimony weights (W_{par} or W_{opt}, respectively) by a certain fixed value (δ^{sparse} or δ^{dense}, respectively). This does not influence the individual minimizations. But nevertheless, this value varies among the different most parsimonious labelings:

$$W_{opt} = W_{opt}^{sparse} + \delta^{sparse} \tag{4.1}$$

$$W_{par} = W_{par}^{sparse} + \delta^{sparse} \tag{4.2}$$

$$W_{opt} = W_{opt}^{dense} + \delta^{dense} \tag{4.3}$$

$$W_{par} = W_{par}^{dense} + \delta^{dense} \tag{4.4}$$

Hence, we have to factor in these neglected costs when we use the weight W_{opt}^{sparse} of the optimal labeling computed in the sparse phase as a bound for the computation for a dense optimum, as done in the two-phase algorithm (Algorithm 5 on page 48). However, in any case, the weights of a most parsimonious and an optimal labeling differ by a certain amount Δ:

$$W_{opt} = W_{par} + \Delta\,.$$

Replacing W_{opt} using Equations 4.1 and 4.3, and W_{par} using Equations 4.2 and 4.4, we get the following two terms for Δ, which can be equated.

$$\underbrace{W_{opt}^{sparse} + \delta^{sparse}}_{(4.1)} = \underbrace{W_{par}^{sparse} + \delta^{sparse}}_{(4.2)} + \Delta$$
$$\Leftrightarrow \Delta = W_{opt}^{sparse} - W_{par}^{sparse}$$

$$\underbrace{W_{opt}^{dense} + \delta^{dense}}_{(4.3)} = \underbrace{W_{par}^{dense} + \delta^{dense}}_{(4.4)} + \Delta$$
$$\Leftrightarrow \Delta = W_{opt}^{dense} - W_{par}^{dense}$$

$$\Rightarrow W_{opt}^{sparse} - W_{par}^{sparse} = W_{opt}^{dense} - W_{par}^{dense}$$

Hence, instead of keeping track of the different neglected costs, we can simply refine the bound for a dense optimum based on the weights of labelings which we compute during the two-phase approach:

$$W_{opt}^{dense} = W_{par}^{dense} + W_{opt}^{sparse} - W_{par}^{sparse} .$$

As already mentioned, in the model of nested common intervals, we cannot avoid to consider an exponential number of clusters. However, as we will see in the following section, practical running times can be achieved by the modified preprocessing as well as bounding the maximal size of clusters.

4.5 Evaluation on Simulated Data

Since the output of our algorithm consists of gene clusters of extinct species, a direct verification on biological data is hardly possible. Therefore, we use simulated data to evaluate the effectiveness of our method.

4.5.1 Simulation Setup

For this evaluation, we simulated gene order data using the program Rose-GEvolutionS, available from the web site `http://bibiserv.techfak.uni-bielefeld.de/rose/`. We constructed a balanced, binary tree with eight leaves as shown in Figure 4.5 and labeled its root with a signed permutation of length $N = 100$. In a top-down fashion, we performed several different rearrangement operations along the edges. To compare different scenarios, we varied the number of operations per edge and the type of operations:

Inversion. Two cutting points are chosen randomly along the gene order. Then, the order and orientation of the genes in the segment between these points is reversed.

Transposition. First, two cutting points are chosen randomly along the gene order. Then, the segment between these points is cut out and inserted either in forward or backward direction at a third point which is chosen randomly along the two remaining parts of the gene order.

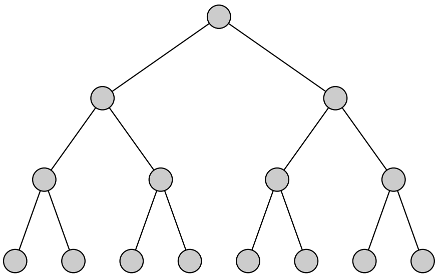


Figure 4.5: Illustration of the topology of the balanced, binary tree with eight leaves as used in the evaluation.

Then, we used the rearranged gene orders at the leaf nodes and the same tree topology for the reconstruction. We have implemented all gene cluster models introduced in the Sections 4.1 to 4.4. After the computation of an optimum, for each reconstructed cluster at each node, we checked its existence in the simulated (true) gene order to count the number of true positives (TP) and false positives (FP). We counted the number of clusters that were neither present in the simulated order, nor reconstructed by the algorithm as true negatives (TN) and the rest as false negatives (FN). Then, we calculated the specificity, the sensitivity and the precision (also known as positive predictive value):

$$\text{Specificity} = \frac{\mathrm{TN}}{\mathrm{FP}+\mathrm{TN}}$$

$$\text{Sensitivity} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

$$\text{Precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

Independent of the actual setup, the number of simulated and reconstructed clusters was very small compared to the overall number of considered clusters. This entailed a very high number of true negatives compared to the number of false positives. Therefore, the specificity was generally close to one. Since this does not provide any valuable information, we omitted the commonly used ROC curve, which includes the sensitivity and the specificity. Instead, we studied the sensitivity and the precision to answer the questions “How many of the ancestral clusters does our method reconstruct?” and “How many of the reconstructed clusters are correct?”.

To obtain meaningful correlations and trends, we averaged the evaluated values over several runs for each parameter setting. We observed fast convergence and have thus set the number of runs to ten. Since the overall evaluation comprised several simulations for many different parameters and models, we chose the small genome length of 100. Table 4.1 on the next page shows stable sensitivity and precision for increasing genome length when the ratio of genome length and evolutionary rate is held constant.

Table 4.1: The results of the sparse variant of the reconstruction algorithm for increasing genome length and evolutionary rate using unsigned adjacencies.

Genome Length	Inversions per Edge	Precision	Sensitivity
100	20	0.99	0.36
250	50	1.00	0.34
500	100	1.00	0.34
1000	200	1.00	0.34
2000	400	1.00	0.34
4000	800	1.00	0.35

Note that some of the results that will be shown later on appear to be poor for high evolutionary rates. After all, a rate of 25 per edge means that two simulated permutations are separated by 50 to 150 rearrangement operations, which for a genome length of 100 rather corresponds to a completely random data set. Hence, the evolutionary rates chosen in this study induce gene orders ranging from nearly perfect data to unrealistically excessively shuffled gene orders where realistic cases may lie somewhere in between.

4.5.2 Results

Figure 4.6 shows the overall results of our method for the gene cluster model of unsigned adjacencies. In Figure 4.6(a), we see a high precision of nearly one which decreases very slowly for an increasing evolutionary rate. Since the precision remained practically constant

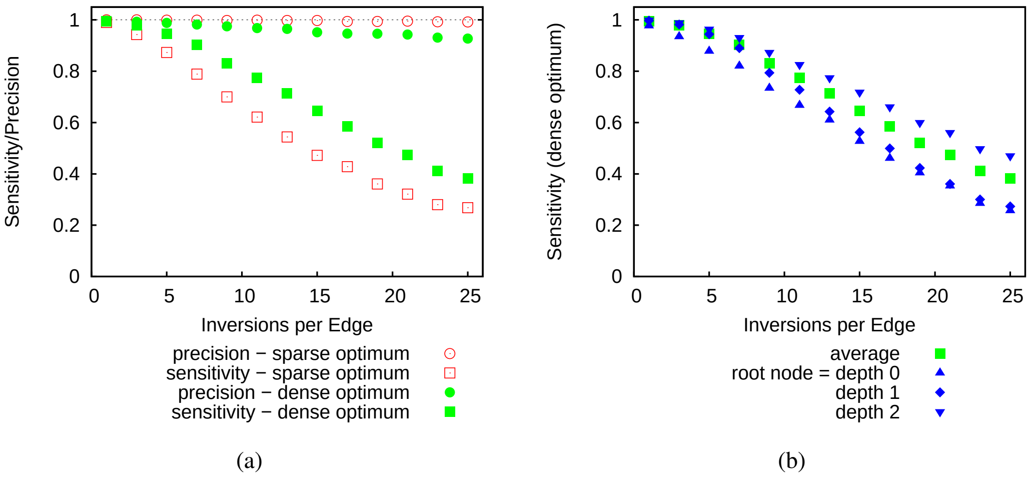
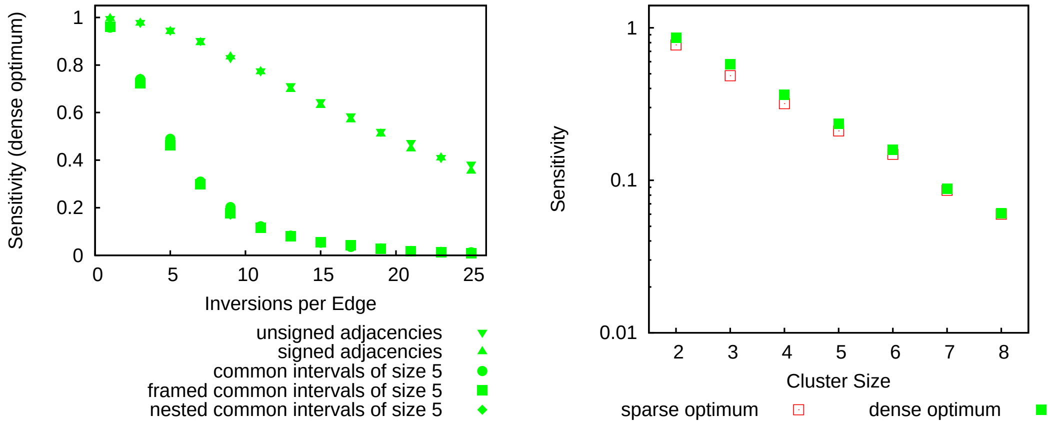


Figure 4.6: The precision and sensitivity of the two-phase approach (Algorithm 5 on page 48) using the gene cluster model of unsigned adjacencies. (a) Results for sparse and dense optima for different evolutionary rates. (b) Results for different levels in the tree.

throughout the whole evaluation, we omit reporting it in all following investigations. In contrast, the sensitivity is lower and declining faster. In comparison to the sparse optimum, the precision of the dense optimum is hardly lower, but the sensitivity is significantly higher. Thus, even though reconstructing more clusters comes at the cost of their correctness, it pays off in total and the dense optimum should be preferred in practice.

Figure 4.6(b) displays the following, unsurprising relation: The closer a node is to the root of the tree, the greater is the distance to the given information at the leaves, and the greater is the inaccuracy.

We got similar results for other gene cluster models. For example, Figure 4.7(a) shows concordant results for unsigned and signed adjacencies. Hence, including orientation does not show any significant effect. The sensitivity for common intervals and their variants strongly depends on their size. For instance, also shown in Figure 4.7(a), only few clusters containing five genes are found. Figure 4.7(b) shows an exponential drop of the sensitivity with respect to the size of the clusters.



(a)

(b)

Figure 4.7: Comparison of the sensitivity of the two-phase approach (Algorithm 5 on page 48) for different gene cluster models. (a) For different evolutionary rates, signed and unsigned adjacencies are contrasted, as well as common intervals and its variants where only clusters comprising five genes have been considered. (b) The dependency of the sensitivity on the cluster size is shown on a logarithmic scale. For each size s, only common intervals containing exactly s genes have been considered for the evaluation. Here, a fixed evolutionary rate of ten inversions per edge has been used.

We performed simulations with different selections of rearrangement operations. This evaluation revealed a decline of the sensitivity with increasing amount of transpositions. However, this effect can simply be explained by the number of broken adjacencies per rearrangement. In particular, an inversion cuts and rejoins two adjacencies, whereas a transposition alters three. Although, in a sequence of operations, an adjacency broken by one rearrangement can be rejoined coincidentally by another, we neglect this effect for the simple normalized

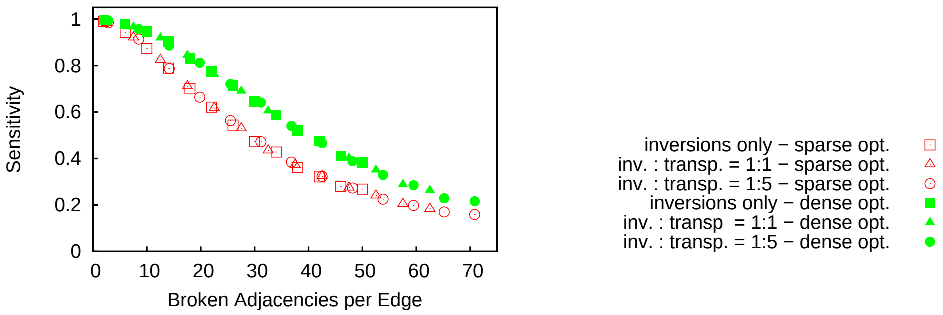


Figure 4.8: Comparison of the sensitivity of the two-phase approach (Algorithm 5 on page 48) for unsigned adjacencies based on different evolutionary scenarios. During the simulation, either inversions or transpositions were chosen randomly according to a given ratio. The sensitivity has been estimated for different evolutionary rates and is plotted with respect to the number of adjacencies broken by one rearrangement operation. An inversion breaks two adjacencies whereas a transposition breaks three. Thus, for instance, one operation chosen according to a ratio of 1 inversion to 5 transpositions breaks $(2 + 5 \cdot 3)/6$ adjacencies on average.

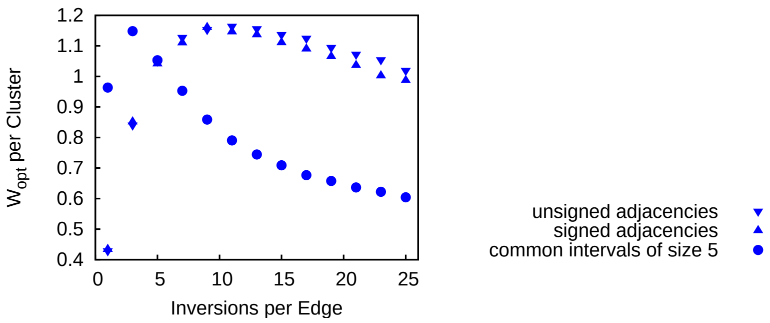


Figure 4.9: Weight of optimal labelings for different cluster models with respect to varying evolutionary rates. The average parsimony weight is normalized by the number of clusters under consideration. Thus, the average number of gains and losses per cluster is depicted.

visualization of the results in Figure 4.8. A comparison with respect to the number of affected adjacencies rather than the number of operations attests unique results for the different evolutionary scenarios.

Independent of whether an actual labeling is computed by the sparse or dense phase, the parsimony weight of a computed optimum is unique and exact as defined by the problem statement (Definition 15 on page 23). Figure 4.9 compares this value for different gene cluster models and evolutionary rates. For very few rearrangements, we get very few gains and losses of clusters and thus a low weight. Analogously, we see a quickly increasing weight for an increasing number of inversions. But for higher evolutionary rates, the reconstructed labelings veer away from the simulated, true history. Since a real scenario has to be consistent but not parsimonious, a reconstructed, optimal labeling can have a lower weight in general. This discrepancy grows for increasing inversion rate. The characteristics of the curve vary for different cluster models. The more restrictive the cluster model the stronger are both effects.

There is hardly any difference between the optimal and the most parsimonious weight. For instance, in the evaluation for unsigned adjacencies for inversion rates of $1, 3, 5, \ldots, 25$ with ten runs each, we observed only three times a difference of W_{opt} and W_{par}, never larger than one. Hence, in most cases, a most parsimonious labeling computed by the sparse approach is consistent and thus optimal. To get an optimum based on a dense, most parsimonious labeling, some conflicting clusters have to be deleted from the labeling to reach consistency. Figure 4.10 shows that the number of deletions is low and follows the same trend as the optimal weight: The more conflicting the data, the higher the weight and the more deletions are necessary.

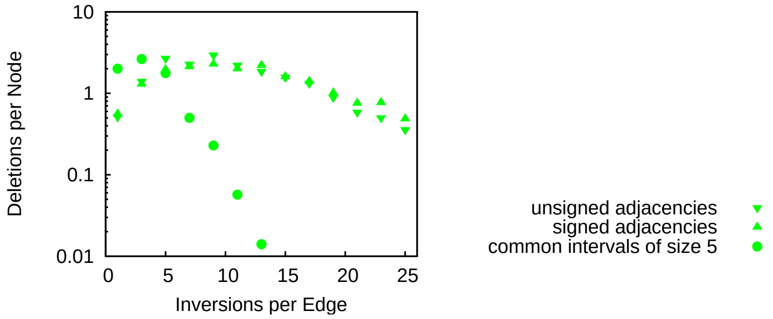


Figure 4.10: Number of clusters which have been assigned to a node by the dense variant of the Fitch-Hartigan method (Algorithm 1 on page 33) but deleted to reach an optimal labeling. The average number of deletions per node is shown for different cluster models and evolutionary rates on a logarithmic scale.

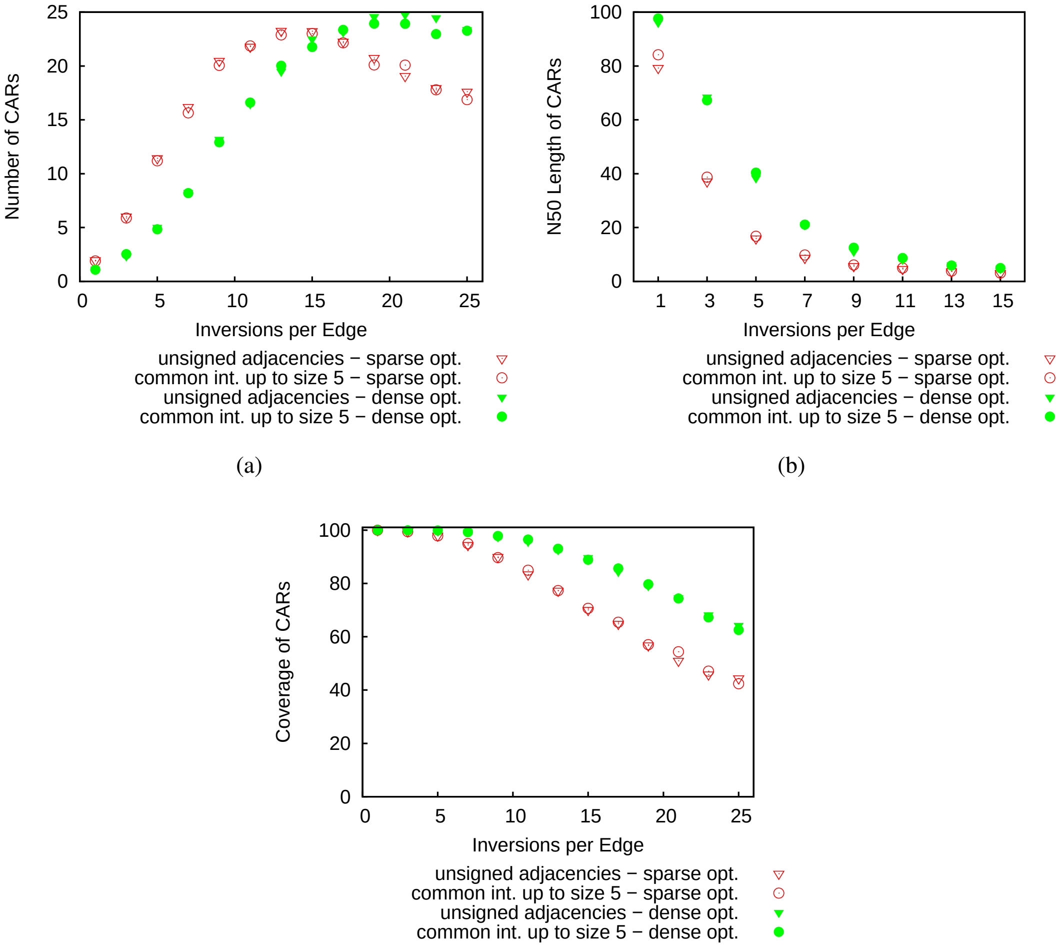
We observed that the largest proportion of reconstructed clusters belongs to segments of the gene order which are fully conserved. That means, in most cases, groups of clusters assigned to a node overlap such that the putative ancestral order of the corresponding genes is determined exactly – without any degree of freedom. In line with other studies [44, 119], we say that overlapping gene clusters belong to one *contiguous ancestral region (CAR)*. If the order of the genes of a CAR is fully resolved, we call it *fixed* and *flexible* otherwise.

Example 12. (Contiguous Ancestral Region (CAR)) Assume that, based on the gene cluster model for common intervals on permutations (Model 3), the following clusters have been assigned to a node:

$$c_1 = \{1, 2\},\ c_2 = \{2, 3, 4\},\ c_3 = \{3, 4, 5, 6\},\ c_4 = \{4, 5\},\ c_5 = \{7, 8, 9\},\ c_6 = \{8, 9, 10\}\,.$$

Then, c_1, c_2, c_3 and c_4 are overlapping and belong to one CAR. This CAR is fixed since the order of the genes $\{1, \ldots, 6\}$ is fully resolved to $1, 2, 3, 4, 5, 6$ (or its reverse $6, 5, 4, 3, 2, 1$). The remaining common intervals c_5 and c_6 also overlap and build a second CAR which is flexible, because the order of the genes 8 and 9 is not determined. Both $7, 8, 9, 10$ and $7, 9, 8, 10$ (and the reverse orders) are valid gene orders. ◇

In practice, a CAR consists of very many clusters. Instead of researching into the evolutionary history of single clusters or examining the functional or evolutionary relation of genes cluster by cluster, we are rather interested in this information condensed in CARs. In Figure 4.11, the reconstruction of CARs is discussed in terms of number, length and coverage. At a low rearrangement rate, nearly all genes are contained in very few, large CARs. With increasing rate, fewer clusters are found, which yields two effects. On the one hand, instead



(a)

(b)

(c)

Figure 4.11: Evaluation of the results of the two-phase approach (Algorithm 5 on page 48) in terms of CARs. In each figure, the results for unsigned adjacencies and common intervals with a size of maximum five are displayed for different evolutionary rates. (a) The average number of CARs per node. (b) The average *N50 length* of the CARs. The N50 length of a set of CARs represents the largest length l such that at least half of the total size of the CARs is contained in CARs larger than l. For instance, the N50 length of CARs of the lengths $\{2, 2, 4, 5, 6, 8\}$ is 5. (c) The average number of genes which are contained in any CAR. For instance, a coverage of 100 means that all genes are covered by CARs.

of one big CAR, several small CARs are formed. But on the other hand, some CARs are not conserved at all. Thus, we get a convex behavior with a maximum depending on the cluster model and the chosen approach: Common intervals form fewer, larger CARs than adjacencies, and a dense labeling contains fewer, larger CARs as well.

Different gene cluster models allow different structures of CARs. Obviously, overlapping adjacencies always build fixed CARs. Hence, broader cluster models have to be used to identify ambiguous but still conserved genomic structures and to reconstruct flexible CARs. However, in our evaluation, we only observed very few flexible CARs. For instance, for common intervals comprising up to 25 genes and an inversion rate of ten, only 3.76% of all CARs were flexible.

Thus, the reconstructed CARs differ only very little among the different studied cluster models. All flexible CARs found by framed or nested common intervals can particularly be found by (basic) common intervals. And due to the high precision and moderate sensitivity, a restriction of the flexible CARs by a nested or framed structure does not seem to be expedient.

We studied the running time of our reconstruction method for various simulation settings and gene cluster models. As expected, in general, the computation of a sparse optimum in the first phase was many times faster than the subsequent second phase, the computation of a dense optimum. This can be explained by the differing numbers of gene clusters and conflicts to be handled by the two phases. Figure 4.12 reveals that the running time behavior roughly corresponds to the above discussed trends of the optimal weight and the number of deletions.

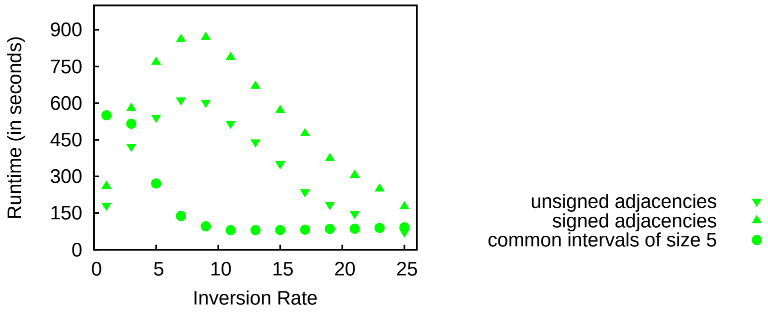


Figure 4.12: The running time of the two-phase method (Algorithm 5 on page 48) for different cluster models with respect to varying evolutionary rates. In rare cases, very complex conflicting structures imply extremely long computations. To avoid an obfuscation of the general trend by such outliers, we show the median of 100 runs. The simulations were performed on a Sparc machine with 32 GB main memory using up to four 2600 MHz processors in parallel to estimate the conflict index for the labelings of the different nodes of the tree.

The findings of this study can be summarized as follows. We observe a high precision in general and a moderate sensitivity for diverse gene orders and higher levels of the tree. In most cases, only few deletions are necessary to reach an optimum from a most parsimonious labeling, which emphasizes the benefit of the two phase approach. Many results are independent of the type of rearrangement operation and gene cluster model. To decrease computation time, one should refrain from considering clusters of arbitrary size. But to increase sensitivity, small clusters should not be excluded. Only few reconstructed CARs are flexible, which makes restricted variants of common intervals redundant.

Finally, we want to point out important limitations of this evaluation. Firstly, in our simulation of gene order evolution, we solely modeled *neutral evolution*: we simulated mutations and did not take other processes into account. But in fact, one basic idea of gene cluster studies is that, in spite of rearrangement, *selective pressure* yields and preserves co-location of specific genes. Another explanation of the origin of gene clusters is *horizontal gene transfer*, which was not included in the simulation either.

Secondly, we defined the sensitivity and precision based on whether a certain cluster is contained in a gene order in the mathematical sense rather than in terms of biologically meaningful clusters. For instance, some of the theoretically correctly reconstructed gene clusters may be undesirable in practice, because they are just conserved by chance and do not provide any information.

However, designing appropriate models and methods for the simulation and evaluation is a hard task, which would go beyond the scope of this thesis. Nevertheless, some of the above observations and conclusions will be confirmed on real data in Chapter 6.

Chapter 5

Sequence-based Gene Cluster Models

In the previous chapter, we discussed several gene cluster models defined on permutations. However, as mentioned in the Introduction, it is not trivial to obtain such mathematically simple representations from real genomic data. In general, it is more appropriate to model gene order by *sequences* of gene identifiers in order to allow individual genes to occur multiple times in one genome or not at all.

We now reconsider the labeling problem and the aforementioned, permutation-based cluster models in a sequence-based context. For each gene cluster model, we define a corresponding variant on sequences and address the problem of how to verify consistency.

The graph theoretic approach to represent adjacencies on permutations as presented in Section 4.1 is extended to sequences and provides a simple and efficient solution. In contrast, we present NP-completeness results for the gene cluster model of common intervals on sequences, including its variants restricted to framed and nested common intervals.

This chapter is organized in the following way. First, we generally discuss and refine the term *consistency* in the context of sequences in Section 5.1. In the remainder of the chapter, we review the gene cluster models. In Section 5.2, we introduce an efficient data structure for adjacencies. Then, we present NP-completeness results for common intervals in Section 5.3 and for its variants in Section 5.4.

5.1 Consistency on Sequences

Before we define gene cluster models on sequences in the following sections, we firstly reconsider the general aim of our reconstruction framework: the optimization criterion and, in particular, the concept of consistency.

So far, we modeled genomes as permutations, which means that any considered genome contains each gene exactly once. We now lift this restriction and represent genomes as *sequences* of genes. Generally, in a sequence, any element can occur multiple times or not at all, which in the context of gene order comparison corresponds to paralogous gene families and gene deletions, respectively.

Recall the definition of optimality: Among all consistent labelings, those with minimal parsimony weight are specified as optimal. If we now allow each gene to appear arbitrarily often in any genome, this would induce the following effect. For a given problem instance, consider a most parsimonious labeling. This labeling is consistent since, for each node, there is a valid gene order containing all assigned clusters. For instance, we can simply create a short sequence of genes according to each cluster separately and then concatenate these sequences to an absurd yet valid gene order. Such a construction is possible for any gene cluster model. As a consequence, consistency always holds and does not contribute to a specification of reasonable labelings.

Even if we replace the naive concatenation approach and instead construct preferably compact valid gene orders, we cannot avoid to include some genes multiple times. In some cases, this causes side effects. An example is given in Figure 5.1. Although a gene is contained in all input genomes only once, according to the original optimization criterion, it is reconstructed to occur multiple times for ancestral nodes. In the example, we consider a subsequence of only three genes in each input genome and obtain a segment of five genes for the examined internal node. In general, this artifact implies longer and longer genomes for higher levels in the tree.

5.1.1 Restricting the Multiplicity of Genes

To preclude the aforementioned effect, we refine the concept of consistency. Instead of simply restricting the total length of a genome, we limit the multiplicity of each individual gene. To this end, we define a restricted variant of a genome set.

Definition 22. (Restricted Genome Set) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, where $\mathcal{U}$ is the set of all sequences over the set of genes $\mathcal{G}_N := \{1, \ldots, N\}$. Further, let $C \subseteq \mathcal{C}$ be a set of gene clusters and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene.*

The genome set of C restricted by m, *denoted* $\mathrm{GS}_m(C)$, *is the subset of all genomes $g \in \mathcal{U}$ for which the following properties hold:*

(i) g contains each gene a at most $m(a)$ times, and

(ii) $c \sqsubseteq g$ for all $c \in C$.

If we consider signed sequences, we do not account for the orientation when counting the occurrences of a gene.

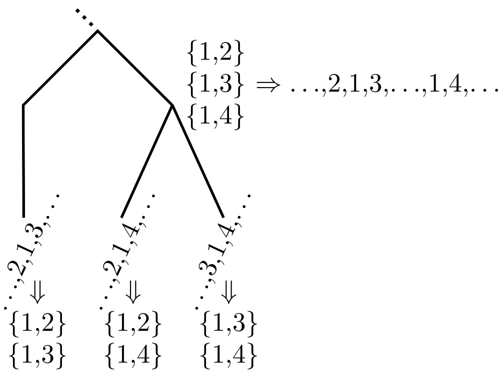


Figure 5.1: Example for an artifact that arises when gene clusters are reconstructed on the basis of gene orders, where any gene can occur arbitrarily often in any genome. A small subtree over three genomes is shown exemplarily. The gene orders on the leaves imply, beside others, the listed adjacencies. Any most parsimonious labeling would assign all three adjacencies to the rightmost internal node. Since genes are allowed to appear multiple times in a genome, valid gene orders exist, all of which contain gene 1 at least two times – for instance the given gene order.

The notions *conflicting set of gene clusters* and *filtered set of genomes* are restricted analogously. We define a multiplicity threshold m_v for each node v in the tree separately. Then, a labeling λ is *consistent* if $\lambda(v)$ does not contain any conflicts with respect to m_v for any node v. Apart from that, the labeling problem remains unchanged as stated in Definition 15 on page 23. In the following, we will omit the index v if a specification of the node is not necessary.

Once we develop a method to filter with respect to a given m for a specific gene cluster model, we can utilize our algorithmic framework presented in Chapter 3 to solve the labeling problem. More precisely, it is sufficient to find a method to decide for a given set of gene clusters C and a specific multiplicity threshold m whether $\mathrm{GS}_m(C)$ is empty or not. None of the presented algorithms is actually affected by this restriction. We only have to reformulate our method to identify minimal conflicting subsets. A respective, slightly modified version of Algorithm 2 is laid out in Algorithm 6 on the next page.

Since we identified testing for $\mathrm{GS}_m(C) = \emptyset$ as an essential question, we state the problem explicitly while phrasing it independently of the term *genome set*.

Definition 23. (Consistency Problem) *Let $(\mathcal{U}, \mathcal{C}, \sqsubseteq)$ be a gene cluster model, where $\mathcal{U}$ is the set of all sequences over the set of genes $\mathcal{G}_N := \{1, \ldots, N\}$. Further, let $C \subseteq \mathcal{C}$ be a set of gene clusters and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene.*

The consistency problem *is to decide if C is* consistent *with respect to m, i.e. whether there exists a gene order $g \in \mathcal{U}$ for which the following properties hold:*

(i) g contains each gene a at most $m(a)$ times, and

(ii) $c \sqsubseteq g$ for all $c \in C$.

```
Algorithm 6: $\mathrm{SearchConf}'(C, m) \longrightarrow \mathrm{Conf}(C)$   Restricted Search for Confl. Subsets
  Input: A set of gene clusters $C = \{c_1, c_2, \dots, c_{|C|}\} \subseteq \mathcal{C}$, and
         a multiplicity threshold $m : \mathcal{G}_N \longrightarrow \mathbb{N}$.
  Output: $\mathrm{Conf}(C)$, the set of minimal conflicting subsets of $C$.
  $\mathrm{ConfList} \longleftarrow$ empty list
  $\mathrm{SearchConfRec}'(C, m, \mathrm{ConfList})$
  return $\mathrm{ConfList}$

  Procedure: $\mathrm{SearchConfRec}'(C, m, \mathrm{ConfList})$
  Input: A set of gene clusters $C = \{c_1, c_2, \dots, c_{|C|}\} \subseteq \mathcal{C}$, $m : \mathcal{G}_N \longrightarrow \mathbb{N}$, and
         $\mathrm{ConfList}$, a list of solutions found so far.
  $M \longleftarrow \emptyset$
  $i \longleftarrow 1$
  while $\mathrm{GS}_m(M) \neq \emptyset$ and $i \leq |C|$ do
      $M \longleftarrow M \cup \{c_i\}$
      $i \longleftarrow i + 1$
  if $\mathrm{GS}_m(M) = \emptyset$ then
      foreach $c \in M$ do
          if $\mathrm{GS}_m(M \backslash \{c\}) = \emptyset$ then $M \longleftarrow M \backslash \{c\}$
      if $M \notin \mathrm{ConfList}$ then append $M$ to $\mathrm{ConfList}$
      /* recursion */
      foreach $c \in M$ do $\mathrm{SearchConfRec}'(C \backslash \{c\}, m, \mathrm{ConfList})$
```

Whenever we want to incorporate a sequence-based gene cluster model into our framework, we have to provide a solution for the consistency problem. As we will see in the following sections, the problem complexity highly depends on the specific cluster definition.

After all, we have to specify the multiplicities m_v for each node v in the phylogenetic tree. The threshold m_l for a leaf l is obviously given by the input: We simply count the occurrences of each gene g in the gene order assigned to l to define $m_l(g)$. However, we have to determine m_v for the internal nodes.

The most accurate but also an elaborate approach is to estimate the putative copy number of each gene in each ancestral gene order by *lineage sorting* [120, 129]. Here, the individual evolutionary history of each gene family, the so-called *gene tree*, is determined separately, for instance based on genomic sequence data. The obtained trees generally differ and are either reconciled with a given phylogeny of the species, the so-called *species tree*, or are used to determine a proper species tree. Such a scenario explains the diverging history of the genes by speciation events, gene duplications and gene losses, and therefore tells us how many copies of a gene g an ancestral genome at node v presumably contained. It thus defines $m_v(g)$.

For some specific datasets, we can rely on knowledge about the genomic history. For instance, several studies suggest two whole genome duplications in the evolution of the Chordate genome in the teleost fishes lineage [47, 67, 98, 130]. Such information can be used to deduce the ancestral number of genes.

One could also utilize approaches which do not require any further data or pre-knowledge. As mentioned above, the input gene orders define the copy numbers of genes for all leaves of the given tree. Based on these, we can estimate the multiplicities of genes for the internal nodes. For example, roughly following our general objective, we could apply parsimony, where the copy number of each gene corresponds to a character and we minimize the amount of copy number differences. That means, we simply define the cost of the multiplicity of gene g changing on edge (u, v) from $m_u(g)$ to $m_v(g)$ as $m_v(g) - m_u(g)$.

A less restrictive solution is to define the multiplicity of a gene g for node u in a bottom-up fashion as the maximum over the multiplicities of its child nodes $v_1, \ldots, v_k$:

$$m_u(g) := \max_{i=1,\ldots,k} \left(m_{v_i}(g)\right) .$$

Instead of fixing the thresholds in advance, one could also try to include the gene multiplicity into the overall objective of the reconstruction. However, this is far from trivial since the multiplicity threshold is directly linked with both principles: *parsimony* and *consistency*. Optimizing for these two criteria for fixed multiplicities is already an intricate task. But including a third criterion would increase the complexity tremendously.

5.1.2 Sequences without Duplications

There is another prevalent model of gene order which lies between permutations and sequences, known as *sequences without duplications*. It is more relaxed than permutations since it allows for varying gene content among the considered genomes: Not all genes have to occur in all genomes. But in contrast to general sequences, it allows each gene to occur *at most* once.

This corresponds to the aforementioned model where we set $m_v(g) = 1$ for all genes g and all nodes v. However, instead of realizing this model in the sequence-based framework, we revert to the permutation-based methods. In most cases, we can still use the data structures presented in Chapter 4 covering all genes. If a gene is not present in some genomes and no clusters containing this gene are reconstructed for a specific internal node, the corresponding object in the data structure is not affected and does not influence the consistency of the labeling.

The only exception is the use of PC trees to model circular gene orders. Here, a cycle comprising only a subset of all genes is precluded. However, instead of using a PC tree over the whole set of genes, we can simply base the filtering operations on the universal PC tree over only those genes contained in any of the gene clusters we filter with.

In conclusion, even if there is no efficient solution for the consistency problem for a specific gene cluster model on sequences, this is often the case for sequences without duplications.

5.2 Adjacencies on Sequences

In this section, we reconsider the gene cluster model of adjacencies. Due to their simple combinatorial structure, adjacencies can easily be defined on sequences and the detection of adjacencies which are conserved among a set of genomes is a straightforward task. For that reason, this model has been widely used in biological studies as already discussed in the beginning of Section 4.1.

We now define a sequence-based variant of adjacencies as a generalization of Model 1 introduced on page 52. Since now, a gene may occur multiple times in a genome, we also take adjacencies of a gene to itself into account. If two genes a and b are adjacent in a genome g, we say that the corresponding adjacency $\{a, b\}$ is contained in g, independently of the location of the other copies of a and b in g.

Model 6. (Unsigned Adjacencies on Sequences) *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of all genes. Then, the gene cluster model of* unsigned adjacencies on sequences *is defined as $(\mathcal{U}^s_u, \mathcal{C}^s_u, \sqsubseteq^s_u)$, where*

- *$\mathcal{U}^s_u$ is the set of all sequences over $\mathcal{G}_N$;*
- *$\mathcal{C}^s_u$ is the set of all pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N$; and*
- *$\{a, b\} \sqsubseteq^s_u g :\Leftrightarrow a$ and b occur adjacently at least once in g.*

The corresponding model of signed adjacencies is refined analogously.

Model 7. (Signed Adjacencies on Signed Sequences) *Again, let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of all genes. Then, the gene cluster model of* signed adjacencies on signed sequences *is defined as $(\mathcal{U}^s_s, \mathcal{C}^s_s, \sqsubseteq^s_s)$, where*

- *$\mathcal{U}^s_s$ is the set of all signed sequences over $\mathcal{G}_N$;*
- *$\mathcal{C}^s_s$ is the set of all pairs $\{a, b\}$ with $a, b \in \{i, -i \mid i \in \mathcal{G}_N\}$; and*
- *$\{a, b\} \sqsubseteq^s_s g :\Leftrightarrow a$ is directly followed by $-b$, or b by $-a$ at least once in g.*

Note that, strictly speaking, the set of all (signed) sequences is not finite since the lengths of the sequences can be infinite. But all sequences have to meet the multiplicity threshold and are thus bounded in length implicitely.

To include these models into our algorithmical framework for solving the labeling problem, it is necessary and sufficient to provide according solutions for the consistency problem: We have given a set of genes $\mathcal{G}_N$, a multiplicity threshold m and a set C of unsigned adjacencies (or signed adjacencies, respectively). We have to decide if there exists a (signed) sequence g over $\mathcal{G}_N$ such that g contains each gene a at most $m(a)$ times and g contains all adjacencies in C. Or, in short: Is C consistent with respect to m?

In the following section, we present an efficient solution for this problem.

5.2.1 The Gene Order Graph

To model the consistency problem for adjacencies on sequences, we use a graph theoretic approach. The data structure is the same as we already defined on page 53 for step-wise filtering with adjacencies on permutations. As a formal basis for the following theoretical considerations, we now define this graph properly with respect to our current needs. We begin with the unsigned case.

Definition 24. (Gene Order Graph) *Let $\mathcal{G}_N = \{1, \dots, N\}$ be a set of genes and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N$.*

Then, the gene order graph *of C, denoted by $G_N(C)$, is the graph with the vertex set $\{v_i \mid i \in \mathcal{G}_N\}$ and the edge set $\big\{\{v_a, v_b\} \mid \{a, b\} \in C\big\}$.*

The gene order graph of a set of adjacencies C can be constructed in $\mathcal{O}\left(N + |C|\right)$ time and space. In this process, we keep track of the degree of each node v_i, denoted by $\deg(v_i)$. Then, the following lemma allows us to check for consistency of C in further $\mathcal{O}\left(N\right)$ steps and thus in a total running time and with a space requirement of $\mathcal{O}\left(N + |C|\right)$.

Lemma 2. *Let $\mathcal{G}_N = \{1, \dots, N\}$ be a set of genes and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \mathcal{G}_N$ and $G_N(C) = (V, E)$ be the gene order graph of C.*

Then, C is consistent with respect to m if and only if the following conditions hold:

(i) $\deg(v_i) \leq 2m(i)$ *for all vertices $v_i \in V$, and*

(ii) $\sum\limits_{v_i \in c} \big(2m(i) - \deg(v_i)\big) > 0$ *for all connected components c in $G_N(C)$.*

The underlying idea which we will use to prove the above lemma, is based on the classical concept of *Eulerian graphs.*

Definition 25. (Eulerian Graph) *A graph is* Eulerian *if it contains an* Eulerian cycle*, i.e. a path starting and ending in the same vertex and traversing each edge exactly once.*

Theorem 4. ([68]) *A graph is Eulerian if and only if it is connected and all vertices have even degree.*

Theorem 4 also holds for *multigraphs*, where two nodes can be connected by multiple edges. That means, in a multigraph, we collect all edges in a multiset, i.e. we assign a multiplicity to each edge. Any multigraph H can easily be reduced to a simple graph G by splitting multiply occurring edges into two edges connected by a unique node each. For instance, we replace an edge $\{v_i, v_j\}$ in H by the edges $\{v_i, v_{ij}^1\}$ and $\{v_{ij}^1, v_j\}$, and $\{v_i, v_{ij}^2\}$ and $\{v_{ij}^2, v_j\}$ etc. Since any path in G uniquely corresponds to a path in H, G is Eulerian if and only if H is Eulerian. Furthermore, this procedure does not influence the connectivity of the graph and all newly added vertices have a degree of two such that the conditions of the theorem are not affected.

Before we formally prove Lemma 2, we sketch the connection between the consistency problem and an Eulerian graph: For a given set of adjacencies, we build its gene order graph.

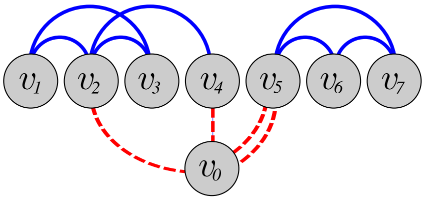


Figure 5.2: An example to illustrate the proof of Lemma 2. Consider the set of genes $\mathcal{G}_7$ with the multiplicities $m(g) = 2$ for $g \in \{2, 5\}$ and $m(g) = 1$, otherwise, and the set $C = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{5, 6\}, \{5, 7\}, \{6, 7\}\}$ of unsigned adjacencies. The gene order graph $G_7(C)$ is depicted including the extensions described in the proof. The solid edges correspond to the original edges as defined by the given adjacencies, and the dashed lines represent the auxiliary edges. The obtained extended graph contains, for instance, the Eulerian cycle $(v_0, v_2, v_1, v_3, v_2, v_4, v_0, v_5, v_6, v_7, v_5, v_0)$, which corresponds to the valid gene order $(2, 1, 3, 2, 4, 5, 6, 7, 5)$.

Then, we extend the graph in a certain way. If the obtained graph is Eulerian, a respective cycle corresponds to a valid gene order. An example is given in Figure 5.2 to illustrate this relation.

Proof. (Proof of Lemma 2) Assume we have given $\mathcal{G}_N$, m, C and $G_N(C)$ as required by the lemma. We extend the gene order graph $G_N = (V, E)$ to a multigraph $H_N = (V', E')$, where the new vertex set contains one additional node v_0, i.e. $V' = V \cup \{v_0\}$. The multiset of edges E' contains all edges in E with multiplicity one and further edges: For each vertex $v_i \neq v_0$ with $deg(v_i) < 2m(i)$ we add the edge $\{v_0, v_i\}$ with multiplicity $2m(i) - deg(v_i)$ to E'.

If condition (i) of the lemma holds, then all nodes in the obtained extended graph have even degree: All vertices $v_i \neq v_0$ are filled up to a degree of $2m(i)$ and v_0 is incident to $\sum_{v_i \in V} \big(2m(i) - deg(v_i)\big) = \sum_{v_i \in V} 2m(i) - 2|C|$ edges. Further, condition (ii) implies that for each connected component of G_N, in the extended graph, at least one edge connects this subgraph to v_0. Hence, H_N is connected.

Following Theorem 4, the conditions (i) and (ii) imply that H_N is Eulerian. That means, there is a path starting and ending in v_0 which contains all edges, especially the edges of the original gene order graph, exactly once. Since each node $v_i \neq v_0$ has a degree of $2m(i)$, it is traversed exactly $m(i)$ times. Each such Eulerian path corresponds to a sequence of genes that contains all adjacencies in C and each gene i $m(i)$ times, as exemplified in Figure 5.2. Thus, C is consistent with respect to m.

On the contrary, if condition (i) is not satisfied, there is at least one gene i that is contained in more given adjacencies than its multiplicity $m(i)$ allows. And, if condition (ii) does not hold for any connected component c, the maximal number of adjacencies of all genes in c is exhausted and the genes cannot be put into a linear order with the remaining genes. In both cases, the existence of a valid gene order is precluded and thus, consistency is disproven. □

We transfer the general idea from the unsigned to the signed case. To this end, we adjust the definition of the gene order graph. Now, each gene i is represented by two nodes in the graph, where each such pair is connected by $m(i)$ edges.

Definition 26. (Signed Gene Order Graph) *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \{i, -i \mid i \in \mathcal{G}_N\}$.*

Then, the signed gene order graph *of C, denoted by $G^s_N(C)$, is the multigraph with the vertex set $\{v_i, v_{-i} \mid i \in \mathcal{G}_N\}$ and the multiset of edges $\big\{\{v_i, v_{-i}\}$ with multiplicity $m(i) \mid i \in \mathcal{G}_N\big\} \cup \big\{\{v_a, v_b\}$ with multiplicity one $\mid \{a, b\} \in C\big\}$.*

Similarly to the unsigned case, we can construct the signed gene order graph in $\mathcal{O}(N + |C|)$ time to verify consistency with a total time complexity of $\mathcal{O}(N + |C|)$.

Lemma 3. *Let $\mathcal{G}_N = \{1, \ldots, N\}$ be a set of genes and let $m : \mathcal{G}_N \longrightarrow \mathbb{N}$ assign a maximum copy number to each gene. Further, let C be a set of pairs $\{a, b\}$ with $a, b \in \{i, -i \mid i \in \mathcal{G}_N\}$ and $G^s_N(C) = (V, E)$ be the signed gene order graph of C.*

Then, C is consistent with respect to m if and only if the following conditions hold:

(i) $deg(v_i) \leq 2m(|i|)$ for all vertices $v_i \in V$, and

(ii) $\sum\limits_{v_i \in c} \big(2m(|i|) - deg(v_i)\big) > 0$ for all connected components c in $G^s_N(C)$.

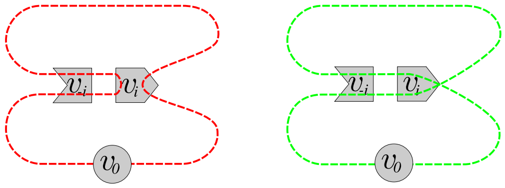
Proof. For given $\mathcal{G}_N$, m, C and $G^s_N(C)$, we extend the signed gene order graph analogously to the unsigned case, described in the proof of Lemma 2: We extend the gene order graph $G^s_N = (V, E)$ to a multigraph $H^s_N = (V', E')$, where the new vertex set contains one additional node v_0, i.e. $V' = V \cup \{v_0\}$. The multiset of edges E' contains all edges in E with multiplicity one and further edges: For each vertex $v_i \neq v_0$ with $deg(v_i) < 2m(|i|)$ we add the edge $\{v_0, v_i\}$ with multiplicity $2m(|i|) - deg(v_i)$ to E'.

Then, again, the conditions (i) and (ii) of Lemma 3 imply the existence of an Eulerian path in $H^s_N(C)$. But in this case, the correspondence of such a path to a valid gene order is not trivial. When the pair of nodes which represents gene i is traversed by a path $(\ldots, v_{-i}, v_i, \ldots)$, this relates to a signed gene order $(\ldots, i, \ldots)$, whereas a path $(\ldots, v_i, v_{-i}, \ldots)$ correlates to a signed gene order $(\ldots, -i, \ldots)$. By definition, $H^s_N(C)$ includes $m(|i|)$ edges $\{v_i, v_{-i}\}$. An Eulerian cycle passes each of these edges, but not necessarily in the above mentioned way. It might also be of the form $(\ldots, v_h, v_{-i}, v_i, v_{-i}, v_j \ldots)$ with $h \neq i \neq j$, which does not represent a signed gene order. However, as illustrated in Figure 5.3 on the next page, for every improper Eulerian cycle, there is also a proper Eulerian cycle, which does represent a signed gene order.

Thus, conditions (i) and (ii) imply not only the existence of an Eulerian path but also the existence of a valid signed gene order and hence consistency of C with respect to m.

The reverse direction of the lemma holds analogously to Lemma 2. □

Based on the definition of a gene order graph, Lemmas 2 and 3 provide algorithms to solve the consistency problem on adjacencies on sequences in time and space linear in the number of genes and in the number of given adjacencies. Both the models and the lemmas can easily be modified to allow one circular gene order or even several circular chromosomes. Only the connectivity requirement has to be relaxed correspondingly. Apart from that, instead of constructing a gene order graph at once and then checking for consistency, we can also add the adjacencies edge by edge and check for consistency in each step. This iterative procedure



(a) (b)

Figure 5.3: Illustration of the relation of improper and proper Eulerian cycles in an extended gene order graph. (a) Assume that the graph contains an Eulerian cycle $(v_0, \ldots, v_{-i}, v_i, v_{-i}, \ldots, v_0)$, which does not represent a signed gene order. Then $m(|i|) \geq 2$. Due to the construction of the extended graph, there are $m(|i|)$ edges $\{v_i, v_{-i}\}$ and at least $m(|i|)$ edges $\{v_i, v_j\}$ with $j \neq -i$. Hence, the considered Eulerian cycle has to pass node v_i again in the form $\ldots, v_k, v_i, v_l, \ldots$ with $k \neq -i \neq l$, as shown in the figure. (b) It is always possible to construct an Eulerian path $(v_0, \ldots, v_{-i}, v_i, \ldots, v_{-i}, v_i, \ldots, v_0)$. If this modification is performed for all such improperties, the obtained Eulerian cycle is proper in the sense that it represents a signed gene order $(\ldots, i, \ldots, i, \ldots)$. Note that, even though the depicted arrow head and tail imply $i > 0$, the argumentation also holds for $i < 0$.

allows us to incorporate these gene cluster models into the filtration-based framework as described in Section 3.2.

5.2.2 Evaluation on Simulated Data

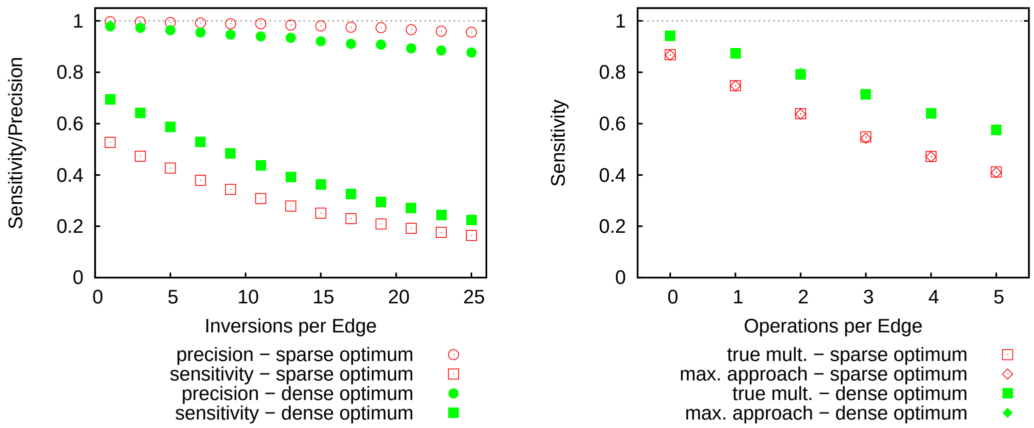
The following evaluation of the sequence-based models for adjacencies builds on the simulation setup described for permutation-based models in Section 4.5.1. Again, we assigned a permutation of length $N = 100$ to the root of a balanced, binary tree with eight leaves as shown in Figure 4.5 on page 67. In the subsequent top-down simulation, for each edge, after rearranging a gene order by a specific number of inversions, we included an additional step to generate gene orders with varying multiplicities of the genes. To this end, we repeatedly performed the following duplication-deletion operation consisting of two steps:

1. A randomly chosen, contiguous subsegment of length five is duplicated and inserted at a random position in the gene order.
2. A randomly chosen, contiguous subsegment of length five is removed from the gene order.

The finally obtained gene orders at the leaves of the tree are used as input for the reconstruction method. Then, we compared the reconstructed labelings to the simulated gene orders to compute the sensitivity and precision. All results are averaged over 100 runs.

The focus of this study is to evaluate the optimization criterion. To avoid any bias caused by a specific approach to estimate the maximal copy number, we used the true multiplicities as observed in the simulated gene orders. For comparison, we additionally implemented the maximum approach described in Section 5.1.1 on page 79.

In Figure 5.4, the precision and sensitivity of the reconstruction method on sequence-based adjacencies are shown. Similar to the permutation-based adjacencies, we observed no significant difference between the results for unsigned and signed adjacencies on sequences (data not shown). The general trend, shown in Figure 5.4(a), is also similar to the permutation-based models: High precision and moderate sensitivity, both decreasing for an increasing inversion rate; and higher sensitivity, but lower precision for the dense optimum compared to the sparse optimum. However, as expected due to the additional amount of broken adjacencies caused by the duplication-deletion operations, all values are lower than in the permutation-based evaluation. As Figure 5.4(b) confirms, the reconstruction quality decreases for an increasing amount of duplication-deletion operations. Furthermore, in this plot, the sensitivity is contrasted for different approaches to restrict the gene multiplicities. Using the simple maximum approach to estimate the copy numbers yields the same results as assuming the true copy numbers. The same holds for the precision (data not shown).



(a)

(b)

Figure 5.4: The precision and sensitivity for sparse and dense optima of the two-phase approach (Algorithm 5 on page 48) using the sequence-based gene cluster models for adjacencies. (a) The results are compared for different evolutionary rates for the model of unsigned adjacencies on sequences (Model 6). Per edge, after the inversions, five duplication-deletion operations have been performed. (b) The precision for the model of signed adjacencies on sequences (Model 7) for a fixed inversion rate of five is compared for different numbers of duplication-deletion operations. The evaluation has been performed assuming the true multiplicity of the genes and using the maximum approach described in Section 5.1.1 on page 79.

Figure 5.5 on the next page elucidates the same trend for the optimal weight as we observed and discussed for the permutation-based models on page 70. The underlying effects are stronger for an increasing number of duplication-deletion operations.

We also evaluated the results in terms of CARs. Here, overlapping (unsigned or signed) adjacencies have been assembled to one CAR if and only if the common gene did not belong to a paralogous family since, otherwise, the overlap would have been ambiguous. The obser-

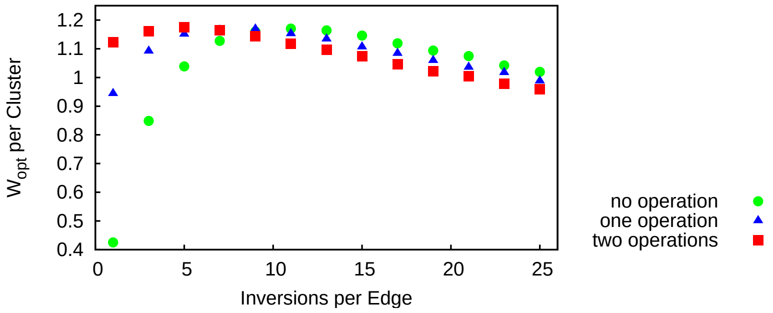
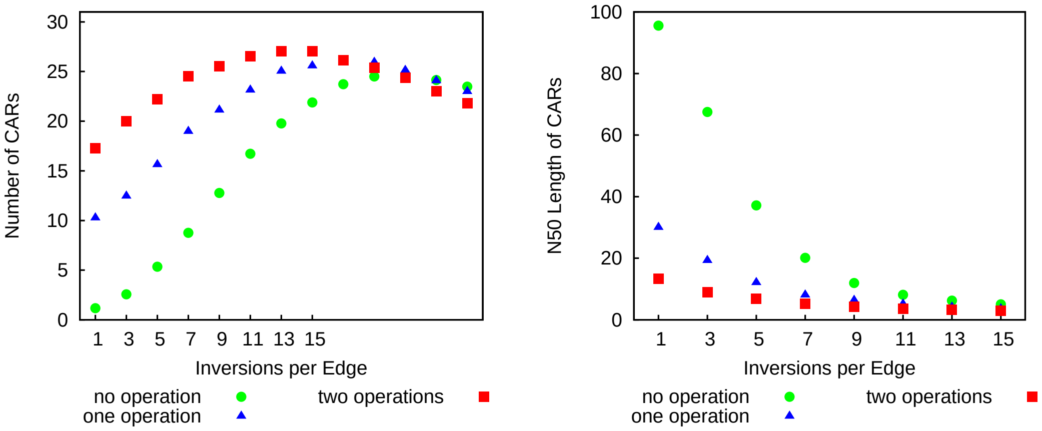


Figure 5.5: Weight of optimal labelings for unsigned adjacencies on sequences with respect to different numbers of duplication-deletion operations. The average parsimony weight is normalized by the number of clusters under consideration. Thus, the average number of gains and losses per cluster is depicted.



(a) (b)

Figure 5.6: Evaluation of the results for the dense optimum of the two-phase approach (Algorithm 5 on page 48) in terms of CARs. In both figures, the results for unsigned adjacencies on sequences are displayed for different numbers of duplication-deletion operations. (a) The average number of CARs per node. (b) The average *N50 length* of the CARs. (See Figure 4.11 on page 72 for the definition of the N50 length.)

vations regarding the number and length of the reconstructed CARs are comparable to those analyzed and explained in Section 4.5 on page 71, shown in Figure 5.6. But here, even for very low inversion rates, the CARs already break up into several smaller fragments due to the duplication-deletion procedure. In particular, the duplication operations induce paralogs, which terminate CARs by definition.

Concluding, for the sequence-based adjacency models, we observed the same trends as for the permutation-based models, where the results are worse due to the higher rate of rearrangements by the simulated duplications and deletions. Furthermore, this evaluation indicates that the maximum approach is appropriate for estimating the gene multiplicities.

5.3 Common Intervals on Sequences

We now consider the model of common intervals defined on sequences. This concept, belonging to the most prevalent definitions of gene clusters, has already been utilized in a rudimentary form on real data in 2002 by Cavalcanti *et al.* [40], who used a window of fixed size to identify regions with a conserved gene content. At the same time, Amir *et al.* [3] discussed common intervals on sequences from an algorithmical perspective. Didier [54] gave an algorithm that detects common intervals conserved in two sequences in $\mathcal{O}\left(n^2 \log n\right)$ time using $\mathcal{O}\left(n\right)$ space, where n is the sequence length. Later, Schmidt and Stoye [149] showed that this algorithm can be modified to obtain quadratic runtime and developed another algorithm with a time and space complexity of $\mathcal{O}\left(n^2\right)$, which can be extended to find common intervals conserved in a set of genomes.

Here, we define common intervals on sequences as an instance of the abstract gene cluster model. In line with other studies, we base our definition on the notion of *character sets*, which enables us to formalize the cluster model in an elegant way. Since we will utilize this term for models on signed sequences later on, we directly define it for the general, signed case.

Definition 27. (Character Set) *Let $g = (a_1, \ldots, a_{|g|})$ be a signed sequence. Then, the* character set *of g, denoted $\mathcal{CS}(g)$, is the set of all elements in g:*

$$\mathcal{CS}(g) := \left\{|a| \mid a \in \{a_1, \ldots, a_{|g|}\}\right\}.$$

Model 8. (Common Intervals on Sequences) *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of genes. Then, the gene cluster model for* common intervals on sequences *is defined as $(\mathcal{U}^s_c, \mathcal{C}^s_c, \sqsubseteq^s_c)$, where*

- *$\mathcal{U}^s_c$ is the set of all sequences over $\mathcal{G}_N$;*
- *$\mathcal{C}^s_c$ is the set of all subsets $c \subseteq \mathcal{G}_N$ with $|c| > 1$; and*
- *$c \sqsubseteq^s_c g :\Leftrightarrow$ genome g contains a substring g' such that $\mathcal{CS}(g') = c$.*

In contrast to the permutation-based model, here, a common interval can occur multiple times in one genome. Furthermore, one occurrence of a cluster in a genome may contain several occurrences of the same gene, as illustrated by the following example.

Example 13. Consider Model 8 for $N = 6$ and genome $g = (5, 4, 2, 1, 2, 3, 6) \in \mathcal{U}^s_c$. Then, the common interval $\{1, 2, 3, 4\} \in \mathcal{C}^s_c$ is contained in g as illustrated below, where the corresponding substring is underlined:

$$(5, \underline{4, 2, 1, 2, 3}, 6).$$

◇

To actually incorporate the model into our algorithmical framework, we would have to provide a method to compute the restricted genome set with respect to any given multiplicity threshold function m, or, at least, an algorithm to solve the consistency problem for this gene cluster definition. However, the latter problem is NP-complete. Thus, there is no efficient solution unless P equals NP, which is commonly assessed to be unlikely.

Theorem 5. *The consistency problem for common intervals on sequences is NP-complete.*

Sketch of proof. To prove the NP-completeness of the problem, we first show that it is in the complexity class NP, by providing an algorithm that verifies a given solution, i.e. a putatively proper gene order, for correctness in polynomial time. Then, we reduce a variant of the Hamiltonian cycle problem, which is known to be NP-complete, to the consistency problem for common intervals in polynomial time, which implies NP-hardness of the latter problem. Together, we get NP-completeness. The details of the proof are laid out in the appendix. □

In the following section, we discuss variants of common intervals on sequences, in particular framed and nested common intervals.

5.4 Variants of Common Intervals on Sequences

As already mentioned in Section 4.2, beside the classical definition of common intervals, there are different generalizations discussed in the literature – also on sequences. For instance, r-window clusters have been used by Friedman and Hughes [79] to identify and finally count duplicated blocks of genes in a genome. However, they do not discuss their method in terms of complexity. Durand and Sankoff [56] analyzed the statistical significance of r-window clusters but without providing a method for cluster detection.

In contrast, for max-gap clusters, search algorithms have been developed. Pasek *et al.* [132] developed a method with reasonable runtime to find clusters which are conserved among a set of genomes. He and Goldwasser [86] proposed a polynomial time algorithm to find max-gap clusters conserved in two sequences.

Recently, further generalizations of the above models, known under the term *approximate gene clusters*, have been introduced. In these models, a cluster is said to be conserved in a set of genomes even if not the exact gene content defined by the cluster occurs in a substring of each genome: Some of the genes may be missing and the substring is allowed to be interrupted by other genes, as long as a certain set-theoretic distance between the cluster occurrences is met. Rahmann and Klau [137] designed integer linear programs for a general class of definitions for approximate clusters. A concrete model, different representations and corresponding algorithms have been discussed by Chauve *et al.* [41]. Recently, Böcker *et al.* [30] proposed the model of *median gene clusters* and a detection algorithm, which they successfully applied on real data.

As shown in the preceding section, the consistency problem is NP-complete for basic common intervals. Hence, any generalization is NP-hard and thus, although the aforementioned models may be formulated as instances of the abstract gene cluster model, they are not suitable for an integration into our algorithmical framework for gene cluster reconstruction.

In contrast to generalizations, there are also other cluster models which are restricted variants of common intervals. For such cluster definitions, the consistency problem is not necessarily NP-hard.

Framed Common Intervals

First, we redefine the model of framed common intervals, which has been introduced on permutations in the previous chapter, for sequences.

Model 9. (Framed Common Intervals on Signed Sequences) *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of all genes. A* framed common interval $[a\, I\, b]$, *defined on signed sequences, consists of two* extremities a *and* b *with* $|a|, |b| \in \mathcal{G}_N$, *and a set of* inner elements $I \subseteq \mathcal{G}_N$. *We say that* $[a\, I\, b]$ *is* contained *in a signed sequence* s, *if and only if in* s, a *is followed by* b *or* $-b$ *by* $-a$, *and the character set of the substring in-between these extremities is equal to* I. *Then, the gene cluster model of* framed common intervals on signed sequences *is defined as* $(\mathcal{U}^s_f, \mathcal{C}^s_f, \sqsubseteq^s_f)$, *where*

- $\mathcal{U}^s_f$ *is the set of all signed sequences over* $\mathcal{G}_N$;
- $\mathcal{C}^s_f$ *is the set of all framed common intervals over* $\mathcal{G}_N$; *and*
- $[a\, I\, b] \sqsubseteq^s_f g :\Leftrightarrow [a\, I\, b]$ *is contained in* g.

Compared to the permutation-based model, here, in one cluster, a gene can be extremity and inner element, or even left and right extremity at the same time. Apart from that, analogously to basic common intervals, a cluster can occur multiple times in one genome, and one gene can be contained several times in one cluster occurrence, as illustrated by the following example.

Example 14. Consider Model 9 for $N = 6$ and genome $g = (5, 4, -2, -1, 2, -3, 6) \in \mathcal{U}^s_f$. Then, the framed common interval $[4\,\{1, 2\}\,{-3}] \in \mathcal{C}^s_f$ is contained in g as illustrated by the *box diagram* below, where the occurrences of the extremities and the inner elements are surrounded by rectangles:

$$g = (\ +5, \boxed{+4,} \boxed{-2,\ -1,\ +2,} \boxed{-3,}\ +6\ ).$$

◇

In the discussion of permutation-based cluster models, we showed an important relation between framed and basic common intervals. Following Observation 2 on page 62, any framed common interval corresponds to a pair of overlapping basic common intervals. For the sequence-based models, this correlation does not hold, as illustrated by the following example.

Example 15. Let $(\mathcal{U}^s_f, \mathcal{C}^s_f, \sqsubseteq^s_f)$ be the gene cluster model of framed common intervals on signed sequences over $\{1, 2, 3\}$ and $(\mathcal{U}^s_c, \mathcal{C}^s_c, \sqsubseteq^s_c)$ be the model of common intervals on unsigned sequences over $\{1^t, 1^h, 2^t, 2^h, 3^t, 3^h\}$.

Consider the signed sequence $g = (+2, -1, -2, +3) \in \mathcal{U}_f^s$ and the framed common interval $[1\,\{2\}\,3] \in \mathcal{C}_f^s$. Then, $\mathrm{ext}(g) = (2^t, 2^h, 1^h, 1^t, 2^h, 2^t, 3^t, 3^h)$ is the corresponding extended, unsigned sequence of g, and $c_a = \{1^h, 2^t, 2^h\}$ and $c_b = \{2^t, 2^h, 3^t\}$ are the common intervalsas defined in Observation 2. Although both c_a and c_b are contained in $\mathrm{ext}(g)$, we have $[1\,\{2\}\,3] \not\sqsubseteq_f^s g$. $\diamond$

Nevertheless, there is another correlation between these models, namely with respect to the consistency problem: Any instance of this problem for common intervals can be reduced to an instance for framed common intervals. Based on this, we can deduce the following statement.

Theorem 6. *The consistency problem for framed common intervals on signed sequences is NP-complete.*

Sketch of proof. To prove the NP-completeness of the problem, we first show that it is in the complexity class NP. Then, we reduce the consistency problem for common intervals to the consistency problem for framed common intervals in polynomial time, which implies NP-hardness of the latter problem. Together, we get NP-completeness. The details of the proof are laid out in the appendix. □

Nested Common Intervals

We now discuss the model of nested common intervals to analyze whether the strict assumption of nestedness is strong enough to allow an efficient verification of consistency. We first give a formal definition of the model. Slightly more restrictive variants, as introduced by Blin and Stoye [28], will be addressed later.

Model 10. (Nested Common Intervals on Sequences) *Let $\mathcal{G}_N := \{1, \ldots, N\}$ be the set of all genes. A* nested common interval*, defined on sequences, is either an unordered pair of genes $\{a, b\}$, or an ordered pair (c, a) of a nested common interval c and a gene a. Let the character set of a nested common interval be the set of all contained genes: $\mathcal{CS}((c, a)) := \mathcal{CS}(c) \cup \{a\}$. The gene cluster model of* nested common intervals on sequences *is defined as $(\mathcal{U}_n^s, \mathcal{C}_n^s, \sqsubseteq_n^s)$, where*

- *$\mathcal{U}_n^s$ is the set of all sequences over $\mathcal{G}_N$;*
- *$\mathcal{C}_n^s$ is the set of all nested common intervals over $\mathcal{G}_N$;*
- *$\{a, b\} \sqsubseteq_n^s g :\Leftrightarrow$ genes a and b are adjacent in genome g; and*
  $(c, a) \sqsubseteq_n^s g :\Leftrightarrow$ in genome g, gene a is adjacent to a substring g' of g such that
  $\mathcal{CS}(c) = \mathcal{CS}(g')$ and $c \sqsubseteq_n^s g'$.

Similar to the other sequence-based cluster models discussed above, any nested common interval may occur multiple times in one genome and one gene may be contained multiple times in the occurrence of a cluster in one genome. Analogously to framed common intervals, one gene may be incorporated in the definition of one cluster several times.

Example 16. Consider Model 10 for $N = 6$ and genome $g = (5, 4, 2, 1, 2, 3, 6) \in \mathcal{U}^s_n$. Then, the nested common interval $((\{2, 3\}, 1), 4) \in \mathcal{C}^s_n$ is contained in g as illustrated below, where the occurrences of the subclusters are indicated by lines:

$$(5, \underline{4}, \underline{2, \underline{1}, \underline{2, 3}}, 6)\,.$$

In contrast, we have $((\{1, 3\}, 2), 4) \not\sqsubseteq^s_n g$ since, although gene 4 is adjacent to a substring with character set $\{1, 2, 3\}$, none of the occurrences of gene 2 is adjacent to a substring with character set $\{1, 3\}$.

Note that cluster $(\{2, 3\}, 3)$ is not contained in g, because 3 is not adjacent to a substring with character set $\{2, 3\}$, whereas cluster $(\{1, 2\}, 2)$ is contained in g:

$$(5, 4, \underline{2, 1}, \underline{2}, 3, 6)\,.$$

◇

The relation, stated in Observation 3 on page 64, that the occurrence of any nested common interval is equivalent to the occurrences of a stack of basic common intervals does not hold in the sequence-based case, as illustrated by the following example.

Example 17. Let $(\mathcal{U}^s_n, \mathcal{C}^s_n, \sqsubseteq^s_n)$ be the gene cluster model of nested common intervals on sequences and $(\mathcal{U}^s_c, \mathcal{C}^s_c, \sqsubseteq^s_c)$ be the model of common intervals on sequences, both over the set of genes $\{1, 2, 3, 4\}$. Consider the nested common interval $((\{1, 2\}, 3), 4) \in \mathcal{C}^s_n$ and the sequence $g = (2, 1, 4, 2, 3, 1) \in \mathcal{U}^s_n = \mathcal{U}^s_c$. Although we have $\{1, 2\} \sqsubseteq^s_c g$, $\{1, 2, 3\} \sqsubseteq^s_c g$ and $\{1, 2, 3, 4\} \sqsubseteq^s_c g$, the nested cluster is not contained in g, because gene 3 is not adjacent to a substring with character set $\{1, 2\}$. ◇

But actually, the common intervals that we used in the NP-hardness proof of the consistency problem for common intervals can be replaced by certain nested common intervals such that the remainder of the argumentation holds similarly. Thus, we get:

Theorem 7. *The consistency problem for nested common intervals on sequences is NP-complete.*

Sketch of proof. Theorem 7 is proven similarly to Theorem 5. The details of the proof are laid out in the appendix. □

Circular Gene Orders

Based on the above NP-completeness results, we can deduce NP-completeness of the consistency problem for all the considered gene cluster models on circular chromosomes in a straight forward fashion.

To redefine the cluster models, we allow any gene cluster to appear at the end of a sequence $(\ldots, g_{|g|-1}, g_{|g|})$ such that its occurrence can be continued at the beginning of the sequence $(g_1, g_2, \ldots)$, i.e. we assume circular gene orders.

Corollary 2. *The consistency problem on circular sequences is NP-complete for the gene cluster models of common intervals, framed common intervals and nested common intervals.*

Proof. The consistency problem for the gene cluster models on linear sequences is NP-complete according to Theorems 5, 6 and 7.

Let the set of genes $\mathcal{G}_N$, the set of clusters C and the multiplicity function m define an instance of the consistency problem for common intervals, framed common intervals or nested common intervals on linear sequences. Then, we reduce this instance to a problem instance of the corresponding consistency problem on circular sequences in polynomial time as follows:

- $\mathcal{G}_{N+2} := \mathcal{G}_N \cup \{N+1, N+2\}$,
- $C' := C \cup \{c\}$, where c is a gene cluster containing exactly the genes $N+1$ and $N+2$ (the common or nested common interval $\{N+1, N+2\}$, or the framed common interval $[N+1\,\{\}\,N+2]$, respectively),
- $m' := m \cup \{N+1 \longmapsto 1, N+2 \longmapsto 1\}$.

If C is consistent with respect to m, then there is a linear sequence $g = (a_1, \dots, a_l)$ satisfying the requirements of m and containing all clusters in C. Obviously, the circular sequence $g' := (a_1, \dots, a_l, N+1, N+2)$ also contains all clusters in C and cluster c and satisfies the requirements of m'.

Now, we show the opposite implication: If C' is consistent with respect to m', then there is a circular sequence $g' = (a_1, \dots, a_l, N+1, N+2)$ satisfying the requirements of m' and containing all clusters in C'. Since no cluster in C spans the genes $N+1$ and $N+2$, the linear sequence $g := (a_1, \dots, a_l)$ contains all clusters in C' except for c, i.e. all clusters in C, and satisfies the requirements of m.

The proofs that the consistency problem on circular sequences is in the complexity class NP for the considered gene cluster models are similar to the corresponding proofs for linear sequences (Lemmas 4, 6 and 8). □

In fact, the reduction procedures on which the above proofs are based produce instances of the consistency problem where the multiplicity for some genes grows with the instance size. This suggests to reconsider the problems in the context of *fixed parameter tractability*. That means, we could try to find algorithms with a runtime complexity that is exponential solely in the multiplicity of the genes. When we assume the multiplicity $m(a)$ being small for all genes, say $m(a) < M$ for all a and a fixed M, the exponent in the running time would be constant and we obtain a polynomial runtime. However, first investigations in this direction were not promising. Moreover, the maximum copy number of genes observed in real data can be large in general. For instance, in the dataset that will be discussed in the next chapter, we observed gene families with up to ten occurrences in one genome.

Apart from that, the maximum cluster sizes required in the proofs are quite small. Even if we restrict the consistency problem to common or nested common intervals containing at most five genes or to framed common intervals of maximal size eight, respectively, the problems remain NP-complete. Furthermore, even if we restrict any occurrence of a common or nested common interval within a genome to contain each gene only once, NP-completeness holds. (Please do not confuse this issue with the model of sequences without duplications, where each gene is allowed to occur at most once in an entire genome.) Moreover, if we preclude a nested common interval to contain any gene multiple times, e.g. $(\{a, b\}, a)$, or if we preclude any gene to be left and right extremity of a framed common interval, e.g. $[a\, I\, a]$, the above proofs apply and thus NP-completeness still holds.

In summary, we identified a severe border between gene cluster models for which we can verify consistency efficiently and those for which we cannot. The complexity raises drastically from linear for adjacencies to NP-hard for more general cluster models, even if they are strongly restrictive. Since the number of consistency problem instances to be solved within the overall reconstruction algorithm is exponential, integrating an exponential-time method to solve these instances would result in an impractical runtime complexity. We thus refrained from implementing any such method. However, as we will see in the following chapter, applying the permutation-based cluster models or the model of adjacencies on sequences yields valuable reconstruction results.

Chapter 6

The Evolution of the *Corynebacterium* Genome

To evaluate our framework on real data and to illustrate the results that can be achieved, we exemplarily applied it to a set of several species of the genus *Corynebacterium*.

These bacteria are Gram-positive, rod-shaped and belong to the phylum *Actinobacteria*. They have attracted attention for a long time and are still of great interest mainly due to two reasons. Firstly, the pathogenicity of several Corynebacteria: There are both bacteria pathogenic for animals and human pathogenic species. Most notable, *C. diphtheriae* causes diphtheria in human. But the spectrum of diseases caused by different members of this genus is as manifold as are their habitats. Beside soil, water and plants, some species are found in the mucosa and on the skin of animals and humans. The second reason is that Corynebacteria are not only relevant to science but also are of great industrial importance. Some species can be utilized for the production of nucleotides, enzymes and further valuable metabolites. The most prominent and also an impressive example is the mass production of millions of tons of amino acids, such as L-glutamate and L-lysine, per year.

For an extensive description and broad collection of knowledge about Corynebacteria, we recommend the classical "Handbook of *Corynebacterium glutamicum*" [62] or the more recent book "Corynebacteria: Genomics and Molecular Biology" [37].

The goal of our evaluation is to confirm evolutionary scenarios reconstructed by our method to be reasonable. Furthermore, we exemplify how our results can raise new open questions, indicate errors in the input data, and can be used for further analyses.

We begin this chapter with a description of the data set, including the preprocessing and some overall observations on the results in Section 6.1. Then, in Section 6.2, we address a crucial characteristic of the preprocessing and exemplify how it can influence our results. In Section 6.3, we evaluate the reconstructed evolutionary history of gene clusters involved in the central metabolism. Finally, in Section 6.4, we give an example on how results of our analysis can guide the functional annotation of a gene.

Table 6.1: Listing of *Corynebacterium* [80] and *Mycobacterium* [125] genomes used in the study. For each strain, its proper identifier, its length, its G+C content and the number of coding regions is given. For the latter value, also putative, automatically predicted coding regions are considered.

Genome	Length *(in base pairs)*	G+C Content *(in %)*	Number of Coding Regions
C. glutamicum ATCC 13032	3,282,708	53.84	3,058
C. glutamicum R	3,314,179	54.13	3,052
C. efficiens YS-314	3,147,090	63.14	2,950
C. diphtheriae NCTC 13129	2,488,635	53.48	2,264
C. ulcerans 809	2,483,825	53.31	2,228
C. aurumicosum ATCC 700975	2,790,189	60.63	2,539
C. jeikeium K411	2,462,499	61.40	2,105
C. urealyticum DSM 7109	2,369,219	64.19	2,024
C. urealyticum DSM 7111	2,269,359	64.40	1,890
C. kroppenstedtii DSM 44385	2,446,804	57.46	2,037
M. tuberculosis KZN 1435	4,398,250	65.61	4,059

6.1 Data Preparation and General Results

The genomic data of ten *Corynebacterium* strains belonging to eight different species was taken from the GenDB database [80, 121]. This platform uses a variety of different gene prediction methods and further databases to automatically analyze and annotate genome sequences. Furthermore, these predictions are permanently refined by experts working on that data. Additionally, *Mycobacterium tuberculosis* was chosen as an outgroup. We downloaded the data of strain KZN 1435 from the well established NCBI database [6, 125, 160]. Table 6.1 gives an overview of the genomes under consideration, including exact identifiers and additional information.

The phylogenetic tree this study is based on is shown in Figure 6.1. It was extracted from a tree comprising 53 Corynebacteria and two outgroup species as proposed by Khamis *et al.* in 2004 [103]. They computed the sequence similarity (percent identity) of the *rpoB* gene contained in all species with the CLUSTAL W program [156] to construct a tree using the neighbor joining method [140] of the PHYLIP package [73].

In the following section, we describe how we generated the actual input for the reconstruction algorithm – orders of gene family identifiers – from the given data.

6.1.1 Homology Assignment and Further Preprocessing

All genomes were available as ordered lists of genes or, generally spoken, coding regions. Each region was specified by its amino acid sequence and its position in the genome. Addi-

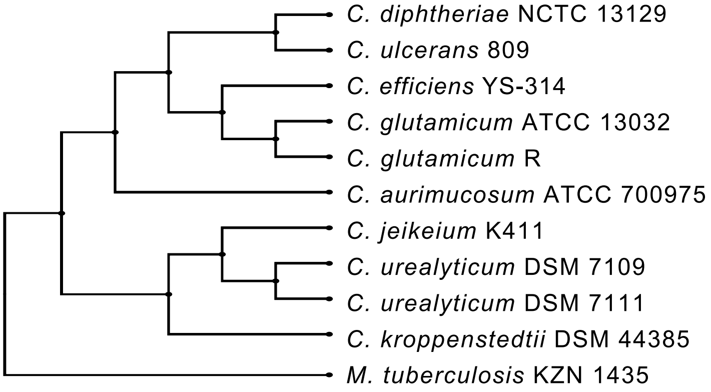

C. diphtheriae NCTC 13129
C. ulcerans 809
C. efficiens YS-314
C. glutamicum ATCC 13032
C. glutamicum R
C. aurimucosum ATCC 700975
C. jeikeium K411
C. urealyticum DSM 7109
C. urealyticum DSM 7111
C. kroppenstedtii DSM 44385
M. tuberculosis KZN 1435

Figure 6.1: Phylogenetic tree used in the study. Note that only the topology is visualized without considering edge lengths since these are not used by the reconstruction method.

tionally, most genes were annotated with the corresponding gene product or function. Other, automatically predicted genes were preliminarily annotated as "hypothetical protein".

We now detail the main steps to transform the given genomes into a sequence of identifiers for homologous genes.

As already discussed in the Introduction, several tools are available for the purpose of homology assignment. We selected GhostFam [150] for this study. It considers paralogous genes such that the potential of the sequence-based cluster models can be exploited. Furthermore, the following pipeline allows to process the genomic data as provided by GenDB and to easily produce the input required for our reconstruction method. It is based on single genes rather than contiguous regions consisting of several genes or gene fragments. This way, we could utilize the generally comprehensive and reliable data in GenDB, which in particular includes detailed information about the individual genes contained in the genomes, to the full extent. Moreover, considering 'real' genes is most suitable to examine the evolution of 'real' gene clusters and the function of comprised genes.

Sequence comparison with BLAST. To identify genes being homologous, one usually compares their sequences: A high sequence similarity indicates homology. As a basis for the further processing, we performed an all-against-all comparison of all coding sequences within and across all genomes. We used BLASTP [2] (with expectation threshold 0.1 and alignment view option 'tabular') to compute pairwise local alignments, characterized by: the rate of matching characters in the alignment (identity in percent), the coverage of the alignment (in percent), and the BLAST E-value.

Homology assignment with GhostFam. The above pairwise similarity values were further processed by GhostFam to assign genes into families of homologs as follows. In a first step, two genes in a genome are considered as paralogous, if the coverage of the local alignment, and either the amount of matches or the E-value meet given thresholds:

$$\textit{coverage} > c_p \text{ and } (\textit{ identity } > i_p \text{ or } \textit{E-value } < e_p)\,.$$

Single-linkage clustering is used to create families of paralogous genes. Then, in an iterative grouping phase, two families of paralogs are merged if at least r percent of the contained genes are considered as orthologous, where orthology is defined similarly to paralogy, but with separate parameters:

$$\textit{coverage} > c_o \text{ and } (\textit{identity} > i_o \text{ or } \textit{E-value} < e_o)\,.$$

For our study, we chose the thresholds given below. This choice will be further discussed in the following Section 6.2.

$$\begin{array}{llll} c_p = 90\% & i_p = 95\% & e_p = 1.0\text{ E-150} & \\ c_o = 60\% & i_o = 60\% & e_o = 1.0\text{ E-70} & r = 70\% \end{array}$$

We obtained 3,964 gene families and 11,371 singleton genes. The distribution of the family sizes, depicted in Figure 6.2, shows a clear peak for gene families containing eleven genes. This supports our parameter setting since our data set consists of eleven genomes and we expect many gene families to appear once in each genome.

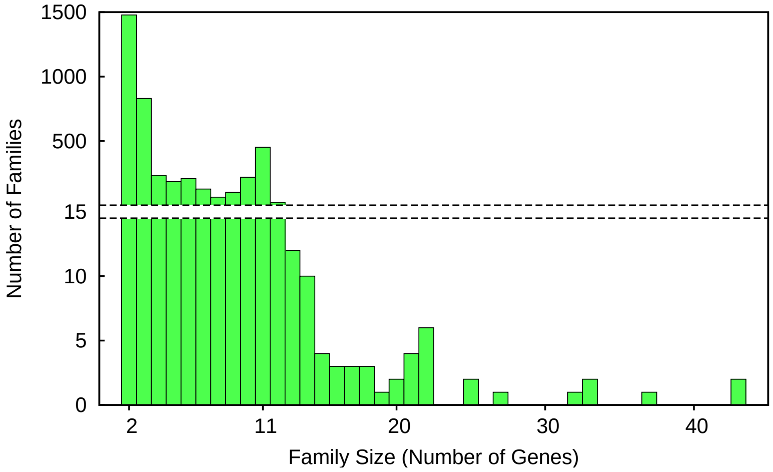


Figure 6.2: The distribution of the gene family sizes obtained by GhostFam for the genome data and parameter setting described in the text. Note the split scale of the ordinate.

Finally, the genomes were exported from GhostFam as a sequence of genes, where each gene is represented by its family identifier. The data is stored in tabular form (tab separated values) with one line per gene, containing its orientation (+ or −), functional annotation and further information.

Removal of IS elements. Insertion sequences (IS) are regions usually coding for transposases and can occur multiple times and distributed over the whole genome. They are known to change their number and position within and across replicons frequently and almost arbitrarily. Since the positioning of IS elements has no deeper meaning with respect to gene clusters, we neglected their appearance in this context. If the annotation of a gene contained a substring indicating a transposase or insertion sequence, we removed all genes belonging to the same family from the gene orders. In our dataset, this affected 65 gene families and 142 singleton genes such that finally 3899 gene families were left.

By applying this filtering *after* the homology assignment, we also removed 61 homologs which were assigned to families containing IS elements, but were annotated as “hypothetical protein”. Beside 608 putative IS elements, only very few differently annotated genes were removed: eleven tRNAs and two other genes. On the one hand, nine of these tRNAs were members of a very large family comprising 43 homologs, which suggests misassignment. On the other hand, recall that the homology thresholds were chosen very restrictively. Hence, this might also indicate misannotation in the other cases. For reasons of transparency, we refrained from any manual correction.

Handling of paralogs. Gene order data exported from GhostFam can be used as input data for the gene cluster reconstruction method without further preprocessing. To estimate the multiplicity of the genes, we implemented the maximum approach as described in Section 5.1.1. We chose this rather relaxed approach to minimize any bias caused by the copy number restriction.

But in Chapter 4, we introduced gene cluster models which are defined on permutations. These models cannot be applied to the data directly. Paralogous gene families, which are families containing genes with multiple occurrences within the same genome, have to be handled. This dataset contained 252 such gene families.

Simply removing them induces new, false gene clusters. To avoid this effect, we chose a very rigorous approach: Only those segments of a genome were considered for the reconstruction which did not contain any paralogous gene. To this end, a further preprocessing step was performed. We substituted each of these paralogous genes by a marker, whereas consecutive paralogous genes were substituted by one single marker. During the scan of a gene order for all the clusters it contains, we skipped all clusters that contained a marker. Strictly speaking, this procedure does still not yield proper permutations. Even so, as already discussed in Section 5.1.2 on page 79, sequences without duplications can be easily handled by simply ignoring the ‘missing’ genes while checking for consistency.

6.1.2 Reconstruction Results

In this section, we give an overview of the overall results. We compare the reconstructed labelings for different gene cluster models in terms of statistics to infer general observations.

For this study, we computed the optimal labelings, both sparse and dense, for unsigned and signed adjacencies on sequences, and for common intervals and framed common intervals on sequences without duplications. For the latter models, we restricted the cluster size to certain maxima to keep the running times within reasonable limits.

The calculations were performed on a Sparc machine with 32 GB main memory using up to eight 2600 MHz processors in parallel to estimate the conflict index for the labelings of the different nodes of the tree. The running times are given in Table 6.2 on the next page. This table also contrasts the weights of the obtained labelings. Normalizing by the total number

Table 6.2: Overall results of the reconstruction of ancestral Corynebacteria gene clusters for different cluster models: unsigned and signed adjacencies on sequences, and common intervals and framed common intervals on sequences without duplications. For (framed) common intervals, the maximal size has been restricted to four and six, respectively. The optimal weight W_{opt} is given in total and normalized by $|\mathcal{C}|$, the number of gene clusters under consideration. The running times are given in CPU minutes. That means, the individual time periods of all processors which have been used in parallel are summed up.

Model	Optimal Weight		Running Time
	total	*per cluster*	*(in minutes)*
Sequence-based			
unsigned adjacencies	6950	1.18	6.33
signed adjacencies	7078	1.19	22.07
Permutation-based			
common intervals ≤ 4	15123	1.14	345.56
common intervals ≤ 6	22440	1.12	817.58
framed common int. ≤ 4	15405	1.15	5213.31

of clusters shows a similar optimal weight for all models. We observe that, on average, each cluster is gained or gets lost less than 1.2 times in its putative evolutionary history, which corresponds to our simulation-based evaluation.

In Table 6.3 on the facing page, the obtained labelings, both the sparse and the dense optimum, are compared. Please recall the definition of *contiguous ancestral regions* (CARs) from page 71. The following general trend can be observed among all cluster models: The dense optimum possesses more clusters, more CARs, more flexible CARs and longer CARs than the sparse labeling. Furthermore, in line with our simulation-based evaluation, in each case, the most parsimonious weight was equal to the optimal weight. That means, the sparse most parsimonious labeling was consistent and thus optimal. In the dense approach, only few clusters had to be deleted from the most parsimonious labeling to reach optimality: at most twelve, on average 9.4 (data not shown). Thus, the potential of the two-phase approach could be fully utilized.

Comparing the results for unsigned and signed adjacencies shows that considering gene orientation yields slightly fewer clusters and CARs. But at the same time, the average CAR length increases, which means that predominantly small CARs are affected. This observation suggests that if a longer segment of the gene order is conserved then the orientation of the genes is conserved as well. Whereas in very short conserved segments, the orientation is not strongly preserved. In the case of common intervals and framed common intervals, this effect is superposed. These models do not only differ in whether orientation is encountered or not: In framed common intervals, the relative localization of the framing elements has to be conserved. Independent of gene orientation, not all conserved regions reconstructed under the model of common intervals are also detectable under framed common intervals. For instance, flexible regions at the end of a conserved segment cannot be spanned by framed common intervals. Therefore, the corresponding CARs of framed common intervals are chopped by

Table 6.3: Statistics of the reconstructed labelings for different cluster models: unsigned and signed adjacencies on sequences, and common intervals and framed common intervals on sequences without duplications. The maximal size of the (framed) common intervals has been restricted to four and six, respectively. For each model, the results for the sparse and dense optima are listed. For each value, the average over all nine internal nodes in the tree is given. The number of clusters reconstructed for one node ($|\lambda(v)|$) is given in total and normalized by $|\mathcal{C}|$, the number of gene clusters under consideration. Further, the number of contiguous ancestral regions (CARs) reconstructed for a node is given, including the number of flexible CARs. Note that CARs composed of adjacencies cannot be flexible by definition. Finally, the average length of all CARs of a node is listed. The definition of CARs can be found on page 71.

Model	Optimum	Clusters		CARs		Average
		total	*normalized*	*total*	*flexible*	CAR Length
Sequence-based						
unsigned adj.	sparse	1071.13	0.18	319.13	-	4.85
	dense	1216.13	0.21	341.13	-	5.25
signed adj.	sparse	1058.13	0.18	308.25	-	4.94
	dense	1205.38	0.20	332.75	-	5.30
Permutation-based						
common int. ≤ 4	sparse	2001.00	0.15	275.63	5.88	4.78
	dense	2260.50	0.17	292.88	4.38	5.11
common int. ≤ 6	sparse	2629.88	0.13	275.63	5.88	4.78
	dense	2958.38	0.15	292.88	4.38	5.11
fr. comm. int. ≤ 4	sparse	1968.00	0.15	271.75	3.50	4.78
	dense	2232.25	0.17	291.25	2.13	5.08

the respective regions as found for common intervals, which decreases the average CAR length. A manual inspection showed that most flexible CARs are of this type. That is also the main factor for the significant difference in the number of flexible CARs.

As expected, the special treatment of paralogous genes yielded fewer and shorter CARs for the permutation-based models. Increasing the maximal size of common intervals from four to six did not change the results, neither the sparse nor the dense labeling. Although the actual number of reconstructed clusters differed, the resulting CARs were exactly the same. We therefore refrained from extending the cluster size further.

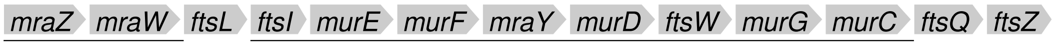
Only very few flexible CARs were reconstructed. An examination of these CARs showed that the degree of freedom which is left only allows a re-ordering of at most two neighboring genes. Since the model of common intervals is the most relaxed definition of gene clusters, other models, like nested common intervals, would not reveal any inner structure of the gene clusters, for example nestedness – at least for this dataset.

In Conclusion, the above observations on the Corynebacteria dataset suggest the following strategy: To detect gene clusters containing paralogous genes, we have to choose the cluster model of adjacencies, where very small clusters can be filtered out by using the signed variant. To detect clusters which span a certain variability in the gene orders, we have to select common intervals. Since these flexible regions are very small and possess no inner structure, basic common intervals strongly restricted in size suffice.

So far, we only discussed the results in terms of statistics. Although these findings give an interesting overview and elucidate characteristics of the specific gene cluster models, they do not shed any light on the biological meaning, nor on the practical use of the actual reconstructed gene clusters. To tackle these questions, in the following three sections, we exemplarily analyze specific gene clusters in the context of their putative evolutionary history.

6.2 Cell Wall Biosynthesis Block

To verify the accuracy of our method, we analyzed the reconstructed history of the *dcw*-cluster. It has been widely studied in literature, especially in Corynebacteria, for instance by Valbuena *et al.* [158]. This gene cluster contains cell-devision and cell-wall related genes and is known to be conserved among all considered species as illustrated by the arrows below. Note that, throughout this chapter, any graphical representation of a cluster is scaled neither with respect to the sizes nor to the absolute locations of the genes.



Although the orientation and order of all 13 genes is exactly conserved in all genomes, our method did not reconstruct the complete cluster for all ancestral nodes. Independently of the cluster model used, for some parts of the phylogenetic tree, only the underlined segments have been found.

However, this is no malfunction of the actual reconstruction method. The splitting is rooted in the preprocessing of the data, namely the homology assignment. As shown in Table 6.4, the genes of the *ftsL* family are not conserved highly enough among all genomes to be assigned

Table 6.4: Blast results for the *ftsL* gene family. The values are exemplarily given with respect to one representative, the family member in *C. glutamicum* ATCC 13032. The members of the first two genomes are assigned to the same family as the representative. Values exceeding the thresholds for homology assignment are highlighted.

Genome	Coverage	Identity	E-value
	(in %)	*(in %)*	*(exponent)*
C. glutamicum R	**62**	**100**	**-72**
C. efficiens YS-314	**67**	**70**	-49
C. diphtheriae NCTC 13129	50	47	-23
C. ulcerans 809	**72**	46	-31
C. aurumicosum ATCC 700975	48	45	-23
C. jeikeium K411	46	41	-19
C. urealyticum DSM 7109	51	39	-15
C. urealyticum DSM 7111	46	40	-15
C. kroppenstedtii DSM 44385	51	41	-19

into one gene family. The same holds for the gene family *ftsQ*. Therefore, in the input of the reconstruction method, these genes were represented as *different* genes. Consequently, corresponding clusters containing them could not be identified as 'the same' cluster in all genomes and were thus not reconstructed uniformly across the whole tree.

One natural way to overcome this obstacle is to relax the thresholds for the homology assignment to ensure that also the more diverse gene families *ftsL* and *ftsQ* are assembled completely. But this would have contrary effects on other gene families: Correctly separated families might be merged wrongly. Table 6.5 exemplifies that, even according to the thresholds as restrictive as they were chosen for this study, differently annotated genes are assigned to the same family. Actually, further tightening the parameters would yield the correct clustering in this example. As usual, we have to trade off sensitivity for specificity.

Table 6.5: Blast results for similar genes of different gene families. The values are exemplarily given with respect to one representative, the *aspA* gene in *C. glutamicum* ATCC 13032. According to the chosen thresholds, all genes are assigned to one family although the annotations suggest a partition into different families. On closer inspection, the Blast values support a corresponding separation. Values exceeding the thresholds for homology assignment are highlighted. The annotations are abbreviated as follows: (1) aspartate ammonia-lyase, (2) hypothetical protein, (3) fumarate hydratase.

Genome	Coverage	Identity	E-value	Annotation
	(in %)	*(in %)*	*(exponent)*	
C. diphtheriae NCTC 13129	**99**	**75**	**-180**	(1)
C. ulcerans 809	**89**	**72**	**-180**	(1)
C. efficiens YS-314	**89**	**87**	**-180**	(1)
C. glutamicum R	**100**	**99**	**-180**	(1)
C. aurumicosum ATCC 700975	**88**	**80**	**-180**	(1)
C. jeikeium K411	**89**	**72**	**-180**	(1)
C. urealyticum DSM 7109	**89**	**70**	**-180**	(1)
C. urealyticum DSM 7111	**89**	**70**	**-180**	(1)
C. kroppenstedtii DSM 44385	**85**	**73**	**-180**	(1)
C. glutamicum R	**86**	53	**-131**	(2)
C. diphtheriae NCTC 13129	**87**	37	**-77**	(3)
C. ulcerans 809	**86**	37	**-74**	(3)
C. efficiens YS-314	**87**	39	**-78**	(3)
C. glutamicum R	**87**	39	**-77**	(3)
C. glutamicum ATCC 13032	**87**	39	**-77**	(3)
C. aurumicosum ATCC 700975	**86**	38	**-70**	(3)
C. jeikeium K411	**87**	38	**-82**	(3)
C. urealyticum DSM 7109	**87**	38	**-84**	(3)
C. urealyticum DSM 7111	**87**	38	**-84**	(3)
C. kroppenstedtii DSM 44385	**87**	36	**-76**	(3)

This example emphasizes that homology assignment is not a trivial task which can be solved by simply setting the parameters 'correctly'. When drawing conclusions from any gene order study like this, one has to be aware that already the preprocessing can influence the outcome immensely.

6.3 Central Metabolism

One essential part of the central metabolism of every living cell is the *citric acid cycle*, also known as the *tricarboxylic acid cycle* (TCA cycle). This metabolic pathway is fed by specific metabolites, used as carbon sources, which are transformed in a chain of chemical reactions. In some steps of this process, carbon is oxidized to CO_2 and the released energy is available for further operations in the cell. Usually, sugars are used for carbon input. But some organisms comprise variants of this pathway. For instance, the so-called *glyoxylate cycle* allows to process other resources, like acetate, ethanol and fatty acids.

For a detailed description of the TCA cycle and related pathways in *Corynebacterium glutamicum*, we refer to the recent review of Bott [33].

We now discuss the evolutionary history of the spatial organization of certain genes which play important roles in the TCA cycle or the glyoxylate cycle, respectively, as illustrated by Figure 6.3.

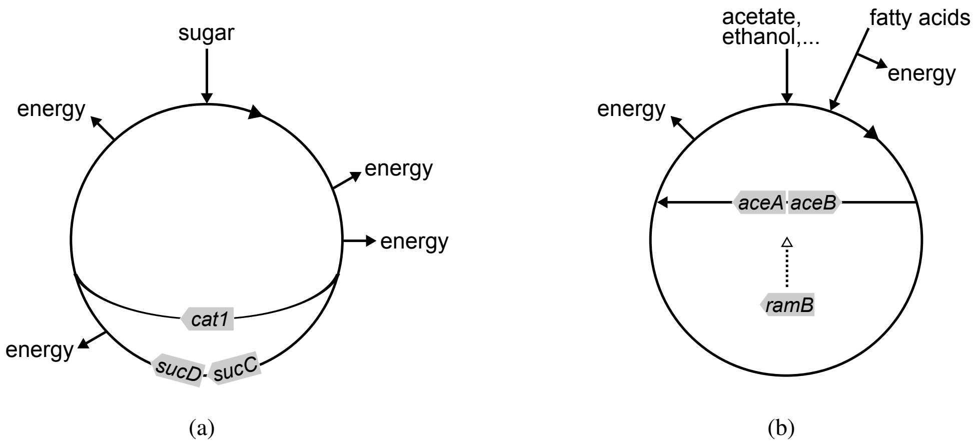


Figure 6.3: Abstract illustrations of two metabolic pathways in the central metabolism of Corynebacteria. For reasons of clarity, only those genes which are discussed in this section are depicted. (a) Sugar is used as a carbon source and stepwise processed in the *TCA cycle*. In different steps, energy is made available. Two subpathways can be passed through alternatively: Either a *cat1* gene or the *sucC-sucD* cluster is involved. In the latter case, additional energy is released. (b) The *glyoxylate cycle* can be fed by different sources. If fatty acids are used, additional energy can be made available during degradation. In this pathway, the *aceA-aceB* cluster, regulated by the *ramB* gene, plays an essential role.

The *aceA-aceB* cluster. To process alternative carbon sources like acetate, ethanol or fatty acids, the genes *aceA* and *aceB* are essential. They occur adjacent in most of the considered genomes. All genomes contain the *ramB* gene, which, beside others, regulates *aceA* and *aceB*. In some species, all three genes are co-located.

The occurrence and clustering of these genes is displayed in Figure 6.4. We can see that the possession of the cluster in the individual genomes is in line with their environmental background. *C. diphtheriae* is for instance found in the human throat, *C. aurumicosum* in the human gut, and *C. ulcerans* on the cow udder. For these habitats, the glyoxylate cycle is not utilized and, therefore, *aceA* and *aceB* are not contained in the genomes of these species. In contrast, *C. jeikeium*, *C. urealyticum* and *C. kroppenstedtii* inhabit the human skin where fatty acids are abundant. Correspondingly, these species possess the two genes. *C. efficiens* and *C. glutamicum* are soil bacteria, adopted to a wide spectrum of metabolites. Depending on the actual sources available, they utilize the glyoxylate cycle. In all these genomes, *aceA*

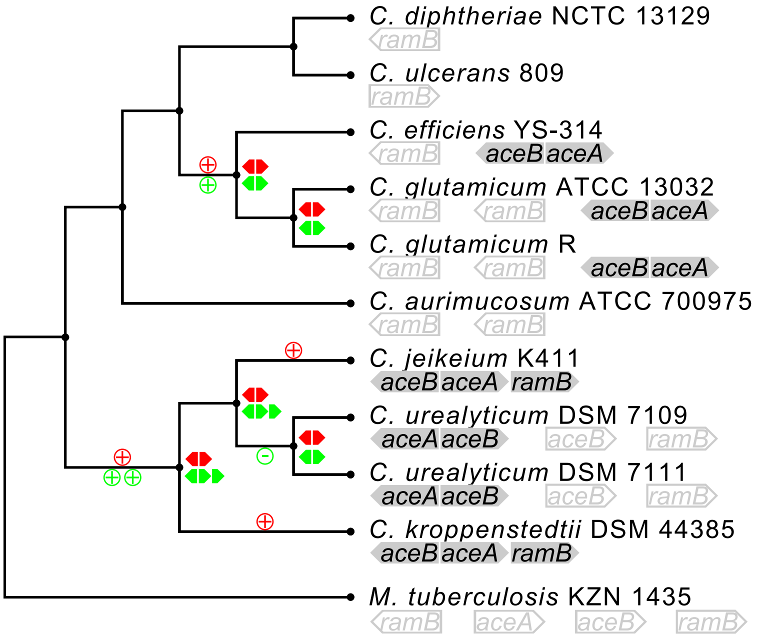


Figure 6.4: Reconstructed evolutionary history of the *aceA-aceB* cluster. Arrows at the leaf nodes represent the genes under consideration, where gray filled arrows are used for genes occurring clustered, and distantly occurring genes are shown in light gray. Note that the order of genes is shown as obtained by the pre-processing, including wrongly predicted paralogs of *ramB*, which however do not affect the clustering.
The optimal labelings are depicted as follows: The clustered genes are stylized by small arrows. Two arrows are shown if only the adjacency of *aceA* and *aceB* is reconstructed, whereas three arrows are shown if *ramB* is reconstructed to be adjacent as well. The sparse labeling is displayed in red. Underneath, the dense labeling is displayed in green. The evolutionary events are indicated by plus and minus symbols, colored red and green, respectively.

and *aceB* are adjacent, with the exception of *M. tuberculosis*, which we used as an outgroup. In this genome, both genes are contained but not co-localized.

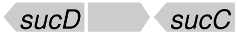
Since *aceB* is a paralogous gene family, the cluster can only be analyzed using sequence-based models. Based on the gene cluster model of (unsigned or signed) adjacencies on sequences, our reconstruction predicted two scenarios for the evolution of the *aceA*-*aceB* cluster. These results are illustrated in Figure 6.4. Both labelings, sparse and dense, propose that the adjacency of *aceA* and *aceB* originated in two subtrees separately. An examination of the genetic neighborhood revealed a diverse context between the subtrees and thus affirmed this proposition. The scenarios disagree in the evolution of the adjacency of *aceA* and *ramB*. According to the sparse labeling, it was gained two times: once in *C. jeikeium* and once in *C. kroppenstedtii*. By contrast, the dense labeling reconstructed that it originated only once for the whole subtree and was then lost again for *C. urealyticum*. A similar genetic context in *C. jeikeium* and in *C. kroppenstedtii* clarifies this disagreement. It is very unlikely that both the co-localization and the context developed twice independently. Hence, the dense labeling is more reliable in this case.

This example demonstrates the benefit of an outgroup. Both scenarios predict four evolutionary events. In this case, no further co-optima exist. Without the outgroup, there are several more solutions.

The *sucC-sucD* cluster. As shown in Figure 6.3(a), there are two alternative pathways in the TCA cycle. Usually, the genes *sucC* and *sucD* are involved. Otherwise, for instance if they are not present in the genome, *cat1* can by used instead. But in this case, less energy is made available.

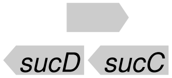
The *cat1* gene is present in all Corynebacteria whereas only *C. aurumicosum*, the soil bacteria *C. efficiens* and *C. glutamicum*, and the outgroup *M. tuberculosis* comprise *sucC* and *sucD*. They appear adjacent in all genomes except for *C. efficiens* where, according to the input gene order, an additional gene lies between them.

Since neither *sucC* nor *sucD* are paralogous gene families, any of the available gene cluster models can be used to analyze the evolution of this cluster. The result is presented in Figure 6.5. Following the sparse optimal labeling, the *sucC*-*sucD* cluster originated three times. On the contrary, the dense approach proposes that it was lost three times. However, one could argue that the cluster is not lost completely in *C. efficiens* and the latter scenario is more reasonable. Double-checking the original data in the data base actually reveals that the cluster is not lost at all. Even though the final gene order data induces a one dimensional picture like



,

according to the underlying original data, the separating gene is encoded in another reading frame:



Further investigation clarified this dubious observation by identifying the third gene as a falsely predicted coding region. Under these circumstances, the dense labeling, actually comprising only two losses of the cluster, is a unique optimal scenario.

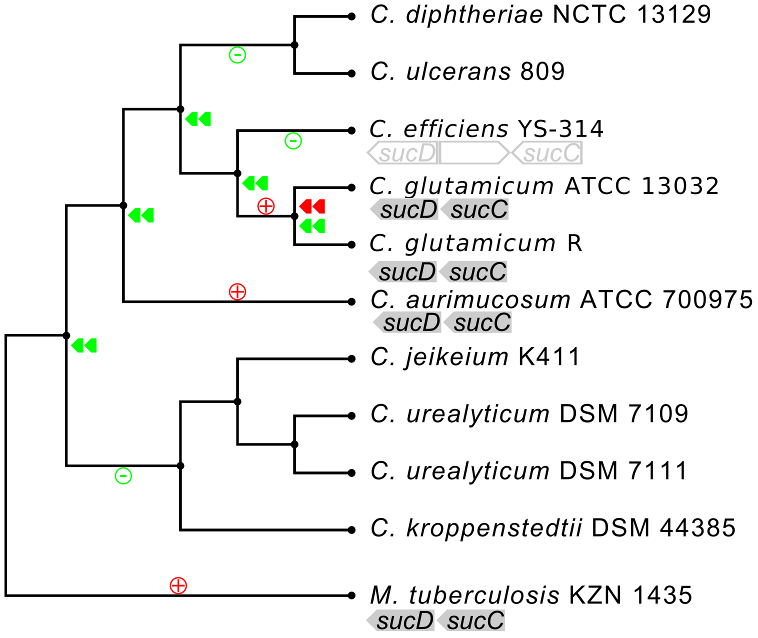


Figure 6.5: Reconstructed evolutionary history of the *sucC-sucD* cluster. The graphical representation is similar to those in Figure 6.4 on page 105.

The reconstructed loss of the *sucC-sucD* cluster in the history of *C. jeikeium*, *C. urealyticum* and *C. kroppenstedtii* can be explained by their living conditions. These bacteria inhabit the human skin where enough fatty acids are available. Therefore the glyoxylate cycle is predominantly used due to the more efficient conversion of energy. If other products of the TCA cycle are needed, the *cat1* variant can be used.

However, there is no obvious reason for the second loss of the cluster in one of the ancestors of *C. diphtheriae* and *C. ulcerans*. These bacteria do not have the genes *aceA* and *aceB* and thus cannot process the glyoxylate cycle. Since performing the *cat1* variant of the TCA cycle is less efficient, a loss of the *sucC-sucD* cluster should have impaired the fitness and during evolution, selection should have prevented the loss. In conclusion, our results raise questions like:

- What was the evolutionary advantage of losing the *sucC-sucD* cluster?
- Is there an hitherto unknown alternative, more efficient pathway?
- In which habitat or under which conditions did the ancestors live?

The analysis of the *aceA-aceB* and *sucC-sucD* cluster confirmed the benefit of using outgroup data and certified that the reconstruction results are reasonable. Furthermore, it posed new open questions pointing at interesting details in the evolution of essential pathways in the central metabolism of Corynebacteria. In the next section, we exemplify that phylogeny-based gene cluster analyses not only raise but also assist to answer questions.

6.4 DNA Synthesis

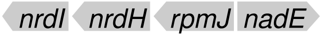
In the previous two examples, we intentionally searched for clusters of genes which are known to be involved in certain pathways and then examined the history. We also took the opposite approach and first scanned the reconstructed histories and then examined the function of the genes.

To this end, we compared the labelings of the different ancestral nodes in the tree. This revealed many CARs which were conserved across almost all genomes and were therefore reconstructed for the whole tree. Another trend we expected based on our simulations and also observed in our results was that longer CARs assigned to nodes near the leaves are reconstructed in fragments for higher levels in the tree. But we also found various CARs with an ‘interesting’ putative history, for instance CARs which:

- were reconstructed only for a subtree,
- similarly evolved in different lineages,
- were found with a different conserved context in different subtrees,
- were reconstructed in similar form for the whole tree, except of single genes or their orientation, etc.

An analysis of all these CARs would go beyond the scope of this thesis. Here, we exemplarily discuss one CAR and show how our results guide the functional annotation of a gene.

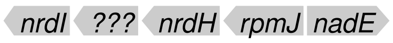
Based on the gene cluster models of (signed or unsigned) adjacencies on sequences, we found a CAR that was reconstructed for almost all nodes of the *Corynebacterium* phylogeny. All but one node were labeled with a CAR containing the segment:



,

where each arrow represents an oriented gene. Following their annotations, the genes *nrdI* and *nrdH* are functionally related. Namely, they are involved in ribonucleotide reduction. In this process, ribose is transformed into deoxyribose, which finally is used for the synthesis of DNA.

As already indicated, this CAR attracted our attention due to one exception. The ancestral node of the two *C. urealyticum* genomes was labeled with a slightly different CAR comprising one additional gene, whose function is not fully understood yet:



.

An obvious conjecture is that this gene is functionally related to the others.

The GenDB system, we extracted our data from, offers to automatically search different databases for sequences similar to the selected gene. In our case, the annotations of the hits indicated that the gene in question codes for a transporter, supposable for sugars. These observations suggest the following hypotheses:

1. The transporter facilitates the diffusion of sugars or phosphorized sugars as a precursor for the synthesis of deoxyribose, for instance RNA molecules, from the environment *into* the cell.
2. Defectively generated DNA is *exported* from the cell to prevent further processing and a possible malfunction later on.

One of the database matches was labeled with “efflux”, which supports the second theory. However, this was a low-scoring hit. Aside from that, a cell would rather reuse the damaged compound than exporting it. In contrast, assimilating external readily preprocessed metabolites, as stated by the first theory, is an effective way to accelerate internal production while saving resources.

To verify our conjecture, one could carry out the following experiment: Clone and insert the gene into another organism. Provide radioactively tagged sugars for a certain time and then filter the cells. Finally, radioactivity in the cells would indicate the import of sugar. A control group is used to certify that no assimilation takes place without the gene.

The realization of such an experiment is too elaborate to being performed within this study and does not serve the actual aim of the project. This gene is just one among many examples which on the one hand pose new questions and on the other hand provide first approaches to answer them.

Chapter 7

Conclusion and Outlook

In this thesis, we have introduced a unified approach, the abstract gene cluster model, that allows to define a variety of different specific models. Based on this abstract concept, we have introduced the problem of reconstructing sets of gene clusters at the inner nodes of a given phylogenetic tree. The thereby optimized objective function includes two criteria: *consistency*, i.e. for each ancestral node, we insist on the existence of a potential gene order that contains all the reconstructed clusters, and *parsimony*, i.e. we seek for a labeling that explains the gene clusters observed in the contemporary genomes with a minimal number of cluster gains and losses along the edges of the evolutionary tree.

We could show that it is always possible to find all optima using a branch-and-bound search starting with a (possibly inconsistent) most parsimonious labeling. The presented exact algorithm combines two phases. In the first phase, a sparse variant is used to find an optimum very quickly. In a second phase, this result is used to speed up a dense variant, which is more sensitive while still precise.

For several different gene cluster models, we discussed whether consistency of a given set of clusters can be verified efficiently. In particular, the models of unsigned and signed adjacencies and the model of common intervals and its variants framed and nested common intervals have been defined as instances of the abstract model and have been analyzed in detail.

Firstly, we presented efficient data structures and verification methods for these models defined on permutations: A simple graph theoretic approach can be used to represent a set of adjacencies and the corresponding set of valid gene orders, and to test for consistency in time linear to the genome length and number of given clusters. Common intervals on permutations can easily be modeled by PQ trees, which provide a polynomial time method to solve the consistency problem. Since both framed and nested common intervals correspond to certain sets of overlapping basic common intervals, consistency can be verified for these models in polynomial time as well.

Secondly, we studied the same models defined on sequences. For adjacencies, we extended the simple graphs introduced for permutations to multigraph-based data structures to represent sequences of genes. We found that a set of adjacencies is consistent if the corresponding graph is Eulerian, which can be tested in linear time. On the contrary, none of the data structures or relations found for permutation-based common intervals and its variants could be generalized to sequences. In fact, we have proven the consistency problem for common intervals on sequences to be NP-complete. Even for the more restrictive variants of framed and nested common intervals, NP-completeness holds.

The above findings are summarized in Table 7.1.

Table 7.1: Brief overview of the time complexity of the consistency problem under different gene cluster models. The space of models elaborately analyzed in this thesis is organized by the cluster concept and the underlying representation of gene order.

Gene Cluster	Gene Order Model	
Concept	Permutations	Sequences
unsigned adjacencies	linear	linear
signed adjacencies	linear	linear
common intervals	polynomial	NP-complete
framed common int.	polynomial	NP-complete
nested common int.	polynomial	NP-complete

We have implemented our reconstruction framework including all gene cluster models for which consistency is verifiable in polynomial time and studied the effectiveness and efficiency of our approach on simulated data. Due to the correctness of our algorithm, the evaluation of our method refers to the characteristics of our definition of optimality. The results span a wide range of sensitivity. However, all reconstructed labelings show a high precision, i.e. most of the predictions are correct. This is especially important if results are to be validated biologically, as such analyses require high effort. A high precision makes a biological verification more promising or feasible at all. Most results depend neither on the simulated genome rearrangement operations nor on the gene cluster model used. Generally, we observed that in most cases the obtained clusters overlapped such that the order of the corresponding genes was determined exactly. This indicates that considering substructures such as nestedness or framing is not valuable.

Finally, we exemplarily applied our method on genomic data of ten *Corynebacterium* strains. The general characteristics of the results were in line with the simulation-based evaluation. Additionally, we confirmed evolutionary scenarios being reasonable and exemplified how our results can initiate and support further analyses.

Future Directions

Like most parsimony-based approaches, our method suffers from some weaknesses. A main drawback is the simple weighting scheme. However, as discussed in Section 2.2.2, our theoretical investigations as well as the algorithmical strategies built thereon also apply for more general weighting functions. Solely, the method to determine an initial, most parsimonious labeling and the corresponding reoptimization procedure have to be adapted accordingly. As mentioned before, appropriate algorithms regarding several scoring schemes have been proposed in the literature.

We intentionally refrained from integrating such extensions into this basic study to avoid any bias or overfitting effects due to an elaborate weighting and to keep the framework independent of specific data and gene cluster models. But we expect that incorporating edge length and cluster size, and weighting the formation of a gene cluster higher than a loss would enhance the reconstruction results. It remains to be seen if respective expansions would increase the running time, or if sophisticated weighting schemes would yield most parsimonious labelings which are more reasonable and thus contain fewer conflicts, which would speed up the branch-and-bound search.

The identification of suitable parameters for generalized parsimony scores involves assumptions on the evolution of gene order or, more specifically, on the probabilities of certain evolutionary events. Such a probabilistic model could also be used to define a maximum-likelihood-based objective to replace the parsimony criterion. Certainly, one has to reconsider the relation to the concept of consistency, and the algorithmical framework would undergo a radical change or not be reusable at all.

It is also arguable that all clusters are weighted individually rather than considering whether a cluster is lost completely or just split up into subclusters. But, in fact, we do account for this implicitly if the subclusters are contained in the model as well: When a cluster splits up, the cluster itself and only a few subclusters get lost. In contrast, when a cluster gets lost completely, the cluster itself and all its subclusters get lost, which increases the weight much more. Further research might explore more sophisticated solutions.

As outlined in Section 5.1.1, one could also refine or relax the determination of the genes' multiplicities for sequence-based models. Although our evaluation indicated that the simple maximum approach is already sufficient, specific datasets might require an adjustment.

From a practical point of view, faster, heuristic approaches to obtain suboptimal yet reasonable labelings would be useful. In order to evaluate the quality and characteristics of such an inexact method, its results can be compared to those of our exact method. Parsimony can be obtained and verified easily whereas the criterion of consistency, particularly the detection of conflicts, is the main bottleneck in our approach. Hence, here, we find one starting point to develop heuristics.

Besides abandoning exactness, a speed-up might also be achieved by committing to a specific gene cluster model. For instance, inconsistencies of adjacencies could be analyzed more effectively using the graph-based data structures introduced in Sections 4.1 and 5.2. As already mentioned in Section 4.2, Chauve *et al.* [43] gave an algorithm that determines for a given set of common intervals on permutations which of them belong to a conflicting subset in polynomial time when the size of the clusters is restricted. This method might be integrated into our reconstruction framework to replace the complex computation of the conflict index.

In our case study on Corynebacteria data, we observed highly conserved gene order fragments which vary only in single genes. Manual inspection in several cases revealed either a dubious homology assignment or erroneous gene prediction. Examples are the cell wall biosynthesis block discussed in Section 6.2 and the *sucC-sucD* cluster discussed in Section 6.3, respectively. In these instances, the corresponding cluster could still be detected. But other clusters might be missed due to such small errors in the input. Hence, one should consider to formulate a robust gene cluster model that covers single missing or dissimilar genes. Even though gene cluster models, such as approximate gene clusters or gene teams, possess a combinatorial structure which is too complex for our purposes, more restrictive models, for instance just allowing for one error per cluster, might be feasible.

To avoid any bias by homology assignment, one could even go a step further and define a gene cluster model that will not be based on such a preprocessing step at all. Instead, a measure for gene homology could be directly integrated into the cluster definition. This way, the membership to gene families is not fixed globally for all genes in all genomes beforehand. It is rather specified implicitly in a flexible fashion in the context of each putative gene cluster individually. However, we expect the resulting complexity being too high that the labeling problem could be solved exactly for such a model.

Beyond refining the objective function and improving the gene cluster model, one could also study the reconstruction framework in the context of large parsimony. That means, instead of determining an optimal labeling for a given phylogeny, we search for such a tree that possesses the lowest optimal weight. However, as already mentioned, the classical large parsimony problem is NP-hard [82] and our tree-wise minimization step takes exponential time. In addition to this computational hurdle, preliminary attempts revealed that our gene-cluster-based objective function is not suited for minimizing over different tree topologies – at least for the basic Wagner parsimony scheme and the gene cluster models integrated so far. Nevertheless, further investigations, as described above, might allow us to reconsider large parsimony.

Apart from theoretical considerations, the quality and use of gene cluster reconstructions can eventually only be confirmed in practice. To this end, concrete results on biological data, such as exemplified in Chapter 6, should be pursued further. This way, more realistic gene cluster models, biologically reasonable parameters for parsimony weighting schemes, or even modified definitions of optimality can be evolved and finally be verified and utilized successfully.

Bibliography

[1] Z. Adam, M. Turmel, C. Lemieux, and D. Sankoff. Common intervals and symmetric difference in a model-free phylogenomics, with an application to streptophyte evolution. *J. Comp. Biol.*, 14(4):436–445, 2007.

[2] 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.

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

[4] Aristotle. Analytica posteriora (posterior analytics) complete. In R. McKeon, editor, *Introduction to Aristotle*. Random House, New York, 1941.

[5] A. Bashir, S. Volik, C. Collins, V. Bafna, and B. Raphael. Evaluation of paired-end sequencing strategies for detection of genome rearrangements in cancer. *PLoS Comput. Biol.*, 4:e1000051, 2008.

[6] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B. A. Rapp, and D. L. Wheeler. Genbank. *NucleicAcidsRes*, 28(1):15–18, 2000.

[7] S. Berard, A. Bergeron, C. Chauve, and C. Paul. Perfect sorting by reversals is not always difficult. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 4(1):4–16, 2007.

[8] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. In *Proc. of the Symposium on Combinatorial Pattern Matching, CPM 2001*, volume 2089 of *LNCS*, pages 106–117. Springer Verlag, 2001.

[9] A. Bergeron. A very elementary presentation of the Hannenhalli-Pevzner theory. *Discrete Appl. Math.*, 146(2):134–145, 2005.

[10] A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve. Reconstructing ancestral gene order using conserved intervals. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2004*, volume 3240 of *LNBI*, pages 14–25. Springer Verlag, 2004.

[11] A. Bergeron, M. Blanchette, A. Chateau, and C. Chauve. Operations on sets of conserved intervals. Report, CGL, UQAM, Montreal, Canada, 2005.

[12] A. Bergeron, C. Chauve, F. de Mongolfier, and M. Raffinot. Computing common intervals of k permutations, with applications to modular decomposition of graphs. In *Proc. of the European Sympoisum on Algorithms, ESA 2005*, volume 3669 of *LNCS*, pages 779–790. Springer Verlag, 2005.

[13] A. Bergeron, C. Chauve, and Y. Gingras. Formal models of gene clusters. In A. Z. Ion Măndoiu, editor, *Bioinformatics Algorithms: Techniques and Applications*, Wiley Book Series on Bioinformatics. Wiley, 2008.

[14] A. Bergeron, S. Corteel, and M. Raffinot. The algorithmic of gene teams. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2002*, volume 2452 of *LNCS*, pages 464–476. Springer Verlag, 2002.

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

[16] A. Bergeron, J. Mixtacki, and J. Stoye. Reversal distance without hurdles and fortresses. In *Proc. of the Symposium on Combinatorial Pattern Matching, CPM 2004*, volume 3109 of *LNCS*, pages 388–399. Springer Verlag, 2004.

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

[18] A. Bergeron and J. Stoye. On the similarity of sets of permutations and its applications to genome comparison. In *Proc. of the Int. Conference on Computing and Combinatorics, COCOON 2003*, volume 2697 of *LNCS*, pages 68–79. Springer Verlag, 2003.

[19] 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.

[20] M. Bernt, D. Merkle, and M. Middendorf. Genome rearrangement based on reversals that preserve conserved intervals. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 3(3):275–288, 2006.

[21] M. Bernt, D. Merkle, and M. Middendorf. Solving the preserving reversal median problem. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 5(3):332–347, 2008.

[22] M. Bernt, D. Merkle, K. Ramsch, G. Fritzsch, M. Perseke, D. Bernhard, M. Schlegel, P. F. Stadler, and M. Middendorf. Crex: inferring genomic rearrangements based on common intervals. *Bioinformatics*, 23(21):2957–2958, 2007.

[23] J. Besemer and M. Borodovsky. GeneMark: web software for gene finding in prokaryotes, eukaryotes and viruses. *Nucleic Acids Res.*, 33(Web Server issue):W451–W454, 2005.

[24] A. Bhutkar, W. M. Gelbart, and T. F. Smith. Inferring genome-scale rearrangement phylogeny and ancestral gene order: A drosophila case study. *Genome Biol.*, 8(11):R236, 2007.

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

[26] E. Blanco, G. Parra, and R. Guigó. Using geneid to identify genes. *Curr. Protoc. Bioinformatics*, Chapter 4:Unit 4.3, 2007.

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

[28] G. Blin and J. Stoye. Finding nested common intervals efficiently. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2009*, volume 5817 of *LNBI*, pages 59–69, 2009.

[29] S. Böcker, K. Jahn, J. Mixtacki, and J. Stoye. Computation of median gene clusters. In *Proc. of the Intern. Conference on Research in Computational Molecular Biology, RECOMB 2008*, volume 4955 of *LNCS*, pages 331–345, 2008.

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

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

[32] M. Borodovski and J. McIninch. Genmark: Parallel gene recognition for both DNA strands. *Computers Chem.*, 17(2):123–133, 1993.

[33] M. Bott. Offering surprises: TCA cycle regulation in corynebacterium glutamicum. *Trends Microbiol.*, 15(9):417–425, 2007.

[34] G. Bourque and L. Zhang. Models and methods in comparative genomics. *Advances in Computers*, 68:60–105, 2006.

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

[36] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268:78–94, 1997.

[37] A. Burkovski, editor. *Corynebacteria: Genomics and Molecular Biology*. Caister Academic Press, 2008.

[38] S. Bérard, A. Bergeron, and C. Chauve. Conservation of combinatorial structures in evolution scenarios. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2004*, volume 3388 of *LNCS*, pages 1–14. Springer Verlag, 2005.

[39] J. H. Camin and R. R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1965.

[40] A. R. O. Cavalcanti, R. Ferreira, Z. Gu, and W. H. Li. Patterns of gene duplication in saccharomyces cerevisiae and caenorhabditis elegans. *J. Mol. Evol.*, 56(1):28–37, 2002.

[41] C. Chauve, Y. Diekmann, S. Heber, J. Mixtacki, S. Rahmann, and J. Stoye. On common intervals with errors. Report 2006-02, Technische Fakultät der Universität Bielefeld, Abteilung Informationstechnik, 2006.

[42] C. Chauve, U. U. Haus, T. Stephen, and V. P. You. Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. Extended version, submitted.

[43] C. Chauve, U. U. Haus, T. Stephen, and V. P. You. Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2009*, volume 5817 of *LNBI*, pages 48–58. Springer Verlag, 2009.

[44] C. Chauve and E. Tannier. A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. *PLoS Comput. Biol.*, 4(11):e1000234, 2008.

[45] F. Chen, A. J. Mackey, C. J. Stoeckert, and D. S. Roos. OrthoMCL-DB: querying a comprehensive multi-species collection of ortholog groups. *Nucleic Acids Res.*, 34(Database issue):D363–D368, 2006.

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

[47] A. Christoffels, E. Koh, J. Chia, S. Brenner, S. Aparicio, and B. Venkatesh. Fugu genome analysis provides evidence for a whole-genome duplication early during the evolution of ray-finned fishes. *Mol. Biol. Evol.*, 21(6):1146–1151, 2004.

[48] T. Cremer and C. Cremer. Chromosome territories, nuclear architecture and gene regulation in mammalian cells. *Nat. Rev. Genet.*, 2(4):292–301, 2001.

[49] M. Csűrös. Ancestral reconstruction by asymmetric wagner parsimony over continuous characters and squared parsimony over distributions. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2008*, pages 72–86. Springer Verlag, 2008.

[50] Y. Cui, L. Wang, D. Zhu, and X. Liu. A (1.5 + epsilon)-approximation algorithm for unsigned translocation distance. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 5(1):56–66, 2008.

[51] T. Dandekar, B. Snel, M. Huynen, and P. Bork. Conservation of gene order: A fingerprint of proteins that physically interact. *Trends Biochem. Sci.*, 23(9):324–328, 1998.

[52] C. Darwin. *On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life*. John Murray, London, UK, 1859.

[53] A. L. Delcher, D. Harmon, S. Kasif, O. White, and S. L. Salzberg. Improved microbial gene identification with Glimmer. *Nucleic Acids Res.*, 27(23):4636–4641, 1999.

[54] G. Didier. Common intervals of two sequences. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2003*, volume 2812 of *LNBI*, pages 17–24. Springer Verlag, 2003.

[55] Y. Diekmann, M.-F. Sagot, and E. Tannier. Evolution under reversals: Parsimony and conservation of common intervals. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 4(2):301–309, 2007.

[56] D. Durand and D. Sankoff. Tests for gene clustering. In *Proc. of the Intern. Conference on Research in Computational Molecular Biology, RECOMB 2002*, pages 144–154. ACM Press, 2002.

[57] R. Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7:420, 1964.

[58] J. Earnest-DeYoung, E. Lerat, and B. Moret. Reversing gene erosion: Reconstructing ancestral bacterial genomes from gene-content and order data. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2004*, number 3240 in LNCS, pages 1–13. Springer-Verlag, 2004.

[59] R. V. Eck and M. O. Dayhoff. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*. National Biomedical Research Foundation, Silver Spring, MD, USA, 1966.

[60] A. Edwards and L. Cavalli-Sfroza. The reconstruction of evolution. *Annals of Human Genetics*, 27:105–106, 1963.

[61] A. Edwards and L. Cavalli-Sfroza. Reconstruction of evolutionary trees. In V. Heywood and J. NcNeill, editors, *Phenetic and Phylogenetic Classification*, volume 6, pages 67–76. Systematics Association, London, UK, 1964.

[62] L. Eggeling and M. Bott, editors. *Handbook of Corynebacterium glutamicum*. CRC Press / Taylor & Francis Group, Boca Raton, FL, USA, 2005.

[63] N. El-Mabrouk. Genome rearrangements with gene families. In O. Gascuel, editor, *Mathematics of Evolution and Phylogeny*, chapter 11, pages 291–320. Oxford University Press, Oxford, UK, 2005.

[64] A. J. Enright, S. V. Dongen, and C. A. Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.*, 30(7):1575–1584, 2002.

[65] A. J. Enright and C. A. Ouzounis. GeneRAGE: a robust algorithm for sequence clustering and domain detection. *Bioinformatics*, 16(5):451–457, 2000.

[66] P. L. Erdős and L. A. Székely. On weighted multiway cuts in trees. *Mathematical Programming*, 65(1):93–105, 1994.

[67] H. Escriva, L. Manzon, J. Youson, and V. Laudet. Analysis of lamprey and hagfish genes reveals a complex history of gene duplications during early vertebrate evolution. *Mol. Biol. Evol.*, 19(9):1440–1450, 2002.

[68] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[69] J. S. Farris. Methods for computing Wagner trees. *Syst. Zool.*, 19(1):83–92, 1970.

[70] J. S. Farris. Phylogenetic analysis under Dollo's Law. *Syst. Zool.*, 26:77–88, 1977.

[71] J. S. Farris. Some further comments on Le Quesne's methods. *Syst. Zool.*, 26:220–223, 1977.

[72] J. Felsenstein. The number of evolutionary trees. *Syst. Zool.*, 27(1):27–33, 1978.

[73] J. Felsenstein. PHYLIP—phylogeny inference package (version 3.2). *Cladistics*, 5:164–166, 1989.

[74] J. Felsenstein, editor. *Inferring Phylogenies*. Sinauer, Sunderland, MA, USA, 2004.

[75] G. Fertin, A. Labarre, I. Rusu, E. Tannier, and S. Vialette. *Combinatorics of genome rearrangements*. MIT Press, 2009.

[76] M. Figeac and J.-S. Varre. Sorting by reversals with common intervals. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2004*, volume 3240 of *LNCS*, pages 26–37. Springer Verlag, 2004.

[77] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Syst. Zool.*, 20(4):406–416, 1971.

[78] E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960.

[79] R. Friedman and A. L. Hughes. Gene duplication and the structure of eukaryotic genomes. *Genome Res.*, 11(3):373–381, 2001.

[80] GenDB. `http://www.cebitec.uni-bielefeld.de/gendb/`. Data downloaded on 06/29/2009.

[81] D. R. Gilbert, M. Schroeder, and J. van Helden. Interactive visualization and exploration of relationships between biological objects. *Trends Biotechnol.*, 18(12):487 – 494, 2000.

[82] R. L. Graham and L. R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Math. Biosci.*, 60(2):133 – 142, 1982.

[83] A. R. Gruber, S. H. Bernhart, I. L. Hofacker, and S. Washietl. Strategies for measuring evolutionary conservation of RNA secondary structures. *BMC Bioinformatics*, 9:122, 2008.

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

[85] J. A. Hartigan. Minimum mutation fits to a given tree. *Biometrics*, 29(1):53–65, 1973.

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

[87] S. Heber, R. Mayr, and J. Stoye. Common intervals of multiple permutations. *Algorithmica*, to appear.

[88] S. Heber and J. Stoye. Finding all common intervals of k permutations. In *Proc. of the Symposium on Combinatorial Pattern Matching, CPM 2001*, volume 2089 of *LNCS*, pages 207–218. Springer Verlag, 2001.

[89] M. D. Hendy and D. Penny. A framework for the quantitative study of evolutionary trees. *Syst. Zool.*, 38:297–309, 1989.

[90] E. A. Herniou, T. Luque, X. Chen, J. M. Vlak, D. Winstanley, J. S. Cory, and D. R. O'Reilly. Use of whole genome sequence data to infer baculovirus phylogeny. *J. Virol.*, 75(17):8117–8126, 2001.

[91] R. Hoberman and D. Durand. The incompatible desiderata of gene cluster properties. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2005*, volume 3678 of *LNBI*, pages 73–87. Springer Verlag, 2005.

[92] M. Hou, C. Riemer, P. Berman, R. C. Hardison, and W. Miller. Aligning two genomic sequences that contain duplications. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2009*, volume 5817 of *LNCS*, pages 98–110. Springer Verlag, 2009.

[93] W.-L. Hsu and R. McConnell. PC trees and circular-ones arrangements. *Theor. Comput. Sci.*, 296:99–116, 2003.

[94] W.-L. Hsu and R. McConnell. PQ-trees, PC-trees, and planar graphs. In D. P. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*. 2004.

[95] T. J. P. Hubbard, B. L. Aken, K. Beal, B. Ballester, M. Caccamo, Y. Chen, L. Clarke, G. Coates, F. Cunningham, T. Cutts, T. Down, S. C. Dyer, S. Fitzgerald, J. Fernandez-Banet, S. Graf, S. Haider, M. Hammond, J. Herrero, R. Holland, K. Howe, K. Howe,

N. Johnson, A. Kahari, D. Keefe, F. Kokocinski, E. Kulesha, D. Lawson, I. Longden, C. Melsopp, K. Megy, P. Meidl, B. Ouverdin, A. Parker, A. Prlic, S. Rice, D. Rios, M. Schuster, I. Sealy, J. Severin, G. Slater, D. Smedley, G. Spudich, S. Trevanion, A. Vilella, J. Vogel, S. White, M. Wood, T. Cox, V. Curwen, R. Durbin, X. M. Fernandez-Suarez, P. Flicek, A. Kasprzyk, G. Proctor, S. Searle, J. Smith, A. Ureta-Vidal, and E. Birney. Ensembl 2007. *Nucleic Acids Res.*, 35(Database issue):D610–D617, 2007.

[96] M. A. Huynen, B. Snel, and P. Bork. Inversions and the dynamics of eukaryotic gene order. *Trends Genet.*, 17(6):304–306, 2001.

[97] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM J. Computing*, 11(4):676–686, 1982.

[98] O. Jaillon, J.-M. Aury, F. Brunet, J.-L. Petit, N. Stange-Thomann, E. Mauceli, L. Bouneau, C. Fischer, C. Ozouf-Costaz, A. Bernot, S. Nicaud, D. Jaffe, S. Fisher, G. Lutfalla, C. Dossat, B. Segurens, C. Dasilva, M. Salanoubat, M. Levy, N. Boudet, S. Castellano, V. Anthouard, C. Jubin, V. Castelli, M. Katinka, B. Vacherie, C. Biemont, Z. Skalli, L. Cattolico, J. Poulain, V. de Berardinis, C. Cruaud, S. Duprat, P. Brottier, J.-P. Coutanceau, J. Gouzy, G. Parra, G. Lardier, C. Chapple, K. J. McKernan, P. McEwan, S. Bosak, M. Kellis, J.-N. Volff, R. Guigo, M. C. Zody, J. Mesirov, K. Lindblad-Toh, B. Birren, C. Nusbaum, D. Kahn, M. Robinson-Rechavi, V. Laudet, V. Schachter, F. Quetier, W. Saurin, C. Scarpelli, P. Wincker, E. S. Lander, J. Weissenbach, and H. Roest Crollius. Genome duplication in the teleost fish tetraodon nigroviridis reveals the early vertebrate proto-karyotype. *Nature*, 431(7011):946–957, 2004.

[99] G. Jean, D. J. Sherman, and M. Nikolski. Mining the semantics of genome super-blocks to infer ancestral architectures. *J. Comp. Biol.*, 16(9):1267–1284, 2009.

[100] L. J. Jensen, P. Julien, M. Kuhn, C. von Mering, J. Muller, T. Doerks, and P. Bork. eggNOG: automated construction and annotation of orthologous groups of genes. *Nucleic Acids Res.*, 36(Database issue):D250–D254, 2008.

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

[102] R. S. Keogh, S. Cathal, and K. H. Wolfe. Evolution of gene order and chromosome number in saccharomyces, kluyveromyces and related fungi. *Yeast*, 14:443–457, 1998.

[103] A. Khamis, D. Raoult, and B. La Scola. rpoB gene sequencing for identification of corynebacterium species. *J. Clin. Microbiol.*, 42(9):3925–3931, 2004.

[104] A. G. Kluge and J. S. Farris. Quantitative phyletics and the evolution of anurans. *Systematic Zoology*, 18(1):1–32, 1969.

[105] D. E. Knuth. *The Art of Computer Programming*, volume 2 – Seminumerical Algorithms. Addison Wesley, Reading, MA, USA, 3rd edition, 1997.

[106] L. T. Kou. Polynomial complete consecutive information retrieval problems. *SIAM J. Computing*, 6(1):67–75, 1977.

[107] A. Krause, J. Stoye, and M. Vingron. Large scale hierarchical clustering of protein sequences. *BMC Bioinformatics*, 6:15, 2005.

[108] U. Kurzik-Dumke and A. Zengerle. Identification of a novel *drosophila melanogaster* gene, it angel, a member of a nested gene cluster at locus 59F4,5. *Biochim. Biophys. Acta*, 1308(3):177–181, 1996.

[109] G. M. Landau, L. Parida, and O. Weimann. Gene proximity analysis across whole genomes via PQ trees. *J. Comp. Biol.*, 12(10):1289–1306, 2005.

[110] W. C. Lathe III, B. Snel, and P. Bork. Gene context conservation of a higher order than operons. *Trends Biochem. Sci.*, 25(10):474–479, 2000.

[111] J. G. Lawrence and J. R. Roth. Selfish operons: Horizontal transfer may drive the evolution of gene clusters. *Genetics*, 143(4):1843–1860, 1996.

[112] W. J. Le Quesne. The uniquely evolved character concept and its cladistic apllication. *Syst. Zool.*, 23:513–517, 1974.

[113] W. J. Le Quesne. The uniquely evolved character concept. *Syst. Zool.*, 26:218–223, 1977.

[114] Z. Li, L. Wang, and K. Zhang. Algorithmic approaches for genome rearrangement: A review. *IEEE Trans. Systems, Man, and Cybernetics*, 36:636–648, 2006.

[115] Y.-L. Lin and T.-S. Hsu. Efficient algorithms for descendent subtrees comparison of phylogenetic trees with applications to co-evolutionary classifications in bacterial genome. In *Proc. of the International Symposium on Algorithms and Computation, ISAAC 1999*, volume 2906/2003 of *LNCS*, pages 339–351. Springer Verlag, 2003.

[116] K. Liolios, I.-M. A. Chen, K. Mavromatis, N. Tavernarakis, P. Hugenholtz, V. M. Markowitz, and N. C. Kyrpides. The genomes on line database (GOLD) in 2009: status of genomic and metagenomic projects and their associated metadata. *Nucleic Acids Res.*, 36(Database issue):D475–D479, 2009.

[117] J. Ma, A. Ratan, B. J. Raney, B. B. Suh, W. Miller, and D. Haussler. The infinite sites model of genome evolution. *Proc. Natl. Acad. Sci. USA*, 105(38):14254–14261, 2008.

[118] J. Ma, A. Ratan, B. J. Raney, B. B. Suh, L. Zhang, W. Miller, and D. Haussler. DUPCAR: reconstructing contiguous ancestral regions with duplications. *J. Comp. Biol.*, 15(8):1007–1027, 2008.

[119] J. Ma, L. Zhang, B. B. Suh, B. J. Raney, R. C. Burhans, J. W. Kent, M. Blanchette, D. Haussler, and W. Z. Miller. Reconstructing contiguous regions of an ancestral genome. *Genome Res.*, 16(12):1557–1565, 2006.

[120] W. P. Maddison. Gene trees in species trees. *Syst. Biol.*, 46(3):523–536, 1997.

[121] F. Meyer, A. Goesmann, A. C. McHardy, D. Bartels, T. Bekel, J. Clausen, J. Kalinowski, B. Linke, O. Rupp, R. Giegerich, and A. Pühler. GenDB–an open source genome annotation system for prokaryote genomes. *Nucleic Acids Res.*, 31(8):2187–2195, 2003.

[122] J. Muller, D. Szklarczyk, P. Julien, I. Letunic, A. Roth, M. Kuhn, S. Powell, C. von Mering, T. Doerks, L. J. Jensen, and P. Bork. eggNOG v2.0: extending the evolutionary genealogy of genes with enhanced non-supervised orthologous groups, species and functional annotations. *Nucleic Acids Res. Advance Access*, 2009.

[123] S. Müller, M. Hollatz, and J. Wienberg. Chromosomal phylogeny and evolution of gibbons (hylobatidae). *Hum. Genet.*, 113(6):493–501, 2003.

[124] J. P. Murnane and L. Sabatier. Chromosome rearrangements resulting from telomere dysfunction and their role in cancer. *BioEssays*, 26(11):1164–1174, 2004.

[125] NCBI Entrez Genome database. `http://www.ncbi.nlm.nih.gov/sites/entrez?db=genome`. Data downloaded on 08/06/2009.

[126] R. Overbeek, M. Fonstein, M. D'Souza, G. D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proc. Natl. Acad. Sci. USA*, 96(6):2896–2901, 1999.

[127] M. Ozery-Flato and R. Shamir. Sorting cancer karyotypes by elementary operations. In *Proc. of the RECOMB Satellite Workshop on Comparative Genomics, RCG 2008*, volume 5267 of *LNCS*, pages 211–225, 2008.

[128] A. Paccanaro, J. A. Casbon, and M. A. S. Saqi. Spectral clustering of protein sequences. *Nucleic Acids Res.*, 34(5):1571–1580, 2006.

[129] R. D. M. Page and M. A. Charleston. From gene to organismal phylogeny: Reconciled trees and the gene tree/species tree problem. *Molecular Phylogenetics and Evolution*, 7(2):231 – 240, 1997.

[130] G. Panopoulou, S. Hennig, D. Groth, A. Krause, A. Poustka, R. Herwig, M. Vingron, and H. Lehrach. New evidence for genome-wide duplications at the origin of vertebrates using an amphioxus gene set and completed animal genomes. *Genome Res.*, 13(6a):1056–1066, 2003.

[131] G. Parra, E. Blanco, and Guigó. `GeneID` in *Drosophila*. *Genome Res.*, 10:511–515, 2000.

[132] S. Pasek, A. Bergeron, J. L. Risler, A. Louis, E. Ollivier, and M. Raffinot. Identification of genomic features using microsyntenies of domains: domain teams. *Genome Res.*, 15(6):867–874, 2005.

[133] S. Penel, A.-M. Arigon, J.-F. Dufayard, A.-S. Sertier, V. Daubin, L. Duret, M. Gouy, and G. Perrière. Databases of homologous gene families for comparative genomics. *BMC Bioinformatics*, 10 Suppl 6:S3, 2009.

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

[135] P. Pipenbacher, A. Schliep, S. Schneckener, A. Schönhuth, D. Schomburg, and R. Schrader. ProClust: improved clustering of protein sequences with an extended graph-based approach. *Bioinformatics*, 18 Suppl 2:S182–S191, 2002.

[136] K. D. Pruitt, T. Tatusova, and D. R. Maglott. NCBI reference sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic Acids Res.*, 33(Database issue):D501–D504, 2005.

[137] S. Rahmann and G. W. Klau. Integer linear programs for discovering approximate gene clusters. In *Proc. of the Int. Workshop on Algorithms in Bioinformatics, WABI 2006*, volume 4175 of *LNBI*, pages 298–306. Springer Verlag, 2006.

[138] V. Ranwez, F. Delsuc, S. Ranwez, K. Belkhir, M.-K. Tilak, and E. J. Douzery. OrthoMaM: a database of orthologous genomic markers for placental mammal phylogenetics. *BMC Evol. Biol.*, 7:241, 2007.

[139] B. Raphael, S. Volik, P. Yu, C. Wu, G. Huang, E. Linardopoulou, B. Trask, F. Waldman, J. Costello, K. Pienta, G. Mills, K. Bajsarowicz, Y. Kobayashi, S. Sridharan, P. Paris, Q. Tao, S. Aerni, R. Brown, A. Bashir, J. Gray, J.-F. Cheng, P. de Jong, M. Nefedov, T. Ried, H. Padilla-Nash, and C. Collins. A sequence-based survey of the complex structural organization of tumor genomes. *Genome Biol.*, 9(3):R59, 2008.

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

[141] A. I. Salaverria, B. Espinet, A. Carrió, D. Costa, L. Astier, J. Slotta-Huspenina, L. Quintanilla-Martinez, F. Fend, F. Solé, D. Colomer, S. Serrano, R. Miró, S. Beà, and E. Campo. Multiple recurrent chromosomal breakpoints in mantle cell lymphoma revealed by a combination of molecular cytogenetic techniques. *Genes, Chromosomes and Cancer*, 47(12):1086–1097, 2008.

[142] 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.

[143] D. Sankoff. Edit distance for genome comparison based on non-local operations. In *Proc. of the Symposium on Combinatorial Pattern Matching, CPM 1992*, volume 644 of *LNCS*, pages 121–135. Springer Verlag, 1992.

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

[145] D. Sankoff and R. J. Cedergren. Simultaneous comparison of three or more sequences related by a tree. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, chapter 9, pages 253–263. Addison-Wesley, Reading, MA, 1983.

[146] D. Sankoff, M. Deneault, P. Turbis, and C. Allen. Chromosomal distributions of breakpoints in cancer, infertility, and evolution. *Theoretical Population Biology*, 61(4):497–501, 2002.

[147] D. Sankoff and P. Rousseau. Locating the vertices of a steiner tree in an arbitrary metric space. *Math. Program.*, 9:240–246, 1975.

[148] G. Sawa, J. Dicks, and I. N. Roberts. Current approaches to whole genome phylogenetic analysis. *Briefings in Bioinformatics*, 4(1):63–74, 2003.

[149] T. Schmidt and J. Stoye. Quadratic time algorithms for finding common intervals in two and more sequences. In *Proc. of the Symposium on Combinatorial Pattern Matching, CPM 2004*, volume 3109 of *LNCS*, pages 347–358. Springer Verlag, 2004.

[150] T. Schmidt and J. Stoye. Gecko and ghostfam: rigorous and efficient gene cluster detection in prokaryotic genomes. *Methods Mol. Biol.*, 396:165–182, 2007.

[151] J. Stoye and R. Wittler. A unified approach for reconstructing ancient gene clusters. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 6(3):387–400, 2009.

[152] J. Tamames. Evolution of gene order conservation in prokaryotes. *Genome Biol.*, 2(6):research0020.1–0020.22, 2001.

[153] J. Tang and B. M. E. Moret. Phylogenetic reconstruction from gene-rearrangement data with unequal gene content. In *Proc. of the Workshop on Algorithms and Data Structures, WADS 2001*, volume 2748, pages 37–46. Springer Verlag, 2003.

[154] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale. The COG database: An updated version includes eukaryotes. *BMC Bioinformatics*, 4:41, 2003.

[155] R. L. Tatusov, E. V. Koonin, and D. J. Lipman. A genomic perspective on protein families. *Science*, 278(5338):631–637, 1997.

[156] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22(22):4673–4680, 1994.

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

[158] N. Valbuena, M. Letek, A. Ramos, J. Ayala, D. Nakunst, J. Kalinowski, L. M. Mateos, and J. A. Gil. Morphological changes and proteome response of corynebacterium glutamicum to a partial depletion of ftsi. *Microbiology*, 152(Pt 8):2491–2503, 2006.

[159] W. H. Wagner. Problems in the classification of ferns. *Recent Advances in Botany*, 1:841–844, 1961.

[160] D. L. Wheeler, C. Chappey, A. E. Lash, D. D. Leipe, T. L. Madden, G. Schuler, T. A. Tatusova, and B. A. Rapp. Database resources of the national center for biotechnology information. *NucleicAcidsRes*, 28(1):10–14, 2000.

[161] T. Wittkop, J. Baumbach, F. P. Lobo, and S. Rahmann. Large scale clustering of protein sequences with FORCE – a layout based heuristic for weighted cluster editing. *BMC Bioinformatics*, 8:396, 2007.

[162] S. Wong, G. Butler, and K. H. Wolfe. Gene order evolution and paleopolyploidy in hemiascomycete yeasts. *Proc. Natl. Acad. Sci. USA*, 99(14):9272–9277, 2002.

[163] G. Yona, N. Linial, and M. Linial. Protomap: automatic classification of protein sequences and hierarchy of protein families. *Nucleic Acids Res.*, 28(1):49–55, 2000.

[164] Y. Zheng, B. P. Anton, R. J. Roberts, and S. Kasif. Phylogenetic detection of conserved gene clusters in microbial genomes. *BMC Bioinformatics*, 6:243, 2005.

Appendix A

NP-Completeness Proofs

In this chapter, we prove the NP-completeness of the consistency problem for the gene cluster models of common intervals on sequences and its variants. For all three models, we first show that the corresponding problem belongs to the complexity class NP before we prove the NP-hardness of the problem, which, together, implies NP-completeness. We show the NP-hardness with respect to common and nested common intervals by reducing the so-called *3-bipartite Hamiltonian cycle problem*, which is known to be NP-complete, to the according consistency problems. The NP-hardness with respect to framed common intervals is deduced from the NP-completeness result for common intervals.

We begin this chapter by introducing the 3-Bipartite Hamiltonian cycle problem in Section A.1. Then, we prove the NP-completeness of the consistency problem for common intervals on sequences in Section A.2. Based on this, we deduce NP-completeness of the consistency problem for framed common intervals on sequences in Section A.3. Finally, in Section A.4, we show analogously to common intervals that the consistency problem for nested common intervals on sequences is NP-complete as well.

A.1 The 3-Bipartite Hamiltonian Cycle Problem

Definition 28. (Hamiltonian Path) *Let $G = (V, E)$ be a graph and p be a path in G. Then, p is a* Hamiltonian path *if and only if p contains each vertex in V exactly once.*

Definition 29. (3-Bipartite Hamiltonian Cycle Problem) *Let $G = (V \cup W, E)$ be a connected, undirected, bipartite graph with $|V| = |W| \geq 3$, $E \subseteq \big\{\{v, w\} \mid v \in V, w \in W\big\}$ and $\deg(u) = |\{e \in E|u \in e\}| \leq 3$. For two given nodes $v' \in V$ and $w' \in W$, decide whether there exists a Hamiltonian path in G that starts in v' and ends in w'.*

Definition 30. (Hamiltonian Cycle) *Let $G = (V, E)$ be a graph and p be a path in G. Then, p is a* Hamiltonian cycle *if and only if p starts and ends in the same vertex $v' \in V$ and in-between, p contains each vertex $v \in V \backslash \{v'\}$ exactly once.*

Definition 31. (3-Bipartite Hamiltonian Cycle Problem) *Let $G = (V \cup W, E)$ be a connected, undirected, bipartite graph with $|V| = |W| \geq 3$, $E \subseteq \big\{\{v, w\} \mid v \in V, w \in W\big\}$ and $\deg(u) = |\{e \in E|u \in e\}| \leq 3$. Decide whether there exists a Hamiltonian cycle in G.*

Theorem 8. ([97]) *The 3-bipartite Hamiltonian cycle problem is NP-complete.*

A.2 Consistency Problem for Common Intervals on Sequences

Lemma 4. *The consistency problem (Problem 23 on page 77) for the gene cluster model of common intervals on sequences (Model 8 on page 87) is in the complexity class NP.*

Proof. The input I for an instance of the problem consists of the number of genes N, a multiplicity function m and a set of gene clusters C. A common interval c can be represented by a list of $|c|$ elements. Let $K := \sum_{c \in C} |c|$ be the total size of all common intervals in C. Further, we need at least N integers to represent the values of m. Altogether, this yields a total input size of $|I| := 1 + K + N$ integers.

The solution would be a genome g that contains each gene a not more than $m(a)$ times and such that for each $c \in C$ there is a substring of g with character set c. If such a genome g exists, there is always a subsequence g' of g of length $\mathcal{O}(K)$ that is also a proper solution. (The longest possible such subsequence would be a concatenation of all common intervals.) We call such a genome g' a *certificate*.

To show that the considered problem is in NP, we now provide an algorithm that can verify any correct certificate g' in time polynomial in the input size $|I|$.

For each common interval c, we proceed as follows. The certificate g' is composed of $\mathcal{O}(K)$ integers. We scan each substring of g' and test whether it has character set c. To this end, for each element in c, we scan the substring for this element, and in one additional scan, we verify that no other element is in the substring. Hence, each test can be performed in $\mathcal{O}(|c|K)$ steps. Thus a common interval can be processed in $\mathcal{O}\left(|c|K^3\right)$ steps and all common intervals can be processed in $\mathcal{O}\left(K^4\right)$ steps. In a second step, we scan g' and count the occurrences of each gene in $\mathcal{O}(K)$ steps. Then, we verify the correct multiplicities in

$\mathcal{O}(N)$ steps. Altogether, we verify any correct certificate in $\mathcal{O}\left(K^4 + N\right)$ steps, which is polynomial in the input size:

$$(K^4 + N) \leq |I|^4 .$$

□

Lemma 5. *The consistency problem (Problem 23) for the gene cluster model of common intervals on sequences (Model 8) is NP-hard.*

The proof is laid out in the following section.

Theorem 5. *The consistency problem (Problem 23) for the gene cluster model of common intervals on sequences (Model 8) is NP-complete.*

Proof. Theorem 5 follows directly from Lemmas 4 and 5. □

Proof of Lemma 5

The 3-bipartite Hamiltonian cycle problem is NP-complete [97]. Now, we show that the 3-bipartite Hamiltonian cycle problem is reducible to the 3-bipartite Hamiltonian path problem in polynomial time.

Let $G = (V \cup W, E)$ be a given instance of the 3-bipartite Hamiltonian cycle problem. Without loss of generality, we assume that $\deg(u) \geq 2$ for all $u \in V \cup W$. Otherwise, G can obviously not contain a Hamiltonian cycle. Consider an arbitrary node $v \in V$. Node v has either two or three neighbors in W. Let $\{v, w_1\}$, $\{v, w_2\}$ and as the case may be $\{v, w_3\}$ be the corresponding edges. Since each Hamiltonian cycle has to pass through v, it has to contain exactly two of these edges. If it does not contain $\{v, w_1\}$, it has to contain $\{v, w_2\}$. That means, G contains a Hamiltonian cycle if and only if G contains a Hamiltonian path from v to w_1 or from v to w_2.

Hence, the 3-bipartite Hamiltonian cycle problem can be solved by verifying the minimal degree in $\mathcal{O}(|V|)$ steps and then (if necessary) solving the 3-bipartite Hamiltonian path problem at most twice: Does G contain a Hamiltonian path from v' to w'?

We now further reduce the 3-bipartite Hamiltonian path problem to the consistency problem for common intervals on sequences. For reasons of comprehensibility, we represent the genes in $\mathcal{G}_N$ by various symbols instead of numbers from one to N. However, we could assign a unique number to each of these symbols and thus do not lose generality in the following problem reduction. We further define $U := V \cup W$, and, since the edge $\{v', w'\}$ cannot be part of a Hamiltonian path, we can remove it from G without altering the result: $G' := (V \cup W, E')$ with $E' := E \backslash \{v', w'\}$.

Reduction. The reduction procedure is based on a technique used by L. T. Kou [106]. Given $G' = (V \cup W, E')$ and the nodes $v' \in V$ and $w' \in W$, we compose an instance of the consistency problem for common intervals on sequences in time polynomial in $|U|$ as follows.

- $\mathcal{G}_N := U \cup E' \cup \{\lhd, \rhd, \bar{s}_v, \bar{s}_w, \bar{\bar{s}}_v, \bar{\bar{s}}_w\}$
  $\cup\, \hat{U}$ with $\hat{U} = \{\hat{u} \mid u \in U\}$
  $\cup\, \bar{U}$ with $\bar{U} = \{\bar{u}, \bar{\bar{u}} \mid u \in U\}$.

- $$m(a) := \begin{cases} \deg(a), & \text{if } a \in U, \\ 1, & \text{if } a \in E', \\ |E'|, & \text{if } a \in \{\lhd, \rhd\}\ , \\ 1, & \text{if } a \in \{\bar{s}_v, \bar{s}_w, \bar{\bar{s}}_v, \bar{\bar{s}}_w\}, \\ 1, & \text{if } a \in \hat{U}, \\ \deg(a) - 2, & \text{if } a = \bar{u} \in \bar{U} \backslash \{\bar{v}', \bar{w}'\}, \\ \deg(a) - 1, & \text{if } a = \bar{u} \in \{\bar{v}', \bar{w}'\}\,. \end{cases}$$

- $C := C_E \cup C_v \cup C_w \cup C_{\hat{U}} \cup C_{\bar{v}} \cup C_{\bar{w}} \cup C_{\bar{v}'} \cup C_{\bar{w}'}$ with

$$\begin{aligned} C_E &= \big\{\{e,v\}, \{e,v,w\}, \{e,v,w,\rhd\}, \{e,v,w,\rhd,\lhd\}, \\ &\qquad \{e,w\}, \{e,w,v\} \mid e = \{v,w\} \in E'\big\}, \\ C_v &= \big\{\{\bar{s}_v, \bar{\bar{v}}_v\}, \{\bar{s}_v, \hat{v}'\}, \{\bar{s}_v, \hat{v}', \rhd\}, \{\bar{s}_v, \hat{v}', \rhd, v'\}\big\}, \\ C_w &= \big\{\{\bar{s}_w, \bar{\bar{w}}_w\}, \{\bar{s}_w, \hat{w}'\}, \{\bar{s}_w, \hat{w}', \lhd\}, \{\bar{s}_w, \hat{w}', \lhd, w'\}\big\}, \\ C_{\hat{U}} &= \big\{\{\hat{u}, \rhd\}, \{\hat{u}, \rhd, u\}, \{\hat{u}, \lhd\}, \{\hat{u}, \lhd, u\} \mid u \in U \backslash \{v', w'\}\big\}, \\ C_{\bar{v}} &= \big\{\{\bar{v}, \bar{\bar{v}}\}, \{\bar{v}, \lhd\}, \{\bar{v}, \lhd, \rhd\}, \{\bar{v}, \lhd, \rhd, v\} \mid v \in V \backslash \{v'\},\ \deg(v) = 3\big\}, \\ C_{\bar{w}} &= \big\{\{\bar{w}, \bar{\bar{w}}\}, \{\bar{w}, w\} \mid w \in W \backslash \{w'\},\ \deg(w) = 3\big\}, \\ C_{\bar{v}'} &= \big\{\{\bar{v}', \bar{\bar{v}}'\}, \{\bar{v}', \lhd\}, \{\bar{v}', \lhd, \rhd\}, \{\bar{v}', \lhd, \rhd, v'\} \mid \deg(v') = 2\big\},\ \text{and} \\ C_{\bar{w}'} &= \big\{\{\bar{w}', \bar{\bar{w}}'\}, \{\bar{w}', w'\} \mid \deg(w') = 2\big\}\,. \end{aligned}$$

This reduction yields $4|U| + |E'| + 6$ genes and at most $8|V| + 6|W| + 6|E'| + 6$ clusters. Since $|E'| \in \mathcal{O}(|U|)$, we have both $|\mathcal{G}_N| \in \mathcal{O}(|U|)$ and $|C| \in \mathcal{O}(|U|)$.

Edge strings. Due to C_E, for each edge $e = \{v, w\}$, any proper gene order g has to contain the common intervals $\{e,v\}, \{e,v,w\}, \{e,v,w,\rhd\}, \{e,v,w,\rhd,\lhd\}, \{e,w\}$ and $\{e,w,v\}$. Since $m(e) = 1$, all occurrences of these clusters in g have to overlap in e. Hence, for each edge e, g has to contain exactly one of the following eight substrings, which we call *edge strings* of e:

$$\begin{array}{ll} (\lhd, w, e, v, \rhd), & (\lhd, \rhd, w, e, v), \\ (\lhd, v, e, w, \rhd), & (\lhd, \rhd, v, e, w), \\ (\rhd, w, e, v, \lhd), & (w, e, v, \rhd, \lhd), \\ (\rhd, v, e, w, \lhd), & (v, e, w, \rhd, \lhd)\,. \end{array}$$

Cap strings. Due to C_v, any proper gene order g has to contain the common intervals $\{\bar{s}_v, \bar{\bar{s}}_v\}, \{\bar{s}_v, \hat{v}'\}, \{\bar{s}_v, \hat{v}', \rhd\}$ and $\{\bar{s}_v, \hat{v}', \rhd, v'\}$. Since $m(\hat{v}') = 1$, all occurrences of these clusters in g have to overlap in $\hat{v}'$. Hence, g has to contain exactly one of the following two substrings, which we call *cap strings* of v':

$$(\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v'), (v', \rhd, \hat{v}', \bar{s}_v, \bar{\bar{s}}_v) .$$

Analogously, due to C_w, any proper gene order g has to contain the common intervals $\{\bar{s}_w, \bar{\bar{s}}_w\}, \{\bar{s}_w, \hat{w}'\}, \{\bar{s}_w, \hat{w}', \lhd\}$ and $\{\bar{s}_w, \hat{w}', \lhd, w'\}$. Since $m(\hat{w}') = 1$, all occurrences of these clusters in g have to overlap in $\hat{w}'$. Hence, g has to contain exactly one of the following two substrings, which we call *cap strings* of w':

$$(\bar{\bar{s}}_w, \bar{s}_w, \hat{w}', \lhd, w'), (w', \lhd, \hat{w}', \bar{s}_w, \bar{\bar{s}}_w) .$$

Internal node strings. Due to $C_{\hat{U}}$, for each node $u \in U \backslash \{v', w'\}$, any proper gene order g has to contain the common intervals $\{\hat{u}, \rhd\}, \{\hat{u}, \rhd, u\}, \{\hat{u}, \lhd\}$ and $\{\hat{u}, \lhd, u\}$. Since $m(\hat{u}) = 1$, all occurrences of these clusters in g have to overlap in $\hat{u}$. Hence, for each node u, g has to contain exactly one of the following two substrings, which we call *internal node strings* of u:

$$(u, \lhd, \hat{u}, \rhd, u), (u, \rhd, \hat{u}, \lhd, u) .$$

External node strings. Due to $C_{\bar{v}}$ and $C_{\bar{v}'}$, for each node $v \in V \backslash \{v'\}$ with degree three and for $v = v'$ if $\deg(v') = 2$, any proper gene order g has to contain the common intervals $\{\bar{v}, \bar{\bar{v}}\}, \{\bar{v}, \lhd\}, \{\bar{v}, \lhd, \rhd\}$ and $\{\bar{v}, \lhd, \rhd, v\}$. Since $m(\bar{v}) = 1$ for all considered v, all occurrences of these clusters in g have to overlap in $\bar{v}$. Hence, for each considered node v, g has to contain exactly one of the following two substrings, which we call *external node strings* of v:

$$(\bar{\bar{v}}, \bar{v}, \lhd, \rhd, v), (v, \rhd, \lhd, \bar{v}, \bar{\bar{v}}) .$$

Analogously, due to $C_{\bar{w}}$ and $C_{\bar{w}'}$, for each node $w \in W \backslash \{w'\}$ with degree three and for $w = w'$ if $\deg(w') = 2$, any proper gene order g has to contain the common intervals $\{\bar{w}, \bar{\bar{w}}\}$ and $\{\bar{w}, w\}$. Since $m(\bar{w}) = 1$ for all considered w, all occurrences of these clusters in g have to overlap in $\bar{w}$. Hence, for each considered node w, g has to contain exactly one of the following two substrings, which we call *external node strings* of w:

$$(\bar{\bar{w}}, \bar{w}, w), (w, \bar{w}, \bar{\bar{w}}) .$$

We will prove the equivalence of the 3-bipartite Hamiltonian path problem to the consistency problem for common intervals on sequences in two steps.

1. **Forward direction.** Firstly, we show: If G' contains a Hamiltonian path from v' to w', then there is a gene order over $\mathcal{G}_N$ containing all common intervals in C and each element a at most $m(a)$ times. In this step, the notions of the above sets of strings will become evident.
2. **Backward direction.** Then, we show the opposite implication: If there is a gene order over $\mathcal{G}_N$ containing all common intervals in C and each element a at most $m(a)$ times, then G' contains a Hamiltonian path from v' to w'.

1. Forward Direction

Let $p = (v', u_1, u_2, \dots, u_{|U|-2}, w')$ be a Hamiltonian path in G. Further, without loss of generality, let $e_1 = \{v', u_1\}$, $e_{|U|-1} = \{u_{|U|-2}, w'\}$ and $e_i = \{u_{i-1}, u_i\}$ for $1 < i < |U| - 1$ be the corresponding edges (otherwise, we rename the edges). Let $\bar{E}$ be the set of all edges not part of the path. Obviously, p has to contain all edges $\{v, w\}$ with $\deg(v) = 2$ (or $\deg(v) = 1$ if $v = v'$, respectively) or $\deg(w) = 2$ (or $\deg(w) = 1$ if $w = w'$, respectively). Hence, for all edges $\{u, v\} \in \bar{E}$, we have $\deg(v) = 3$ (or $\deg(v) = 2$ if $v = v'$, respectively) and $\deg(w) = 3$ (or $\deg(w) = 2$ if $w = w'$, respectively). We construct the following strings:

$$S = (\overbrace{\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \mathrlap{\underbrace{}_{\text{edge string}}}\rhd, v'}^{\text{cap string}}, e_1, \overbrace{u_1, \lhd, \hat{u}_1, \mathrlap{\underbrace{}_{\text{edge string}}}\rhd, u_1}^{\text{internal node string}}, e_2, u_2, \lhd, \dots, e_{|U|-1}, \overbrace{w', \lhd, \hat{w}', \lhd, \bar{s}_w, \bar{\bar{s}}_w}^{\text{cap string}})\,;$$

$$E_e = (\overbrace{\bar{\bar{v}}, \bar{v}, \mathrlap{\underbrace{}_{\text{edge string}}}\lhd, \rhd, v}^{\text{external node string}}, e, \overbrace{w, \bar{w}, \bar{\bar{w}}}^{\text{external node string}}) \text{ for each } e = \{v, w\} \in \bar{E}\,.$$

One can easily check that the above strings posess all common intervals in C: For each edge, exactly one edge string is included and, for each node, exactly one inner node string (or cap string, respectively) is included. Further, for each node $u \notin \{v', w'\}$ with $\deg(u) = 3$ and each node $u \in \{v', w'\}$ with $\deg(u) = 2$, there has to be exactly one edge in $\bar{E}$ containing u. Thus, a corresponding external node string is included as well. As discussed above, the edge, cap, internal node and external node strings comprise all common intervals in C.

Moreover, the strings constructed above contain every $a \in \mathcal{G}_N$ at most $m(a)$ times: Each edge e, and thus the symbol e itself, is used exactly once. The symbols $\rhd$ and $\lhd$ are used by each edge string and there are no further occurrences. Hence, these symbols are included exactly $|E'|$ times. Each node $u \notin \{v', w'\}$ with $\deg(u) = 2$ is only used in the path and the corresponding internal node string overlaps in both occurrences of u. And each node $u \in \{v', w'\}$ with $\deg(u) = 1$ is only used in the path and the corresponding cap string overlaps in the occurrence of u. For any other node u, there is an external node string overlapping in u with an edge in $\bar{E}$. The other way around, each occurrence of u in an edge string overlaps either with an internal node, external node, or cap string. In any case, the restriction $m(u)$ is satisfied. Each other symbol a is used by exactly one cap, internal node or external node string and thus meets $m(a) = 1$.

Since the constructed strings satisfy all requirements of C and m, any concatenation of these strings is a proper gene order.

2. Backward Direction

Overlaps of edge strings. Consider two edges $e_1 = \{v_1, w_1\} \neq e_2 = \{v_2, w_2\}$. Assume that a substring s of g contains their corresponding edge strings overlapping in $\rhd$ or $\lhd$.

Without loss of generality, we only consider those cases in which the edge string of e_1 precedes the edge string of e_2 (otherwise, we rename e_1 and e_2). The only possibilities are:

$$\begin{array}{ll}
(w_1, e_1, \boldsymbol{v_1}, \rhd, \lhd, \rhd, w_2, e_2, v_2), & (v_1, e_1, \boldsymbol{w_1}, \rhd, \lhd, \rhd, w_2, e_2, v_2), \\
(w_1, e_1, \boldsymbol{v_1}, \rhd, \lhd, \rhd, v_2, e_2, w_2), & (v_1, e_1, \boldsymbol{w_1}, \rhd, \lhd, \rhd, v_2, e_2, w_2), \\
(w_1, e_1, \boldsymbol{v_1}, \rhd, \lhd, w_2, e_2, v_2, \rhd), & (v_1, e_1, \boldsymbol{w_1}, \rhd, \lhd, w_2, e_2, v_2, \rhd), \\
(w_1, e_1, \boldsymbol{v_1}, \rhd, \lhd, v_2, e_2, w_2, \rhd), & (v_1, e_1, \boldsymbol{w_1}, \rhd, \lhd, v_2, e_2, w_2, \rhd), \\
(\lhd, w_1, e_1, \boldsymbol{v_1}, \rhd, w_2, e_2, v_2, \lhd), & (\lhd, v_1, e_1, \boldsymbol{w_1}, \rhd, w_2, e_2, v_2, \lhd), \\
(\lhd, w_1, e_1, \boldsymbol{v_1}, \rhd, v_2, e_2, w_2, \lhd), & (\lhd, v_1, e_1, \boldsymbol{w_1}, \rhd, v_2, e_2, w_2, \lhd), \\
(\rhd, w_1, e_1, \boldsymbol{v_1}, \lhd, w_2, e_2, v_2, \rhd), & (\rhd, v_1, e_1, \boldsymbol{w_1}, \lhd, w_2, e_2, v_2, \rhd), \\
(\rhd, w_1, e_1, \boldsymbol{v_1}, \lhd, v_2, e_2, w_2, \rhd), & (\rhd, v_1, e_1, \boldsymbol{w_1}, \lhd, v_2, e_2, w_2, \rhd) .
\end{array}$$

If g contains a substring s as listed in the left column, we set $u_1 := v_1$. Otherwise, we set $u_1 := w_1$. The corresponding elements are highlighted in the above list. We distinguish between the following cases.

Case (1): $u_1 \notin \{v', w'\}$ and $\deg(u_1) = 2$
The string g has to contain one of the internal node strings of u_1. Since $m(u_1) = 2$, this has to overlap with s in u_1. Such an overlap is not possible, which reveals a contradiction to the assumption.

Case (2): $u_1 \in \{v', w'\}$ and $\deg(u_1) = 1$
The string g has to contain one of the cap strings of u_1. Since $m(u_1) = 1$, this has to overlap with s in u_1. Such an overlap is not possible, which reveals a contradiction to the assumption.

Case (3): $u_1 \notin \{v', w'\}$ and $\deg(u_1) = 3$
Let e_1, $e_3 = \{u_1, u_3\}$ and $e_4 = \{u_1, u_4\}$ be the three edges incident to u_1. In this case, g has to contain one of the edge strings of e_3 and e_4 each, one of the internal node strings of u_1, and one of the external node strings of u_1. None of these strings can overlap with s in u_1. Since $m(u_1) = 3$, g can contain at most two further occurrences of u_1. That implies that the internal node string $(u_1, \lhd, \hat{u}_1, \rhd, u_1)$ or $(u_1, \rhd, \hat{u}_1, \lhd, u_1)$ has to overlap in both occurrences of u_1. On each side, it can only overlap with either one edge string or an external node string. In any case, one of the three required substrings remains and cannot overlap with any u_1 in g. This conflicts $m(u_1) = 3$ and thus contradicts the assumption. Note that this argumentation also holds in the cases $e_2 = e_3$ or $e_2 = e_4$.

Case (4): $u_1 \in \{v', w'\}$ and $\deg(u_1) = 2$
Let e_1 and $e_3 = \{u_1, u_3\}$ be the two edges incident to u_1. In this final case, g has to contain one of the edge strings of e_3, one of the cap strings of u_1, and one of the external node strings of u_1. None of these strings can overlap with s in u_1. Since $m(u_1) = 2$, g can contain at most one further occurrence of u_1. But only two of the three required strings can overlap in u_1. In any case, one of the strings remains and cannot overlap with any u_1 in g. This conflicts $m(u_1) = 2$ and thus contradicts the assumption. This argumentation also holds in the case $e_2 = e_3$.

In all cases, the assumption that the edge strings of two edges overlap in $\rhd$ or $\lhd$ is violated. Since $m(\rhd) = m(\lhd) = |E'|$, the occurrences of $\rhd$ or $\lhd$ in edge strings are the only occurrences in g. That implies that all cap strings, and internal and external node strings have to overlap in $\rhd$ and $\lhd$ with some edge string.

Path extension. Consider that g contains a substring $s = (e, u, \lhd)$ for some edge e and some node $u \notin \{v', w'\}$, $u \in e$. By definition of the problem statement, the degree of u can be either two or three.

Case (1): $\deg(u) = 2$

Let $e_1 = \{u, u_1\}$ be the second edge incident to u. Then, g has to contain one of the edge strings of e_1 and one of the inner node stings of u. Since $m(u) = 2$, the occurrence of the inner node string has to overlap in one u with the considered subsequence s and in the other u with the edge string of e_1. The only possibility is $(e, u, \lhd, \hat{u}, \rhd, u, e_1, u_1, \lhd)$.

Case (2): $\deg(u) = 3$

Let $e_1 = \{u, u_1\}$ and $e_2 = \{u, u_2\}$ be the other edges incident to u. Then, g has to contain one of the edge strings of e_1, one of the edge strings of e_2, one of the inner node stings of u, and one of the external node strings of u. The inner node string has to overlap in $\rhd$ and $\lhd$ with some edge strings. Assume that g contains a substring s' containing an inner node string of u overlapping with edge strings of e_1 and e_2. Since no external node string can overlap with s or s' in u, this would imply four occurrences of u in g, which conflicts $m(u) = 3$ and thus contradicts the assumption. Hence, the internal node string has to overlap with s in one u und with one edge string of e_1 or e_2 in the other u. Without loss of generality, we assume an overlap with e_1 (otherwise, we rename e_1 and e_2), which implies that s has to occur in the context $(e, u, \lhd, \hat{u}, \rhd, u, e_1, u_1, \lhd)$.

In conclusion, whenever g contains a substring $s = (e, u, \lhd)$ for some edge e and some node $u \notin \{v', w'\}$, $u \in e$, there has to be a node u_1 which is connected to u by an edge e_1 and s has to be embedded in the substring $(e, u, \lhd, \hat{u}, \rhd, u, e_1, u_1, \lhd)$. Analogously, whenever g contains a subsequence $s = (\rhd, u, e)$ for some edge e and some node $u \notin \{v', w'\}$, $u \in e$, there has to be a node u_1 which is connected to u by an edge e_1 and s has to be embedded in the context $(\rhd, u_1, e_1, u, \lhd, \hat{u}, \rhd, u, e)$.

Path capping. Consider that g contains a substring $s = (e, w', \lhd)$ for some edge e with $w' \in e$. By definition of the problem statement, the degree of w' can be either one or two.

Case (1): $\deg(w') = 1$

Then, g has to contain one of the cap strings of w'. Since $m(w') = 1$, the occurrence of the cap string has to overlap in w' with the considered subsequence s. The only possibility is $(e, w', \lhd, \hat{w}', \bar{s}_w, \bar{\bar{s}}_w)$.

Case (2): $\deg(w') = 2$

Let $e_1 = \{w', v_1\}$ be the other edge incident to w'. Then, g has to contain one of the edge strings of e_1, one of the cap strings of w', and one of the external node strings of w'. The cap string has to overlap in $\lhd$ with any edge string. Assume that g contains a substring s' containing a cap string of w' that overlaps with an edge string of e_1. Since no external node string can overlap with s or s' in u, this would imply three occurrences of u in g, which conflicts $m(u) = 2$ and thus contradicts the assumption. Hence, the cap string has to overlap with s, which implies that s has to occur in the context $(e, w', \lhd, \hat{w}', \lhd, \bar{s}_w, \bar{\bar{s}}_w)$.

In conclusion, whenever g contains a substring $s = (e, w', \lhd)$ for some edge e, $w' \in e$, s has to be embedded in the substring $(e, w', \lhd, \widehat{w'}, \lhd, \bar{s}_w, \bar{\bar{s}}_w)$. Analogously, whenever g contains a substring $s = (\rhd, v', e)$ for some edge e, $v' \in e$, s has to be embedded in the substring $(\bar{\bar{s}}_v, \bar{s}_v, \widehat{v'}, \rhd, v')$.

Existence of a Hamiltonian path. Any proper gene order has to contain a cap string of v', i.e. $s = (\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v')$ or its reverse. Without loss of generality, we assume that g contains the forward version of s (otherwise, we reverse g). This substring has to overlap in $\rhd$ with one edge sequence of an edge. The only possibility is $(\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v', e_1, u_1, \lhd)$ for some edge $e_1 = \{v', u_1\}$. Since we excluded the edge $\{v', w'\}$, we have $u_1 \neq w'$. Hence, following the paragraph *path extension*, there has to be an incident edge $e_2 = \{u_1, u_2\}$ and the considered substring has to be followed by $(\hat{u}_1, \rhd, u_1, e_2, u_2, \lhd)$. This last extension argument can be repeated until we get $(\dots, e_k, w', \lhd)$, which has to be capped by $(\hat{w}', \lhd, \bar{s}_w, \bar{\bar{s}}_w)$, following the paragraph *path capping*. Eventually, we showed that g has to contain the string

$$S = (\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v', e_1, u_1, \lhd, \hat{u}_1, \rhd, u_1, e_2, u_2, \lhd, \dots, e_k, w', \lhd, \hat{w}', \lhd, \bar{s}_w, \bar{\bar{s}}_w)$$

and G' has to comprise the corresponding path $p = (v', u_1, u_2, \dots, w')$. Since $m(\hat{u}) = 1$ for all $u \in U$, S cannot contain any internal node string more than once and the corresponding path passes through no node more than once.

We now show by contradiction that S contains *all* internal node strings. Assume a node $u \in U$ is not contained in S. String g has to contain the internal node string $(u, \lhd, \hat{u}, \rhd, u)$ or $(u, \rhd, \hat{u}, \lhd, u)$. Without loss of generality, we assume that g contains the first variant (otherwise, we reverse g). Any internal node string has to overlap with some edge string in $\rhd$ and $\lhd$. In this case, the only possibility is $(\rhd, u_1, e_1, u, \lhd, \hat{u}, \rhd, u, e_2, u_2, \lhd)$. Following the above extension argument, this string has to be embedded in a string S' that is capped with $\bar{\bar{s}}_v$ on the left side and with $\bar{\bar{s}}_w$ on the right side. Due to $m(\bar{\bar{s}}_v) = m(\bar{\bar{s}}_w) = 1$, we get $S' = S$, which contradicts the assumption. In conclusion, S contains all internal node strings and thus corresponds to a path in G that passes all nodes, which in fact is the desired Hamiltonian path.

□

A.3 Consistency Problem for Framed Common Intervals on Sequences

Lemma 6. *The consistency problem (Problem 23 on page 77) for the gene cluster model of framed common intervals on sequences (Model 9 on page 89) is in the complexity class NP.*

Proof. The argumentation is analogous to the proof of Lemma 4.

The input I for an instance of the problem consists of the number of genes N, a multiplicity function m and a set of gene clusters C. A framed common interval $c = [a\, I_c\, b]$ can be represented by a list of $2 + |I|$ integers. Let $K := \sum_{c \in C} (2 + |I_c|)$ be the total size of all framed common intervals in C. Further, we need at least N integers to represent the values of m. Altogether, this yields a total input size of $|I| := 1 + K + N$ integers.

The solution would be a genome g that contains all framed common intervals and each gene a not more than $m(a)$ times. If such a genome g exists, there is always a subsequence g' of g of length $\mathcal{O}(K)$ that is also a proper solution. (The longest possible such subsequence would be a concatenation of all nested common intervals.) We call such a genome g' a *certificate*.

To show that the considered problem is in NP, we now provide an algorithm that can verify any correct certificate g' in time polynomial in the input size $|I|$.

For each framed common interval $c = [a\, I_c\, b]$, we proceed as follows. Firstly, we scan g' for substrings beginning with a and ending with b, or beginning with $-b$ and ending with $-a$. For each of the $\mathcal{O}\left(K^2\right)$ matches, we examine the character set of the match. To this end, for each element of the considered substring, we verify whether it is allowed to occur. This takes $\mathcal{O}(K|I_c|)$ steps for each substring and thus $\mathcal{O}\left(K^3|I_c|\right)$ per cluster. Altogether, we verify any correct certificate in $\mathcal{O}\left(K^4\right)$ steps, which is polynomial in the input size:

$$K^4 \leq (1 + K + N)^4 = |I|^4 .$$

□

Lemma 7. *The consistency problem (Problem 23) for the gene cluster model of framed common intervals on sequences (Model 9) is NP-hard.*

Proof. The consistency problem for the gene cluster model of common intervals on sequences is NP-complete by Theorem 5. Now, we show that the latter problem is reducible to the consistency problem for the gene cluster model of framed common intervals on sequences in polynomial time.

Let the set of genes $\mathcal{G}_N$, the set of common intervals C and the multiplicity function m define an instance of the consistency problem for common intervals on sequences. Then we reduce this instance to a problem instance of the consistency problem for framed common intervals on sequences in polynomial time as follows.

- $\mathcal{G}_{N+2} := \mathcal{G}_N \cup \{x, y\}$ with $x := N + 1$ and $y := N + 2$,
- $C' := \big\{[x\, I_c\, y] \mid c \in C\big\}$ with $I_c := c \cup \{x, y\}$,
- $m'(a) = \begin{cases} \sum_{a \in \mathcal{G}_N} m(a), & \text{if } a \in \{x, y\}, \\ m(a), & \text{otherwise}. \end{cases}$

Firstly, we show: If C is consistent with respect to m, then C' is consistent with respect to m'. Let $g = (a_1, a_2, \ldots, a_{|g|})$ be a valid sequence over $\mathcal{G}_N$. We construct the following signed sequence over $\mathcal{G}_{N+2}$:

$$g' = (x, a_1, y, x, a_2, y, x, \ldots, y, x, a_{|g|}, y) .$$

This sequence contains $|g|$ occurrences of x and y, and thus satisfies the requirements of m'. Moreover, for each common interval c, g contains a substring $(a_i, \dots, a_j)$ with character set c and g' contains a corresponding substring $(x, a_i, y, x, \dots, y, x, a_j, y)$ which in turn contains the framed common interval $[x\, I_c\, y]$ with $I_c = c \cup \{x, y\}$. Hence, g' contains all clusters in C'.

Now, we show the opposite implication: If C' is consistent with respect to m', then C is consistent with respect to m. Let $g' = (a_1, a_2, \dots, a_{|g'|})$ be a valid sequence over $\mathcal{G}_{N+2}$. We construct the following unsigned sequence over $\mathcal{G}_N$:

$$g := \bigotimes_{i=1}^{|g'|} \left\{ \begin{array}{ll} |a_i|, & \text{if } |a_i| \notin \{x, y\}, \\ \epsilon, & \text{otherwise}; \end{array} \right\}$$

where $\otimes$ denotes the concatenation of strings and ϵ is the empty string.

Since, in g, all occurrences of x are removed, g is a sequence over $\mathcal{G}_N$ satisfying the requirements of m. Moreover, for each framed common interval $[x\, I_c\, y]$, g' contains a substring with character set I_c. Thus, g contains a corresponding substring containing character set $I_c \backslash \{x, y\} = c$. Hence, g contains all clusters in C. □

Theorem 6. *The consistency problem (Problem 23) for the gene cluster model of framed common intervals on sequences (Model 10) is NP-complete.*

Proof. Theorem 6 follows directly from Lemmas 6 and 7. □

A.4 Consistency Problem for Nested Common Intervals on Sequences

Lemma 8. *The consistency problem (Problem 23 on page 77) for the gene cluster model of nested common intervals on sequences (Model 10 on page 90) is in the complexity class NP.*

Proof. The argumentation is analog to the proof of Lemma 4.

The input I for an instance of the consistency problem comprises the number of genes N, a multiplicity function m and a set of gene clusters C. A nested common interval $c = (\cdots(\{a_1, a_2\}, a_3)\cdots a_{|c|})$ can be represented by a list of $|c|$ integers. Let $K := \sum_{c \in C} |c|$ be the total size of all nested common intervals in C. Further, we need at least N integers to represent the values of m. Altogether, this yields a total input size of $|I| := 1 + K + N$ integers.

The solution would be a genome g that contains all nested common intervals and each gene a not more than $m(a)$ times. If such a genome g exists, there is always a subsequence g' of g of length $\mathcal{O}(K)$ that is also a proper solution. (The longest possible such subsequence would be a concatenation of all nested common intervals.) We call such a genome g' a *certificate*.

To show that the considered problem is in NP, we now provide an algorithm that can verify any correct certificate g' in time polynomial in the input size $|I|$.

For each nested common interval $c = (\cdots(\{a_1, a_2\}, a_3)\cdots a_{|c|})$, we proceed as follows. Firstly, we scan g' for occurrences of the pair $\{a_1, a_2\}$ in $\mathcal{O}(K)$ steps. For each of the $\mathcal{O}(K)$ matches, we examine the neighborhood of the match for the existence of the complete cluster. To this end, we use a bit array of size N to store the character set of the considered sub-cluster. At the beginning, only the entries for a_1 and a_2 are set to one. For each i from $i = 3$ to $|c|$, we try to extend the match to the left as long as we observe only genes from the character set $\{a_1, \ldots, a_{i-1}\}$ until we find a_i. If this is not possible, we try to extend the match to the right analogously. Then, we set the entry for a_i in the array to one, increase i and proceed with the next iteration. Since g' contains the cluster, at least one of the initial pair matches is extendible properly. Altogether, we verify any correct certificate in $\mathcal{O}\big(|C|(K + K(N + K^2))\big) = \mathcal{O}\big(|C|(KN + K^3)\big)$ steps, which is polynomial in the input size:

$$\begin{aligned} |C|(KN + K^3) &\leq K(KN + K^3) \\ &= K^2N + K^4 \\ &\leq |I|^4 . \end{aligned}$$

□

Lemma 9. *The consistency problem (Problem 23) for the gene cluster model of nested common intervals on sequences (Model 10) is NP-hard.*

Proof. The argumentation is analogous to the the proof of the NP-hardness of the consistency problem for common intervals on sequences (Lemma 5). In the reduction step, we use the same set of genes and the same multiplicities. But in this case, the set C of gene clusters is not composed of common intervals. Instead we define a corresponding set of nested common intervals:

$C := C_E \cup C_v \cup C_w \cup C_{\hat{U}} \cup C_{\bar{v}} \cup C_{\bar{w}} \cup C_{\bar{v}'} \cup C_{\bar{w}'}$ with

$$\begin{aligned} C_E &= \big\{((((\{e, v\}, w), \triangleright), \triangleleft), (\{e, w\}, v) \mid e = \{v, w\} \in E'\big\}, \\ C_v &= \big\{\{\bar{s}_v, \bar{\bar{v}}_v\}, (((\{\bar{s}_v, \hat{v}'\}, \triangleright), v')\big\}, \\ C_w &= \big\{\{\bar{s}_w, \bar{\bar{w}}_w\}, (((\{\bar{s}_w, \hat{w}'\}, \triangleleft), w')\big\}, \\ C_{\hat{U}} &= \big\{(\{\hat{u}, \triangleright\}, u), (\{\hat{u}, \triangleleft\}, u) \mid u \in U \backslash \{v', w'\}\big\}, \\ C_{\bar{v}} &= \big\{\{\bar{v}, \bar{\bar{v}}\}, (((\{\bar{v}, \triangleleft\}, \triangleright), v) \mid v \in V \backslash \{v'\},\ \deg(v) = 3\big\}, \\ C_{\bar{w}} &= \big\{\{\bar{w}, \bar{\bar{w}}\}, \{\bar{w}, w\} \mid w \in W \backslash \{w'\},\ \deg(w) = 3\big\}, \\ C_{\bar{v}'} &= \big\{\{\bar{v}', \bar{\bar{v}}'\}, (((\{\bar{v}', \triangleleft\}, \triangleright), v') \mid \deg(v') = 2\big\}, \text{ and} \\ C_{\bar{w}'} &= \big\{\{\bar{w}', \bar{\bar{w}}'\}, \{\bar{w}', w'\} \mid \deg(w') = 2\big\} . \end{aligned}$$

One can easily see that these nested common intervals induce the same edge, cap, internal node and external node strings. Since the rest of the proof of Lemma 5 is solely based on these strings and the multiplicity threshold function $m(a)$ and both are used equally here, the same argumentation applies in this case: G' contains a Hamiltonian path from v' to w' if and only if there is a gene order over $\mathcal{G}_N$ containing all nested common intervals in C and each element a at most $m(a)$ times. □

Theorem 7. *The consistency problem (Problem 23) for the gene cluster model of nested common intervals on sequences (Model 10) is NP-complete.*

Proof. Theorem 7 follows directly from Lemmas 8 and 9. □

Overview of Important Terms used in the Proof of Lemma 5

Objects:

$$\begin{aligned}\mathcal{G}_N := U \cup E' \cup \{\lhd, \rhd, \bar{s}_v, \bar{s}_w, \bar{\bar{s}}_v, \bar{\bar{s}}_w\} \\ \cup\ \hat{U} \text{ with } \hat{U} = \{\hat{u} \mid u \in U\} \\ \cup\ \bar{U} \text{ with } \bar{U} = \{\bar{u}, \bar{\bar{u}} \mid u \in U\}\end{aligned}$$

Multiplicities:

$$m(a) := \begin{cases} \deg(a), & \text{if } a \in U, \\ 1, & \text{if } a \in E', \\ |E'|, & \text{if } a \in \{\lhd, \rhd\}\,, \\ 1, & \text{if } a \in \{\bar{s}_v, \bar{s}_w, \bar{\bar{s}}_v, \bar{\bar{s}}_w\}, \\ 1, & \text{if } a \in \hat{U}, \\ \deg(a) - 2, & \text{if } a = \bar{u} \in \bar{U} \backslash \{\bar{v}', \bar{w}'\}, \\ \deg(a) - 1, & \text{if } a = \bar{u} \in \{\bar{v}', \bar{w}'\}\,. \end{cases}$$

Edge strings:

$$\begin{array}{ll} (\lhd, w, e, v, \rhd) & (\lhd, \rhd, w, e, v) \\ (\lhd, v, e, w, \rhd) & (\lhd, \rhd, v, e, w) \\ (\rhd, w, e, v, \lhd) & (w, e, v, \rhd, \lhd) \\ (\rhd, v, e, w, \lhd) & (v, e, w, \rhd, \lhd) \end{array}$$

Cap strings:

$$\begin{array}{ll} (\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v') & (v', \rhd, \hat{v}', \bar{s}_v, \bar{\bar{s}}_v) \\ (\bar{\bar{s}}_w, \bar{s}_w, \hat{w}', \lhd, w') & (w', \lhd, \hat{w}', \bar{s}_w, \bar{\bar{s}}_w) \end{array}$$

Internal node strings:

$$(u, \lhd, \hat{u}, \rhd, u) \qquad (u, \rhd, \hat{u}, \lhd, u)$$

External node strings:

$$\begin{array}{ll} (\bar{\bar{v}}, \bar{v}, \lhd, \rhd, v) & (v, \rhd, \lhd, \bar{v}, \bar{\bar{v}}) \\ (\bar{\bar{w}}, \bar{w}, w) & (w, \bar{w}, \bar{\bar{w}}) \end{array}$$

Strings composition:

$$S = (\mathrlap{\overbrace{\phantom{\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \rhd, v'}}^{\text{cap string}}}\bar{\bar{s}}_v, \bar{s}_v, \hat{v}', \underbrace{\rhd, v', e_1, \mathrlap{\overbrace{\phantom{u_1, \lhd, \hat{u}_1, \rhd, u_1}}^{\text{internal node string}}}u_1, \lhd}_{\text{edge string}}, \hat{u}_1, \underbrace{\rhd, u_1, e_2, u_2, \lhd}_{\text{edge string}}, \ldots, e_{2n}, \overbrace{w', \lhd, \hat{w}', \lhd, \bar{s}_w, \bar{\bar{s}}_w}^{\text{cap string}})$$

$$E_e = (\overbrace{\bar{\bar{v}}, \bar{v}, \mathrlap{\underbrace{}_{\text{edge string}}}\lhd, \rhd, v}^{\text{external node string}}, e, \overbrace{w, \bar{w}, \bar{\bar{w}}}^{\text{external node string}}) \text{ for each } e = \{v, w\} \in \bar{E}$$