Polynomial Supertree Methods in Phylogenomics

Algorithms, Simulations and Software

Dissertation

zur Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)

vorgelegt dem Rat der Fakultät für Mathematik und Informatik

der Friedrich-Schiller-Universität Jena

von Dipl.-Inf. Malte Brinkmeyer

geboren am 09. April 1980 in Dortmund

Gutachter:

1. **Prof. Dr. Sebastian Böcker, Friedrich-Schiller-Universität Jena**
2. **Prof. Dr. Arndt von Haeseler, Universität Wien**

Tag der öffentlichen Verteidigung: 29.01.2013

Abstract

One of the objectives in modern biology, especially phylogenetics, is to build larger clades of the *Tree of Life*. Large-scale phylogenetic analysis involves several serious challenges. The aim of this thesis is to contribute to some of the open problems in this context:

Perhaps most important in general is the design of fast and efficient algorithms to reconstruct large phylogenetic trees. In order to compare the usefulness and accuracy of alternative approaches under various conditions, simulation studies are needed. Another requirement towards estimating parts of the tree of life is a sometimes neglected, but nevertheless quite important factor: the ability of user friendly software supporting the steps of a phylogenetic analysis.

In computational phylogenetics, supertree methods provide a way to reconstruct larger clades of the Tree of Life. The supertree problem can be formalized in different ways to cope with contradictory information in the input. In particular, there exist methods based on encoding the input trees in a matrix and methods based on finding minimum cuts in some graph. Matrix representation methods compute supertrees of superior quality, but the underlying optimization problems are computationally hard. In contrast, graph-based methods have polynomial running time, but the resulting supertrees are inferior in quality. In this thesis, we present a novel polynomial time approach for the computation of supertrees called FlipCut supertree. Our method combines the computation of minimum cuts from graph-based methods with a matrix representation method, namely Minimum Flip Supertrees. Here, the input trees are encoded in a 0/1/?-matrix. We present a heuristic to search for a minimum set of 0/1-flips such that the resulting matrix admits a directed perfect phylogeny. In contrast to other polynomial time approaches, our results can be interpreted in the sense that we we try to minimize a global objective function, namely the number of flips in the input matrix. We extend our approach by using edge weights to weight the columns of the 0/1/?-matrix.

In order to compare our new FlipCut supertree method with other recent polynomial supertree methods and matrix representation methods, we present a large scale simulation study using two different data sets. Our findings illustrate the trade-off between accuracy and running time in supertree construction, as well as the pros and cons of different supertree approaches.

Furthermore, we present EPoS, a modular software framework for phylogenetic analysis and visualization. It fills the gap between command line-based algorithmic packages and visual tools without sufficient support for computational methods.

By combining a powerful graphical user interface with a plugin system that allows simple integration of new algorithms, visualizations and data structures, we created a framework that is easy to use, to extend and that covers all important steps of a phylogenetic analysis.

Zusammenfassung

Eines der Ziele der modernen Biologie, insbesondere der molekularen Phylogenetik, ist es, größere Teile vom *Baum des Lebens* zu erstellen. Allerdings bringen umfangreiche phylogenetische Analysen einige Herausforderungen mit sich. Das Ziel dieser Arbeit ist es, zur Lösung von einem Teil der Probleme in diesem Kontext beizutragen:

Am vielleicht wichtigsten ist die Entwicklung von schnellen und genauen Algorithmen, um große phylogenetische Bäume zu rekonstruieren. Simulationsstudien werden benötigt, um diverse Leistungsaspekte von verschiedenen methodischen Ansätzen unter unterschiedlichen Bedingungen vergleichen zu können. Eine weitere Anforderung in Hinblick auf das Erstellen von Teilen des Baum des Lebens ist ein manchmal missachteter, aber nicht desto trotz wichtiger Faktor: Das Vorhandensein benutzerfreundlicher Software, die die Schritte einer phylogenetischen Analyse unterstützt.

In der Phyloinformatik bilden sogenannte Supertree Methoden einen Weg, um größere Kladen vom Baum des Lebens zu erstellen. Das Supertree Problem kann auf verschiedene Art und Weise formalisiert werden, um mit widersprüchlicher Information in den Eingabedaten umzugehen. Insbesondere gibt es Methoden, die die Eingabedaten in eine Matrix kodieren und Methoden, die auf dem Finden von minimum cuts in Graphen basieren. Matrixbasierte Methoden erstellen Supertrees mit besserer Qualität, allerdings sind die zugrunde liegenden Optimierungsprobleme NP-schwer. Im Gegensatz dazu haben graphbasierte Methoden polynomiale Laufzeiten, wobei die rekonstruierten Supertrees jedoch von geringerer Qualität sind. In dieser Arbeit präsentieren wir einen neuen Ansatz zur Berechnung von Supertrees, FlipCut. Unsere Methode vereint die Berechnung von minimum cuts von graphbasierten Methoden mit einer matrixbasierten Methode, Minimum Flip Supertrees. In dieser werden die Eingabedaten in einer 0/1/- Matrix kodiert. Wir präsentieren eine Heuristik, um nach der geringsten Menge von 0/1- Flips zu suchen, so das die entstehende Matrix eine gerichtete perfekte Phylogenie erlaubt. Im Gegensatz zu anderen Methoden mit polynomialer Laufzeit sind die Ergebnisse unserer Methode interpretierbar in dem Sinne, als das wir eine globale Optimierungsfunktion minimieren, nämlich die Anzahl der Flips in der Eingabematrix. Unser Ansatz wird erweitert, indem wir Kantengewichte benutzen, die den Gewichtungen der Spalten der 0/1 Matrix entsprechen.

Wir präsentieren eine umfangreiche Simulationsstudie auf zwei verschiedenen Datensätzen, um unsere neue Methode FlipCut Supertree mit anderen aktuellen polynominalen Supertree Methoden und matrixbasierten Methoden zu vergleichen. Unsere Ergebnisse verdeutlichen das Verhältniss von Genauigkeit und Laufzeit bei der Supertree Rekonstruktion, sowie die Vor- und Nachteile der einzelnen Ansätze.

Weiterhin stellen wir EPoS, ein modulares Softwarepaket für phylogenetische Analyse und Visualisierung, vor. EPoS schliesst die Lücke zwischen kommandozeilenbasierten, rein algorithmischen Programmen, und Programmen zur Visualisierung die oft keine ausreichende Unterstützung für Methoden mit sich bringen. Durch die Kombination einer leistungsfähigen graphischen Benutzeroberfläche mit einem Plugin-System, durch das sich neue Algorithmen, Visualisierungen und Datenstrukturen einfach integrieren lassen, haben wir ein Softwarepaket entwickelt, das einfach zu benutzen und zu erweitern ist und alle wesentlichen Schritte einer phylogenetischen Analyse abdeckt.

Acknowledgements

This thesis would not have been possible without support of many people. First and foremost, I would like to thank Prof. Dr. Sebastian Böcker for being a great supervisor, who has been extremely supportive with lots of valuable ideas and stimulating discussions. Without his ideas, comments and contributions, this thesis would not exist. A special thanks goes to Prof. Dr. Arndt von Haeseler from the University of Vienna for being my second advisor. My thanks also go to all members of the Chair Bioinformatics at the University of Jena, including Thasso Griebel, Franziska Hufsky, Florian Rasche, Imran Rauf, Kerstin Scheubert, Anke Truss, Sascha Winter and for an enjoyable working atmosphere. In particular I would like to thank Anke Truss and Sascha Winter, my office neighbors. I really enjoyed their company. Moreover, I would like to thank Thasso Griebel for his friendship throughout the years of our undergraduate study and the time in Jena. A special thank goes to Kathrin Schowtka, our secretary, who always helps not only with administrative work and for being such a nice person in general. I heartly appreciate Thasso Griebel, Anke Truss, Franziska Hufsky, Sascha Winter and Kerstin Scheubert not only for proofreading parts of this thesis, but also for their advices to improve its legibility. My parents Barbara and Erhard Brinkmeyer receive my deepest gratitude and love for everything. They always encouraged me and showed me great support. Thanks to all the people who made my life in Jena pleasant.

Contents

Preface

This thesis covers topics from bioinformatics, in particular phyloinformatics. Due to the interdisciplinary nature of this field, it is almost impossible to provide background information on all touched topics. I tried to include sufficient background to make the topics of this thesis as understandable as possible, hopefully also for someone not directly involved in the phyloinformatic community. However, I assume that the reader is familiar with basic knowledge in molecular biology as well as computer science. As usual in scientific presentations, I use the scientific 'we' instead of the personal pronoun 'I' in this thesis.

Parts of this thesis have been published in the following papers:

- T. Griebel, M. Brinkmeyer, and S. Böcker. EPoS: a modular software framework for phylogenetic analysis. *Bioinformatics*, 24(20):2399-2400, 2008.
- M. Brinkmeyer, T. Griebel, and S. Böcker. Polynomial supertree methods revisited. In *Proc. of Pattern Recognition in Bioinformatics (PRIB 2010)*, volume 6282 of *Lecture Notes Computer Science.*, pages 183-194. Springer, 2010.
- M. Brinkmeyer, T. Griebel and S. Böcker. Polynomial Supertree Methods Revisited. *Advances in Bioinformatics*,(Article ID 524182): 21 pages, 2011.
- M. Brinkmeyer, T. Griebel, and S. Böcker. FlipCut Supertrees: Towards Matrix Representation Accuracy in Polynomial Time. In *Proc. of Computing and Combinatorics Conference (COCOON 2011)*, 2011.
- M. Brinkmeyer, T. Griebel, and S. Böcker. FlipCut Supertrees: Towards Matrix Representation Accuracy in Polynomial Time. Accepted for publication in *Algorithmica* (special issue of selected papers from COCOON 2011), 2012.

Introduction

Today, all biological disciplines are united by the idea that contemporary organisms, genes and genomes share common ancestors and thus a common history. Molecular phylogenetics is a subfield of biology in which molecular characters are used to infer the historical relationship between a set of *taxa*[1], usually corresponding to species. Historical relationships are usually represented as rooted phylogenetic trees with the taxa as leaves and in which an inner node corresponds to a hypothetical last common ancestor of the taxa below this node. The knowledge of evolutionary relationships is nowadays essential to various fields of modern biology as well as other disciplines.

The growth of available sequence data for phylogenetic analysis in the last decades is overwhelming. An impressive comparison was given by Goldman and Yang [73]: the EMBL Nucleotide Sequence Database[2] grows faster than computer processing power and the rate even outpaces "Moore's Law". On the one hand, today's abundance of homologous characters that can be compared in phylogenetic analysis offers great opportunities for research projects like "Assembling the Tree of Life"[3], but on the other hand large-scale phylogenetic analysis involves several serious challenges. The aim of this thesis is to contribute to some of the open problems in this context:

Perhaps most important in general is the design of fast and efficient algorithms to reconstruct very large phylogenetic trees. In order to compare the usefulness and accuracy of alternative approaches under various conditions, simulation studies are needed. Another requirement towards estimating parts of the tree of life is a sometimes neglected, but nevertheless quite important factor; the ability of user friendly software supporting the steps of a phylogenetic analysis.

This thesis is organized as follows:

In **Chapter 1** we provide a general introduction to phylogenetics to give the reader an impression how phylogenetic trees are inferred. We outline how modern evolutionary thinking developed throughout the centuries and recapitulate fundamental concepts, data and methods used to reconstruct phylogenetic trees. Moreover, we present and discuss the two main approaches to large-scale phylogenetic inference, the supermatrix and the supertree approach, which have been handled mutually exclusive in the past: The supermatrix approach uses large matrices of primary sequence data as input whereas supertree methods use reconstructed, overlapping phylogenetic trees to reconstruct large phylogenetic trees. In this thesis, we focus on the latter approach. One promising approach for the future to continuously build ever-larger parts of the tree of life is to use supertree methods

[1] a *taxon* (plural: taxa) refers to a systematically named group of organisms, in molecular phylogenetics often represented by gene or protein sequences.

[2] `http://www.ebi.ac.uk/embl/`

[3] `http://www.phylo.org/atol/`

as part of a divide-and-conquer search strategy for large molecular supermatrices.

In **Chapter 2**, "Supertree Methods", we first provide the theoretical framework that will be used throughout the rest of this thesis. Supertree methods combine input trees with overlapping taxa sets into one large and more comprehensive tree. To gain advantage of a divide-and-conquer framework for large scale phylogenetic analysis in terms of speed and accuracy compared to conventional phylogenetic analysis, it is of particular interest to use fast and accurate supertree methods. In this chapter, we thus review several supertree methods relevant in this context and for the rest of this thesis.

In **Chapter 3**, we introduce our new supertree method, FlipCut supertree. The FlipCut algorithm tries to minimize the number of 0/1-flips in the matrix representation of the input trees, motivated by the Matrix Representation with Flipping (MRF) supertree formulation [37, 38]. Our algorithm constructs the phylogenetic tree top-down, similar to the Build algorithm [3], minimizing in each step the number of required flips. The algorithm is also closely related to that of Pe'er *et al.* [117]. For n taxa and m internal nodes in the input trees, worst-case running time is $O(mn^3)$. In contrast to MinCut supertrees [147], our results can be interpreted in the sense that we try to minimize a global objective function, namely the number of flips in the input matrix.

In **Chapter 4**, "Polynomial Supertree Methods Revisited", we present large scale simulations focusing on our new FlipCut supertree method and other recent polynomial supertree methods in comparison to the Matrix Representation with Parsimony (MRP) and the Matrix Representation with Flipping (MRF) methods.

In **Chapter 5**, we present EPoS, a modular software framework for phylogenetic analysis and visualization. It fills the gap between command line-based algorithmic packages and visual tools without sufficient support for computational methods. By combining a powerful graphical user interface with a plug-in system that allows simple integration of new algorithms, visualizations and data structures, we created a framework that is easy to use and to extend. EPoS covers all important steps of a phylogenetic analysis.

Chapter 6 concludes this thesis by recalling and summarizing our main results. Furthermore, we present an outlook on our objectives for future work.

CHAPTER 1

A General Introduction to Phylogenetics

One fundamental question for humankind is where we come from. Many theories and hypotheses suggest possible answers. For centuries, biologists have tried to detect and classify the diversity in the biological world, this effort is known as *systematics*. Modern biology assumes that the biological diversity today is based on *evolution* and that species have a *phylogeny* representing their evolutionary development. These two concepts gave rise to the science of *Phylogenetic Systematics*, where organisms are classified into groups by their phylogeny. In this chapter we present a short historical tour how phylogenetics arose as a scientific field, cover some fundamental concepts and shortly review data and methods used to reconstruct phylogenies. We focus on reconstruction algorithms relevant for understanding the rest of this thesis. A more comprehensive description of the different reconstruction methods and programs can be found in relevant books on phylogenetics (e.g. Felsenstein [59], Higgs and Attwood [89], Salemi *et al.* [138]).

1.1 Towards modern phylogenetic thinking

Already in the early ancient time, philosophers and naturalists from many cultures tried to grasp and classify the biodiversity populating the Earth.

Aristotle, for example, expressed his view of a natural order in his *scala naturae*, which classifies organisms according to there complexity of structure and function in a hierarchical "Chain of Being" or "Ladder of Life". Nevertheless, he assumed that species are constant and eternal.

In the 18th and 19th centuries, several fundamental scientific works have been published which can be seen as precursor for the modern evolutionary thought, the conception that species change over time.

The Swedish botanist Linné (1707-1778, original name Linneaus) introduced a new way to classify and name species. He proposed to group them by shared similarities into higher taxa, being: genera, orders, classes and kingdoms. He also invented the *binomial system* for naming species, where a name is composed of genus and species in latin, as in *Homo sapiens*. This system rapidly became the standard system for naming species and its basic form is still in use today. Jean-Baptiste Lamarck was one of the first to believe in some kind of evolution - although not assuming an overall ancestor of all species. In 1809, he published his *Philosophie zoologique* in which he proposed his idea that altering of one species into another results from the inheritance of adjustments, which are acquired by the parents to adapt to living environment. In 1859, Charles Darwin published his famous book *On the Origin of Species*. At that time the idea of some sort of evolution was

not new and many biologists did not believe in the notion of fixed species anymore. However, Darwin was the one who was able to explain how this evolution could have occurred. His concept of evolution by natural selection may be expressed as a very simple set of statements:

1. The individual organisms in a population vary.
2. They overproduce (if the available resources allow).
3. Natural selection favors the reproduction of those individuals that are best adapted to the environment.
4. Some of the variations are inherited to the offspring.
5. Therefore organisms evolve.

The term *phylogeny* was first used by Ernst Haeckel in 1866, to designate the history of organismal lineages as they change through time. In 1900, Gregor Mendel discovered that traits were inherited in a predictable manner. However, this appeared to be a contradiction to the natural selection concept of Darwin. In 1930, Ronald Fisher resolved this contradiction in his publication *Genetical Theory of Natural Selection* and set the foundations for the establishment of *population genetics*. In the 1940s, Oswald Avery and colleagues identified DNA as the genetic material. In 1953, James Watson and Francis Crick published the structure of DNA, which provides the physical basis for inheritance. The discovery of DNA and the invention of sequencing techniques in the following years made molecular data available as new source of information.

The ever-growing amount of molecular data available today offers a great opportunity for the field of phylogenetics, but also poses various serious challenges for standard phylogenetic reconstruction methods. Some of these challenges will be covered in this thesis.

In phylogenetics, evolutionary relationships are usually represented as *phylogenetic trees*[1], also called *phylogenies*, with a set of taxa as *leave* nodes. One distinguishes between *rooted* and *unrooted* phylogenetic trees. See Fig. 1.1 for an example. A *rooted* phylogeny has one distinguished *root* node with an in degree of zero. Each inner node represents a hypothetical last common ancestor of the taxa below this node. Branches often have additional information, such as an estimate of evolutionary time or amount that took place between the connected nodes.

1.2 Molecular phylogenetics

Relevant data for the inference of a phylogeny are obtained by examining variable *characters* in the organisms being compared. Prior to the molecular revolution, phylogenetics was founded on morphological data. Today,

[1] for a formal definition see Chapter 2, page 21

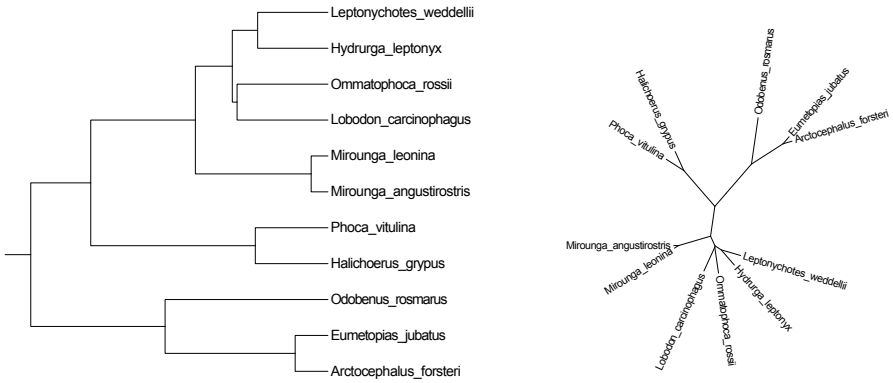


Figure 1.1: **Phylogenetic trees for mitochondrial genes (*MT-ND6*) from eleven pinniped species, obtained by a maximum likelihood (ML) analysis [88]** - The phylogenetic tree in (i) is rooted, whereas the tree in (ii) is unrooted. The branch length in both trees represent amounts of evolution between species. In the rooted case, this only holds for the length of the vertical branches. The fundamental difference between a rooted and an unrooted tree is that in a rooted tree the last common ancestor of the leaves is given and thereby the direction of evolution, whereas in an unrooted tree this information is not provided.

molecular data, especially DNA sequences, but also amino acids, codons, SINE, LINE and other, is used. In the following, we will concentrate on molecular phylogenetics. Molecular data can be considered as sequences of characters, which can take several *states* (A,C,G,T for nucleotide sites, etc).

Species trees and Gene trees

The traditional objective of a phylogenetic analysis is to represent the historical relationship between species in a *species tree*. In molecular phylogenetics, a *gene tree* is usually inferred by analyzing a gene family, that are *homologous* molecular sequences in the genome of different organisms. Homologous genes[2] have evolved from a common ancestor (for a detailed discussion of these concepts see for example [63, 101]). The important point here is that gene trees can be used to estimate species trees. As more and more genes are analyzed, conflicts between individual gene trees (or gene trees and species trees) arise because of methodological or biological reasons. Concerning methodology, the main reason for conflicting gene trees

[2]homology is further subdivided into orthology and paralogy. Orthologues genes have evolved from a common ancestor by a speciation event, whereas paralogous genes have descend from the same ancestor gene but their last common ancestor is a duplication event.

is the use of an inappropriate evolutionary model to reconstruct the gene trees. Model inadequacy has several causes, see [51] for an overview. Most important are compositional bias, heterotachy and rapidly evolving lineages. Biological macro events in genome evolution can also lead to conflicts between individual gene trees (see for example [109]). These include gene duplication and loss, horizontal gene transfer, incomplete lineage sorting, interspecific recombination and interspecific hybridization.

Multiple Sequence Alignment

Basis for a molecular phylogenetic analysis is usually a *multiple sequence alignment* (MSA) of a set S of molecular sequences. A MSA is defined by a matrix where the rows are the sequences in S, and the entries within each column have evolved from a common ancestor (positional homology). An additional gap character '-' is used to present insertions and deletions. This assures that really the same characters in all species are compared in subsequent phylogenetic analysis. Several algorithms, including dynamic programming methods, heuristic algorithms and probabilistic methods, have been designed to solve the MSA problem.

1.3 Reconstructing Phylogenies

Two kinds of methods are available to reconstruct phylogenies:

1. **Character-based methods** retrieve relationships by comparing the states taken by species at different characters and usually take as input a MSA. Character-based methods are subdivided into parsimony methods, likelihood methods and bayesian methods.

2. **Distance-based methods** rely on a distance measure between the taxa under consideration, resulting in a pairwise distance matrix. Distance measures usually take a MSA as input. After the distance measure is performed, sequence information is not used anymore.

Before we outline character and distance-based reconstruction methods, we define characters and states, take a closer look on character compatibility and introduce one of the classical problems in phylogenetics, the *perfect phylogeny problem*. Although we will see that this problem is not relevant in real-world phylogenetic inference, it is essential for understanding Chapter 3 of this thesis.

Characters, States and Perfect Phylogenies

To find out about the evolutionary relationships between a group of species, we need to find certain variable characters of these species. Today, the most common used characters in phylogenetics are DNA sequences. In general, the following must hold for characters appropriate for phylogenetic reconstruction:

- We can decide if a species has this character or not.
- We can measure the quality or quantity of the character.

The actual quality or quantity of a character is called its state. Formally, we have the following:

Definition 1.1. A character is a pair $c = (\lambda, X)$ consisting of a character name λ and an arbitrary set X, where the elements from X are called *character states.*

Usually, we are given n species and for each species the character states of m characters, where each character takes on one of the possible states from X. Thus, the data can be summarized in an $n \times m$ *character-state matrix* M , where $M_{i,j}$ represents the state of the j^{th} character of the i^{th} species. The general aim of character-based reconstruction methods is to find a phylogeny that includes the character vectors of all internal nodes that best explains the data. As example, consider the following alignment of DNA sequences, representing taxa $t_1, t_2, t_3, t_4, t_5 \in S$ and five characters $c_1, c_2, c_3, c_4, c_5 \in C$. The character states are elements of $X = \{A, C, G, T\}$:

	c_1	c_2	c_3	c_4	c_5
t_1	A	C	C	G	G
t_2	A	G	C	G	T
t_3	A	C	T	G	G
t_4	T	G	C	G	G
t_5	T	C	C	G	G

Definition 1.2. Let c be a character, and T be a tree with its leaves labeled with states of c. Then c is *compatible* with T if all internal nodes of T can be labeled such that each character state induces one connected subtree.

If a tree is compatible with all given characters, this is called a *perfect phylogeny.* Formally stated:

Definition 1.3. Let C be a set of characters, and T be a tree with its leaves labeled with states for each character $c \in C$. Then T is called a *perfect phylogeny (PP)* for C if all characters $c \in C$ are compatible with T .

The *Perfect Phylogeny Problem* (PPP) addresses the question if for a given matrix M of characters there exists a perfect phylogeny, and, given its existence, how to construct it. For an arbitrary number of character-states, the PPP is NP-hard.

Given binary characters with character states $\in \{0,1\}$, the problem is also known as *complete directed perfect phylogeny problem* and stated as following:

	c_1	c_2	c_3	c_4	c_5
t_1	1	0	0	0	0
t_2	1	1	0	0	0
t_3	1	1	1	0	0
t_4	0	0	0	1	0
t_5	0	0	0	1	1

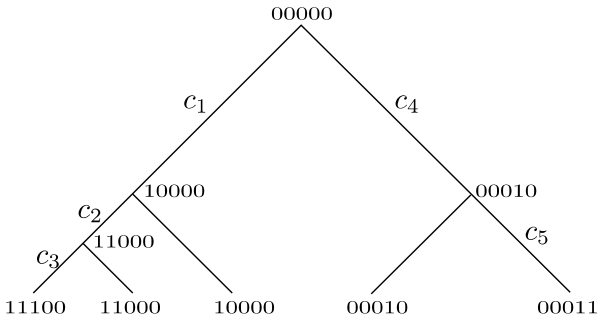


Figure 1.2: Complete Directed Perfect Phylogeny

Definition 1.4. Let M be an $n \times m$ matrix representing n taxa and m binary characters. A *complete directed perfect phylogeny* of M, if it exists, is a rooted tree T with the following properties:

- Each leaf is labeled by a row of M, and corresponds with the taxon characterized by that row.
- Each column of M labels at most one edge.
- The characters associated with the edges along the unique path from the root to a leaf exactly specify the character vector of the corresponding taxon, i.e., the character vector of the taxon has '1' entries in all columns corresponding to characters associated to the edges of the path and a '0' entry otherwise.

Note that not every edge of T has to be labeled by a character, but every character has to label exactly one edge. See Fig. 1.2 for an illustration. The definition of perfect phylogeny implies that the root of the perfect phylogeny is labeled by the null vector and each character changes from state '0' to state '1' at most once and never changes back from state '1' to state '0'.

Gusfield [83] proved the following theorem:

Theorem 1. Let M be a complete binary matrix representation of n taxa characterized by m characters, and O_i denote the set of taxa with a '1' in column i. M admits a perfect phylogeny if and only if every two characters i and j are *compatible,* that means, it holds

- $O_i \subseteq O_j$, *or*
- $O_j \subseteq O_i$, *or*
- $O_i \cap O_j = \emptyset$.

Gusfield [83] also introduced an algorithm with running time $O(mn)$ to test if a binary matrix M admits perfect phylogeny, and how to construct

it, if existent. Agarwala *et al.* [2] present a method with running time $O(k)$, where k is the number of '1's in the matrix. Fern [62] presented a detailed and well written review on the perfect phylogeny problem.

When no perfect phylogeny exists, the input character-state matrix M is incompatible. This is understood to be the rule for biological data rather than the exception due to several reasons. In case of character data like DNA sequences, conflicts between characters are usually interpreted as error in scoring of those characters or as real evolutionary events that disturb inference, such as multiple gains and losses of the same character state. Both are in general referred to as *homoplasy*. Phylogenetic reconstruction methods rely on objective functions in order to decide which tree explains the data best. We shortly present common optimization approaches to phylogenetic reconstruction in the following.

Parsimony methods

The main assumption of these methods is that evolution is parsimonious. Therefore, the objective is to find phylogenies with the minimum amount of evolution; that is, with the fewest number of evolutionary changes. process that species undergo. Input data consists in a set S of n discrete character sequences $s_0, \ldots, s_n$ of length m, one for each species under consideration.

Different weighting schemes are used to assign specific costs to each character state change and parsimony methods seek for an evolutionary scenario with minimal total cost, called the *most parsimonious tree*. Note that, if a perfect phylogeny exists, this is in particular most parsimonious.

The two most commonly used cost functions for multiple state characters are Fitch parsimony, where the cost of substituting a character state with another is equal to 1 for all states [64], and Sankoff parsimony, where a substitution cost is associated to each pair of character states x, y, with $x \neq y$ [144]. Other cost models have been proposed *e.g.*, the Dollo parsimony [58, 106] and Camin-Sokal parsimony [35]. For a broader and detailed discussion on different variants of parsimony see for example [59].

Each character c_j in a parsimony approach is considered independently and its number of substitutions needed along the branches of a given tree T is called parsimony score $P(c_j|T)$. The total parsimony score of the set S of character sequences is the (weighted) sum of parsimony scores of each character:

$$P(S|T) = \sum_{j=0}^{m} w_j P(c_j|T) \tag{1.1}$$

where w_j is the weight of character c_j. If the internal sequences for a given phylogeny T are known (see Fig. 1.3 for an example), $P(c_j|T)$ is simply the sum of substitutions necessary to explain different states for c_j along the branches of T, weighted by the substitution costs. Since only terminal sequences are known, the problem is to find a most parsimonious assignment

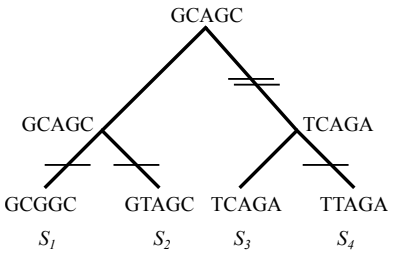


Figure 1.3: One of the most parsimonious trees for the set of sequences $S = \{S_1, S_2, S_3, S_4\}$. The five required substitutions are indicated by horizontal lines.

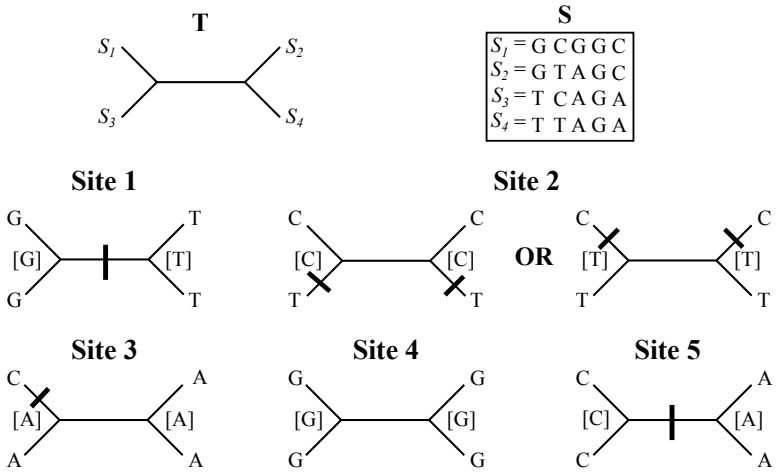


Figure 1.4: Most parsimonious reconstructions per sites for the set of sequences S given the tree T. Two equally parsimonious reconstructions are possible for site 2. Deduced characters are shown in square brackets.

of character states to internal nodes that minimizes $P(c_j|T)$ (see Fig. 1.4 for an example). This problem is referred to as *small parsimony problem*. Fitch [64] presented an $O(n \times m)$ algorithm to calculate $P(S|T)$ for unit costs on binary trees. Sankoff developed a polynomial-time generalization for symmetric, edge independent costs which may vary for certain events [142–144]. However, the tree is not known and finding the tree T that yields the minimal total parsimony score $P(S|T)$ among all possible trees is an NP-hard problem [68]. This is known as the *Maximum Parsimony Problem*, for which several heuristic search strategies have been proposed (see [59] for an overview) and implemented (e.g. PAUP* [163]).

A standard objection to the parsimony criterion is that it is not consistent [36, 60]; that is, even with infinite amount of data (infinitely long sequences), one will not necessarily obtain the correct tree. The standard example for this behavior of parsimony methods is subsumed under the term

long branch attraction, which refers to the phenomenon when rapidly evolving lineages are inferred to be closely related, regardless of their true evolutionary relationships (for a review on the history of long-branch attraction and methods suggested to detect and avoid the artifact see for example [11]).

Likelihood methods

The first application of a likelihood method to molecular sequences was proposed by Neyman [113], improved by Kashyap and Subas [95] and Felsenstein [61]. Given a quantitative evolutionary model, we can calculate the likelihood that a set of sequences would have evolved on any proposed phylogenetic tree. The maximum-likelihood (ML) criterion seeks to choose the tree that maximizes this likelihood [61]. To calculate the likelihood of a set of sequences on a tree, the tree topology including branch length and all parameters of a substitution model have to be specified. Many nucleotide substitution models have been suggested, varying in complexity regarding their parameters. Some common DNA models are (listed in ascending complexity) JC69 [93], Kimura-2-Parameter model (K2P) [99], the Felsenstein model (F81) [61], HKY85 [85], TN93 [164] and the General Time Reversible model (GTR) [85]. Models of protein evolution are often based on empirical matrices stating the relative rates of replacement from one amino acid to another. Several protein models exist, commonly used models are PAM [49], GONNET [77], Blosum62 [86] and WAG[170]. In principle the same heuristic tree search strategies can be used to find an ML solution as in the case of a parsimony analysis. For simple evolutionary models efficient heuristics have been developed and implemented in several programs, e.g. PAUP* [163], PHYML [82], IQPNNI [168], RAxML [153].

Distance-based Methods

Distance-based tree building methods rely on a distance measure between the taxa, resulting in a distance matrix. The idea when using distance based tree building methods is that knowledge of the "true evolutionary distances" between homologous sequences should enable us to reconstruct their evolutionary history. Methods to infer phylogentic trees from a pairwise distance matrix include the least squares method [65] and clustering methods such as Neighbour Joining [136].

1.4 Approaches for large-scale phylogenetic inference

The reconstruction of large phylogenetic trees comprises various computational difficult problems. It is well known that inferring optimal trees from sequences under the maximum likelihood (ML) [130] and the maximum parsimony (MP) criterion [68] are computationally hard problems, so we have to rely on heuristics that cannot guarantee to find the optimal solution. Even

for a moderate number of species, the sheer size of the tree space prohibits searching for optimal trees under these criteria. Although some studies with thousands of taxa have been reported in the literature, they mostly intend to investigate new concepts of implementation and computation, for example, a ML analysis of 10 000 taxa on a parallel computer [154]. It remains unclear how close the resulting tree is to the optimal one, considering the hardness of the problem together with the enormous number of trees to be searched: For 10 000 taxa there exist $8.0 \cdot 10^{38658}$ unrooted binary phylogenetic trees, a number much larger than the number of atoms in the observable universe.

Two different frameworks approaching the problem of large-scale phylogenetic inference exist. The most obvious procedure is to concatenate smaller character matrices into a single larger *supermatrix*, where unknown character states are coded by question marks. Then, all characters are analyzed simultaneously under standard reconstruction criteria, with the implicit advantage that all information of each individual source is retained. This methodology, first advocated explicitly by Kluge [100], is also called 'total-evidence' approach due to the accordance with the philosophical principle that the best hypothesis is the one derived from all available data.

Instead analyzing primary character data, the *supertree* approach combines input trees with overlapping taxa sets into one larger and more comprehensive tree [19].

The supermatrix and the supertree frameworks are classically considered as mutually exclusive and competitive approaches for analyzing large data sets at fundamentally different levels [21]. Over the years, there has been much debate about the pros and cons of both strategies (e.g. [21, 26, 71, 72]). Both approaches have certain limitations, but especially supertree methods have been strongly criticized (e.g. [131, 132, 149]) mainly in that they don't combine primary character data directly and thus represent a meta-analysis one step removed from real data. For some authors, this possibly entails a loss of valuable information [70, 71, 152]. Without dealing with all issues exhaustively, we will briefly outline some of the most relevant points of discussion in the following. Major points of criticism regarding supertree methods relate to:

1. **Data independence**. In a supertree approach, where trees instead primary characters are combined, the same character data can contribute to more than one source tree. This effect can be prevented to a large extent by the use of a formal data collection protocol as proposed by Bininda-Emonds *et al.* [16].

2. **Signal enhancement**. It has been shown [8, 42], that a supermatrix analysis of two data sets can yield conflicting phylogenetic trees, whereas a simultaneous analysis of the same data sets results in a phylogeny in which the congruent subsignals from each data set overcome the individual conflicting primary signals. This effect, called *signal*

enhancement, cannot occur in the supertree framework, because the combination of trees does not allow easily to account for subsignals in the original data sets [120]. The missing ability of all supertree methods to account for signal enhancement and the fact that supertree methods can produce *novel clades*, relationships that are not present in or implied by the set of source trees [17, 18], have been criticized by various authors (including Gatesy *et al.* [71, 72], Goloboff and Pol [75], Pisani and Wilkinson [120], Springer and de Jong [152]). Using simulated data, Bininda-Emonds [18] showed that supertree analysis (in particular the Matrix Representation with Parsimony (MRP) method, see Chapter 2, page 23) is often on par with supermatrix analysis of the combined primary character data regarding accuracy of the resulting trees, and produce few, if any, novel clades. Furthermore, Bininda-Emonds [19] points out that several supertree methods allow differential weighting of the source trees and thus in principle can account for differential signal strength in the primary data. Nevertheless, to which extent and under which circumstances unsupported and undesired novel clades occur is not yet adequately documented and subject of ongoing debate, see for example Goloboff [74].

3. **Supertrees as phylogenetic hypothesis**. Some authors argued, that supertrees represent summaries of summaries and not valid phylogenetic hypothesis and hence should not be used to propose new phylogenies (*e.g.* Gatesy *et al.* [71, 72], Springer and de Jong [152]. Bininda-Emonds [21] argues that supertrees propose hypotheses of relationships between taxa, that can be evaluated as any other phylogenetic hypotheses. Discrepancies between supertree and supermatrix analysis should be treated in the same sense as it is done for conventional phylogenetic analysis.

An advantage of the supertree framework is that combining trees instead of analyzing primary data implies that the input trees to a supertree method can be based on different kind of data, for example DNA of different genes, morphological traits or DNA-DNA hybridization. Further, it doesn't matter by which method the trees have been reconstructed (*e.g.* maximum parsimony, maximum likelihood or a distance-based method) what allows to use the most adapted for each data set.

Despite all criticism, over the years supertree construction has established itself as a very feasible approach to construct large and comprehensive phylogenetic hypotheses for various organismal groups. The first large-scale supertree of Primates was published by Purvis [122] in 1995. Many of the supertrees published thereafter represented the first complete phylogenetic estimation for the groups under consideration (see [21] for a now outdated list from 2004) and remain in many cases the only such estimate. This is due

to the often criticized meta-analysis characteristic of the supertree framework, which also can be seen as a potential strength in comparison to the supermatrix approach with its inherent issues of character compatibility (see below). Bininda-Emonds [22] assumed that probably none of these complete phylogenetic hypotheses could have been constructed using a supermatrix approach. This assumption relies mainly on the observation that data collection is largely uncoordinated and opportunistic Sanderson and Driskell [140]. Thus, some species are overrepresented, whereas, in comparison, others are dramatically underrepresented. As example, Bininda-Emonds [22] points out that in March 2004 the GenBank database (`http://www.ncbi.nlm.nih.gov/Genbank/index.html`) contained nearly two million carnivore sequences, but 99.6% of them relate to the domestic dog, which means that the other species were represented by very few genes and sequence.

The supermatrix approach also implicates certain limitations and difficulties. First, the issue of *missing data*, referring to empty cells in a concatenated supermatrix representing cases for which the systematist did not observe a particular character (the GenBank example from Bininda-Emonds [22], see above, might be seen as a typical example). In general, missing data is seen as problematic, as they might lead to spurious relationships, lack of resolution and accuracy [98, 141, 171]. A substantial amount of missing data is an almost unavoidable fact for any supermatrix analysis attempting to sample large groups comprehensively, because only few genes are sampled very completely across many taxa. For example, one of the largest supermatrices ever analyzed, with > 2000 taxa had 95% missing taxa [110], or the supermatrix analysed by [52] containing 469,497 sites for 70 taxa, issued from 1131 protein alignments, had 92% missing data. Even without missing taxa, methods like maximum likelihood tend to become computational intractable when the datasets get too large. In contrast to the supertree approach, the supermatrix approach allows only to include data of the same type. Using conventional supermatrix approaches, combined primary data is analyzed using a single evolutionary model, what can be problematic as genes can have different stationary frequencies or different evolutionary rates. For these reasons, the view of a supermatrix as a single super-gene may be inadequate, whereas the "separate-model" introduced by Yang [173] in which each gene has its own evolutionary parameters, requires the design of new dedicated optimization heuristics. To overcome these problems, Bayesian analysis have been proposed, which consists in partitioning the supermatrix and then applying appropriate models and their specified parameter estimates to each data partition and subsequently incorporate this into a single tree search. As has been shown by Bevan *et al.* [14], that this is the best approach so far to account for gene rate heterogeneity. Another disadvantage of the supermatrix approach is that some kind of data (e.g. DNA-DNA hybridization, distance data, morphometric data) cannot be analyzed under any of the frameworks developed for more usual kind

of data (molecular sequences or morphological traits) *i.e.*, maximum parsimony, maximum likelihood and Bayesian methods Bininda-Emonds [18].

Future directions for the supertree and supermatrix framework

Despite the inherent advantages and disadvantages of the widely divergent supermatrix and supertree approach, we think that for the moment none of the two frameworks is significantly better than the other and should be seen as simply different. Even if considered as strictly competitive ways for large-scale phylogenetic inference, in our opinion a choice between both approaches has to be done for each data set, depending on various factors like data set size, kind of data, availability of data and so on.

Certainly the two approaches can be used to analyze the same data set simultaneously in order to reveal parts of the tree that both methods indicate in common. These consistent hypothesis might be trusted with increased confidence, whereas conflicting regions rather suggest deeper investigation. This counterbalancing strategy accords to the global congruence approach [104], advocated and used in the supermatrix/supertree context by several authors (e.g. [21, 88]). Nevertheless, a global congruence framework applying supermatrix and supertree methods at the same time is clearly ultimately limited computationally as problem sizes continuously increase [23].

With the advent of Next Generation Sequencing and - probably within the next 5-10 years - Third Generation Sequencing technologies, an ever-increasing amount of sequence data of more and more different species becomes available. This will change the afore described situation of unequal source data distribution and means that trees with comparable taxonomic coverage for many groups will soon be available using supermatrix approaches.

We think a promising approach for the future to build larger portions of the tree of life is to subsume supertree construction within supermatrix analysis in a *divide-and-conquer* meta-technique. This has been advocated by many authors over the years, including even critics of the supertree aproach, see for example Bininda-Emonds *et al.* [15], Bininda-Emonds [19, 21], Bininda-Emonds *et al.* [26], de Queiroz and Gatesy [50], Gatesy and Springer [70], Huson *et al.* [91], Pennisi [118], Soltis and Soltis [151].

A divide-and-conquer strategy in phylogenetic inference amounts to breaking a larger data matrix in many smaller ones, with the trees derived from their analysis being combined to obtain the complete phylogenetic supertree. Here, the supertree is not the end result, but rather functions as a starting tree for the subsequent phylogenetic analysis that can take into account the complete data.

The potential advantages of such a divide-and-conquer approach derives from two bases. First, smaller phylogenetic problems can be solved much faster than larger ones, due to the NP-hardness of phylogenetic analysis using ML or MP. As an example, Bininda-Emonds and Stamatakis [25]

showed that in the time needed to analyze a single 4096-taxon dataset with 1000 nt sequences using MP, is the same as for analyzing approximately 250 000 datasets with 16 taxa each, derived from the large dataset by pruning taxa. To derive a useful supertree, we may need one hundred such small phylogenies, resulting in a speedup of about 2500-fold for this part of the analysis. A similar speedup can be expected for ML analysis. In practice, smaller but still significant speedups can be expected. Second, Roshan *et al.* [135] argue that the solutions to the smaller subproblems arising in a divide-and-conquer might be be comparatively more accurate. On the one hand, faster analysis times allow the use of more robust search strategies and on the other hand the smaller subproblems have a smaller phylogenetic diameter (*i.e.* the taxa tend tend to be more closely related) and thus represent a less difficult problem.

A key to the potential advantages of a divide-and-conquer framework for large-scale phylogenetic analysis regarding both speed and accuracy is the used supertree method. In the next chapter, we present several current supertree methods. Then we discuss necessary demands to be satisfied from possible candidates for a viable divide-and-conquer strategy. We will see that none of the existent supertree method meets all requirements in satisfying manner.

CHAPTER 2

Supertree Methods

In recent years, supertree methods have become a familiar tool for building large phylogenetic trees. Supertree approaches combine *source* trees with overlapping taxa sets into one large and more comprehensive tree [19]. Source trees are phylogenetic trees inferred from primary data (e.g. amino acids, SINEs or morphological traits) using standard reconstruction methods such as maximum parsimony (MP) or maximum likelihood (MP). Systematists have probably used informal supertree methods since the beginning of systematics itself, pasting together hierarchically nested taxa. Since the introduction of the term *supertree* and the first explicit formal supertree method [79], there has been a continuous development of such methods, see for example [20].

The supertree problem is a generalization of a simpler one, called the *consensus* problem, which consists in summarizing a set of trees, each containing the same set of taxa, into one consensus tree. Thus, supertree methods can also be applied in the consensus setting, but consensus methods can not be used to solve the supertree problem.

As outlined at the end of Chapter 1, a divide-and-conquer strategy where supertree methods and the supermatrix approach are combined could play a key role in constructing very large phylogeny in the future. Current supertree methods can roughly be subdivided into two major families: matrix representation (MR) and polynomial, mostly graph-based methods. The former encode the inner vertices of all input trees as partial binary characters (0, 1 and ?) in a matrix, which is analyzed using an optimization or agreement criterion to yield the supertree. Matrix representation with parsimony (MRP) [9, 124], the first matrix-based method, is still by far the most widely used supertree method today. The MRP problem is to find one or more equally most parsimonious trees over all matrices that result from replacing ?s with 1s or 0s.

Another particular matrix representation supertree method is matrix representation with flipping: Utilizing the parsimony principle, MRF seeks the minimum number of "flips" $0 \rightarrow 1$ or $1 \rightarrow 0$ in the input matrix that make the resulting matrix consistent with a phylogenetic tree, where '?'-entries can be resolved arbitrarily. Evaluations indicate that MRF is on par with the "gold standard" MRP [39]. All MR methods have in common that the underlying optimization problem are computationally hard, and heuristic search strategies have to be used. As for ML and MP, it is unclear how close the resulting tree is to the optimal one.

Graph-based methods make use of a graph to encode the topological information given by the input trees. This graph is used as a guiding structure to build the supertree top-down from the root to the leaves. The first

graph-based supertree method was the BUILD algorithm [3], which is only applicable to non-conflicting input trees and thus only of limited use, because "as most systematics know, phylogenies usually conflict with each other" [20]. This led to the development of the MINCUTSUPERTREE algorithm (MC) [147] and a modified version, MODIFIEDMINCUTSUPERTREE (MMC) [116], that use a minimum cut approach to construct a supertree if the input trees are conflicting. The *Build-with-distances* algorithm (BWD) [172] is the first graph-based method that includes branch length information from the input trees to build the supertree. All methods mentioned so far, MRP, MRF, MC, MMC and BWD have in common that they apply a *voting* procedure to deal with conflicts among the input trees. They resolve conflicts by asking the input trees to vote for a clade in the supertree, such that, to a certain extent, the most frequent alternative is chosen. In case of conflicting input trees, voting methods such as MRP can infer supertrees in which clades are present that contradict each of the input trees [44, 74, 75]. To which extent and under which circumstances these unsupported and undesired novel clades occur, is subject of ongoing debate (see for example [18, 74]). Ranwez *et al.* [127] presented a new graph-based method, the *PhySIC* algorithm. The method ensures that the reconstructed supertree satisfies two properties: it contains no clade that directly or indirectly contradicts the input trees, and each clade in the supertree is present in an input tree, or is collectively induced by several input trees. Supertree methods guaranteeing the first property are called *veto* methods and, for highly conflicting and/or overlapping input trees, tend to produce unresolved supertrees. Scornavacca *et al.* [146] presented a modified version of *PhySIC*, *PhySIC_IST*, that tries to overcome this drawback by proposing non-plenary supertrees (that is, supertrees that do not necessarily contain all taxa from the input trees), while still assuring the properties mentioned above. *PhySIC_IST* works in a stepwise fashion, iteratively adding leaves to a starting tree consisting of two nodes. In contrast to the MR methods, the MC, MMC, BWD, *PhySIC* and *PhySIC_IST* algorithms have polynomial running time.

This chapter is organized as follows: First, we introduce basic notions and definitions. In Sections 2.2 -2.4, we present the above mentioned and other supertree methods relevant in the context of this thesis. As evoked in Chapter 1, Section 1.4, a divide-and-conquer framework teaming the supermatrix approach and supertree methods might be a useful strategy for large scale phylogenetic inference in the future. In Section 2.5, we continue this idea and have a closer look on requirements that potential supertree methods have to fulfill in this context. Until now, the theoretical advantages of a divide-and-conquer strategy in terms of speed and accuracy over more conventional heuristic search strategies have not been realized.

2.1 Basic notions and definitions

In the following, we present the notions and definitions used throughout the rest of this thesis.

2.1.1 Graphs

A *graph*, denoted $G = (V, E)$, is a structure consisting of a set V of *vertices*, and a set $E \subseteq \{\{x, y\} : x, y \in V\}$ of connections called *edges*.

We call graph G a *undirected graph*, if for every edge $e = \{x, y\}$ connecting vertices x and y, the order of vertices in G is not fixed and write $e = xy$ instead of $e = \{x, y\}$ for brevity. Vertices in an undirected graph G belonging to an edge are *end vertices* of e. Two vertices connected by an edge in G are called *adjacent* to to each other and *incident* to e. Two adjacent vertices are *neighbors* of each other. We denote the *neighborhood* of a vertex x in an undirected graph by $N(x) := \{y \in V \mid xy \in E\}$. The *degree* of a vertex x is defined as the number of its neighbors and is denoted by $\deg(x) := |N(x)|$.

A *length-l path* is a sequence $p := x_{p_1}, x_{p_2}, \ldots, x_{p_l}$ of l vertices, where two subsequent vertices are connected by an edge. Vertex x_{p_1} is the *starting point* of p and x_{p_l} is the *end point* of p. A *cycle* is a path whose starting point and end point are identical. An undirected graph that does not contain any cycle is an *acyclic* graph. An undirected graph G is called *simple* if it has no *loops* (i.e an edge that connects a vertex to itself) or *parallel edges*, that is G contains no multiple edges incident to the same two vertices. We call graph G a *directed graph*, if for every edge e connecting vertices x and y, the order of vertices in G is fixed. We denote an edge e in a directed graph by an ordered pair $e = (x, y)$, where x is considered to be directed from x to y. y is called the head and x is called the tail of e. In a directed graph one distinguishes between different degrees of a vertex: For a vertex, the number of head endpoints adjacent to a vertex is called the *indegree* of the vertex and the number of tail endpoints is its *outdegree*.

Given $E' \subseteq E$, we let $G \setminus E'$ denote the graph obtained from G by deleting all edges in E'. If $G \setminus E'$ is disconnected, E' is called a *cut set* of G. A undirected or directed graph G is called *weighted*, if if a number (weight) is assigned to each edge in G. In a weighted graph, $w(E') := \sum_{e \in E'} w(e)$ is the *cut weight* of E'. A cut set of minimum weight is called a *minimum cut*.

A *connected component* C of a graph G is a subgraph of connected vertices, that is, for any pair of vertices x and $y \in C$ there is a path from x to y. Vertices of C are not connected to any additional vertices. If a graph is connected itself, it has exactly one connected component consisting of the whole graph.

A graph $G = (V, E)$ is a *bipartite graph* if V can be partitioned into two disjoint subsets V_1 and V_2 such that each edge of G connects a vertex in V_1 with a vertex in V_2. We write $G = (V_1, V_2, E)$ to denote a bipartite graph

whose partition has the parts V_1 and V_2.

A *network* $N = (V, A)$ is a directed weighted graph. For readability, we will call the edges in a network *arcs*. Note that a network can also be bipartite, and we will write $N = (V_1, V_2, A)$ in this case.

2.1.2 Trees

A tree $T = (V, E)$ is a simple, connected graph with no cycles. A vertex $v \in V$ is *internal* if the degree of v is greater than one, otherwise v is a *leaf*. In phylogenetics, the leafs of a tree usually correspond to the taxa under consideration.

For a given tree T, $L(T)$ denotes the set of leaves of T. If $L(T) = X$, and T has one distinguished internal vertex, denoted *root*, and no vertex but the root may have degree two, then T is called *rooted phylogenetic tree (on X)*. In this thesis, we restrict ourselves to rooted trees, as most supertree methods do. Usually, supertree and parent tree problems become much harder if we consider the unrooted case [157]. As most of the supertree methods under consideration in this paper require rooted trees as input, we neglect the unrooted case. For brevity, we often use the terms "rooted phylogenetic tree" ,"phylogenetic tree" or simply "tree" synonymously in the following. In a rooted phylogenetic tree, the *outdegree* of the root is simply its degree, whereas the outdegree of all other vertices is the degree minus one. A tree is *binary* if there is no vertex with out-degree larger than two, a internal vertex with an outdegree larger than two is called *polytomy*. In a tree T with weighted edges, the *path length* between two vertices x, $y \in V$, denoted $pl(x, y)$, is the sum of weights of all edges between x and y.

Let T be a tree on X. An element of X is a *descendant* of an internal vertex v of T if the path from this element to the root contains v. Given a particular internal vertex v, a *clade* of T is a subset of X that consists of all elements of X that are descendants of v.

For a given subset $X' \subseteq L(T)$ we define the *last common ancestor* of X' in T to be the lowest internal vertex of T that has all leaves $\in X'$ as descendants. The last common ancestor of X' in T is the shared ancestor of v and w that is located farthest from the root. For shortness, we will refer to the last common ancestor of X' as lca(X')).

Given $X' \subseteq L(T)$, we construct the *restriction* of T to X', denoted $T|X'$, by first finding the minimal subtree of T containing X', and then suppressing all vertices of degree two except for the root.

For a set of phylogenetic $\mathcal{T} = \{T_1, ..., T_x\}$, let $L(\mathcal{T})$ denote the set of leaves that appear at least in one tree. Let T_1 and T_2 be two trees with $L(T_1) \subseteq L(T_2)$. The tree T_2 *displays* T_1 if $T_2|L(T_1)$ is a *refinement* of T_1, i.e. T_1 can be obtained from T_2 by contracting edges. Informally, T_2 displays T_1 if, up to polytomies, all the ancestral relationships of T_1 are preserved in T_2. For a set of trees $\mathcal{T}$ with possibly overlapping leaves, we say that $\mathcal{T}$ is *compatible* if there exists a tree T^* on $L(\mathcal{T})$ that displays every tree $T_i \in \mathcal{T}$.

In this case, we call T^* a *parent tree* for $\mathcal{T}$. Otherwise, $\mathcal{T}$ is *incompatible*. When $\mathcal{T}$ is incompatible, it is desirable to find a tree T^* over $L(\mathcal{T})$ that minimizes some objective function measuring the incompatibility. Then T^* is denoted *supertree* and the problem of finding T^* is called the *supertree problem*. Since biological data is incompatible for a range of reasons, including sampling errors, inaccuracies, or biases in tree building algorithms, incompatible input trees are what has to be expected in reality.

Triplets

A *triplet* is a binary tree with three leaves, a non-binary tree with three leaves is called a *fan*. In some sense, triplets are the basic building blocks of rooted phylogenetic trees. A triplet with leaves x, y, z, is denoted $xy|z$ if the path from x to y does not intersect the path from z to the root. Given a rooted tree T and three leaves x, y, z from $L(T)$, $T|_{x,y,z}$ denotes the homeomorphic subtree of T induced by the leaves labeled by x, y, and z. If T is binary, $T|_{\{x,y,z\}}$ can be any of the three possible triplets on x, y, z. We say that T *induces* or *displays* the triplet $xy|z$ if $T|_{x,y,z} = ((x, y), z)$. If T is not binary, it can happen that $T|_{\{x,y,z\}}$ only induces the *fan* (x, y, z) and we say that {x,y,z} is *unresolved* in T. Any rooted tree T can be equivalently described by the set of triplets homeomorphic to its subtrees connecting three leaves. Given a tree T, $rt(T)$ denotes the set of all triplets of T. For a collection of trees $\mathcal{T}$, $rt(\mathcal{T})$ denotes the set of triplets present in at least one tree of $\mathcal{T}$, *i.e.* $rt(\mathcal{T}) = \bigcup_{T_i \in \mathcal{T}} rt(T_i)$. A set rt of triplets is compatible if and only if there is a tree T that displays all triplets in rt. The compatibility of a set of triplets and thus the compatibility of a set of rooted phylogenetic trees can be decided in polynomial time with the algorithm presented by Aho *et al.* [3], see page 26.

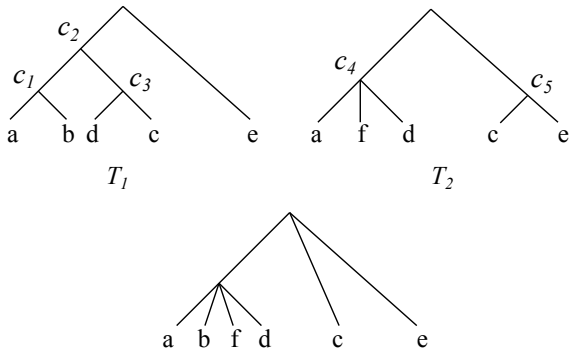
2.2 Matrix Representation-based Methods

This set of supertree methods encodes inner vertices of all input trees as incomplete binary characters in a matrix, which is then analyzed using an optimization or agreement criterion. We will explain the matrix encoding of input trees in context of the Matrix Representation with Parsimony method, see below.

2.2.1 Matrix Representation with Parsimony (MRP)

In 1992, Baum [9] and Ragan [125] independently proposed the matrix representation with parsimony (MRP) method. MRP is by far the most widely used but also most controversially discussed supertree method today. Several studies have shown that constructed supertrees are of comparatively high quality (see Chapter 4). The method consists in the following steps:

1. All input trees are rooted by a taxon common to all trees; if trees are already rooted, re-root it [9]. Alternatively, or if no taxon is common



	c_1	c_2	c_3	c_4	c_5
a	1	1	0	1	0
b	1	1	0	?	?
c	0	1	1	0	1
d	0	1	1	1	0
e	0	0	0	0	1
f	?	?	?	1	0

Matrix representation M(T_1, T_2)

Figure 2.1: Shown are two rooted input trees T_1, T_2, their corresponding matrix representation M and the MRP supertree using Fitch parsimony.

to all trees, all trees are rooted with a dummy (all-zero) outgroup [123, 124].

2. Matrix encoding of the input trees. In the commonly used matrix encoding, each tree is expressed by additive binary coding [57]. This is sometimes also referred to as Baum-Ragan coding. Each inner vertex of each input tree is transformed into one column of the matrix representation. The taxa below the inner vertex are encoded '1', all other taxa in the input tree are encoded '0', whereas taxa missing from the input tree are encoded '?' [9, 10, 124].

3. Analyze the obtained matrix by the parsimony criterion. In principle, the same kinds of parsimony as in a standard parsimony analysis (see Chapter 1, page 1.3) can be used.

See Fig. 2.1 for an example of the MRP method.

In case that many equal-best trees exist, usually a strict consensus of these most parsimonious trees is returned. As in standard parsimony analysis, the underlying Maximum Parsimony problem is computationally hard [68]. Beside standard tree search strategies, heuristics have been proposed aiming to ensure that MRP remains computationally feasible for large data sets, among them MCMC based methods (e.g. [133]) or the ratchet technique [76].

The MRP method was subject of very lively debates throughout the years, Baum and Ragan [10] provide an overview of most points of criticism and discussion. For example, Purvis [123] suggested an alternative coding aiming to avoid redundant information, which has become known as Purvis-coding. Here, a taxon below an inner vertex is encoded '1', a '0' is used for

each taxon induced by sibling nodes of this inner vertex and '?' is used to encode the remaining taxa.

One of the most discussed facts concerning the MRP method is that, in case of conflicting input trees, it can propose *novel clades* that are not supported by any input tree alone or by combinations of the input trees (e.g. [17]). Moreover, in a MRP supertree, clades can be present that contradict each of the input trees [44, 74, 75]. To which extent and under which circumstances these unsupported and undesired novel clades occur, is subject of ongoing debate (see for example [18, 74]).

2.2.2 Matrix Representation with Flipping (MRF)

The Matrix Representation with Flipping (MRF) method [37, 38] assumes that conflicts between input trees correspond to *errors* consisting of taxa present or absent in a clade where they should not be. Such errors correspond to "flips" from $0 \rightarrow 1$ or $1 \rightarrow 0$ in the matrix representation of the input trees [40], which prevent the matrix from representing any phylogenetic tree perfectly. MRF seeks the minimum number of flips in the input matrix that make the resulting matrix consistent with a phylogenetic tree, where '?'-entries can be resolved arbitrarily. Evaluations indicate that MRF is on par with the "gold standard" MRP [39, 56].

We first consider the subject that '?'- entries can be resolved arbitrarily in the input character state matrix. Let $M = [m_{ij}]$ be an $n \times m$ incomplete binary input matrix with $m_{ij} \in \{0, 1, ?\}$ for all i, j. A *completion* of M is a $n \times m$ binary matrix $M' = [m'_{ij}]$ without question marks, such that, for all $i, j, m_{ij} = m'_{ij}$ whenever $m_{ij} \neq ?$. The incomplete matrix M is said to be *compatible* if it has a compatible completion. Thus, a compatible completion of M only exits, if and only if all $?s$ in M can be changed to $0s$ or $1s$ such that every two characters of M' are disjoint or one of them contains the other (see Chapter 1, page 10). On pages 26 and 37, we will introduce two polynomial-time algorithms introduced by Aho *et al.* [3] and Pe'er *et al.* [117] to test for the existence of a compatible completion and to construct one if it exists.

Given two matrix representations M and M' on the same taxa set, we denote the i^{th} character of M by $M[i]$. We denote the flip-distance between M and M' by $dist(M, M')$, where:

- the flip-distance between two characters $dist(M[i], M'[j])$ is the minimum number of flips of 1s and 0s needed to convert $M[i]$ into $M'[j]$. Any positions where $M[i]$ or $M'[j]$ is a ? are not considered.
- the flip-distance for a character in M and matrix M', $dist(M[i], M')$ is the minimum flip-distance from $M[i]$ to any column of M'.
- the flip-distance of two matrices M and M', $dist(M, M')$, is $\sum_i dist(M[i], M')$.

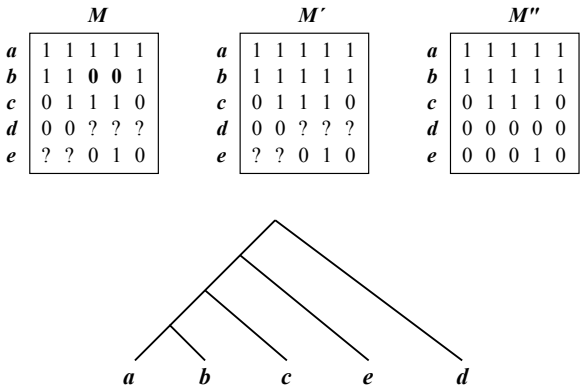


Figure 2.2: Example of MRF [40]. Top, from left to right: An incompatible input matrix on taxon set $M = \{a, b, c, d, e\}$. M' is the compatible matrix that results from flipping the highlighted entries of M. M" is a completion of M'. Bottom: The tree corresponding to M''.

Note that the flip-distance is not symmetric.

See Fig. 2.2 for an example of the MRF method.

The MRF problem is NP-hard [37, 40] and heuristics based on branch swapping have been proposed [40, 56]. Furthermore, the MRF supertree problem is W[2]-hard and has no constant factor approximation unless P = NP [29]. Chimani *et al.* [41] introduced an Integer Linear Program to find exact solutions, which is limited to very small and "simple" instances that do not require too many flips. Finally, Böcker [27] developed a data reduction for the MRF problem that runs in polynomial time, but cannot guarantee to simplify the instance. It turns out that running times of the data reduction, although polynomial, are too high in applications; on the other hand, the amount of data reduction actually performed in practice was rather small.

2.3 Supertree methods based on the Build algorithm

This set of supertree methods rely on variants of a graph introduced by Aho *et al.* [3] to encode the topological information given by the input trees. The graph is used as a guiding structure to build the supertree top-down from the root to the leaves. In contrast to the MR methods, graph-based supertree methods have in common that they provide polynomial running time.

2.3.1 Build, OneTree supertree and variants

The first graph-based supertree method is the Build algorithm which was originally developed in the context of relational databases. Build is an all-or-nothing algorithm that returns a tree T only if the input set rt of triplets on a set L of vertices is compatible. In phylogenetic context, the most

interesting property of BUILD is that it allows to test in polynomial-time if two or more rooted input trees are compatible and if so, how to construct the supertree for the compatible input trees. To achieve this, BUILD tries to construct a tree T displaying all triplets in rt *i.e.*, to find a tree such that $L(T) = L$ and $rt \subseteq rt(T)$. The tree T is constructed from the root to the leaves, guided by a graph that we will call *Build graph*. In the first iteration of the algorithm, all leaves from the input trees are used as vertices in the Build graph. Two vertices x, y are connected in the Build graph if and only if there exists $xy|z \in rt$. See Fig. 2.3 for an example. Here, the input is the set of triplets $rt = rt(\mathcal{T})$, where $\mathcal{T} = \{T_1, T_2, T_3\}$: The Build graph is the displayed graph without the dashed edges, and has three connected components, namely $\{a, b, d\}$, $\{c, e, f\}$, and $\{g, h\}$. The resulting connected components correspond to the clades beneath the root of tree T. In the recursion step, rt is restricted to the vertices of one connected component respectively, and the Build graph is constructed as described above. This process is repeated until a connected component consists of only one or two vertices. In both cases, the vertices are directly grafted into the supertree as leaves. Concerning real biological data, the main drawback of the BUILD algorithm is that it is not able to resolve incompatibilities between input trees. It was shown by Bryant and Steel [32], that when the input triplets are incompatible, the algorithm will construct a Build graph consisting of only one connected component with three or more vertices. Now, the algorithm has no information on how to build clusters in tree T, and terminates. The graphical version of the BUILD algorithm as presented here was used in [3] to show that their method has $O(mn)$ complexity when applied to a set of m rooted triplets and n leaves. In 1992, Steel [156] proposed to use the *Build* algorithm for phylogenetic studies. The ONETREE supertree method presented by Ng and Wormald [114] is based on the BUILD algorithm and differs only in small details: For a rooted forest $\mathcal{T}$, ONETREE applies the BUILD algorithm to the triplet set $rt(\mathcal{T})$, to obtain a supertree T such that $L(T) = L(\mathcal{T})$ and $rt(\mathcal{T}) \subseteq rt(T)$. In extension to the BUILD algorithm, Ng and Wormald consider the compatibility of triplets and fans in the original version of the OneTree algorithm [114]. Further, they provided an algorithm called AllTrees that takes a compatible set of triplets and fans and returns all trees T displaying this set. Both algorithms run in polynomial time, see [114] for details.

Bryant and Steel [32] presented a version of the ONETREE algorithm which not explicitly considers fans and thus is equivalent to the BUILD algorithm as presented above. In case of binary input trees, there exists a faster implementation of this method [87] that runs in $O(m \cdot |L(\mathcal{T})|^{0.5})$ time, where $m = \sum_{T_i \in \mathcal{T}} \mathcal{I}(T_i)$ and $\mathcal{I}(T)$ is the set of interior nodes in T. By changing the dynamic connectivity algorithm it resorts to, Berry and Semple [13] provided an improvement of this algorithm with running time $O(m \cdot \log^2(|L(\mathcal{T})|))$.

In general, there often exist more than one tree displaying set of a compatible triplets rt. According to Semple and Steel [148], the number of rooted phylogenetic trees with this property may be exponential in $|rt|$. In addition to AllTrees from Ng and Wormald [114], two more methods based on the BUILD algorithm exist that deal with the set of trees displaying a compatible set of triplets rt: Constantinescu and Sankoff [43] presented a method called SUPERB, that returns all binary trees displaying a compatible set of triplets rt, each in polynomial time.

Semple and Steel [148] provided the algorithm ALLMINTREES that returns, for a given set of triplets $rt(\mathcal{T})$, all trees $\mathcal{T}_{rt}^{min}$ that display rt and are minimal i.e., such that, $\forall T' \in \mathcal{T}_{rt}^{min}$ no internal edge of T' can be contracted such that the resulting tree also displays rt.

2.3.2 MinCutSupertree

Semple and Steel [147] list five desirable properties of a supertree method:

1. Changing the order of the trees in the set of input trees $\mathcal{T}$ does not change the resulting supertree.

2. Relabeling the input species results in the corresponding relabeling of species in the supertree.

3. If there are one or more parent trees with which every tree in $\mathcal{T}$ is compatible then the supertree is one of those parent trees.

4. Any leaf that occurs in at least one input tree occurs in the supertree.

5. The supertree can be computed in polynomial time.

In the same paper, they introduced the first supertree method satisfying all five desirable properties, the MINCUTSUPERTREE algorithm (MC). MC was the first extension of BUILD capable of returning a supertree if the input trees are not compatible. The incompatibilities are resolved by deleting a minimal amount of information present in the input trees in order to allow the algorithm to proceed. This is done by disconnecting a modification of the Build graph, which we call *MC graph*, whenever it consists of only one connected component. In order to decide which edges to delete in the MC graph, the MC algorithm allows the input trees to be weighted. In the simplest case, all trees have unit weight. In the MC graph, an edge between x and y is weighted with the sum of weights of the trees in which x and y occur in a triplet $xy|z$. The algorithm disconnects the MC graph, if necessary, by removing all edges that are contained in at least one minimum cut. For example, the minimum cut in Fig. 2.5 has a cut weight of 1.5, and partitions the graph into $\{a\}$ and $\{b, c, d\}$. To prevent that clades present in all input trees are split, the MC algorithm first contracts edges in the MC graph that are supported by triplets in all input trees.

Semple and Steel [147] also proved that the MC supertree method satisfies several other interesting properties. For example, they show that a relationship exists between the Adams consensus tree and the MC tree in the consensus setting. See Semple and Steel [147] for details.

2.3.3 Modified MinCutSupertree

Page [116] presented a modified version of MC, that uses more information from the input trees. First, Page recapitulates two appealing properties of MC, namely that it is quick to compute and that any nesting found in all input trees will appear in the in the supertree [147]. Then Page shows using a simple example that MC can fail to include information from the input trees resulting in unresolved clades in the supertree due to the following reasons: At least in part the size of the input trees matters and further MC can fail to include information that is not contradicted in the set of input trees. And he concludes "although there is no no consensus method, that can always display all the uncontradicted information in a set of trees [157], it would be desirable to maximize in some sense the amount of uncontradicted information a supertree displays". Using a modified MC graph construction, the Modified MinCutSupertree (MMC) algorithm ensures to incorporate all clades from the input trees with which no single tree directly disagrees.

We omit further details, see [116].

2.3.4 Build-with-distances supertrees

In 2000, Willson [172] presented another extension of Build, the Build-with-distances (BWD) algorithm that, in addition to the branching information in the input trees, uses branch lengths to build the supertree. The method follows the same recursive schema as Build, MC, and MMC: In each iteration, a graph is constructed in which the connected components correspond to the clades of the growing supertree. The main observation underlying the BWD algorithm is that branch lengths may carry phylogenetic information, such as an estimated number of mutations. In a biological application, using branch length is apparently only justified if these are comparable amongst the input trees. For a discussion, see Chapter 6. We briefly recapitulate the BWD algorithm, for a more detailed description, see [172].

For two leaves x, y, the BWD algorithm employs the distance from x to the last common ancestor of x and y, denoted as $\lambda(x, y)$. Note that λ is not symmetric. If more than one input tree contains both x and y, the average of these distances is used.

The graph used by the BWD method, called *BWD graph* in the following, extends the Build graph by additional edges. These edges arise through examination of the branch lengths in the input trees: Two leaves x and y may be in one input tree and x and z are in another input tree, but no input tree contains all three leaves. If $\lambda(x, y) < \lambda(x, z)$, the BWD graph will still

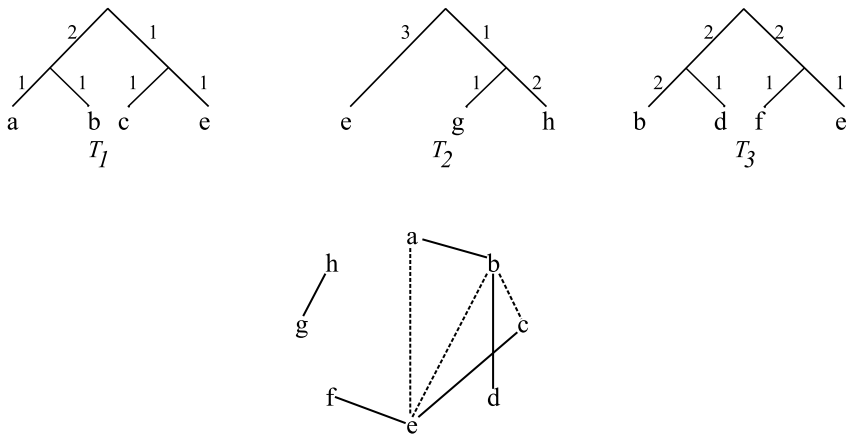


Figure 2.3: Compatible input trees with branch length and the corresponding initial BWD graph. Dashed edges are present in the BWD graph, but not in the corresponding Build graph.

contain an edge $\{x, y\}$. Consider the input trees in Fig. 2.3, showing the initial BWD graph for the three input trees. Dashed edges indicate new edges not present in the Build graph. Consider taxa a and e: The distance $\lambda(e, a)$ in tree T_1 is smaller than the distance $\lambda(e, h)$ in T_2, indicating a triplet $ea|h$ which is not explicitly found in any of the input trees. Hence, an edge $\{a, e\}$ is inserted into the BWD graph.

In case the BWD graph is connected, edge weights are determined using *support functions*, which estimate the evidence that two taxa should be in the same clade of the supertree. Let U be a non-empty subset of $L(\mathcal{T})$, corresponding to the connected component from a previous level of the algorithm. Initially, we have $U = L(\mathcal{T})$. For a triplet $xy|z$ with $x \neq y$, we define the *primary evidence* as $p(x, y, z) := \max\{0, \lambda(x, z) - \lambda(x, y)\}$. In case $\{x, y\}$ or $\{x, z\}$ are not together in an input tree, we set $p(x, y, z) := 0$.

For $x, y \in U$, $x \neq y$, Willson introduced four support functions: The *primary support function* (SP) equals $\max\{p(x, y, z), p(y, x, z) : z \in U\}$, the *confirmed support function* (SC) is $\max\{\min\{p(x, y, z), p(y, x, z)\} : z \in U\}$, and the *accumulated primary support function* (SAP) is $\sum_{z \in U}\big(p(x, y, z) + p(y, x, z)\big)$. Finally, the *accumulated confirmed support function* (SAC) is defined as

$$SAC(x, y, U) := \sum_{z \in U} \min\{p(x, y, z), p(y, x, z)\} \tag{2.1}$$

where again U is the clade from the previous step of the algorithm. In our simulations (see Chapter 4) we find that supertrees built using SAC consistently outperform those built using the other support functions.

Consider Fig. 2.4 for an example. The shown input trees are incom-

patible, and the initial BWD graph consists of one connected component. Also shown is the calculation of $p(x, y, z)$ and $p(y, x, z)$ for $x = a, y = b$ and $z \in U = \{a, \ldots, g\}$. As an example, consider the calculation of $p(a, b, e) = \lambda(a, e) - \lambda(a, b) = 4.7$ and $p(b, a, e) = \lambda(b, e) - \lambda(b, a) = 0.2$. Although no single input tree contains all three leaves, branch lengths from the other input trees suggest the clade $\{a, b\}$ to be nested in clade $\{a, b, e\}$, because $\lambda(a, e)$ in T_3 is larger than $\lambda(a, b)$ in T_1, and $\lambda(b, e)$ in T_2 is larger than $\lambda(b, a)$ in T_1. In this case, $p(a, b, e) = 4.7$ and for the same three vertices $p(b, a, e) = 0.2$. The edge $\{a, b\}$ is weighted by the different support functions as follows: $w_{SP}(a, b) = 6.2$, $w_{SC}(a, b) = 4$, $w_{SAP}(a, b) = 35.4$, and $w_{SAC}(a, b) = 10.3$.

In contrast to the minimum cut approach of MC and MMC, the BWD method uses the *bisection method* to disconnect the BWD graph in case it consists of one component. We determine the minimum threshold θ so that, after removing all edges e with weight $w(e) \leq \theta$ from the BWD graph, the resulting graph is disconnected. Consider Fig. 2.5 for an example. The bisection method returns a threshold of 0.7, edges $\{a, c\}$, $\{c, b\}$ and $\{c, d\}$ are deleted, and the graph is disconnected into $\{c\}$ and $\{a, b, d\}$.

Different from MC and MMC, BWD does not guarantee to return the parent tree in case the input trees are compatible. This behavior is intended, as distance information in the input tree might hint towards incompatibilities not observable in the topological structures of the input trees.

Modifications of the *Build-with-distances* algorithm

The idea behind support functions is that groupings with larger support from the distance information should take precedence over groupings with smaller support [172]. In some sense, the support functions suggested in [172] are conservative: for example, bounding the primary evidence to zero is somewhat arbitrary. To this end, we removed this restriction. In particular, we modify the accumulated primary support function as:

$$SAC_{max} := \sum_{z \in U} \min\big\{p(x, y, z), p(y, x, z)\big\}. \tag{2.2}$$

Note that one can easily construct several modifications of support function considered in [172]. To this end, we also investigated several other such modifications. In our simulations (see Chapter 4), we found that supertrees built using the SAC and SAC_{max} support functions, consistently outperform those built using all other support functions. Hence, we omit further details, and concentrate on these two in our evaluation in Chapter 4.

2.3.5 *PhySIC* and *PhySIC_IST*

Ranwez *et al.* [127] presented another extension of BUILD. Unlike all methods mentioned before, their algorithm *PhySIC* follows a veto philosophy. Following Ranwez *et al.* [127], supertree methods apply either a *voting* or

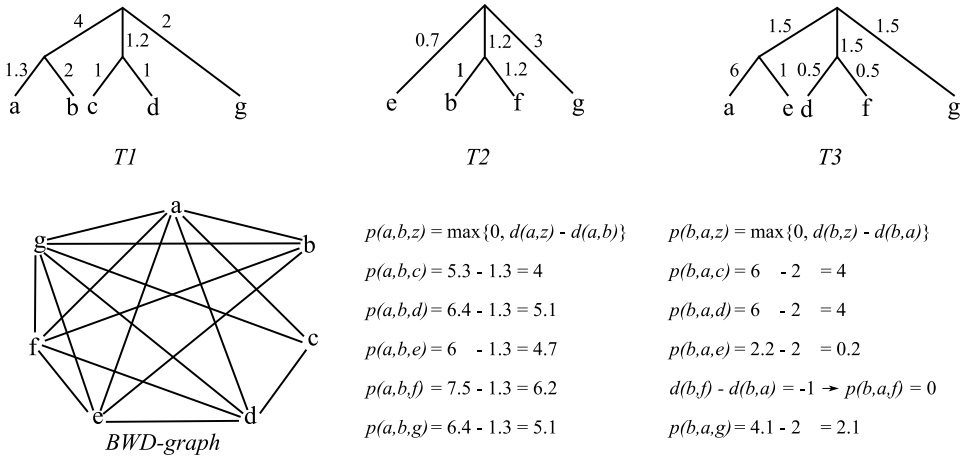


Figure 2.4: A set of incompatible input trees. Also shown is the initial BWD graph of the first iteration of the algorithm and the corresponding calculation of $p(x, y, z)$ and $p(y, x, z)$ for $x = a, y = b$ and $z \in U = L(\mathcal{T})$. Note that the existence of an edge depends on the used support function. For example, edge $\{e, d\}$ is only existent when using SAP or SP.

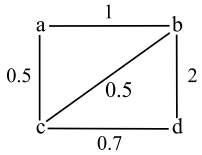


Figure 2.5: A simple weighted graph. The MinCut strategy partitions the graph into $\{a\}$ and $\{b, c, d\}$. The bisection method splits the graph into $\{c\}$ and $\{a, b, d\}$.

veto procedure. A characteristic of the voting approach is that the input trees are asked to vote for clades in the phylogeny to be inferred; the most frequent alternatives are chosen. Voting methods resolve conflicts by using an optimization criterion in order to select between different possible topologies [167]. When input trees conflict, voting methods as MRP and MRF can infer supertrees in which clades are present that are contradicted by each of the input trees [44, 74, 75]. To which extent and under which circumstances these unsupported and undesired novel clades occur, is subject of an ongoing debate: Bininda-Emonds [18] concludes that "although unsupported clades can occur in MRP supertrees [...], their virtual absence in both the simulated and empirical supertrees [...] is encouraging". In contrast, Goloboff [74] examines cases with small input trees on the same set of taxa, where "this situation is, clearly, not very unlikely". In con-

trast to voting methods, the veto approach is more conservative in handling conflicts among the input trees: the inferred supertree has to respect the phylogenetic information of each source tree, and is not allowed to contain any clade that is contradicted by one of the input trees. Thus, conflicts among the input trees are removed [167], by proposing multifurcations in the supertree or by pruning rogue taxa.

Ranwez *et al.* [127] introduced two properties of supertree methods, namely the *non-contradiction property* (PC) and the *induction property* (PI). PC demands that no clade directly contradicting one or indirectly contradicting a combination of the input trees is included into the supertree. PI requires that each clade in the supertree is present in an input tree, or is collectively induced by several input trees. *PhySIC* is a heuristic based on the BUILD method that always returns a supertree satisfying PI and PC simultaneously. First, a supertree is built using a strategy similar to that of MC. Next, two recursive procedures transform this tree such that PC and PI are guaranteed.

Scornavacca *et al.* [146] presented *PhySIC_IST*, a modification of the *PhySIC* algorithm [127], aiming to circumvent a main drawback of veto supertree methods: These tend to return highly unresolved supertrees if the input trees imply a high degree of incompatibility, or do not have a high degree of overlap. To overcome this shortcoming, *PhySIC_IST* modifies the original approach by allowing non-plenary supertrees (that is, supertrees that do not necessarily contain all taxa present in the input trees) while still assuring PC and PI. The rationale behind non-plenary supertrees is that excluding taxa, for which the position greatly differs among the input trees, can enhance the resolution of the supertree. In contrast to the original version, *PhySIC_IST* does not use a graph to guide the supertree construction process. It works in a stepwise fashion, iteratively adding leaves to a starting tree consisting of two nodes. The decision, if and where a leave is added, is based on the informativeness of the proposed, growing supertree. This is measured using a variation of the Cladistic Information Content (CIC) criterion [166], that takes into account both the presence of multifurcations and the absence of some taxa. In addition, *PhySIC_IST* uses a preprocessing step called source tree correction (STC), which analysis conflicting triplets among the input trees. If conflicting triplet topologies for the same three taxa are encountered, STC drops all alternatives but the one that is statistically best supported. To this end, STC modifies the input trees: either multifurcations in the input trees are inserted, or taxa are removed that have highly different positions in the input trees. The extent with which STC corrects the input trees is determined by a user-defined parameter in $[0, 1]$. For parameter value 1, *PhySIC_IST* behaves like a pure veto method, while for parameter value 0 it mimics a voting method. See [146] for further details of the STC preprocessing.

2.4 Other approaches to the supertree problem

There are some methods that are neither matrix representation nor graph-based. Here, we will not cover all such methods but only few examples. In most cases, these formulations of the supertree problem again lead to NP-hard problems. Also, none of these methods was shown to consistently outperform MRP with respect to supertree quality, with the possible exception of Quartet MaxCut.

2.4.1 Strict Consensus Merger (SCM)

The Strict Consensus Merger (SCM) by Huson *et al.* [91] seeks to deliver an analog of the strict consensus of the input trees by computing the strict consensus of pairs of trees at a time. It has the unappealing property that it is dependent on the order of input trees [135]. Furthermore, it tends to produce poorly resolved supertrees, and rather aims at the nodes that are present to be accurate. SCM is part of the Rec-I-DCM3 divide-and-conquer approach [134], see below. We are not aware that this method has been used as a stand-alone supertree method.

2.4.2 Super Distance Matrix (SDM) supertrees.

Basis for the SDM method by Criscuolo *et al.* [47] is the average consensus procedure (ACS) [103]. The first step of ACS is to compute distance matrices corresponding to the path-length in the input trees. After standardizing each input matrix, ACS computes the average of the standardized matrices which is used to build the distance supermatrix. The presented standardization procedure relies on dividing all distances in each matrix by the maximum distance in that matrix. This distance supermatrix is than analyzed using a least-squares method. Similar averaging method are presented to generate the distance supermatrix directly from sequences and gene trees. Other standardizing methods have been explored, but seem to be inaccurate for more than two trees [105]. Similar to ACS, the distance-based method SDM uses a more involved standardization procedure and is able to use both sequences and gene trees as input. See [47] for details. The possibly still incomplete super distance matrix is then processed using MRV* (a variant of Minimum Variance Reduction), BioNJ* or NJ* (variants of Neighbor-Joining) [46].

2.4.3 Quartet MaxCut supertrees

The Quartets MaxCut supertree method by [150] is one of the few examples of a successful supertree method that deals with unrooted input trees. The method combines several heuristic steps, as many NP-hard problems have to be solved along the way; most notably, the Maximum Cut problem. Evaluations by Snir and Rao [150] and by Swenson *et al.* [162] indicate that Quartets MaxCut is on a par with MRP, and sometimes even better. Unfortunately, running times of the method are usually even worse than

for MRP. Furthermore, there is no freely available implementation of the method.

2.4.4 The SuperTriplets method

SuperTriplets by Ranwez *et al.* [128] combines an initial heuristic search, with a local search to find the asymmetric median supertree under the triplet distance. The first, heuristic step is similar in spirit to agglomerative clustering, and runs in polynomial time. For solving the second, computationally hard step, SuperTriplets uses a branch swapping heuristic. Supertrees from the first heuristic step are not reported in [128], so we can only assume that these supertrees are of rather bad quality. Hence, the good quality of SuperTriplets supertrees can be attributed mostly to the local search heuristic.

2.5 Supertrees in a divide-and-conquer strategy

A variety of supertree methods exist, and we have not covered all of them; see for example [7, 12, 20, 45, 108, 111, 112, 155] for other approaches. Most approaches for supertree reconstruction lead to computationally hard problems. The most widely used supertree method is still Matrix Representation with Parsimony (MRP), some 20 years after it has been proposed; this underlines that no other method performed significantly better in terms of supertree quality, and most performed much worse, see for example [102, 162]. For most graph-based methods like MC and MMC it has been shown that on the one hand they provide a much lower running time compared for example to MRP, but, on the other hand result in supertrees of inferior quality, see for example [56, 102].

In Chapter 1, page 17, we introduced the divide-and-conquer approach to phylogenetics, in which supertree construction is subsumed within supermatrix analysis, as a promising future development for large-scale phylogenetic inference.

The divide-and-conquer approach in phylogenetic inference has been implicitly pioneered by Strimmer and von Haeseler [159] back in 1996 with their quartet puzzling method: Here, the smallest possible subset of four taxa are used to construct quartet trees, and these quartet trees are combined to derive the global phylogeny, weighting the quartet trees by their support from the data. Although the apparently good performance has been combined with an existing maximum likelihood-based implementation in the software TREE-PUZZLE [145], quartet puzzling has not yet found a widespread use in the phylogenetic community. The same holds for the recently presented Quartets MaxCut method by Snir and Rao [150], a variant of quartet puzzling.

The possible amalgamation of the supertree and the supermatrix framework in a divide-and-conquer strategy is possibly best exemplified by the Recursive-Iterative-Disk-Covering Method 3 (Rec-I-DCM3) from Roshan *et al.* [134]. Here, disk-covering [91, 92] is used to choose optimal sub-

problems from a large input matrix, which are analyzed using conventional reconstruction methods. The subproblem solutions are combined using the Strict Consensus Merger (SCM) [91] supertree method, see also page 34. Finally, the supertree is refined based on the original input data matrix. Thus, the supertree is not an overall end result and rather functions as input for the subsequent supermatrix analysis. Regarding speed and accuracy, Rec-I-DCM3 appears to be especially effective in a maximum parsimony framework [135], whereas in a maximum likelihood results are not likewise improved [25]. However, as in the case of quartet puzzling, Rec-I-DCM3 is not yet established as a standard procedure for large scale phylogenetic inference.

Whereas the prospects of divide-and-conquer strategies in phylogenetics appear promising, this approach has not yet proven its usefulness in phylogenetic analysis. Bininda-Emonds [23] notes that "the divide-and-conquer approach has yet to realize its theoretical advantages (in terms of both speed and accuracy) over more conventional heuristic search strategies." Furthermore, Bininda-Emonds [23] formulates three goals for a supertree method to be useful in a divide-and-conquer strategy:

1. It is important "that the supertree method being used is fast given that it too has to confront a problem comprising all taxa in the original data set."

2. Speeding up the final step of the analysis that uses the entire data matrix and that is computationally very challenging, requires "a more accurate starting tree for the analysis to build on."

3. The output supertree has to be reasonably resolved, as "a supertree that is too poorly resolved [...], even if it is accurate, will not realize sufficient time savings for the subsequent global analysis."

At present, we are not aware of any supertree method that meets all three requirements simultaneously.

In summary, the divide-and-conquer approach has yet to prove its usefulness for building phylogenetic trees. Here, the limiting factor appears to be the non-existence of a supertree method that is both swift and accurate.

In the next chapter, we will introduce our new supertree method Flip-Cut supertree, that was developed with this set of issues in mind.

CHAPTER 3

FlipCut: A new supertree method

Supertree methods could play a key role in heuristic divide-and-conquer search strategies for large molecular supermatrices in the attempt to reconstruct larger parts of the Tree of Life. But for a truly effective and competitive divide-and-conquer approach, especially fast *and* accurate supertree methods are necessary. These requirements are not fulfilled in satisfactory manner by any supertree method so far.

In the following, we present a novel algorithm for the computation of supertrees called FlipCut supertree. Our method combines the computation of minimum cuts from polynomial-time graph-based supertree methods (see Chapter 2, page 28 ff.) with a matrix representation method, namely Matrix Representation with Flipping (MRF), as introduced in Chapter 2, page 25. Using a graph as guiding structure, which maps the matrix representation of input trees, FlipCut supertree constructs the supertree top-down, minimizing the number of required 0/1-flips necessary to deal with incompatibilities between the input trees in each step. Thus, our method is a heuristic search for a minimum set of 0/1-flips such that the resulting matrix admits a phylogeny. The running time of our algorithm is comparable to that of the MinCut algorithm and similar graph-based supertree methods: For n taxa and m internal nodes in the input trees, the running time is $O(mn^3)$. Our results can be interpreted in the sense that we try to minimize a global objective function, namely the number of flips in the input matrix.

This chapter is organized as follows: First, we introduce the *incomplete directed perfect phylogeny* problem and recapitulate an algorithm introduced by Pe'er *et al.* [117] to solve it, on which our approach is based. In Section 3.2, we present our new method, FlipCut supertree. In Section 3.3 we extend our approach by using branch lengths from the input trees. In Section 3.4 we introduce a preprocessing step for our algorithm aiming to reduce a seemingly rare but still undesirable effect of the used objective function. Finally, we provide some implementation details in Section 3.5.

3.1 Incomplete directed perfect phylogeny

Before we introduce the incomplete directed perfect phylogeny problem, we reconsider the complete directed version introduced on page 10 in Chapter 1. The question if a complete binary matrix M admits a perfect phylogeny can be seen to be equivalent to a compatibility problem (see Theorem 1, page 10), a graph-theoretical problem or a matrix avoiding problem. To see that, we need further definitions of which we will use several to introduce our new method FlipCut supertree. The following definitions are partly

modified versions of those given by Pe'er *et al.* [117].

Definition 3.1. Let $S = \{t_1, ..., t_n\}$ be the set of taxa and and $C = \{c_1, ..., c_m\}$ the set of characters.

- The *FlipCut graph* $G(M)$ of a given character-state matrix M is the bipartite graph $G(M) = G(C, S, E)$ where $E = \{t_i c_j | M[t_i, c_j] = 1\}$. Given G, we will refer to C as the set of *character vertices* and to S as the set of *taxa vertices*.
- The Σ *subgraph* is an induced path of length four in $G(M)$.
- $G(M)$ is called Σ-free if there is no induced path of length four in $G(M)$ starting and ending at a taxon vertex.

The FlipCut graph $G(M)$ maps the matrix representation of trees (see Chapter 2, page 24) into a graph structure. Fig. 3.1 shows a Σ-graph and its corresponding submatrix.

	c_1	c_2
t_1	1	0
t_2	1	1
t_3	0	1

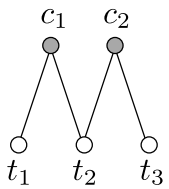


Figure 3.1: A Σ-graph and its corresponding submatrix.

For completeness and a better understanding of the following, we also recapitulate the following two definitions from Pe'er *et al.* [117], although we will not directly use them in context of our new method FlipCut supertree.

Definition 3.2. A matrix A is said to *avoid* a matrix B, if B is not equal to any submatrix of A.

Definition 3.3. A binary matrix A is called canonical if it can be decomposed as follows:

- The leftmost $k_0 \geq 0$ columns are all zero.
- The next $k_1 \geq 0$ columns are all one.
- There exist canonical matrices $A_1, ..., A_n$ such that A is of the form shown in Fig. 3.2.

With these definitions, we can now give several equivalent characterizations of when the complete directed perfect phylogeny problem is solvable. A proof of the following theorem can be found in [117].

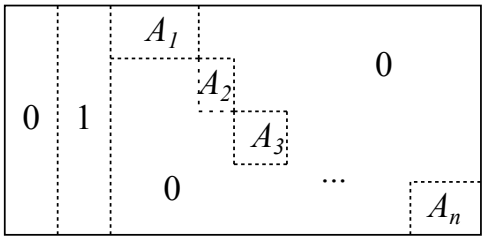


Figure 3.2: A canonical matrix, where the submatrices are defined recursively.

Theorem 2. Let $S = \{t_1, ..., t_n\}$ be the set of species, $C = \{c_1, ..., c_m\}$ the set of characters an $n \times m$ matrix M be a complete binary character-state matrix. Then the following statements are equivalent:

(i) M has a phylogenetic tree.

(ii) The 1-sets of every two characters $c_i, c_j \in C$ are compatible.

(iii) $G(M)$ is Σ-free.

(iv) Every ordering of the rows and the columns of M results in a matrix that avoids the matrix

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

(v) There exists a permutation of the rows and columns of M such that the resulting matrix is a canonical matrix.

(vi) There exists a permutation of the rows and columns of M which yields a matrix avoiding the following matrices:

$$A_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix},\ A_3 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix},\ A_4 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

We now consider the *incomplete directed perfect phylogeny problem*: We are given an incomplete binary $n \times m$ matrix M with character-states $\in \{0, 1, ?\}$, in which '?' indicates that the character-state at a given position is undetermined. Such a matrix can arise in different context: in the supertree context through the matrix encoding of phylogenetic trees with an overlapping set of taxa (see Chapter 2, page 24), or if certain types of biological data are used to reconstruct phylogenies (e.g. SINEs, see [117]).

As in the complete directed version of the problem, we assume that the characters are directed and that each transition from '0' or '1' happens at

most once in the tree: an invented character state never disappears and is never invented twice. We now ask if M allows for a perfect phylogeny, where '?'-entries can be arbitrarily resolved to '0' or '1'. The incomplete directed perfect phylogeny problem solves the question if, for an incomplete matrix M, there exists a compatible completion (see Chapter 2, page 25).

Note that the statement analogous to *(ii)* from Theorem 2 does not hold in case of incomplete data. Even if every two characters are compatible and thus admit a phylogenetic tree, the full matrix might not have one. See Fig. 3.3 for an example.

	c_1	c_2	c_3
t_1	1	0	0
t_2	1	1	0
t_3	?	1	1
t_4	0	?	1

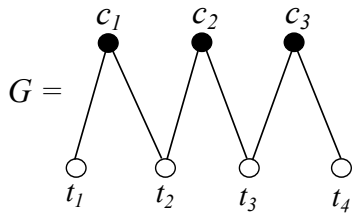


Figure 3.3: *Left*: Pe'er *et al.* [117]. An incomplete matrix M which has no phylogenetic tree although every pair of its columns has one. *Right*: The corresponding FlipCut graph $G(M)$. Note that $G(M)$ is connected if M does not allow an incomplete directed perfect phylogeny.

Before we present the algorithm from Pe'er *et al.* [117], called IDPsolver in the following, we need one further definition.

Definition 3.4. A character vertex $c \in C$ in the FlipCut graph G is *semiuniversal* if $M[t, c] \in \{1, ?\}$ holds for all $t \in S$.

The pseudo-code of IDPsolver is shown in Fig. 1. After the FlipCut graph G is constructed from the input character-state matrix, all semiuniversal character vertices are immediately removed as all '?'-entries can be resolved to '1' without flipping [117]. If $G(M)$ is connected, the algorithm has no information on how to build the clades of the parent tree for the input character-state matrix M, and stops. Otherwise, the taxa set of each connected component is returned. If the incomplete input matrix allows for a parent tree, the taxa sets returned by IDPsolver correspond to a hierarchy defining this phylogeny.

For a given set of rooted trees $\mathcal{T}$, Pe'er *et al.* [117] showed that this algorithm can be implemented in $O\left(mn \cdot \log^2(m+n)\right)$, where m is the total number of non-root inner vertices in $\mathcal{T} = |C|$ and $n = |L(\mathcal{T})| = |S|$.

3.2 The FlipCut algorithm

We now show how to apply the idea of finding minimum cuts to the algorithm IDPsolver from Pe'er *et al.* [117]. Our method implicitly employs the Matrix Representation with Flipping (MRF) framework to deal with

Algorithm 1: IDPSOLVER

Input: A set $S \subseteq \{t_1, \ldots, t_n\}$ of taxa, a set $C \subseteq \{c_1, \ldots, c_m\}$ of characters, an $n \times m$ matrix $M \in \{0, 1, ?\}$.
Output: a hierarchy on S corresponding to a perfect phylogeny T for M, if existent.

```
1 Output taxa set S.
2 if |S| = 1 then return.
3 Construct FlipCut graph G := G(S, C, E) from M, immediately
  remove all semiuniversal character nodes from G.
4 if G is connected then return False and halt.
5 else
6     foreach connected component of G with vertex set S', D' of G do
7         IDPSOLVER(S', D', M)
8     end
9 end
```

incompatibilities between the input trees: we assume that conflicts between input trees correspond to *errors*. The notion of error here relates to taxa that are present or absent in a clade where they should not be, making the input trees incompatible. Such errors correspond to "flips" from $0 \to 1$ or $1 \to 0$ in the matrix representation of the input trees [40], which prevent the matrix from representing any phylogenetic tree perfectly.

Recall that the MRF problem has been investigated in many algorithmic aspects (parameterized and approximation algorithms, ILP, data reduction, see Chapter 2, page 26). From an algorithmic standpoint, the "last resort" appears to be a heuristic. To this end, we have developed the FLIPCUT supertree method which we will explain in the following.

As in the original version of the IDPSOLVER algorithm, we immediately remove all semiuniversal character vertices from the FlipCut graph $G(C, S, E)$, as all '?'-entries can be resolved to '1' without flipping [117].

If G is connected, IDPSOLVER terminates as there is no perfect phylogeny resolving M. Consider the FlipCut graph G for the incomplete character-state matrices in Fig. 3.3 and Fig. 3.4 .

Assume that $G(C, S, E)$ is connected at some point of the algorithm — how can we disconnect the graph by means of modifying the input matrix M? Obviously, it does not help to insert new edges in $G(C, S, E)$. Removing an edge tc from $G(C, S, E)$ can be achieved by two different operations: either flip $M[t, c]$ from '1' to '0', or make character c semiuniversal by flipping all entries satisfying $M[t', c] = 0$ to '1', for $t' \in S$. Recall that any semiuniversal character c is deleted immediately, resulting in the deletion of *all* edges incident to c. This comes at the cost of $w(c) := \#\{t \in S | M[t, c] = 0\}$ flips

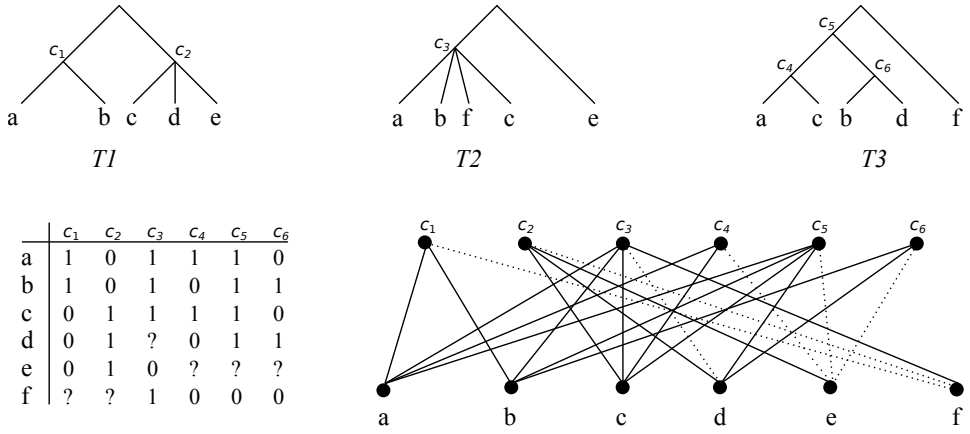


Figure 3.4: An example illustrating the initial step of the FlipCut algorithm. Three incompatible input trees, a matrix M encoding the input trees and the corresponding FlipCut graph $G(M)$ are shown. Dotted lines in G correspond to '?'-entries.

in the matrix. To disconnect $G(C, S, E)$ we can use an arbitrary combination of these edge deletion operations.

Formally, we assume all edges in $G(C, S, E)$ to have unit weight, and that each character vertex c has weight $w(c)$. The weight of a bipartition of taxa vertices is the minimal cost of a set of edge and vertex deletions, such that the two subsets of taxa vertices lie in separate components of the resulting graph. We search for a bipartition of minimal weight.

Clearly, this problem is closely related to finding minimal cuts in an undirected graph. For the later problem, numerous efficient algorithms have been developed in the last years [30, 94]. Unfortunately, there exist two important differences here: First, we are not searching for an arbitrary cut in the graph $G(C, S, E)$ but instead require that the set of taxa vertices is partitioned. Second, these algorithms do not allow us to delete vertices. We conjecture that the first modification is relatively easy to overcome. However, it is not obvious how to include vertex deletions in these algorithms.

To this end, we drop back to an older approach for finding minimum cuts: We fix one taxon vertex $t_1 \in S$, and for all other taxa vertices $t_i \in S$ we search for a minimum t_1-t_i-cut, allowing vertex deletions. Among these cuts, the cut with minimal weight is the solution to the above problem. To find a minimum t_1-t_i-cut with vertex deletions, we transform $G(C, S, E)$ into a directed bipartite network $N(C', S, A)$ with capacities: Each taxon vertex $t \in S$ is also a vertex in the network, each character vertex $c \in C$ is transformed into two vertices $c_- \in C'$ and $c_+ \in C'$ plus an arc (c_-, c_+) in the network, and an edge tc in $G(C, S, E)$ is transformed into two arcs (t, c_-)

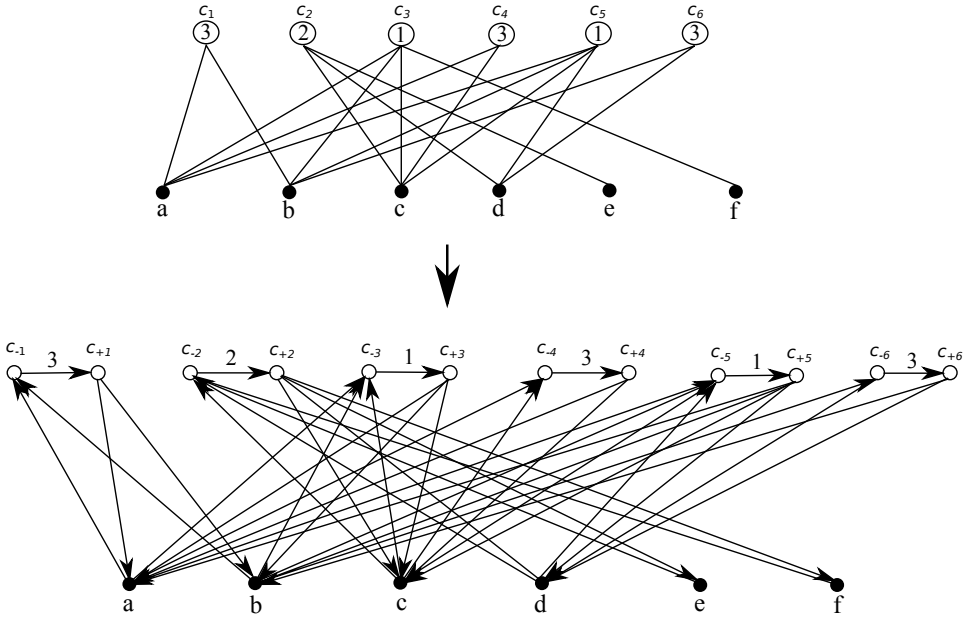


Figure 3.5: Workflow of the FlipCut algorithm. The FlipCut graph $G(C, S, E)$ with vertex deletion costs from Fig. 3.4 and the corresponding network $N(C', S, A)$ are shown.

and (t, c_+) in the network. Arcs (c_-, c_+) have capacity $w(c)$, all other arcs have unit capacity. By the generalized max-flow min-cut theorem, finding a minimum cut in $G(C, S, E)$ is equivalent to computing a maximum flow in the network $N(C', S, A)$ [67]. Note that for all taxa vertices in C', the maximum t_1-t_i-flow in $N(C', S, A)$ equals the maximum flow between the taxa in C'. See Fig. 3.5 for an example. We reach:

Lemma 3.1. Let $S \subseteq \{t_1, \ldots, t_n\}$, $C \subseteq \{c_1, \ldots, c_m\}$ the set of characters and $M \in \{0, 1, ?\}^{m \times n}$. We construct the network $N := N(C', S, A)$ for the input matrix M. The minimum number of 0/1 flips required in M to disconnect the induced FlipCut graph $G(C, S, E)$ equals the minimum cost of a minimum t_1-t_i-cut in the network N, over all $i = 2, \ldots, n$.

We now proceed in a recursive top-down procedure to construct the supertree, similar to the approaches of Page [116], Semple and Steel [147], Willson [172] (see Chapter 2, page 28 ff.), and Pe'er *et al.* [117]. The pseudocode of our algorithm is depicted in Fig. 2; we initially call the procedure as FlipCut(S, C, M). The subsets $S \subseteq \{t_1, \ldots, t_n\}$ that are output during the course of the algorithm form a hierarchy corresponding to the desired supertree.

As the algorithm employs the algorithm of Pe'er *et al.* [117] in case the

Algorithm 2: FlipCut

Input: A set $S \subseteq \{t_1, \ldots, t_n\}$ of taxa, a set $C \subseteq \{c_1, \ldots, c_m\}$ of characters, an $n \times m$ matrix $M \in \{0, 1, ?\}$.
Output: a hierarchy on S corresponding to the FlipCut supertree

```
1  Output taxa set S
2  if |S| = 1 then return
3  Construct FlipCut graph G := G(C, S, E) from M, immediately
   remove all semiuniversal character nodes from G.
4  if G is connected then
5      Construct weighted network N form G
6      for i ← 2 to n do
7          Find maximum t_1- t_i-flow in N.
8          if maximum t_1-t_i-flow is lighter than current minimum cut
           then
9              Construct cut-of-the-phase in G from maximum flow in N
10             Store cut-of-the-phase as the current minimum cut
11         end
12     end
13     Remove current minimum cut from G
14 end
15
16 foreach connected component of G with vertex set C', S' of G do
17     FlipCut(C', S', M)
18 end
```

input trees are compatible or, equivalently, in case the input matrix allows for a perfect phylogeny without flipping, we infer:

Lemma 3.2. *If the input matrix M allows for a perfect phylogeny without flipping, then the* FlipCut *algorithm returns the perfect phylogeny tree.*

What is the running time of the above algorithm? At most $n - 1$ minimum cuts have to be computed in total, as this is the number of inner nodes in the resulting phylogenetic tree. We reach a running time of $O(n \cdot T(m, n))$ where $T(m, n)$ is the time required for computing all maximum t_1-t-flows in the networks $N(C', S, A)$ with at most m character vertices and n taxa vertices. The running time is dominated by the algorithm we use for constructing maximum flows. For a network $N = (V, A)$, Hao and Orlin [84] compute maximum flows from one source to all other vertices in $O\big(|V| \cdot |E| \cdot \log(|V|^2 / |E|)\big)$ time, using the maximum flow algorithm of Goldberg and Tarjan. For a bipartite graph with vertex set $V_1 \cup V_2$ and $|V_1| \leq |V_2|$, running time can be improved to $O\big(|V_1| \cdot |E| \cdot \log(|V_1|^2 / |E|)\big)$ [84]. Our net-

works $N(C', S, A)$ are bipartite and have $O(n + m)$ vertices and $O(mn)$ edges, and we may assume $n \leq m$. So, a minimum cut with vertex deletions in $G(C, S, E)$ can be computed in $O(mn^2)$ time. We infer:

Lemma 3.3. *Given an $n \times m$ input matrix M over $\{0, 1, ?\}$ for n taxa and m characters, the* FlipCut *algorithm computes a supertree in $O(mn^3)$ time.*

As presented here, the FlipCut algorithm may compute different solutions for the same input: This is because there can be several co-optimal minimum cuts, and our algorithm arbitrarily chooses one of these cuts. We can solve this by removing all edges and vertices that are part of *at least one* minimum cut, similar to the MinCut algorithm [147]. We do not have to enumerate all such minimum cuts, since it can be a hard problem to find these edges and vertices [119]. In the following, we ignore this: By using the branch lengths from the input trees, we weight all edges in the FlipCut graph with real numbers, so the existence of several minimum cuts of identical weight is practically impossible.

Steel *et al.* [157] list five desirable properties of a supertree method (see Chapter 2, page 28), and it is easy to see that the FlipCut algorithm satisfies three of them: If the input trees are compatible then the supertree is a parent tree (Lemma 3.2); the supertree can be computed in polynomial time (Lemma 3.3); and no species is missing from the supertree. By using the approach indicated in the previous paragraph, changing the order of input trees does not change the resulting supertree; and that relabeling the input species results in the same supertree with correspondingly relabeled species. Hence, the modified FlipCut algorithm satisfies all five desirable properties from [157].

3.3 Using branch lengths

We use branch lengths in a straightforward fashion: We weight each column of the matrix by the length of the branch that was responsible for generating the column. This can be easily incorporated into the FlipCut graph, by weighting edges and character vertices. This way, flipping an entry is cheaper for those branches that are possibly wrong, and harder for those branches that are most likely true.

In our evaluations (see Chapter 4), a different weighting called "Edge & Level" showed a better performance: Each character vertex c corresponds to an internal edge $e = (u, v)$ in one of the input trees, inducing the corresponding column in the matrix M. We set the weight of character c and, hence, the corresponding column in M to $l(c) := w(e) \cdot \mathit{depth}(v)$. Here, $l(c)$ is the length of branch e, and $\mathit{depth}(v)$ is the *number* of edges on the path from the root to v in the input tree.

The "Edge & Level" weighting does not cover all the information encoded in the input trees: As suggested by Willson [172], if taxa t, t' have a last

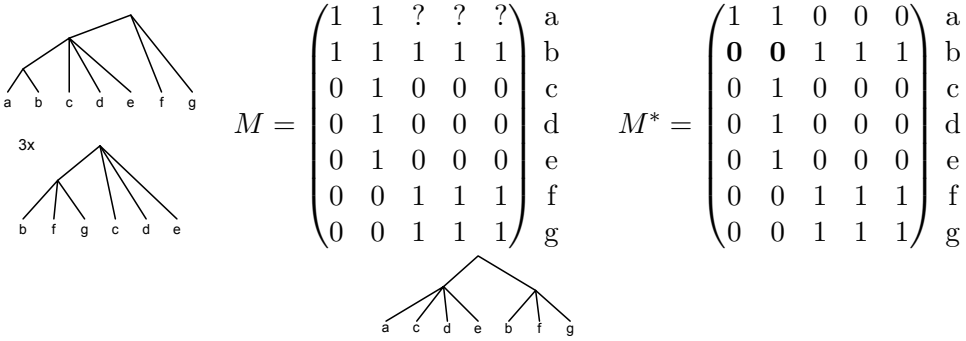


Figure 3.6: The undisputed sibling problem. Left: Four input trees, the lower tree appears three times in the input. Taxon 'a' appears only in the upper tree, and is a sibling of 'b'. Middle left: The resulting matrix representation M. Middle right: The optimal MRF solution M^* with two flips. Right: The solution tree for M^*. Taxa 'a' and 'b' are no siblings in this tree.

common ancestor much more recent than t, t'' then, even if this is observed in different input trees, we can still infer that t, t' are more closely related to each other than to t''. In the future, we will investigate how to introduce this concept into the FlipCut framework, as better weightings will surely improve the performance of our method.

3.4 The undisputed sibling problem

Given a set of input trees, assume that some taxon x appears as a sibling of another taxon y in all input trees in which it is present. In other words, for all trees where x is present, we also find y, and both are siblings. We call such x an *undisputed sibling*. It is reasonable to assume that x is also a sibling of y in the supertree, possibly accompanied by other siblings. Unfortunately, FlipCut Supertree does not necessarily enforce this: Minimizing the number of flips, it is sometimes cheaper to separate x and y. This is a seemingly rare but still undesirable effect of this objective function, see Fig. 3.6 for an example.

We stress that for practically every known supertree method, there exist certain pathological examples where the supertree method will produce unexpected results. This is generally no indication that the algorithm does not work well in practice. Instead, we see this as a possibility to improve upon the method.

To counter the above effect, we use a data reduction rule that is applied to all input trees before we compute FlipCut supertrees: If there is an undisputed sibling x of y, then remove x from all input trees. We repeat this until we find no more undisputed siblings. Note that by removing an

undisputed sibling, we might produce new undisputed siblings. After we have computed the supertree, we re-insert all undisputed siblings in reverse order. If y has more than one undisputed sibling at the same time, we re-insert all siblings in one node, resulting in a polytomy in the supertree.

There exist two possibilities to remove an undisputed sibling x: Either we simply delete x from the input trees, resulting in a deletion of row x from the input matrix, and subsequent deletion of all columns that have only a single '1'-entry. Or we decide to add up the weight of x and y in those trees where x is removed. In the matrix, we then treat 0/1-entries to be weighted by a positive integer. In our implementation, we use the first variant.

After running the FlipCut supertree algorithm, we re-insert all removed undisputed siblings in reverse order. We have to make sure that no superfluous internal edges are inserted when two or more siblings are re-inserted to the same taxon of the supertree.

One can easily implement the undisputed sibling preprocessing to run in $O(ln^2)$ time for l input trees: We iterate over all trees. For each tree T_i we do a tree traversal to store the sibling of each taxon in an array. This is done for those taxa that have a sibling in T_i. If we encounter a conflict, we label the taxon accordingly. For all taxa that do not have a conflict at this stage, we iterate over all trees again, and check whether the sibling is not present in trees where it is not marked as such. If a taxon passes this test, it is an undisputed sibling and can be merged. We repeat this until we find no more undisputed siblings.

3.5 Implementation

We implemented the FlipCut algorithm and the undisputed sibling preprocessing in Java as part of the EPoS framework [81] (see Chapter 5). In order to illustrate the influence of branch length on our approach, we implemented two different weighting schemes for edges and character vertices in the graph model. In the "Unit Cost" weighting scheme branch lengths are ignored. Here, the cost of deleting an edge is one, and the cost of deleting a character vertex c is the number of zeros in the corresponding column in matrix M. Furthermore, we implemented the "Edge & Level" weighting scheme as explained in Section 3.3.

CHAPTER 4

Polynomial Supertree Methods Revisited

As an increasing number of supertree methods is available, studies using either simulated or empirical data are needed that compare behavior and performance of the methods under various conditions.

In simulation studies results of different methods can be compared to a known model tree and thus the methods can be compared at an absolute scale. Empirical datasets on the other hand usually offer a more realistic setting, however, the true tree is usually not known.

Several studies comparing the supermatrix approach and supertree methods using empirical data sets have been carried out, see for example Baker *et al.* [6], Dutilh *et al.* [53], Fitzpatrick *et al.* [66], Gatesy *et al.* [72], Salamin *et al.* [137] and Buerki *et al.* [34]. To evaluate the performance of methods using of empirical data, the constructed supertrees can be scored using a generally accepted reference tree (e.g. [53]), the supertrees can be compared among each other (e.g. [6]), or the amount of agreement between supertrees and input trees can be measured (e.g. [34]).

In simulation studies, supertrees are compared to a known model tree using distance or similarity measures. Over the years, a couple of simulations focusing on different aspects of the investigated supertree or supermatrix methods have been published, see for example Bininda-Emonds [18], Bininda-Emonds and Sanderson [24], Chen *et al.* [38], Eulenstein *et al.* [56], Kupczok *et al.* [102], Levasseur and Lapointe [107] and Swenson *et al.* [161].

In this chapter, we present simulations based on two different data sets, explained below. As outlined at the end of Chapter 2, a main factor impeding a breakthrough of divide-and-conquer strategies in large-scale phylogeny inference appears to be the non-existence of a supertree method that is both, swift and accurate. For this reason, our simulations focus on the polynomial-time supertree methods outlined in Chapter 2. In particular, we want to study the effects of the continuing improvement in the class of graph-based supertree approaches. Furthermore, we included other promising polynomial, non graph-based supertree approaches in our simulations.

We evaluate the accuracy and the resolution of the polynomial-time supertree methods MinCutSupertree (MC), Modified MinCutSupertree (MMC), Build-with-distances (BWD) (including our modification, see Chapter 2, page 29 and following), *PhySIC*, *PhySIC_IST*, *super distance matrix (SDM)*, SuperTriplets (ST) and our new method FlipCut. The results of these methods are compared to representatives from matrix representation based supertree methods, namely the still most widely used supertree

method, matrix representation with parsimony (MRP) and matrix representation with flipping (MRF). With respect to the methods we consider in our simulations, former studies indicate that no other supertree method performed significantly better than MRP in terms of accuracy. It has been shown that MRF is sometimes on a par with MRP regarding the quality of reconstructed supertrees [40]. Due to the NP-hard nature of the underlying problems, MR methods tend to require long running times. In comparison, graph-based methods (e.g. MC and MMC) are swift but usually perform worse concerning supertree accuracy (e.g. [102, 162]) and, in case of *PhySIC* and *PhySIC_IST*, produce possibly less resolved supertrees. To the best of our knowledge, Build-with-Distances (BWD), the first graph-based method that uses branch-length information from the input trees to construct the supertree, as well as our new FlipCut supertree method have not been investigated in any simulation study. Furthermore, the methods we consider here have not yet been investigated on the same dataset such that direct comparisons between the methods were not possibly. By considering the accuracy and the resolution of the supertrees (as well as other features, explained in the text), our findings not only illuminate the trade-off between accuracy and running time in supertree construction, but also pros and cons of voting and veto approaches.

We used two different data set in our simulations: Our first data set was generated according to the standard protocol to assess the performance of supertree methods (e.g. used in [18, 24, 56, 107]): (1) Construction of a model tree under a Yule process, (2) simulation of DNA alignments along that tree, (3) random deletion of a proportion of taxa, (4) reconstruction of trees by ML. Afterwards, the inferred trees are used as input for all supertree methods under consideration and the resulting supertrees are compared to the model tree using distance and similarity measures. The second data set was generated using the SMIDgen protocol, described in [161]. Compared to previous protocols, this protocol is said to better reflect data collection processes used by systematists when gathering empirical data, including creation of densely-sampled clade-based trees as well as sparsely-sampled scaffold trees. This chapter is organized as follows: In Section 4.1, we explain the generation of the data sets and the supertrees in detail. In the following, we present how we measured accuracy and resolution. Then, in Section 4.2 we present the results for both data sets.

4.1 Simulation Setting

In the following, we give a detailed explanation how the standard protocol and the SMIDgen data were generated.

4.1.1 The Standard Protocol Data Set

We first present the layout of the simulation protocol used to generate our first data set. Here, we evaluate the accuracy and resolution of the methods

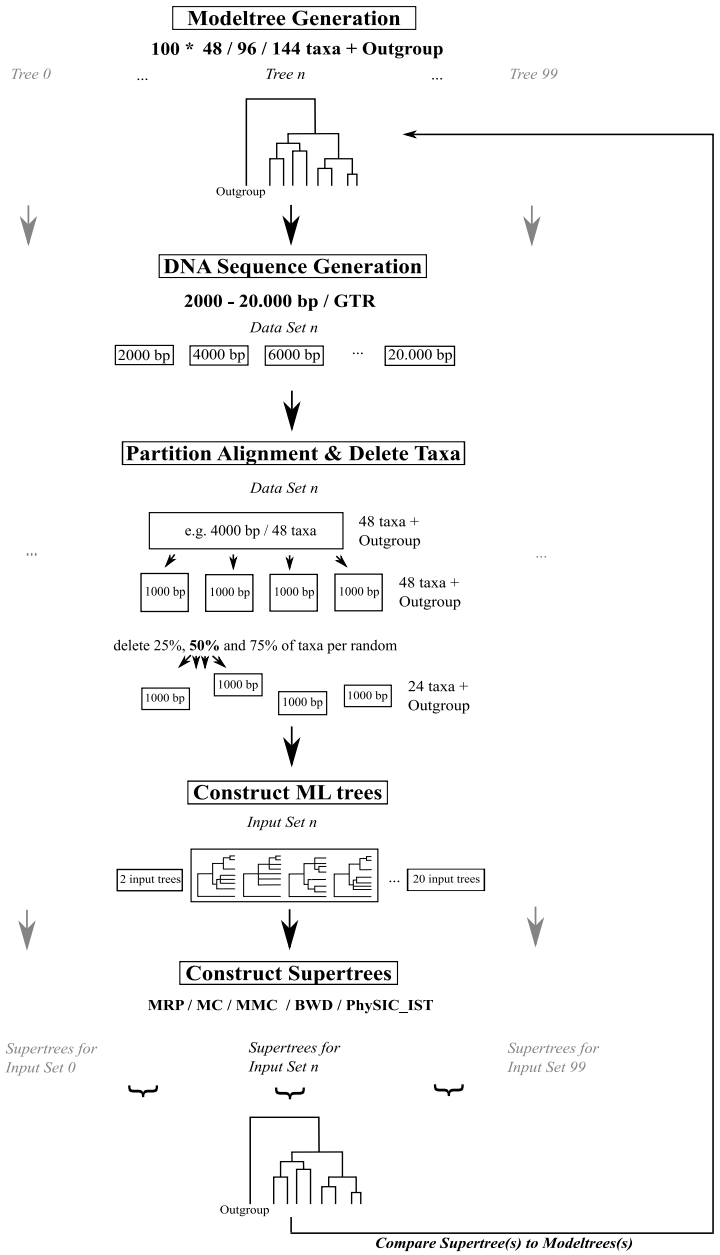


Figure 4.1: Simulation Pipeline Overview

MRP, MRF, MC, MMC, *PhySIC*, *PhySIC_IST*, BWD (including our new support function), SDM, super triplet (ST), and FlipCut supertree. An overview of the simulation layout can be found in Fig. 4.1. Each step is described in detail below.

Generating Model Trees and DNA Sequences

We generated model trees according to a stochastic Yule birth process using the default parameters of the YULE_C procedure from the program r8s [139], with either 48, 96, 144 or 524 taxa. To deviate branch lengths on the trees from molecular clock, branch-specific rates of evolution were determined by drawing random normal variates (mean of 1.0 and standard deviation of 0.5, truncated outside of [0.1, 2.0]) and multiplying by an overall tree-wide rate of substitution. These branch-specific rates are used to determine the branch-length by multiplying them with branch-durations obtained from the Yule process model. In each tree we set an additional outgroup since most of the supertree methods under investigation can only handle rooted trees. To determine the branch length for the outgroup, we proceeded as following: First, the taxon with the largest distance (d_{max}) to all other taxa is identified. If this distance exceeds 75% compared to all other distance relations, we shortened it accordingly. Then, we added a outgroup taxon with a branch length to the root that equals $1.25 \times d_{max}$. For model tree sizes of 48, 96, 144 we generated 100 different model tree replicates and ten in case of 525 taxon model trees. The smaller number of replicates for 525 taxa model trees was due to the longer running times for this data set. Using Seq-gen v1.3.2 [126], nucleotide sequences were simulated along each of the model trees according to the general time reversible process (GTR) model [174] with parameters Lset Base = $(0.3468, 0.3594, 0.0805)$, Rmat = $(0.6750, 27.9597, 1.1677, 0.4547, 20.8760)$, gamma rate heterogeneity $\alpha = 1.1999$ and PINVAR = 0.4954, taken from [88]. After sequence generation, we checked if the outgroup sequence has a larger distance to all other sequences than any two other sequences among each other. For each model tree we generated sequences ranging from 2000 to 20 000 base pairs in steps of 2000, yielding ten different "multiple sequence alignments" per model tree.

Generating Input Trees

The models of sequence evolution implemented in Seq-Gen assume evolution is independent and identical at each site. Hence, we can partition the "multiple sequence alignment" with 2 000–20 000 bases blocks, representing independent datasets. We chose an outgroup sequence/taxon to root the trees generated by Maximum Likelihood. In case of 48, 96, 144 taxon trees, we partitioned each alignment into blocks of 1000 base pairs each. From each alignment block, we randomly deleted 25 %, 50 %, or 75 % of the sequences/taxa, to simulate different taxa overlaps observed in real datasets.

For each resulting alignment block, we inferred a maximum likelihood tree using RAxML v 7.0.0 [153] with default parameters. This results in instances with 2 to 20 input trees corresponding to the same model tree. Summarizing, we have three different numbers of taxa; three different deletion ratios; and ten different input sizes. For each of these parameter combinations, we generated 100 instances. In case of 524 taxon model trees, we proceeded slightly differently from the described procedure: here, each alignment was partitioned into blocks of 500 base pairs, and 50 % or 75 % of the sequences/-taxa were deleted. Again, a maximum likelihood tree was inferred for each resulting alignment block. In contrast to the 48, 96, 144 taxon model tree data sets, this results in instances with 4 to 40 input trees.

4.1.2 The SMIDgen data set

For further simulations, we used a dataset[1] that was generated using the SMIDgen protocol, described in [161]. On this data set, we evaluate the accuracy and resolution of the methods MRP, MRF, *PhySIC_IST*, BWD (using the SAC support function), ST, and FlipCut supertree. We omit the methods MC, MMC and SDM, as the results on the standard protocol data set show that they are clearly outperformed by all other methods.

Compared to the standard protocol, the SMIDgen protocol better reflects data collection processes used by systematists when gathering empirical data, including creation of densely-sampled clade-based trees as well as sparsely-sampled scaffold trees.

Model trees having either 100, 500 or 1000 taxa were generated with 30 replicates for the 100 and 500 taxon case, and ten replicates for the 1000 taxa case. We now explain the generation of the data set in detail, see also [161]. Basically, a set of source trees for the supertree methods consists of one scaffold tree and several clade-based trees. First, model trees are generated under a pure birth process. Branch lengths are perturbed from the ideal, ultrametric situation (no molecular clock). For each model tree, DNA sequence data sets are simulated under the GTR+Γ+I model, which differ in the taxa and genes used and whether they are scaffold or clade-based.

For the scaffold data sets, genes appearing at the root of a model tree are evolved along the tree without going extinct. Five of these so called universal genes are evolved for each model tree. A subset of taxa from the model tree is selected uniformly at random with a fixed probability p, called the "scaffold factor". This results in scaffold data sets having $p \times n$ taxa on average, where n is the number of taxa in the model tree. The scaffold data sets are generated using scaffold factors of either 0.2, 0.5, 0.75 or 1. For each of the resulting scaffold data set, maximum likelihood trees are inferred using RAxML [153] in its GTRMIX default setting.

[1] `http://www.cs.utexas.edu/~phylo/datasets/supertrees.html`

Genes for inferring clade-based source trees do not occupy the entire tree. For each of these 100 genes, called non-universal genes, a single birth node as well as lineages for which the given gene is lost, are determined using the process described in [161]. To choose clades for each clade-based data set, the same process is used, whereby the selection is restricted by setting bounds on the number of extant taxa in a clade to avoid selection of either very small or very large clades. For each 100-taxon model tree, five clades are selected with a clade size of at least 20. For each 500-taxon model tree, 15 clades with a clade size of at least 30, and for each 1000-taxon model tree 25 clades ranging in size between 30 and 500 are selected. For each of the clades chosen, the three non-universal genes are selected that cover the largest number of taxa in the clade. Afterwards, the taxa in the clade are restricted to those that have all three of the genes. As this process could produce data sets with small numbers of taxa, any clade-based data set with fewer than ten taxa is excluded.

Summarizing, a source tree set consists of one scaffold tree, that either covers 20%, 50%, 75% or 100% of the taxa from the corresponding model tree on average. Additionally, a source tree set contains five clade-based trees for a 100-taxon model tree, 15 clade-based trees for a 500-taxon model tree and 25 clade-based trees for a 1000-taxon model tree. We exemplary calculated the average number of taxa for three cases: the 30 source tree sets with 25% scaffold trees belonging to the 100-taxon model trees contain 41.17 taxa on average, the 30 source tree sets with 50% scaffold trees belonging to the 500 taxon model trees contain 86.17 taxa on average and the 10 source tree sets with 75% scaffold trees belonging to the 1000-taxon model trees contain 113.73 taxa on average.

4.1.3 Supertree Construction

We used slightly different settings for some supertree methods on the two data sets and outline the generation of supertrees separately in the following. The implementations used for the standard protocol data set were also used for the SMIDgen data set. The computations on the standard protocol data set were performed on a Sun Solaris cluster of AMD Opteron-275, 2.2 GHz CPUs, with 6 GB of memory. All computations on the SMIDgen Data Set were performed on a Linux cluster of AMD Opteron-2378, 2.4 GHz CPUs, with 16 GB of memory.

Standard Protocol Data Set

MRP supertrees were estimated using PAUP* 4.0b10 [163] with TBR branch swapping as heuristic search, random addition of sequences, and a maximum 10 000 trees in memory. In case of 48, 96, 144 taxon model trees, the search time for a single MRP supertree run was delimited to 5 minutes and for 524 taxon model trees to one hour. Since we explicitly did not delete the outgroup sequence from the alignment, the root of each in-

put tree is known. The strict consensus tree of all most-parsimonious trees was used as the final MRP tree. MRF supertrees were generated using the implementation provided by Duhong Chen[2], also with the TBR branch swapping strategy. For 48, 96, 144 taxon model trees, we used 30 replicates for the search. In case of 524 taxa model trees, on our cluster the MRF implementation failed independent of the number of replicates. We computed MC, BWD as well as FlipCut supertrees using our own implementations embedded in our software framework EPoS[3] [81]. For this data set we used the "Edge & Level" weighting scheme for the FlipCut algorithm, as explained in Chapter 3 on page 47. MMC trees were generated using Rod Page's implementation[4]. For the *PhySIC* and *PhySIC_IST* supertrees we used the implementations provided by the authors of the corresponding papers[5][6] [127, 146]. We did not collapse any branches from the input trees on the basis of a bootstrap threshold before applying the *PhySIC* and *PhySIC_IST* method (-b option). *PhySIC_IST* offers a parameter for the STC preprocessing (-c option), that allows to tune the method from "veto" to "voting-like". We tested the method with parameter 0, 0.5, 0.8, and 1. We found that results for parameters 0 and 0.5 are similar, and so are those for parameters 0.8 and 1. Therefore, we report only results for parameters 0 and 1 for this data set. In the following, we refer to these two parameter settings as *PhySIC_IST* 0, and *PhySIC_IST* 1. SDM+BioNJ* supertrees were computed using the implementation by Criscuolo *et al.* [47][7]. BioNJ* is part of the PhyD* package by Criscuolo and Gascuel [46][8]. We choose BioNJ* instead of FastME, because distance matrices in our simulation were incomplete. We generated SuperTriplets supertrees [128] using the implementation provided by the authors [9].

SMIDgen Data Set

On this data set, MRP supertrees were computed with TBR branch swapping strategy, random addition of sequences and no limit on the maximal number of trees in memory. MRF supertrees were generated also with the TBR branch swapping strategy. For 100 taxa model trees, we used 30 replicates for the search, and in case of 500 and 1000 taxa model trees only ten replicates, because on our cluster the implementation failed with more replicates. We also used the FlipCut algorithm as implemented in the EPoS framework [81]. In order to to illustrate the influence of branch-

[2] http://genome.cs.iastate.edu/CBL/download/
[3] http://bio.informatik.uni-jena.de/epos/
[4] http://darwin.zoology.gla.ac.uk/~rpage/supertree/
[5] http://www.atgc-montpellier.fr/physic/binaries.php
[6] http://www.atgc-montpellier.fr/physic_ist/
[7] http://www.atgc-montpellier.fr/sdm/binaries.php
[8] http://www.atgc-montpellier.fr/phyd/binaries.php
[9] http://www.supertriplets.univ-montp2.fr/download.php

length to our approach, we use two different weighting schemes for edges and character vertices in the graph model on this data set: First, unit costs, and second the "Edge & Level" weighting scheme as explained in Chapter 3 on page 47.

For *PhySIC_IST* supertrees, we used 0.5 and 1 as parameter for the STC preprocessing (-c option). On this data set, BWD supertrees were constructed using the implementation by Stephen J. Willson.[10]. We generated SuperTriplets as explained above.

4.1.4 Measuring accuracy and resolution

To evaluate accuracy of the supertrees build by the different methods, we compared the supertrees to the corresponding model trees using different distances and similarity scores. Recall that *PhySIC_IST* usually computes non-plenary supertrees: In this case, we first restrict the model tree to the taxon set of the supertree. Consider a rooted tree where all but two taxa have been removed: Obviously, this tree will always coincide with the correct topology. So, we can obtain better distance and similarity scores by removing taxa, in particular those that we consider "doubtful". In contrast, the MAST score (see below) will get smaller if we output a smaller tree. Hence, this approach favors *PhySIC_IST* for all distance measures except the MAST score, so *PhySIC_IST* results must be interpreted with some caution.

The Robinson-Foulds distance (*RF distance*) counts the number of clades that belong to only one of the two trees [129]. We normalize the RF distance by the number of internal nodes of both trees, yielding a value in $[0, 1]$. The Robinson-Foulds distance was measured using our own implementation embedded in our software framework EPoS [81]. Page [116] introduced the *triplet distance*, which is the rooted equivalent of the quartet metric [48].

The triplets of a model tree T and a supertree T' can be partitioned into five sets: $S(T, T')$ and $D(T, T')$, the triplets resolved in T and T' that have the same and different topologies, respectively; $R_1(T, T')$, the triplets resolved in T but not resolved in T'; $R_2(T, T')$, the triplets not resolved in T but resolved in T'; $X(T, T')$ the triplets unresolved in T and T'. Given these five triplet sets and the according triplet rates s, d, r_1, r_2, x, the type I error is defined as $et_I = (d + r_2)/(d + s + r_2)$. This corresponds to the proportion of triplets that are in T' but not in T and is sometimes called false positive. Accordingly, the type II error (sometimes called false negative) corresponds to the proportion of triplets that are not in T' but in T, and is defined as $et_{II} = (d + r_1)/(d + s + r_1)$. Note that in case of the standard protocol data set, all our model trees are fully resolved and, hence, $r_2 = x = 0$. Accordingly, we only report the triplet type II error for this data set.

For the standard protocol data set, we measured the maximum agree-

[10] `http://www.public.iastate.edu/~swillson/software.html`

ment subtree score, or *MAST score* for short [78], which counts the number of leaves in the maximum agreement subtree of the model tree and the supertree. The maximum agreement subtree was calculated using PAUP*. We normalize the MAST score using the number of leaves in the model tree. This indicates the fraction of the model tree that is recovered by the different methods. Note that in case of the SMIDgen Data Set, we measured the *MAST distance* instead of the MAST score.

Resolution was measured as the number of clades in the inferred supertree relative to the total number of clades on a fully binary tree of the same size ($n-2$ for an unrooted tree, where n is the number of taxa). Resolution varies between 0 and 1, where 0 indicates a unresolved bush and 1 indicates a binary supertree. The resolution as well as the triplet type I and type II error was measured using our own implementation embedded in our software framework EPoS [81]

We do not want to assess the pros and cons of the three distance methods but instead propose to use them as a *relative* measure to assess the quality of supertrees computed by the different supertree methods.

4.2 Simulation Results

We first present the results for the Standard Protocol Data Set, followed by the results obtained for the SMIDgen data set.

4.2.1 Results Standard Protocol Data Set

Results of our simulation with respect to supertree resolution, MAST score, RF distance and triplet type II error using the standard protocol data set and 96, 144 and 524 taxa model trees can be found in Figures 4.2 to 4.7 and for 48 taxa model trees in the appendix, Fig. A.1 and Fig. A.2. We omitted standard deviation plots in these figures for the sake of readability.

Concerning the accuracy of the supertrees from the different methods, generally one would expect that results improve if more input data becomes available, as this helps to identify bogus information. Hence, in general the triplet distance and RF distance should decrease, whereas the MAST score should increase when more input trees are available to the supertree method. The relative performance of the methods is in most cases similar for the different model tree sizes.

Note that the MRF implementation did not finish in case of 524 taxa modeltree instances. Unlike for all other model tree sizes, in case of 524 taxa modeltrees we investigated the FlipCut supertree method with and without undisputed sibling preprocessing, as described in Chapter 3, page 46. It turns out that the undisputed sibling preprocessing has no significant effect on this data set. This can be explained by a property of the standard protocol: As the deleted taxa are chosen per random, there are practically no undisputed siblings. For any empirical data set the number of undisputed siblings is certainly much higher.

In the following, we first discuss the observed patterns concerning accuracy and resolution for all methods under consideration in detail. Afterwards, we present results concerning *PhySIC_IST* in particular. *PhySIC_IST* is the only method under consideration able to produce non-plenary supertrees *i.e.*, supertrees that not necessarily contain all taxa from the input trees. To evaluate the extent of excluded taxa, we report the average number of taxa not included in the supertrees for each simulation result in Tables 4.1, 4.2, 4.3 and 4.4.

At the end of this section, we consider the running times of the methods under consideration, listed in Table 4.6.

Resolution

Results concerning the resolution of supertrees from all methods under consideration can be found in Figures, 4.2, 4.4, 4.6 and A.1. It must be understood that a highly resolved supertree does not imply that this supertree is of good quality. But on the other extreme, it is clear that a supertree method returning a highly unresolved tree is quite useless in application.

In our evaluation, *PhySIC* mostly returns star trees even for 25 % deletion ratio. For this reason, we decided to exclude the method from further investigation.

For 96 and 144 taxa model trees and 25 % and 50 % deletion ratio, most methods produce well resolved supertrees in general. SDM+BioNJ* builds the most resolved supertrees, followed by the two variations of BWD, FlipCut, MRF, ST, MMC, MRP and MC. The resolution of the supertrees build by those methods is independent from the number of input trees. In contrast, the resolution of MC supertrees increases with a higher number of input trees.

In case of 25 % deletion ratio and 96 and 144 taxa model trees, the resolution of *PhySIC_IST* 0 supertrees is comparable to MC supertrees, whereas in case of 50 % deletion ratio the supertrees are more unresolved. Except for 75 % deletion ratio, *PhySIC_IST* 1 constructs significantly more unresolved supertrees compared to all other methods. The same ordering of methods can be found in case of 524 taxa model trees and 50 % deletion ratio. The higher number of taxa has a significant positive effect on both *PhySIC_IST* variants.

For all model tree sizes and 75 % deletion ratio, we see that again SDM+BioNJ* builds the most resolved supertrees, followed by MRF, the BWD variants, FlipCut, MMC, MC, *PhySIC_IST* 0, ST and *PhySIC_IST* 1. The resolution of the supertrees is constant in the number of input trees, except for MC which produces more resolved supertrees with an increasing number of input trees. Furthermore, in case of 524 taxa model trees *PhySIC_IST* 1 shows heavy fluctuations. Compared to all other methods, MRP builds significantly less resolved supertrees for 75 % deletion ratio.

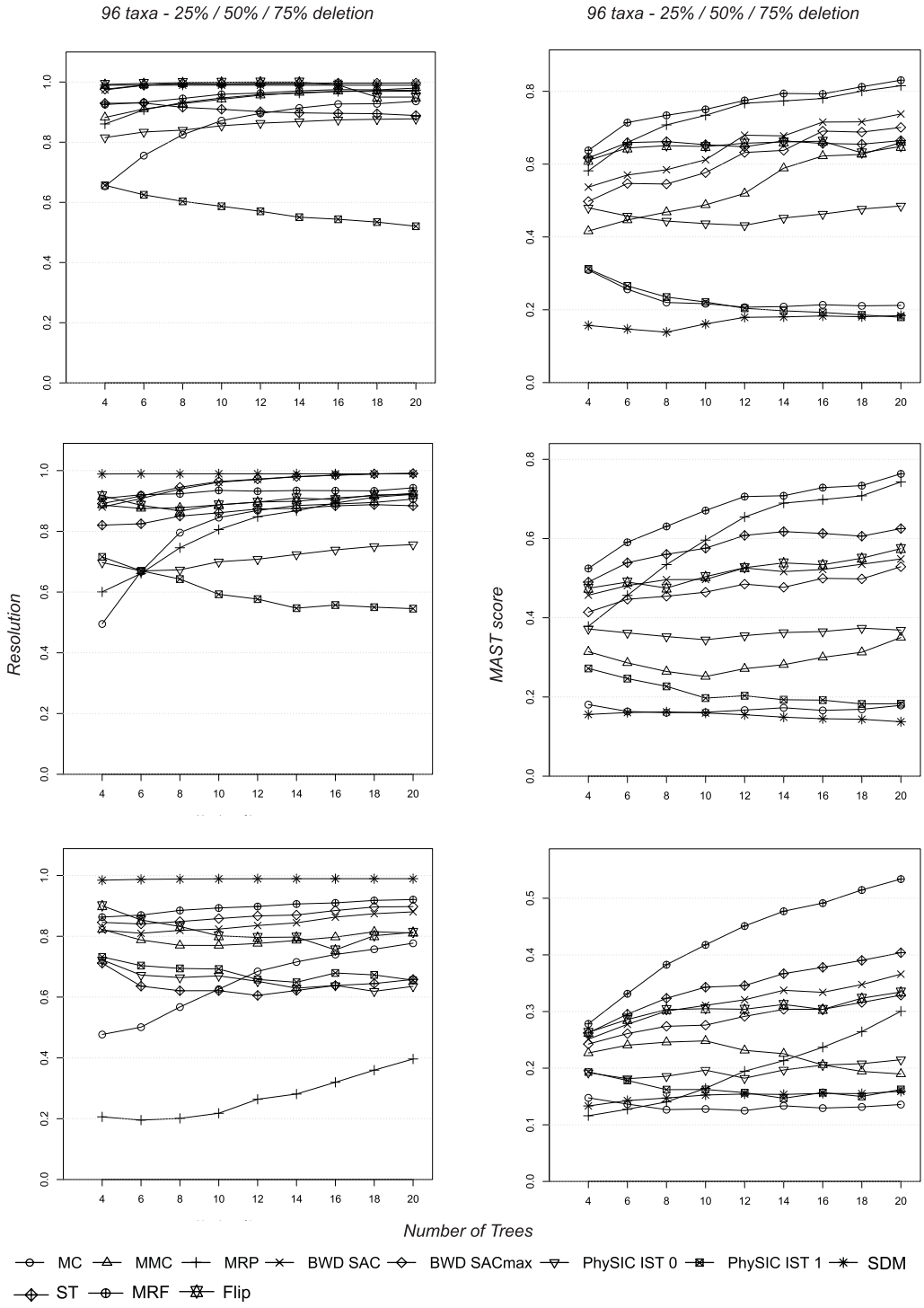


Figure 4.2: The left column of the figure shows the average resolution of the supertrees constructed from model trees with 96 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average MAST score of the supertrees constructed from model trees with 96 taxa.

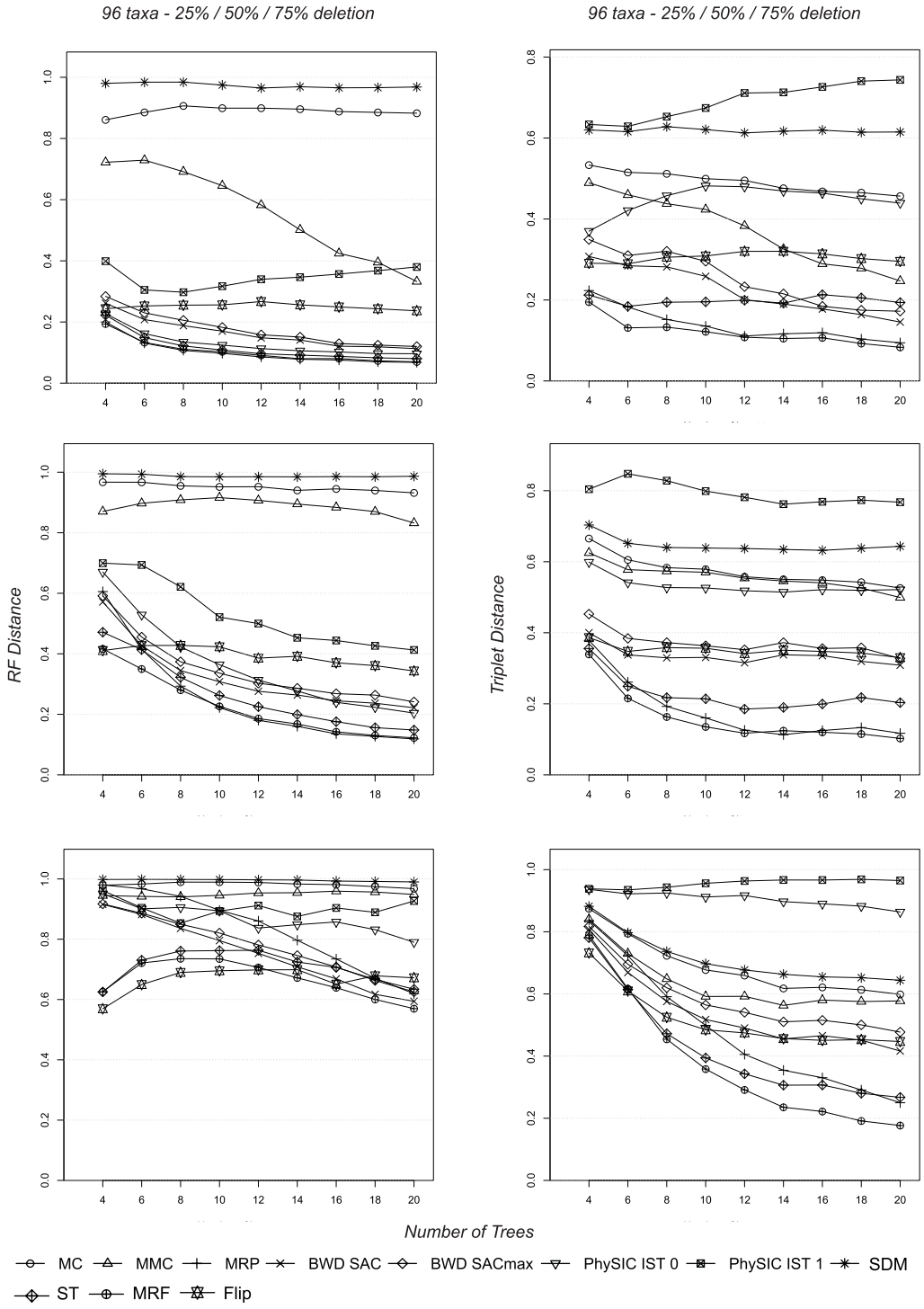


Figure 4.3: The left column of the figure shows the average RF distance of the supertrees constructed from model trees with 96 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average triplet distance of the supertrees constructed from model trees with 96 taxa.

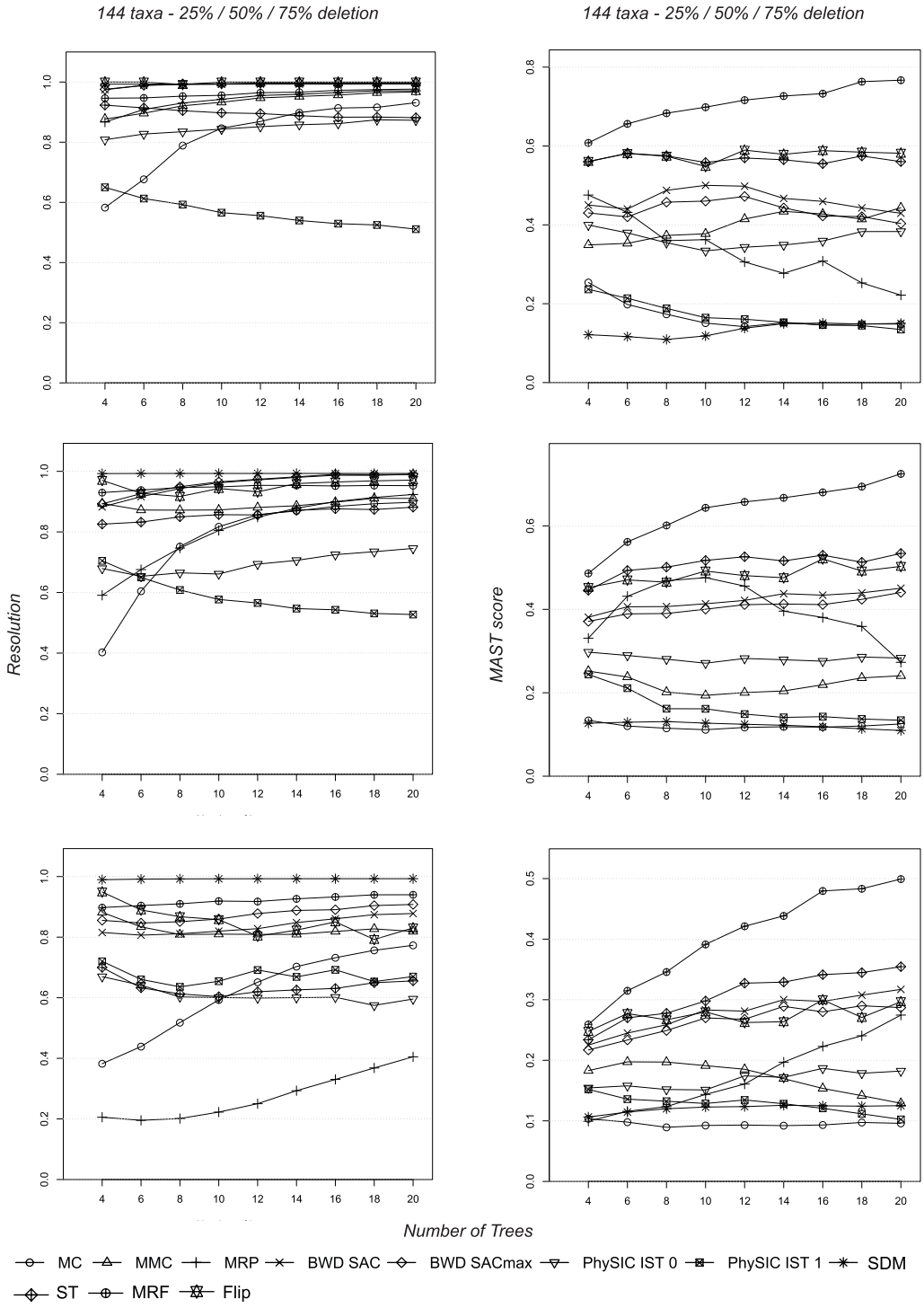


Figure 4.4: The left column of the figure shows the average resolution of the supertrees constructed from model trees with 144 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average MAST score of the supertrees constructed from model trees with 144 taxa.

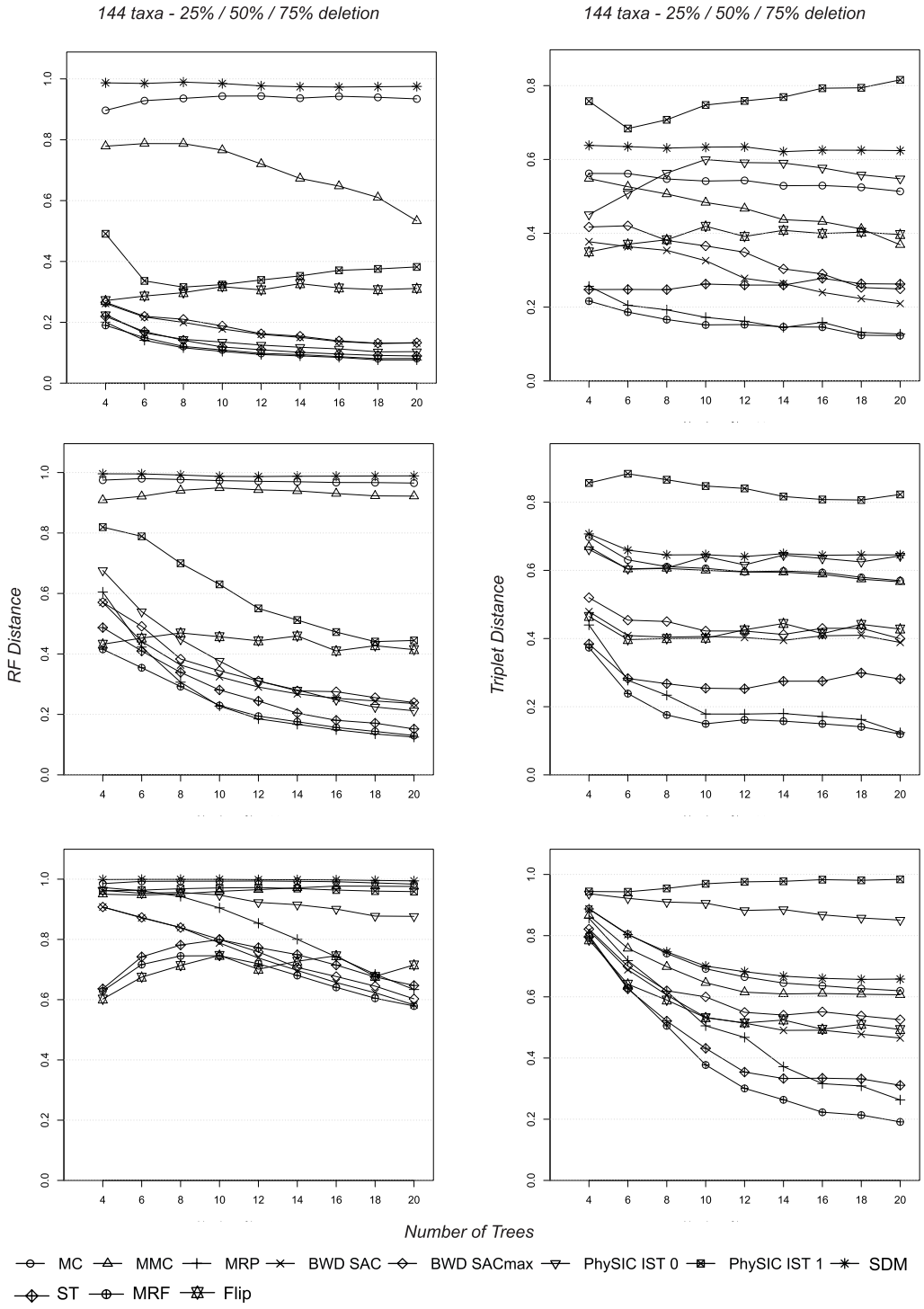


Figure 4.5: The left column of the figure shows the average RF distance of the supertrees constructed from model trees with 144 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average triplet distance of the supertrees constructed from model trees with 144 taxa.

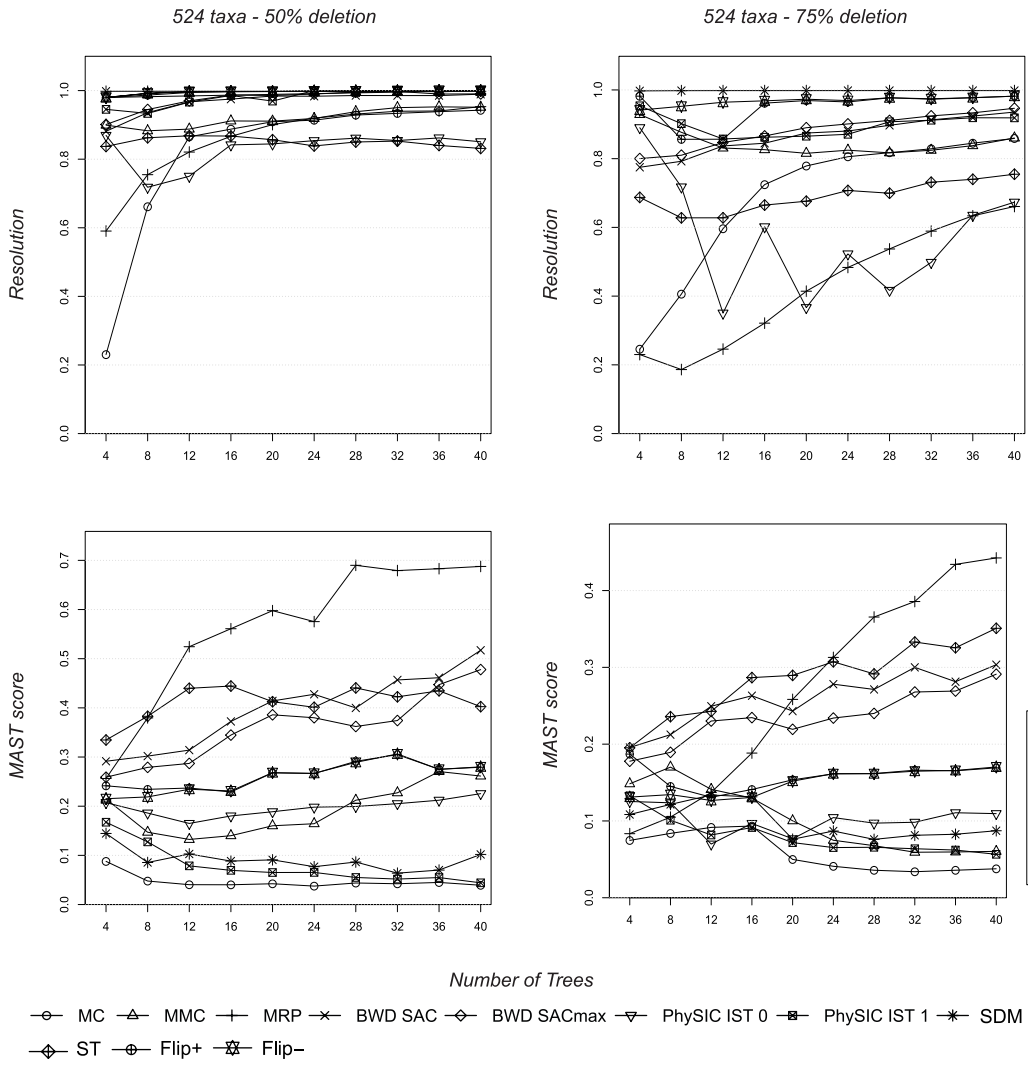


Figure 4.6: The upper row shows the average resolution of the supertrees constructed from model trees with 524 taxa and different taxon deletion ratios (left 50 %, right 75 %). The lower row shows the average MAST score of the supertrees constructed from model trees with 524 taxa and different taxon deletion ratios (left 50 %, right 75 %).

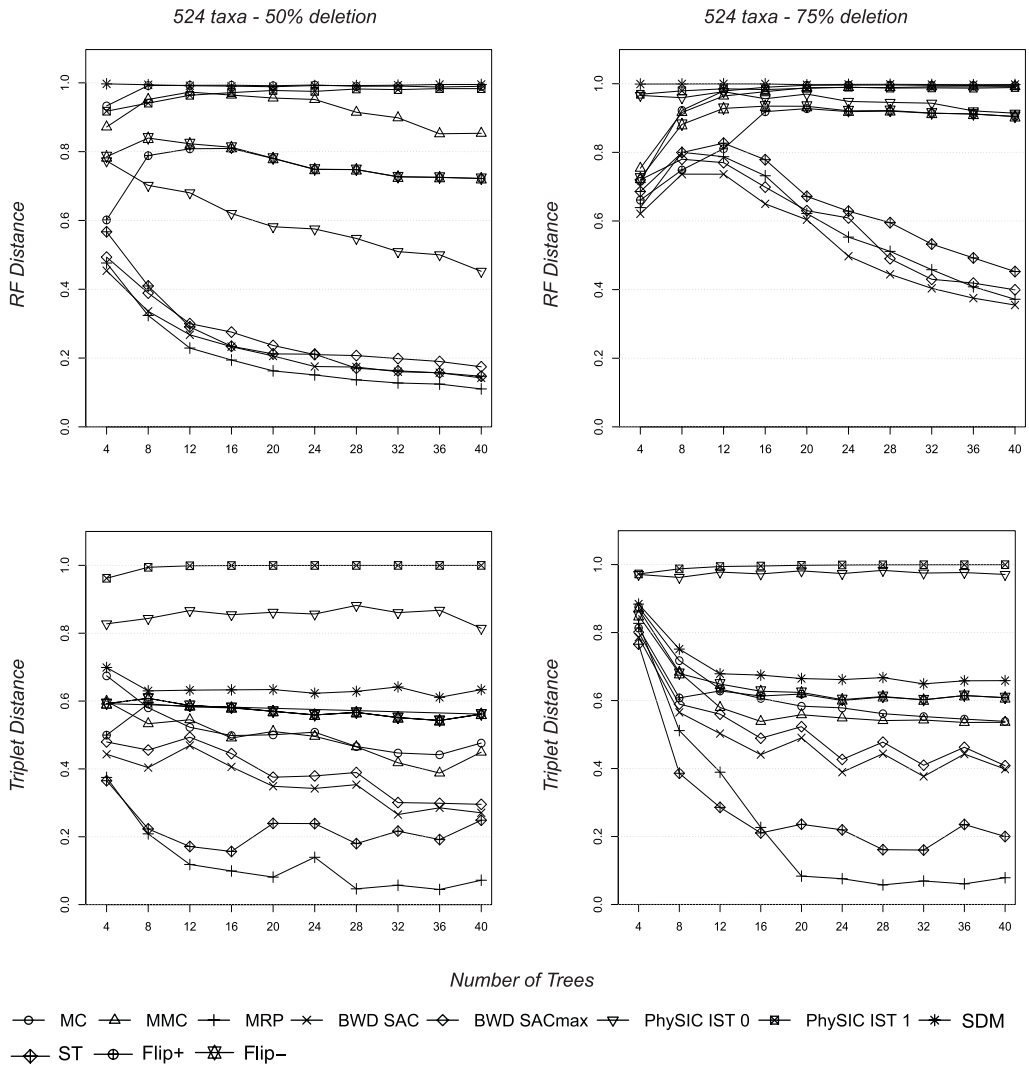


Figure 4.7: The upper row shows the average RF distance of the supertrees constructed from model trees with 524 taxa and different taxon deletion ratios (left 50 %, right 75 %). The lower row shows the average triplet distance of the supertrees constructed from model trees with 524 taxa and different taxon deletion ratios (left 50 %, right 75 %)

MAST score

With respect to the MAST score, the MRF method performs better than all other methods in all cases of deletion ratios, most significantly in case of 75 % deletion ratio. MRF generally benefits from more input trees. Note that MRF did not finish the 524 taxa model tree instances.

For 96 and 144 taxa model trees and all deletion ratios, the next best methods are ST, FlipCut, the BWD variants, *PhySIC_IST* 0 and MMC, which are all constant in the number of input trees. For the same parameter combination, we see that the group performing worst includes *PhySIC_IST* 1, MC and SDM+BioNJ*. Again, the methods behave constant in the number of input trees.

The MRP method outperforms all other methods except MRF in case of 25 % and 50 % deletion ratio and 48 or 96 taxa model trees. Here, MRP significantly benefits from a growing number of source trees. For 145 taxa model trees and 25 % and 50 % deletion ratios, the MRP curve shows a peculiar zig-zag pattern. We repeated this experiment twice, but obtained similar results. This behavior might be explained by local maxima in which the heuristic gets stuck for larger input trees and five minutes running time.

In case of 524 taxa model trees, the ranking of methods described above also applies, although all methods have a low MAST score in general. Here, MRP shows a significantly better behavior, which can be explained due to the longer running time for this size of model trees.

RF distance

Now we evaluate the different supertree methods with respect to the Robinson-Foulds distance. Recall that this distance is "conservative", as only perfectly reconstructed clades are counted as correct. The relative behavior of all methods is similar for all combinations of model tree sizes and deletion ratios. SDM+BioNJ*, MC and MMC group together and perform worst compared to all other methods. In this group, MMC is better than the others. Independent from the model tree size and 25 % and 50 % deletion ratio, the next best methods are both *PhySIC_IST* variants and the FlipCut method. For 96, 144 and 524 taxa model trees and 25 % and 50 % deletion frequency, performance from best to worst is roughly MRP, MRF, the BWD variants and ST. Note that MRF did not finish the 524 taxa model tree instances. In the extreme case of 75 % deletion, the RF distance of all methods is very high. Here, for 96 and 144 taxa model trees, FlipCut supertrees are best, followed by MRF, ST, BWD and MRP. The BWD variants and MRP benefit from larger model trees.

Triplet type II error

Next, we evaluate the different supertree methods with respect to the triplet distance. Note that for the standard protocol data set, the triplet

distance equals the triplet type II error, as the model tree is fully resolved. We also evaluated the triplet type I error but found that results do not differ significantly from those reported here; we omit the details. On page 68, we take a closer look on the values the triplet type II error is calculated from.

In case of 96 and 144 taxa model trees and 25 % and 50 % deletion ratio all methods behave relatively constant in the number of input trees. The group with the lowest triplet type II error consists of MRP, MRF and ST. In the middle, we see the methods FlipCut, both BWD variants, *PhySIC_IST* 0, MMC and MC. Compared to all other methods, *PhySIC_IST* 1 performs worst, followed by SDM.

For 96 and 144 taxa model trees and 75 % deletion ratio the worst methods are *PhySIC_IST* 1 and *PhySIC_IST* 0. The relative performance of all other methods is the same as for 25 % and 50 % deletion ratio, but here all methods benefit from a growing number of input trees.

For 524 taxa model trees again *PhySIC_IST* 1 and *PhySIC_IST* 0 build supertrees with the highest triplet type II error. The performance of all other methods in case of 50 % and 75 % deletion ratio is similar to that observed for 96 and 144 taxa model trees. Here, MRP significantly benefits from the size of the input trees and the longer running time.

PhySIC_IST: Number of excluded taxa

As mentioned in Chapter 2, page 31, *PhySIC_IST* can produce non-plenary supertrees, that is, supertrees that not necessarily contain all taxa from the input trees. For each simulation condition, Tables 4.1, 4.2, 4.3 and 4.4 show the average number of taxa excluded from supertrees built by *PhySIC_IST*, which depends on the number and size of the source trees. As one would expect, the more conservative *PhySIC_IST* 1 excludes more taxa than *PhySIC_IST* 0. For any number of input trees the amount of excluded taxa increases with a higher deletion ratio: For example, for the 96 taxon model trees and 10 input trees, *PhySIC_IST* 1 excludes 4.23, 22.80, and 49.17 taxa on average at deletion ratios 25 %, 50 %, and 75 %, respectively. This could be explained by the decreasing degree of overlap between the input trees, which impedes the insertion of taxa by the PI property. On the one hand, more input trees provide *PhySIC_IST* with more information on how to place taxa unambiguously in the supertree. On the other hand, more input trees also imply more conflicts between the input trees and, thus, the PC property will prevent the insertion of some taxa. For low deletion ratios, the amount of excluded taxa is also low. But for higher deletion ratios, the number of excluded taxa increases rapidly, and in many cases, more than half of the input taxa were excluded. For 145 taxa, 20 input trees, and 75 % deletion ratio, almost three-fourths of the taxa were excluded, a result which might be problematic in application.

No. input trees	25 % deletion		50 % deletion		75 % deletion	
	c = 0	c = 1	c = 0	c = 1	c = 0	c = 1
2	4.53	4.53	3.77	3.77	4.91	4.91
4	0.83	5.6	7.29	12.22	10.2	10.23
6	0.36	2.17	5.36	13.96	15.39	15.41
8	0.23	1.45	3.77	11.26	18.89	19.9
10	0.13	4.23	2.69	8.66	22.13	23.22
12	0.17	1.38	2	7.49	23.15	24.65
14	0.16	1.62	1.48	4.94	25.76	27.18
16	0.07	2.22	1.08	4.44	25.35	27.43
18	0.16	2.05	1.1	4.17	25.52	28.87
20	0.05	2.34	0.89	3.15	26.41	28.64

Table 4.1: The table shows average numbers of taxa excluded from supertrees build by *PhySIC_IST* from model trees of 48 taxa. Numbers are shown for different quantities of input trees, different deletion ratios and different parameters of the STC process (-c option).

No. input trees	25 % deletion		50 % deletion		75 % deletion	
	c = 0	c = 1	c = 0	c = 1	c = 0	c = 1
2	13.87	13.87	8.72	8.72	9.04	9.04
4	0.9	13.31	12.13	26.47	21.73	22.22
6	0.52	4.79	9.17	35.62	30.11	33.29
8	0.26	2.83	6.15	33.17	39.94	40.94
10	0.19	4.23	4.72	22.8	42.12	49.17
12	0.23	5.08	3.29	19.07	41.57	54.71
14	0.21	4.31	2.51	12.34	41.43	55.8
16	0.18	4.62	1.78	11.94	42.31	59.45
18	0.11	5.88	1.43	10.32	39.69	60.12
20	0.16	5.56	0.16	8.7	37.95	62.09

Table 4.2: The table shows average numbers of taxa excluded from supertrees build by *PhySIC_IST* from model trees of 96 taxa. Numbers are shown for different quantities of input trees, different deletion ratios and different parameters of the STC process (-c option).

Triplet Error II: Values of d, r_1 and s for 96 taxon model trees

In order to compare the veto and voting approach in more detail, it is interesting to compare the values the triplet error II is calculated from (see page 56). Table 4.5 shows the rounded average values of d, r_1 and s for *PhySIC_IST* 0, *PhySIC_IST* 1, BWD SAC and MRP in case of 96 taxon model trees and two, ten and twenty input trees. The described behavior of the methods is similar for 48 and 144 taxa model trees. Regarding *PhySIC_IST* 0, which mimics a voting approach, the number of non-identical triplets between model tree and supertree, d, generally decreases with more input trees, whereas the number of star triplets in the model tree, r_1, and the number of identical triplets, s, increase. In case of *PhySIC_IST*, d does not decrease as much as for *PhySIC_IST* 0, and can, as in case of 25 % and 75 %, also increase for more than ten input trees. Moreover, the number of star triplets increases and the number of identical triplets rather decreases in correspondence to the rather increasing value of d for the same number of input trees. For all deletion ratios, the voting based BWD method produces less non-identical triplets with a higher number of input trees. Star triplets are practically absent whereas the number of identical triplets clearly decreases for a higher number of input trees. The same behavior holds for the voting based MRP method, although it produces a significantly higher number of star triplets, which decreases in case of 25 % and 50 %. To summarize, the voting based methods BWD SAC and MRP produce more identical triplets on the one hand (whereby the former produces practically no star triplets), but also high numbers of non-identical triplets on the other hand when compared to *PhySIC_IST* 0. This behavior is generally counterbalanced by the veto based methods by producing more star triplets.

Running times for 525 taxa model trees

Running times of the polynomial supertree methods in case of 524 taxa model trees are listed in Table 4.6. Note that MRF did not finish these instances due to implementation issues. Unfortunately, for this data set the FlipCut as well as the *PhySIC_IST* implementation begin to use swap space due to implementation issues. This leads to very high running times, which exceeds several hours for both methods. Due to this, we do not show the running times for the *PhySIC_IST* and the FlipCut method. MC, MMC, BWD, SDM+BioNJ* and ST are relatively fast, the fastest being SDM+BioNJ*. MRP was constrained to a running time limit of one hour to reach a somewhat fair comparison.

No. input trees	25 % deletion		50 % deletion		75 % deletion	
	c = 0	c = 1	c = 0	c = 1	c = 0	c = 1
2	30.62	30.6	15.59	15.59	11.41	11.41
4	2.2	36.8	18.74	53.2	31.67	34.04
6	1.52	12.24	13.8	67.92	48.41	55.25
8	1.75	8.1	10.6	58.8	57.18	70.5
10	1.76	6.23	7.13	48.53	64.1	86.8
12	1.54	7.17	4.77	38.61	61.28	95.15
14	1.64	7.26	4.07	28.72	63.2	99.17
16	1.5	9.96	3.21	22.62	62.26	103.78
18	1.6	9.94	2.91	16.35	57.4	104.02
20	2.21	9.11	2.86	17.46	59.57	105.81

Table 4.3: The table shows average numbers of taxa excluded from supertrees build by *PhySIC_IST* from model trees of 145 taxa. Numbers are shown for different quantities of input trees, different deletion ratios and different parameters of the STC process (-c option).

No. input trees	50 % deletion		75 % deletion	
	c = 0	c = 1	c = 0	c = 1
4	150.1	300.5	180	179.2
8	145.2	429.8	269.3	338.1
12	124.8	468.7	299.9	410.9
16	130.3	483.8	314.6	432.6
20	118.4	490	307.6	455
24	118.7	492.6	307.3	470.4
28	105.7	498.7	313.8	478.1
32	96	499.1	303.9	483.9
36	96	498.7	305.6	485.4
40	81.5	502.8	304.6	488.6

Table 4.4: The table shows average numbers of taxa excluded from supertrees build by *PhySIC_IST* from model trees of 524 taxa. Numbers are shown for different quantities of input trees, different deletion ratios and different parameters of the STC process (-c option).

Method	No.	d			r_1			s		
		25 %	50 %	75 %	25 %	50 %	75 %	25 %	50 %	75 %
PhySIC	2	79560	113096	143171	12445	3512	249	55435	30832	4020
_IST 0	10	945	21989	132238	70095	55607	2226	76400	69843	12976
	20	769	5951	121243	63994	70944	6112	82676	70545	20085
PhySIC	2	79560	113096	143171	12445	3512	249	55435	30832	4020
_IST 1	10	17313	78432	138623	82097	39358	2321	48029	29650	6496
	20	22228	33872	139835	874160	79315	2426	37796	34253	5178
BWD SAC	2	53189	100519	13931	16	87	208	94246	46834	7841
	10	38105	48742	76237	1	7	92	109334	98691	71110
	20	21443	45524	61371	0	1	31	125997	101914	86037
MRP	2	45816	94097	137804	19002	11143	2776	82621	42201	6860
	10	12701	13033	32666	7273	10597	40878	127466	123810	73897
	20	10013	11712	15234	3882	5553	21545	133545	130174	110601

Table 4.5: The table shows the rounded average numbers of d, r_1 and s (triplet error II) for *PhySIC_IST* 0, *PhySIC_IST* 1, BWD SAC and MRP from model tree with 96 taxa. Numbers are shown for different quantities of input trees and different deletion ratios.

Nr. input trees	BWD		SDM		MC		MMC		ST	
	$d = 50\%$	75%	50%	75%	50%	75%	50%	75%	50%	75%
4	18:17	5:54	1:16	0:12	3:28	0:33	0:33	0:04	19:14	5:58
8	35:33	26:12	1:48	0:20	36:27	5:22	18:36	0:54	25:13	15:37
12	41:45	46:35	2:28	0:30	60:21	17:18	72:24	4:58	26:50	19:35
16	44:22	44:56	3:35	0:43	67:21	29:02	94:04	14:13	28:08	21:31
20	51:11	56:57	5:16	1:03	73:40	39:41	109:20	33:17	30:18	22:32
24	46:30	48:54	7:24	1:12	80:41	43:15	113:33	51:01	30:27	24:37
28	43:02	59:19	9:35	1:51	84:02	48:07	110:60	63:27	30:51	27:52
32	29:21	62:07	12:38	2:24	87:26	51:49	96:40	79:33	30:16	27:01
36	29:04	59:56	17:03	2:46	91:41	57:12	95:23	89:09	29:54	25:10
40	20:51	64:33	20:07	3:24	93:31	59:52	89:43	101:59	32:34	27:42

Table 4.6: Average running times (min:sec) of the polynomial supertree algorithms in case of 524 taxa model trees, different quantities of input trees and different deletion ratios. Note that for this data set the running time of MRP was limited to 60 minutes, and all instances reached this time limit.

4.2.2 Results SMIDgen data set

We now present the results obtained for the SMIDgen data set. We first consider the running times of the investigated methods, see Table 4.7), followed by results with respect to Robinson-Foulds (RF) distance, MAST distance, triplet type I and type II error and supertree resolution, shown in Figures 4.8 to 4.10. We omitted standard deviation plots in these figures for the sake of readability.

Running Times

The running times of MRP, MRF, BWD, *PhySIC_IST*, ST and the FlipCut algorithm ares shown in Table 4.7. Unlike for the 524 model tree instances from the standard protocol data set, in case of the SMIDgen data set the FlipCut implementation did not use swap space because of two reasons: First, the number of taxa is smaller compared to the standard protocol data set. Second, the cluster used in case of the SMIDgen data set provides a significantly larger amount of memory than the cluster used for the standard protocol data set.

For each instance, we use a running time limit of one hour in case of 100 and 500 taxon model trees, and two hours in case of 1000 taxon model trees. Entries '–' indicate that *no instance* was finished within the time limit: Regarding MRP, PAUP* returns a consensus of the current trees in memory if the time limit is exceeded. In contrast, the MRF implementation returns no tree. *PhySIC_IST* crashes for model tree sizes of 500 and 1000 taxa, and the ST implementation crashes for model tree sizes of 1000 taxa.

PAUP* often runs into timeouts even for the smallest instances containing only 100 taxa; similarly, *PhySIC_IST* can process only instances of this size. MRF, BWD, and SuperTriplets can process instances with up to 500 taxa in less than one hour; for MRF, this implies that the heuristic used to solve the underlying hard problem, considers only a smaller part of the search space. In contrast, our FlipCut method is several orders of magnitude faster than any other method; even large instances with 1000 taxa can be processed in a matter of minutes. The "Edge & Level" version requires less than seven minutes on average, for any of the parameter settings.

Robinson-Fould distance

For all model tree sizes and scaffold factors, MRP supertrees are of the best quality. For 100 taxa model trees, we see four different groups of methods: The best group consists of MRP, the *PhySIC_IST* variants, and MRF. The second-best group is FlipCut "Edge & Level"; BWD and FlipCut with unit cost show the worst performance. ST performs good for the small and large scaffold density. For 500 taxa model trees, performance from best to worst is: MRP, MRF, FlipCut "Edge & Level", BWD, and FlipCut unit costs. Again, ST performs good for small and large scaffold density. For

1000 taxa model trees, performance from best to worst is: MRP, FlipCut "Edge & Level" and FlipCut unit costs.

We also investigated some FlipCut supertrees in detail, in particular those that show a comparatively high RF distance to the model tree: We often find that a single taxon is wrongly separated from a larger clade at an early stage of the algorithm. This has strong impact on the RF distance which counts common clades, as it usually affects a large number of clades in the supertree, and all of these contribute to the RF distance.

MAST distance

The MAST distance represents the size of the largest subtree that is common to both the model tree and the supertree. For 100 taxa model trees, *PhySIC_IST* 1 performs significantly worse than all other methods, whereas MRP outperforms the other methods only with input tree sets with a scaffold density of 75% and 100%. Both FlipCut "Edge & Level" and MRP compute supertrees such that the MAST between supertree and model tree consistently contains more than half of the taxa. MRP, MRF, and FlipCut "Edge & Level" perform almost on par. For 500 taxa model trees, we see three groups: MRP and MRF perform best, closely followed by FlipCut "Edge & Level" and ST. Performance of FlipCut unit cost and BWD is much worse. Finally, for 1000 taxa model trees, MRP performs best; FlipCut "Edge & Level" is on second place with similar performance except for scaffold density 100%; and FlipCut unit cost is much worse. For the MAST distance, we can see that the early separation of single taxa, as discussed above, does not have a big impact: Cutting away single taxa early, removes only one taxon from the MAST.

Triplet type I and type II error

We find that the type I and type II error (shown in Figures 4.11 and 4.12) are very similar for the SMIDgen data set, thus we will discuss them together in the following. For 100 taxa model trees, *PhySIC_IST* 1 performs worse than all other methods, whereas *PhySIC_IST* 0.5 is significantly better, but still outperformed by all other methods. All other methods compute relatively similar for this model tree size. For 500 taxa model trees, performance from best to worst is: BWD, MRF, ST, MRP, FlipCut "Edge & Level", and FlipCut unit costs. Finally, for 1000 taxa model trees, MRP performs best in case of the triplet type I error; for the triplet type II error, FlipCut "Edge & Level" is on par with MRP. FlipCut unit costs performs significantly worse in both cases.

Resolution

Supertree resolution is shown in Fig. 4.10. For 100 taxa model trees, *PhySIC_IST* 1 produces supertrees with a significant lower resolution than all methods. *PhySIC_IST* 0.5 supertrees are more resolved, reflecting the

influence of the STC preprocessing. The resolution of all other methods is high (mostly over 0.95). Both FlipCut variants produce nearly fully resolved supertrees for all scaffold factors and model tree sizes. MRP builds more resolved supertrees for larger taxa model trees, whereas ST returns slightly less resolved supertrees in case of 500 taxa model trees.

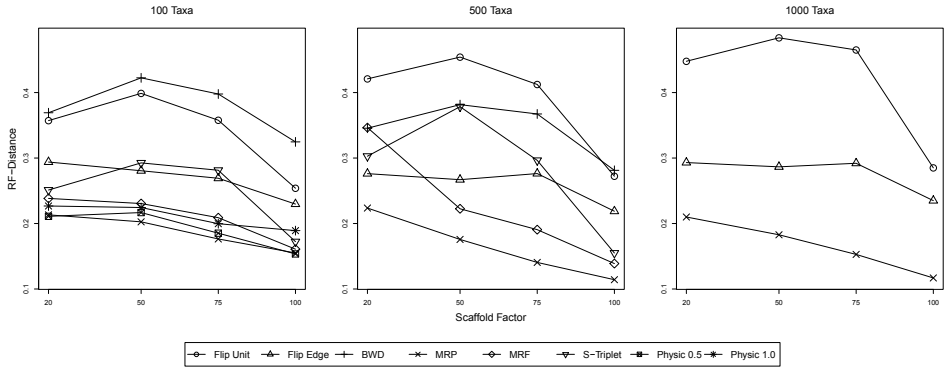


Figure 4.8: Simulation results, quality of reconstructed supertrees. We plot the Robinson-Foulds distance between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, model trees with 100, 500, and 1000 taxa.

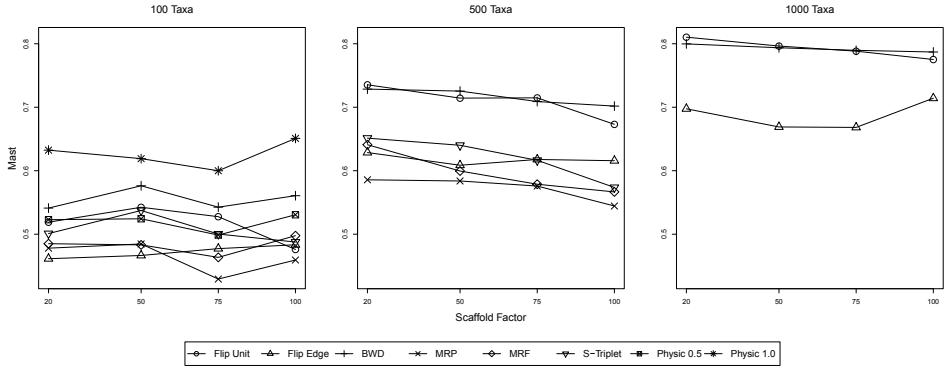


Figure 4.9: Simulation results, quality of reconstructed supertrees. We plot the MAST distance between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, model trees with 100, 500, and 1000 taxa.

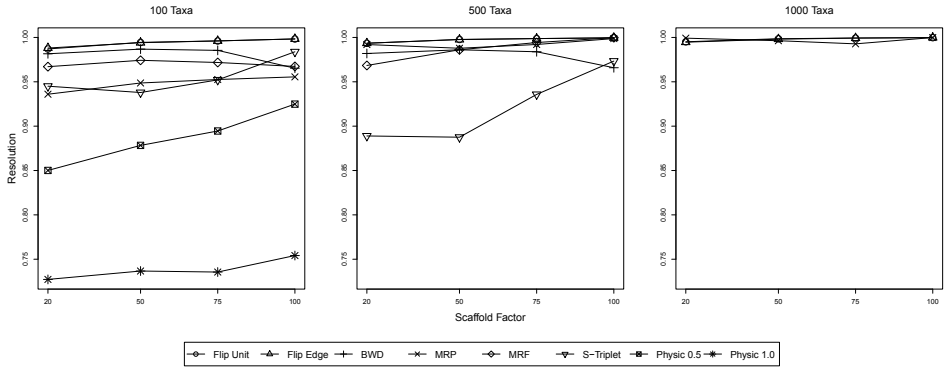


Figure 4.10: Simulation results, resolution of reconstructed supertrees. We plot the resolution of supertrees averaged over all simulation replicates. From left to right, the figure shows resolution of supertrees belonging to 100, 500 and 1000 taxon model trees.

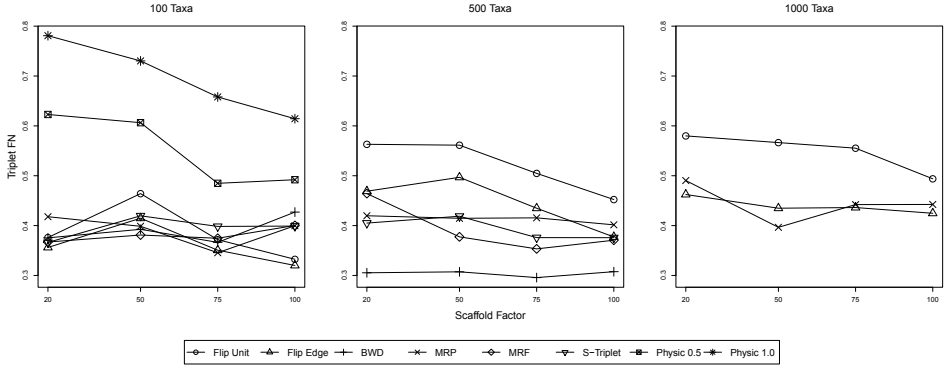


Figure 4.11: Simulation results, quality of reconstructed supertrees. We plot the triplet-based type II error between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, the figure shows type II errors between the supertrees and 100, 500 and 1000 taxon model trees.

Model tree	Scaff. factor	MRP #TO	MRP avg*	MRF	BWD	PhySIC IST $c = 0.5$	PhySIC IST $c = 1$	ST	FlipCut unit	FlipCut E&L
100	20%	19/30	18:47	3:01	≈ 1 s	16:16	28:02	0:05	< 1 s	< 1 s
taxa	50%	20/30	4:36	5:15	≈ 1 s	17:04	10:57	0:05	< 1 s	< 1 s
	75%	14/30	16:47	5:40	≈ 1 s	17:02	12:52	0:06	< 1 s	< 1 s
	100%	0/30	0:36	4:51	≈ 1 s	5:54	08:33	0:06	< 1 s	< 1 s
500	20%	30/30	–	45:37	20:41	–	–	8:56	0:30	0:22
taxa	50%	30/30	–	18:36	29:49	–	–	11:59	0:55	0:42
	75%	30/30	–	16:36	33:30	–	–	15:46	0:40	0:38
	100%	30/30	–	34:14	31:54	–	–	18:55	0:12	0:15
1000	20%	10/10	–	–	–	–	–	–	8:21	2:28
taxa	50%	10/10	–	–	–	–	–	–	15:42	4:41
	75%	10/10	–	–	–	–	–	–	13:23	6:59
	100%	10/10	–	–	–	–	–	–	1:51	1:50

Table 4.7: Running times (min:sec) of the different algorithms. *MRP #TO* is the number of timeouts of MRP where computation was stopped after one hour, and *MRP avg** is the average running time of those runs that stopped before the time limit. *ST* is the SuperTriplets method. BWD computed four supertrees for different distance models within the measured time.

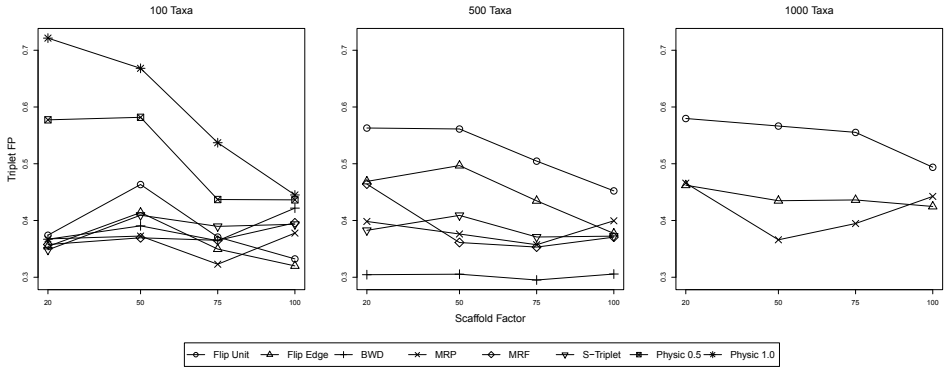


Figure 4.12: Simulation results, quality of reconstructed supertrees. We plot the triplet-based type I error between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, the figure shows type I errors between the supertrees and 100, 500 and 1000 taxon model trees.

CHAPTER 5

EPoS

In this chapter we present EPoS (Estimating Phylogenies of Species), a modular and extendable Java software framework for phylogenetic analysis, visualization and data management.

Since EPoS started as a student research project in 2003 it has developed into a comprehensive software framework, consisting of several hundred thousand lines of code in about 740 different Java classes.

Chief developer and driving force behind EPoS is Thasso Griebel. His PHD thesis [80] provides an in-depth presentation of concepts and implementation details. As part of the developers team I was involved in the implementation of some core modules, algorithms and other functions as well as project planning. In the following, we outline the motivation for a modular software approach in phylogenetics, provide a general overview of the EPoS platform and introduce some important underlying concepts. EPoS is freely[1] available at `http://bio.informatik.uni-jena.de/epos/`.

5.1 Motivation

A phylogenetic analysis, from planning a study to the production of a publishable phylogenetic tree, is a multi-step procedure. The complete process of inferring a phylogeny, from its conception to the assessment of support for the final phylogeny, is beyond the scope of this thesis. Here, we consider only the most basic steps from a software user perspective in order to give the reader an impression of the various requirements a phylogenetic analysis comprises. Afterwards, we show the usefulness of our modular and extendable framework EPoS.

One of the major general problems in practical bioinformatics and phylogenetics nowadays is the abundance of formats biological data can be stored in. Due to the lack of standardized data formats, bioinformatic tools and web-services often implement their own formats for data in- and output.

The initial step in a phylogenetic project is the construction of an appropriate dataset. Sequences have to be imported from a chosen database or loaded from an existing project. Homologous sequences have to be identified via Blast searches against public databases and incorporated into the dataset. In a next step, a model of sequence evolution has to be chosen and a multiple sequence alignment has to be constructed. Several stand-alone programs and web-services exist for these tasks, again often implementing their own data formats. Once an alignment is constructed, subsequent editing, visualization and management of the alignment is needed.

After a suitable multiple sequence alignment is constructed the actual

[1] EPoS is a free software: it can be redistributed and/or modified under the terms of the GNU Lesser General Public License as published by the Free Software Foundation.

phylogenetic analysis begins. Different tools implementing the various approaches to infer phylogenetic relationships can be used, each with it's own advantages and disadvantages and scope of application. Ideally, a user should be able to apply preferably any kind of method or tool on the dataset that is appropriate for the demands of the project in question. Finally, inferred phylogenies have to be subsequently manipulated, visualized and exported into several formats by the user. Furthermore, given a set of phylogenetic trees, a range of applications come to mind that are useful for a phylogenetic project, such as consensus and supertree calculations or general comparison of different trees.

Existing software in phylogenetics can roughly be divided into two groups: Algorithmic packages that provide computational methods for a specific problem and visualization tools to analyze results. Usually algorithms are implemented by members from the algorithmic community who are often not experts in user interface design. Thus, algorithm implementations are often command line based and therefore may lack in usability. Visualization tools often suffer from insufficient support for computational methods. To summarize, from a computational point of view a phylogenetic analysis includes various types of data and formats, consists of several algorithmic steps and demands for a consistent data handling and management strategy as well as data visualization.

Our modular and extendable software framework EPoS tries to improve the complex and multifaceted process of phylogenetic analysis by integrating required data types (sequences, distance matrices, alignments and trees), software and web services for a phylogenetic analysis into one framework. It combines a powerful graphical user interface with a plugin system that allows simple integration of new algorithms, existing implementations, visualizations and data structures. A consistent user interface is used to collect, manage and link data and to employ available computational methods which are integrated in a pipeline system. This allows combinations of methods to be executed sequentially, while the data flow is handled automatically by the system. Furthermore, EPoS provides a scripting functionality to automate common tasks as well as the ability to execute processes either locally or on a remote compute cluster.

5.2 Overview of the EPoS framework

We now present an overview of the EPoS framework. To give the reader a first impression, a screenshot of EPoS can be found in Fig. 5.1.

EPoS main user interface and the workspace

The main user interface is the central point for an user to interact with the EPoS framework. It provides ways to start computational methods, to switch between different views, to visualize data etc. It also allows the user to import and export various types and formats of phylogenetic data.

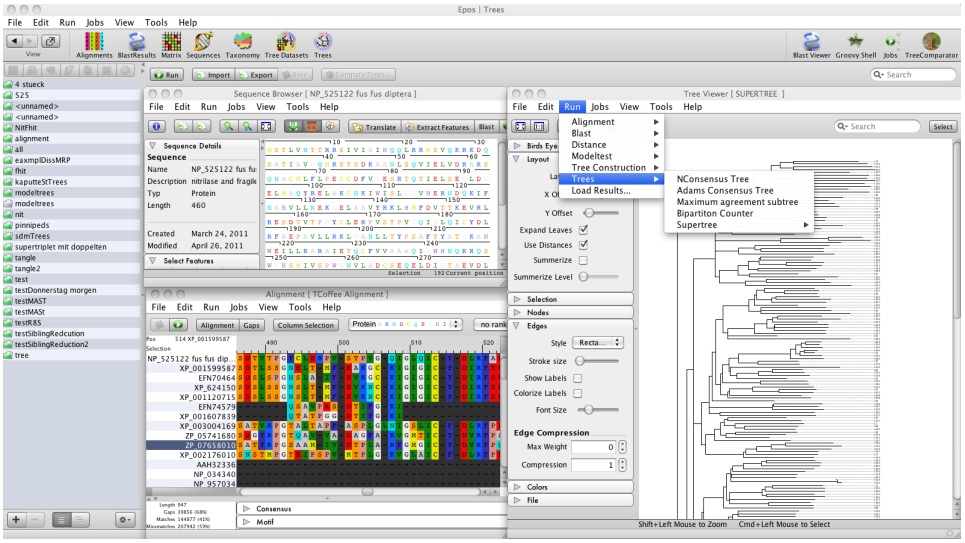


Figure 5.1: A screenshot of the EPoS platform and some of the main components. The menu bar on top of the application allows to access all functions of EPoS, to switch between different views, start computational methods etc. The folder structure on the left represents the virtual file system provided by the EPoS workspace. It allows the user to organize different kinds of data from the workspace into hierarchical structures corresponding to specific projects. In the main application area, three different *Views* are opened: the *Sequence View*, the *Alignment View* and the *Tree view*.

Sequences can be imported and exported from and to raw sequence files in all common formats. Furthermore, sequence data can also be fetched directly from the NCBI using GeneID's or accession numbers. Alignments and trees can be imported and exported in all common formats. Both types of data can also be directly reconstructed in the EPoS framework. The same holds for distance matrices. Phylogenetic data is stored and managed by the persistent *Workspace* using a transparent and extendable back-end module. The Workspace abstracts the major background database and interactions, enables searching and provides a virtual file system which allows the user to organize data in a folder structure. Furthermore, the Workspace allows linkage between stored objects, for example between trees and sequence alignments. Besides comfortable management of data in hierarchical folder structures, EPoS provides data depended overviews of the different data types in the Workspace: The *Sequence Overview* shows all sequences in the current Workspace. Accordingly, the *Alignment Overview* and the *Tree Overview* (see Fig. 5.2) visualize all alignments and trees. Furthermore, EPoS provides a special *Blast Viewer* (see Fig. 5.3) that allows to import NCBI Blast results in XML format to view the hits. Currently the Blast viewer is able to create the standard NCBI Blast alignment output and export the results as CSV file.

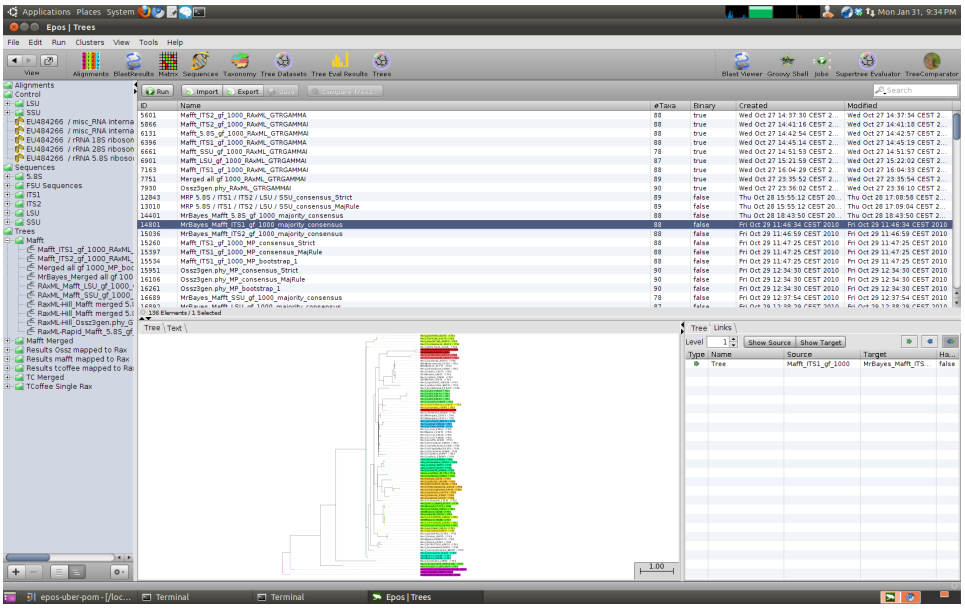


Figure 5.2: In contrast to the virtual file system provided by the Workspace (left), which allows to organize phylogenetic data into a folder structure, the *Tree Overview* provides an overview of all trees in the Workspace. Similar overviews are also available for all other types of data.

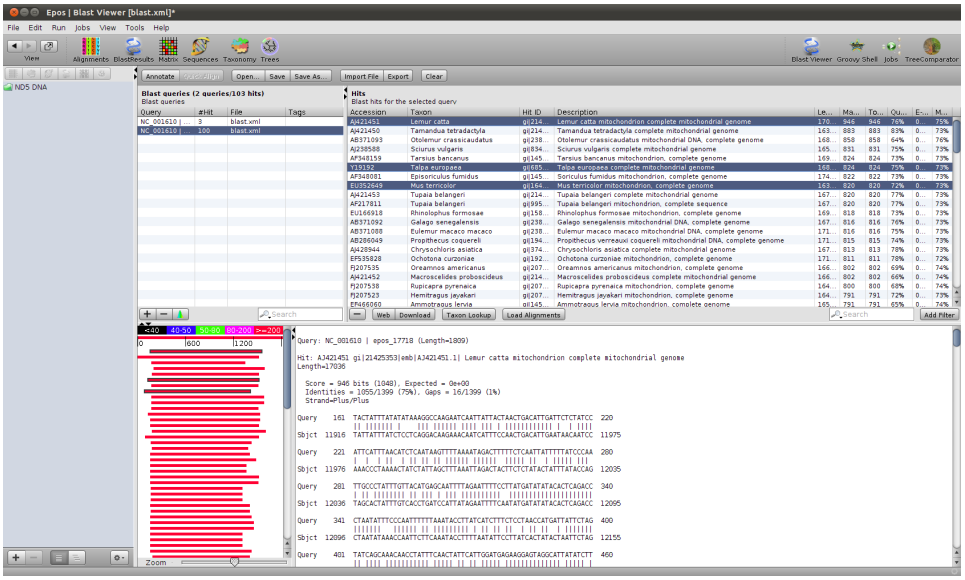


Figure 5.3: The Blast Viewer allows to view Blast results and to import interesting hits into the workspace.

Visualizations for different types of phylogenetic data

EPoS contains several views for sequences, alignments, trees and distance matrices. The *Sequence Viewer*, shown in Fig. 5.4, enables the user to view and translate sequences. It also allows to extract feature regions and to Blast against the NCBI database. The *Alignment Viewer*, shown in Fig. 5.5, allows common manual manipulations such as modifying gaps and provides various additional information about the alignment. Trees can be visualized and manipulated with the comprehensive *Tree Viewer* (see Fig. 5.6 and 5.7). It offers different layouts, colorizations, annotations, search and export functions. The *Tree Viewer* focuses on interactive tree analysis and provides functionality to display large trees with several thousand leaves, without loosing the ability to smoothly interact with the view. Furthermore, EPoS provides a *Tree Comparator* (see Fig. 5.8), that allows to visually compare trees by building tanglegrams. Furthermore, trees can be compared with several distance measures.

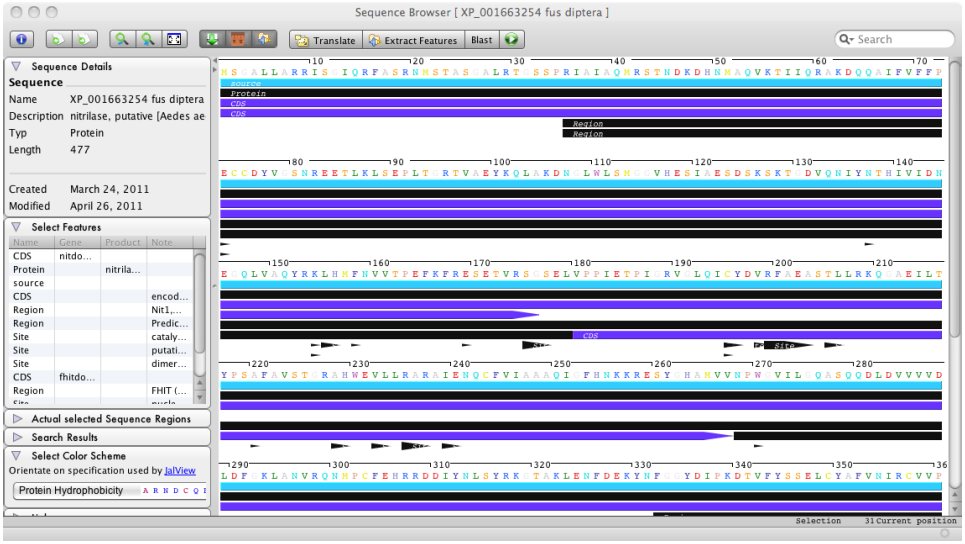


Figure 5.4: The Sequence Viewer showing an annotated sequence. It allows to translate sequences, to extract feature regions and to Blast against the NCBI database.

Figure 5.5: The Alignment Viewer allows common manual manipulations such as modifying gaps and provides various additional information about the alignment.

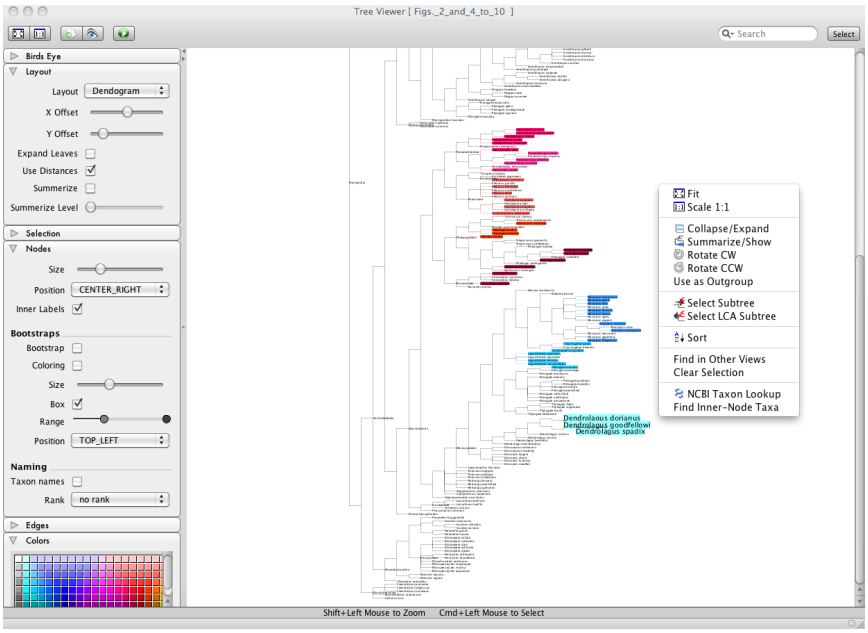


Figure 5.6: The Tree Viewer provides different layouts and various ways to interact with trees.

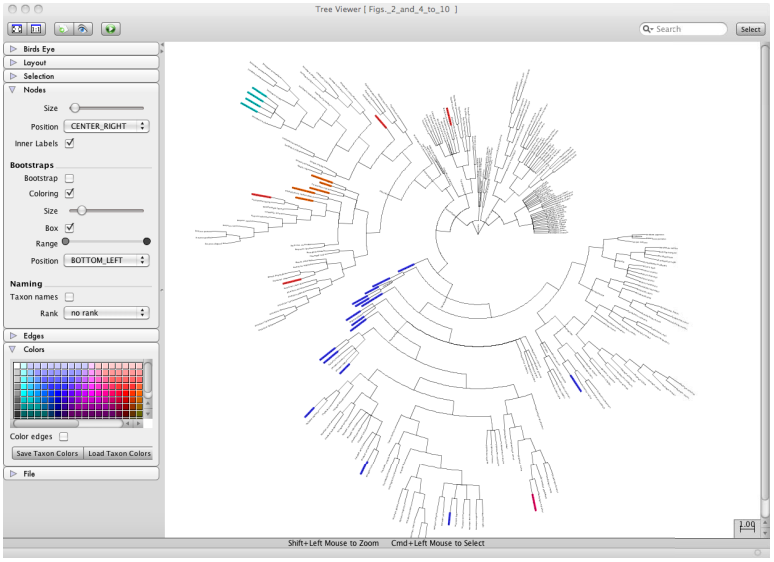


Figure 5.7: EPoS Tree Viewer showing a tree with circular layout. The tree viewer is able to display large trees with several thousand leaves, without loosing the ability to smoothly interact with the view.

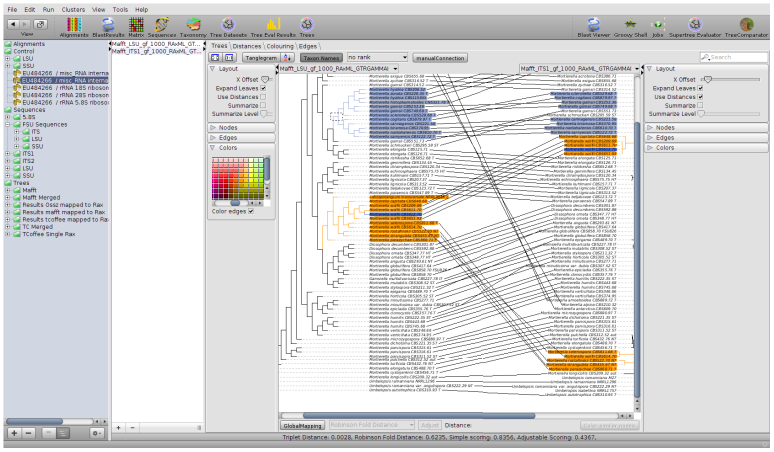


Figure 5.8: EPoS tree comparator showing a tanglegram. The tree comparator also allows to compare trees using various distance measures.

Integrated Methods And Programs

In the following, we list the different phylogenetic methods that are currently integrated in the EPoS system, either as supported external programs or direct implementations. The integration of methods into the system will be explained below. If a supported external tool is not yet installed, the EPoS framework provides comfortable ways to install and setup the program.

At the time of writing this thesis, EPoS supports the following programs and methods:

- Multiple Sequence Alignments
  - ClustalW (Thompson *et al.* [165])
  - Mafft (Katoh *et al.* [96])
  - Muscle (Edgar [55])
  - Dialign-TX (Subramanian *et al.* [160])
  - TCoffee and MCoffee (Notredame *et al.* [115], Wallace *et al.* [169])
  - DCA (Stoye [158])
- Model Tests
  - ModelGenerator (Keane *et al.* [97])
  - ModelTest (Posada and Crandall [121])
- Homologue Search and Sequence Annotation
  - Blast (Altschul *et al.* [4])
- Tree Reconstruction and Bootstrapping
  - RaxML (Stamatakis [153])
  - MrBayes (Ronquist and Huelsenbeck [133])
  - Paup, *Maximum Parsimony* (Swofford [163])
  - QuickTree, *Neighbour Joining* and *Agglomerative Clustering* (Howe *et al.* [90])
- Tree Comparison
  - NConsensus and Adams Consensus (Amenta *et al.* [5],Adams III [1])
  - Paup, *Maximum Agreement Subtree* (Swofford [163])
  - Tanglegrams (Böcker *et al.* [28])
  - Robinson-Fould Distance (Robinson and Foulds [129])

  - Triplet Distance (e.g. Page [116], Ranwez *et al.* [128])

- Supertree Methods
  - Build and OneTree (Page [116], Semple and Steel [147])
  - Ancestral Build (Berry and Semple [13])
  - Ranked Tree (Bryant *et al.* [33])
  - Paup, *Matrix Representation with Parsimony* (Swofford [163])
  - HeuristicMRF2, *Matrix Representation with Flipping* (e.g. Chen *et al.* [39])
  - FlipCut (Brinkmeyer *et al.* [31])
  - MinCut and Modified MinCut supertree (Page [116], Semple and Steel [147])
  - BuildWithDistances (Willson [172])
  - PhySIC and PhySIC_IST (Ranwez *et al.* [127], Scornavacca *et al.* [146])

Scripting support

EPoS provides a build-in Groovy Editor, which allows to automate common tasks via the Groovy scripting language. The scripting tool has some comfortable feature. For example, data can be simply dragged and dropped into a new variable. Note that the scripting functionality is not limited to the build-in editor. All features of EPoS can also be used via Groovy scripts in a non-graphical environment.

Cluster support

The EPoS framework provides a comfortable way to directly integrate compute clusters based on the Sun Grid Engine and thus to outsource computations. Once a remote cluster is integrated, jobs can be directly submitted from within the framework to the cluster and the results are fetched when available. From a user perspective there is no difference where the job is actually computed.

5.3 Software architecture and fundamental concepts

We now describe some basic concepts and important parts of the software architecture underlying the EPoS framework. EPoS is implemented in the object-oriented programming (OOP) language Java. The OOP paradigm is based on concepts like data abstraction, encapsulation, messaging, modularity, polymorphism and inheritance. A detailed description and definition of these concepts and OOP in general can be found in various books on the topic, for example in Gamma *et al.* [69]. In the following, we presume

that the reader is familiar with object-oriented principles and concentrate on the concepts of modularity and extendability and provide an overview of their implementation in EPoS. Instead focusing on technical details, our main objective here is to give the reader an impression of EPoS's advantages for users as well as developers.

The EPoS framework is based on a three layer architecture [54] and consists of various *components* or *modules*. We will use these terms interchangeably in the rest of this chapter. All components can be assigned to one of the following layers:

- *Data layer.* In general, the data layer is responsible for storing, accessing and retrieving data from a persistent data storage. Usually, this layer connects to a database and transparently provides access to the data sources. Instead of using a direct database layer, we split up the classical tasks of the data layer by creating an abstract persistence layer, the EPoS Workspace. Basically, a workspace contains all data for user defined projects, manages storing and retrieving domain objects, provides ways to extend domain objects and allows to add new data types.

- *Logic layer.* The logic layer is responsible for the applications logical behavior. In EPoS, several components belong either completely or in parts to this layer. For example, this includes all algorithms and methods provided by the EPoS framework and the integrated connections to web-services (e.g. the NCBI web-service).

- *Presentation layer.* The presentation layer deals with user interaction and the graphical user interface. EPoS's user interface and its conceptual coherence is one of the most important parts of the system. In EPoS, all user interface related components are called *applications*. The EPoS application framework (mainly consisting of the *Plugin System* and the *Application Core*, see below) allows to define applications independent from their representation in terms of look and feel. Moreover, the application framework in EPoS auto-generates the visual representation for program logic and application features and creates user interfaces for new implementations as well as integrated external programs.

  Automatically generated visualizations ensure that new interfaces are consistent with the user's previous experience and thus allow a homogeneous handling experience throughout the framework. From a developer perspective, the application framework allows to concentrate on concrete implementations without bothering about user interface design.

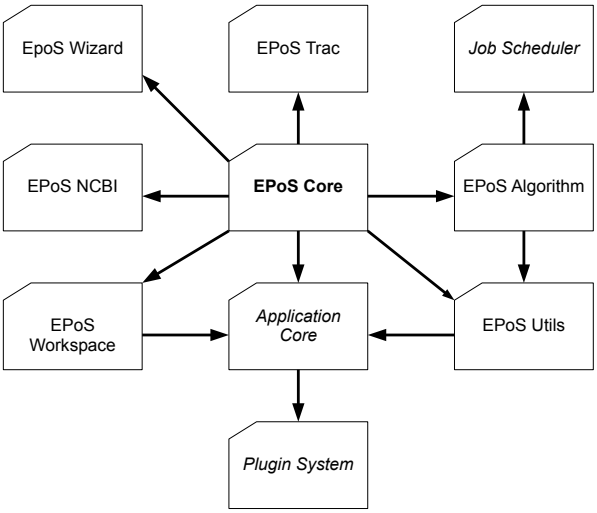


Figure 5.9: Overview of the core components that provide the basic functionality of EPoS. Shown are the dependencies between the EPoS Core and the surrounding core modules. The EPoS Core directly supports the *algorithm* component to submit jobs for execution and the *workspace* for data storage. The core also provides access to the web-service layer to connect to services like NCBI's GenBank.

EPoS Core Components

The basic functionality of EPoS is provided by the *EPoS Core*, which consists of several components:

- Plugin System and Application Core
- Algorithm integration and execution
- Workspace
- User interface and general utilities
- Web-Service integration

An overview of the EPoS Core components can be found in Fig. 5.9. Due to the comprehensiveness of EPoS, an introduction to all underlying concepts and mechanisms is beyond the scope of this thesis In the following, we take a closer look on some of the most important components and functions that have a direct impact on developers and users. For details see [80].

Essential to the EPoS framework is the *Plugin System*, which provides the extendable foundation of the software. The EPoS plugin system is based

on the *Java Plugin Framework (JPF)*[2], an open source plugin system that provides a runtime engine that dynamically discovers and loads plugins. A plugin specifies a set of XML-based meta-data, containing information about the plugin such as the name, the version and dependencies. This information is sufficient to construct a plugin graph and to boot existing plugins, but not yet enables to extend the system. Extensions are provided as part of the meta-data that describe a plugin. There are two descriptions the meta-data can contain: *Extension-Points* and *Extensions*. The former describe locations in the API or the framework that can be extended. The latter connect to those locations and provide concrete implementations for a given Extension-Point. Thus, extending the framework is basically done by defining an Extension for a given Extension Point. An Extension Point consists of a set of parameter definitions. Each definition must provide an identifier and can optionally specify its multiplicity and default value. The basic interplay of Extension-Points and Extensions pervades the EPoS framework on all levels. The EPoS core provides a variety of Extension-Points, which makes it easy to extend the system in any direction. For example, Extension-Points exist for the addition of new data types, visualizations or algorithms.

Upon the extension mechanism resides the *Application Core*. This is a general abstraction layer for applications integrated into the framework. It allows to define and add applications to the framework by means of functionality and logic, omitting user interface and layout. To auto create visual representations for integrated methods, the Application Core evaluates certain code annotations representing the parameters of an algorithm. For example, here are parts from the RaxML implementation:

```
@Property(value = "Bootstrap Replicates")
@Range(min = 1, max = Integer.MAX_VALUE)
 private int runs = 100;

@Component(separator = "General Configuration", innerComponent = true)
 public JComponent generalConfig(){
        return new RaxMLGeneralConfig(this);
}
@Component(value = "Multi Model partitioning")
public JComponent multiModelConfig(){
        return new RaxMLMultimodelSelection(this);

 @Input(name = "Input Alignment", description = "Source Alignment",
    minSize = 1, maxSize = 1, type = Alignment.class)
    public void setAlignment(Alignment alignment) {
    ...
}
```

Consider the *@Property* annotation: This annotation expresses that a property of an algorithm needs to be reflected in the generated user interface. The *interface generator* provided by the EPoS application framework evaluates properties annotated with *@Property* and is able to create visual representation for several types of data, including booleans, strings, numbers

[2] http://jpf.sourceforge.net/

and also collections of options. The interface generator translates different types of data into the corresponding visual representation, e.g. a boolean parameter is represented as a check-box, numbers as combo-boxes etc. As shown in the example, EPoS also supports another annotation, *@Component*, which can be added to methods returning a user interface component. This component is then automatically included this into the generated interface. In Fig. 5.10 the auto-generated user interface corresponding to the complete implementation is shown. The left side of the interface contains a white area. This was automatically created due to the *@Input* annotation, which states that the input to the method consists of exactly one alignment. All parameters are exactly reflected by the user interface. Furthermore, the component representing the parameters also provides functionality to search the Workspace and to add data. In this example, the search is limited to the specific data type, alignments, and a list of alignments is presented to the user. Also the drag-and-drop and copy-and-paste functions (data can be dragged-and-dropped or copy-and-pasted from the virtual file system or other views) is limited to the specified data type.

The interface generator is not restricted to algorithm interfaces. The same mechanisms and rules apply for example to *input* and *output* implementations and *application preferences*. Thus, with the interface generator provided by EPoS, not only parameters the for an algorithm, but also for new parser, preference panes etc. are generated without the need of implementing any user interface code. On the one hand, this strategy ensures that all interface structures are consistent and that the user has a homogeneous handling experience throughout the complete framework. On the other hand, it allows algorithm developers to concentrate on the implementation of the method without bothering with user interface design and implementation.

The *EPoS Algorithm* module supports algorithms and job execution. It simplifies algorithm development and the integration of algorithms into the user interface. Two types of algorithms can be integrated: *internal* methods that are implemented in Java and directly integrated into the system, and *external* methods that are based on an external executable that is started in a separate process. The basic ideas and concepts of algorithm integration concerning visual representation and user interface integration have been described above. Here we will present an overview of the main steps to add a new method to the framework. The EPoS core provides an Extension Point for algorithms that is used by all internal as well as external methods. The interface that has to be implemented is *Algorithm* and specifying the class that provides the implementation is the only mandatory parameter of the Extension Point. The EPoS framework provides an abstract Algorithm implementation, which can be extended. This forces the developer to implement a *run* method, which starts computations, and a *validate* method, which ensures that the algorithm is configured properly before execution.

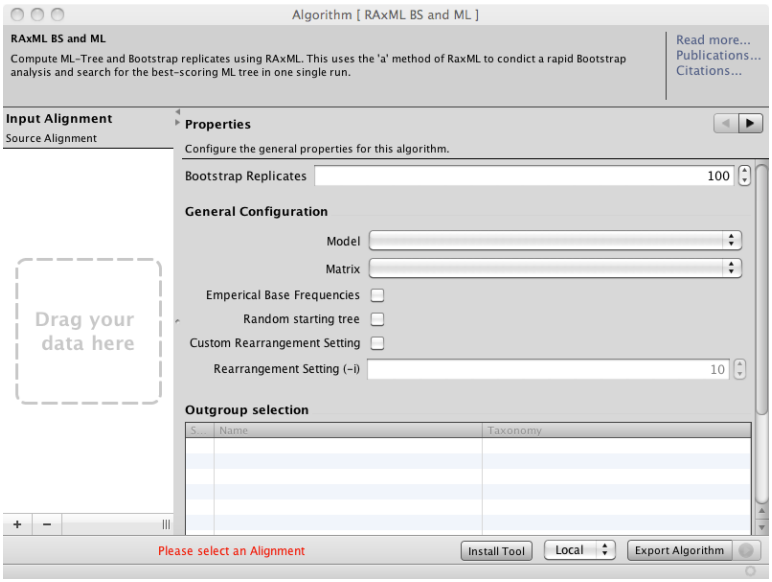


Figure 5.10: An auto-generated interface for RaXML.

Beside these methods, input and output data of an algorithm are essential for the system to integrate it properly. Two annotations exist, that cover these properties: *@Input* and *@Output*. External methods based on their own executable basically follow the same principles as described above. The main difference to internal methods is that they need to be executed in a separate process. Also the implementation of the execution is rather different from internal implementations. Instead of implementing the actual algorithm, adding an external method involves preparation and execution of an external process. To simplify the integration of external methods, EPoS provides a super class called *External Algorithm*, that can be extended to easily embed external methods. It forces the developer to implement two methods, one that prepares a working directory that contains the input files in proper format, and one that specifies the name and location of the external methods and the parameters in terms of command line parameters.

When an algorithm is fully integrated into the framework (the method itself and the visualization), the system is capable to identify algorithms that use a given set of input data. This means that the system is able to present all method with specific input to the user. Moreover, the system can be used to easily create *pipelines* of tools. With known input and output, algorithm executions can be concatenated easily.

As mentioned above, the EPoS Workspace provides ways to store, to retrieve and to query data. It consists of three components:

- The *Storage Service* is responsible for the fundamental persistence and

query functions. It abstracts the major database and persistence operations. It provides the basic database operations create, refresh, update and delete. The *JPAWorkspace* used in EPoS is an implementation based on the *Java Persistence API*(JPA)[3].

- The EPoS workspace implements the *Search Service* interface, which allows query based and full text search in the database. EPoS uses a combination of the *Lucene*[4] and the *Compass*[5] search engines. This allows developers to make their database objects searchable and to add them into the full text index by just one code annotation: *@Searchable*. This annotation is recognized by Workspace and annotated classes are automatically indexed. This annotation is recognized by the Workspace,

- The EPoS contains a *virtual file system*. This not only allows users to organize their data but also brings along some advantages: As part of the EPoS core, the virtual file system enables a deeper integration of virtual files and folders. This comprises that facilities using data from the workspace are also able to handle virtual files and folders directly. For example, if input data is handled by drag and drop between some components, in consequence the retrieving component sets it input to the content of the input folder, not to the folder itself.

The three mentioned components allows access to data and data management and also the addition of data objects. New data objects are registered as Extensions and managed by the Workspace. Two types of data objects can be added: "Simple" data objects, that are self contained and without any given functionality. Moreover, a base object can be extended, called *BaseEntity*. This creates a new database object with a common set of properties. Both types of data objects are registered as Extensions and managed by the Workspace, which includes all underlying database scheme modifications.

The EPoS NCBI module tightly integrates the NCBI Blast service, to submit jobs directly to the NCBI. The component simplifies web-service integration in general and implements the rules specific to the NCBI web-service layer in particular. We added support for:

- Blast - The NCBI Blast service allows users to submit Blast queries directly;

- Sequence Databases - EPoS allows users to search all NCBI nucleotide and protein sequence databases and import sequences into the workspace;

[3] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html
[4] http://lucene.apache.org
[5] http://compass-project.org

- Taxonomy Database - EPoS integrates the NCBi Taxonomy Database to search and fetch taxonomic information.

Further, we developed a web-service integration for the EPoS documentation and bug database. Both are based on Trac[6], which provides wiki and bug tracking functions.

The EPoS Job Scheduler module is based on a library, that was specifically developed for this purpose. *JQStat*[7] provides the ability to execute processes either locally or on a remote cluster. The JQStat library itself is extendable and implementations can be provided for different kinds of remote compute locations. Currently, the EPoS framework is able to integrate compute clusters based on the *Sun Grid Engine*[8].

To summarize, EPoS provides an easy-to-use, modular software framework with a consistent graphical user interface. On the one hand, developers can concentrate on the implementation of methods without the burden to create user interfaces. On the other hand, users are enabled to use a variety of methods and visualizations within the same environment.

[6] http://trac.edgewall.org/
[7] https://bio.informatik.uni-jena.de/trac/jqstat/
[8] http://wikis.sun.com/display/GridEngine/Home

CHAPTER 6

Conclusion and further research

The contribution of this thesis consists of three major parts: We introduced a new algorithm to construct supertrees, FlipCut supertree, we investigated the performance of several supertree methods using simulated data, and we presented EPoS, our modular and extendable software framework for phylogenetic analysis. In this chapter, we recall and summarize our results and present objectives and ideas for further research.

FlipCut supertree

We have presented a novel supertree approach named FlipCut supertree. It combines the objective function used by the Matrix Representation with Flipping (MRF) with the MinCut supertree method. In every step of the *FlipCut* supertree algorithm we seek a cut of minimum cost in the FlipCut graph and greedily proceed with the resulting subgraphs. We have presented a preprocessing method that ensures that undisputed siblings will be present in the constructed supertree. The FlipCut supertree method has polynomial running time and is extremely swift in practice.

Our FlipCut algorithm already produces supertrees of high quality regarding both accuracy and resolution in our simulations. In the presented version, FlipCut supertree is usually one of the most accurate polynomial-time methods and in some cases even nearly on par with Matrix Representation with Parsimony (MRP) and Matrix Representation with Flipping (MRF).

To further narrow the gap between FlipCut supertree and matrix-based approaches, we have two general ways to improve our method in the future: On the one hand, we can make improvements by using certain special properties of the FlipCut method, explained below. On the other hand, we can use some general, well-known strategies from supertree construction, including the use of a branch swapping heuristic to further optimize the objective function (costs) and contracting edges in the supertree with comparatively low support. We will explain some of them in more detail below. Our general objective when improving the FlipCut supertree algorithm in the future is to preserve the running time of the method. Our aim is to fulfill all three criteria regarding fast running time, high accuracy and a high resolution pointed out by [23], see also Chapter 2, page 36. Certainly, these criteria have to be ensured before we can address further challenges that arise when using FlipCut supertree as part of a divide-and-conquer framework.

FlipCut supertree offers a fundamental advantage over MinCut supertree and its derivates: We have defined a global objective function (the

number of flips in the input matrix) that we want to minimize. Using this objective function has been shown to result in supertrees of good quality [39]. Besides the theoretical amenity of such an objective function, this has practical implications: We can compare the quality of different supertrees based on our objective function as well as the quality of *partial solutions*. This allows us to compare solutions that are built by a randomized version of our algorithm and we can use a beam search in which we keep several suboptimal solutions "alive" when building the trees. We conjecture that these modifications will further improve the quality of FlipCut supertrees, as it will address the problem of greedily separating a single taxon in an early stage.

In the future we also want to evaluate methods for weighting the edges in the FlipCut graph. This includes using bootstrap values as character weights, as well as inferring ancestry using branch lengths as introduced by Willson [172], see page 2.3.4. The "Edge & Level" weighting presented in this thesis, is merely meant to demonstrate the impact of weighting edges on the quality of constructed supertrees.

A method that has been proven to be beneficial to other supertree methods is to subsequently use a *branch swapping search heuristic* that tries to further minimize the objective function (costs). We assume that the use of branch swapping can correct "rogue taxa" in the FlipCut supertree. Due to the quality of the initial FlipCut supertree presumably only few tree rearrangement operations will be necessary, so that the increase in running time should be moderate. Branch swapping heuristics have repeatedly been used in computational phylogenetics and are used by most approaches for Maximum Parsimony and Maximum Likelihood. In supertree analysis, several methods of this kind have been reported [7, 39, 108]. Complementing the FlipCut supertree algorithm with a branch swapping heuristic is therefore a natural idea.

Polynomial supertree methods revisited

We presented a large-scale simulation study to assess and compare the accuracy and the resolution of various polynomial time supertree methods and the matrix-based supertree methods MRP and MRF. We used two different datasets in our simulations: The first data set was created according to the standard protocol used to evaluate supertree methods whereas the second data set was generated using the SMIDgen protocol (see page 54 and 55). Our results show that recent polynomial time supertree methods can sometimes compete with the classical MRP approach while guaranteeing a significantly better running time.

As mentioned in Chapter 1 and Chapter 2, one future approach to build larger clades of the Tree of Life might be a divide-and-conquer framework coupled with supertree methods. As stated by Bininda-Emonds [23] "particular focus needs to be placed on developing fast supertree methods that yield ac-

curate and well resolved solutions".

The incorporation of branch length information from the input trees, as done by Build-with-Distances (BWD) and the FlipCut supertree method, significantly enhances the graph-based supertree approach concerning accuracy without sacrificing short running times. To improve the applicability of methods using branch length in real-world studies, further investigations are necessary: On the one hand, methods to reconcile different distances need to be evaluated. On the other hand, as indicated above, different weighting schemes for edges in the BWD graph and the FlipCut graph should be considered, including the use of bootstrap values from the input trees.

Veto approaches such as PhySIC have certain appealing properties but also certain drawbacks: the resolution of reconstructed supertrees rapidly decreases with small taxon overlap and/or when there are too many conflicts among input trees. On our standard protocol data set, we saw that PhySIC only returned star trees. Certainly, the setting we used is a rather extreme case, as we decided not to use the PhySIC preprocessing step, which collapses branches from the input trees with low bootstrap support. This preprocessing would improve the results of PhySIC and we assume that other methods could benefit from it as well. The effects of different preprocessing steps as well as postprocessing steps on different methods are, however, beyond the scope of this thesis. In general, this is certainly an interesting aspect for further investigation.

PhySIC_IST, in combination with STC preprocessing, significantly enhances the veto approach in terms of resolution and accuracy, but at the cost of taxa not being included in the supertree. The extent to which *PhySIC_IST* excludes taxa clearly depends on the used data set. The data used in our simulation represents a rather extreme case, and data that is carefully selected by hand should be more suitable for *PhySIC_IST*. Furthermore, analysis of the triplet distance (or triplet type II error for our study) reveals that veto approaches can be too conservative, whereas voting approaches as BWD can be too decisive. For future polynomial supertree methods, a compromise between the two approaches seems desirable.

The SuperTriplets (ST) [128] method is an example of the combination of an initial heuristic step with a branch swapping heuristic. Our simulations show that this strategy yields supertrees with good accuracy and resolution. As indicated above, it is interesting to investigate such a post-processing step not only for our *FlipCut* method, but also for other methods minimizing or maximizing an objective function.

Simulation studies as the one conducted here have the general advantage that we can compare the reconstructed supertree with the true model tree. Our findings are generally in accordance with other supertree simulation studies. Both Eulenstein *et al.* [56] and Kupczok *et al.* [102] found that MRP outperforms MinCutSupertree (MC) and Modified MinCutSu-

PERTREE (MMC). Eulenstein *et al.* [56] also found that MRF shows a similar performance as MRP. Regarding SDM, we decided to use SDM+BioNJ* in our study because of running time considerations and incomplete distance matrices. Unfortunately, SDM+BioNJ* did not show competitive results. According to Buerki *et al.* [34] and Kupczok *et al.* [102] SDM+MW* performs much better than SDM+BioNJ*, but this again comes at the price of significantly increased running times. The BWD method has not been considered in previous simulation studies, but our results clearly indicate that BWD supertrees can be of good quality.

Despite all developments in the field during the last 19 years, MRP must still be regarded a "gold standard" of supertree reconstruction; notwithstanding its advanced age, there appears to be no method that clearly outperforms it. In particular, many methods (even those claiming polynomial running time) have running times that are unfavorable when compared to highly developed MRP search heuristics. The empirical study of Buerki *et al.* [34] showed, somewhat surprisingly, that supertree quality for MC and MMC are comparable to MRP and MRF. Other methods such as average consensus or split fit showed much poorer performance. But according to Swenson *et al.* [162], the topological distance to source trees and the topological distance to the true is is only weakly correlated, so empirical studies might be misleading.

Today, a divide-and-conquer approach to large scale phylogenetic inference using polynomial supertree methods as sub-procedure has not fully been realized. Meanwhile, we propose to use several of the supertree methods presented here for medium-sized studies with hundreds of taxa and tens of trees, and to manually compare the results. But if the sheer size of the problem makes it impossible to use matrix-representation methods such as MRP, novel polynomial-time methods such as those investigated in our simulations can greatly improve the quality of results, compared to early methods such as MC or MMC.

Although formal supertree methods have been around for a quarter of a century, our simulations show that there is still much room for improvement and that novel ideas and methods can greatly improve the quality of constructed supertrees.

EPoS

We presented EPoS, our modular software framework for phylogenetic inference. We hope that EPoS will be a supporting tool for the different members of the phylogenetic community. The simplicity of the underlying plugin mechanism allows developers to easily integrate their own tools and algorithm into the framework, and to benefit from methods provided by others. The process of connecting algorithms to data and data to visualization is completely covered by the system. Developers do not have to worry about persistency and data integrity. The core modules make it easy to establish

connections to external data sources, using web services and local or remote databases. Biologists conducting a phylogenetic study benefit from the variety of integrated programs and visualizations, which means that they do not have to adapt to various environments and programs. Due to the modular and extendable nature of EPoS, several possible directions for future work come into mind. For example, the system could be extended with methods and visualizations for phylogenetic networks. Our aims for the near future include to integrate TreeBASE[1] as another data source. Furthermore, we want to create a plugin that supports all steps of a supertree simulation study, from the generation of data to the evaluation of the supertrees, in a comfortable manner.

[1] http://treebase.org/treebase-web/home.html

APPENDIX A

Appendix for Chapter 4

A.1 Results Standard Protocol Data Set 48 taxa model trees

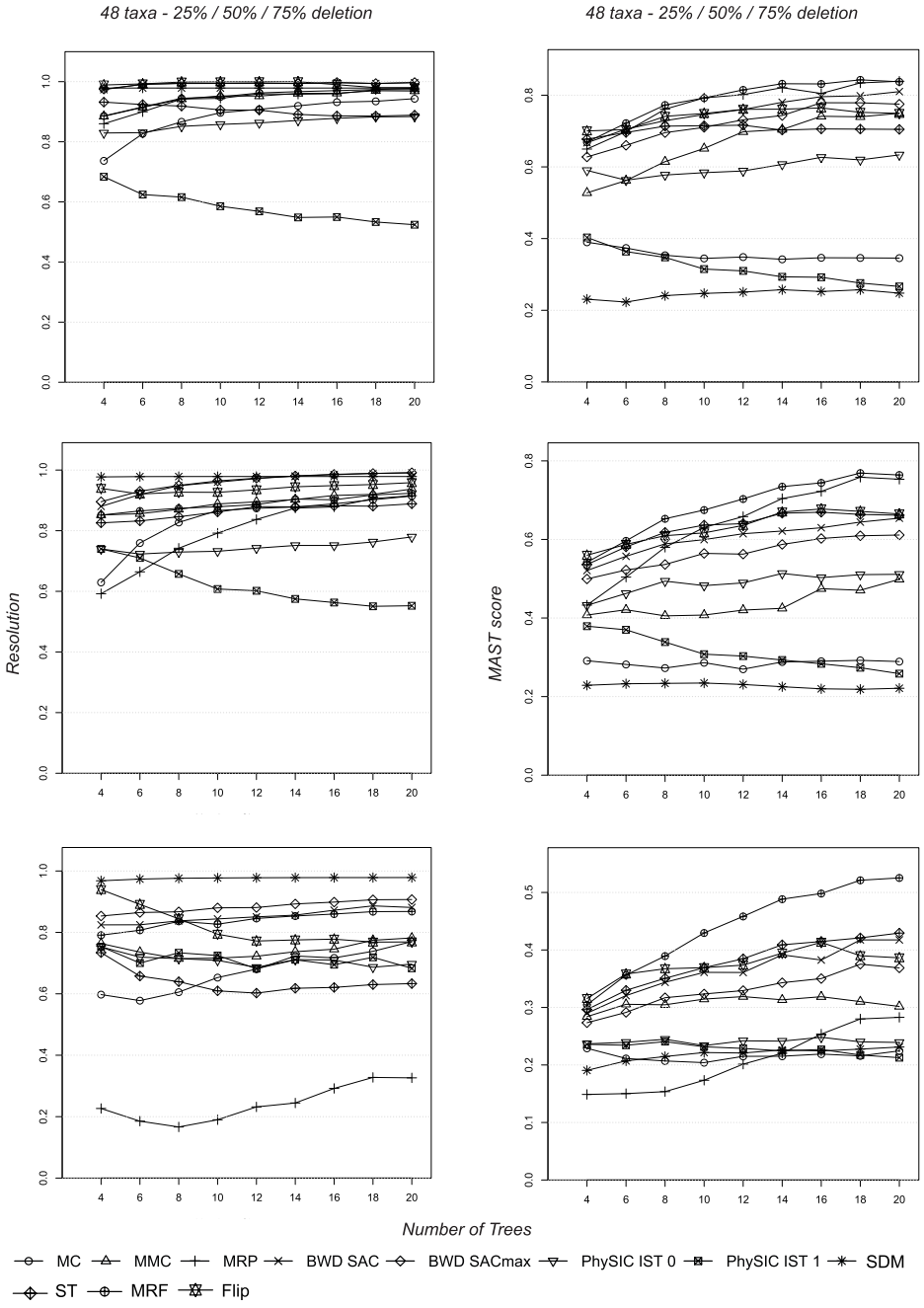


Figure A.1: The left column of the figure shows the average resolution of the supertrees constructed from model trees with 48 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average MAST score of the supertrees constructed from model trees with 48 taxa.

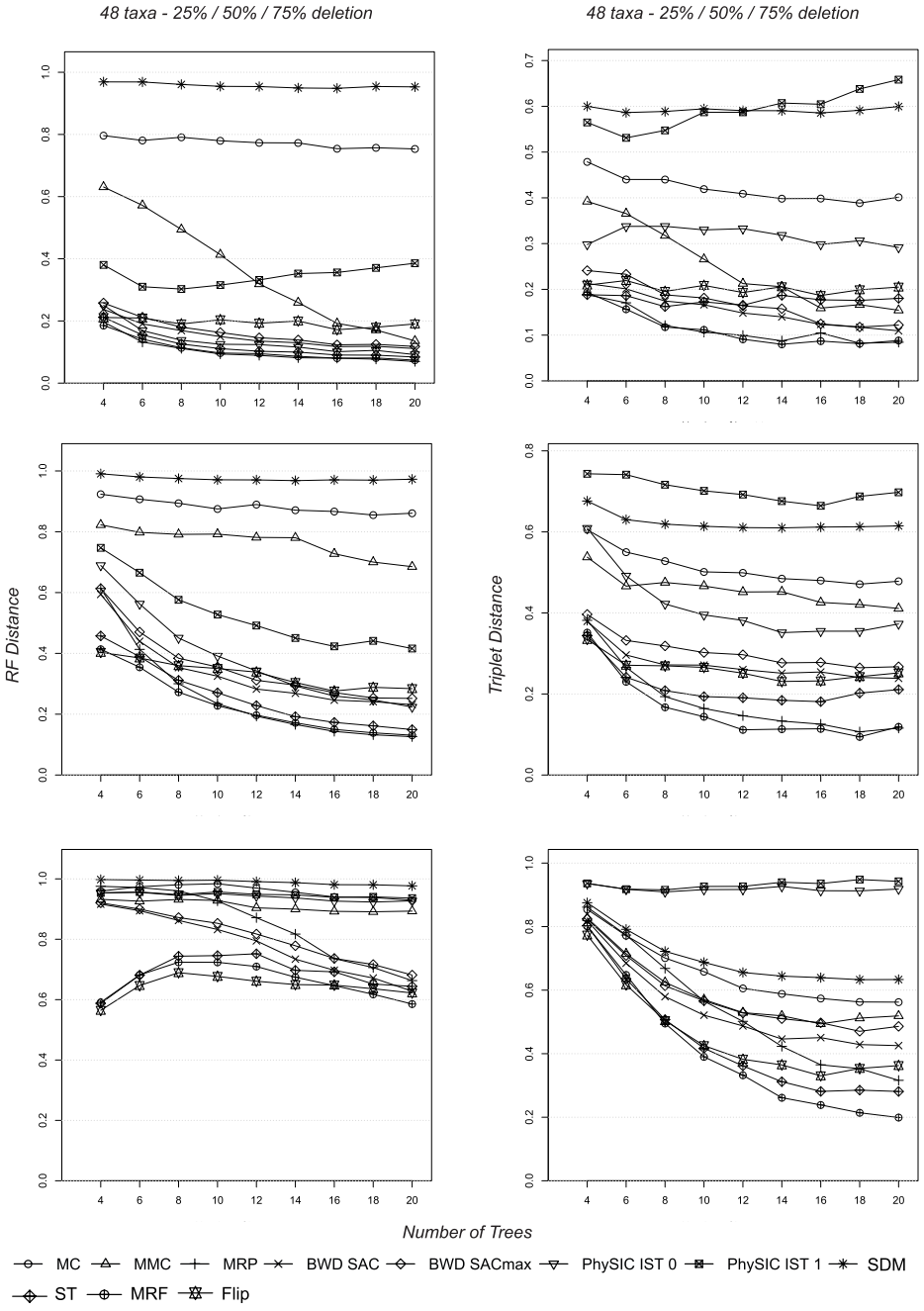


Figure A.2: The left column of the figure shows the average RF distance of the supertrees constructed from model trees with 48 taxa and different taxon deletion ratios (top 25 %, middle 50 %, bottom 75 %). The right column shows the average triplet distance of the supertrees constructed from model trees with 48 taxa.

Bibliography

[1] E. N. Adams III. Consensus techniques and the comparison of taxonomic trees. *Syst. Zool.*, 21(4):390–397, 1972. 87

[2] R. Agarwala, D. Fernández-Baca and F. Andez-baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM J. Comput.*, 23:1216–1224, 1994. 11

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J. Comput.*, 10(3):405–421, 1981. 4, 20, 23, 25, 26, 27

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990. 87

[5] N. Amenta, F. Clarke and K. S. John. A linear-time majority tree algorithm. In *Proc. of Algorithms in Bioinformatics, WABI 2003*, 2003. 87

[6] W. J. Baker, V. Savolainen, C. B. Asmussen-Lange, M. W. Chase, J. Dransfield, F. Forest, M. M. Harley, N. W. Uhl, and M. Wilkinson. Complete generic-level phylogenetic analyses of palms (arecaceae) with comparisons of supertree and supermatrix approaches. *Systematic Biology*, 58(2):240–256, 2009. 49

[7] M. S. Bansal, J. G. Burleigh, O. Eulenstein and D. Fernández-Baca. Robinson-Foulds supertrees. *Algorithms Mol. Biol.*, 5:18, 2010. 35, 98

[8] M. Barrett, M. J. Donoghue and E. Sober. Against consensus. *Systematic Biology*, 40(4):486–493, 1991. 14

[9] B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41(1):3–10, 1992. 19, 23, 24

[10] B. R. Baum and M. A. Ragan. The MRP method. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, chapter 1, pages 17–34. Kluwer Academic, 2004. 24

[11] J. Bergsten. A review of long-branch attraction. *Cladistics*, 21(2):163–193, 2005. 13

[12] V. Berry and F. Nicolas. Maximum agreement and compatible supertrees. *J. of Discrete Algorithms*, 5:564–591, 2007. 35

[13] V. Berry and C. Semple. Fast computation of supertrees for compatible phylogenies with nested taxa. *Syst. Biol.*, 55(2):270–288, 2006. 27, 88

[14] R. B. Bevan, D. Bryant and B. F. Lang. Accounting for gene rate heterogeneity in phylogenetic inference. *Syst Biol*, 56(2):194–205, 2007. 16

[15] O. Bininda-Emonds, J. Gittleman and M. Steel. The (super)tree of life: Procedures, problems, and prospects. *ANNUAL REVIEW OF ECOLOGY AND SYSTEMATICS*, 33:265–289, 2002. 17

[16] O. Bininda-Emonds, K. E. Jones, S. Price, M. Cardillo, R. Grenyer and A. Purvis. *Garbage in, garbage out: data issues in supertree construction.* Kluwer Academic Publishers, Dordrecht, 2004. 14

[17] O. R. Bininda-Emonds and H. N. Bryant. Properties of matrix representation with parsimony analyses. *Systematic Biology*, 47(3):497–508, 1998. 15, 25

[18] O. R. P. Bininda-Emonds. Novel versus unsupported clades: assessing the qualitative support for clades in MRP supertrees. *Syst. Biol.*, 52(6):839–848, 2003. 15, 17, 20, 25, 32, 49, 50

[19] O. R. P. Bininda-Emonds. The evolution of supertrees. *Trends Ecol. Evol.*, 19(6): 315–322, 2004. 14, 15, 17, 19

[20] O. R. P. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 4 of *Computational Biology Series*. Kluwer Academic, 2004. 19, 20, 35

[21] O. R. P. Bininda-Emonds. Trees versus characters and the supertree/supermatrix paradox. *Systematic Biology*, 53(2):356–359, 2004. 14, 15, 17

[22] O. R. P. Bininda-Emonds. Supertree construction in the genomic age. *Methods Enzymol.*, 395:745–757, 2005. 16

[23] O. R. P. Bininda-Emonds. The future of supertrees: bridging the gap with supermatrices. In *Proc. of Willi-Hennig-Symposium 2009*, volume 3 Supplement of *Palaeodiversity*, pages 99–106, 2010. 17, 36, 97, 98

[24] O. R. P. Bininda-Emonds and M. J. Sanderson. Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Syst. Biol.*, 50(4):565–579, 2001. 49, 50

[25] O. R. P. Bininda-Emonds and A. Stamatakis. Taxon sampling versus computational complexity and their impact on obtaining the tree of life. In J. A. N. Hotkinson, T. R. & Parnell, editor, *Reconstructing the Tree of Life: taxonomy and systematics of species rich taxa*, pages 77–95. CRC Press, Boca Raton, 2006. 17, 36

[26] O. R. P. Bininda-Emonds, K. E. Jones, S. A. Price, R. Grenyer, M. Cardillo, M. Habib, A. Purvis, and J. L. Gittleman. Supertrees are a necessary not-so-evil: a comment on gatesy et al. *Syst. Biol.*, 52(5):724–729, 2003. 14, 17

[27] S. Böcker. Towards a data reduction for the minimum-flip supertree problem. Technical report, Cornell University Library, 2011. arXiv:1104.4471v1. 26

[28] S. Böcker, F. Hüffner, A. Truss and M. Wahlström. A faster fixed-parameter approach to drawing binary tanglegrams. In *Proc. of International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *Lect. Notes Comput. Sci.*, pages 38–49. Springer, Berlin, 2009. 87

[29] S. Böcker, B. Bui, F. Nicolas and A. Truss. Intractability of the minimum flip supertree problem and its variants. Technical report, Cornell University Library, 2011. arXiv:1112.4536v1. 26

[30] M. Brinkmeier. A simple and fast min-cut algorithm. *Theory Comput. Syst.*, 41(2): 369–380, 2007. 42

[31] M. Brinkmeyer, T. Griebel and S. Böcker. FlipCut supertrees: Towards matrix representation accuracy in polynomial time. In *Proc. of Computing and Combinatorics Conference (COCOON 2011)*, volume 6842 of *Lect. Notes Comput. Sci.*, pages 37–48. Springer, Berlin, 2011. 88

[32] D. Bryant and M. A. Steel. Extension operations on sets of leaf-labelled trees. *Adv. Appl. Math.*, 16(4):425–453, 1995. 27

[33] D. Bryant, C. Semple and M. A. Steel. Supertree methods for ancestral divergence dates and other applications. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, chapter 6, pages 129–150. Kluwer Academic, 2004. 88

[34] S. Buerki, F. Forest, N. Salamin and N. Alvarez. Comparative performance of supertree algorithms in large data sets using the soapberry family (Sapindaceae) as a case study. *Syst Biol*, 60(1):32–44, 2011. 49, 100

[35] J. H. Camin and R. R. Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1974. 11

[36] J. A. Cavender. Taxonomy with confidence. *Mathematical Biosciences*, 40(34):271 – 280, 1978. 12

[37] D. Chen, O. Eulenstein, D. Fernández-Baca and M. Sanderson. Supertrees by flipping. In *Proc. of Conference on Computing and Combinatorics (COCOON 2002)*, volume 2387 of *Lect. Notes Comput. Sci.*, pages 391–400. Springer, Berlin, 2002. 4, 25, 26

[38] D. Chen, L. Diao, O. Eulenstein, D. Fernández-Baca and M. J. Sanderson. Flipping: A supertree construction method. In M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, B. Mirkin and F. S. Roberts, editors, *Bioconsensus: DIMACS Working Group Meetings on Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 135–160. Amer. Math. Soc., 2003. 4, 25, 49

[39] D. Chen, O. Eulenstein, D. Fernández-Baca and J. G. Burleigh. Improved heuristics for minimum-flip supertree construction. *Evol. Bioinform. Online*, 2:347–356, 2006. 19, 25, 88, 98

[40] D. Chen, O. Eulenstein, D. Fernández-Baca and M. Sanderson. Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(2):165–173, 2006. 25, 26, 41, 50

[41] M. Chimani, S. Rahmann and S. Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proc. of ACM Conf. on Bioinformatics and Computational Biology (ACM-BCB 2010)*, pages 147–153. ACM, 2010. 26

[42] P. T. Chippindale and J. J. Wiens. Weighting, partitioning, and combining characters in phylogenetic analysis. *Systematic Biology*, 43(2):278–287, 1994. 14

[43] M. Constantinescu and D. Sankoff. An efficient algorithm for supertrees. *J. Classif.*, 12:101 – 112, 1995. 28

[44] J. Cotton, C. Slater and M. Wilkinson. Discriminating supported and unsupported relationships in supertrees using triplets. *Syst. Biol.*, 55(2):345–350, 2006. 20, 25, 32

[45] C. J. Creevey and J. O. Mcinerney. Clann: investigating phylogenetic information through supertree analyses. *Bioinformatics*, 21(3):390–2, 2005. 35

[46] A. Criscuolo and O. Gascuel. Fast NJ-like algorithms to deal with incomplete distance matrices. *BMC Bioinformatics*, 9(1):166, 2008. 34, 55

[47] A. Criscuolo, V. Berry, E. J. P. Douzery and O. Gascuel. SDM: a fast distance-based approach for (super) tree building in phylogenomics. *Syst. Biol.*, 55(5):740–755, 2006. 34, 55

[48] W. Day. Analysis of quartet dissimilarity measures between undirected phylogenetic trees. *Syst. Biol.*, 35:325–333, 1986. 56

[49] M. O. Dayhoff, R. M. Schwartz and B. C. Orcutt. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure*, volume 5, chapter 22, pages 345–351. National Biomedical Research Foundation, 1978. 13

[50] A. de Queiroz and J. Gatesy. The supermatrix approach to systematics. *Trends Ecol. Evol.*, 22(1):34–41, 2007. 17

[51] F. Delsuc, H. Brinkmann and H. Philippe. Phylogenomics and the reconstruction of the tree of life. *Nat. Rev. Genet.*, 6(5):361–375, 2005. 8

[52] A. C. Driskell, C. Ané, J. G. Burleigh, M. M. McMahon, B. C. O'meara and M. J. Sanderson. Prospects for building the tree of life from large sequence databases. *Science*, 306(5699):1172–1174, 2004. 16

[53] B. E. Dutilh, V. van Noort, R. T. J. M. van der Heijden, T. Boekhout, B. Snel and M. A. Huynen. Assessment of phylogenomic and orthology approaches for phylogenetic inference. *Bioinformatics*, 23(7):815–824, 2007. 49

[54] W. W. Eckerson. Three tier client/server architecture: Achieving scalability, performance, and efficiency in client server applications. *Open Information Systems*, 10(1), 1995. 89

[55] R. C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004. 87

[56] O. Eulenstein, D. Chen, J. G. Burleigh, D. Fernández-Baca and M. J. Sanderson. Performance of flip supertree construction with a heuristic algorithm. *Syst. Biol.*, 53(2):299–308, 2004. 25, 26, 35, 49, 50, 99, 100

[57] J. Farris, A. Kluge and M. Eckhardt. A numerical approach to phylogenetic systemetics. *Syst. Zool.*, 19:172–189, 1970. 24

[58] J. S. Farris. Phylogenetic analysis under dollo's law. *Syst. Zool.*, 26:77–88, 1977. 11

[59] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, Massachusetts, 2004. 5, 11, 12

[60] J. Felsenstein. Cases in which parsimony or compatibility methods will be positively misleading. *Systematic Zoology*, 27:401–410, 1978. 12

[61] J. Felsenstein. Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, 17(6):368–376, 1981. 13

[62] D. Fern. The perfect phylogeny problem. *Steiner Trees in Industry*, pages 1–32, 2000. 11

[63] W. M. Fitch. Homology: a personal view on some of the problems. *Trends in Genetics*, 16(5):227 – 231, 2000. 7

[64] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Syst. Zool.*, 20(4):406–416, 1971. 11, 12

[65] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155: 279–84, 1967. 13

[66] D. Fitzpatrick, M. Logue, J. Stajich and G. Butler. A fungal phylogeny based on 42 complete genomes derived from supertree and combined gene analysis. *BMC Evolutionary Biology*, 6(1):99, 2006. 49

[67] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, New Jersey, 1962. 43

[68] L. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Adv. Appl. Math.*, 3(1):43–49, 1982. 12, 13, 24

[69] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. 88

[70] J. Gatesy and M. Springer. A critique of matrix representation with parsimony supertrees. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, chapter 17, pages 369–388. Kluwer Academic, 2004. 14, 17

[71] J. Gatesy, C. Matthee, R. DeSalle and C. Hayashi. Resolution of a supertree/supermatrix paradox. *Systematic Biology*, 51(4):652–664, 2002. 14, 15

[72] J. Gatesy, R. H. Baker and C. Hayashi. Inconsistencies in arguments for the supertree approach: supermatrices versus supertrees of crocodylia. *Systematic Biology*, 53(2):342–355, 2004. 14, 15, 49

[73] N. Goldman and Z. Yang. Introduction. statistical and computational challenges in molecular phylogenetics and evolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1512):3889–3892, 2008. 3

[74] P. A. Goloboff. Minority rule supertrees? MRP, compatibility, and minimum flip may display the least frequent groups. *Cladistics*, 21(3):282–294, 2005. 15, 20, 25, 32

[75] P. A. Goloboff and D. Pol. Semi-strict supertrees. *Cladistics*, 18(5):514–525, 2002. 15, 20, 25, 32

[76] P. A. Goloboff, J. S. Farris and K. C. Nixon. Tnt, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786, 2008. 24

[77] G. H. Gonnet, M. A. Cohen and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256(5062):1443–1445, 1992. 13

[78] A. D. Gordon. On the assessment and comparison of classifications. In R. Tomassine, editor, *Analyse de Données et Informatique*, pages 149–160. Le Chesnay, INRIA, France, 1980. 57

[79] A. D. Gordon. Consensus supertrees: the synthesis of rooted trees containing overlapping sets of labeled leaves. *J. Classif.*, 3(2):335–348, 1986. 19

[80] T. Griebel. *Modular Software for Phylogenetic Analysis Connecting Algorithms, Applications and Data*. PhD thesis, Friedrich-Schiller-Universität Jena, 2012. 79, 90

[81] T. Griebel, M. Brinkmeyer and S. Böcker. EPoS: a modular software framework for phylogenetic analysis. *Bioinformatics*, 24(20):2399–2400, 2008. 47, 55, 56, 57

[82] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst Biol*, 52(5):696–704, 2003. 13

[83] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1): 19–28, 1991. 10

[84] J. X. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J. Algorithms*, 17(3):424 – 446, 1994. 44

[85] M. Hasegawa, H. Kishino and T. Yano. Dating of the humanape splitting by a molecular clock of mitochondrial DNA. *J. Mol. Evol.,*, 22:160–174, 1985. 13

[86] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–10919, 1992. 13

[87] M. R. Henzinger, V. King and T. Warnow. Constructing a tree from homeomorphic subtrees with applications to computational evolutionary biology. *Algorithmica*, 24: 1– 13, 1999. 27

[88] J. W. Higdon, O. R. P. Bininda-Emonds, R. M. D. Beck and S. H. Ferguson. Phylogeny and divergence of the pinnipeds (carnivora: Mammalia) assessed using a multigene dataset. *BMC Evol. Biol.*, 7:216, 2007. 7, 17, 52

[89] P. G. Higgs and T. K. Attwood. *Bioinformatics and Molecular Evolution*. Wiley-Blackwell, 2005. 5

[90] K. Howe, A. Bateman and R. Durbin. Quicktree: building huge neighbour-joining trees of protein sequences. *Bioinformatics*, 18(11):1546–1547, 2002. 87

[91] D. H. Huson, S. M. Nettles and T. J. Warnow. Disk-Covering, a fast-converging method for phylogenetic tree reconstruction. *J. Comput. Biol.*, 6(3-4):369–386, 1999. 17, 34, 35, 36

[92] D. H. Huson, L. Vawter and T. J. Warnow. Solving large scale phylogenetic problems using DCM2. In *Proc. of Intelligent Systems for Molecular Biology (ISMB 1999)*, pages 118–129, 1999. 35

[93] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, volume III, chapter 24, pages 21–132. Academic Press, New York, 1969. 13

[94] D. R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. 42

[95] R. Kashyap and S. Subas. Statistical estimation of parameters in a phylogenetic tree using a dynamic model of the substitutional process. *Journal of Theoretical Biology*, 47(1):75 – 101, 1974. 13

[96] K. Katoh, K. ichi Kuma, H. Toh and T. Miyata. MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res.*, 33(2):511–518, 2005. 87

[97] T. M. Keane, C. J. Creevey, M. M. Pentony, T. J. Naughton and J. O. Mclnerney. Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. *BMC Evol Biol*, 6:29, 2006. 87

[98] M. Kearney. Fragmentary taxa, missing data, and ambiguity: mistaken assumptions and conclusions. *Syst Biol*, 51(2):369–381, 2002. 16

[99] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120, 1980. 13

[100] A. G. Kluge. A concern for evidence and a phylogenetic hypothesis of relationships among epicrates (boidae, serpentes). *Systematic Zoology*, 38(1):pp. 7–25, 1989. 14

[101] E. V. Koonin. Orthologs, paralogs, and evolutionary genomics. *Annual Review of Genetics*, 39(1):309–338, 2005. 7

[102] A. Kupczok, H. A. Schmidt and A. von Haeseler. Accuracy of phylogeny reconstruction methods combining overlapping gene data sets. *Algorithms Mol. Biol.*, 5: 37, 2010. 35, 49, 50, 99, 100

[103] F. J. Lapointe and G. Cucumel. The average consensus procedure: combination of weighted trees containing identical or overlapping sets of taxa. *Systematic Biology*, 46(2):306–312, 1997. 34

[104] F.-J. Lapointe, J. A. W. Kirsch and J. M. Hutcheon. Total evidence, consensus, and bat phylogeny: A distance-based approach. *Molecular Phylogenetics and Evolution*, 11(1):55 – 66, 1999. 17

[105] L. C. Lapointe FJ. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, chapter EVERYTHING YOU ALWAYS WANTED TO KNOW ABOUT THE AVERAGE CONSENSUS AND MORE., pages 87–106. Kluwer Academic Publishers, 2004. 34

[106] W. J. Le Quesne. The uniquely evolved character concept and its cladistic application. *Systematic Biology*, 23(4):513–517, 1974. 11

[107] C. Levasseur and F.-J. Lapointe. Total evidence, average consensus and matrix representation with parsimony: What a difference distances make. *Evol. Bioinform.*, 2:249–253, 2006. 49, 50

[108] H. T. Lin, J. G. Burleigh and O. Eulenstein. Triplet supertree heuristics for the tree of life. *BMC Bioinformatics*, 10(Suppl 1):S8, 2009. 35, 98

[109] W. P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997. 8

[110] M. M. McMahon and M. J. Sanderson. Phylogenetic supermatrix analysis of genbank sequences from 2228 papilionoid legumes. *Systematic Biology*, 55(5):818–836, 2006. 16

[111] S. Moran, S. Rao and S. Snir. Using semi-definite programming to enhance supertree resolvability. In *Algorithms in Bioinformatics*, volume 3692 of *Lect. Notes Comput. Sci.*, pages 89–103. Springer, Berlin, 2005. 35

[112] C. Mosses. *Triplet Supertrees*. PhD thesis, University of Aarhus, Denmark, 2005. 35

[113] J. Neyman. Molecular studies of evolution: a source of novel statistical problems. In S. S. Gupta and J. Yackel, editors, *Statistical decision theory and related topics*, pages 1–27. Academic Press, New York, 1971. 13

[114] M. P. Ng and N. C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Appl. Math.*, 69(1-2):19–31, 1996. 27, 28

[115] C. Notredame, D. G. Higgins and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302(1):205–217, 2000. 87

[116] R. D. M. Page. Modified mincut supertrees. In *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2002)*, volume 2452 of *Lect. Notes Comput. Sci.*, pages 537–552. Springer, Berlin, 2002. 20, 29, 43, 56, 88

[117] I. Pe'er, T. Pupko, R. Shamir and R. Sharan. Incomplete directed perfect phylogeny. *SIAM J. Comput.*, 33(3):590–607, 2004. 4, 25, 37, 38, 39, 40, 41, 43

[118] E. Pennisi. Modernizing the tree of life. *Science*, 300(5626):1692–1697, 2003. 17

[119] J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. *Math. Prog. Study*, 13:8–16, 1980. 45

[120] D. Pisani and M. Wilkinson. Matrix representation with parsimony, taxonomic congruence, and total evidence. *Syst. Biol.*, 51:151–155, 2002. 15

[121] D. Posada and K. A. Crandall. Modeltest: testing the model of dna substitution. *Bioinformatics*, 14(9):817–818, 1998. 87

[122] A. Purvis. A composite estimate of primate phylogeny. *Phil. Trans. R. Soc. Lond.*, B 348:405–421, 1995. 15

[123] A. Purvis. A modification to baum and ragan's method for combining phylogenetic trees. *Systematic Biology*, 44(2):251–255, 1995. 24

[124] M. A. Ragan. Matrix representation in reconstructing phylogenetic relationships among the eukaryotes. *Biosystems*, 28(1-3):47–55, 1992. 19, 24

[125] M. A. Ragan. Phylogenetic inference based on matrix representation of trees. *Mol. Phylogenet. Evol.*, 1(1):53–58, 1992. 23

[126] A. Rambaut and N. C. Grassly. Seq-gen: an application for the monte carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, 13(3):235–238, 1997. 52

[127] V. Ranwez, V. Berry, A. Criscuolo, P.-H. Fabre, S. Guillemot, C. Scornavacca and E. J. P. Douzery. PhySIC: a veto supertree method with desirable properties. *Syst. Biol.*, 56(5):798–817, 2007. 20, 31, 33, 55, 88

[128] V. Ranwez, A. Criscuolo and E. J. P. Douzery. SuperTriplets: a triplet-based supertree approach to phylogenomics. *Bioinformatics*, 26(12):i115–i123, 2010. 35, 55, 88, 99

[129] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Math. Biosci.*, 53(1-2):131–147, 1981. 56, 87

[130] S. Roch. A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(1):92–94, 2006. 13

[131] A. G. Rodrigo. A comment on baum's method for combining phylogenetic trees. *Taxon*, 42(3):631–636, 1993. 14

[132] A. G. Rodrigo. On combining cladograms. *Taxon*, 45(2):pp. 267–274, 1996. 14

[133] F. Ronquist and J. P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, 2003. 24, 87

[134] U. Roshan, B. Moret, T. Warnow and T. Williams. Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proc. of IEEE Computational Systems Bioinformatics Conference (CSB 2004)*, pages 98–109, 2004. 34, 35

[135] U. Roshan, B. M. E. Moret, T. L. Williams and T. Warnow. Performance of supertree methods on various data set decompositions. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, chapter 3, pages 301–328. Kluwer Academic, 2004. 18, 34, 36

[136] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4(4):406–425, 1987. 13

[137] N. Salamin, T. R. Hodkinson and V. Savolainen. Building supertrees: An empirical assessment using the grass family (poaceae). *Systematic Biology*, 51(1):136–150, 2002. 49

[138] M. Salemi, P. Lemey and A. Vandamme. *The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press, 2009. 5

[139] M. J. Sanderson. r8s: inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock. *Bioinformatics*, 19(2):301–302, 2003. 52

[140] M. J. Sanderson and A. C. Driskell. The challenge of constructing large phylogenetic trees. *Trends Plant Sci.*, 8(8):374–379, 2003. 16

[141] M. J. Sanderson, A. Purvis and C. Henze. Phylogenetic supertrees: Assembling the trees of life. *Trends Ecol. Evol.*, 13(3):105–109, 1998. 16

[142] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal on Applied Mathematics*, 28(1):35–42, 1975. 12

[143] 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*, pages 253–263. Addison-Wesley: Reading, Mass., 1983. 12

[144] D. Sankoff and P. Rousseau. Locating the vertices of a steiner tree in an arbitrary metric space. *Math. Program.*, 9:240–246, 1975. 11, 12

[145] H. A. Schmidt, K. Strimmer, M. Vingron and A. von Haeseler. TREE-PUZZLE: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18:502–504, 2002. 35

[146] C. Scornavacca, V. Berry, V. Lefort, E. J. P. Douzery and V. Ranwez. PhySIC_IST: cleaning source trees to infer more informative supertrees. *BMC Bioinformatics*, 9: 413, 2008. 20, 33, 55, 88

[147] C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Appl. Math.*, 105(1-3):147–158, 2000. 4, 20, 28, 29, 43, 45, 88

[148] C. Semple and M. A. Steel. *Phylogenetics*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2003. 28

[149] J. B. Slowinski and R. D. M. Page. How should species phylogenies be inferred from sequence data? *Systematic Biology*, 48(4):pp. 814–825, 1999. 14

[150] S. Snir and S. Rao. Quartets MaxCut: a divide and conquer quartets algorithm. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 7(4):704–718, 2010. 34, 35

[151] P. S. Soltis and D. E. Soltis. Molecular systematics: Assembling and using the tree of life. *Taxon*, 50:663–677, 2001. 17

[152] M. S. Springer and W. W. de Jong. Which mammalian supertree to bark up? *Science*, 291(5509):1709–1711, 2001. 14, 15

[153] A. Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006. 13, 53, 87

[154] A. Stamatakis, T. Ludwig and H. Meier. Parallel inference of a 10.000-taxon phylogeny with maximum likelihood. In *Proc. of International Euro-Par Conference (Euro-Par 2004)*, volume 3149 of *Lect. Notes Comput. Sci.*, pages 997–1004. Springer, Berlin, 2004. 14

[155] M. Steel and A. Rodrigo. Maximum likelihood supertrees. *Syst. Biol.*, 57(2):243–250, 2008. 35

[156] M. A. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *J. Classif.*, 9(1):91–116, 1992. 27

[157] M. A. Steel, A. W. Dress and S. Böcker. Simple but fundamental limitations on supertree and consensus tree methods. *Syst. Biol.*, 49(2):363–368, 2000. 22, 29, 45

[158] J. Stoye. Multiple sequence alignment with the divide-and-conquer method. *Gene*, 211(2):GC45–GC56, 1998. 87

[159] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13(7):964–969, 1996. 35

[160] A. R. Subramanian, M. Kaufmann and B. Morgenstern. DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms for molecular biology : AMB*, 3(1):6+, 2008. 87

[161] M. S. Swenson, F. Barbancon, T. Warnow and C. R. Linder. A simulation study comparing supertree and combined analysis methods using smidgen. *Algorithms Mol. Biol.*, 5(1):8, 2010. 49, 50, 53, 54

[162] M. S. Swenson, R. Suri, C. R. Linder and T. Warnow. An experimental study of quartets maxcut and other supertree methods. *Algorithms Mol. Biol.*, 6:7, 2011. 34, 35, 50, 100

[163] D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates, 2002. 12, 13, 54, 87, 88

[164] K. Tamura and M. Nei. Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Molecular Biology and Evolution*, 10(3):512–526, 1993. 13

[165] 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. 87

[166] J. Thorley, M. Wilkinson and M. Charleston. The information content of consensus trees. In *Advances in Data Science and Classification*, Studies in Classification, Data Analysis, and Knowledge Organization. Springer, Berlin, 1998. 33

[167] J. L. Thorley and M. Wilkinson. A view of supertree methods. In M. Jannowitz, F. Lapointe, F. McMorris and F. Roberts, editors, *Bioconsensus*, volume 61. The American Mathematical Society, 2003. 32, 33

[168] L. S. Vinh and A. von Haeseler. Iqpnni: Moving fast through tree space and stopping in time. *Molecular Biology and Evolution*, 21(8):1565–1571, 2004. 13

[169] I. M. Wallace, O. O'Sullivan, D. G. Higgins and C. Notredame. M-coffee: combining multiple sequence alignment methods with t-coffee. *Nucleic Acids Research*, 34(6): 1692–1699, 2006. 87

[170] S. Whelan and N. Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Mol Biol Evol*, 18(5):691–699, 2001. 13

[171] J. J. Wiens. Does adding characters with missing data increase or decrease phylogenetic accuracy? *Syst Biol*, 47(4):625–640, 1998. 16

[172] S. J. Willson. Constructing rooted supertrees using distances. *Bull. Math. Biol.*, 66 (6):1755–1783, 2004. 20, 29, 31, 43, 45, 88, 98

[173] Yang. Maximum-likelihood models for combined analyses of multiple sequence data. *J Mol Evol*, 42(5):587–596, 1996. 16

[174] Z. Yang. Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites: approximate methods. *J. Mol. Evol.*, 39(3):306–314, 1994. 52

Ehrenwörtliche Erklärung

Hiermit erkläre ich

- dass mir die Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angfertigt habe, keine Textabschnitte oder Ergebnisse eines dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönliche Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe.

Ich habe weder die gleiche, noch eine ähnliche oder eine andere Arbeit an einer anderen Hochschule als Dissertation eingereicht.

Jena, den 29.01.2013

Malte Brinkmeyer