Fast and Accurate Supertrees: Towards Large Scale Phylogenies

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.-Bioinf. Markus Fleischauer

geboren am 06. Okt. 1987 **in** Erfurt, Deutschland

Gutachter:

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

Tag der öffentlichen Verteidigung: 29. Juni 2018

Gedruckt auf alterungsbeständigem Papier nach DIN-ISO 9706

Abstract

Phylogenetics is the study of evolutionary relationships between biological entities; phylogenetic trees (phylogenies) are a visualization of these evolutionary relationships. Accurate approaches to reconstruct phylogenies from sequence data usually result in NP-hard optimization problems, hence local search heuristics have to be applied in practice. These methods are highly accurate and fast enough as long as the input data is not too large. Divide-and-conquer techniques are a promising approach to boost scalability and accuracy of those local search heuristics on very large datasets. A divide-and-conquer method breaks down a large phylogenetic problem into smaller sub-problems that are computationally easier to solve. The sub-problems (overlapping trees) are then combined using a supertree method. Supertree methods merge a set of overlapping phylogenetic trees into a supertree containing all taxa of the input trees. The challenge in supertree reconstruction is the way of dealing with conflicting information in the input trees. Many different algorithms for different objective functions have been suggested to resolve these conflicts. In particular, there are methods that encode the source trees in a matrix and the supertree is constructed applying a local search heuristic to optimize the respective objective function. The most widely used supertree methods use such local search heuristics. However, to really improve the scalability of accurate tree reconstruction by divide-and-conquer approaches, accurate polynomial time methods are needed for the supertree reconstruction step.

In this work, we present approaches for accurate polynomial time supertree reconstruction in particular Bad Clade Deletion (BCD), a novel heuristic supertree algorithm with polynomial running time. BCD uses minimum cuts to greedily delete a locally minimal number of columns from a matrix representation to make it compatible. Different from local search heuristics, it guarantees to return the directed perfect phylogeny for the input matrix, corresponding to the parent tree of the input trees if one exists. BCD can take support values of the source trees into account without an increase in complexity. We show how reliable clades can be used to restrict the search space for BCD and how those clades can be collected from the input data using the Greedy Strict Consensus Merger. Finally, we introduce a beam search extension for the BCD algorithm that keeps alive a constant number of partial solutions in each top-down iteration phase. The guaranteed worst-case running time of BCD with beam search extension is still polynomial. We present an exact and a randomized subroutine to generate suboptimal partial solutions.

In our thorough evaluation on several simulated and biological datasets against a representative set of supertree methods we found that BCD is more accurate than the most accurate supertree methods when using support values and search space restriction on simulated data. Simultaneously BCD is faster than any other evaluated method. The beam search approach improved the accuracy of BCD on all evaluated datasets at the cost of speed. We found that BCD supertrees can boost maximum likelihood tree reconstruction when used as starting tree. Further, BCD could handle large scale datasets where local search heuristics did not converge in reasonable time. Due to its combination of speed, accuracy, and the ability to reconstruct the parent tree if one exists, BCD is a promising approach to enable outstanding scalability of divide-and-conquer approaches.

Zusammenfassung

Die Phylogenetik studiert die evolutionären Beziehungen zwischen biologischen Entitäten. Phylogenetische Bäume sind eine Visualisierung dieser Beziehungen. Akkurate Ansätze zur Rekonstruktion von Phylogenien aus Sequenzdaten führen in der Regel zu NP-schweren Optimierungsproblemen, sodass in der Praxis lokale Suchheuristiken angewendet werden müssen. Diese Methoden liefern akkurate Bäume und sind schnell genug, solange die Eingabedaten nicht zu groß werden. Teile-und-herrsche-Verfahren sind ein vielversprechender Ansatz, um Skalierbarkeit und Genauigkeit dieser lokalen Suchheuristiken auf sehr großen Datensätzen zu verbessern. Beim Teile-und-herrsche-Ansatz zerlegt man ein großes phylogenetisches Problem in kleinere Teilprobleme, die einfacher und schneller zu lösen sind. Die Teilprobleme, in diesem Fall überlappende Teilbäume, müssen dann zu einem gesamtheitlichen Baum kombiniert werden. Superbaummethoden verschmelzen solche überlappenden phylogenetischen Bäume zu einem Superbaum, der alle Taxa der Eingangsbäume enthält. Die Herausforderung bei der Superbaumrekonstruktion besteht darin, mit widersprüchlichen Eingabebäumen umzugehen. Es wurden viele verschiedene Algorithmen mit unterschiedlichen Zielfunktionen entwickelt, um solche Widersprüche möglichst sinnvoll aufzulösen. Verfahren, die auf der Kodierung der Eingabebäume als Matrixrepräsentation basieren, sind am weitesten verbreitet. Die zum Auflösen der Konflikte verwendeten Zielfunktionen führen in der Regel zu NP-schweren Optimierungsproblemen, sodass in der Praxis auch hier lokale Suchheuristiken zum Einsatz kommen. Da diese Ansätze nicht wesentlich besser mit der Größe der Eingabedaten skalieren als die direkte Rekonstruktion aus Sequenzdaten, werden für die Superbaumrekonstruktion in Teile-und-herrsche-Ansätzen akkurate Polynomialzeitmethoden benötigt.

Diese Arbeit beschäftigt sich mit der akkuraten Rekonstruktion von Superbäumen in Polynomialzeit. Wir präsentieren Bad Clade Deletion (BCD), eine neue Polynomialzeitheuristik zur Superbaumrekonstruktion. BCD verwendet minimale Schnitte in Graphen, um eine minimale Anzahl von Spalten aus der Matrixrepräsentation zu löschen, sodass diese konfliktfrei wird. Im Gegensatz zu lokalen Suchheuristiken garantiert BCD die Rekonstruktion einer perfekten Phylogenie, sofern eine solche für die Eingabematrix existiert. BCD ermöglicht es, Gütekriterien der Eingabebäume zu berücksichtigen, ohne dass sich dadurch die Komplexität erhöht. Weiterhin zeigen wir, wie zuverlässige Kladen verwendet werden können, um den Suchraum für BCD einzuschränken und wie man diese mit Hilfe des Greedy Strict Consensus Mergers aus den Eingabedaten gewinnen kann. Schließlich stellen wir eine Strahlensuche für BCD vor. Diese erlaubt es eine bestimmte Anzahl suboptimaler Teillösungen (anstatt nur der optimalen) zu berücksichtigen, um so das Gesamtergebnis zu verbessern. Die Worst-Case-Laufzeit der Strahlensuche ist immer noch polynomiell. Zur Berechnung suboptimaler Teillösungen stellen wir einen exakten und einen randomisierten Algorithmus vor.

In einer ausführlichen Evaluation auf mehreren simulierten und biologischen Datensätzen vergleichen wir BCD mit einer repräsentativen Auswahl an Superbaummethoden. Wir haben herausgefunden, dass BCD bei Verwendung von Gütekriterien und Suchraumbeschränkung auf simulierten Daten genauer ist als die akkuratesten evaluierten Superbaummethoden.

Gleichzeitig ist BCD deutlich schneller als alle evaluierten Methoden. Die Strahlensuche verbessert die Qualität der BCD-Bäume auf allen Datensätzen, allerdings auf Kosten der Laufzeit. Weiterhin fanden wir heraus, dass ein BCD-Superbaum, der als Startbaum verwendet wird, die Qualität einer Maximum-Likelihood-Baumrekonstruktion verbessern kann. Außerdem kann BCD Datensätze verarbeiten, die so groß sind, dass lokale Suchheuristiken auf diesen nicht mehr in angemessener Zeit konvergieren. Aufgrund der Kombination aus Geschwindigkeit, Genauigkeit und der Fähigkeit, den Elternbaum zu rekonstruieren, sofern ein solcher existiert, ist BCD ein vielversprechender Ansatz um die Skalierbarkeit von Teile-und-herrsche-Methoden entscheidend zu verbessern.

Acknowledgements

First and foremost, I thank my supervisor Sebastian Böcker. He found a great balance between giving me the freedom and confidence to realize my own ideas and guiding me into promising directions. He always had an open door for questions and discussions, came up with new ideas whenever we needed one and kept me motivated even in lean periods. I thank Arndt von Haeseler for motivational and helpful discussions as at Phylomania or during my visit in Vienna as well as for being my second advisor. I am very grateful to Franziska Hufsky for her advice and knowledge in scientific writing, for proofreading this thesis and for giving many helpful suggestions to improve the readability. Another special thank for proofreading large parts of this thesis goes to Lisa Schmölz. I am grateful to the members of our research group for the amazing working atmosphere; work is always fun with you guys. In particular, I thank my office neighbour Marcus Ludwig for always having a sympathetic ear for my ideas and questions although they were not related to his research field. Furthermore, I thank Thasso Griebel for hiring me as a student research assistant and thereby bringing me to Sebastians group, and for its great work on the EPoS framework which was a great basis for developing algorithms in phylogenetics. A special thank goes to Kathrin Schowtka for her kindness and patience while helping with bureaucracy or taking over administrative work, so that we can mainly focus on science.

I gratefully acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG), within the project “FlipCut supertrees: Computing larger, better phylogenies faster” (BO 1910/12), to undertake my PhD.

I thank my friends in Jena for making me happy, for being my memory, and for their great support during the process of writing. Finally, I thank my mother for always believing in me and supporting me. Also, I thank my ”little” brother for motivating me by always being interested and impressed by the things I have been working on.

Preface

This thesis covers large parts of my research on polynomial time algorithms for phylogenetic supertree reconstruction. During this time, I was working at the Bioinformatics Group of Professor Sebastian Böcker at the Friedrich-Schiller-Universität Jena. My research was financed by the project "FlipCut supertrees: Computing larger, better phylogenies faster" funded by the Deutsche Forschungsgemeinschaft (DFG) and later by the university's basic funding. The results presented in this work have been published and have been achieved in cooperation with my supervisor Sebastian Böcker:

- **M. Fleischauer** and S. Böcker. Collecting reliable clades using the Greedy Strict Consensus Merger. *PeerJ*, 2016.
- **M. Fleischauer** and S. Böcker. Bad Clade Deletion Supertrees. *Molecular Biology and Evolution*, 2017.
- **M. Fleischauer** and S. Böcker. BCD Beam Search: Considering suboptimal partial solutions in Bad Clade Deletion supertrees. *PeerJ*, 2018.

I also participated in the software development of SIRIUS [15] and the CSI:FingerID web service [41]. As student research assistant I participated in the development of EPoS [64] and FlipCut [21]. I wrote my diploma theses on cut enumeration strategies for FlipCut.

This thesis consists of six chapters; the main results are presented in Chapters 4, 5 and 6. Chapter 4 describes the algorithms to maximize the resolution of the Greedy Strict Consensus Merger supertrees by simultaneously preventing unsupported (bogus) clades. Sebastian Böcker and I developed different new scoring criteria for the tree merging order (see Section 4.4) and randomization strategies (see Section 4.6). Algorithm implementation, parallelization and evaluation as well as data simulations (see, SMIDGenOG in Section 3.2.2) was done by me. This work has been presented by me at the German Conference on Bioinformatics (GCB 2015) and is published in [54]. Chapter 5 presents the Bad Clade Deletion (BCD) supertrees algorithm. Sebastian Böcker and I introduced a new objective function that better suits the greedy manner of the FlipCut idea and allows for several algorithm engineering tricks. We developed a search space reduction based on a given set of clades and graph weighting functions to use support values of the source trees. Algorithm implementation, parallelization and evaluation as well as data simulations (see, SMIDGenOG-5500 in Section 3.2.3) was done by me. This work is published in [55]. Chapter 6 introduces a Beam Search algorithm to consider suboptimal partial solutions within the BCD algorithm. Sebastian Böcker and I developed the beam search algorithm. We developed an exact vertex-cut enumeration algorithm and a vertex-cut sampling algorithm. Algorithm implementation and evaluation was done by me. This work is published in [56].

For the remainder of this thesis, I will use "we" as the first person pronoun, as it is common in scientific literature. This may be interpreted as "the reader and I" or as "my collaborators and I", whichever suits best in the situation.

Contents

1 Introduction

Phylogenetics is the study of evolutionary relationships between biological entities; the amount of evolution that took place can be seen as the central criterion to compare those entities. Biological entities can be species, individuals or genes and are referred to as *taxa*. Phylogenetic trees (*phylogenies*) are a visualization of these evolutionary relationships. The leaves of a phylogenetic tree correspond to the extant entities (taxa) whereas the inner nodes are common ancestors.

Where do we come from, what is the "origin of species" [37]; or simply, the search for the "Tree of Life" are the most obvious questions, that the field of phylogenetics tries to answer. CIPRES[1] (Cyber Infrastructure for Phylogenetic Research), ATOL[2] [104] (Assembling the Tree of Life project) and "the Open Tree of Life"[3] [73] are large inter-institutional projects that aim at reconstructing the "Tree of Life". Besides this fundamental question many other applications exist. For example, phylogenetics has been used to determine the origin of HIV-1 [90, 132, 156], for drug discovery [152], to identify disease associated mutations [18], for vaccine strain selection [25, 60, 143], to reveal the global migration of pathogens [118], for protein structure prediction [157] as well as for gene [28] and protein function prediction [44, 93].

A rooted phylogenetic tree is often a sufficient model to describe the evolution of species. However, some evolutionary processes, such as horizontal gene transfer or hybridization (when two species form a new species), do not allow for a treelike representation. In such cases, phylogenetic networks are a better representation [80]. In this thesis, we will concentrate on treelike evolution and its reconstruction. Phylogenies are reconstructed from a set of characters that allow to discriminate the taxa of interest. Initially, such characters were morphological traits whereas nowadays, molecular sequences are the characters of choice. Since the number of trees grows superexponentially with the number of taxa, computational methods are essential even for small problem sizes. To use computational approaches for phylogeny reconstruction, we store the characters in a matrix where each line corresponds to a taxon and each column is a character. We will focus on molecular sequence alignments as character matrices. Accurate approaches to reconstruct phylogenies from sequence data usually result in NP-hard optimization problems, so that heuristics have to be applied in practice. Often, non-deterministic local search heuristics are used to reconstruct accurate trees in a reasonable amount of time under sophisticated statistical models.

In the earlier days of molecular phylogenetics, the availability of molecular data was often a limiting factor. In this context, supertree methods, which combine trees with overlapping taxon set into a large tree containing all of the taxa, became popular. Merging smaller trees together enables a fast and straightforward possibility to combine incomplete data from various sources. With increasing availability of sequence data, combining data on the

[1] `http://www.phylo.org/`
[2] `http://tolweb.org/`
[3] `https://tree.opentreeoflife.org`

sequence level turned out to be the more accurate approach. Since evolution of different loci can be contradictory, combining molecular data introduced the problem of "gene tree species tree reconciliation". In times of 3rd generation sequencing, where (re-)sequencing starts becoming cheaper than storing the raw data, sequence availability is no longer a limiting factor. Nowadays, the tree reconstruction methods struggle processing the large amounts of data available in reasonable time and with sufficient accuracy. While increasing sequence lengths can be handled fairly well by massive parallelization [91, 119], this does not really address the challenge of searching in a tree space increasing exponentially with the number of taxa. In this context, supertree methods become important again as part of divide-and-conquer approaches [116] that cut large datasets into handy bits for accurate but computational intense methods. Here, supertree methods are used to merge the smaller pieces into an overall solution. Whereas the decomposition into pieces can be done in polynomial time, the supertree reconstruction is usually done by local search heuristics that solve an NP-hard problem. Even if this optimization is much faster to compute (Maximum Parsimony) than the one used to estimate a tree from sequence data (Maximum Likelihood), we still search an exponentially growing tree space using a local search heuristic.

In this thesis, we present approaches for polynomial supertree reconstruction with accuracy comparable to the established local search heuristics. This thesis is structured as follows: In Chapter 2 we give the theoretical background for understanding this thesis, namely graph theoretical basics, formal definitions from computational phylogenetics and an overview about the current state of the art and open problems. Chapter 3 describes setup, criteria and data we use to evaluate the presented methods. We evaluate on multiple simulated and biological datasets and compare against a representative set of other methods. Chapter 4 covers our work on improved scoring functions and tree merging strategies for the Greedy Strict Consensus merger. We focused in particular on reducing the amount of unsupported clades, so that these clades can be used for search space restriction without the risk of adding too many unsupported clades to the supertree. We found a scoring function that significantly reduces the amount of unsupported clades by simultaneously improving the number of reliable clades. In Chapter 5 we present a new polynomial time supertree method called, *Bad Clade Deletion* (BCD). The method is a greedy heuristic that minimizes the number of clade deletions and guarantees to return a supertree compatible with all input trees if those are compatible. Our evaluation shows that our new method is faster and can be more accurate than leading supertree methods when using support values from the input trees. Chapter 6 treats a beam search extension for the greedy BCD algorithm that allows to consider a fixed number of suboptimal partial solution. We present an enumeration and a sampling algorithm to compute suboptimal solutions. The beam search improved the accuracy on every dataset at the cost of an increased running time. Finally, Chapter 7 concludes the thesis by recalling the main results, presenting an outlook on further applications and discussing possible strategies for the integration of BCD in divide-and-conquer approaches.

2 Phylogenetic tree reconstructions and their Theoretical Notation

In this chapter, we give a brief overview about the theoretical concepts, techniques and problems that are required to understand this thesis. First, we define all necessary graph theoretical basics (see Section 2.1). Second, we give formal definitions about phylogenetic trees, give a short introduction how to estimate them from molecular data and explain some, in this context, widely used methods (see Section 2.2). Based on this introduction we briefly discuss problems and concepts of large scale phylogeny reconstruction (see Section 2.2.2). Furthermore, we explain all necessary basics about supertree reconstruction (see Section 2.3) and how supertrees can be used in practice (see Sections 2.4 and 2.5). In this context, we will give a more detailed introduction into divide-and-conquer approaches as a promising strategy for large scale phylogeny reconstruction and explain how polynomial time supertree methods can be beneficial in this context (see Section 2.5).

For all introduced fields, we can cover only the most important aspects needed to understand this work. We refer interested readers to relevant textbooks: Nei and Kumar [115] for phylogenetics, Bininda-Emonds [12] for phylogenetic supertrees, Felsenstein [50], Huson *et al.* [80], Semple and Steel [154] and Warnow [181] for methods in phylogenetics.

2.1 Graph Theoretical Definitions

Here, we give graph theoretical definitions needed for a formal definition of phylogenetic concepts and to describe and understand the graph theoretical algorithms presented later on in this thesis. For further information about graph theory, we refer to Diestel [40].

Definition 1 (Graph)**.** An *(undirected) graph* $G = (V, E)$ is the pair of a vertex set V and the edge set $E \subseteq \{e \subseteq V : |e| = 2\}$. Vertices as well as edges can be weighted. The *weight* $\omega(e)$ of an edge $e \in E$ is a function $\omega : E \to \mathbb{R}_{\geq 0}$. Analogously, the weight $\omega_V(v)$ of a vertex $v \in V$ is a function $\omega_V : V \to \mathbb{R}_{\geq 0}$. An edge e is called *incident* to a vertex v if $v \in e$ holds.

Definition 2 (Directed graph – digraph)**.** A *digraph* $D = (V, A)$ is a graph with vertex set V and arc set $A \subseteq V \times V$ of ordered pairs (v, w) with $v, w \in V$. An edge (v, w) is called an outgoing arc of v whereas (w, v) is called an incoming arc of v. For a digraph $D = (V, A)$, the corresponding (undirected) graph $G = (V, E)$ is the graph where for each arc $(u, v) \in A$ there exists an edge $\{u, v\} \in E$.

Definition 3 (Degree)**.** The degree $d(v)$ of a vertex $v \in V$ is the number of edges $e \in E$ with $v \in e$. For digraphs, we differ between outdegree $d^+(v)$ and indegree $d^-(v)$ where $d(v) = d^+(v) + d^-(v)$. The outdegree of a vertex is the number of its outgoing arcs whereas its indegree is the number of incoming arcs.

Definition 4 (Adjacency)**.** Two vertices $u, v \in V$ are *adjacent* if there exists an edge $\{u, v\} \in E$. Analogous to the degree, for digraphs we differ between incoming and outgoing adjacency.

Definition 5 (Trails, paths, cycles and circles)**.** In a graph $G = (V, E)$, a *trail* is an alternating sequence $v_1, e_1, v_2, e_2 \ldots, e_{k-1}, v_k$ of vertices $v_i \in V$ and distinct edges $e_i \in E$, with $e_i = \{v_i, v_{i+1}\}$ for all $0 < i < k$. A *path* is a trail in which all vertices are distinct. A trail is called *cycle* if $v_1 = v_k$. A *circle* is a cycle where $v_1, \ldots, v_{k-1}$ is a path. In a tree T the path that connects the two vertices u and v, is denoted a uTv.

Definition 6 (Chords and chordal (triangulated) graphs)**.** Given a graph $G(V, E)$ and a cycle defined by the vertex set $V' \subseteq V$ and the edge set $E' \subsetneq E$. A *chord* is an edge $\{u, v\} \notin E' : u, v \in V'$. G is called *chordal* (*triangulated*) if all cycles in G with length > 3 have a chord.

Definition 7 (Connectivity)**.** A graph $G(V, E)$ is *connected* if there exists a path from u to v for each $u, v \in V$. For digraphs, we distinguish between strong and weak connectivity. Analogous to undirected graphs, a digraph is *(strongly) connected* if there exists a (directed) path between all u, v. We call a digraph *weakly connected* if its corresponding (undirected) graph is connected.

Definition 8 (Maximal clique)**.** A *clique* in $G = (V, E)$ is a subset $V' \subseteq V$ where all $v \in V'$ are pairwise adjacent. A clique is maximal, if no such $V'' \supseteq V'$ with $|V''| \geq |V'|$ exists.

Definition 9 (Minimal vertex separator and clique separator)**.** Given a connected graph $G = (V, E)$ and the vertices $u, v \in V$. We will refer to any $V' \subsetneq V$ that separates u and v as a $u - v$ *separator*. V' is a *minimal* $u - v$ *separator*, if no $V'' \subsetneq V'$ is a $u - v$ separator. We call V' a *minimal vertex separator* if it is a minimal $u - v$ separator for some pair of vertices u, v. Separators that are cliques are called *clique separators*.

Definition 10 (Cut)**.** Let $V' \subsetneq V$ and $V'' = V \setminus V'$ be the partitions of the vertex set V. A *cut* of a connected graph $G(V, E)$ is a set of edges $C(V', V'') = \{\{u, v\} \in E : u \in V', v \in V''\}$ that disconnects V' from V''. For a digraph $D(V, A)$ the definition is analogue. A cut $C(V', V'')$ is a set of arcs so that V' and V'' are no longer (strongly) connected. Note that in general, $C(V', V'') \neq C(V'', V')$. The cost of a cut is the sum over its edge/arc weights. A *minimum cut* is a cut with minimal costs.

Definition 11 (Vertex-cut)**.** Analogous to a cut, a *vertex-cut* $G(V, E)$ is a set of vertices $W \subsetneq V$ whose deletion disconnects $G(V, E)$.

Definition 12 (Bipartite graphs)**.** A graph $G(V, E)$ is called bipartite if and only if its vertex set V can be separated into two subsets $V' \subsetneq V$ and $V'' = V \setminus V'$ such that each edge contains one vertex V' and one from V''.

Definition 13 (Forests, trees and rooted trees)**.** A graph $G(V, E)$ is a *forest* if it is cycle-free. A forest that is connected is a *tree*. A vertex of a tree $v \in V$ with $d(v) = 1$ is called a *leaf*; all other vertices are *inner vertices*. An edges that is not incident to a leaf is an *inner edges*. A digraph $D(V, A)$ is called *rooted tree* if the corresponding (undirected) graph is a tree that has a given *root(-vertex)* $r \in V$. For each $v \in V \setminus \{r\}$ there has to be a definite directed path $r, \ldots, v$. All $v \in V \setminus \{r\}$ with $d(v) = 1$ are called *leaves*. A rooted tree is binary if and only if $d^+(v) = 2$ holds for all inner vertices.

Definition 14 (Networks and flows)**.** A *network* $H(V, A)$ is a digraph with vertex set V, arc set A, and a given *source* $s \in V$ and *target* $t \in V$. Each arc $e \in A$ has a capacity $c : A \to \mathbb{R}_{\geq 0}$, which is the maximal amount of flow that e can take. A *flow* is a function $f : A \to \mathbb{R}_{\geq 0}$ with $f(u, v) \leq c(u, v)$ that maps a non-negative flow value to each arc $e \in A$. For each $v \in V \setminus \{s, t\}$, the amount of flow entering v has to be equal to the amount of flow exiting it. The *value* of a flow is the amount of flow entering t; a *maximum s-t flow* is a flow from s to t of maximum value.

2.2 Reconstructing Trees from Molecular Sequences

The basis for a phylogenetic tree reconstruction is a set of characters that is able to discriminate between the taxa of interest. The most basic type of characters are binary characters, which determine whether a trait is present or not. The Perfect Phylogeny Problem searches for a tree topology, where the ancestor (inner) nodes can be labeled with character states (*character assignment*), so that each character state evolved only once in the tree [50]. Characters can disagree with each other, so that no such tree exists. Checking whether a set of binary characters is compatible can be solved in polynomial time; but for multi-state characters, this problem is NP-hard [17, 167]. The reconstruction of a phylogenetic tree for a set of taxa from molecular sequences consists of three major steps:

1. Collect comparable data for the taxa of interest. In our case, these data are molecular sequences.
2. Create an alignment of the collected sequences to make them comparable.
3. Estimate trees based on this alignment.

We will not elaborate the process of data collection, but we give a short introduction to the sequence alignment problem before introducing phylogenetic trees and techniques to reconstruct them from sequence alignments. We will assume molecular sequences as our initial source data for tree reconstruction and therefore describe most methods for sequence data as input. However, most principles can also be applied to other kind of character data in particular, binary characters, which we will discuss in the context of supertree reconstruction later on.

(Multiple) sequence alignment. Since insertion and deletion are typical processes of sequence evolution, biological sequences may not have the same length. We have to align these sequences before we are able to compare them against each other. Let $S_1, \ldots S_k : S_i \in \Sigma^*$ be a set of sequences over a given alphabet Σ. A multiple sequence alignment (MSA) of $S_1, \ldots S_k$ is a matrix A of size $k \times N$ with the following properties:

- $A_{i,j} \in \Sigma \cup \{-\}$, where '$-$' is the gap symbol;
- when removing all gap symbols from a row A_i we get the initial sequence S_i;
- no column in A contains only gap symbols.

A single character of a MSA is called *site*. Given a substitution matrix $(\Sigma \cup \{-\} \times \Sigma \cup \{-\})$, the quality of an alignment can either be measured by a distance function $D(A)$ or a similarity function $S(A)$. The optimal alignment is the alignment where $D(A)$ is minimal or $S(A)$ is

maximal. Whereas the multiple alignment problem is NP-hard [45], the pairwise alignment problem ($k = 2$) can be solved in polynomial time by dynamic programming [114]. In practice, the multiple sequence alignment problem is usually solved by progressive heuristics (see e.g. [42, 72, 85, 121, 170, 176]). These methods are complemented by techniques that co-estimate alignment and phylogenetic tree (see e.g. [71, 96–98, 100, 113, 122, 123, 134, 182]. For an overview of the development of MSA methods and their main applications over the past decades, see [30, 43, 86, 158, 177].

Models of sequence evolution. Tree estimation from molecular sequences is usually described as a statistical inference problem where the sequences evolve down a tree via a stochastic process. Several models of sequence evolution (e.g. JC69 [83], GTR [175] or GM [165]) have been developed [4, 190] and allow for phylogeny estimation using statistical methods, such as maximum likelihood or Bayesian phylogenetics. These tree reconstruction methods take advantage of what is known (or hypothesized) about the stochastic process that generated the data. Besides statistical methods, tree reconstruction methods usually do not need a model of sequence evolution but can take advantage of it.

Phylogenetic trees. A *phylogenetic tree* or *phylogeny* is a tree, where each leaf corresponds to a unique taxon. Taxa are the biological entities (e.g. species, genes or RNAs) whose evolutionary relationship is represented by the tree. In principle, the evolutionary process is described best as rooted binary phylogenetic tree and most stochastic models of evolution assume a rooted binary model tree. In contrast, most models of sequence evolution are time-reversible, which means that the root of the tree is unidentifiable. Therefore, most tree reconstruction methods return unrooted trees.

For brevity and because we do not deal with "non phylogenetic trees" at all, we use the terms "phylogenetic tree" and "tree" synonymously later on. Since the novel algorithmic concepts presented in this thesis are for rooted phylogenetic trees, we make all necessary formal definitions for rooted trees. Based on this, we will define some required properties for unrooted trees. Notice that many problems on trees become harder when considering unrooted trees [14, 167, 168]. For readability, vertices of a phylogenetic tree will be called *nodes* whereas we use the term "vertices" for all other graphs.

Rooted phylogenetic trees. Let $\mathcal{V}(T)$ be the node set of a rooted phylogenetic tree. Further, let $\mathcal{L}(T) \subsetneq \mathcal{V}(T)$ be the set of all leaves (nodes of outdegree zero) in T, corresponding to the set of taxa. All nodes $c \in \mathcal{V}(T) \setminus \mathcal{L}(T)$ are inner nodes. Each inner node v induces a clade $C = \mathcal{L}(T^v) \subseteq \mathcal{L}(T)$. Two clades C_1 and C_2 are *compatible* if $C_1 \cap C_2 \in \{C_1, C_2, \emptyset\}$. A set of trees is compatible if all their clades are pairwise compatible. A clade C is *supported* by a tree T if $C' = C \cap \mathcal{L}(T)$ is a clade of T [185]. The *resolution* of a rooted tree is defined as $\frac{|\mathcal{V}(T)|-|\mathcal{L}(T)|}{|\mathcal{L}(T)|-1}$. Hence, a completely unresolved (*star-*)tree has resolution 0, whereas a fully resolved (*binary-*)tree has resolution 1. For a given collection of *input trees* $\mathcal{T} = \{T_1, \ldots, T_k\}$, a supertree T of $\mathcal{T}$ is a phylogenetic tree with leaf set $\mathcal{L}(T) = \bigcup_{T_i \in \mathcal{T}} \mathcal{L}(T_i)$. A supertree T is called a *consensus tree* if $\mathcal{L}(T_i) = \mathcal{L}(T_j)$ for all $T_i, T_j \in \mathcal{T}$. A *strict consensus* of $\mathcal{T}$ is a consensus tree that only contains clades present in every $T_i \in \mathcal{T}$. A *semi-strict consensus* of $\mathcal{T}$ is a consensus tree that only contains clades present in at least one $T_i \in \mathcal{T}$ and is compatible with every $T_i \in \mathcal{T}$. For a set of taxa $X \subsetneq \mathcal{L}(T)$, we define *the X-induced subtree of T*, $T_{|X}$ as the tree obtained by taking the (unique) minimal subgraph $T(X)$ of T that connects the elements of X and then suppressing

all vertices with outdegree one: that is, for each inner vertex v with outdegree one, replace the adjacent edges (p, v) and (v, c) by a single edge (p, c) and delete v. A tree T *refines* T' if T' can be reached from T by contracting inner edges. We say that a supertree T of $T_1, \ldots, T_l$ is a *parent tree* if $T|_{\mathcal{L}(T_i)}$ refines T_i, for all $i = 1, \ldots, l$.

Unrooted phylogenetic trees. Remember that we defined rooted trees as digraphs (see Definition 13). The unrooted version of a rooted tree is its corresponding undirected graph (see Definition 2), where the vertex with degree two (if one exists) is removed and its two adjacent vertices are connected by an edge. Given an unrooted tree T and an inner edge $\{v, w\}$. Let $\{T^v, T^w\}$ be the set of rooted subtrees connected at $\{v, w\}$. We call the induced bipartition of the taxon set $\mathcal{L}(T^v)|\mathcal{L}(T^w)$ a *(non-trivial) split*. Let $S(T)$ be the set of all possible (non-trivial) splits of T. The set of splits of a rooted tree is defined as the set of splits of its unrooted version.

Rooting an unrooted tree. As already mentioned, tree reconstruction based on sequence data can usually not identify the root of the tree; but the correct model for species evolution is a rooted binary tree. In practice, the position of the root can be determined by an *outgroup*. An outgroup is a taxon (or a group of taxa) that is clearly less closely related to the other taxa than all of them between each other. By adding an outgroup to the data, it is clear that the root has to be on the branch that connects the outgroup with the rest of the tree. However, rooting using an outgroup is not trivial. On the one hand side, if the outgroup is too far away from the ingroup taxa, it may fit equally good at each position in the tree and thus produce wrong rootings (see *rogue taxa* [146]). On the other hand side, it is hard to distinguish between taxa that are closely related to the ingroup and taxa that are indeed part of the same group but branched of very early.

2.2.1 Tree Reconstruction Methods

In this section we will introduce some important concepts of tree reconstruction with the focus of sequences as input data. As mention above, sites (characters) do usually not agree with each other, so we have to search for a tree that fits best for the given set of sites (multiple sequence alignment). This can be done by either an optimality criterion, or by pairwise distances calculated from the alignment.

Distance-based tree reconstruction. Distance-based methods estimate trees based on pairwise distances between taxa [22, 27, 36, 39, 52, 61, 144]. These distances are usually computed from sequence alignments. Distance-based methods are fast [39, 95]) but require compromises on topological accuracy.

Optimization-based tree reconstruction. Further, there are methods that search for a tree that explains the given multiple sequence alignment best for some optimality criterion. Finding the optimal tree under such optimality criterion leads often to NP-hard optimization problems so that local search heuristics are common in practice.

Local search heuristics. Heuristics to find trees for NP-hard optimization problems typically use hill-climbing to find local optima and randomization to get out of these local optima. All search heuristics begin with a starting tree obtained by a fast, often distance-based, method. The hill-climbing approach modifies this tree topology using operations, such as nearest-neighbor interchange (NI), subtree pruning and regrafting (SPR), and tree bisection and reconnection (TBR) [50]. If a better tree is found, the procedure restarts with this better tree. If a local optimum is reached, the procedure starts with a new tree usually obtained by a method that employs randomness. This process is repeated until a stopping criterion is reached (e.g. time limit or evidence that it is unlikely to find a better tree). The challenge for these heuristics is that they only ensure to find local optima, which are not necessarily global optima. However, these heuristics work very well for fairly large datasets; but due to the super-exponential growth of the tree space, there will be some point where it becomes very unlikely to find a good local optimum in reasonable time.

Maximum Parsimony. Given a tree T, let each leaf of T be assigned with its corresponding row from the MSA. Maximum Parsimony (MP) searches for an assignment to the inner nodes that needs a minimum number of site state changes along T. Such an assignment can be computed with linear dependence on the input (small parsimony problem) [51, 68, 148]. Finding a tree that allows for the assignment with a minimal number of changes (most parsimonious) is proven to be NP-hard [38]. Therefore, local search heuristics have to be applied in practice instead of the exact branch-and-bound approach.

Maximum Likelihood (ML). Given a model of sequence evolution $\mathcal{M}$ (e.g. JC69 or GTR) and a multiple sequence alignment A, ML estimation searches for a tree T with branch length ω that maximizes the likelihood $L(T) = P(A|T, \omega, \mathcal{M})$ of generating the observed sequence data A. Finding an optimal ML tree is NP-hard [32, 136], so local search heuristics are used in practice. Compared to MP, heuristic searches for ML are more complicated because scoring a tree (optimizing branch length and model parameters) is computationally more intensive than for MP.

Bayesian Phylogenetics. As ML, Bayesian methods calculate likelihoods of trees based on mathematical models of evolution. Instead of seeking the tree with maximum probability to generate the input alignment, Bayesian methods try sample from the set of possible trees with frequency proportional to their likelihood under the observed alignment. These methods are usually implemented using Markov Chain Monte Carlo (MCMC) [62, 69, 191].

FastTree 2. FastTree 2 [128] is a heuristic algorithm based on *minimum-evolution* (it tries to find a topology that minimizes the sum of the branch lengths) and ML. FastTree 2 uses a heuristic variant of neighbor joining to quickly find a starting tree and uses minimum-evolution (NNIs and SPRs) as well as ML (NNIs) to further refine this topology. Due to the restriction of moves (NNIs and SPRs) done for the different optimization criteria (minimum-evolution and ML) FastTree 2 cannot guarantee to reach a local optimum but can guarantee a theoretical running time of $O(n\sqrt{n}\log(n)m\,|\Sigma|)$, where n are the taxa, m is the sequence length and Σ is the alphabet.

Statistical Consistency. A tree reconstruction method is called *statistically consistent* under a model if the method converges to the correct answer for an increasing number of sites. Further, we call a non-consistent method *positively misleading* if it converges to the wrong tree for an increasing number of sites. The distance matrix used by distance-based methods can be computed from a given sequence alignment under a model of sequence evolution (see, *Jukes–Cantor distance correction* for JC69 [83] or *logdet* [165] for GM or GTR). Pairwise distances calculated by these formulas converge to an additive matrix defining the model tree for an increasing number of sites. Methods such as Neighbor Joining [144] or FastMe [39, 95] that guarantee to return the tree defined by an additive matrix are statistically consistent for such matrices. Unfortunately MP and MC are proven to be statistically inconsistent and even positively misleading for several model conditions [48, 88, 194]. If the optimal solution is found, ML is statistically consistent under the JC69, GTR, and GM models [29].

Support Values. ML as well as MP aim to return a single optimal tree (for MP possibly a consensus of multiple optimal trees). Such a tree is not generally sufficient, without any information about its robustness (statistical support). A standard statistical method to tackle this question is (nonparametric) bootstrapping [49]: We sample replicate alignments from the input alignment and calculate trees from this replicate alignments. After a sufficiently large number of bootstrap replicates, we can estimate the support for a split in the original result by counting its occurrence in the replicate trees. Since the classical bootstrapping approach does a complete tree estimation for each replicate, the computation of a sufficient number of bootstrap replicates can be very time intense, heuristics and approximation strategies have been developed [74, 110, 164]. Contrary Bayesian MCMC outputs a distribution of trees, so that the frequency of a split among this tree distribution can be used as support value; but this estimates can be misleading for short branches [192].

2.2.2 Tree Reconstruction in Application

If a statistical model of evolution is available and can be determined for the given data, the probabilistic methods (ML, Bayesian [77, 139]) are at least as accurate than any other technique available. Due to their running time Bayesian methods are usually not applied to large datasets. From the ML methods available, IQ-TREE [119], RAxML [162] and PhyML [66] are arguably the most efficient. If no statistical model is available or the model cannot be identified, MP [63, 75, 174] is usually a good choice. When the data becomes large, methods, such as FastTree 2 [128], IQ-TREE [119] or ExaML [91, 161, 163] can be applied. ExaML is a massively parallel implementation of RAxML that uses GTR+CAT approximation instead of GTR+Gamma to speed up computation. ExaML allows parallelization up to the number of sites in the alignment which allows it to handle very long sequences and also more taxa than RAxML. However, making such a local search heuristic scalable for large datasets comes to the cost of accuracy. For example, the likelihoods of the ExaML/RAxML-light trees are usually worse compared to RAxML trees. FastTree 2 restricts its tree search heuristics so that it can guarantee a polynomial worst case running time and does therefore scale better with the number of taxa but is not designed for long sequences. Although FastTree 2 is not as accurate as methods, such as RAxML, IQ-TREE or PhyML, it is still the most accurate method for taxa numbers where the non deterministic search heuristics are not applicable anymore [128].

Liu *et al.* [99] indicate that tree quality of ML methods decreases on large alignments due to alignment quality. Sievers *et al.* [158] made similar findings about the alignment quality for increasing number of taxa. One approach to lessen the problem of decreasing alignment accuracy is the co-estimation of alignments and trees (e.g. SATé [98, 100]). These methods are usually at least as computational intense as the tree reconstruction methods used for co-estimation.

A promising approach to boost the accuracy and scalability of ML tree estimation and multiple alignment estimation on large scale datasets is divide-and-conquer. Some divide-and-conquer methods do not need to perform either the alignment estimation or the tree reconstruction on the entire taxon set. We discuss divide-and-conquer methods in more detail in Section 2.5.

2.3 Reconstructing Supertrees

Supertree methods assemble phylogenetic trees with non-identical but overlapping taxon sets into one supertree that contains all taxa of the source trees. The initial application of supertrees was the assembly of a large species tree from multiple overlapping species trees that have been reconstructed by different authors from various (e.g. molecular, morphological or even unknown) types of data. In this scenario, the only available information are trees and supertree methods allow us to combine this data. Constructing a rooted supertree from non-conflicting source trees is easy [1], whereas resolving such conflicts in a reasonable way, usually results in NP-hard optimization problems. In this section, we give an overview about the theoretical basics and methodology of supertree reconstruction.

Many supertree approaches have been proposed over the years; some of them may return multiple supertrees which then have to be combined [7, 130], or may return supertrees not containing all taxa [151]. See Bininda-Emonds [11] for early methods, and [6, 9, 21, 31, 33, 35, 76, 107, 109, 120, 133, 142, 151, 155, 159, 166, 173, 178] for recent ones. In the following, we will introduce some basic concepts needed for understanding and describe a selection of supertree methods that are either accurate, widely used, important for understanding or used in later evaluations (see Section 3.4).

Matrix representation of phylogenetic trees. Encoding a set of (input) trees $\mathcal{T}$ in a matrix representation is a fundamental concept in parent- and supertree reconstruction [7, 130, 147, 184]. A Matrix Representation (MR) encodes inner nodes (or branches in the unrooted case) of all source trees as partial binary characters in a matrix, which can then be analyzed using an optimization or agreement criterion to yield the supertree. Here, we give a formal definition for Baum-Ragan [7, 130] encoding of rooted phylogenetic trees:

We transform a set of trees $\mathcal{T}$ into an incomplete binary matrix $M(\mathcal{T})$ with elements in $\{0, 1, ?\}$(see Figure 2.1): Each row of the matrix corresponds to one taxon $1, \ldots, n$, and each clade C (except the root) in each tree is encoded in one column of the matrix. A '1' indicates that the corresponding taxon is part of C, whereas all other taxa of the tree are encoded '0'. The state of taxa that are not part of the tree is unknown, and represented by a question mark ('?'). A binary matrix (no '?') has a perfect phylogeny if all matrix columns are pairwise compatible. Two matrix columns are compatible if the corresponding clades are compatible. For a single tree T, the matrix $M(T) := M(\{T\})$ does not contain '?'-entries. According to the classical directed *perfect phylogeny* model [188], T is a (directed) *perfect phylogeny* of $M(T)$. In the following, "perfect phylogeny" always refers to "directed perfect

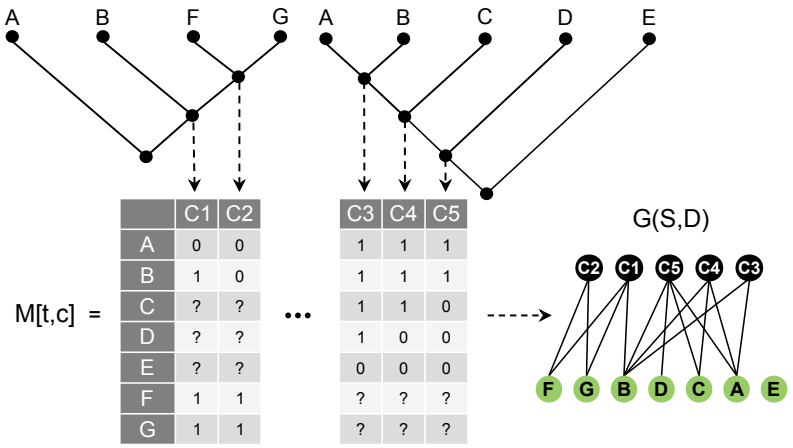


Figure 2.1: Exemplary matrix representation ($M[t, c]$) of two trees and the corresponding Pe'er graph $G(S, D)$, where S (green) are the taxon vertices and D (black) are the clade vertices

phylogeny". An incomplete binary matrix allows for a perfect phylogeny if we can resolve all '?' to either '1' or '0' so that the resulting binary matrix has a perfect phylogeny. For trees T, T' with identical taxa, T refines T' if and only if $M(T')$ can be obtained from $M(T)$ by column deletion [21]. A collection of trees $\mathcal{T}$ has a parent tree if and only if $M(\mathcal{T})$ can be transformed into a perfect phylogeny by resolving each '?' entry to either '0' or '1' [21].

Matrix $M(\mathcal{T})$ has size $n \times m$ where m is the total number of non root inner nodes in $T_1, \ldots, T_l$. The matrix can be computed in $O(mn)$ time, using a tree traversal and lists of taxa.

How to decide whether am MR allows for a perfect phylogeny? Pe'er *et al.* [125] gave an $O(mn \text{ polylog}(m, n))$-time algorithm to prove whether an instance $M(\mathcal{T})$ allows for a perfect phylogeny by resolving all '?'-entries to '1' ore '0'. This algorithm is based on a graph representation of input trees $\mathcal{T}$ or their MR respectively (see Figure 2.1). This *Pe'er graph* is defined as follows: For a subset $S \subseteq \{1, \ldots, n\}$ of taxa and a subset $D \subseteq \{1, \ldots, m\}$ of clades, the Pe'er graph $G(S, D)$ is a bipartite graph with disjoint vertex sets S and D. An edge $\{t, c\}$ is present if and only if $M[t, c] = 1$, for $t \in S$ and $c \in D$. A clade vertex $c \in D$ is *semiuniversal* (in S, D) if $M[t, c] \in \{1, ?\}$ holds for all $t \in S$. Semiuniversal clade vertices do not contain any information about the partition of the taxon set and can be removed from G. The algorithm proceeds in a recursive top-down fashion. In each recursive call, a subset of taxa and a subset of characters are provided; the subset of taxa is returned as a clade of the supertree. If the Pe'er graph is disconnected after deleting all semiuniversal clade vertices, the algorithm directly recurses on the connected components; otherwise, a conflict occurred and the matrix does not allow for a directed perfect phylogeny.

Conservative Methods – Compatibility Supertrees. A *compatibility supertree* contains only clades (or splits in the unrooted case) that do not conflict with *any* of the source trees. We call a supertree method *conservative* if the resulting supertree is a compatibility supertree.

Matrix Representation with Parsimony. *Matrix Representation with Parsimony* (MRP) [7, 130] was among the earliest methods proposed but remains the most frequently used. Despite its computational complexity [59], heuristics have been developed that allow datasets with several hundred taxa to be analyzed in reasonable time. On the theoretical side, MRP has certain undesirable properties, such as introducing clades into the supertree that are contradicted by all source trees [10, 126, 185], and its non-convergence to the true tree for arbitrarily large sets of input trees [166].

Matrix Representation with Flipping. *Matrix Representation with Flipping* (MRF) [31] tries to resolve incompatibilities by flipping '0/1'-entries in the matrix. Finding a tree with a minimum number of flips is again NP-hard but also W[2]-hard, and has no constant factor approximation unless P = NP [16].

FlipCut. Brinkmeyer *et al.* [21] introduced FlipCut, a greedy top-down heuristic for MRF which is based on Pe'er's algorithm for deciding the incomplete directed perfect phylogeny problem. It uses minimum cuts to disconnect the Pe'er graph whenever it is connected due to a conflict in $M(\mathcal{T})$. Each time a conflict is resolved by cutting the graph, it is done with a minimum number of flips in $M(\mathcal{T})$. FlipCut has polynomial running time, and outperforms other polynomial-time supertree methods [124, 151, 153, 187] with regards to supertree accuracy and running time [19, 21].

Matrix Representation with Compatibility. *Matrix representation with Compatibility* (MRC) [129, 137, 142], also known as Maximum Split Fit (SFIT) [34], searches for the largest compatible subset of matrix columns in the MR. The MRC problem is again NP-hard, and it is likely that no PTAS exists [5, 16]. MRC can be reduced to the Maximum Clique problem on the compatibility graph, or to the Maximum Independent Set problem on the incompatibility graph of the matrix. MRC (SFIT) generalizes asymmetric median consensus methods [186].

Bad Clade Deletion. *Bad Clade Deletion* (BCD) is the new supertree algorithm we present in Chapter 5. As FlipCut, the algorithm is a polynomial time top-down heuristic but minimizes the number of column (character) deletions of a MR, so that the resulting matrix allows for a directed perfect phylogeny [125]. *BCD Beam Search* is a beam search extension for BCD to consider suboptimal partial solutions which we present in Chapter 6.

Greedy Strict Consensus Merger. The Strict Consensus Merger (SCM) generalizes the two tree strict consensus problem to a two tree supertree problem [79, 140]. It restricts the two input trees to the subtrees of their common taxa, and calculates a strict consensus tree of these restricted input trees. Afterwards, it re-inserts previously removed taxa into this strict consensus tree. The Greedy Strict Consensus Merger (GSCM) algorithm is the generalization of the SCM for more than two input trees. It combines the set of input trees by greedily applying the SCM algorithm to two of the remaining trees, until only one (super-)tree is left. The GSCM method is sensitive to the order in which the input and intermediate trees are merged. Therefore, the scoring for selecting the tree pairs has high influence on the supertree quality. In Chapter 4, we present new and improved scoring functions as well as improved merging strategies for the GSCM. The GSCM is a conservative supertree method.

SuperFine. Swenson *et al.* [173] introduced the meta-method SuperFine which uses the Greedy Strict Consensus Merger (GSCM) [79, 140] as preprocessing to improve an arbitrary supertree method. SuperFine with MRP was superior to the other evaluated SuperFine variants, namely SuperFine with Quartets MaxCut (QMC) [159] and SuperFine with Matrix Representation with Likelihood (MRL) [120]. To this end, "SuperFine" will refer to the "SuperFine with MRP" method throughout the rest of this paper.

Fast Robinson-Foulds Supertrees. Vachaspati and Warnow [178] introduced *Fast Robinson-Foulds Supertrees* (FastRFS), a polynomial time dynamic programming algorithm that finds the supertree with minimal Robinson-Foulds-distance [135] (see Section 3.1.1) to the input trees under a predefined set of allowed bipartitions X. FastRFS uses ASTRAL-II [111] to compute X from the source trees; but in general any bipartiton of the taxon set can be added to X. Let Z be a set of supertrees we have calculated by arbitrary methods. When adding all bipartitions from the supertrees in Z to the set X, then the supertree returned by FastRFS is as least as good as any supertree in Z with respect to the RF-distance. FastRFS runs in $O(|X|^2nk)$ where n is the number of taxa and k is the number of input trees.

2.4 Supertrees to combine incomplete multi-locus data.

When reconstructing species phylogenies from molecular data, it is common practice to combine multiple loci; as not every locus is available for every taxon, methods have to deal with incomplete data. In the past, the reason for incompleteness of the data was often the unavailability of sequence data. Nowadays, this is not really a problem anymore. Nevertheless, when reconstructing large scale species trees, a locus that is informative for some closely related taxa may not be present in the less related taxa.

Combining incomplete data can be done on different levels [92, 149]: *Low-level* approaches (total evidence, supermatrix, superalignment, combined analysis) combine multiple loci on the sequence level by concatenating the alignments of the different loci; a special character in the resulting data matrix corresponds to missing data. The resulting *supermatrix* can be analyzed by conventional tree reconstruction methods, such as MP, ML or Bayesian MCMC. *Medium-level* strategies combine the loci on a further processed analysis stage than concatenating the raw sequence data but do not estimate a complete tree for each locus. Such stages can e.g. be quartets [150, 169] or distance matrices [35]. *High-level* approaches estimate a phylogenetic tree for each locus independently, and combine those using a supertree method. It turned out that the low-level strategy using a highly accurate statistical method, such as ML, is the most accurate strategy on combined data, see [92, 171] and Chapter 5.

A major challenge when estimating a species tree from multi-locus data is that the evolution of different loci within the genome can be contradictory due to various biological processes [105, 106, 127]. This problem is known as the "gene tree species tree reconciliation problem" [105]. None of the tree reconstruction methods described above are proven to be statistically consistent under these circumstances and classical supertree methods do also perform not well for these type of data. Therefore, novel methods that incorporate coalescent processes have been developed [3, 94, 101–103, 111, 112, 183] to address this problem. These methods are more accurate than the strategies we described earlier in this paragraph when coalescent events happened in the data [112].

2.5 Supertrees in divide-and-conquer approaches

As already mentioned in Section 2.2, statistical tree reconstruction methods (ML, Bayesian MCMC) generally provide sufficient accuracy and speed for fairly large datasets. However, due to the necessary use of heuristic search strategies, the accuracy of these methods decreases with increasing dataset size. Furthermore, alignment accuracy also decreases with increasing dataset size, resulting in limiting input data quality. Running times also strongly increase with increasing dataset size.

In this section, we investigate divide-and-conquer techniques, a promising strategy to evade accuracy decrease and inapplicable running times of computational intensive tree reconstruction methods on very large datasets. The general idea of divide-and-conquer is to break a large problem into smaller subsets that are computationally easier to solve. The sub-problem results have then to be combined into an overall result.

Quartet puzzling. The first approach that used the divide-and-conquer idea for tree reconstruction was quartet puzzling [169]: First, all possible ML quartet trees are generated in the initial step. For each quartet all quartet trees with maximal likelihood score are kept. Second, the heuristic puzzling step combines these quartets into an overall tree. The puzzling step is repeated several times to explore the landscape of optimal trees. Finally, all resulting trees are combined into a majority consensus tree, the quartet puzzling tree.

2.5.1 Disk-Covering Methods

Disk-covering methods (DCMs) are a general divide-and-conquer technique for phylogeny reconstruction that are able to boost tree reconstruction methods when using them as *base methods*. Unlike quartet puzzling, the strategy of disk-covering methods is not decomposing the taxon set into all smallest possible subsets (quartet trees) but rather into handy closely related subsets. The idea is to choose subsets that are easier and faster to solve for the base method. Supertree methods are the key concept to merge the subset trees, generated by the base method, into an overall result.

We mentioned above (see Section 2.4) that classical supertree approaches do not perform well if conflicts in the input trees are caused by evolutionary processes, such as incomplete lineage sorting or horizontal gene transfer. Though, the conflicts between subset trees in a DCM are mainly caused by estimation errors made during the tree estimation with the base method and not by evolution. Therefore, classical supertree methods which search for a supertree that has the smallest error or the highest agreement with the input trees are suitable for this problem.

Different DCMs for a variety of base methods have been developed over time. However, all DCMs follow this basic format (see Figure 2.2): Given the taxon set S and the corresponding sequences, construct a chordal graph G with vertex set S. Calculate an overlapping decomposition of S based on G. Use a supertree method to merge the trees on the decomposed subsets of S into a tree with $\mathcal{L}(T) = S$. The challenge in a DCM is to gain more accuracy by keeping the subsets small enough for a high accurate tree reconstruction than losing by the supertree reconstruction. Therefore, it is important how to design the decomposition (with regards to subset size, subset overlap or subset heterogeneity), to get the best out of the base method and the supertree method. Representing the relations within S by a chordal graph is important to find the decomposition of S in polynomial time. Decomposing G leads us to the maximal clique problem. In chordal graphs there exist at

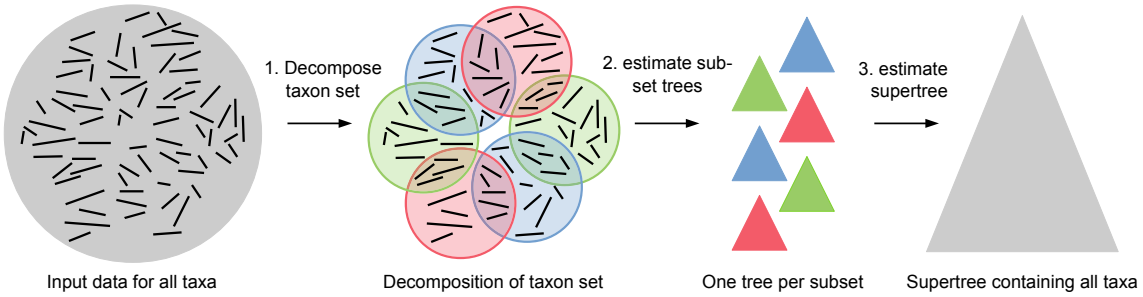


Figure 2.2: Overview about the three basic steps that every DCM follows. First, decompose the input data into overlapping subsets. Second, estimate a tree for each of the subsets. Third, reconstruct a supertree that contains all of the input taxa from the subset trees. Different colors indicate the decomposition. Grey stands for the entire data set.

most $|S| - 1$ maximal cliques which can be found in polynomial time. Furthermore, each minimal separator in a chordal graph is a clique and can also be found in polynomial time.

Two different decomposition strategies have been presented, namely *maximal clique decomposition* and *separator-component decomposition*. A maximal clique decomposition [78] is the set of maximal cliques in G. Let $X \subsetneq S$ be one of the clique separators in G. A separator-component decomposition [79] is the set $\{X \cup B_1, X \cup B_2, \ldots, X \cup B_k\}$, where each B_i is one of k connected components in $G \setminus X$. Since there exist multiple clique separators in G, multiple decompositions are possible with this method. It is possible to use some optimization criterion to choose a single decomposition or, alternatively, all decompositions can be used.

Generally an arbitrary supertree method can be used to combine the trees calculated from the decomposed subsets (*subset trees*). To guarantee to reconstruct the correct overall tree when all subset solutions are correct, it is important to use a supertree method, that guarantees to reconstruct the true tree when all estimated subset trees are induced subtrees of the true tree. We can distinguish between two groups of DCMs, distance-based DCMs [78, 79] and tree-based DCMs [141].

Distance-based DCMs. Several versions of distance-based DCMs have been developed. Here, we explain the general idea. For a given set of taxa $S = \{s_1, s_2, \ldots, s_n\}$, distance-based DCMs create the chordal graph using a distance matrix $d_{i,j}$ on S and a positive threshold q. The *matrix threshold graph* $TG(d, q)$ has vertex set S and an edge (s_i, s_j) for all i, j with $d_{i,j} \leq q$. This graph is proven to be chordal when d is an additive matrix [78]. Remember, d converges to an additive matrix defining the model tree when created from an appropriately corrected model of sequence evolution (see Section 2.2.1). However, in practice, the sequences length may not be sufficient for d to be an additive matrix. Therefore, in practice, heuristics have to be applied to turn $TG(d, q)$ into a chordal graph. We create a matrix threshold graph for each q contained in d. For each of the connected matrix threshold graphs, we create a decomposition using either maximal clique decomposition or separator-component decomposition, see [78, 79] for details.

Tree-based DCMs. Tree-based DCMs use a tree T with $\mathcal{L}(T) = S$ to decompose the taxon set S [141]. The initial tree can be obtained by some fast, not necessarily accurate, method. Since we can construct an additive matrix d from each T with positive branch

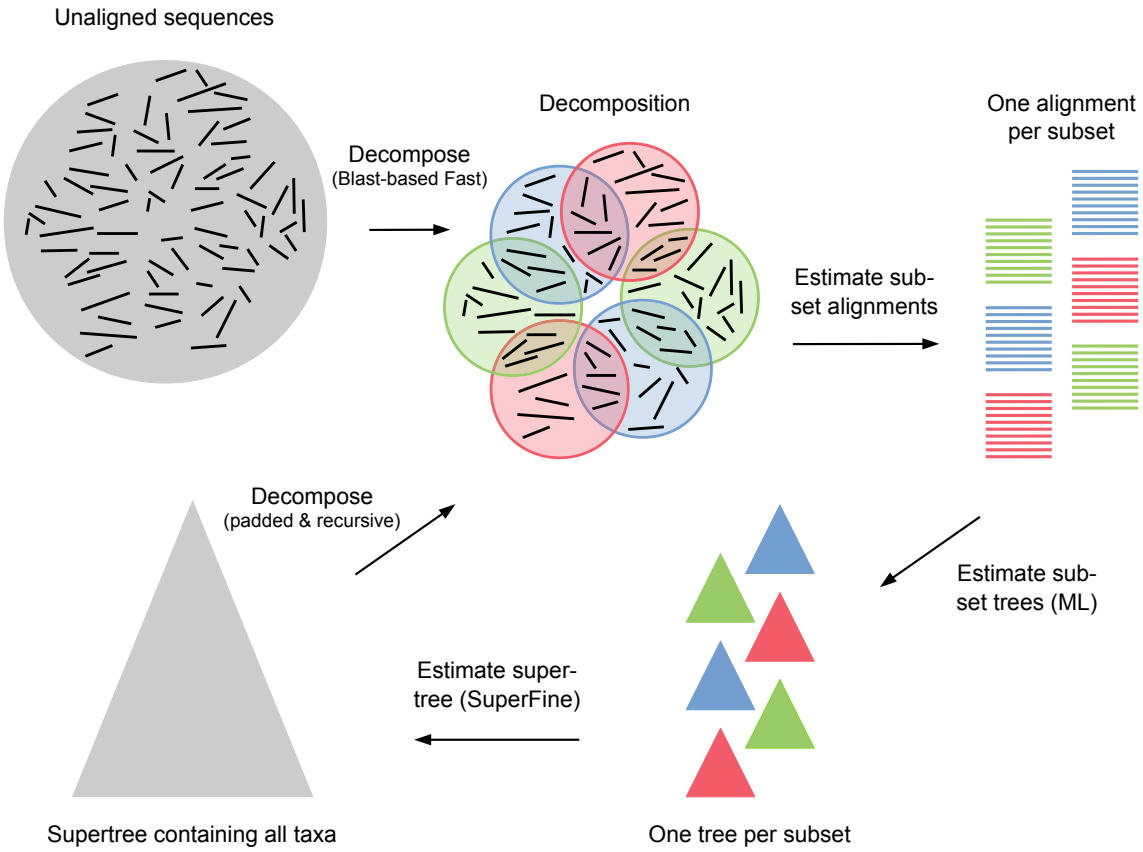


Figure 2.3: Basic workflow of DACTAL starting with a set of unaligned sequences (upper left). In parenthesis we show the method that is usually used for the respective step. Different colors indicate the decomposition. Grey stands for the entire data set.

length, we can also compute a chordal graph (the *short subtree graph*) from T. The short subtree graph is defined as follows: For a branch $e = \{u, v\}$ of a binary tree T, each t_i is one of the four subtrees rooted at r_i for which there exists a branch $\{w, r_i\} : w \in e$, with $1 \leq i \leq 4$. Let $X_i \subseteq \mathcal{L}(t_i)$ denote those leaves that are closest to e (defined by the path length of $r_i T w$). The short subtree graph $SSG(T)$ has vertex set S and edges so that all $X(e) = \bigcup_i X_i$ form a clique in $SSG(T)$, for each inner branch e in T. Since $SSG(T)$ is a chordal graph we can now decompose S using the maximal clique decomposition or the separator-component decomposition.

2.5.2 DACTAL

Nelesen *et al.* [116] presented DACTAL (divide-and-conquer trees almost without alignments), an iterative recursive padded DCM that allows to reconstruct ML trees without needing to estimate an alignment for the entire taxon set. An initial, not necessarily accurate, tree is needed to do the first decomposition. DACTAL introduces *BLAST-based Fast* [117], a blast-based method to produce an initial decomposition without computing a full multiple sequence alignment. However, any method could be used to estimate an initial tree for the first decomposition, see Figure 2.3 for an overview.

BLAST-based Fast. In the following, we give a high level description of BLAST-based Fast and refer to Nelesen [117] for details. The method has two parameters, namely the overlap o and the maximal subset size s. First, a *blastdb* of all input sequences is created. Then an arbitrary sequence is chosen and "blasted" against the whole blastdb. The best s hits form a subset. Then the first hit on position at least s which is not already in a set, is used as seed for the next blast search. This procedure is repeated until all sequences are part of a set. The resulting decomposition may has insufficient overlap for a supertree reconstruction and needs to be post processed. Given the set S_i, let S_j be the set with highest overlap among all created sets. For each S_i with $|S_i \cap S_j| < o$, we choose some sequence $s \in S_i \cap S_j$ as seed for another blast search. We now iterate through the hit list and add a sequence s' to S_i if and only if $s' \in S_j$ until $|S_i \cap S_j| = o$. The resulting sets are the initial decomposition for the subset tree reconstruction step.

DACTAL decomposition. DACTAL uses a recursive padded separator-component decomposition, where the padding factor p and the maximal subset size s can be specified by the user. The separator-component $X(e)$ is chosen using the centroid edge method described in [141]. In a given tree T, the centroid edge e is the edge that induces the most balanced split. When $X(e)$ fails to be a separator in the (padded) short subtree graph of T, the maximal clique decomposition is used; though, Roshan *et al.* [141] never observed this in practice. However, another strategy would be to simply do a breath first search among the edges around e until one of these edges induces a valid separator.

Merging the subset trees. DACTAL uses SuperFine, which uses MRP to refine the Strict Consensus Merger tree, to merge the subset trees. Let $\mathcal{L}(t_1), \mathcal{L}(t_2), \ldots, \mathcal{L}(t_k)$ be the leaf sets of the subset trees estimated on a DACTAL decomposition of T. Using the SCM (and therefore also SuperFine) guarantees that the correct tree T is returned if each of the subset trees $t_i = T|_{\mathcal{L}(t_i)}$ [116].

Performance. DACTAL produces at least as accurate trees than ML analyses and even more accurate trees when analyzing very large "difficult-to-align" datasets. DACTAL showed to be as accurate as SATé [98, 100] but much faster. DACTAL was able to analyze larger datasets than SATé, including one dataset with about 28.000 rRNA sequences [116].

Limitations and possible improvements. As mentioned above, DACTAL uses SuperFine to reconstruct the supertree from the subset trees. This is a good choice in the way that it inherits the guarantee to return the true tree if all subset trees are correct trees from the SCM. Usually the subset trees lead to reasonable well resolved SCM trees. The resolution of the SCM may further improve during iteration. For a SCM tree with reasonable maximum degree, SuperFine can resolve the polytomies in reasonable time using MRP. However, the initial decomposition estimated by BLAST-based Fast tend to produce SCM trees with high degrees [117]. For example, the SCM tree calculated from the initial Blast-based Fast decomposition on the 16S.B.ALL (27643 taxa) dataset contained a polytomy of size 15533, so that MRP failed to refine this supertree [117]. In this case a starting tree, computed on a full alignment, had to be used to handle the dataset. This may be a limiting factor for data with even more taxa and longer sequences. Further, there is no guarantee that the SCM tree will not have high degrees regardless of which method is used for decomposition. Another point for scalability is that we need to keep the subset size on a level that the base

method can handle efficiently. For an increasing number of taxa this results in an increase of the number of subsets. For more input trees, it becomes more likely that the SCM tree is unresolved. A polynomial time supertree method that can handle thousands of taxa and is at least as accurate as MRP would guarantee that DACTAL scales well for even larger datasets.

The new approaches presented in this thesis focus on developing a polynomial time solution to combine the subset trees in DACTAL-like divide-and-conquer approaches even if the SCM tree may be (almost) unresolved.

3 Evaluation of Phylogenetic Tree Reconstruction Methods

This chapter describes the setup (datasets, methods and metrics) we use for a thorough evaluation of our new algorithms. We compare them under multiple criteria on different simulated and biological datasets against commonly used (low-level and high-level) tree reconstruction approaches. Since most of these methods produce unrooted trees, we ignore the roots for all quality measurements, comparing the induced splits instead of the induced clades.

3.1 Evaluation Criteria

3.1.1 Criteria for Simulated Data

For simulated data, we can compare estimated trees to the actual model tree. Unless indicated otherwise, we assume the model tree to be fully resolved. Let $P = \mathcal{L}(T_m) - 3$ be the number of (positive/correct) splits induced by the fully resolved model tree T_m. We call a split false negative (FN) if it is not in the estimated tree but in the model tree; and false positive (FP) if it is in the estimated tree but not in the model tree. Splits present in both, the model tree and the estimated tree are true positives (TP). False negative ($\frac{FN}{P}$) and false positive rates ($\frac{FP}{P}$) are common, tree size-independent metrics to measure the quality of estimated trees. Comparing both, FN and FP rates, provides information about the resolution of estimated trees. For a fully resolved tree, FN equals FP. Making an absolute decision based on multiple quality criteria can be difficult, as there are different possibilities to combine the split-based difference between two trees into a single metric. Which suit best, depends on the trees that are compared.

Robinson-Foulds distance. A common way to compare trees by one single criterion is the Robinson-Foulds distance (RF-distance) [135]. When comparing an estimated tree against a fully resolved model tree, the RF-distance corresponds to $FN + FP$. To be comparable among instances with different numbers of taxa, the Robinson-Foulds distance needs to be normalized to $\frac{FN+FP}{2P}$. The RF-distance overestimates the quality of unresolved trees, e.g. the empty (star) tree has the same normalized RF-distance (0.5) as a fully resolved tree with 50% correct splits. Thus, the RF-distance is inappropriate when comparing trees with varying resolutions.

F_1-score. Hence, we will prefer the well-known F_1-score (harmonic mean of *precision* and *recall*) as a single criterion to evaluate tree quality,

$$F_1 = \frac{2}{1/\text{precision} + 1/\text{recall}} = \frac{2\,TP}{2\,TP + FP + FN}.$$

To visualize results for multiple data replicates, we use boxplots. The data replicates are independent and may have highly varying complexity; even for highly overlapping boxes, it is possible that one method constantly outperforms another one for every data replicate. Therefore, we additionally report the absolute number of how often one method outperforms another for a given quality criterion (e.g. F_1-score).

3.1.2 Criteria for Biological Data

For biological data, the "true tree" to compare against is unknown; to this end, we resort to other evaluation criteria: For *supermatrix datasets*, we evaluate the estimated trees against the sequence data (supermatrix), using both the *parsimony score* and the *log-likelihood score*. We use RAxML to optimize branch length and calculate the log-likelihood. For non-binary trees, we resolve all polytomies randomly; this may discriminate against methods that return highly unresolved trees.

The *supertree datasets* do not even contain sequence data to compare against. To this end, we compare the estimated supertrees against the source trees. We measure the *sum of false negatives* (*SFN*) and *sum of false positives* (*SFP*) rate of a supertree T' compared to the set source trees $\mathcal{T}$,

$$SFN \text{ rate} = \frac{\sum_{T\in\mathcal{T}} \left|\mathcal{S}(T) \setminus \mathcal{S}(T'|_{\mathcal{L}(T)})\right|}{\sum_{T\in\mathcal{T}} |\mathcal{S}(T)|} \quad \text{and} \quad SFP \text{ rate} = \frac{\sum_{T\in\mathcal{T}} \left|\mathcal{S}(T'|_{\mathcal{L}(T)}) \setminus \mathcal{S}(T)\right|}{\sum_{T\in\mathcal{T}} |\mathcal{S}(T')|},$$

where $T'|_{\mathcal{L}(T)}$ is the subtree of T' induced by the taxon set of tree T and $\mathcal{S}(T)$ is the set of splits induced by tree T. Note that optimal values of *SFN* rate and *SFP* rate depend on the source trees: For conflicting source trees, it is not possible to find a supertree with both *SFN* rate = 0 and *SFP* rate = 0. Hence, *SFN* rate and *SFP* rate cannot be compared among different datasets.

3.2 Simulated Datasets

Simulated datasets allow us to evaluate the quality of the computed supertrees in an absolute sense, by comparison to the model tree.

3.2.1 SMIDGen

We use the *SMIDGen*[1] dataset as described by Swenson *et al.* [171]. The underlying SMIDGen-protocol follows data collection processes used by systematists when gathering empirical data, e.g. the creation of several densely-sampled *clade-based source trees*, and a sparsely-sampled *scaffold source tree*. The dataset contains 30 model trees with 500 taxa and 10 model trees with 1000 taxa. For each model tree, 15 clade-based source trees (for 500 taxa) or 25 clade-based source trees (for 1000 taxa) were generated. In addition, four scaffold source trees containing 20 %, 50 %, 75 %, or 100 % of the taxa in the model tree (the *scaffold density*) were generated. All source trees are unrooted. As the bootstrap weighting schemes for BCD supertrees requires bootstrap values, we performed bootstrap analyses for the source trees using RAxML. The result trees of the combined analysis (CA) were taken from Swenson *et al.* [171].

[1] `https://sites.google.com/eng.ucsd.edu/datasets/dactalsuperfine`

Ten instances from *SMIDGen* were excluded from the datasets as there is a bipartition S, S' of the taxa set such that each source tree of the instance contains taxa from either S or S', but no source tree exists that spans both S and S'. For such instances the supertree problem is underdetermined, as we can arbitrarily combine a supertree for the source trees on S with a supertree for the source trees on S'.

Rooting unrooted source trees. While MRP uses unrooted trees as input, for BCD the source trees have to be rooted. Unfortunately, the source trees of the SMIDGen and some biological datasets (see Table 3.3) are unrooted, and rooting trees is not straightforward. It is obvious that the position of the roots in the source trees influences the BCD supertree, and badly rooted source trees decrease the quality of the resulting supertree. A simple method for this purpose is "midpoint rooting" that searches for the longest distance between any two taxa in the tree, then roots the tree at the midpoint between these two taxa. However, this method is generally considered error-prone. The arguably best method to root phylogenetic trees is to use an *outgroup* which is considered during computation of the multiple sequence alignments for each source tree.

Computing the GSCM supertree during preprocessing allows for a more powerful alternative: Since the source trees are unrooted, the root of the supertree is of no interest, either. To this end, we do not have to ensure that all source trees are correctly rooted but rather that the roots we insert into the source trees are *consistent* and do not conflict with each other. To this end, we first root the GSCM supertree by, say, midpoint rooting. Next, we use the rooted GSCM supertree to find compatible rootings for the source trees: We root each unrooted source tree T such that it has minimum conflict with the GSCM tree. For that, we consider the induced subtree of the GSCM supertree, restricted to the leaf set $\mathcal{L}(T)$. We consider the split $A|B$ that corresponds to removing the root node from the induced tree. Next, we search for an edge in T such that the corresponding split $C|D$ has minimum conflict $\min\{|A \cap C|, |B \cap D|\}$ with this root split, and root T at this edge. After transitionally rooting all input trees, we compute the BCD supertree for this set of rooted trees. Finally, we remove the root from the BCD supertree and output its unrooted counterpart. This method is just a heuristic to minimize the error we add to the source trees by rooting them.

For a fair comparison between rooted an unrooted supertree methods, data with rooted input trees should be used and the resulting supertrees should be evaluated without considering the roots as described above.

3.2.2 SMIDGen Outgroup

We simulated the rooted SMIDGen Outgroup (SMIDGenOG) dataset following the SMIDGen-protocol [171]. Each replicate consists of multiple *clade-based source trees* using a densely-sampled subset of taxa from one clade of the model tree, plus a single *scaffold source tree* which uses a sparsely-sampled subset of the taxa from the entire model tree. We generate 30 model trees with 1000 (500/100) taxa. For each model tree, we generate a set of 30 (15/5) clade-based source trees and four scaffold source trees containing 20 %, 50 %, 75 %, or 100 % of the taxa in the model tree (the *scaffold density*). We set up four different source tree sets: each of them containing all clade-based trees and one of the scaffold trees, respectively. Unless indicated otherwise, we strictly follow the protocol of Swenson *et al.* [171] (see also Figure 3.1 for details):

1. **Generate model trees.** We generate model trees using r8s [145] as described by Swenson *et al.* [171]. To each model tree, we add an outgroup. The branch to the outgroup gets the length of the longest path in the tree, plus a random value between 0 and 1. This outgroup placement guarantees that there exists an outgroup for every possible subtree of the model tree.

2. **Generate sequences.** Universal genes appear at the root of the model tree and do not go extinct. We simulate five universal genes along the model tree. Universal genes are used to infer scaffold trees. To simulate non-universal genes, we use a gene "birth-death" process (as described by Swenson *et al.* [171]) to determine 200 subtrees (one for each gene) within the model tree for which a gene will be simulated. For comparison, the *SMIDGen* dataset evolves 100 non-universal genes. Simulating a higher number of genes increases the probability to find a valuable outgroup. Genes (both universal and non-universal) are simulated under a GTR+Gamma+Invariable Sites process along the respective tree, using Seq-Gen [131].

3. **Generate source alignments.** To generate a clade-based source alignment, we select a clade of interest from the model tree using a "birth" node selection process (as described by Swenson *et al.* [171]). For each clade of interest, we select the three non-universal gene sequences with the highest taxa coverage to build the alignment. For each source alignment, we search in the model tree for an outgroup where all three non-universal genes are present and add it to the alignment.

   To generate a scaffold source alignment, we randomly select a subset of taxa from the model tree with a fixed probability (scaffold factor) and use the universal gene sequences.

4. **Estimation of source trees.** We estimate Maximum Likelihood (ML) source trees using `RAxML` with GTR-GAMMA default settings and 100 bootstrap replicates. We root all source trees using the outgroup, and remove the outgroups afterwards.

The resulting source tree sets contain between 8% and 15% contradicting clades compared to the model tree. A single source tree contains between 0% and 35% contradicting clades (see Table 3.1 for details).

3.2.3 SMIDGen Outgroup 5500

To show the practical benefit of a polynomial time method, we needed a dataset of reasonable size to lead the local search heuristics to a point where they would not converge in reasonable time. We created the large and also challenging SMIDGen Outgroup 5500 (SMIDGenOG-5500) dataset containing an average number of 5500 taxa and between 37104 and 62495 characters for each of the 10 replicates. We still follow the idea of several densely-sampled *clade-based source trees*, and sparsely-sampled *scaffold source trees* as described by Swenson *et al.* [171]. Since the reconstruction time of the source trees increases exponentially with the number of taxa, we created 505 (500 clade-based and 5 scaffold) source trees with a size between 75 and 125 taxa for each replicate. This results in source trees of roughly the same size, so that the data look more like created during a divide-and-conquer approach.

1. **Generate model trees.** We generated 10000 taxa model trees as described in Section 3.2.2 for the SMIDGenOG dataset.

Table 3.1: Summary statistics over all replicates for all SMIDGenOG datasets. Since all source trees and model trees are fully resolved, the number of false negative clades is always equal to the number of false positive clades. We report the amount of contradicting clades of clade-based and scaffold source trees compared to the model tree. Conflicts within the source trees is the amount of clades in the source trees that conflict with at least one clade in the source tree set.

			100 Taxa	500 Taxa	1000 Taxa	5500 Taxa
Dataset	#Source trees		6	16	31	505
	#Scaffold trees		1	1	1	5
	#Taxa	Maximum	101	501	1001	7245
		Mean	94.36	457.79	906.96	5376.6
		Median	79.5	407	843	5046.5
		Minimum	72	326	691	3910
Per clade-based source tree	Conflicts to model tree (%)	Maximum	80	309	568	124
		Mean	35.3	65.12	72.69	94.56
		Median	35.5	102.5	46.5	79.5
		Minimum	15	17	18	74
Per scaffold source tree		Maximum	50	38.1	35	35.05
		Mean	8.4	9.06	9.09	5.95
		Median	10.44	10.42	8.76	5.91
		Minimum	0	0	0	0
Per source tree set		Maximum	36.36	24.72	22.91	5.43
		Mean	9.13	10.92	11.39	3.44
		Median	15.89	7.77	12.21	3.58
		Minimum	0	4.31	6.32	1.27
	Conflicts within source trees (%)	Maximum	23.13	21.87	22.17	14.46
		Mean	11.9	15.05	16.55	10.78
		Median	15.4	12.67	16.84	9.98
		Minimum	2.14	7.12	10.52	7.91

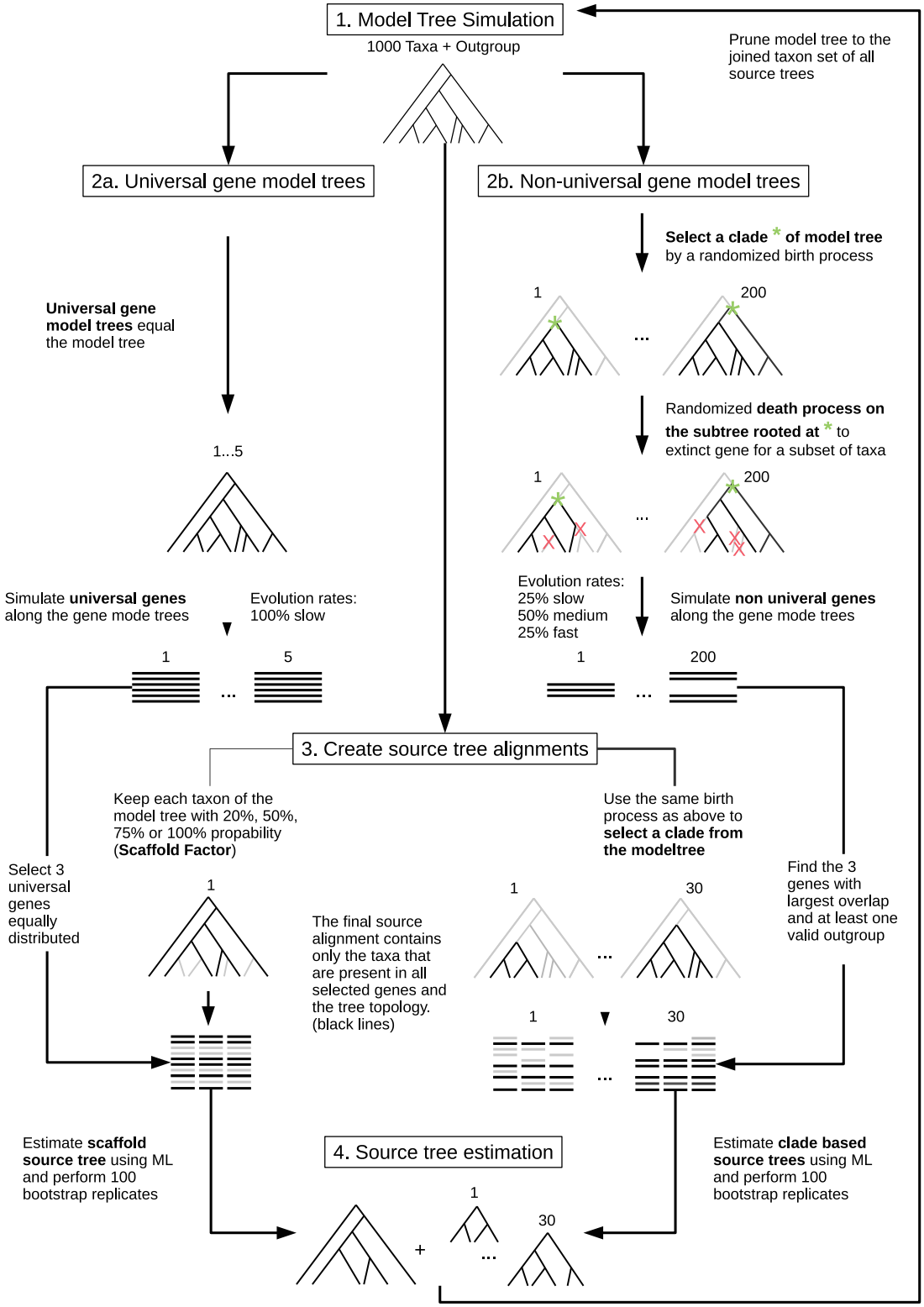


Figure 3.1: Exemplary work flow of data simulation for one replicate of the SMIDGenOG dataset with 1000 taxa. Grey lines in trees and alignments indicate deleted data.

2. **Generate sequences.** Universal genes appear at the root of the model tree and do not go extinct. We simulated 45 universal genes along the model tree. Universal genes are used to infer scaffold trees. To simulate non-universal genes, we used the gene "birth-death" process, as described by Swenson *et al.* [171], and determined 1000 subtrees (one for each gene) of the model tree. Gene sequences (both universal and non-universal) are simulated under a GTR+Gamma+Invariable sites process along the respective tree, using Seq-Gen [131] and contain 5000 nucleotides.

3. **Generate source alignments.** Each clade-based source alignment was generated by concatenating three randomly selected non-universal gene sequences and shrinking them to their common taxon set. We repeated this process until we found 500 different source alignments with 75 to 125 taxa each. We reapplied the process if the resulting 500 source alignments did not had sufficient taxon overlap.

   To generate the scaffold source alignments, we concatenated three randomly selected universal gene sequences and shrank them to 100 randomly picked taxa.

4. **Estimation of source trees.** Source trees are estimated as described for the SMIDGenOG dataset (see Section 3.2.2). For time reasons, we performed only 25 bootstrap replicates.

3.2.4 SuperTriplets Benchmark

The creation protocol of this dataset is very different from the SMIDGen protocol, and contains neither branch lengths nor bootstrap values.

Ranwez *et al.* [133] generated an ultrametric model tree with 100 ingroup taxa using r8s [145]. One outgroup taxon was a posteriori added resulting in a 101-taxon model tree. They created 50 rooted 101-taxon trees with the same topology as the model tree but with different branch lengths and thus different global evolutionary rates. For each of these trees, they generated nucleotide alignments of 101 taxa using Seq-Gen [131] under a Kimura [89] model with transition to transversion ratio of 2.0. The number of aligned sites was uniformly drawn from 200 to 1000 bp. For each alignment, ingroup taxa were randomly deleted with a probability d = 25, 50 and 75% following the procedure firstly introduced by Eulenstein *et al.* [47]. For each of these alignments, a Maximum Likelihood tree was created using PhyML [65] under the Kimura [89] model. Each source tree returned by PhyML was then rooted using the outgroup. This procedure was repeated 100 times to get 100 data replicates for every deletion rate (25%, 50% and 75%). After all, they generated subsets of (10, 20, 30, 40 and 50 trees) from each source tree set that still contain the full taxon set. For summary statistics of this dataset see Table 3.2

Table 3.2: Summary statistics of the SuperTriplets Benchmark [133]. All source trees and model trees are fully resolved, thus the number of false negative clades is always equal to the number of false positive clades. We report the amount of contradicting clades compared to the model per source tree. Conflicts within the source trees is the amount of clades in the source trees that conflict with at least one clade in the source tree set.

			101 Taxa					
Dataset	Deletion Rate (%)		25	50	75	25	50	75
	#Source trees		10	10	10	50	50	50
Per source tree	#Taxa	Max.	101	101	101	101	101	101
		Mean	78.22	55.3	35.06	76.55	51.83	29.63
		Median	93.5	75	63.5	86	75.5	62.5
		Min.	61	38	20	61	34	20
	Conflicts to model tree (%)	Max.	79.71	80	81.48	79.71	80	84.21
		Mean	18.17	19.99	23.46	18.36	20.17	23.76
		Median	18.22	6.02	22.16	20.54	22.96	21.43
		Min.	0	0	0	0	0	0
Per source tree set	Conflicts within source trees (%)	Max.	74.77	66.67	44.7	89.43	79.55	69.21
		Mean	48.04	40.84	28.99	71.15	61.92	48.61
		Median	40.59	48.33	38.63	70.83	65.33	54.74
		Min.	22.05	15.9	8.84	49.25	42.74	36.89

3.3 Biological Datasets

Simulated data tend to have the disadvantage that the signal is "too strong", and almost any method returns high quality results. Therefore, we have to ensure that findings made for simulated data are also supported by evaluations on biological data. We use data from three large-scale supermatrix studies on *bees* (1376 taxa, 19 source trees, see Hedtke *et al.* [70]), *saxifragales* (950 taxa, 51 source trees, see Soltis *et al.* [160]), and *legumes* (2228 taxa, 38 source trees, see McMahon and Sanderson [108]). To generate source trees for the supermatrix datasets, we split the combined alignment into its components. For every resulting alignment with more than three taxa, we calculated an ML source tree with bootstrap values, using RAxML with GTR-GAMMA default settings and 100 bootstrap replicates. The combined analysis trees of the bees [70] and saxifragales [160] datasets are Maximum Likelihood trees (CA-ML). The legumes combined analysis tree is a Maximum Parsimony tree (CA-MP) [108]. Furthermore, we evaluate methods on seven supertree datasets, namely *placental mammals* (116 taxa, 726 source trees, see Beck *et al.* [8]), *marsupials* (267 taxa, 158 source trees, see Cardillo *et al.* [26]), *seabirds* (121 taxa, 7 source trees, see Kennedy and Page [87]), *temperate herbaceous tribes* (THPL, 586 taxa, 22 source trees, see Wojciechowski *et al.* [189]), *primates* (85 taxa, 46 source trees, see Purvis [129]), a *mammalian phylogenomics case study* (OMM, 33 taxa,12,958 source trees, see Ranwez *et al.* [133]) and a supertree of the *bats* (916 taxa, 16 source trees, see Jones *et al.* [82]). Most of these datasets were previously used to evaluate supertree methods; see Table 3.3 for additional information about the datasets.

Table 3.3: Overview of the biological datasets used in our evaluation. Here, '#' is the number of input trees. We use two different types of studies, namely Supermatrix (SM) and Supertree (ST) studies. BS (bootstrap values), BL (branch length) and R (roots) indicate whether the input trees contain the respective feature. "Conflicts" amount of clades in the source trees that conflict with at least one clade in the source tree set.

			Input Trees								
Name	Type	#Taxa	#	Min	Max	Mean	Med.	Conflicts	BL	BS	R
Bees [70]	SM	1376	19	4	1273	264.68	24	77.33%	✓	✓	✗
Saxafragales [160]	SM	950	51	10	419	53.76	30	70.61%	✓	✓	✗
Legumes [108]	SM	2228	38	4	1648	102.74	13.5	52.32%	✓	✓	✗
Marsupials [26]	ST	267	158	3	267	16.39	11.5	62.42%	✗	✗	✓
Mammals [8]	ST	116	726	3	116	12.8	7	85.58%	✗	✗	✓
Seabirds [87]	ST	129	7	15	90	32.14	22	23.71%	✗	✗	✓
THPL [189]	ST	587	20	10	140	45.9	41.5	29.46%	✗	✗	✗
Primates [129]	ST	85	48	9	67	21.62	19.5	48.82%	✗	✓	✓
OMM [133]	ST	33	12958	6	33	27.8	29	96.25%	✓	✗	✓
Bats [82]	ST	936	16	5	209	58.5	52.5	0.00%	✗	✗	✓

3.4 Evaluated Methods

We apply the evaluation setup described above to compare the performance of our new developed methods against a representative set of established tree reconstruction methods, namely Matrix Representation with Parsimony (MRP), SuperFine(+MRP), FastRFS, and a combined analysis approach using either ML (CA-ML) or MP (CA-MP), see Table 3.4. For MRP, we use the majority consensus in those cases where more than one most parsimonious tree is found, as this variant performed better than the strict consensus in our evaluations. BCD Beam Search also returns multiple trees; in such cases, we return the tree with the best BCD score.

Previous evaluations clearly suggest that other supertree methods [6, 13, 21, 31, 34, 35, 47, 124, 133, 151, 153, 159, 173, 187] are inferior to MRP and SuperFine(+MRP) with respect to supertree accuracy (e.g. *FN*-,*FP* rate or *RF*-distance) and, in some cases, also running time [20, 21, 92, 171–173]. We do not evaluate weighted MRP [138], as the above-mentioned evaluation studies indicate that it is not more accurate than SuperFine but often slower than CA-ML. SuperFine uses the GSCM with Overlap scoring as a preprocessing method. Since its resolution has a remarkable impact on the performance of SuperFine, we also report results for the GSCM implementation used by SuperFine (see Section 5.2). BCD on the other hand, uses our improved implementation of the GSCM with the new Unique-Clade-Lost scoring which is evaluated in Section 4.7.

3.5 Reliability of accuracy measurements against the source trees

For biological datasets, we do not know the true supertree, so that we have to compare the estimated supertrees against the source trees for evaluation. In this section we use the simulated datasets, for which we know the model tree, to evaluate how the *SFN* rate and the *SFP* rate behave for different methods on the different datasets. This is important to get an intuition how to interpret the results on biological data. We calculate *SFN* rates

Table 3.4: Overview of the methods we compare in our evaluation and their major differences. For each method, we refer to the implementation(s) we have evaluated.

Method	Combination	Algorithm	Objective Function
GSCM [173] (see Chapter 4)	Supertree (High-Level)	Deterministic, polynomial time	none
BCD (see Chapter 5)			Minimizes the number of character deletions in the matrix representation of the source trees (complementary to MRC/SFIT).
FastRFS [178]			Minimizes the Robinson-Foulds distance to the source trees (constrained search space).
MRP [174]		Local search heuristic	Minimizes the number of character-state changes in the matrix representation of the source trees.
SuperFine [173]			Minimizes the number of character-state changes in the matrix representation of the source trees (after dividing the problem into subproblems).
CA-ML [161]	Supermatrix (Low-Level)		Maximizes the likelihood for a given supermatrix.
CA-MP [174]			Minimizes the number of character-state changes for a given supermatrix.

and *SFP* rates for all simulated datasets and compare them with the true *FN* rates and *FP* rates. To do so, we have to pre-empt partial results for methods we present and evaluate later on in this thesis. Here, we focus on the behaviour of evaluation criteria and not on the performance of tree reconstruction methods.

Swenson *et al.* [172] showed that *SFN* rate and *SFP* rate do not correlate well with true *FN* rate and *FP* rate on SMIDGen and, in some cases, may actually show a slightly negative correlation. Here, we made similar findings. In general, *SFN* rate and *SFP* rate do not correlate well with true *FN* rate and *FP* rate and are also less distinguishing than *FN* rate and *FP* rate (see Figures 3.2 to 3.4 and A.4 to A.10).

The behaviour of *SFN* rate and *SFP* rate is dataset dependent. On the SMIDGenOG dataset, neither the method with the best *FN* rates and *FP* rates nor the model tree show the best *SFN* rates and *SFP* rates (see Figure A.1). In contrast, on the SMIDGen 5500 dataset, the model tree has the best *SFN* rates and *SFP* rates, but the supertree that is most similar to the model tree has worse *SFN* rates and *SFP* rates than other supertrees that are less similar to the model tree (see Figure A.2). For the SuperTriplet benchmark, the results for *SFN* rate and *SFP* rate are similar to those of *FN* rate and *FP* rate, but again we found that the model tree has not the best *SFN* and *SFP* rate (see Figure A.3). Furthermore, *SFN* rate and *SFP* rate are less distinguishing than *FN* rate and *FP* rate.

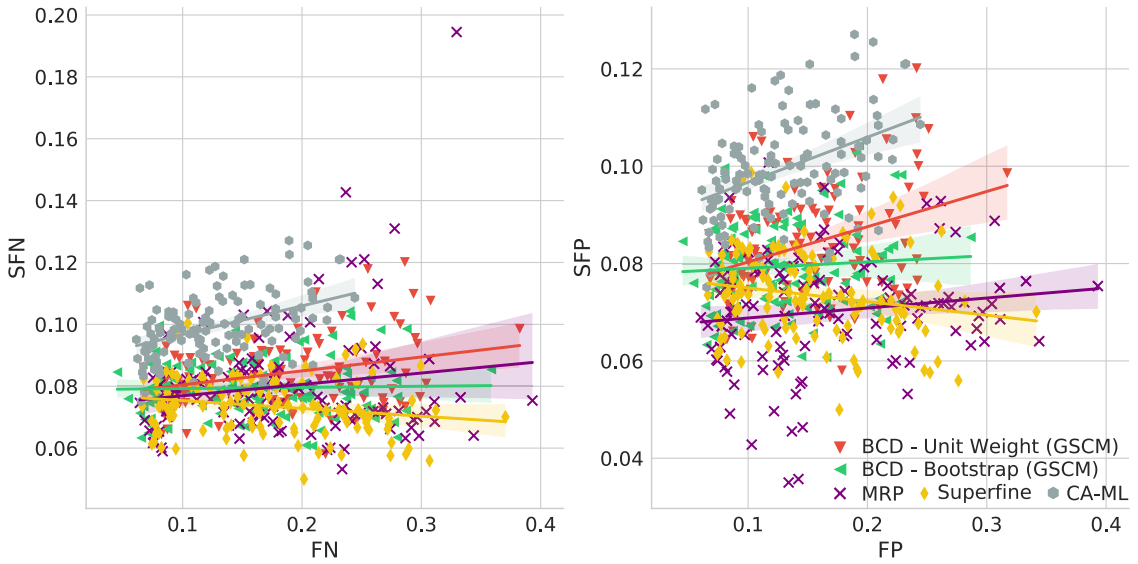


Figure 3.2: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 1000 taxa SMIDGenOG dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

SFN rate and *SFP* rate show also biases dependent on tree reconstruction methods. For all simulated datasets we found that MRP and SuperFine, when evaluated with *SFN* and *SFP* rate, are better than they appear in a comparisons against the model tree (see Figures 3.2 to 3.4). Furthermore, we found slightly negative correlations between *SFN* rate and *FN* rate as well as between *SFP* rate and *FP* rate for SuperFine and in some cases also for MRP. For methods that take support values into account (e.g. bootstrap values or branch length), we found that they, when evaluated with *SFN* rate and *SFP* rate, are worse than they appear in a comparisons against the model tree (see Figures 3.2 and 3.3). This is due to the fact that support values are not considered when calculating *SFN* rates and *SFP* rates. CA-ML trees have the worst *SFN* rates and *SFP* rates by simultaneously having the best *FN* rates and *FP* rates (see Figures 3.2, A.4 and A.5). This is not surprising, hence the CA-ML trees are not computed from the source tree. Sampling errors in the source trees are one explanation for these discrepancies. Nevertheless, the source trees are the information we have to use when reconstructing a supertree, and *SFN* rate and *SFP* rate allow us to compare how good a supertree reflects the input trees. Note, if most of the input trees are wrong, no method will estimate the correct tree, but a method may still reach good *SFN* and *SFP* rates.

For this reasons, results of source tree-based evaluation criteria have to be interpreted with some care and we will report likelihoods or parsimony scores calculated on the underlying sequence data if available. Finally, the *MRP score* is the parsimony score against the MR; we found that results for this score have the same qualitative characteristics as to those for *SFN* and *SFP* rates, and are deferred to the Appendix A.

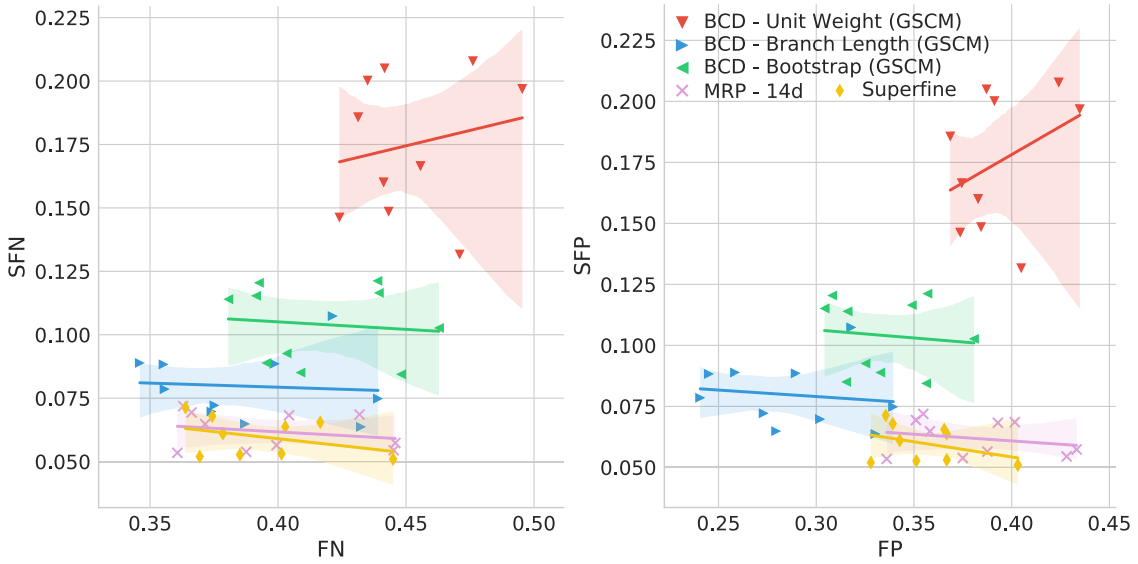


Figure 3.3: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight, Branch length and Bootstrap values), MRP and SuperFine on the 5500 taxa SMIDGenOG dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

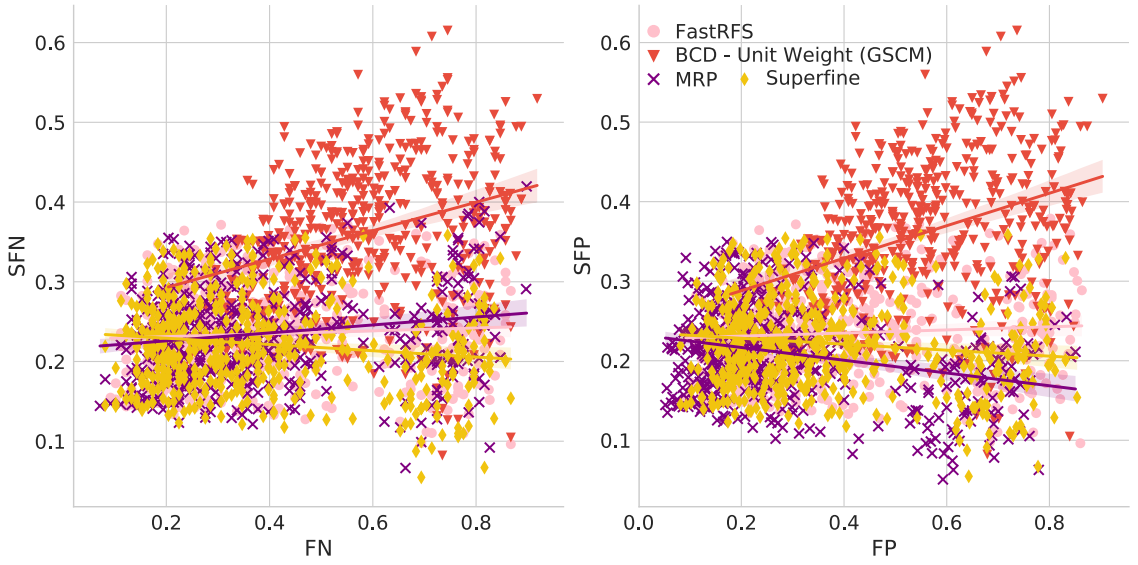


Figure 3.4: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD Unit weight, FastRFS, MRP and SuperFine on the SuperTriplets Benchmark with 75% deletion rate. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

4 Collecting Reliable Clades Using the Greedy Strict Consensus Merger

In Section 2.3 we classified the GSCM [79, 140] as a conservative supertree method; it produces *reliable clades* which are likely to be part of the correct supertree. We can use these clades to reduce the search space when estimating supertrees using a better resolving supertree method (e.g. Bad Clade Deletion Supertrees). The number of reliable clades returned by GSCM is highly dependent on the merging order of the source trees. Although the GSCM only returns clades that are compatible with all source trees, we find that it likewise produces clades which are not supported by any of the source trees (*bogus clades*). These bogus clades do not necessarily have to be part of the supertree. With the objective of improving the GSCM approach as a preprocessing method for rooted input trees, we describe a rooted version of the GSCM, introduce new scoring functions, describe a new randomized GSCM algorithm, and show how to combine multiple GSCM results.

Algorithm 1 Strict Consensus Merger

function SCM(tree T_1, tree T_2)
  $X \leftarrow \mathcal{L}(T_1) \cap \mathcal{L}(T_2)$
  if $|X| \geq 3$ **then** ▷ *Otherwise, the merged tree will be unresolved.*
    calculate $T_{1|X}$ and $T_{2|X}$
    $T_X \leftarrow$ STRICTCONSENSUS$(T_{1|X}, T_{2|X})$
    for all removed subtrees of T_1 and T_2 **do**
      if collision **then** ▷ *Subtrees of T_1 and T_2 attach to the same edge e in T_X (see Figure 4.1)*
        Insert all colliding subtrees at the same point on e by generating a polytomy.
      else
        Reinsert subtree into T_X without violating any of the bipartitions in T_1 or T_2.
      end if
    end for
    return T_X
  end if
end function

4.1 Strict Consensus Merger (SCM)

For a given pair of trees T_1 and T_2 with overlapping taxon sets, the SCM [79, 140] calculates a supertree as follows (see Algorithm 1). Let $X = \mathcal{L}(T_1) \cap \mathcal{L}(T_2)$ be the set of common taxa and $T_{1|X}$ and $T_{2|X}$ the X-induced subtrees. Calculate $T_X =$ STRICTCONSENSUS$(T_{1|X}, T_{2|X})$.

Insert all subtrees, removed from T_1 and T_2 to create $T_{1|X}$ and $T_{2|X}$, into T_X without conflicting with any of the clades in T_1 or T_2. If removed subtrees of T_1 and T_2 attach to the same edge e in T_X, a collision occurs. In that case all subtrees attaching to e will be inserted at the same point by subdividing e and creating a polytomy at the new vertex (see Figure 4.1). Note that neither the strict consensus nor the collision handling inserts clades into the supertree T_X that conflict with any of the source trees.

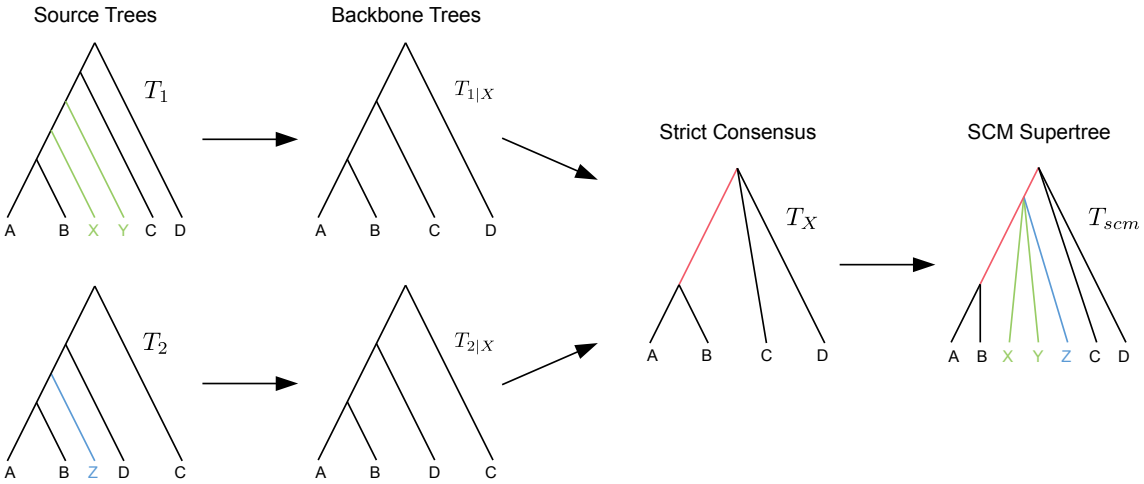


Figure 4.1: Example SCM run including collision handling. The backbone trees $T_{1|X}$ and $T_{2|X}$ are merged using the strict consensus. The remaining subtrees of T_1 and T_2 are colored in green and blue, respectively. Both subtrees attach to the same edge in T_X (red). The green and blue subtrees are inserted into T_X by generating a polytomy (collision handling).

4.2 Greedy Strict Consensus Merger (GSCM)

The GSCM [79, 140] algorithm generalizes the SCM idea to combine a collection $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ of input trees into a supertree T with $\mathcal{L}(T) = \bigcup_{i=1}^{k} \mathcal{L}(T_i)$ by pairwise merging trees until only the supertree is left (see Algorithm 2). Let $score(T_i, T_j)$ be a function returning an arbitrary score of two trees T_i and T_j. At each step, the pair of trees that maximizes $score(T_i, T_j)$ is selected and merged, resulting in a greedy algorithm. Since the SCM does not insert clades that contradict any of the source trees, the GSCM returns a supertree that only contains clades that are compatible with all source trees.

4.3 Tree Merging Order

Although the SCM for two trees is deterministic, the output of the GSCM is influenced by the order of selecting pairs of trees to be merged, since the resulting number and positions of collisions may vary.

Let $T_1, \ldots, T_n$ be a collection of input trees we want to merge into a supertree using the GSCM. When merging two trees, the strict consensus merger (SCM) accepts only clades that can be safely inferred from the two source trees. In case of a collision during reinsertion of unique taxa, the colliding subtrees are inserted as a polytomy on the edge where the collision occurred.

Algorithm 2 Greedy Strict Consensus Merger

```
1: function GSCM(trees {T_1, T_2, ..., T_k})
2:     S ← {T_1, T_2, ..., T_k}
3:     while |S| ≥ 2 do
4:         T_i, T_j ← PICKTREEPAIR(S)        ▷ return best tree pair regarding score(T_i, T_j)
5:         S ← S \ {T_i, T_j}
6:         T_scm ← SCM(T_i, T_j)
7:         S ← S ∪ {T_scm}
8:     end while
9:     return T_scm
10: end function
```

If collisions of different merging steps occur on the same edge, the polytomy created by the first collision may cause the following collisions to not occur. Such obviated collisions induce *bogus clades* (see Figure 4.2) which cannot be inferred unambiguously from the source trees and hence should not be part of the supertree. A clade C of a supertree $T = \text{GSCM}(T_1, \ldots, T_n)$ is a *bogus clade* if there is another supertree $T' = \text{GSCM}(T_1, \ldots, T_n)$ (based on a different tree merging order) that contains a clade C' conflicting with C (see Figures 4.2a and 4.2c). Note that bogus clades cannot be recognized by comparison to the source trees since they do not conflict with any of the source trees $T_1, \ldots, T_n$. All clades in the GSCM supertree that are not bogus, are called *reliable clades*.

Because of these bogus clades the GSCM supertree with the highest resolution may not be the most accurate supertree. Since, clades that are used to limit the search space of better resolving methods will be part of the final supertree (even if they are wrong), it is important to prevent bogus clades. But to use the GSCM as an efficient preprocessing, we want to determine a preferably large number of the reliable clades. Hence, we need a scoring functions that maximizes the number of reliable clades by simultaneously minimizing the number of bogus clades.

4.4 Scoring Functions

We present three novel tree selection scoring functions that produce high quality GSCM supertrees with respect to F_1-score and number of unique clades (unique in terms of not occurring in a supertree resulting from any of the other scorings). In addition, we evaluate the original *Resolution* scoring [140], as well as the *Unique-Taxa* and *Overlap* scorings [173]. Let $uc(T, T') = \mathcal{V}(T_{|\mathcal{L}(T)\setminus\mathcal{L}(T')} \setminus \mathcal{L}(T))$ be the set of unique clades of T compared to T'.

Unique-Clades-Lost scoring (UCL): Minimize the number of unique clades that get lost:

$$score(T_i, T_j) = -\Big(\big(|uc(T_i, T_j)| - |uc(scm(T_i, T_j), T_j)|\big) + \big(|uc(T_j, T_i)| - |uc(scm(T_i, T_j), T_i)|\big)\Big)$$

Unique-Clade-Rate scoring: Maximize the number of preserved unique clades:

$$score(T_i, T_j) = \frac{|uc(T_i, T_j)| + |uc(T_j, T_i)|}{|uc(scm(T_i, T_j), T_i)| + |uc(scm(T_i, T_j), T_j)|}$$

Collision scoring: Minimize the number of collisions:

$$score(T_i, T_j) = -(\text{ number of edges in } \text{SCM}(T_i, T_j) \text{ where a collision occurred})$$

Unique Taxa scoring [173]: Minimize the number of unique taxa:

$$score(T_i, T_j) = -|\mathcal{L}(T_i) \Delta \mathcal{L}(T_j)|$$

Overlap scoring [173]: Maximize the number of common taxa:

$$score(T_i, T_j) = |\mathcal{L}(T_1) \cap \mathcal{L}(T_2)|$$

Resolution scoring [140]: Maximize the resolution of the SCM tree:

$$score(T_i, T_j) = \frac{|\mathcal{V}(\text{SCM}(T_i, T_j))| - |\mathcal{L}(\text{SCM}(T_i, T_j))|}{|\mathcal{L}(\text{SCM}(T_i, T_j))| - 1}$$

4.5 Combining Multiple Scorings

In general, supertrees created with the GSCM using different scoring functions contain different clades. To collect as many reliable clades as possible, we compute several GSCM supertrees using different scoring functions and combine them afterwards.

Reliable clades between all possible GSCM supertrees are pairwise compatible for a given set of source trees. In contrast, bogus clades can be incompatible between different GSCM supertrees (see Figure 4.2). Thus, every conflicting clade has to be a bogus clade. By removing incompatible clades we may eliminate bogus clades but none of the reliable clades when combining the GSCM supertrees. To do so, we simply calculate a semi-strict consensus [24] tree of all GSCM supertrees. It should be noted that bogus clades are only eliminated if they induce a conflict between at least two supertrees (see Figure 4.2). Hence, there is no guarantee to eliminate bogus clades.

Let *Combined-3* be the combination of the Collision, Unique-Clade-Rate and Unique-Clades-Lost scoring functions. Furthermore, *Combined-5* combines the Collision, Unique-Clade-Rate, Unique-Clades-Lost, Overlap and Unique-Taxa scoring functions.

4.6 Randomized Tree Merging Order

Generating many different GSCM supertrees increases the probability of both detecting all reliable clades and eliminating all bogus clades. To generate a larger number of GSCM supertrees, randomizing the tree merging order of the GSCM algorithm may be more suitable than using a variety of different tree selection scorings. To this end, we replace pickTreePair(S) in Algorithm 2 with randomlyPickTreePair(S) that picks a pair of trees from S with probability:

$$P(T_i, T_j) = \frac{score(T_i, T_j)}{\sum\limits_{T_a, T_b \in \mathcal{S}, a \neq b} score(T_a, T_b)}, i \neq j$$

Running the randomized GSCM for different scoring functions multiple (k) times allows us to generate a large number of supertrees containing different clades. The resulting trees are combined using a semi-strict consensus as described in the previous section. For combined scorings (Combined-n) with n different scoring functions we calculate $\frac{k}{n}$ supertrees for each of the scoring functions and combine all k supertrees using the semi-strict consensus.

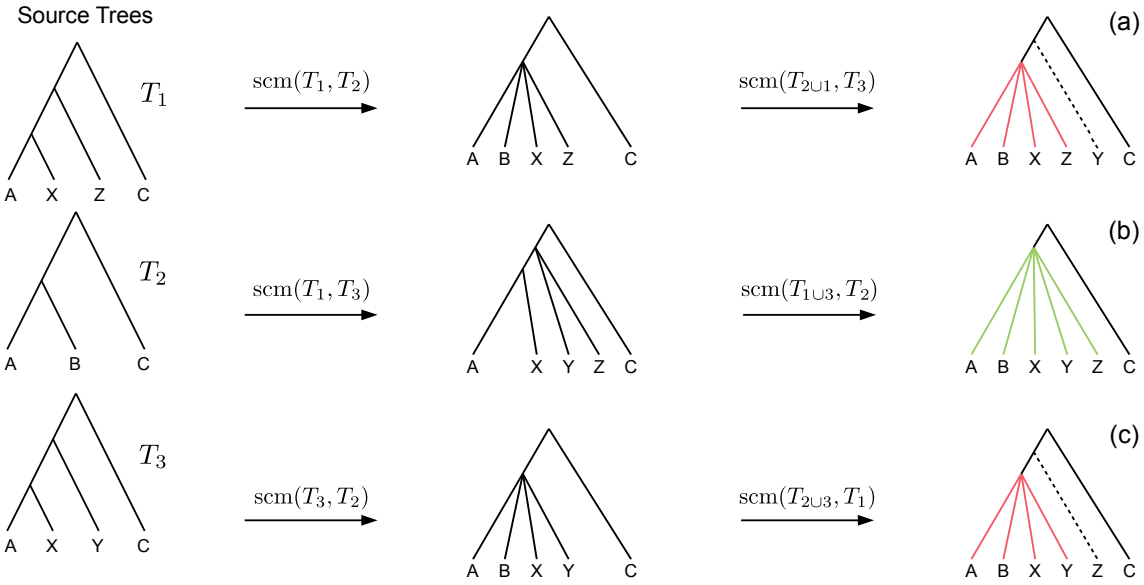


Figure 4.2: Example where the collision handling inserts bogus clades (red) into the supertree. Bogus clades are induced by obviated collisions, which are are prevented by a previous collision on the same edge. Supertrees (a) and (c) are estimated on the same set of source trees but contain conflicting clades ((ABXZ) conflicts with (ABXY)) induced by different merging orders. The correct supertree is (b).

4.7 Results

We find the influence of scoring functions and randomization to increase with the size of the input data (as expected for greedy algorithms). Thus, we present only the 1000 taxa dataset. The overall effects are similar for all datasets (see Appendix A).

The scaffold factor highly influences the quality of the supertrees (see Figures 4.3 and 4.4). In general, all scoring functions profit from a large scaffold tree. In particular, for a scaffold factor of 100 % nearly all scorings perform equally well and better than for all other scaffold factors. In that case, the scaffold tree contains already all taxa of the supertree which simplifies the GSCM problem since no collision occurs, when the taxon set of one tree is a subset of the taxon set of the other tree. Thus, starting with the scaffold tree and merging the remaining source trees in arbitrary order leads to the optimal solution. However, the Resolution and Unique-Taxa scoring functions do not necessarily pick the scaffold tree in the first step, and therefore, do not necessarily lead to an optimal solution. In contrast, the Overlap scoring — which does not perform well for small scaffold tree sizes (20 %, 50 %) — produces optimal solutions for a scaffold factor of 100 %.

Comparing the different scoring functions, we find that in general, the *FN*-rate varies more than the *FP*-rate (see Figure 4.3). Our presented scoring functions (Collision, Unique-Clade-Lost, Unique-Clade-Rate) decrease the *FN*-rate, without increasing the *FP*-rate (see Figure 4.3). This leads to the highest F_1-scores for all scaffold factors (see Figure 4.4a). They clearly outperform the Resolution, Overlap and Unique-Taxa scorings for scaffold factors 50 % and 75 %. The differences in the F_1-scores are significant (p-values below 0.000033). For a scaffold factor of 20 % the improvements of our scoring functions in comparison to Unique-Taxa are not significant. For a scaffold factor of 100 % the Overlap

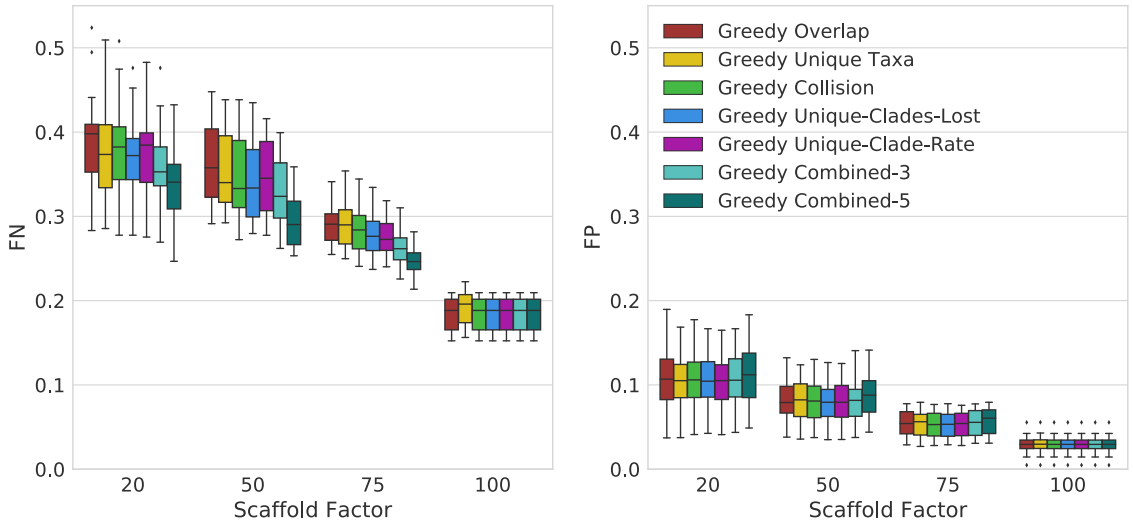


Figure 4.3: *FN*-rates (left) and *FP*-rates (right) of single scorings functions (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5) for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 1000-taxon dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scoring functions. The error bars show the standard error.

scoring function is on par with our scoring functions (all of them return the trivial optimal solution). The differences between Collision, Unique-Clade-Lost and Unique-Clade-Rate are not significant. Nevertheless Unique-Clade-Lost provides the most robust and input independent results. For scaffold factors of 20 % and 50 %, Resolution and Overlap show significantly worse (p-values ≤ 0.000006) F_1-scores than all other scoring functions (see Figure 4.4a). There is no significant difference (p-values > 0.09) between Resolution and Overlap scoring. For scaffold factors of 75 % and 100 %, the Resolution scoring function performs significantly worse than all others. For a scaffold factor of 75 %, there is no significant difference between Unique-Taxa and Overlap scoring. For a scaffold factor of 100 %, the Overlap scoring function performs better than Unique-Taxa, which is still significantly better than Resolution.

Even for equally-performing scoring functions, the resulting trees are often different (except for scaffold factor 100 %). Thus, we combine the GSCM supertrees computed with different scorings using the semi-strict consensus. Since the Resolution scoring function performs badly, we only combine the remaining five scoring functions. The combination of different scoring functions strongly improves the *FN*-rate. Thus, the combined supertrees have improved F_1-scores for all scaffold densities (see Figure 4.4b). The combination of Collision, Unique-Clade-Lost, Unique-Clade-Rate, Overlap and Unique-Taxa (Combined-5) results in the best F_1-score. However, Combined-5 has a significantly worse *FP*-rate than all other scorings. In contrast, the combination of Collision, Unique-Clade-Lost, Unique-Clade-Rate scoring (Combined-3) shows no significant decline of the *FP*-rate.

To collect as many *TP* clades as possible, we use a randomized tree merging order generating multiple (k) supertrees which are combined using the semi-strict consensus. Generally we found that randomization further improves the F_1-score in comparison to the single scoring functions (see Figure 4.4d). Compared to the Combined-5 scoring there is only an improvement of the F_1-score for scaffold factors of 50 % and 75 %. Again, these

improvements come with a significant increase of the *FP*-rate. Already for 25 random iterations, all presented scoring functions perform on almost the same level (see Figure 4.4c). As the number of random iterations increases, the difference between the reported scoring functions vanishes.

4.8 Discussion

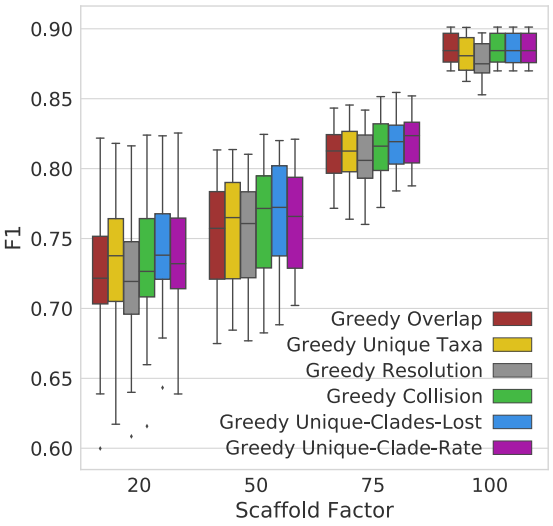
We found that collisions not only destroy source tree clades but also introduce bogus clades to the supertree. Thus, the scoring functions that minimize the number of collisions perform best. Combining multiple GSCM supertrees using a semi-strict consensus method helps to better resolve the supertree.

We presented three novel scoring functions (Collision, Unique-Clades-Lost, Unique-Clade-Rate) that increase the number of true positive clades and decrease the number of false positive clades of the resulting supertree. Unique-Clades-Lost score is the overall best-performing scoring function.

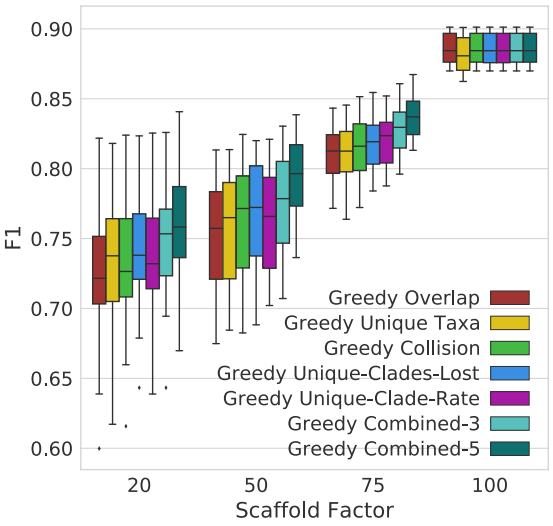
Combining the supertrees calculated by these three scorings using a semi-strict consensus algorithm further increases the number of true positive clades without a significant increase of the false positives.

For almost all presented scoring functions, the highest F_1-scores and best resolved trees are achieved using randomized GSCM. Randomization indeed increases the number of true positive clades but also significantly increases false positive clades. Thinking of GSCM as a preprocessing method, those false positive clades will have a strongly negative influence on the quality of the final supertree.

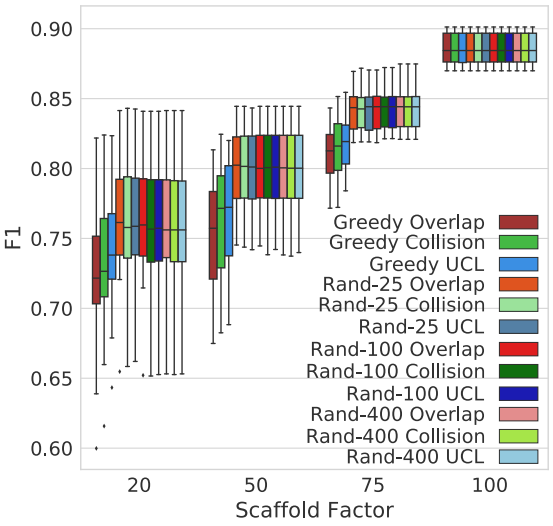
Depending on the application, "best performance" is characterized differently. The most conservative approach is our Unique-Clade-Lost scoring function which increases the *TP*-rate by 5 % while decreasing the *FP*-rate by 2 % compared to Overlap. To use GSCM as a preprocessing method, we recommend a combination of Collision, Unique-Clade-Lost and Unique-Clade-Rate (Combined-3) scoring. In comparison to the Overlap scoring function, this increases the number of true positive clades by 9 % without a significant increase of false positive clades. The overall best F_1-score can be achieved with a combination of randomized Collision, Unique-Clade-Lost, Unique-Clade-Rate, Overlap and Unique-Taxa (Combined-5) scoring.



(a) Comparison of scoring functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clades-Rate).



(b) Comparison of single scoring functions (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5).



(c) Comparison of scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.

(d) Comparison of single (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost) and combined (Combined-5) scorings. Both with 400 random iterations and without randomization.

Figure 4.4: F_1-scores of different scoring functions (including combined scorings) with and without randomization for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 1000 taxa dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scorings. The integer value after the keyword "Rand" represents the number of randomized iterations. The error bars show the standard error.

5 Bad Clade Deletion Supertrees

We introduce the Bad Clade Deletion (BCD) supertree algorithm, a polynomial time top-down heuristic that minimizes the number of column deletions of a matrix representation (Baum-Regan encoding), so that the resulting matrix allows for a directed perfect phylogeny.

BCD applies the FlipCut idea for a new objective function, namely *Minimum Column Deletion*. It inherits the polynomial worst-case running time but is even faster in practice. We integrate meta information (bootstrap values and branch lengths) into BCD. Further, we present a preprocessing that uses a set of reliable clades to reduce the search space. In particular, we use the Greedy Strict Consensus Merger (see Section 4.2) to calculate the set of reliable clades.

5.1 The Bad Clade Deletion Algorithm

Let $\mathcal{T}$ be the set of source trees. Assume that the matrix $M := M(\mathcal{T})$ does not allow for a perfect phylogeny; how can we "correct" the matrix M accordingly? Brinkmeyer *et al.* [21] introduced a top-down heuristic for the MRF problem that uses minimum cuts in a graph representation of M. This algorithm, in turn, is based on the method of Pe'er *et al.* [125] for deciding the incomplete perfect phylogeny problem.

Minimum Column Deletion. Here, we consider a different way of "correcting" the matrix M: We remove a minimum number of columns (characters) from M, so that the resulting matrix allows for a perfect phylogeny. This formulation allows for an intuitive phylogenetic interpretation: Instead of removing columns from the matrix, an equivalent formulation of the problem is to remove clades from the source trees. Let the *incompatibility graph* of M be a graph that contains a vertex for each matrix column and an edge between all vertices that correspond to incompatible matrix columns. Minimizing the number of columns to delete corresponds to a minimum vertex cover on the incompatibility graph. This is complementary to maximizing the number of characters to keep (maximum independent set on the incompatibility graph), which is the objective function of the well known MRC. Hence, the optimal solution for one of these problems is also an optimal solution for the other one; both optimization problems are NP-hard but not identical with respect to parameterized complexity and approximability.

Algorithm description. The Bad Clade Deletion supertree algorithm can be considered as the FlipCut algorithm with a particular choice of weights in the underlying graph (see below). Hence, all algorithmic results from Brinkmeyer *et al.* [21] directly carry over to the BCD algorithm (see Figure 5.1).

A high-level description of the algorithm is as follows: The algorithm proceeds in a recursive top-down fashion. In each recursive call, a subset of taxa and a subset of characters are provided; the subset of taxa is added as a clade to the supertree. A graph is constructed from the input matrix M and the two subsets, as proposed in Section 2.3. If this graph

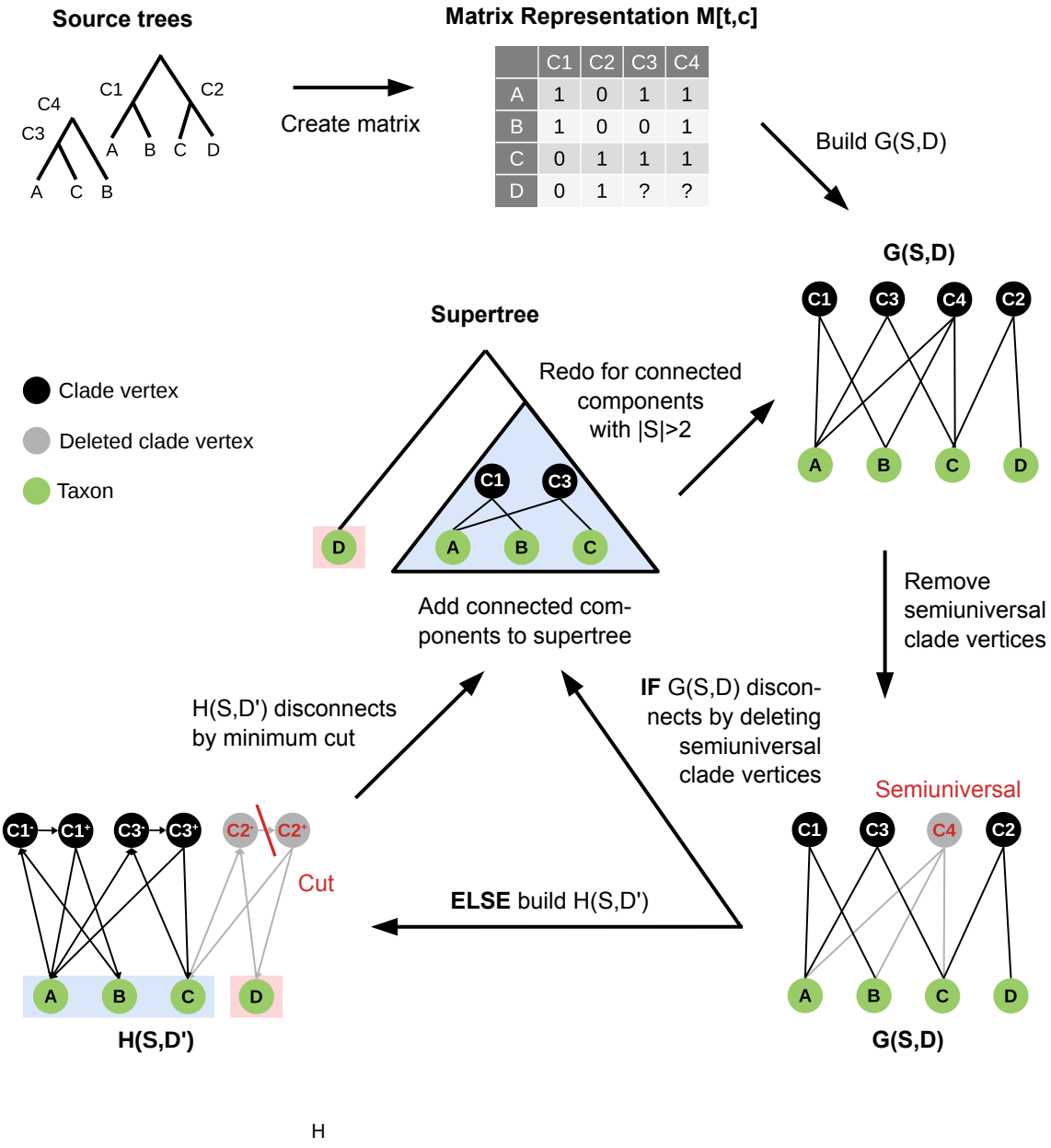


Figure 5.1: The BCD algorithm: For a subset S of taxa (green) and a subset D of characters (black), $G(S, D)$ is a bipartite graph, where an edge $\{t, c\}$ is present if and only if $M[t, c] = 1$, for $t \in S$ and $c \in D$. A character vertex is semiuniversal if $M[t, c] \in \{1, ?\}$ for all $t \in S$. For minimum cut computation, we transform $G(S, D)$ into $H(S, D')$. Supertree clades have the same colors (red or blue) as their corresponding connected components.

is disconnected, the algorithm directly recurses on the connected components; otherwise, we search the graph for a minimum cut and remove the cut before recursing. If multiple optimal cuts exist, we choose one randomly. Recursion stops when the subset of taxa contains only a single taxon.

We now give the details of the BCD algorithm (an example is given in Figure 5.1). For a subset $S \subseteq \{1, \ldots, n\}$ of taxa and a subset $D \subseteq \{1, \ldots, m\}$ of characters, $G(S, D)$ is a bipartite graph with vertex sets S and D, and edges as follows: First, we build a graph such that an edge $\{t, c\}$ is present if and only if $M[t, c] = 1$, for $t \in S$ and $c \in D$. A character vertex $c \in D$ is *semiuniversal* (in S, D) if $M[t, c] \in \{1, ?\}$ holds for all $t \in S$. We immediately remove all semiuniversal character vertices from the graph [125].

The BCD algorithm proceeds as follows: We start with $S \leftarrow \{1, \ldots, n\}$ and $D \leftarrow \{1, \ldots, m\}$. We then construct the graph $G(S, D)$. If this graph is not connected, we recurse on each connected component S', D' of the BCD graph with $|S'| > 1$. The sets S' of taxa computed during the course of the algorithm form a hierarchy which is transformed into the desired supertree.

If $G(S, D)$ is connected at some point, the algorithm disconnects the graph by means of modifying the input matrix M. In contrast to the FlipCut algorithm, we do not allow edges to be removed, so all edges in $G(S, D)$ get weight infinity. The only valid operation to split $G(S, D)$ is deleting a subset of character nodes from D. For the moment, we assume all characters $c \in D$ in $G(S, D)$ to have unit weight $w(c) := 1$. The weight of a bipartition of taxon vertices is the minimal cost of a set of character deletions, such that the two subsets of taxon vertices lie in separate components of the resulting graph. We search for a bipartition of minimal weight. To efficiently find a minimum bipartition, we fix one taxon vertex s, and for all other taxon vertices t we search for a minimum s-t-cut, allowing only character deletions. Among these cuts, the cut with minimal weight is the solution to the above problem. To find a minimum s-t-cut with character deletions, we transform $G(S, D)$ into a directed network $H(S, D')$ with capacities: Each taxon vertex t is also a vertex in the network, each character vertex c is transformed into two vertices c_- and c_+ plus an arc (c_-, c_+) in the network, and an edge $\{t, c\}$ in $G(S, D)$ is transformed to two arcs (t, c_-) and (c_+, t) in the network. Arcs (c_-, c_+) have capacity $w(c)$, all other arcs have *infinite* capacity. By the generalized min-cut max-flow theorem [46, 57], finding a minimum cut in $G(S, D)$ is equivalent to computing a maximum flow in the network $H(S, D')$ [58]. Note that for all taxa s, t, the maximum s-t-flow in $H(S, D')$ equals the maximum t-s-flow.

The BCD algorithm proceeds in $n - 1$ *phases*; in each phase, the clade $S \subseteq \{1, \ldots, n\}$ is added to the output, and a bipartition of S is computed. The algorithm proceeds greedily, by choosing the best bipartition in every phase.

Lemma 1 (Brinkmeyer *et al.* [21])**.** *Given an input matrix M over $\{0, 1, ?\}$ for n taxa and m clades, the BCD algorithm computes a supertree in $O(mn^3)$ time.*

From the algorithm we can infer that each clade in the supertree is supported by at least one of the source trees: We only ever remove columns from the matrix; we never modify or add to the matrix.

Lemma 2. *Given an input matrix M over $\{0, 1, ?\}$ the BCD algorithm computes a supertree where each clade is supported by at least one source tree and no clade contradicts all of the source trees.*

On the other hand, the reconstructed supertree does not necessarily minimize the number of inner nodes among all supertrees that are consistent with the same matrix columns:

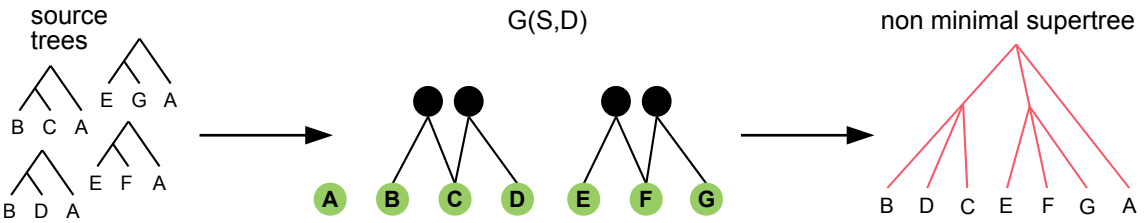


Figure 5.2: Example where Pe'er's algorithm does not return a minimal tree

BCD supertrees may contain several clades where a single joint clade would be sufficient [23, 81]. Our method inherited this property from the underlying algorithm of Pe'er *et al.* [125], and shares it with supertree methods that build on the algorithm by Aho *et al.* [1] (see Figure 5.2). If there are are no contradictions in the input data the BCD algorithm falls back to the algorithm of Pe'er *et al.* [125] which guarantees to return a perfect phylogeny if one exists.

Lemma 3 (Pe'er *et al.* [125])**.** *Given a compatible input matrix M over $\{0, 1, ?\}$ the BCD algorithm returns the corresponding directed perfect phylogeny.*

Note that this is not guaranteed by supertree methods that use local search heuristics (e.g. MRP, MRF, MRL, MRC).

5.1.1 Weighting Strategies

By weighting $G(S, D)$, we can incorporate information about the "reliability of clades" (characters). Here, we show how BCD uses branch lengths and bootstrap values:

- **Unit weights (UW).** All characters in the BCD graph have weight one. Using unit weights, a minimum cut is the removal of the smallest set of character vertices that disconnects the graph.
- **Branch lengths (BL).** Brinkmeyer *et al.* [21] found that a clade that stems from a long branch is more stable (and, therefore, more likely to be correct) than a clade with a short branch. The weight of a character c is set to $w_{\mathrm{BL}}(c) := \dfrac{l(e)}{l_{max}}$, where $l(e)$ is the length of the branch e in the source tree that generated clade/character c, and l_{max} the longest branch of all source trees.
- **Bootstrap values (BS).** If available, we can use bootstrap values to weight the BCD graph. A bootstrap value tells us how sure we are about a clade. The weight of a character c is set to $w_{\mathrm{BS}}(c) := \dfrac{b(v)}{100}$, where $b(v)$ is the bootstrap value of the node v in the source tree corresponding to c.
- **Tree weight.** In addition to the weightings above, it is possible to modify the weights for source trees independently. For this, we multiply any of the above scores with a factor individually given for each source tree. This is not used in our evaluations.

5.1.2 GSCM Preprocessing

Since BCD is a greedy heuristic, restricting the search space in a sensible way will help to improve its accuracy. One way to do so, is to use a set of *reliable clades*, for which it is

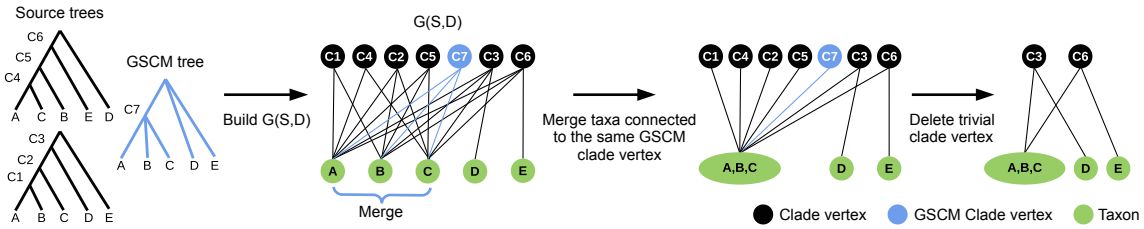


Figure 5.3: Data reduction of $G(S, D)$ using GSCM clades. Taxon vertices (S) displayed in green, character (D) vertices in black. Character vertices induced by the GSCM tree are blue.

highly likely that they are part of the correct supertree. To use reliable clades for BCD, we add them to the matrix M, and give each corresponding character vertex infinite weight. Reliable character vertices cannot be deleted during minimum cut computations; they are only deleted from $G(S, D)$ when they become semiuniversal, implying that they are already part of the supertree. All taxon vertices $t_1 \dots t_n$ in $G(S, D)$ that are connected to the same reliable character vertex have to end up in the same connected component of $G(S, D)$, and can be merged into a single taxon vertex. We then delete all trivial character vertices, each of which is connected to exactly one taxon vertex. This reduces the number of vertices in $G(S, D)$ without changing its minimum cut (see Figure 5.3).

We estimate reliable clades using the Greedy Strict Consensus Merger (GSCM) supertree method as it is described in Chapter 4. Per default we apply the Unique-Clades-Lost scoring, which outperformed other known scorings but scales quadratically instead of linearly with the number of input trees.

5.1.3 Merging Clade Vertices

The most time-consuming part of the BCD algorithm is searching for minimum cuts of $G(S, D)$. Using the GSCM tree can strongly reduce running times of this step; here, we describe another algorithm engineering trick we use to further improve running times. To reduce the size of matrix M, it is common practice to merge identical matrix columns, summing up their weights. Identical matrix columns occur only rarely in the input matrix, as this requires input trees with identical taxon sets. Recall that our method allows the user to define an individual weight for each input tree. When searching for minimum cuts in $G(S, D)$, '0' and '?' entries in M are treated identical, which allows us to merge clade vertices corresponding to trees with different taxon sets. Hence, we merge all clade vertices in $G(S, D)$ that are adjacent to the same set of taxon vertices, and sum up their weights. This can vastly reduce the number of vertices in $G(S, D)$ without changing the minimum cut.

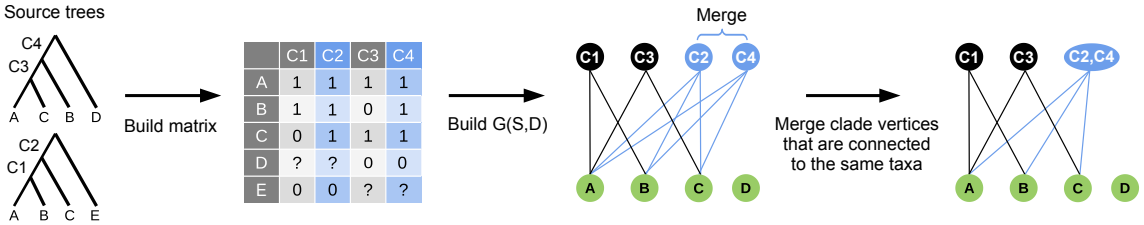


Figure 5.4: Merging clades connected to the same set of taxa in $G(S, D)$

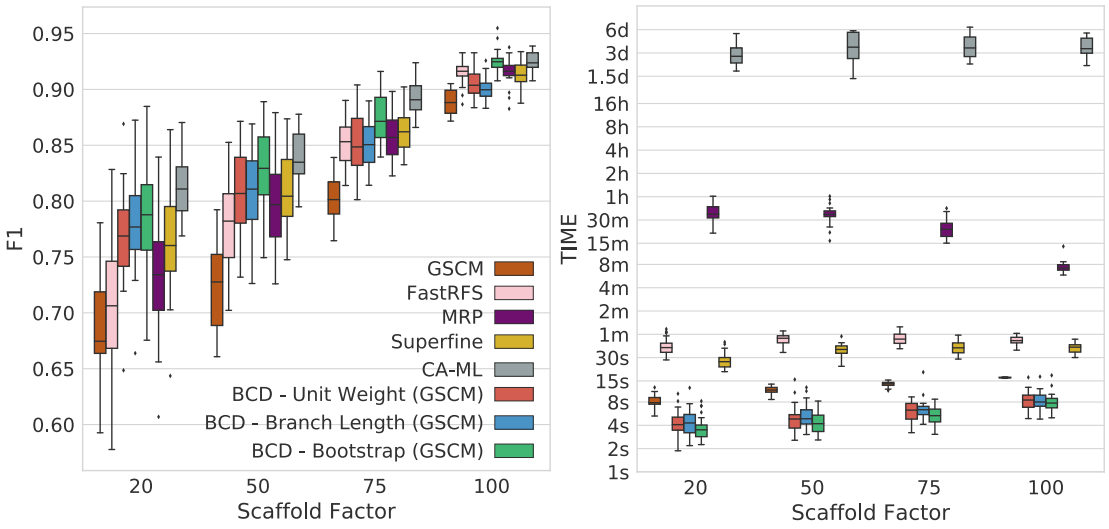


Figure 5.5: F_1-score (left) and running times (right) of the evaluated tree reconstruction methods on the simulated SMIDGenOG (1000 Taxa) dataset. Running times are on a logarithmic scale. On the x-axis are the different scaffold factors plotted.

5.2 Results

SMIDGenOG Results

We now describe results for the SMIDGenOG dataset with 30 replicates and 1000 taxa in the model tree. We found that using the GSCM "guide tree" consistently improves BCD supertree accuracy (see Figures A.24 to A.26); on large datasets, it also reduces running times. In the following, we omit results for BCD without GSCM. We repeated our analysis for smaller datasets with 500 and 100 taxa model trees, as well as the original SMIDGen dataset, but found no significant differences (see Figures A.20 to A.22, A.24 and A.25). Finally, we found that the FlipCut supertree method, being the predecessor of BCD, performs considerably worse than BCD (see Figures A.20 to A.22); to this end, we will not consider FlipCut in the following.

Accuracy. Using GSCM as an independent supertree method shows the by far worst overall performance (F_1-score) of all methods (see Figure 5.5) but shows best *FP* rate which qualifies it as a preprocessing method (see Figures A.26c and A.26d). BCD (Unit Weight and Branch Length) already beats MRP for scaffold factors 20% and 50%. For scaffold factor 75%, the difference to MRP is not significant, and for scaffold factor 100% it performs worse than MRP. The differences between BCD Unit Weight and BCD Branch Length are not significant. BCD Bootstrap performs significantly better than any other evaluated supertree method, for all scaffold factors. CA-ML shows the overall best performance (significant for scaffold factors 20%, 50%, 75%). For scaffold factor 100%, the difference between BCD Bootstrap and CA-ML is not significant. Whereas SuperFine has a significantly higher F_1-score than MRP for scaffold factors 20% and 50%, it performs equally for scaffold factor 75% and 100%. FastRFS results quality depends on the scaffold factor: While it performs much worse than MRP for small scaffold factors, it is almost on par with MRP and SuperFine for scaffold factor 100%. Nevertheless, the small difference

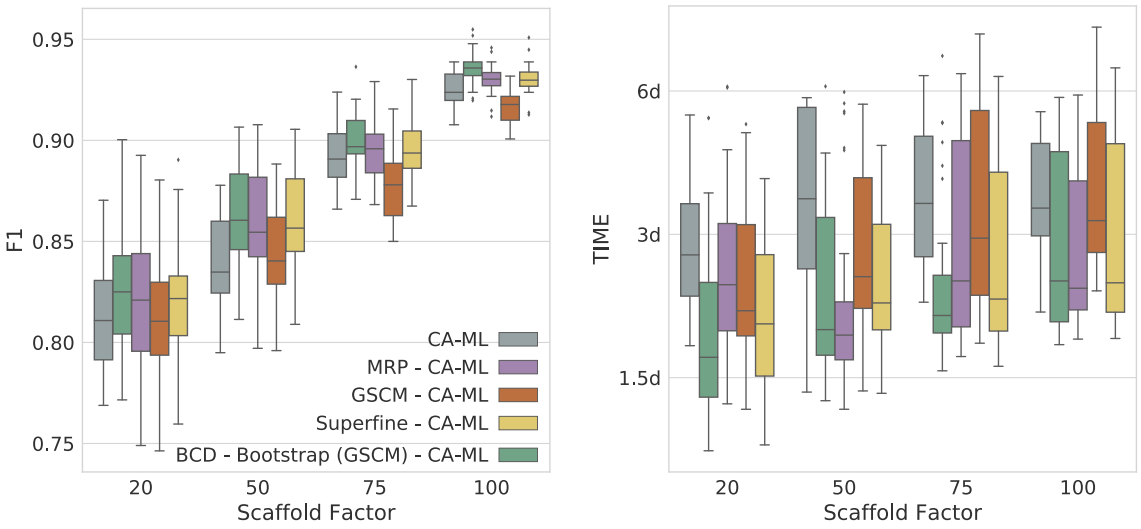


Figure 5.6: Performance of different supertrees as a starting tree for CA-ML with RAxML regarding F_1-score (left) and running times (right) on the simulated SMIDGenOG (1000 Taxa) dataset. Running times are on a logarithmic scale. On the x-axis are the different scaffold factors plotted.

to SuperFine and MRP is significant. Comparing CA-ML with different starting trees demonstrates that supertrees can be used to improve CA-ML trees (see Figure 5.6). Here, all CA-ML trees show a significantly better F_1-score than the supertree methods. We found that a better starting tree results in a better CA-ML tree: CA-ML using the BCD Bootstrap supertree as starting tree has a significantly higher F_1-score than the default CA-ML tree, for all scaffold factors. The trees with the overall best F_1-score on the SMIDGenOG dataset were estimated using CA-ML with the BCD Bootstrap starting tree.

Running time. With running times between 4 and 8 seconds (including GSCM preprocessing), all BCD variants are much faster than all other evaluated methods (see Figure 5.5). Note that BCD (including GSCM preprocessing) is faster than the GSCM implementation used by SuperFine. SuperFine needs around 30s on average for one replicate of this dataset; FastRFS need at most 1 minute. MRP is by far the slowest supertree method, with running times of 40 minutes for scaffold factor 20%, and 8 minutes for scaffold factor 100%. Local search heuristics for MRP seem to converge faster for data with large scaffold trees, whereas the increased number of characters appears to increase the running time of the other methods. CA-ML with default starting tree requires between 2 and 6 days, whereas BCD requires between 4 and 8 seconds. Running times of CA-ML decrease with increasing starting tree accuracy, and CA-ML with BCD starting tree is up to two times faster than CA-ML with default starting tree (see Figure 5.6).

SMIDGenOG-5500 Results

MRP did not finish in reasonable time; hence, we report its results after 1 day, 7 days and 14 days of running time. In view of the small number of replicates, we refrain from reporting significance.

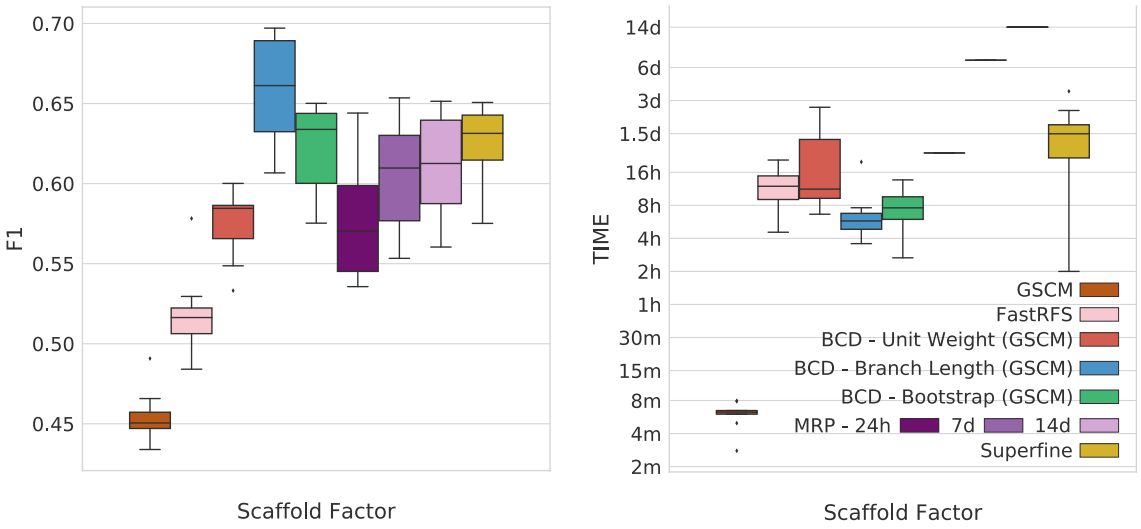


Figure 5.7: F_1-score (left) and running times (right) of MRP, SuperFine, GSCM, FastRFS and BCD on the simulated SMIDGenOG (5500 Taxa) dataset. MRP did not finish after 14 days of computation; we report MRP results after 1 day, 7 days and 14 days. Running times are shown on a logarithmic scale.

Accuracy. BCD Branch Length reaches the overall best accuracy, outperforming all other methods for 10 of 10 replicates. BCD Bootstrap accuracy is considerably worse, which might be attributed to the small number of bootstrap replicates, see the discussion below; it outperforms MRP and SuperFine in 8 of 10 cases. GSCM is consistently outperformed by all other methods. In all cases, FastRFS shows a lower F_1-score than all methods but BCD Unit Weight with GSCM (for which it is outperformed in 9 of 10 cases) and GSCM (see Figure 5.7).

Running time. This dataset clearly demonstrates the benefit of polynomial time algorithms, see Figure 5.7: MRP did not finish after 14 days of computation, whereas BCD requires about 7h. FastRFS requires about about 12h. SuperFine needs up to 2 days, and will fall back to MRP computation when the GSCM preprocessing is not effective.

SuperTriplets Benchmark Results

For the SuperTriplets Benchmark, we discuss the results for different deletion rates with 10 and 50 source trees (see Figure 5.8). For additional evaluations on the SuperTriplets dataset see Figures A.31 to A.33.

Accuracy. For both, 10 and 50 input tree data, BCD performs worse than MRP, SuperFine and FastRFS; this is particularly the case for 50 input trees and 75% taxa deletion. If the number of characters is sufficiently large (25% deletion), BCD Unit Cost performs almost on par with MRP, SuperFine and FastRFS. With increasing data deletion rates BCD Unit Cost performs worse. For deletion rates of 50% and 75%, BCD Unit Cost shows a significantly lower F_1-score than MRP, SuperFine and FastRFS. MRP, SuperFine and FastRFS are on par for deletion rates of 25% and 50%. For a deletion rate of 75%, MRP outperforms SuperFine and FastRFS performs worse than SuperFine. Hence the accuracy

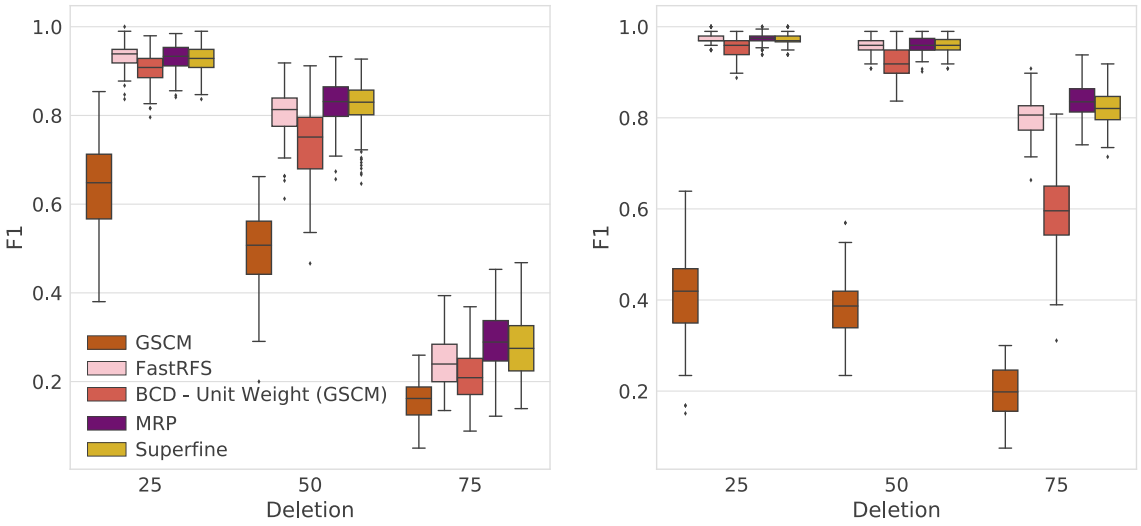


Figure 5.8: F_1-score of SuperFine, MRP, FastRFS and BCD on the simulated SuperTriplets Benchmark. Results for 10 input trees (left) and 50 input trees (right). The x-axis shows different data deletion rates within each source tree, see Section 3.2.4 for details.

of BCD - Unit Cost is similar to the evaluation on SMIDGenOG, we would also expect an comparable accuracy increase for BCD Branch Length and BCD Bootstrap.

Results on biological datasets

Accuracy. Next, we describe *SFN* rate and *SFP* rate results for all biological datasets, see Figures 5.9 and 5.10. Recall that low *SFN* and *SFP* rates do not necessarily correspond to a supertree of good quality, see the supplementary material for details. Bootstrap values and branch lengths are available for all supermatrix datasets (bees, saxifragales and legumes). Further, bootstrap values are available for the primates dataset, whereas branch length are available for the OMM dataset. All other supertree datasets contain neither bootstrap values nor branch length, prohibiting the use of BCD Bootstrap (BCD-BS) and BCD Branch Length (BCD-BL). By definition, the GSCM tree contains only splits that do not conflict with any source tree, which results in *SFP* rate = 0 for all datasets. For the biological datasets the GSCM tree is less resolved than for the simulated datasets; in addition, resolution varies strongly between biological datasets, see Figure 5.9. Consequently, we find that SuperFine trees are largely identical to MRP supertrees, and both methods show comparable performance. Again, BCD performs best when used in conjunction with GSCM and bootstrap values. BCD supertrees and CA-ML trees show higher *SFN* and *SFP* rates than SuperFine and MRP trees. The CA-MP tree of the legumes dataset has lower *SFP* rate than the SuperFine, MRP and BCD trees, but the highest *SFN* rate of all estimated trees. For the OMM dataset, the SuperFine GSCM calculation did not finish within 1 day, and the GSCM tree used by BCD did not contain a single clade; hence, no results are reported for these methods. Findings regarding the MRP-Score are qualitatively similar to those for *SFN* and *SFP* rates, see Figure A.23. For the supermatrix datasets, we evaluated parsimony and log-likelihood scores, see Figure 5.11. We find that MRP, SuperFine, and at least one variant of BCD have better parsimony scores than the CA-ML/-MP trees. This is despite the fact that for the legumes dataset, the

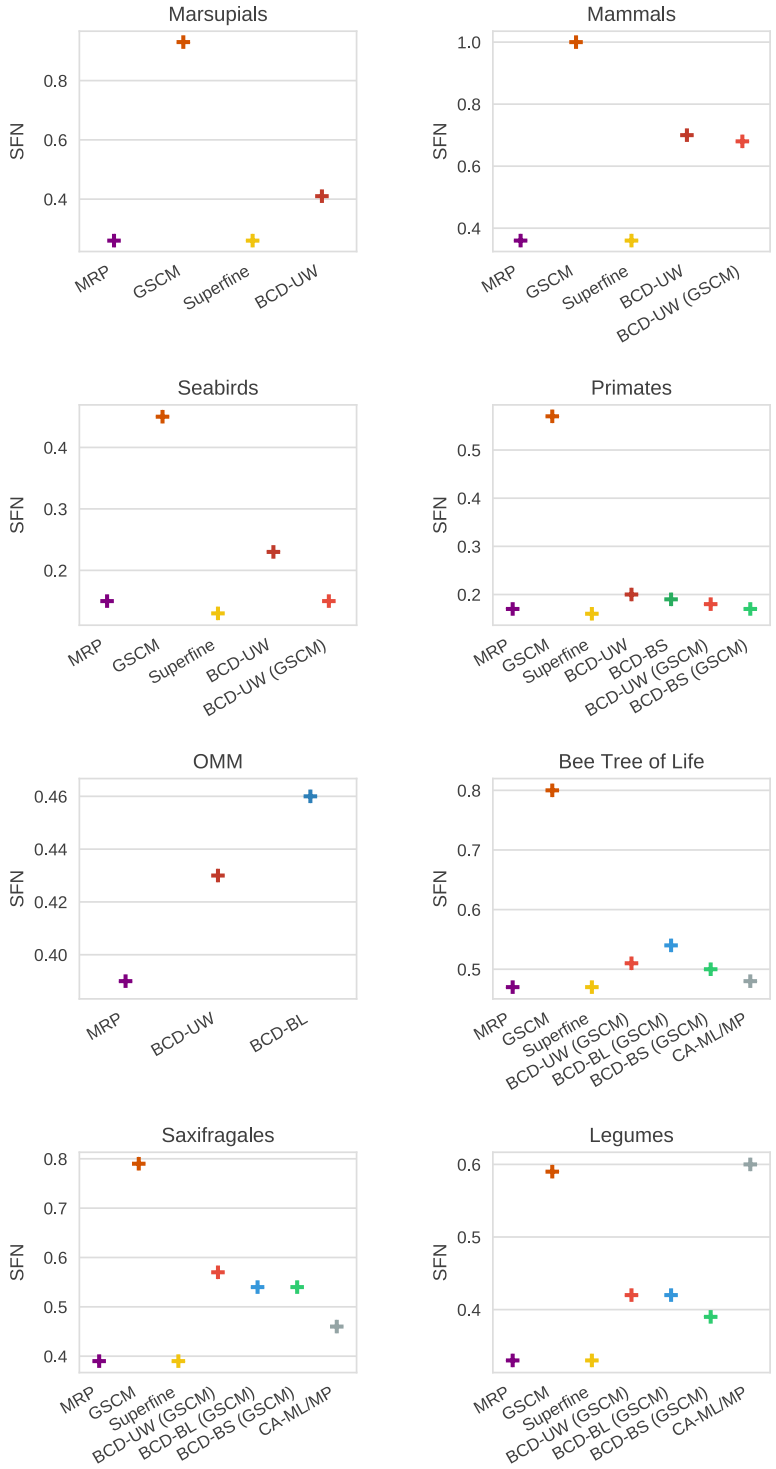


Figure 5.9: *Sum of false negative rates* (*SFN* rate) of supertrees against source trees for the biological supertree datasets. Most of the supertree datasets contain neither bootstrap values nor branch lengths, prohibiting the use of BCD bootstrap and BCD branch length. CA-ML/-MP cannot be applied to the supertree instances.

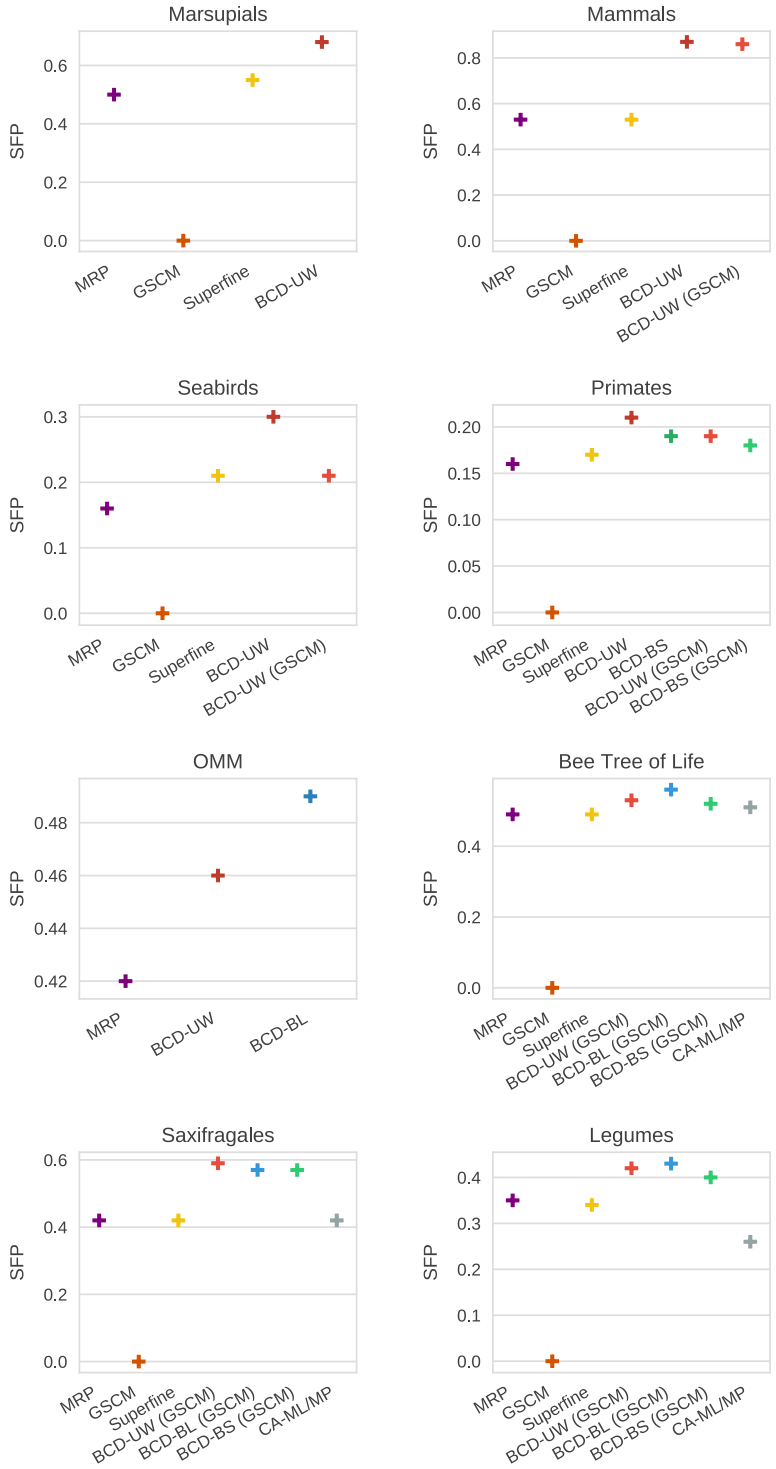


Figure 5.10: *Sum of false positive rates* (*SFP* rate) of supertrees against source trees on the biological supertree datasets. Most of the supertree datasets contain neither bootstrap values nor branch lengths, prohibiting the use of BCD bootstrap and BCD branch length. CA-ML/-MP cannot be applied to the supertree instances.

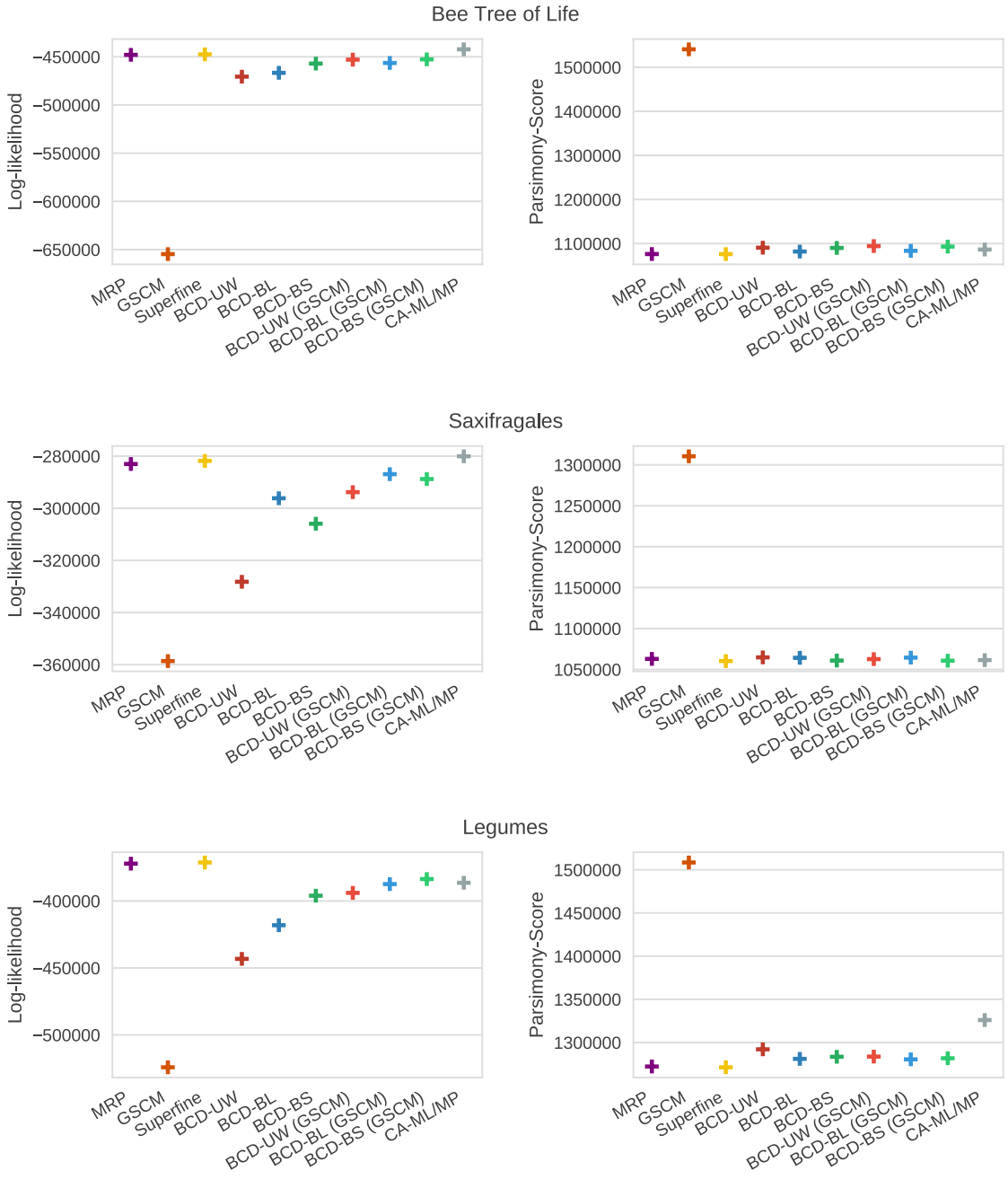


Figure 5.11: Comparison of GSCM, MRP, SuperFine, BCD and the Combine Analysis (CA-ML/MP) with regards to parsimony scores (top) and log-likelihood scores (bottom) for the biological supermatrix datasets.

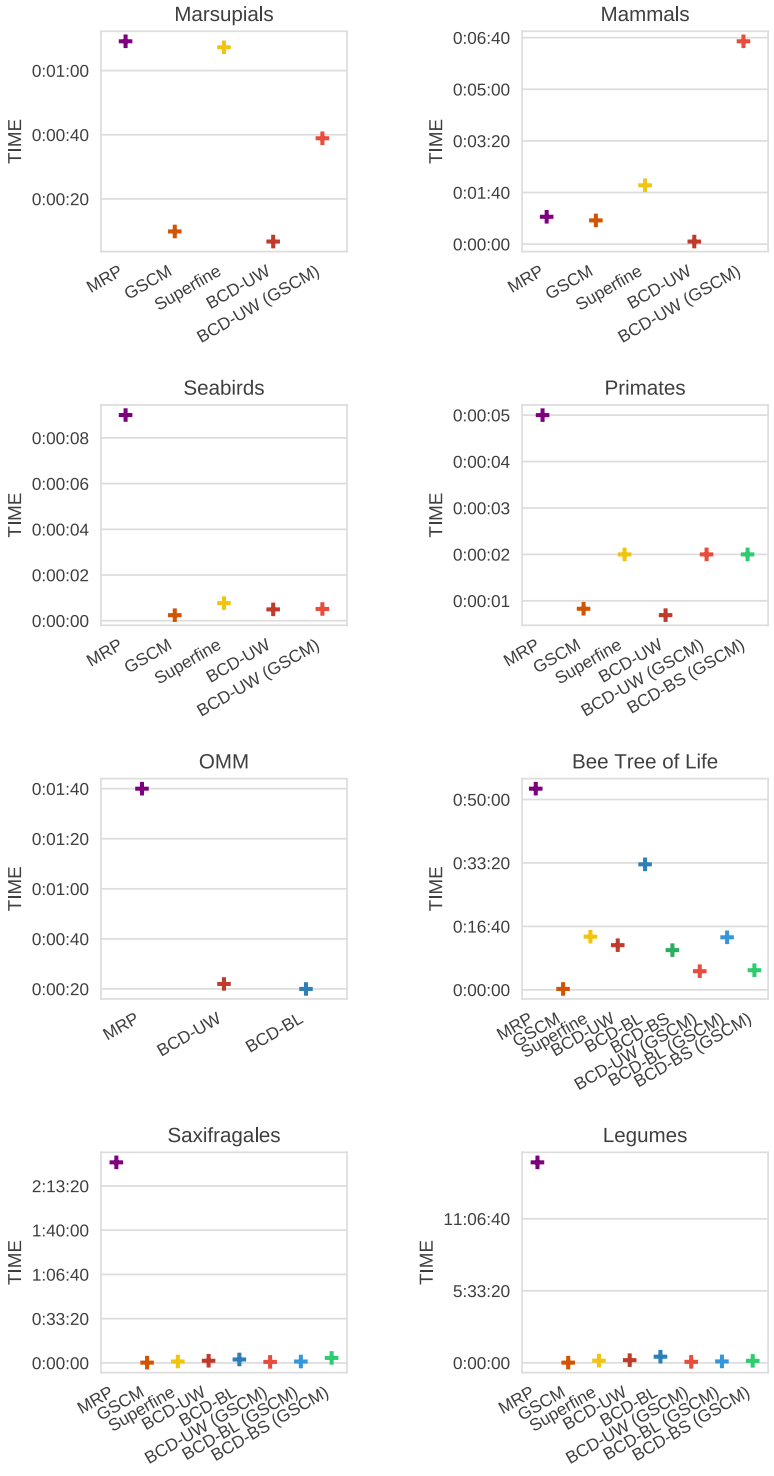


Figure 5.12: Running times (in hh:mm:ss) of GSCM, MRP, SuperFine and BCD for all evaluated biological datasets. Running times of SuperFine and BCD-GSCM include the time of the GSCM preprocessing step.

combined analysis tree has been computed using the parsimony optimization criterion. In all cases, SuperFine obtains the best parsimony scores. For the saxifragales dataset, BCD Unit Weight (BCD-UW) and BCD-BS have better parsimony scores than MRP; on the other datasets, BCD produces slightly worse parsimony scores than MRP. For saxifragales and bees, CA-ML obtains the best log-likelihood scores, followed by SuperFine, MRP, and BCD. For legumes, SuperFine obtains the best log-likelihood score, followed by MRP, BCD-BS, and CA-MP.

Running time. MRP is again the slowest supertree method by far (see Figure 5.12), where a variant of BCD is always the fastest. For data with many input trees compared to the number of taxa (primates, mammals, OMM, marsupials), using the GSCM tree does not speed up BCD and SuperFine.

5.3 Discussion

Our experiments with simulated data show that BCD can be an accurate and very fast supertree method. In particular, combining Bad Clade Deletion supertree computation with support values such as bootstrap values leads to excellent results, which are superior to MRP and SuperFine. Unlike SuperFine and MRP, BCD has guaranteed polynomial worst-case running time. Among three proposed weightings, we find that BCD supertree accuracy is usually highest when using bootstrap values, followed by branch lengths and unit weights. Even with unit weights, BCD can outperform Matrix Representation with Parsimony for simulated data. Using the Greedy Strict Consensus Merger as a preprocessing step turned out to be both robust and effective. We also evaluated the undisputed sibling reduction as a preprocessing method [21] but found that results of this combination were clearly dominated by BCD with GSCM. We also evaluated whether we can replace neighboring clades by a single joint clade during post processing [23, 81, 125] but found that the effect on supertree accuracy are negligible. The 5500 taxon dataset demonstrates the advantage of a polynomial-time supertree method: MRP did not finish after 14 days on this dataset, whereas BCD with branch lengths or bootstrap values never required more than a day of running time. Recall that SuperFine is identical to MRP when the GSCM tree is fully unresolved.

We also found that using an accurate supertree as starting tree for CA-ML can improve its accuracy and reduce the running time. Using CA-ML with a BCD starting tree produced the most accurate trees in our evaluation on simulated data, and also converges faster than CA-ML with the default starting tree.

For the biological datasets, assessing accuracy is more intricate, as we lack an optimality criterion that is known to correlate reliably with the structural supertree accuracy. Using branch lengths and bootstrap values does not improve the accuracy of BCD as much as for the simulated data. This may be attributed to the fact that the source trees in the biological datasets have much weaker bootstrap support than those in the simulated datasets: For example, 40% of the clades in the saxifragales dataset have bootstrap values below 50%. The BCD supertrees contains only clades that are supported by at least one source tree; consequently, the BCD algorithm may have to choose from a set of clades which are all wrong. Ignoring clades with low bootstrap support is easy but will produce less resolved results. The performance of SuperFine is similar to MRP for these datasets; this comes as no surprise, since the GSCM tree is often largely unresolved, and SuperFine optimization

is identical to the classical MRP optimization in these cases. Comparing the supertrees against the input trees (*SFN* and *SFP* rates), MRP and SuperFine show considerably better scores than BCD, but also than the CA-ML/MP, despite that the CA-ML trees may be assumed to be the most accurate trees. These results coincide with findings for simulated data evaluating *SFN* and *SFP* rates: There, MRP and SuperFine also showed superior *SFN* and *SFP* rates compared to BCD and CA-ML, even when *FN* and *FP* rates *compared to the model tree* were significantly worse. Therefore, we cannot safely conclude which method performs best for biological data.

6 BCD Beam Search - Considering suboptimal partial solutions in BCD Supertrees

In Chapter 5 we found that the greedy search strategy performed by BCD struggles to find a close-to-optimum solution on highly conflicting data when the meta information (e.g. bootstrap values or branch length) is unreliable or not available (see Section 5.3).

Here, we present a *beam search* approach for BCD, to consider not only the best but the k best solutions in every phase of the top-down construction of the supertree. We introduce and evaluate an exact and a randomized subroutine to calculate suboptimal solutions. All variants still have guaranteed polynomial running time. In our evaluation, we found that the beam search approaches consistently outperform the "classical" BCD algorithm.

6.1 Bad Clade Deletion Beam Search

The BCD algorithm tries to minimize a global objective function: Namely, the total weight of column deletions in the input matrix. Besides the theoretical amenity of this feature, this allows us to compare solutions based on the objective function. But in fact, we can extend this evaluation to partial solutions: At any point of the algorithm, we know the quality of a partial solution, that is, the total weight of clade deletions that were required up to this point. Clearly, this weight only increases during later steps of the algorithm. As mentioned in Chapter 5 the BCD algorithm proceeds in $n-1$ phases. An alternative view of the algorithm will be helpful in the following: BCD computes the supertree by iteratively refining a *partial* phylogenetic tree, which is a phylogenetic tree where several taxa can be mapped to the same leaf. Initially, we have a partial phylogenetic tree with a single node and all taxa attached to it. In each phase, the partial tree is refined by finding a bipartition of the taxa attached to one of the leaves. For the moment, we ignore the case that taxa are partitioned into $q > 2$ sets, corresponding to polytomies in the supertree, see below. Before phase p, the partial tree has p leaves, partitioning the taxa into p sets.

We will now extend the greedy BCD algorithm by keeping more than one partial solution "alive" in each phase, resulting in a beam search algorithm. The parameter $k \geq 1$ determines the number of partial solutions that are considered simultaneously; for $k = 1$ this is equivalent to the original BCD algorithm. Formally, a *partial solution* $P = (\mathcal{M}, T, \mathit{cost})$ of *order* p consists of

1. a set $\mathcal{M} = \{(S_1, D_1), \ldots, (S_p, D_p)\}$ such that $S_1, \ldots, S_p$ is a partition of the taxa $S = \{1, \ldots, n\}$, and $D_1, \ldots, D_p$ is a partition of the clade vertices $D = \{1, \ldots, m\}$;

2. a partial phylogenetic tree T with p leaves, labeled by $S_1, \ldots, S_p$; and

3. a real number *cost*, the cost for matrix modifications up to this point.

Before phase p of the beam search, we have a set $\mathcal{P}$ of $|\mathcal{P}| = k$ partial solutions of order p. We transform this into a set $\mathcal{P}'$ of $|\mathcal{P}'| = k$ partial solutions of order $p+1$. In the first step of the algorithm, we start with a single partial solution with cost zero.

Now, we describe how to transform a partial solution $P = (\mathcal{M}, T, \mathit{cost})$ of order p, into k new partial solution of order $p+1$. For each of the p graphs $G(S_i, D_i)$ for $i = 1, \ldots, p$, we compute the k best bipartitions. Out of the resulting pk cuts, we extract the best k cuts in any of the graphs. We iterate over these cuts: Assume that the cut happens in the graph $G(S, D)$ for $(S, D) \in \mathcal{M}$. By this cut, both the taxon set S and the clade vertex set D are bipartitioned into sets $S', S'' := S \setminus S'$, and $D', D'' := D \setminus D'$. We build a new partial solution $P' = (\mathcal{M}', T', \mathit{cost}')$ as follows:

1. Set $\mathcal{M}' := \mathcal{M} \setminus \{(S, D)\} \cup \{(S', D'), (S'', D'')\}$;

2. resolve the node in T labeled S by two nodes S', S'' in T';

3. compute the new costs $\mathit{cost}' := \mathit{cost} + \mathit{cut}$ where cut are the costs of the cut in $G(S, D)$.

We now evaluate the partial solutions that belong to the same phase p, based on the costs generated so far: In each phase, we do our computations for each of the k partial solutions $P \in \mathcal{P}$. For each partial solution P, we compute k cuts instead of a single one, resulting in k^2 partial solutions. We then keep only the best k partial solutions in phase $p+1$, each of which is used for computation of cuts in the next phase of the algorithm. See Algorithm 3 for a pseudo code of the algorithm described up to this point.

There is another pitfall we have to consider: In the original algorithm, the order in which we processed the leaves of a partial solution was of no importance, as we eventually had to resolve each leaf. For the beam search, this is no longer the case, as we search for the k best partial solutions. To this end, the algorithm described above computes, for each partial solution in phase p, k cuts in each of the p graphs $G(S_i, D_i)$ for $i = 1, \ldots, p$. This would result in an additional $O(n)$ factor in the total running time. But this is in fact not a problem: In each phase, we record all k cuts for each of the p graphs $G(S_i, D_i)$. For each new partial solution of order $p+1$, only *one* cut in some graph $G(S, D)$ is chosen, whereas all other graphs will reappear unchanged in the next phase. Hence, in the next phase, only two graphs have to be searched for k best cuts: namely, the graphs $G(S', D')$ and $G(S'', D'')$. For all other graphs, we already know the k best cuts from the previous phase.

Finally, let us consider the case that a minimum cut results in more than two connected components. This simply means that the resulting partial solution is of higher order than $p+1$. When the algorithm is in phase p it will ignore all partial solutions of order above p; in fact, if there exist k' partial solution of order above p then we only have to compute $k - k'$ instead of k cuts. The same holds true if $G(S, D)$ does not require cutting, because it already consists of two or more connected components.

It is understood that BCD Beam Search can be applied for any type of vertex weighting in $G(S, D)$, including Bootstrap weights, Branch Length weights, and Unit weights.

Algorithm 3 BCD Beam Search

```
1: function bcdBeamSearch(G(S, D),k)
2:     #Taxa ← |S|
3:     P ← initPartialSolutions(G(S, D))
4:     while ∃(M, T, cost) ∈ P : |M| ≤ #Taxa do
5:         P' ← {}
6:         for all (M, T, cost) ∈ P do
7:             for all (S, D) ∈ M do
8:                 Cuts ← calcuateCuts(G(S, D), k)        ▷ Cuts may already have been calculated in a previous iteration.
9:                 for all ((S', D'), cutCost) ∈ Cuts do
10:                    S'' ← S \ S', D'' ← D \ D'
11:                    M' ← M \ {(S, D)} ∪ {(S', D'), (S'', D'')}
12:                    T' ← T where S is resolved by two nodes S', S''
13:                    cost' ← cost + cutCost
14:                    P' ← P' ∪ {(M', T', cost')}
15:                end for
16:            end for
17:        end for
18:        P ← P' reduced to the best k partial solutions
19:    end while
20:    return T_1, ..., T_k from P
21: end function
```

```
1: function initPartialSolutions(G(S, D))
2:     M ← {(S, D)}
3:     T ← tree with a single root S
4:     return {({(S, D)}, T, 0)}
5: end function
```

```
1: function calcuateCuts(G(S, D),k)
2:     if G(S, D) is disconnected then
3:         return {((S', D'), 0)}
4:     else
5:         return {((S'_1, D'_1), c_1), ((S'_2, D'_2), c_2), ..., ((S'_k, D'_k), c_k)}  ▷ If cuts do not exist, calculate them with cut enumeration (Section 6.2.1) or cut sampling (Section 6.2.2).
6:     end if
7: end function
```

6.2 Searching Suboptimal Vertex-cuts in the Pe'er Graph

For the beam search algorithm described above, we have to compute k different vertex-cuts instead of a single minimum vertex-cut in the given graph $G(S, D)$. This is achieved by computing suboptimal cuts in the network graph $H(S, D')$. In the following, we present two strategies for doing so; namely, suboptimal cut enumeration and random cut sampling (CS).

6.2.1 Cut Enumeration

Vazirani and Yannakakis [179] introduced an algorithm to enumerate the cuts of a network with $|V|$ vertices and $|A|$ arcs in the order of non-decreasing weights, using $O(|V|)$ maximum flow computations between two successive outputs. Let $H(S, D')$ be a bipartite network with $n = |S|$ taxon vertices, $m = |D'|$ clade vertices and $n \leq m$. The network contains $|V| = n+m$ vertices and $|A| = O(nm)$ arcs. A maximum flow of $H(S, D')$ can be calculated in $O(n^2m)$ time by using the bipush variant of the preflow algorithm with dynamic trees, as described in Ahuja *et al.* [2]. Since we are only interested in enumerating cuts that separate the taxon set S, we do only need n instead of $n + m$ max flow computations [53], which leads to a running time of $O(n^3m)$ per cut. We have to calculate at most k^2 cuts in each of the $O(n)$ partition steps of the BCD algorithm, which leads us to the following lemma:

Lemma 4. *Given an input matrix M over $\{0, 1, ?\}$ for n taxa and m clades and an integer $k \geq 1$, the Bad Clade Deletion beam search algorithm computes a supertree in $O(k^2n^4m)$ when using the Vazirani and Yannakakis [179] cut enumeration algorithm.*

Hao and Orlin [67] found that the minimum cut of a network can be computed in $O(|V||A| \log(2 + \frac{|V|^2}{|A|}))$. Yeh *et al.* [193] presented an improved cut enumeration algorithm that requires one maximum flow and two invocations of Hao and Orlins minimum cut algorithm between two successive outputs. Further, Hao and Orlin [67] showed that their algorithm calculates the minimum cut of a bipartite network with $|S| \leq |D'|$ in $O(|S||A| \log(2 + \frac{|S|^2}{|A|}))$. For our network $H(S, D')$ this leads to $O(n^2m)$ to calculate a minimum cut.

Lemma 5. *Given an input matrix M over $\{0, 1, ?\}$ for n taxa and m clades and an integer $k \geq 1$, the Bad Clade Deletion beam search algorithm computes a supertree in $O(k^2n^3m)$ when using the Yeh* et al. *[193] cut enumeration algorithm.*

Algorithm engineering. We note that in practice, we usually have to compute much fewer cuts than the k^2 mentioned above: We start by computing k cuts for the best partial solution; this gives us an *upper bound* for the k-th best cost in the active phase. Now, when we consider the second-best partial solution, we can stop as soon as the computed cost exceeds the upper bound; and we can update the upper bound in case we find partial solutions that belong to the top k.

6.2.2 Cut Sampling

The cut sampling algorithm is inspired by the randomized algorithm of Karger and Stein [84] for finding all minimum cuts in an undirected graph $U(S, E)$ with a certain probability. The algorithm recursively contracts edges of the graph and merges the connected vertices into sets, until only two vertex sets are left. Edges are randomly drawn with probability

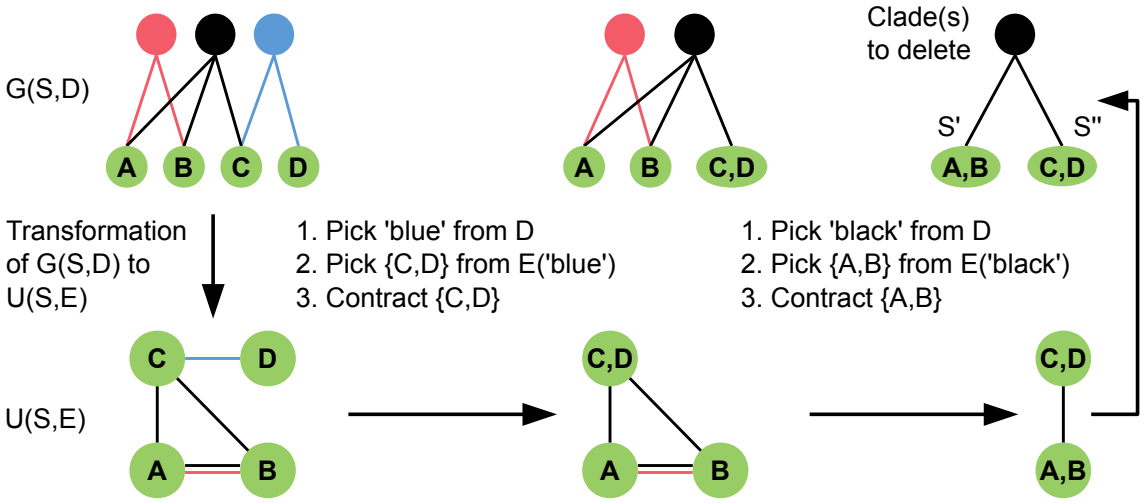


Figure 6.1: Workflow of the cut sampling algorithm (see Section 6.2.1) including the transformation from $G(S, D)$ to $U(S, E)$ and the two-step edge selection process we use in the recursive contraction algorithm. Edges $E(c) \subseteq E$ are those edges with color (clade-vertex) c.

proportional to their edge weight. This *contraction* algorithm runs in $O(|S|^2)$ time and has $\Omega(1/|S|^2)$ probability of outputting a minimum cut. The algorithm requires $O(|S|^2 \log |S|)$ iterations to find a minimum cut in one of the trials with high probability (probability converging to 1). Karger and Stein further presented a *recursive contraction* algorithm, where the trials share their work, so that each of the trials can be executed in $O(1)$ time. The recursive contraction algorithm runs in $O(|S|^2 \log |S|)$ time, and outputs a minimum cut with probability $\Omega(1/\log |S|)$. After $O(\log^2 |S|)$ iterations, the probability to find a minimum cut converges to 1, which results in an total running time of $O(|S|^2 \log^3 |S|)$.

Each of the trials returns a (potentially suboptimal) cut, and cuts with lower costs are sampled with higher probability than cuts with higher costs. To this end, the algorithm can also be used to generate a large number of suboptimal cuts without additional overhead: In detail, we can calculate $O(|S|^2 \log^2 |S|)$ cuts in $O(|S|^2 \log^3 |S|)$ time.

To apply the contraction idea here, we transform the bipartite graph $G(S, D)$ into a simple undirected graph $U = (S, E)$ with vertex set S. We insert edges to E so that for each $c \in D$, all $t \in S$ adjacent to c form a clique in U. Let $E(c) \subseteq E$ be the set of edges of the clique induced by c. Given the graph U, we choose the edge $e \in E$ to contract in a two-step approach. First, we randomly pick a clade vertex $c \in D$ with probability proportional to its weight. Second, we draw an edge $e \in E(c)$ equally distributed, which is then contracted as described by Karger and Stein [84]. We contract edges until only two sets of vertices are left. These two vertex sets are a bipartition S', S'' of our taxon set S, and the edges $e \in E$ connecting S' and S'' correspond to the clade vertices we need to delete in $G(S, D)$ to induce this bipartition of S. See Figure 6.1 for an exemplary workflow of the algorithm.

The above algorithm allows us to sample low-weight cuts with higher probability than high-weight cuts. When selecting the edges as described above, there exists for each cut in $U(S, E)$ a vertex-cut in $G(S, D)$ with the same partition and identical weight. With

our modified edge selection process, we ensure that the probability of a clade vertex to be chosen in each contraction step is proportional to its weight. But clade vertices with higher degree are more likely to be deleted than clade vertices with lower degree and same weight. Therefore, the above algorithm has *no guarantee* to find a minimum cut with a certain probability, and we cannot guarantee that a minimum cut will be part of the output. But we can calculate a minimum cut in $O(|D|\,|S|^2)$ time using the maximum flow approach, and add it to the list of cuts.

The two-step approach needs $O(|D| \cdot |S|)$ time to choose and contract an edge, and $O(|S|)$ contractions are needed to produce a cut. Using the recursive contraction algorithm, we need $O(|D|\,|S|^2 \log^3 |S|)$ time to calculate $O(|S|^2 \log^2 |S|)$ cuts. If we assume $k \in O(|S|^2 \log^2 |S|)$, which is realistic in practice, this leads us to the following lemma:

Lemma 6. *Given an input matrix M over $\{0, 1, ?\}$ for n taxa and m clades and an integer $k \geq 1$ with $k \in O(n^2 \log^2 n)$, the Bad Clade Deletion Beam Search algorithm using cuts sampling computes a supertree in $O(kmn^3 \log^3 n)$ time.*

Algorithm engineering. Again, we can do some algorithm engineering to improve running times in practice: We start by sampling cuts for the best partial solution; this again gives us an *upper bound* for the k-th best cost in the active phase. Now, we check if the score of the second-best partial solution plus the weight of an optimal cut (computed using the max flow approach) exceeds the upper bound; in this case, no sampling is required for this partial solution. We note that running time improvements by this trick are presumably smaller than for cut enumeration, as we have to run the full sampling process if the optimal cut does not result in a violation of the upper bound.

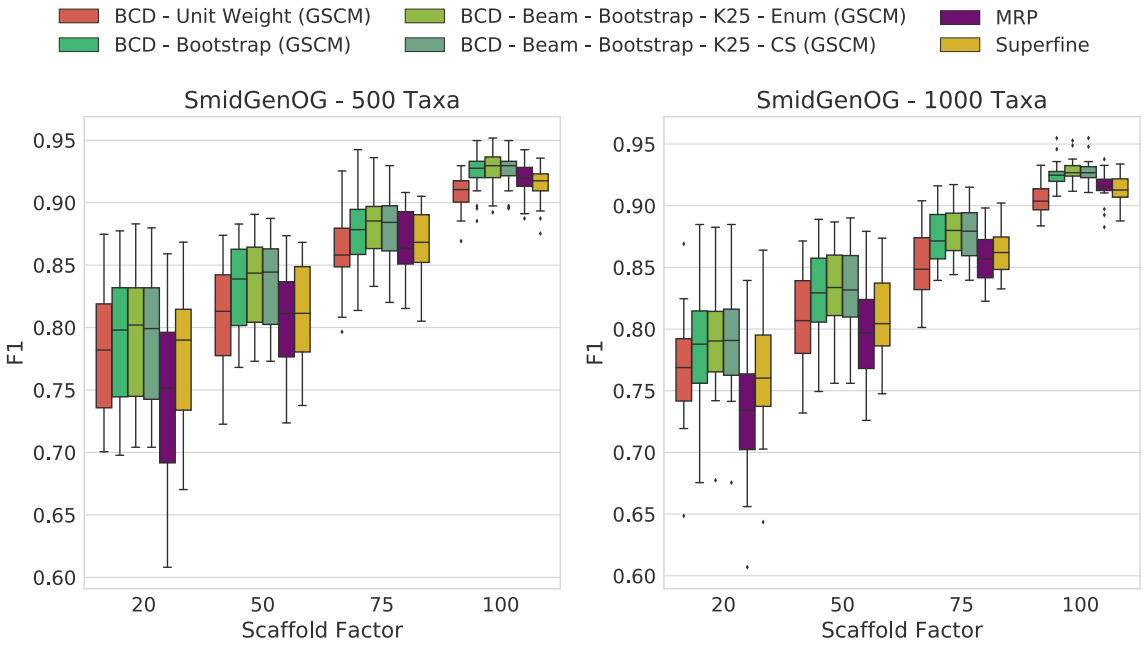
6.3 Results

We now describe results for different BCD Beam Search variants in comparison to BCD, MRP and SuperFine on simulated and biological data.

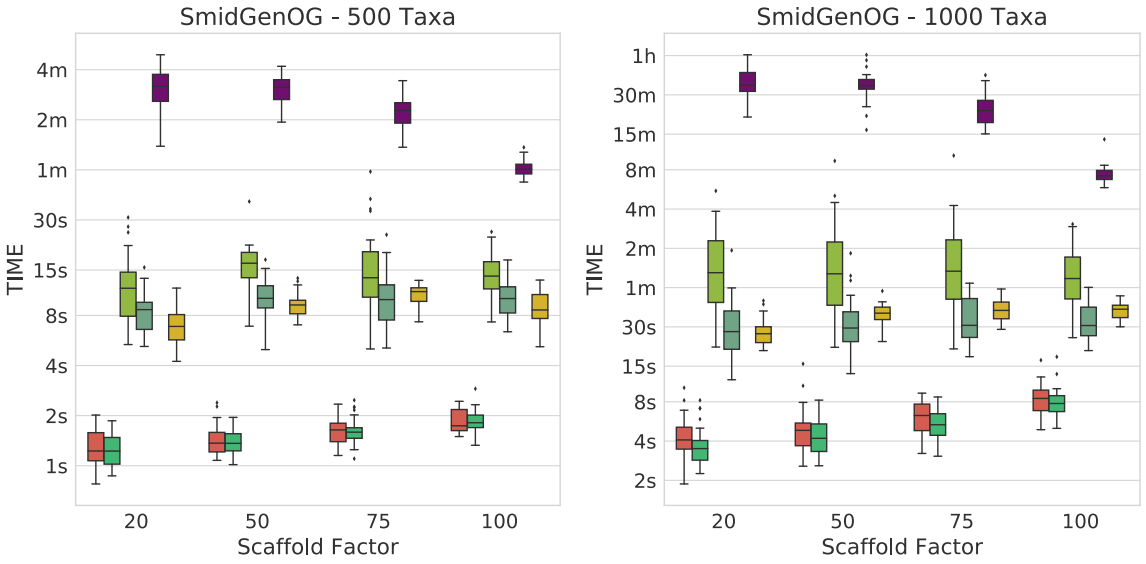
SMIDGenOG Results

The SMIDGenOG dataset contains bootstrap values; to this end, we can evaluate BCD and BCD Beam Search using the Bootstrap weighting.

Accuracy. On this dataset, BCD with unit weights is already on par with MRP and SuperFine; BCD with bootstrap weights outperforms MRP and SuperFine. We ran BCD Beam Search with $k = 25$ partial solutions. We find that BCD Beam Search with bootstrap weights consistently outperforms any other evaluated methods with respect to F_1-score (see Figure 6.2a); this is true for both the 500 and 1000 taxa dataset. On the 500 taxa dataset with 120 instances, BCD Beam Search with cut enumeration outperforms BCD on 83 instance (19 ties), SuperFine on 111 instances (1 tie) and MRP on 113 instances (1 tie). When using cut sampling it outperforms BCD on 68 instances (32 ties), SuperFine on 111 instances (1 tie) and MRP on 113 instances. On the 1000 taxa dataset with 120 instances, BCD Beam Search outperforms BCD on 100 instance (7 ties) when using cut enumeration and on 94 instances (17 ties) when using cut sampling. Both BCD Beam Search variants outperform MRP on 119 and SuperFine on all 120 instances.



(a) F_1-scores



(b) Running times

Figure 6.2: F_1-scores (a) and running times (b) of SuperFine, MRP, BCD and BCD Beam Search variants on the simulated SMIDGenOG dataset. Results for the 500 taxa dataset (left) and the 1000 taxa dataset (right). The x-axis shows different scaffold factors in percent.

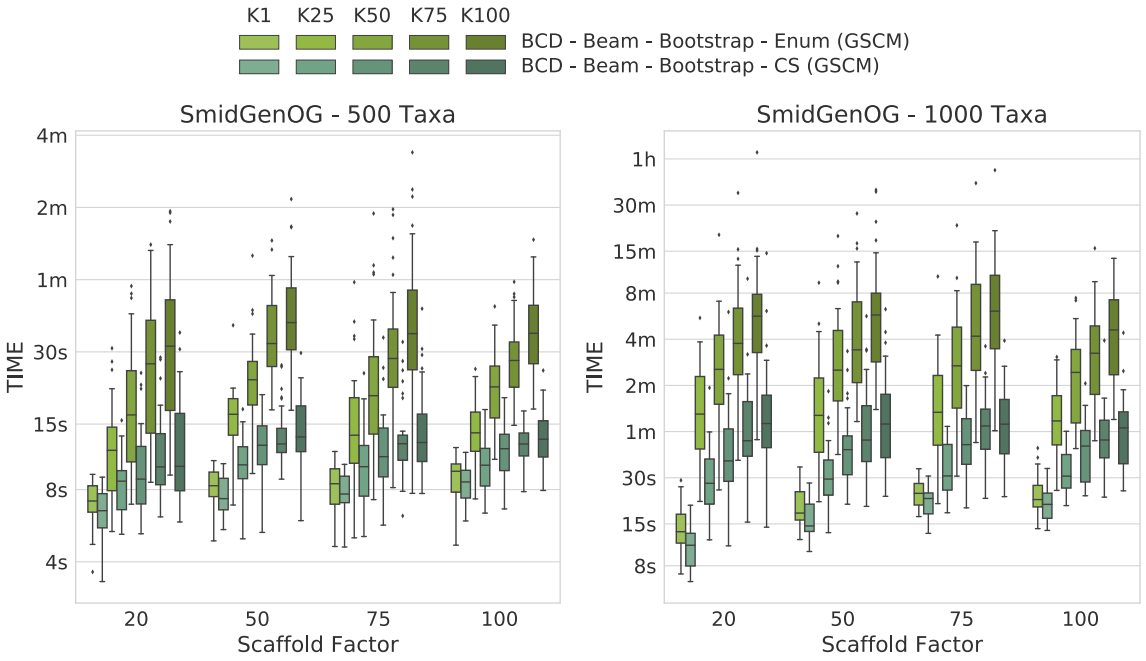


Figure 6.3: Running times of BCD Beam Search with different numbers of suboptimal solutions ($k = 1$, $k = 25$, $k = 50$, $k = 75$ and $k = 100$) on the simulated SMIDGenOG dataset. Results for the 500 taxa dataset (left) and the 1000 taxa dataset (right). The x-axis shows different scaffold factors in percent.

Running time. We report running times for the SMIDGenOG dataset in Figure 6.2b. We see that BCD Beam Search with cut enumeration is about two- (500 taxa) to three-fold (1000 taxa) slower than SuperFine. Whereas BCD Beam Search with cut sampling is slightly slower than SuperFine on the 500 taxa dataset, it is already slightly faster on the 1000 taxa dataset. As expected, the beam search is slower than the regular BCD algorithm; namely up to 20-fold slower for cut enumeration and five-fold slower for cut sampling. But notably, it is on average 10-/15-fold faster than MRP (500/1000 taxa) for cut enumeration and 15-/46-fold faster than MRP (500/1000 taxa) for cut sampling. For the 1000 taxa dataset, the average running time of BCD is less than 6 s. BCD Beam Search with cut enumeration needs less than 2 min; BCD Beam Search with cut sampling and SuperFine need less than 1 min; and MRP needs about 27 min.

The number of suboptimal solutions (k) shows a quadratic impact on the running time for the beam search with cut enumeration, whereas the running time increases only linear for beam search with cut sampling (see Figure 6.3). Further, we found that even for $k = 100$ the beam search with cut sampling is still less than 2-fold slower than SuperFine and still clearly faster than MRP. With $k = 100$ the beam search with cut enumeration is always faster than MRP.

SuperTriplets Benchmark Results

This dataset does not contain bootstrap values or branch lengths; to this end, BCD has to be run with Unit Weights (UW) and, hence, showed suboptimal performance in our previous evaluations (see Figure 5.8). We ran BCD Beam Search with $k = 25$ and $k = 50$ partial solutions (see Figure 6.4).

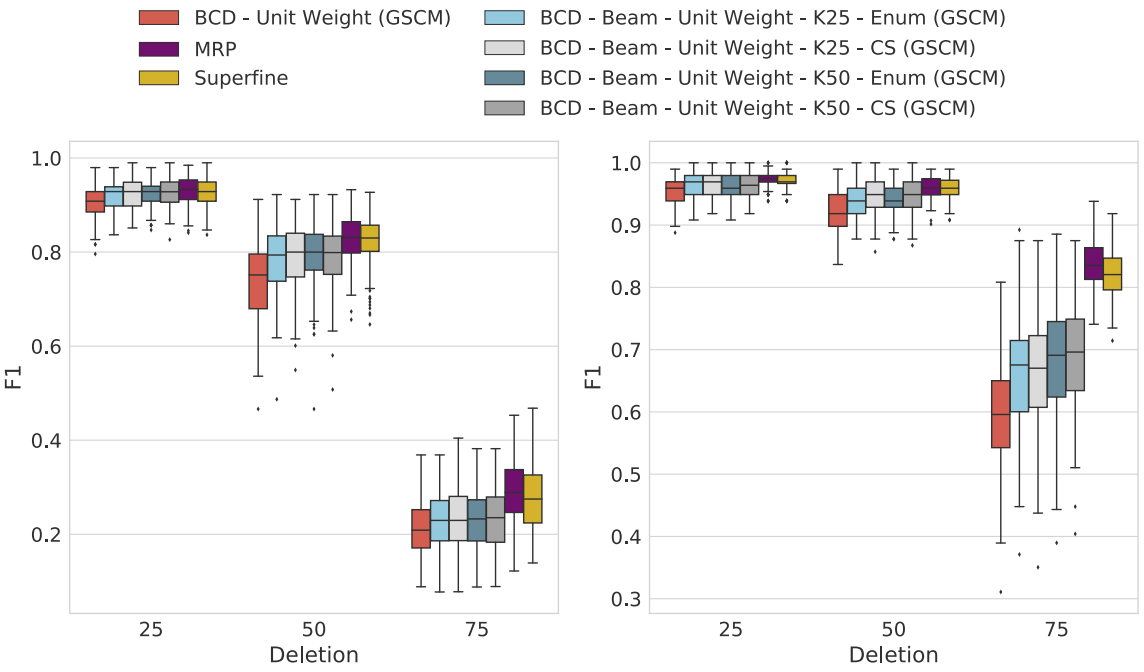


Figure 6.4: F_1-score of SuperFine, MRP, BCD and BCD Beam Search variants on the simulated SuperTriplets Benchmark. Results for 10 input trees (left) and 50 input trees (right). The x-axis shows different data deletion rates within each source tree.

Accuracy. It is still the case that BCD supertrees are generally of lower quality than MRP and SuperFine supertrees: For 10 input trees, BCD Beam Search performs on par with MRP for deletion rate 25%; for 50 input trees, it is on par with MRP for deletion rates 25% and 50%. For the remaining configuration it performs worse than MRP; this is particularly the case for 50 input trees and 75% taxa deletion. But we observe that BCD Beam Search reaches a significantly higher F_1-score than BCD, for all numbers of input trees and taxa deletion rates. In contrast, we do not observe a significant increase of F_1-score when considering $k = 50$ instead of $k = 25$ partial solutions.

BCD Beam Search with cut enumeration and $k = 25$ partial solutions produced a supertree with higher F_1-score than BCD without beam search for 1212 of 1500 replicates; of the remaining, 144 are ties. This is very similar for BCD Beam Search with cut sampling, with 1191 wins and 146 ties. We stress that many replicates resulted in supertrees with identical BCD Score, indicating the combinatorial complexity of this dataset.

Results for Biological data.

BCD Beam Search shows better (lower) *SFN* rates and *SFP* rates on all biological datasets than the original BCD, see Table 6.1: We find that best *SFN* rates and *SFP* rates are distributed between MRP, SuperFine, and BCD Beam Search, but no method constantly outperforms the others over all datasets. This is remarkable, as most of the datasets does not contain bootstrap values or branch lengths, and BCD Beam Search with Unit Weights had to be applied. Recall, the bats data allow for a perfect phylogeny. The results for this data show that BCD finds a perfect phylogeny if one exists whereas MRP may not. For the supermatrix datasets BCD Beam Search reaches still constantly lower *SFN* rates

Table 6.1: *Sum of false negative rates* and the *sum of false positive rates* of supertrees against source trees on biological datasets. Most datasets do not contain bootstrap values, prohibiting the use of Bootstrap weights. Best rates in each column are marked in bold.

	SFN rate / *SFP* rate				
Dataset	Bees	Saxafragales	Legumes	Primates	
MRP	**0.471**/**0.494**	0.393/0.421	0.335/0.346	0.169/**0.165**	
SuperFine	**0.471**/**0.494**	**0.388**/**0.417**	**0.331**/**0.337**	**0.165**/0.172	
BCD UW	0.504/0.526	0.536/0.562	0.394/0.398	0.178/0.185	
BCD BS	0.506/0.523	0.523/0.548	0.393/0.398	0.174/0.180	
BCD-K25-Enum UW	0.495/0.513	0.523/0.537	0.400/0.393	0.172/0.178	
BCD-K25-Enum BS	0.495/0.506	0.512/0.531	0.376/0.371	**0.165**/0.169	
BCD-K25-CS UW	0.498/0.514	0.530/0.548	0.400/0.400	0.176/0.183	
BCD-K25-CS BS	0.507/0.52	0.537/0.557	0.376/0.371	0.169/0.176	
	SFN rate / *SFP* rate				
Dataset	Seabirds	THPL	Mammals	OMM	Bats
MRP	0.153/**0.159**	**0.190**/0.328	0.361/**0.532**	**0.386**/0.425	0.063/0.015
SuperFine	0.127/0.206	n/a	**0.358**/0.534	n/a	n/a
BCD UW	0.153/0.206	0.454/0.523	0.679/0.855	0.425/0.457	**0**/**0**
BCD-K25-Enum UW	**0.122**/0.175	0.239/**0.320**	0.633/0.808	**0.386**/**0.417**	**0**/**0**
BCD-K25-CS UW	**0.122**/0.175	0.250/0.334	0.682/0.858	0.388/0.420	**0**/**0**

and *SFN* rates but still improves BCD. As already shown in Section 5.2 the bootstrap values do not improve results for the datasets (bees and saxafragales) where the bootstrap support is low in general, whereas they work as expected for the legumes dataset with better bootstrap support. Due to the stochastic nature of the cut sampling procedure, we see that BCD Beam Search results differ between cut enumeration and cut sampling; the cut enumeration approach tends to be more robust, as expected.

We do not have results for SuperFine on OMM because the GSCM did not finish in reasonable time. Since the GSCM tree of the OMM data does not contain a single clade (calculated with BCD), SuperFine results are identical to MRP anyways. For Bats and THPL, SuperFine did not return a result due to too less overlap of the input trees in the unrooted case.

6.4 Discussion

We presented a beam search algorithm that allows the BCD algorithm to consider the k best partial solutions instead of only the optimal one, when partitioning the taxon set in a top-down manner. BCD Beam Search has still guaranteed polynomial running time. We introduced an algorithm to enumerate suboptimal solutions in non-decreasing order, and a second algorithm to sample good partial solutions. Our evaluations on simulated and biological data showed that both beam search approaches consistently improve BCD on all evaluated datasets for $k \geq 25$. Both methods for computing suboptimal cuts perform roughly on par, but the enumeration algorithm tends to be more robust. However, the sampling algorithm scales linearly with the number of suboptimal solutions to be considered, whereas the exact enumeration algorithm scales quadratically in the worst case. We further found that BCD Beam Search, especially when used together with Bootstrap weights, is on

par with MRP and SuperFine for most of the biological datasets with regards to supertree quality.

7 Conclusion

Accurate solutions for almost every task in phylogenetics result in NP-hard optimization problems, so that non deterministic search heuristics are widely used. These methods are highly accurate and fast enough as long as the data becomes not too large. Whereas increasing sequence length can be handled fairly well by parallelization, many taxa are more challenging, due to the problem that the accuracy of the heuristic approaches decreases for a large number taxa.

In Chapter 2, we discussed the current state in phylogenetic tree reconstruction. We discussed how the many different approaches work together and how they can be combined to overcome limitations in scalability and accuracy. It turned out that supertree methods are a key technology for divide-and-conquer approaches that can make highly accurate but computational intense methods applicable for data with thousands of taxa. In divide-and-conquer strategies, supertrees methods are used to combine subsets of the data where phylogenetic trees were previously calculated by a more accurate base method. In this context, the supertree method has to be able to handle the complete dataset in reasonable running time; otherwise, it would be the limiting factor at some point.

In this thesis, we presented polynomial time approaches for the supertree problem that can be more accurate than local search heuristics that are usually used. But measuring accuracy is not trivial and should be done on different independent datasets. In Chapter 3, we described our evaluation setup. We used multiple simulated datasets created by different protocols. We developed a rooted version of the SMIDGen dataset and in particular modified the protocol to be more feasible for a large scale dataset. Simulated data is very imported, hence it is the only way to compare against a known truth. However, simulated data reflects always what we think how the biological data were created and, hence, methods often work as expected on these datasets. For that reason, it is important to check if results can be validated on biological data. Thus, we also used ten different biological datasets for evaluation. Three of them are supermatrix studies which we split apart to create source trees for a supertree analysis. This allowed us to compare our results on biological data against the sequence data instead of against the source trees. All this data is available[1] [2] and can be used for future evaluations.

In Chapter 4, we described improved scoring functions and randomized approaches to collect reliable clades using the Greedy Strict Consensus Merger. With the "Unique-Clades-Lost" scoring, we found a scoring function that increases the number of reliable clades by simultaneously reducing collision induced bogus clades. We further found that combining multiple (e.g. randomized) GSCM trees improves the ratio of reliable clades per bogus clade. However, from the point of view to use these clades to restrict the search space for a subsequent supertree method, the "unique clades lost" is, due to its low *FP* rate, the most feasible option.

[1] `https://bio.informatik.uni-jena.de/data/`

[2] `https://figshare.com/articles/BCD_Beam_Search_evaluation_data/6189113`

The main result of this thesis is Bad Clade Deletion, a polynomial time algorithm for the supertree problem that is as accurate as the most accurate supertree methods when using support values from the input trees. It is a greedy heuristic to minimize the number of clades that have to be deleted from the source trees to make them compatible. BCD can use the GSCM to restrict its search space. With unit weights BCD shows accuracy on par with MRP for data with a moderate proportion of conflicts in the input. A strength of this algorithm is that it can easily take advantage of support values (e.g. bootstrap values) without an increase in complexity. When using bootstrap values, BCD can be even more accurate than MRP and SuperFine.

We created the SMIDGenOG 5500 dataset with up to 62495 characters per replicate to show that BCD easily handles very large datasets. BCD finished the computation in less than 8 hours on average, whereas MRP could not finish after 2 weeks on any replicate. Furthermore, BCD was significantly more accurate than any other evaluated method on this large dataset. We also evaluated if a supertree can boost maximum likelihood estimation when used as a starting tree. Here, we found that BCD with support values was the only supertree method that significantly improved the accuracy of an ML analysis. The combination of a BCD starting tree and maximum likelihood analysis produced the most accurate trees on the SMIDGenOG dataset.

Our evaluations showed that BCD (especially without reliable support values) loses accuracy when the input data is highly conflicting. For such data, we developed a beam search approach that allows to take suboptimal partial solution into account to find a better overall result. We developed two different strategies to compute suboptimal solutions: The cut enumeration algorithm enumerates suboptimal solution with non-decreasing BCD score and scales quadratically with the number of suboptimal solutions to be considered. The cut sampling algorithm randomly samples solutions of good quality and scales linearly with the number of suboptimal solutions to be considered. Both variants of the Beam Search improved the accuracy of BCD on *every* dataset for already 25 partial solutions. The improved accuracy comes to the cost of an increased running time, but it is still polynomial.

The BCD algorithm has, besides its running time, some promising characteristics for the use in divide-and-conquer approaches. It guarantees to return the true tree if the input trees are induced subtrees of the true tree. Minimizing clade deletions does never create clades that are not supported by at least one source tree. Hence, we will not introduce errors during the supertree step that were not already in the source trees. BCD can be efficiently parallelized up to the number of taxa, which makes it even more scalable for large datasets. Efficient thread-based parallelization is already implemented for the BCD algorithm and has to be done for the Beam Search. It is worth to be noted that all runtime measurements have been done in single threaded mode for better comparability, so that even faster running times can be expected when using BCD in a multi core setup.

With BCD supertrees we presented an ultra-fast, very accurate and highly scalable supertree approach that can easily take advantage of support values. Our evaluations showed that BCD can improve phylogenetic tree reconstruction in many different scenarios.

Future Work

As mentioned in Lemma 5, the running time of the cut-enumeration can be improved when using the Yeh *et al.* [193] cut enumeration algorithm.

We argued that support values can be very useful for supertree estimation. When using Bayesian MCMC we get a distribution of trees and thus also a distribution of clades that can be used as support values, but Yang and Zhu [192] showed recently that posterior probabilities can be spuriously high on short internal ranches. Therefore, scoring function that takes posterior probabilities and branch length into account has to be developed for this data. Further, it is conceivable to include suboptimal clades from the source tree sampling process in the supertree analysis with BCD. The challenge here is to find a good trade-off between additional information and too much conflicting data for the BCD supertree reconstruction. Evaluations with whole "suboptimal" tree topologies from Bayesian MCMC runs showed that this approach adds too many conflicting information to the source trees [180].

The basis for BCD is that the rooted incomplete perfect phylogeny problem can be solved in polynomial time, whereas this task is NP-hard for the uprooted case. Unfortunately, trees reconstructed from sequence data are usually unrooted. On the other hand, biologists are usually interested in rooted trees and will therefore add an outgroup to their data. We already discussed that finding an outgroup is not an easy task. But assuming we have an appropriate outgroup for our data, then this could be easily added to the subsets of the decomposition; based on this outgroup all subset trees could be rooted. For supertree reconstruction this outgroup can be removed. As a second strategy, where no global outgroup is needed, we could use the tree-based decomposition itself to find individual outgroup for each subsets. Here, the outgroup of a subset is a taxon that has high distance in the guide tree and is not part of an overlapping subset. As the general idea of adding an outgroup is contrary to dividing the data into subsets of closely related taxa, it may be useful to do an initial alignment without the outgroup and adding the outgroup afterwards (e.g. using MAFFT). Future evaluations have to show if such an approach can be as accurate as using an unrooted supertree method, such as SuperFine. If this is the case, BCD is the missing piece for a very flexible and scalable divide-and-conquer setup for highly accurate large scale phylogeny reconstruction.

Bibliography

[1] Aho, A. V., Sagiv, Y., Szymanski, T. G., and Ullman, J. D. (1981). Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J Comput*, 10(3), 405–421. 10, 42

[2] Ahuja, R. K., Orlin, J. B., Stein, C., and Tarjan, R. E. (1994). Improved algorithms for bipartite network flow. *SIAM Journal on Computing*, 23(5), 906–933. 58

[3] Allman, E., Degnan, J., and Rhodes, J. (2016). Species tree inference from gene splits by unrooted STAR methods. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 62, 1–1. 13

[4] Arenas, M. (2015). Trends in substitution models of molecular evolution. *Frontiers in genetics*, 6, 319. 6

[5] Badger, J., Kearney, P., Li, M., Tsang, J., and Jiang, T. (2004). Selecting the branches for an evolutionary tree. *Journal of Algorithms*, 51(1), 1–14. 12

[6] Bansal, M. S., Burleigh, J. G., Eulenstein, O., and Fernández-Baca, D. (2010). Robinson-Foulds supertrees. *Algorithms for Molecular Biology*, 5(1), 1–12. 10, 27

[7] Baum, B. R. (1992). Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41(1), 3–10. 10, 12

[8] Beck, R. M. D., Bininda-Emonds, O. R. P., Cardillo, M., Liu, F.-G. R., and Purvis, A. (2006). A higher-level MRP supertree of placental mammals. *BMC Evol Biol*, 6, 93. 26, 27

[9] Berry, V., Bininda-Emonds, O. R. P., and Semple, C. (2013). Amalgamating source trees with different taxonomic levels. *Syst Biol*, 62(2), 231–249. 10

[10] Bininda-Emonds, O. R. and Bryant, H. N. (1998). Properties of matrix representation with parsimony analyses. *Syst Biol*, 47(3), 497–508. 12

[11] Bininda-Emonds, O. R. P. (2004a). The evolution of supertrees. *Trends Ecol Evol*, 19(6), 315–322. 10

[12] Bininda-Emonds, O. R. P., editor (2004b). *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 4 of *Computational Biology Series*. Kluwer Academic. 3

[13] Bininda-Emonds, O. R. P. and Sanderson, M. J. (2001). Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Syst Biol*, 50(4), 565–579. 27

[14] Böcker, S. (2004). Unrooted supertrees: Limitations, traps, and phylogenetic patchworks. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, chapter 15, pages 331–351. Kluwer Academic. 6

[15] Böcker, S., Letzel, M., Lipták, Zs., and Pervukhin, A. (2009). SIRIUS: Decomposing isotope patterns for metabolite identification. *Bioinformatics*, 25(2), 218–224. ix

[16] Böcker, S., Bui, B., Nicolas, F., and Truss, A. (2011). Intractability of the minimum flip supertree problem and its variants. Technical report, arXiv preprint arXiv:1112.4536. arXiv:1112.4536v1. 12

[17] Bodlaender, H. L., Fellows, M. R., and Warnow, T. J. (1992). Two strikes against perfect phylogeny. In *Proc. of International Colloquium on Automata Languages and Programming*, pages 273–283. 5

[18] Bonifati, V., Rizzu, P., Van Baren, M. J., Schaap, O., Breedveld, G. J., Krieger, E., Dekker, M. C., Squitieri, F., Ibanez, P., Joosse, M., *et al.* (2003). Mutations in the dj-1 gene associated with autosomal recessive early-onset parkinsonism. *Science*, 299(5604), 256–259. 1

[19] Brinkmeyer, M., Griebel, T., and Böcker, S. (2010). Polynomial supertree methods revisited. In *Proc. of Pattern Recognition in Bioinformatics (PRIB 2010)*, volume 6282 of *Lect Notes Comput Sci*, pages 183–194. Springer, Berlin. 12

[20] Brinkmeyer, M., Griebel, T., and Böcker, S. (2011). Polynomial supertree methods revisited. *Adv Bioinformatics*, 2011(Article ID 524182), 21 pages. 27

[21] Brinkmeyer, M., Griebel, T., and Böcker, S. (2013). FlipCut supertrees: Towards matrix representation accuracy in polynomial time. *Algorithmica*, 67(2), 142–160. ix, 10, 11, 12, 27, 39, 41, 42, 52, 98

[22] Bruno, W. J., Socci, N. D., and Halpern, A. L. (2000). Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction. *Mol Biol Evol*, 17, 189–197. 7

[23] Bryant, D. (1997). *Building trees, hunting for trees, and comparing trees: Theory and methods in phylogenetic analysis*. Ph.D. thesis, University of Canterbury, Christchurch, New Zealand. 42, 52

[24] Bryant, D. (2003). A classification of consensus methods for phylogenetics. In *Bioconsensus*, volume 61 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. AMS, USA. 34

[25] Bush, R. M., Bender, C. A., Subbarao, K., Cox, N. J., and Fitch, W. M. (1999). Predicting the evolution of human influenza a. *Science*, 286(5446), 1921–1925. 1

[26] Cardillo, M., Bininda-Emonds, R. P., Boakes, E., and Purvis, A. (2004). A species-level phylogenetic supertree of marsupials. *J Zool*, 264(1), 11–31. 26, 27

[27] Cavalli-Sforza, L. L. and Edwards, A. W. (1967). Phylogenetic analysis: models and estimation procedures. *Evolution*, 21(3), 550–570. 7

[28] Chang, B. S. and Donoghue, M. J. (2000). Recreating ancestral proteins. *Trends in ecology & evolution*, 15(3), 109–114. 1

[29] Chang, J. T. (1996). Full reconstruction of markov models on evolutionary trees: identifiability and consistency. *Mathematical biosciences*, 137(1), 51–73. 9

[30] Chatzou, M., Magis, C., Chang, J.-M., Kemena, C., Bussotti, G., Erb, I., and Notredame, C. (2015). Multiple sequence alignment modeling: methods and applications. *Briefings in bioinformatics*, 17(6), 1009–1023. 6

[31] Chen, D., Eulenstein, O., Fernández-Baca, D., and Sanderson, M. (2006). Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Trans Comput Biology Bioinform*, 3(2), 165–173. 10, 12, 27

[32] Chor, B. and Tuller, T. (2005). Maximum likelihood of evolutionary trees: Hardness and approximation. *Bioinformatics*, 21 Suppl 1, i97–106. 8

[33] Cotton, J. A. and Wilkinson, M. (2007). Majority-rule supertrees. *Syst Biol*, 56(3), 445–452. 10

[34] Creevey, C. J. and Mcinerney, J. O. (2005). Clann: Investigating phylogenetic information through supertree analyses. *Bioinformatics*, 21(3), 390–2. 12, 27

[35] Criscuolo, A., Berry, V., Douzery, E. J. P., and Gascuel, O. (2006). SDM: A fast distance-based approach for (super) tree building in phylogenomics. *Syst Biol*, 55(5), 740–755. 10, 13, 27

[36] Csurös, M. (2002). Fast recovery of evolutionary trees with thousands of nodes. *Journal of Computational Biology*, 9(2), 277–297. 7

[37] Darwin, C. (1859). *The origin of species by means of natural selection: or, the preservation of favoured races in the struggle for life and the descent of man and selection in relation to sex*. Modern library. 1

[38] Day, W., Johnson, D., and Sankoff, D. (1986). The computational complexity of inferring rooted phylogenies by parsimony. *Math Biosci*, 81(1), 33–42. 8

[39] Desper, R. and Gascuel, O. (2002). Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of computational biology*, 9(5), 687–705. 7, 9

[40] Diestel, R. (2010). *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, 4th edition. 3

[41] Dührkop, K., Shen, H., Meusel, M., Rousu, J., and Böcker, S. (2015). Searching molecular structure databases with tandem mass spectra using CSI:FingerID. *Proc Natl Acad Sci U S A*, 112(41), 12580–12585. ix

[42] Edgar, R. C. (2004). Muscle: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, 32(5), 1792–1797. 6

[43] Edgar, R. C. and Batzoglou, S. (2006). Multiple sequence alignment. *Current opinion in structural biology*, 16(3), 368–373. 6

[44] Eisen, J. A. and Fraser, C. M. (2003). Phylogenomics: intersection of evolution and genomics. *Science*, 300(5626), 1706. 1

[45] Elias, I. (2006). Settling the intractability of multiple alignment. *J Comput Biol*, 13(7), 1323–1339. 6

[46] Elias, P., Feinstein, A., and Shannon, C. (1956). A note on the maximum flow through a network. *IEEE Transactions on Information Theory*, 2(4), 117–119. 41

[47] Eulenstein, O., Chen, D., Burleigh, J. G., Fernández-Baca, D., and Sanderson, M. J. (2004). Performance of flip supertree construction with a heuristic algorithm. *Syst Biol*, 53(2), 299–308. 25, 27

[48] Felsenstein, J. (1978). Cases in which parsimony or compatibility methods will be positively misleading. *Syst Zool*, 27, 401–410. 9

[49] Felsenstein, J. (1985). Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, 39(4), 783–791. 9

[50] Felsenstein, J. (2004). *Inferring Phylogenies*. Sinauer Associates, Sunderland, Massachusetts. 3, 5, 8

[51] Fitch, W. M. (1971). Toward defining the course of evolution: Minimum change for a specific tree topology. *Syst Zool*, 20(4), 406–416. 8

[52] Fitch, W. M. and Margoliash, E. (1967). Construction of phylogenetic trees. *Science*, 155, 279–84. 7

[53] Fleischauer, M. (2012). *Bessere Supertrees durch suboptimale Schnitte im FlipCut Graphen*. Diplomarbeit, Friedrich-Schiller-Universität Jena. 58

[54] Fleischauer, M. and Böcker, S. (2016). Collecting reliable clades using the greedy strict consensus merger. *PeerJ*, 4, e2172. ix

[55] Fleischauer, M. and Böcker, S. (2017). Bad Clade Deletion supertrees: A fast and accurate supertree algorithm. *Mol Biol Evol*, 34, 2408–2421. ix

[56] Fleischauer, M. and Böcker, S. (2018). BCD Beam Search: Considering suboptimal partial solutions in Bad Clade Deletion supertrees. *PeerJ*, 6, e4987. ix

[57] Ford, L. and Fulkerson, D. (1956). Maximal flow through a network. *Can J Math*, 8, 399–404. 41

[58] Ford, L. R. and Fulkerson, D. R. (1962). *Flows in Networks*. Princeton University Press, Princeton, New Jersey. 41

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

[60] Gaschen, B., Taylor, J., Yusim, K., Foley, B., Gao, F., Lang, D., Novitsky, V., Haynes, B., Hahn, B. H., Bhattacharya, T., *et al.* (2002). Diversity considerations in hiv-1 vaccine selection. *Science*, 296(5577), 2354–2360. 1

[61] Gascuel, O. (1997). Bionj: an improved version of the nj algorithm based on a simple model of sequence data. *Molecular biology and evolution*, 14(7), 685–695. 7

[62] Geyer, C. J. (1991). Markov chain monte carlo maximum likelihood. 8

[63] Goloboff, P. A., Farris, J. S., and Nixon, K. C. (2008). Tnt, a free program for phylogenetic analysis. *Cladistics*, 24(5), 774–786. 9

[64] Griebel, T., Brinkmeyer, M., and Böcker, S. (2008). EPoS: A modular software framework for phylogenetic analysis. *Bioinformatics*, 24(20), 2399–2400. ix

[65] Guindon, S. and Gascuel, O. (2003). A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst Biol*, 52(5), 696–704. 25

[66] Guindon, S., Dufayard, J.-F., Lefort, V., Anisimova, M., Hordijk, W., and Gascuel, O. (2010). New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0. *Systematic biology*, 59(3), 307–321. 9

[67] Hao, J. X. and Orlin, J. B. (1994). A faster algorithm for finding the minimum cut in a directed graph. *J Algorithms*, 17(3), 424–446. 58

[68] Hartigan, J. A. (1973). Minimum mutation fits to a given tree. *Biometrics*, 29(1), 53–65. 8

[69] Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1), 97–109. 8

[70] Hedtke, S. M., Patiny, S., and Danforth, B. N. (2013). The bee tree of life: a supermatrix approach to apoid phylogeny and biogeography. *BMC Evol Biol*, 13(1), 138. 26, 27

[71] Herman, J. L., Challis, C. J., Novák, Á., Hein, J., and Schmidler, S. C. (2014). Simultaneous bayesian estimation of alignment and phylogeny under a joint model of protein sequence and structure. *Molecular biology and evolution*, 31(9), 2251–2266. 6

[72] Higgins, D. G. and Sharp, P. M. (1988). CLUSTAL: A package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1), 237–244. 6

[73] Hinchliff, C. E., Smith, S. A., Allman, J. F., Burleigh, J. G., Chaudhary, R., Coghill, L. M., Crandall, K. A., Deng, J., Drew, B. T., Gazis, R., *et al.* (2015). Synthesis of phylogeny and taxonomy into a comprehensive tree of life. *Proceedings of the National Academy of Sciences*, 112(41), 12764–12769. 1

[74] Hoang, D. T., Chernomor, O., von Haeseler, A., Minh, B. Q., and Le, S. V. (2017). Ufboot2: Improving the ultrafast bootstrap approximation. *Molecular biology and evolution*, page msx281. 9

[75] Hoang, D. T., Vinh, L. S., Flouri, T., Stamatakis, A., von Haeseler, A., and Minh, B. Q. (2018). Mpboot: fast phylogenetic maximum parsimony tree inference and bootstrap approximation. *BMC evolutionary biology*, 18(1), 11. 9

[76] Holland, B., Conner, G., Huber, K., and Moulton, V. (2007). Imputing supertrees and supernetworks from quartets. *Syst Biol*, 56(1), 57–67. 10

[77] Huelsenbeck, J. P., Ronquist, F., Nielsen, R., and Bollback, J. P. (2001). Bayesian inference of phylogeny and its impact on evolutionary biology. *Science*, 294(5550), 2310–2314. 9

[78] Huson, D. H., Nettles, S. M., and Warnow, T. J. (1999a). Disk-Covering, a fast-converging method for phylogenetic tree reconstruction. *J Comput Biol*, 6(3-4), 369–386. 15

[79] Huson, D. H., Vawter, L., and Warnow, T. J. (1999b). Solving large scale phylogenetic problems using DCM2. In *Proc. of Intelligent Systems for Molecular Biology (ISMB 1999)*, pages 118–129. 12, 13, 15, 31, 32

[80] Huson, D. H., Rupp, R., and Scornavacca, C. (2011). *Phylogenetic Networks*. Cambridge University Press. 1, 3

[81] Jansson, J., Lemence, R. S., and Lingas, A. (2012). The complexity of inferring a minimally resolved phylogenetic supertree. *SIAM J Comput*, 41(1), 272–291. 42, 52

[82] Jones, K. E., Purvis, A., Maclarnon, A., Bininda-Emonds, O. R., and Simmons, N. B. (2002). A phylogenetic supertree of the bats (mammalia: Chiroptera). *Biological Reviews*, 77(2), 223–259. 26, 27

[83] Jukes, T. H. and Cantor, C. R. (1969). Evolution of protein molecules. In H. N. Munro, editor, *Mammalian Protein Metabolism*, volume III, chapter 24, pages 21–132. Academic Press, New York. 6, 9

[84] Karger, D. R. and Stein, C. (1996). A new approach to the minimum cut problem. *J ACM*, 43(4), 601–640. 58, 59

[85] Katoh, K., ichi Kuma, K., Toh, H., and Miyata, T. (2005). MAFFT version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Res*, 33(2), 511–518. 6

[86] Kemena, C. and Notredame, C. (2009). Upcoming challenges for multiple sequence alignment methods in the high-throughput era. *Bioinformatics*, 25(19), 2455–2465. 6

[87] Kennedy, M. and Page, R. D. (2002). Seabird supertrees: combining partial estimates of procellariiform phylogeny. *The Auk*, 119(1), 88–108. 26, 27

[88] Kim, J. (1996). General inconsistency conditions for maximum parsimony: effects of branch lengths and increasing numbers of taxa. *Systematic Biology*, 45(3), 363–374. 9

[89] Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J Mol Evol*, 16, 111–120. 25

[90] Korber, B., Muldoon, M., Theiler, J., Gao, F., Gupta, R., Lapedes, A., Hahn, B., Wolinsky, S., and Bhattacharya, T. (2000). Timing the ancestor of the hiv-1 pandemic strains. *science*, 288(5472), 1789–1796. 1

[91] Kozlov, A. M., Aberer, A. J., and Stamatakis, A. (2015). Examl version 3: a tool for phylogenomic analyses on supercomputers. *Bioinformatics*, 31(15), 2577–2579. 2, 9

[92] Kupczok, A., Schmidt, H. A., and von Haeseler, A. (2010). Accuracy of phylogeny reconstruction methods combining overlapping gene data sets. *Algorithms Mol Biol*, 5, 37. 13, 27

[93] La, D., Sutch, B., and Livesay, D. R. (2005). Predicting protein functional sites with phylogenetic motifs. *Proteins: Structure, Function, and Bioinformatics*, 58(2), 309–320. 1

[94] Larget, B. R., Kotha, S. K., Dewey, C. N., and Ane, C. (2010). BUCKy: Gene tree/species tree reconciliation with Bayesian concordance analysis. *Bioinformatics*, 26(22), 2910–2911. 13

[95] Lefort, V., Desper, R., and Gascuel, O. (2015). Fastme 2.0: a comprehensive, accurate, and fast distance-based phylogeny inference program. *Molecular biology and evolution*, 32(10), 2798–2800. 7, 9

[96] Li, C., Medlar, A., and Löytynoja, A. (2016). Co-estimation of phylogeny-aware alignment and phylogenetic tree. *bioRxiv*, page 077503. 6

[97] Liu, K. and Warnow, T. (2012). Treelength optimization for phylogeny estimation. *PLoS One*, 7(3), e33104.

[98] Liu, K., Raghavan, S., Nelesen, S., Linder, C. R., and Warnow, T. (2009a). Rapid and accurate large-scale coestimation of sequence alignments and phylogenetic trees. *Science*, 324(5934), 1561–1564. 6, 10, 17

[99] Liu, K., Linder, C. R., and Warnow, T. (2011a). Raxml and fasttree: comparing two methods for large-scale maximum likelihood phylogeny estimation. *PloS one*, 6(11), e27731. 10

[100] Liu, K., Warnow, T. J., Holder, M. T., Nelesen, S. M., Yu, J., Stamatakis, A. P., and Linder, C. R. (2011b). Sate-ii: very fast and accurate simultaneous estimation of multiple sequence alignments and phylogenetic trees. *Systematic biology*, 61(1), 90–106. 6, 10, 17

[101] Liu, L. and Yu, L. (2011). Estimating species trees from unrooted gene trees. *Systematic Biology*, 60(5), 661–667. 13

[102] Liu, L., Yu, L., Pearl, D. K., and Edwards, S. V. (2009b). Estimating species phylogenies using coalescence times among sequences. *Systematic Biology*, 58(5), 468–477.

[103] Liu, L., Yu, L., and Edwards, S. V. (2010). A maximum pseudo-likelihood approach for estimating species trees under the coalescent model. *BMC Evolutionary Biology*, 10(1), 302. 13

[104] Maddison, D. R., Schulz, K.-S., and Maddison, W. P. (2007). The tree of life web project. *Zootaxa*, 1668(1), 19–40. 1

[105] Maddison, W. P. (1997). Gene trees in species trees. *Syst Biol*, 46(3), 523–536. 13

[106] Mallo, D. and Posada, D. (2016). Multilocus inference of species trees and dna barcoding. *Phil. Trans. R. Soc. B*, 371(1702), 20150335. 13

[107] Markin, A. and Eulenstein, O. (2016). Manhattan path-difference median trees. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics - BCB 16*. Association for Computing Machinery (ACM). 10

[108] McMahon, M. M. and Sanderson, M. J. (2006). Phylogenetic supermatrix analysis of genbank sequences from 2228 Papilionoid legumes. *Syst Biol*, 55(5), 818–836. 26, 27

[109] McMorris, F. R. and Wilkinson, M. (2011). Conservative supertrees. *Systematic Biology*, 60(2), 232–238. 10

[110] Minh, B. Q., Nguyen, M. A. T., and von Haeseler, A. (2013). Ultrafast approximation for phylogenetic bootstrap. *Molecular biology and evolution*, 30(5), 1188–1195. 9

[111] Mirarab, S. and Warnow, T. (2015). Astral-ii: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, 31(12), i44–i52. 13

[112] Mirarab, S., Reaz, R., Bayzid, M. S., Zimmermann, T., Swenson, M. S., and Warnow, T. (2014). ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, 30(17), i541–i548. 13

[113] Mirarab, S., Nguyen, N., Guo, S., Wang, L.-S., Kim, J., and Warnow, T. (2015). Pasta: ultra-large multiple sequence alignment for nucleotide and amino-acid sequences. *Journal of Computational Biology*, 22(5), 377–386. 6

[114] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48, 443–453. 6

[115] Nei, M. and Kumar, S. (2000). *Molecular evolution and phylogenetics*. Oxford university press. 3

[116] Nelesen, S., Liu, K., Wang, L.-S., Linder, C. R., and Warnow, T. (2012). DACtal: divide-and-conquer trees (almost) without alignments. *Bioinformatics*, 28(12), i274–i282. 2, 16, 17

[117] Nelesen, S. M. (2009). *Improved methods for phylogenetics*. Ph.D. thesis, University of Texas at Austin. 16, 17

[118] Nelson, M. I., Simonsen, L., Viboud, C., Miller, M. A., and Holmes, E. C. (2007). Phylogenetic analysis reveals the global migration of seasonal influenza a viruses. *PLoS pathogens*, 3(9), e131. 1

[119] Nguyen, L.-T., Schmidt, H. A., von Haeseler, A., and Minh, B. Q. (2014). Iq-tree: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. *Molecular biology and evolution*, 32(1), 268–274. 2, 9

[120] Nguyen, N., Mirarab, S., and Warnow, T. (2012). MRL and SuperFine+MRL: new supertree methods. *Algorithms Mol Biol*, 7(1), 3. 10, 13

[121] Notredame, C., Higgins, D. G., and Heringa, J. (2000). T-coffee: A novel method for fast and accurate multiple sequence alignment. *J Mol Biol*, 302(1), 205–217. 6

[122] Novák, Á., Miklós, I., Lyngsø, R., and Hein, J. (2008a). Statalign: an extendable software package for joint bayesian estimation of alignments and evolutionary trees. *Bioinformatics*, 22(16), 2047–2048. 6

[123] Novák, Á., Miklós, I., Lyngsø, R., and Hein, J. (2008b). Statalign: an extendable software package for joint bayesian estimation of alignments and evolutionary trees. *Bioinformatics*, 24(20), 2403–2404. 6

[124] Page, R. D. M. (2002). 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. 12, 27

[125] Pe'er, I., Pupko, T., Shamir, R., and Sharan, R. (2004). Incomplete directed perfect phylogeny. *SIAM J Comput*, 33(3), 590–607. 11, 12, 39, 41, 42, 52

[126] Pisani, D. and Wilkinson, M. (2002). Matrix representation with parsimony, taxonomic congruence, and total evidence. *Syst Biol*, 51, 151–155. 12

[127] Posada, D. (2016). Phylogenomics for systematic biology. *Systematic biology*, 65(3), 353–356. 13

[128] Price, M. N., Dehal, P. S., and Arkin, A. P. (2010). Fasttree 2–approximately maximum-likelihood trees for large alignments. *PloS one*, 5(3), e9490. 8, 9

[129] Purvis, A. (1995). A composite estimate of primate phylogeny. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 348(1326), 405–421. 12, 26, 27

[130] Ragan, M. A. (1992). Phylogenetic inference based on matrix representation of trees. *Mol Phylogenet Evol*, 1(1), 53–58. 10, 12

[131] Rambaut, A. and Grassly, N. C. (1997). Seq-gen: An application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Comput Appl Biosci*, 13(3), 235–238. 22, 25

[132] Rambaut, A., Robertson, D. L., Pybus, O. G., Peeters, M., and Holmes, E. C. (2001). Human immunodeficiency virus: Phylogeny and the origin of hiv-1. *Nature*, 410(6832), 1047. 1

[133] Ranwez, V., Criscuolo, A., and Douzery, E. J. P. (2010). SuperTriplets: A triplet-based supertree approach to phylogenomics. *Bioinformatics*, 26(12), i115–i123. 10, 25, 26, 27

[134] Redelings, B. D. and Suchard, M. A. (2005). Joint Bayesian estimation of alignment and phylogeny. *Syst Biol*, 54(3), 401–418. 6

[135] Robinson, D. F. and Foulds, L. R. (1981). Comparison of phylogenetic trees. *Math Biosci*, 53(1-2), 131–147. 13, 19

[136] Roch, S. (2006). A short proof that phylogenetic tree reconstruction by maximum likelihood is hard. *IEEE/ACM Trans Comput Biology Bioinform*, 3(1), 92–94. 8

[137] Rodrigo, A. G. (1996). On combining cladograms. *Taxon*, 45(2), pp.267–274. 12

[138] Ronquist, F. (1996). Matrix representation of trees, redundancy, and weighting. *Syst Biol*, 45(2), 247–253. 27

[139] Ronquist, F. and Huelsenbeck, J. P. (2003). MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12), 1572–1574. 9

[140] Roshan, U., Moret, B., Warnow, T., and Williams, T. (2003). Greedy strict-consensus merger: A new method to combine multiple phylogenetic trees. Technical report, Department of Computer Science, University of Texas at Austin. 12, 13, 31, 32, 33, 34

[141] Roshan, U., Moret, B., Warnow, T., and Williams, T. (2004). 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. 15, 17

[142] Ross, H. and Rodrigo, A. (2004). *An assessment of matrix representation with compatibility in supertree construction*, volume 4 of *Computational Biology Book Series*, chapter 2, pages 35–63. Springer, Netherlands. 10, 12

[143] Russell, C. A., Jones, T. C., Barr, I. G., Cox, N. J., Garten, R. J., Gregory, V., Gust, I. D., Hampson, A. W., Hay, A. J., Hurt, A. C., *et al.* (2008). Influenza vaccine strain selection and recent studies on the global migration of seasonal influenza viruses. *Vaccine*, 26, D31–D34. 1

[144] Saitou, N. and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol Biol Evol*, 4(4), 406–425. 7, 9

[145] Sanderson, M. J. (2003). r8s: Inferring absolute rates of molecular evolution and divergence times in the absence of a molecular clock. *Bioinformatics*, 19(2), 301–302. 22, 25

[146] Sanderson, M. J. and Shaffer, H. B. (2002). Troubleshooting molecular phylogenetic analyses. *Annual review of ecology and Systematics*, 33(1), 49–72. 7

[147] Sanderson, M. J., Purvis, A., and Henze, C. (1998). Phylogenetic supertrees: Assembling the trees of life. *Trends Ecol Evol*, 13(3), 105–109. 10

[148] Sankoff, D. (1975). Minimal mutation trees of sequences. *SIAM J Appl Math*, 28(1), 35–42. 8

[149] Schmidt, H. A. (2003). *Phylogenetic Trees from Large Datasets*. Ph.D. thesis, Universitï¿½t Dï¿½sseldorf. 13

[150] Schmidt, H. A., Strimmer, K., Vingron, M., and von Haeseler, A. (2002). TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18, 502–504. 13

[151] Scornavacca, C., Berry, V., Lefort, V., Douzery, E. J. P., and Ranwez, V. (2008). PhySIC_IST: Cleaning source trees to infer more informative supertrees. *BMC Bioinf*, 9, 413. 10, 12, 27

[152] Searls, D. B. (2003). Pharmacophylogenomics: genes, evolution and drug targets. *Nature Reviews Drug Discovery*, 2(8), 613. 1

[153] Semple, C. and Steel, M. (2000). A supertree method for rooted trees. *Discrete Appl Math*, 105(1-3), 147–158. 12, 27

[154] Semple, C. and Steel, M. A. (2003). *Phylogenetics*. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press. 3

[155] Sevillya, G., Frenkel, Z., and Snir, S. (2016). Triplet maxcut: a new toolkit for rooted supertree. *Methods in Ecology and Evolution*, 7(11), 1359–1365. 10

[156] Sharp, P. M. and Hahn, B. H. (2010). The evolution of hiv-1 and the origin of aids. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1552), 2487–2494. 1

[157] Shindyalov, I., Kolchanov, N., and Sander, C. (1994). Can three-dimensional contacts in protein structures be predicted by analysis of correlated mutations? *Protein Engineering, Design and Selection*, 7(3), 349–358. 1

[158] Sievers, F., Dineen, D., Wilm, A., and Higgins, D. G. (2013). Making automated multiple alignments of very large numbers of protein sequences. *Bioinformatics*, 29(8), 989–995. 6, 10

[159] Snir, S. and Rao, S. (2010). Quartets MaxCut: a divide and conquer quartets algorithm. *IEEE/ACM Trans Comput Biology Bioinform*, 7(4), 704–718. 10, 13, 27

[160] Soltis, D. E., Mort, M. E., Latvis, M., Mavrodiev, E. V., O'Meara, B. C., Soltis, P. S., Burleigh, J. G., and Rubio de Casas, R. (2013). Phylogenetic relationships and character evolution analysis of saxifragales using a supermatrix approach. *Am J Bot*, 100(5), 916–929. 26, 27

[161] Stamatakis, A. (2006). RAxML-VI-HPC: Maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21), 2688–2690. 9, 28, 113

[162] Stamatakis, A. (2014). Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9), 1312–1313. 9

[163] Stamatakis, A. and Aberer, A. J. (2013). Novel parallelization schemes for large-scale likelihood-based phylogenetic inference. In *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium On*, pages 1195–1204. IEEE. 9

[164] Stamatakis, A., Hoover, P., and Rougemont, J. (2008). A rapid bootstrap algorithm for the RAxML web servers. *Syst Biol*, 57(5), 758–771. 9

[165] Steel, M. (1994). Recovering a tree from the leaf colourations it generates under a markov model. *Applied Mathematics Letters*, 7(2), 19–23. 6, 9

[166] Steel, M. and Rodrigo, A. (2008). Maximum likelihood supertrees. *Syst Biol*, 57(2), 243–250. 10, 12

[167] Steel, M. A. (1992). The complexity of reconstructing trees from qualitative characters and subtrees. *J Classif*, 9(1), 91–116. 5, 6

[168] Steel, M. A., Dress, A. W., and Böcker, S. (2000). Simple but fundamental limitations on supertree and consensus tree methods. *Syst Biol*, 49(2), 363–368. 6

[169] Strimmer, K. and von Haeseler, A. (1996). Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. *Mol Biol Evol*, 13(7), 964–969. 13, 14

[170] Subramanian, A. R., Kaufmann, M., and Morgenstern, B. (2008). DIALIGN-TX: Greedy and progressive approaches for segment-based multiple sequence alignment. *Algorithms Mol Biol*, 3(1), 6+. 6

[171] Swenson, M. S., Barbançon, F., Warnow, T., and Linder, C. R. (2010). A simulation study comparing supertree and combined analysis methods using smidgen. *Algorithms for Molecular Biology*, 5(1), 1–16. 13, 20, 21, 22, 25, 27

[172] Swenson, M. S., Suri, R., Linder, C. R., and Warnow, T. (2011). An experimental study of Quartets MaxCut and other supertree methods. *Algorithms Mol Biol*, 6, 7. 28

[173] Swenson, M. S., Suri, R., Linder, C. R., and Warnow, T. (2012). SuperFine: Fast and accurate supertree estimation. *Syst Biol*, 61(2), 214–227. 10, 13, 27, 28, 33, 34, 113

[174] Swofford, D. L. (2002). *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates. 9, 28, 113

[175] Tavaré, S. (1986). *Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences*, volume 17, pages 57–86. American Mathematical Society. 6

[176] Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). 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. 6

[177] Thompson, J. D., Linard, B., Lecompte, O., and Poch, O. (2011). A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives. *PloS one*, 6(3), e18093. 6

[178] Vachaspati, P. and Warnow, T. (2016). FastRFS: Fast and accurate robinson-foulds supertrees using constrained exact optimization. *Bioinformatics*, 33, btw600. 10, 13, 28, 113

[179] Vazirani, V. and Yannakakis, M. (1992). Suboptimal cuts: Their enumeration, weight and number. In *Proc. of International Colloquium on Automata, Languages and Programming (ICALP 1992)*, volume 623 of *Lect Notes Comput Sci*, pages 366–377. Springer, Berlin. 58

[180] Vogel, B. (2013). *Improving FlipCut Supertrees with Posterior Probabilities and Suboptimal Source Trees*. Diplomarbeit, Friedrich-Schiller-Universität Jena. 69

[181] Warnow, T. (2017). *Computational phylogenetics: An introduction to designing methods for phylogeny estimation*. Cambridge University Press. 3

[182] Wheeler, W. C., Aagesen, L., Arango, C. P., Faivovich, J., Grant, T., D'Haese, C., Janies, D., Smith, W. L., Varón, A., and Giribet, G. (2006). *Dynamic homology and phylogenetic systematics: a unified approach using POY*. American Museum of Natural History. 6

[183] Whidden, C., Zeh, N., and Beiko, R. G. (2014). Supertrees based on the subtree prune-and-regraft distance. *Syst Biol*, 63(4), 566–581. 13

[184] Wilkinson, M., Cotton, J. A., and Thorley, J. L. (2004). The information content of trees and their matrix representations. *Systematic Biology*, 53(6), 989–1001. 10

[185] Wilkinson, M., Pisani, D., Cotton, J. A., and Corfe, I. (2005a). Measuring support and finding unsupported relationships in supertrees. *Syst Biol*, 54(5), 823–831. 6, 12

[186] Wilkinson, M., Cotton, J. A., Creevey, C., Eulenstein, O., Harris, S. R., Lapointe, F.-J., Levasseur, C., Mcinerney, J. O., Pisani, D., and Thorley, J. L. (2005b). The shape of supertrees to come: Tree shape related properties of fourteen supertree methods. *Syst Biol*, 54(3), 419–431. 12

[187] Willson, S. J. (2004). Constructing rooted supertrees using distances. *Bull Math Biol*, 66(6), 1755–1783. 12, 27

[188] Wilson, E. O. (1965). A consistency test for phylogenies based on contemporaneous species. *Syst Zool*, 14(3), 214–220. 10

[189] Wojciechowski, M. F., Sanderson, M. J., Steele, K. P., and Liston, A. (2000). Molecular phylogeny of the "temperate herbaceous tribes" of papilionoid legumes: a supertree approach. *Advances in legume systematics*, 9, 277–298. 26, 27

[190] Yang, Z. (2014). *Molecular evolution: a statistical approach*. Oxford University Press. 6

[191] Yang, Z. and Rannala, B. (1997). Bayesian phylogenetic inference using DNA sequences: a Markov chain Monte Carlo method. *Molecular Biology and Evolution*, 14(7), 717–724. 8

[192] Yang, Z. and Zhu, T. (2018). Bayesian selection of misspecified models is overconfident and may cause spurious posterior probabilities for phylogenetic trees. *Proceedings of the National Academy of Sciences*, page 201712673. 9, 69

[193] Yeh, L.-P., Wang, B.-F., and Su, H.-H. (2010). Efficient algorithms for the problems of enumerating cuts by non-decreasing weights. *Algorithmica*, 56(3), 297–312. 58, 68

[194] Zharkikh, A. and Li, W.-H. (1993). Inconsistency of the maximum-parsimony method: the case of five taxa with a molecular clock. *Systematic Biology*, 42(2), 113–125. 9

A Appendix

Source Tree-based Criteria – Supplementary Results

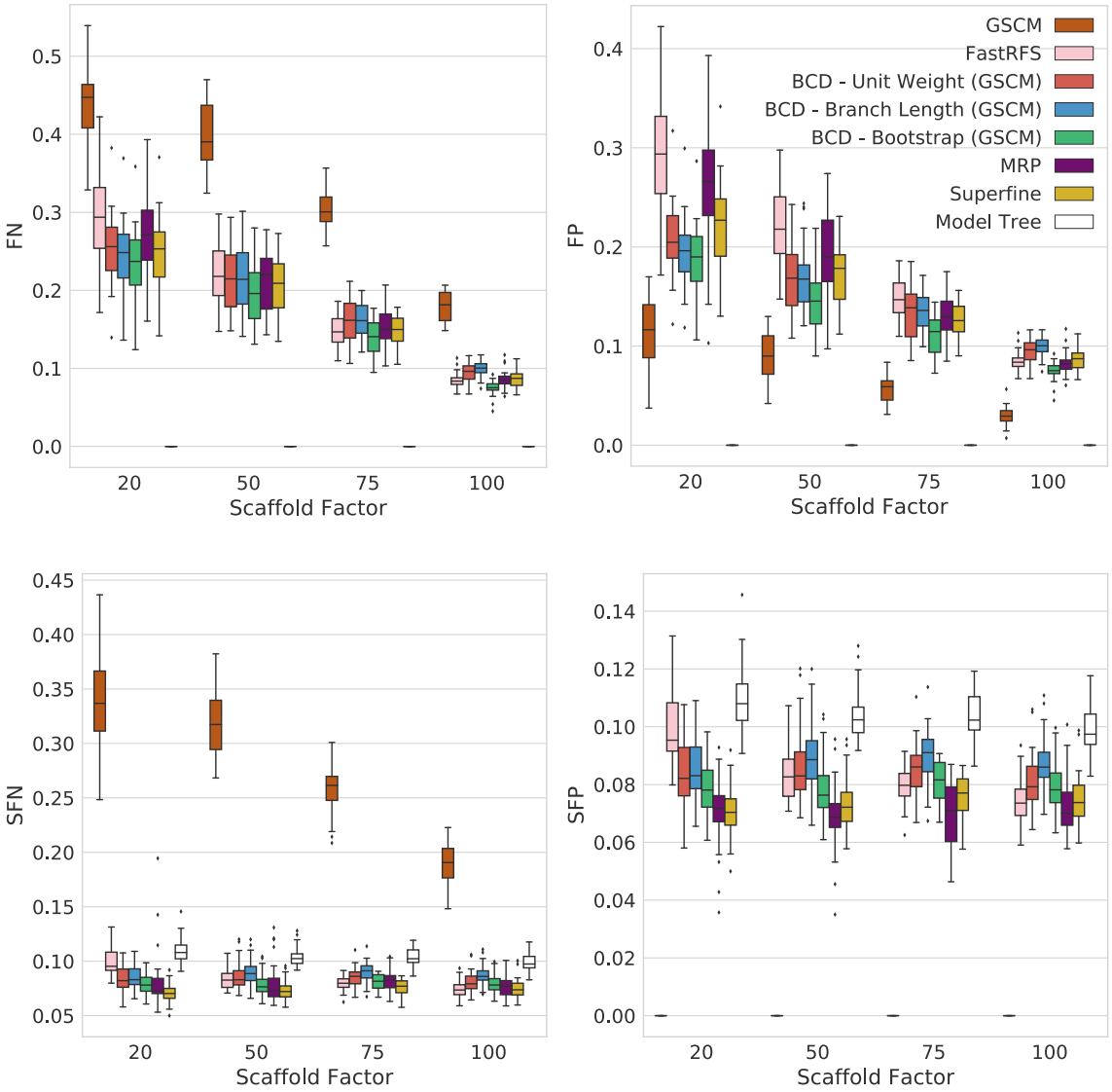


Figure A.1: Comparison of *FN* rate and *FP* rate (top) against *SFN* rate and *SFP* rate (bottom) for different tree reconstruction methods on the SMIDGenOG 1000 taxa dataset. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

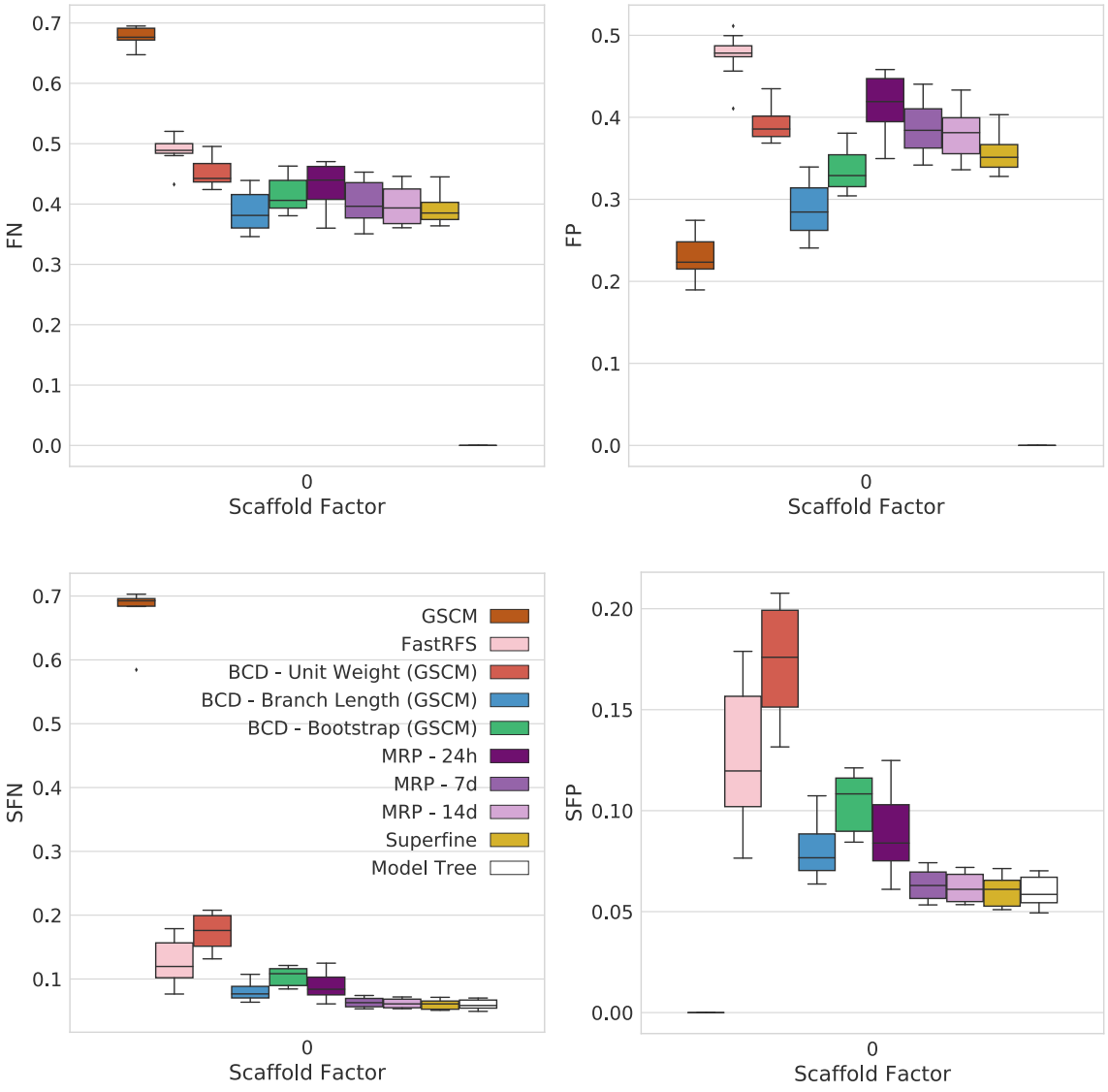


Figure A.2: Comparison of *FN* rate and *FP* rate (top) against *SFN* rate and *SFP* rate (bottom) for different tree reconstruction methods on the SMIDGenOG 5500 taxa dataset. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

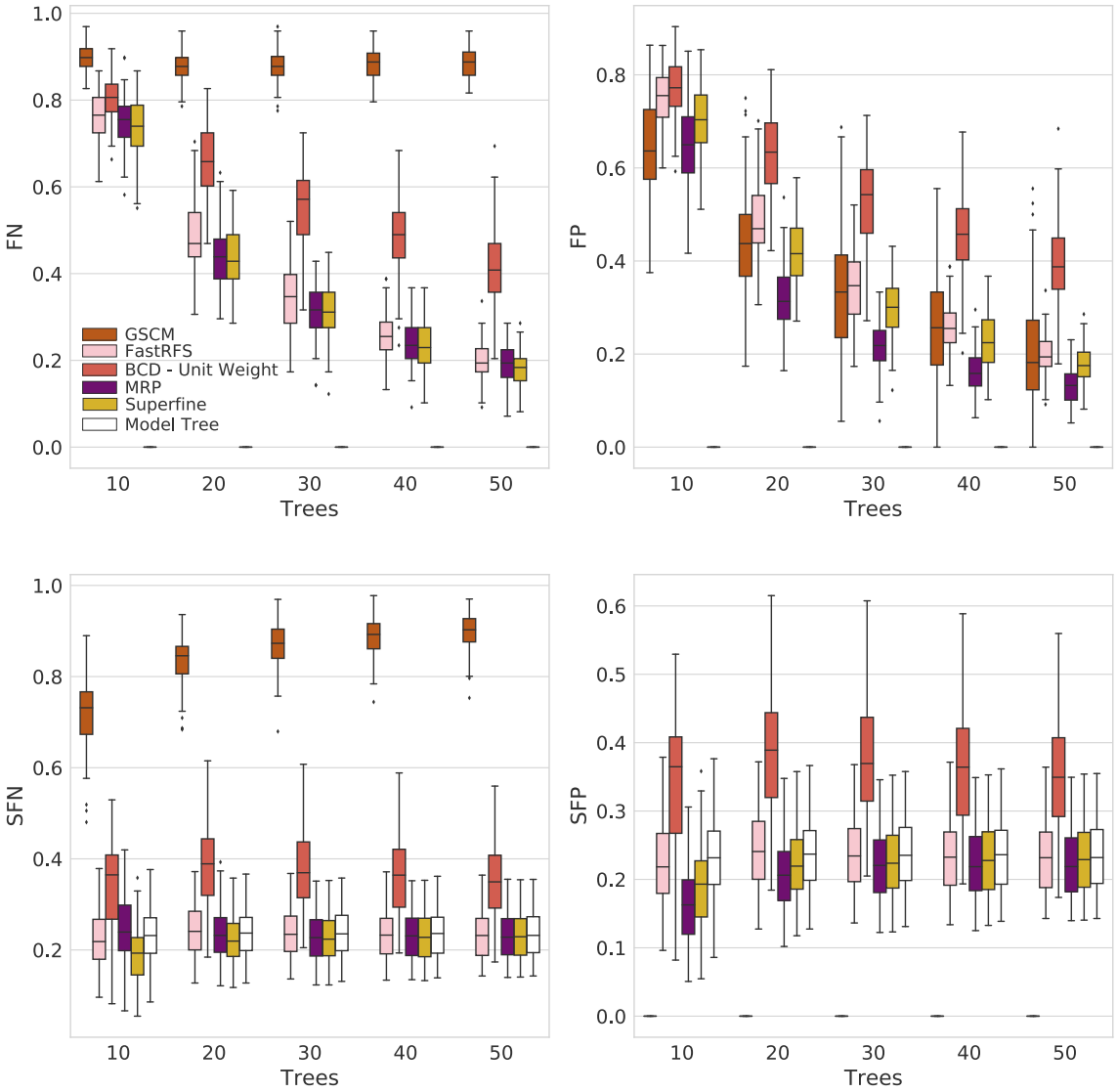


Figure A.3: Comparison of *FN* rate and *FP* rate (top) against *SFN* rate and *SFP* rate (bottom) for different tree reconstruction methods on the Supertriplet dataset with 75% data deletion for different numbers of input trees (x-axis).

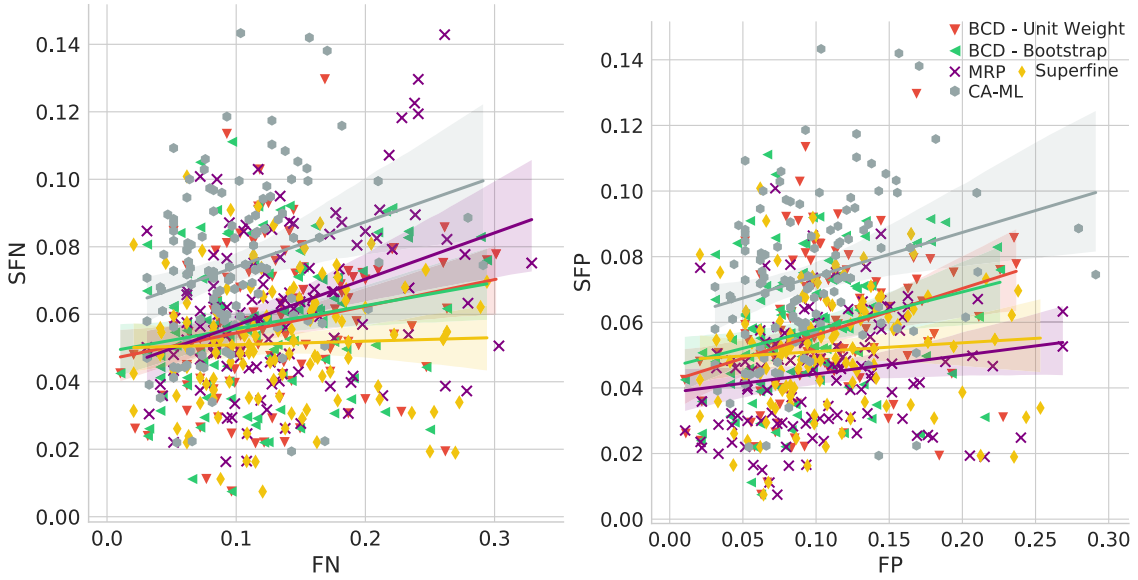


Figure A.4: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 100 taxa SMIDGenOG dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

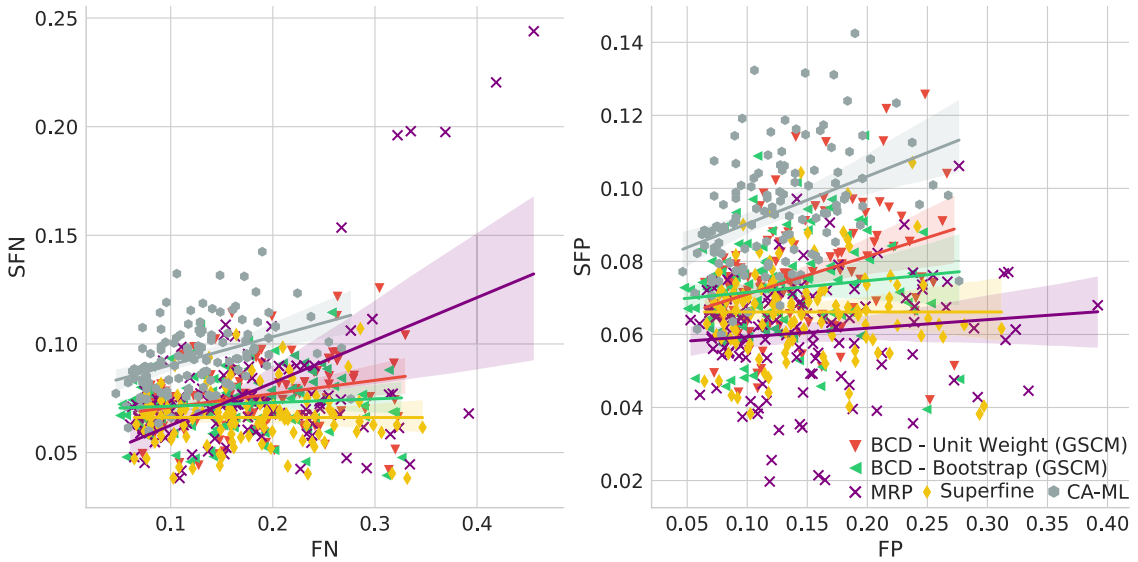


Figure A.5: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 500 taxa SMIDGenOG dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

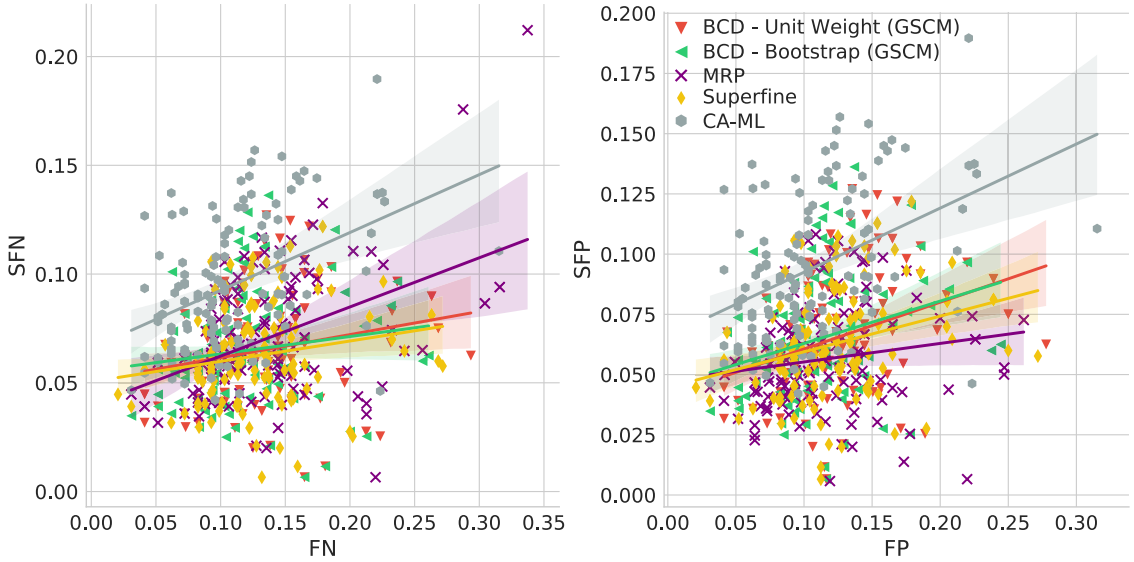


Figure A.6: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 100 taxa SMIDGen dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

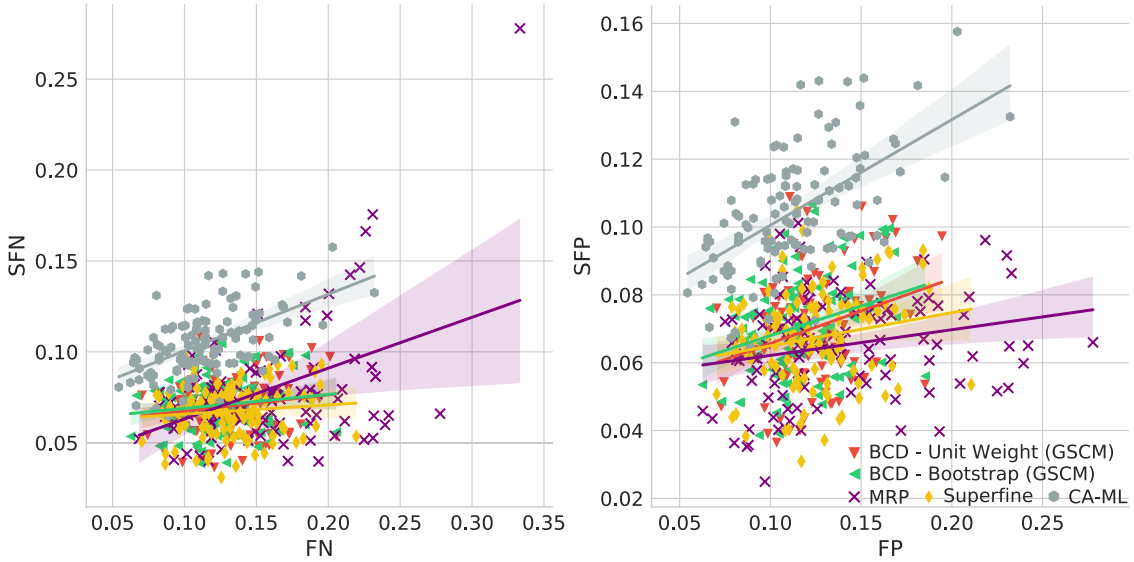


Figure A.7: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 500 taxa SMIDGen dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

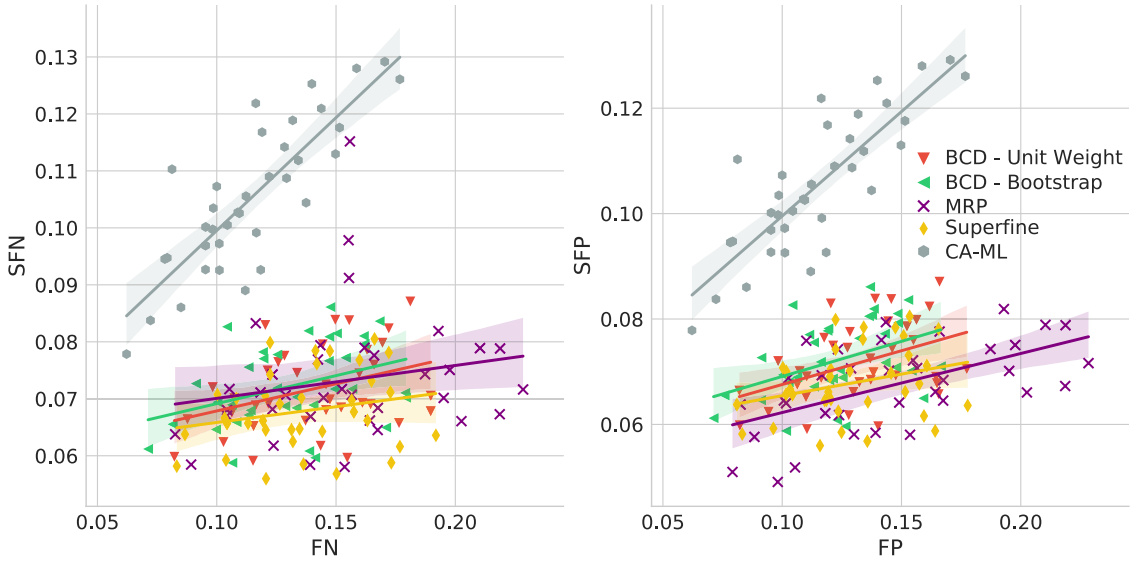


Figure A.8: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD (Unit weight and Bootstrap values), MRP, SuperFine and CA-ML on the 1000 taxa SMIDGen dataset. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

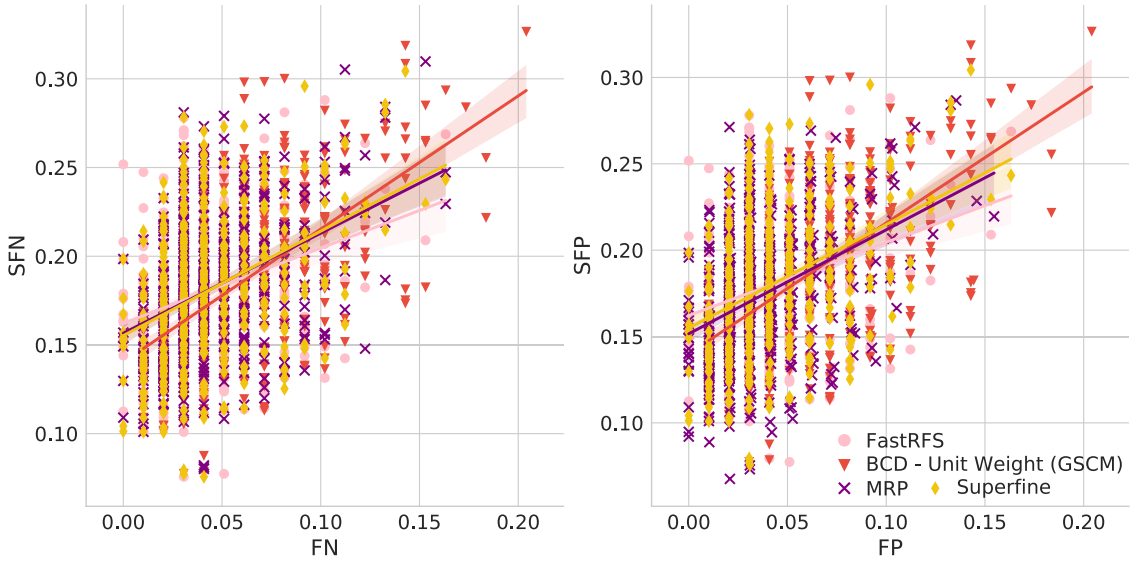


Figure A.9: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD Unit weight, FastRFS, MRP and SuperFine on the SuperTriplets Benchmark with 25% deletion rate. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

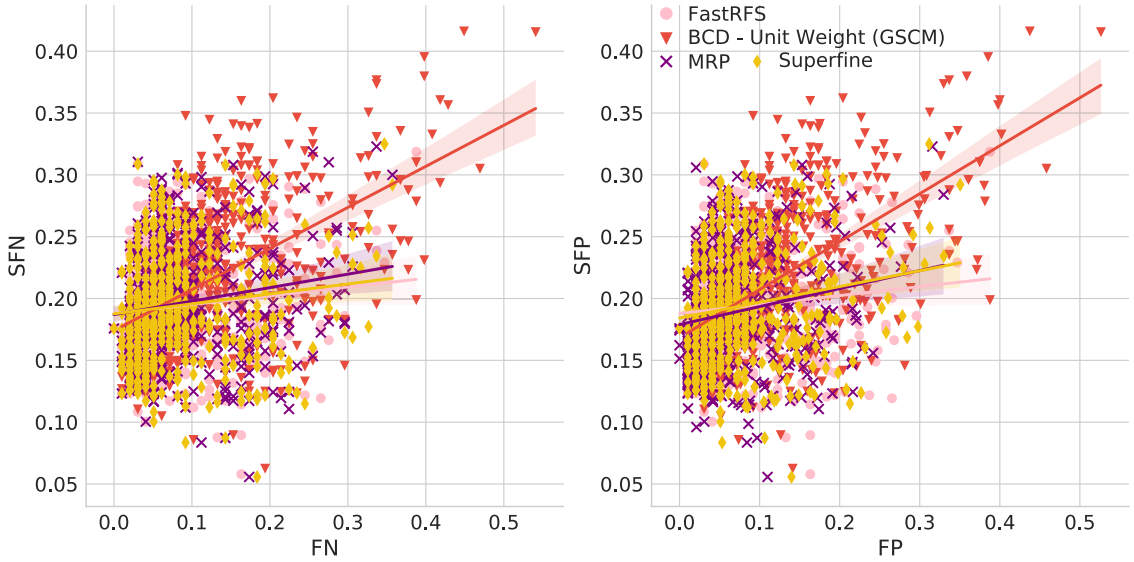


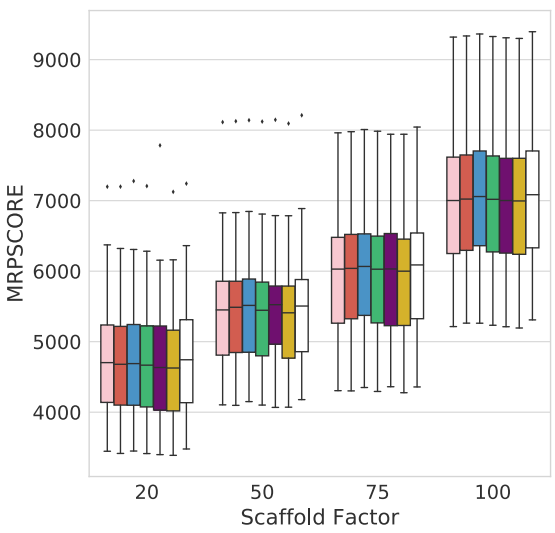
Figure A.10: Comparison of *SFN* rate against *FN* rate (left) and *SFP* rate against *FP* rate (right) between BCD Unit weight, FastRFS, MRP and SuperFine on the SuperTriplets Benchmark with 50% deletion rate. The curves are the linear regression line and the shadow around a line shows the 95% confidence interval for that regression.

MRP-Score

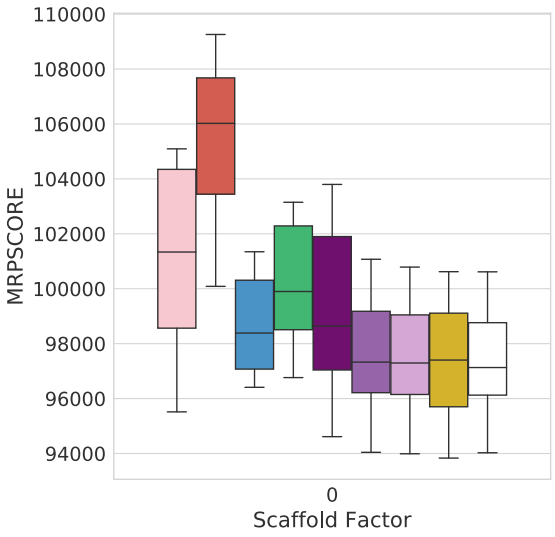
Neither has the model tree usually the best MRP score nor is the order of the different algorithms the same as for the comparison against the model tree.

For the SMIDGenOG 1000 (Figure A.11a) dataset SuperFine produces the by far best MRP scores. BCD is better (20%), equal(50%, 75%) or worse(100%) than MRP. RAxML-CA and the model tree reach always one of the worst MRP-Scores. The GSCM shows always the by far worst MRP-Scores. For the SMIDGenOG 5500 (Figure A.11b) the model tree has one of the best MRP-Scores. SuperFine is always best. MRP is worse than the model tree and on par with BCD with branch length. BCD with unit costs gets the worst MRP-Score. For the SuperTriplet dataset (Figure A.11c) no significant differences of the MRP-Scores can be noticed.

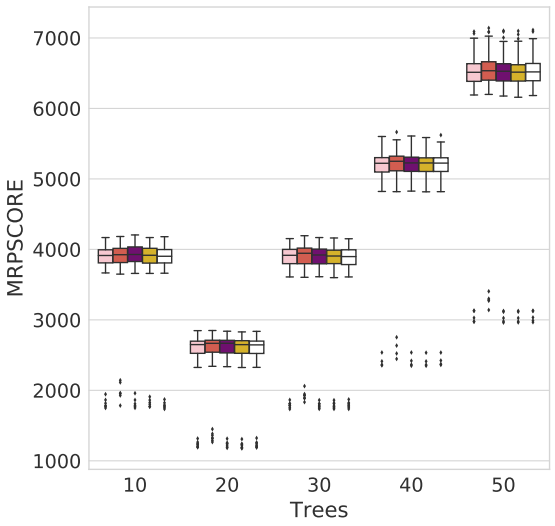
For three different datasets the MRP-Score evaluation results differ three times from the results of the model tree based evaluation.

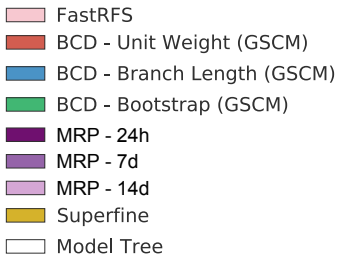


(a) SMIDGenOG 1000



(b) SMIDGenOG 5500





(c) SuperTriplets 70% Deletion

Figure A.11: MRP-Scores for different tree reconstruction methods and the model tree of simulated datasets; namely SMIDGenOG 1000 taxa (a), SMIDGenOG 5500 taxa (b) and SuperTriplet Benchmark with 75% data deletion (c). For the SMIDGenOG datasets each dash on the ordinal x-axis shows the results for a different scaffold factor. For the SuperTriplet Benchmark the x-axis shows results for different numbers of input trees. the Error bars showing the standard mean error.

Greedy strict Consensus Merger – Supplementary results

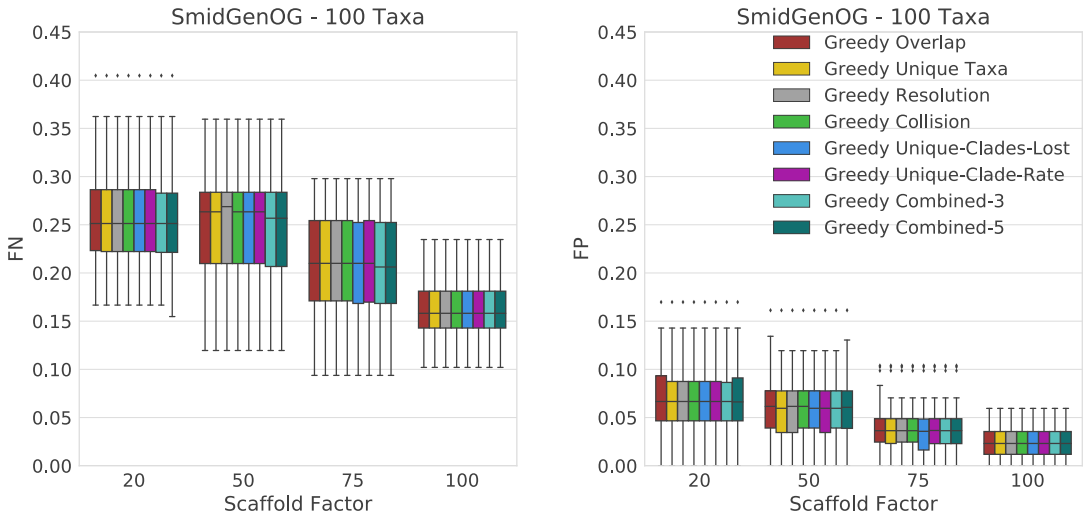


Figure A.12: *FN*-rates (left) and *FP*-rates (right) of single scorings functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5) for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 100-taxon dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scoring functions. The error bars show the standard error.

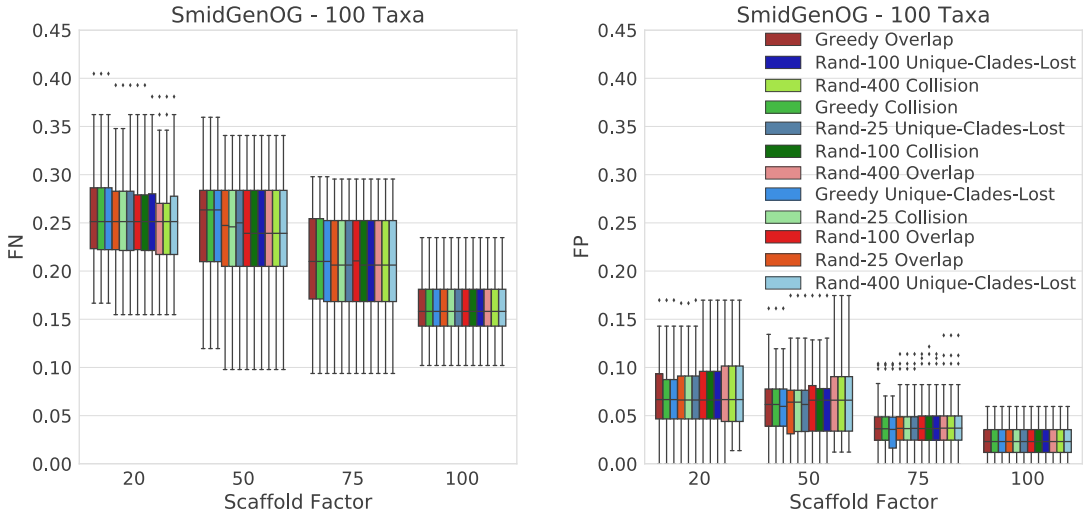
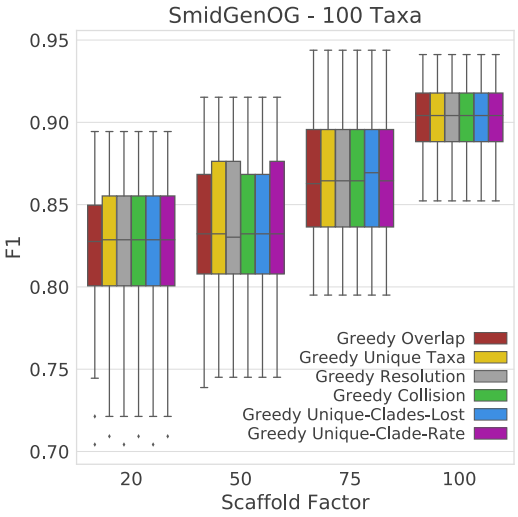
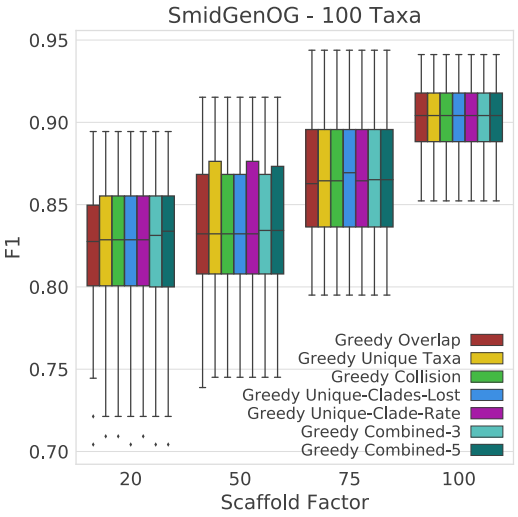


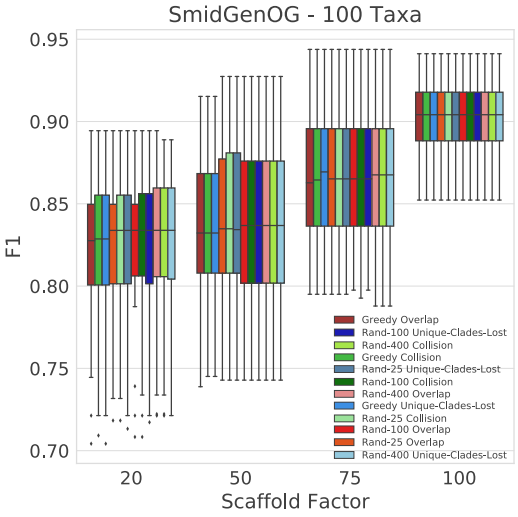
Figure A.13: *FN*-rates (left) and *FP*-rates (right) of different scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.



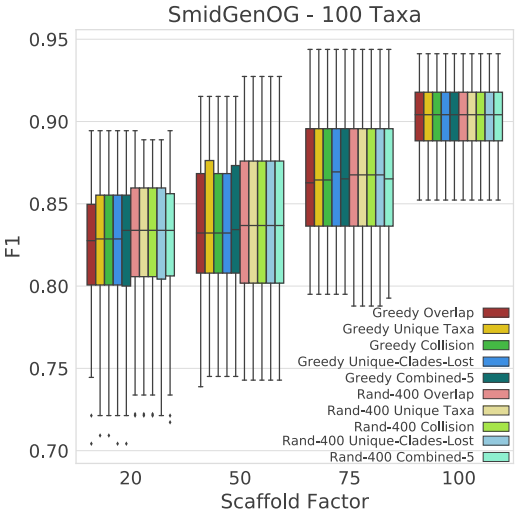
(a) Comparison of scoring functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clades-Rate).



(b) Comparison of single scoring functions (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5).



(c) Comparison of scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.



(d) Comparison of single (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost) and combined (Combined-5) scorings. Both with 400 random iterations and without randomization.

Figure A.14: F_1-scores of different scoring functions (including combined scorings) with and without randomization for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 100 taxa dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scorings. The integer value after the keyword "Rand" represents the number of randomized iterations. The error bars show the standard error.

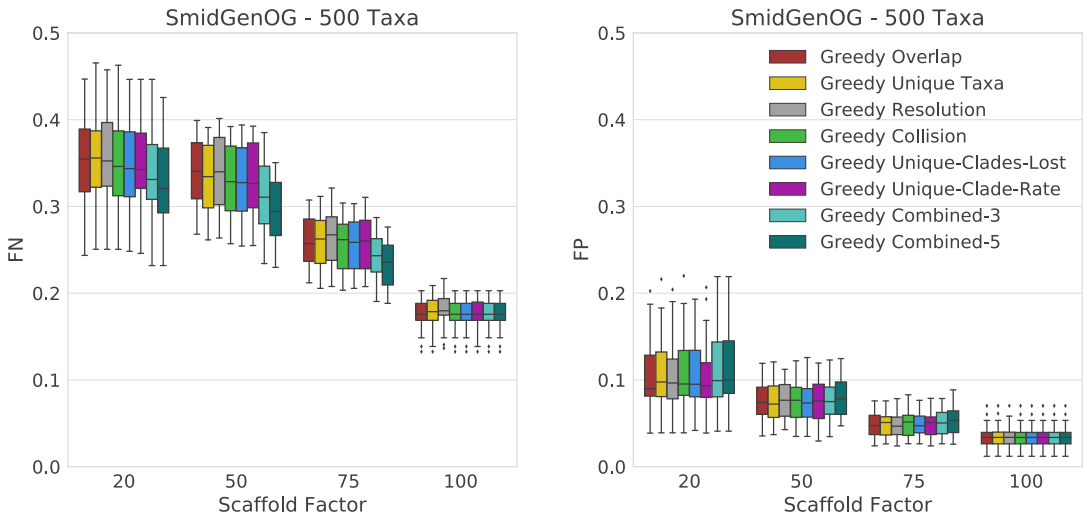


Figure A.15: *FN*-rates (left) and *FP*-rates (right) of single scorings functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5) for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 500-taxon dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scoring functions. The error bars show the standard error.

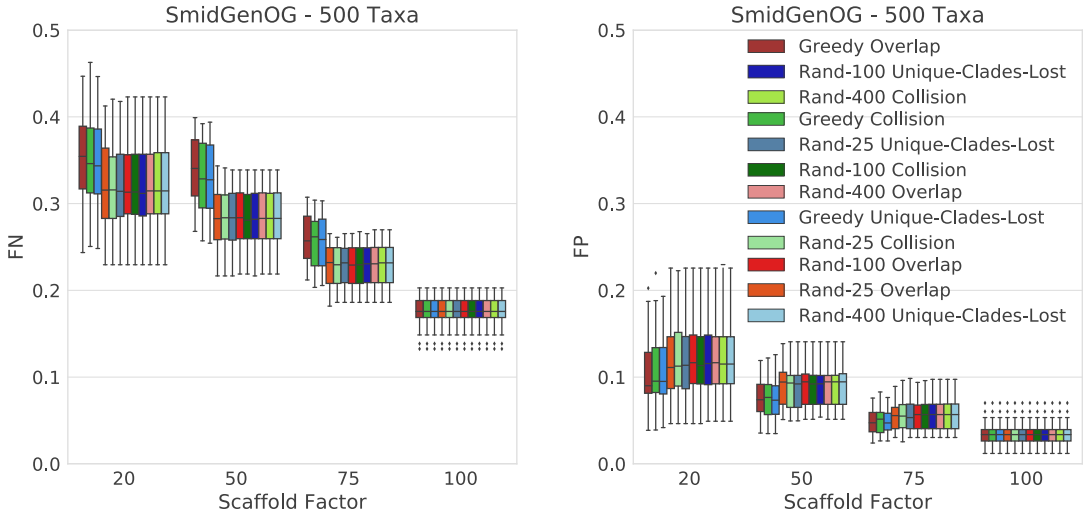
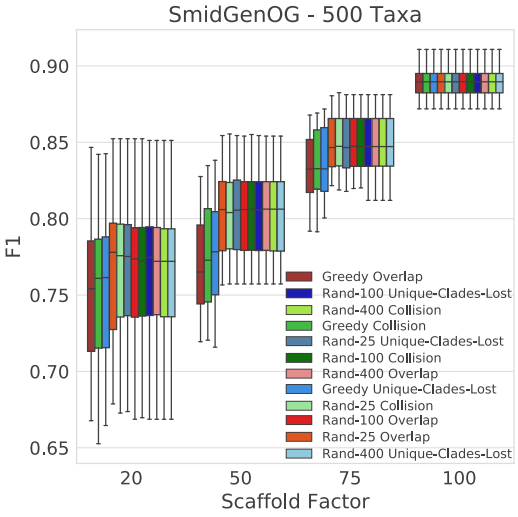


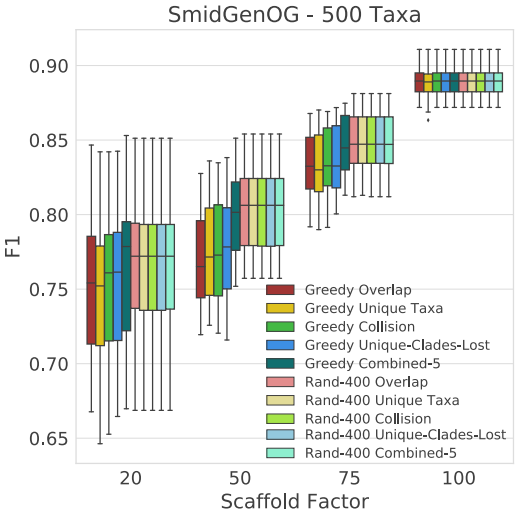
Figure A.16: *FN*-rates (left) and *FP*-rates (right) of different scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.

(a) Comparison of scoring functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clades-Rate).

(b) Comparison of single scoring functions (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3, Combined-5).



(c) Comparison of scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.



(d) Comparison of single (Overlap, Unique-Taxa, Collision, Unique-Clades-Lost) and combined (Combined-5) scorings. Both with 400 random iterations and without randomization.

Figure A.17: F_1-scores of different scoring functions (including combined scorings) with and without randomization for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 500 taxa dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scorings. The integer value after the keyword "Rand" represents the number of randomized iterations. The error bars show the standard error.

Figure A.18: *FN*-rates (left) and *FP*-rates (right) of single scorings functions (Overlap, Unique-Taxa, Resolution, Collision, Unique-Clades-Lost, Unique-Clade-Rate) and their combinations (Combined-3,Combined-5) for all scaffold factors (20 %, 50 %, 75 %, 100 %) of the 1000-taxon dataset. The Combined scorings are the semi-strict consensus of the supertrees calculated by the respective scoring functions. The error bars show the standard error.

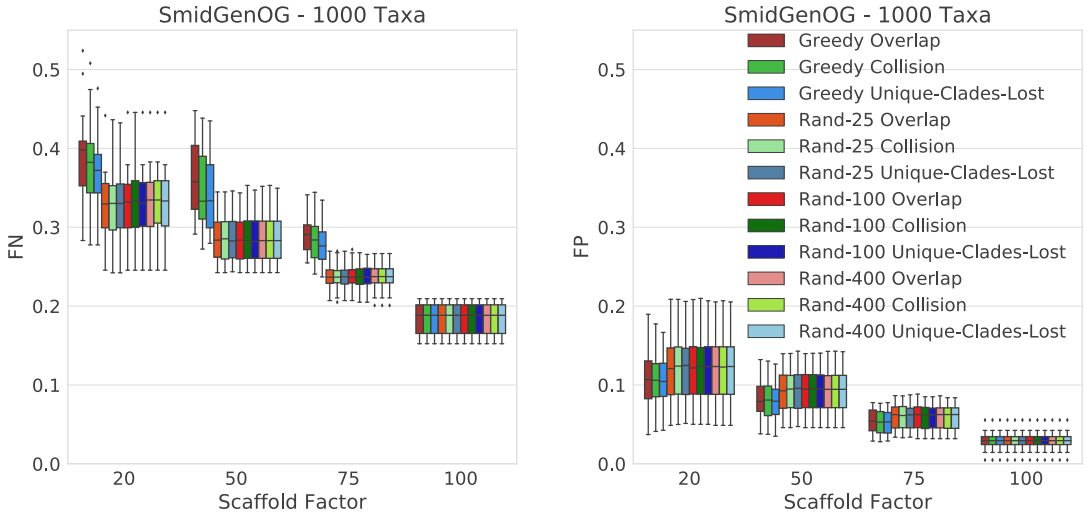


Figure A.19: *FN*-rates (left) and *FP*-rates (right) of different scoring functions (Overlap, Collision, Unique-Clades-Lost) with (25, 100 and 400 random iterations) and without randomization.

BCD Supertrees – Supplementary results

SMIDGen results.

Source trees in this dataset are unrooted, so roots for BCD calculations are chosen based on the GSCM tree as described above. Results for 100,500 and 1000 taxa model trees are shown in Figures A.20, A.21 and A.22.

Accuracy. In agreement with Brinkmeyer *et al.* [21] we find that the Edge and Level weighting with undisputed sibling preprocessing performs best among all FlipCut variants; to this end, we will not consider any other variant here. We observe, BCD compares favorably to FlipCut and any other supertree method. Different weightings of the BCD graph influence the quality of BCD supertrees, with unit weights performing worst, and bootstrap weights performing best. BCD with unit weights nevertheless outperforms MRP by a large margin for scaffold density 20 % and 50 %. BCD with branch lengths outperforms SuperFine. Finally, BCD with bootstrap weights (GSCM-BS) outperforms all other supertree methods for the simulated datasets; for small scaffold densities, it is even on par with the Combined Analysis supermatrix approach.

Running time. We report running times on the *SMIDGen* dataset in Figures A.20, A.21 and A.22; For this dataset, the number of instances may differ for different scaffold factors, due to certain instances having insufficient taxa overlap. BCD running times depend on the weighting, which determines the number of cut operations that have to be performed. BCD is faster than any other evaluated method on this dataset and, to the best of our knowledge, considerably faster than any other supertree algorithm. On the 100 taxa dataset BCD is slower than FlipCut. This is due to some static overhead coming with a user friendly implementation that is not present in the FlipCut implementation.

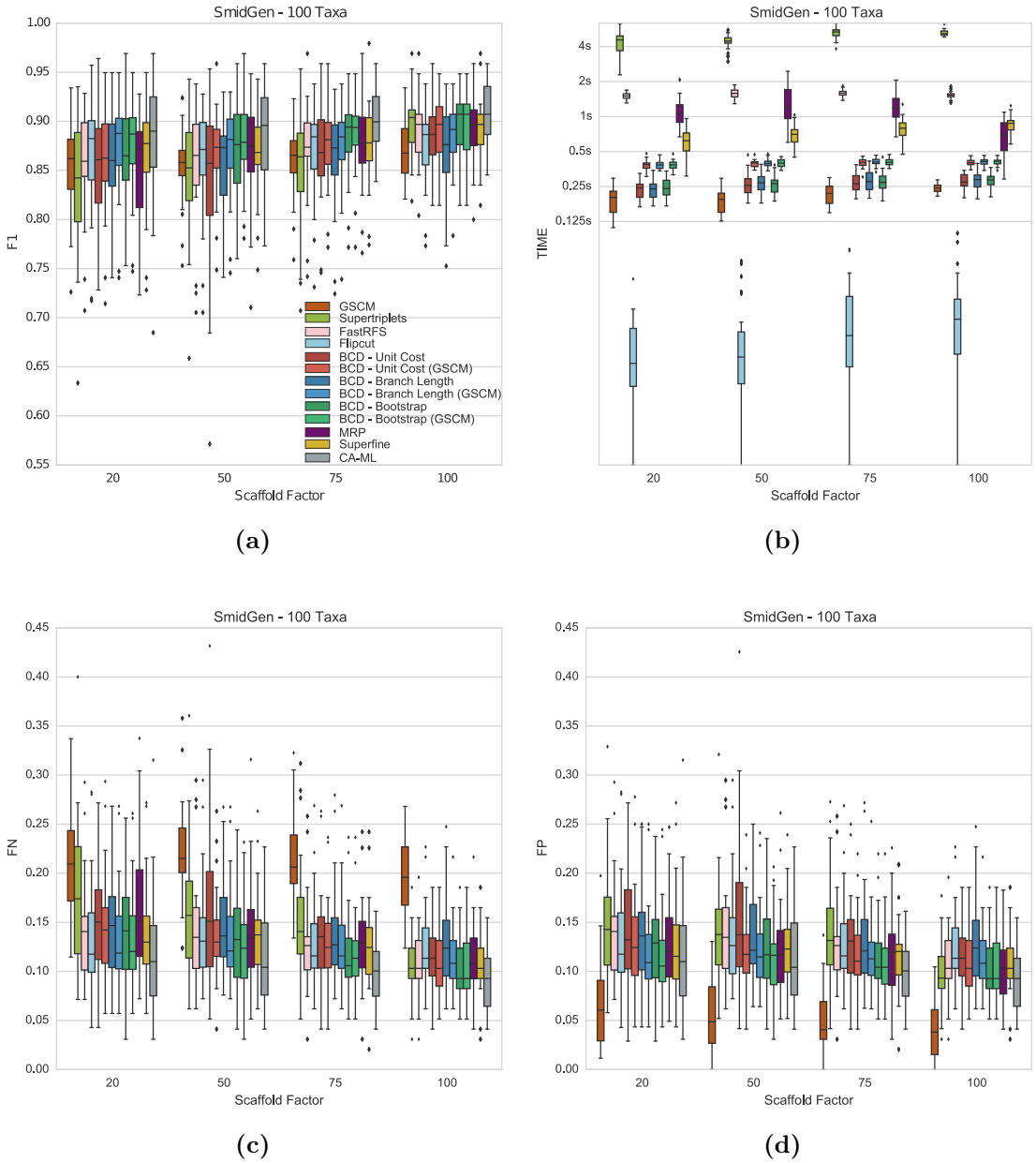


Figure A.20: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGen (100 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

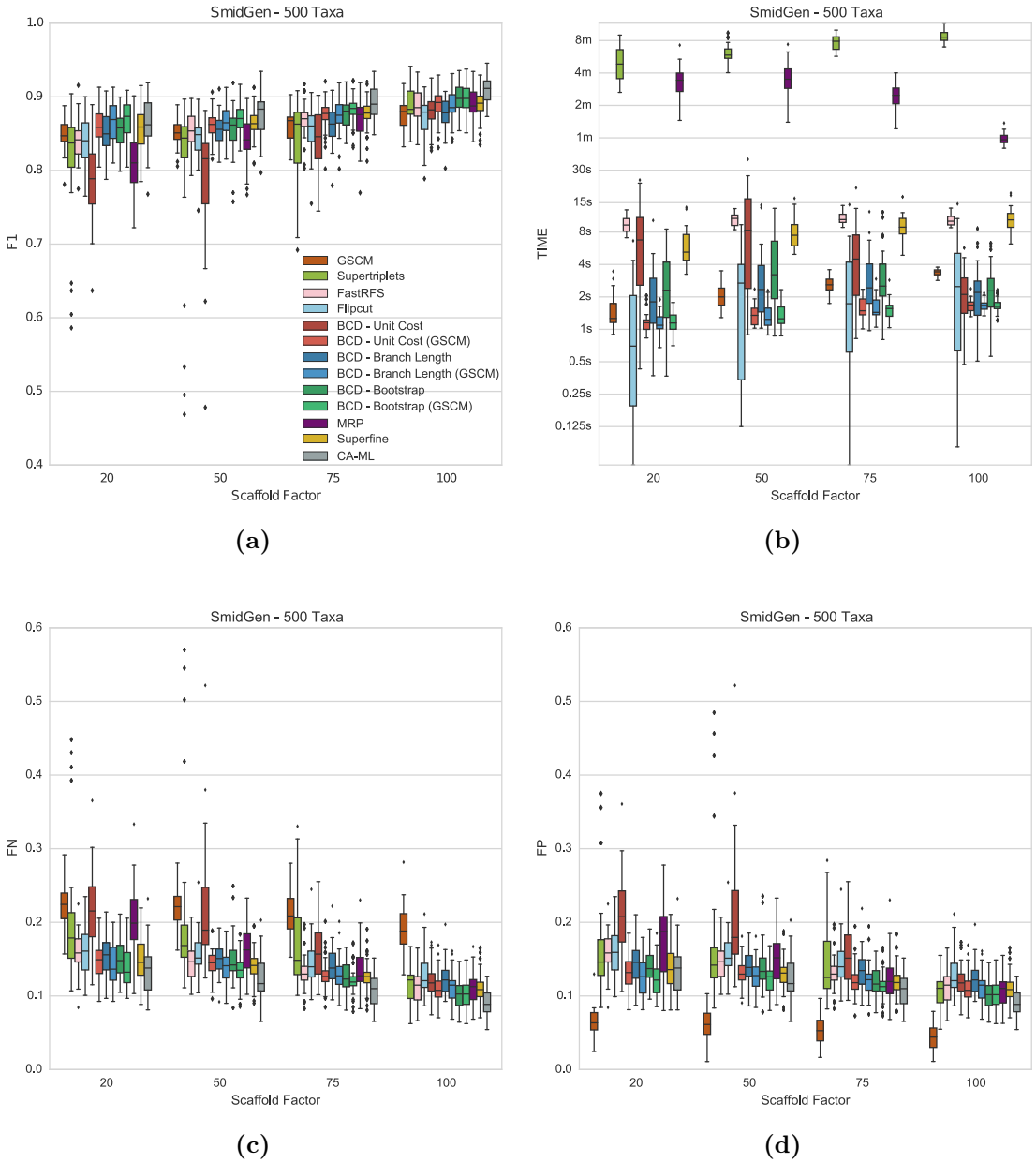


Figure A.21: F_1-score (a), running times (b), FN rate (c) and FP rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGen (500 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

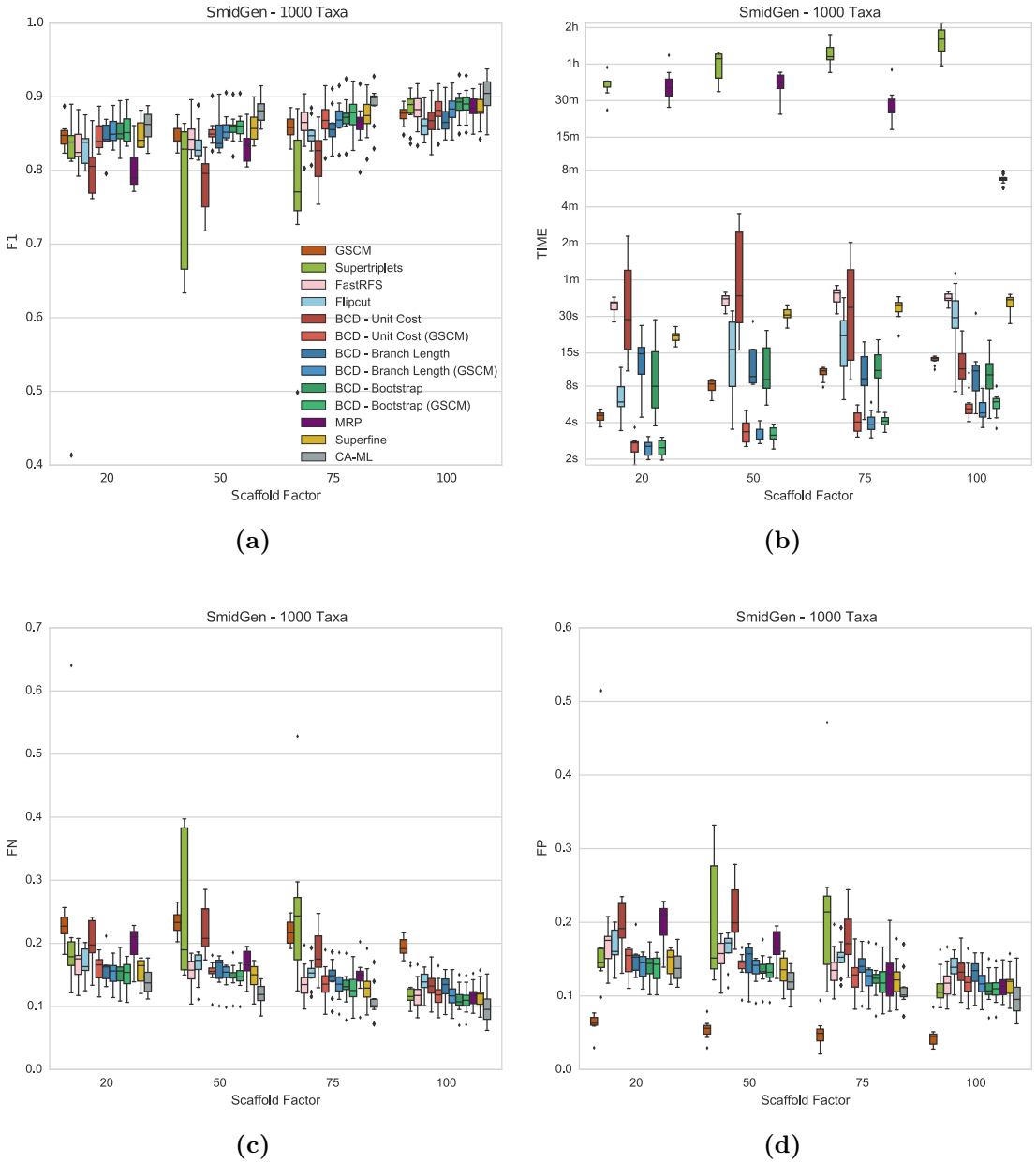


Figure A.22: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGen (1000 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

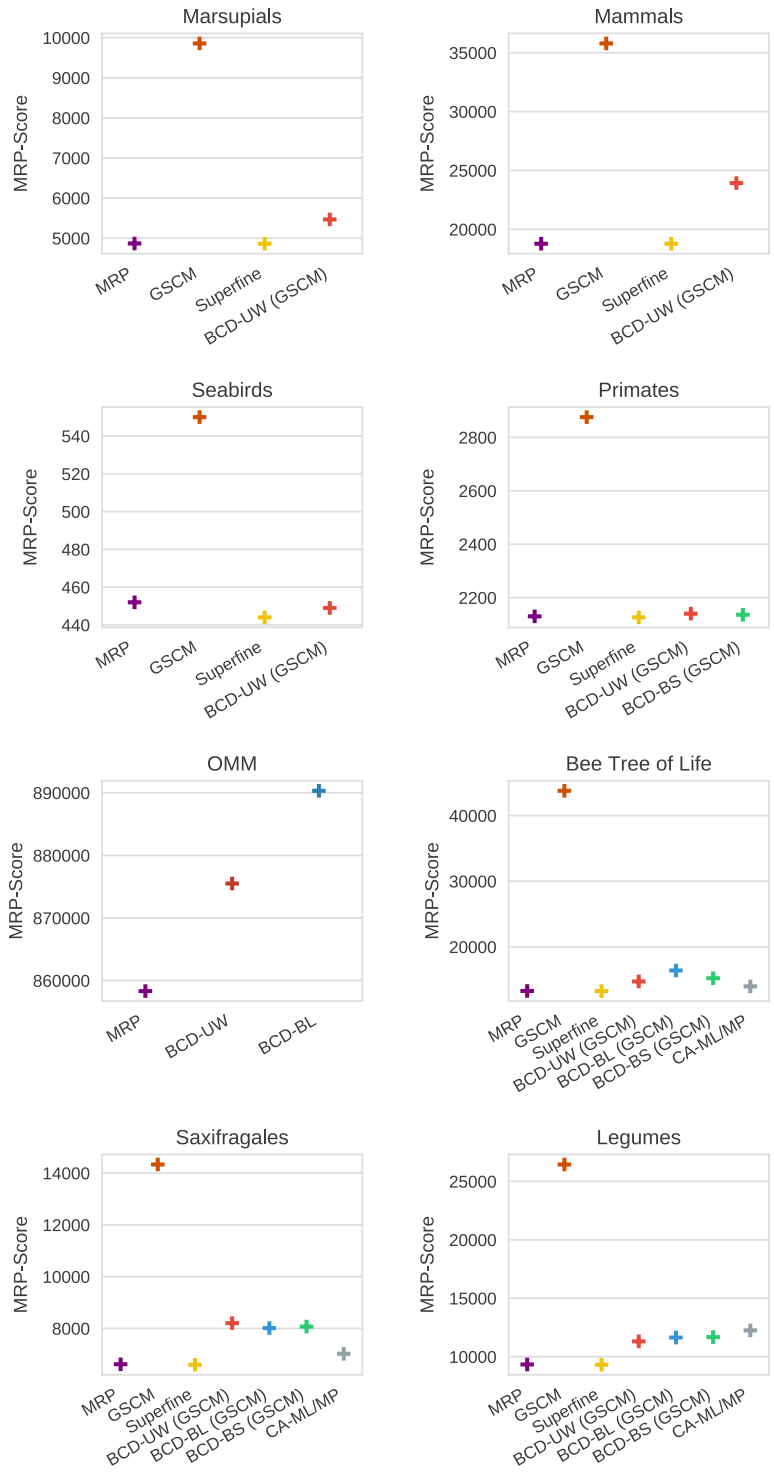


Figure A.23: MRP-Scores of supertrees against source trees on the biological supertree datasets. Most of the supertree datasets contain neither bootstrap values nor branch lengths, prohibiting the use of bootstrap values (BS) and branch length (BL) weighting. Combined Analysis cannot be applied to the supertree instances. Best rates marked in bold for each column.

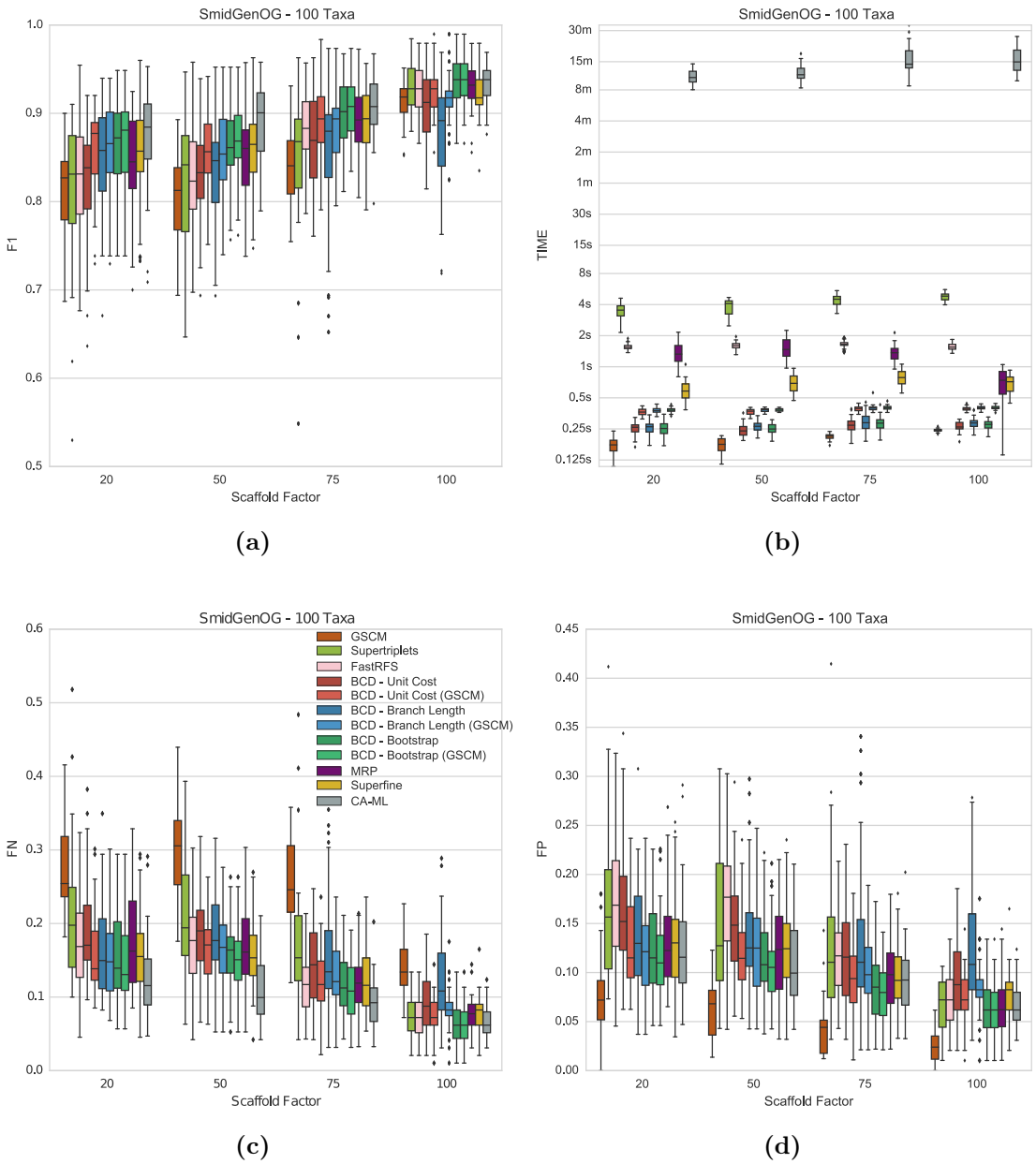


Figure A.24: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGenOG (100 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

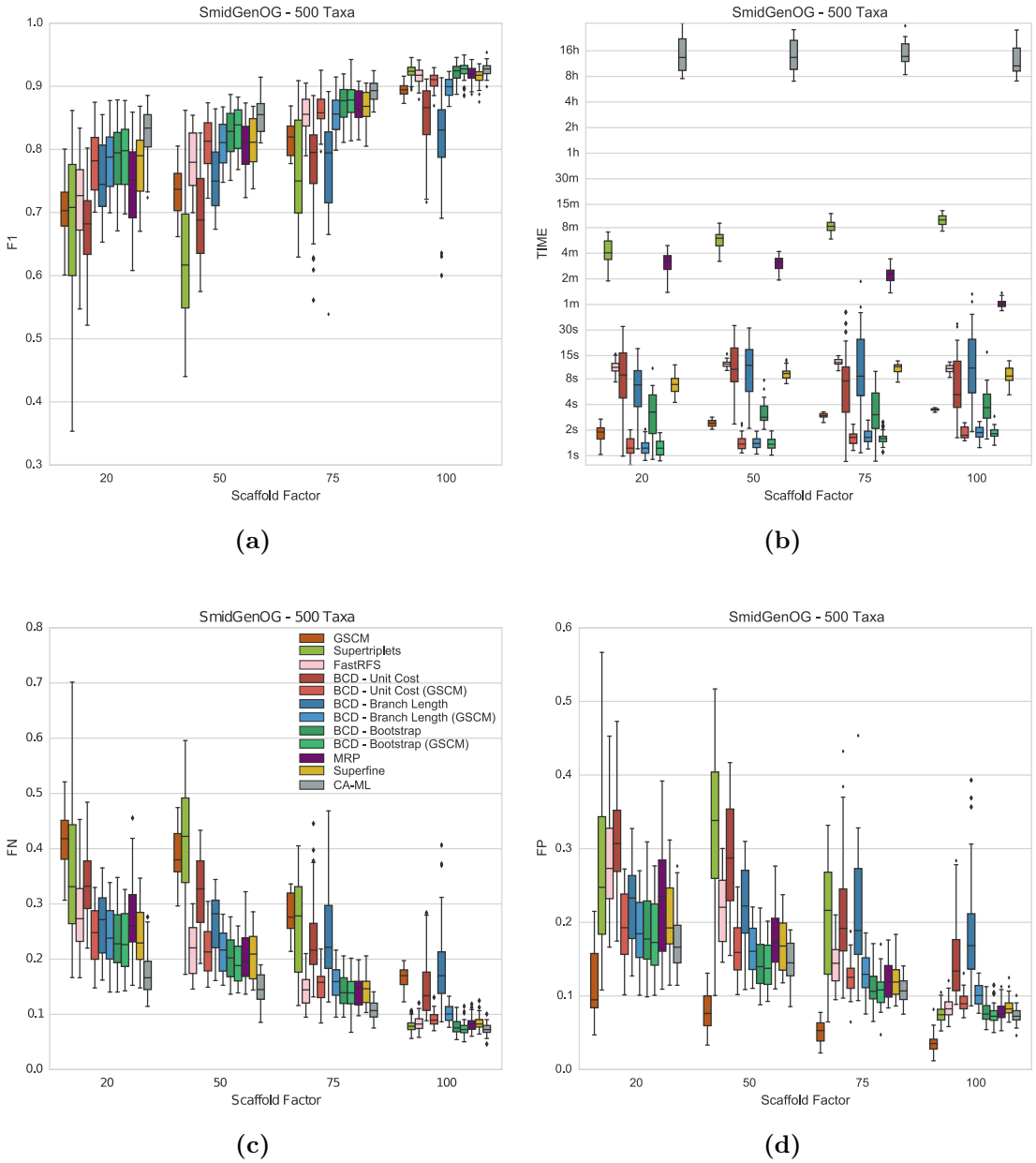


Figure A.25: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGenOG (500 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

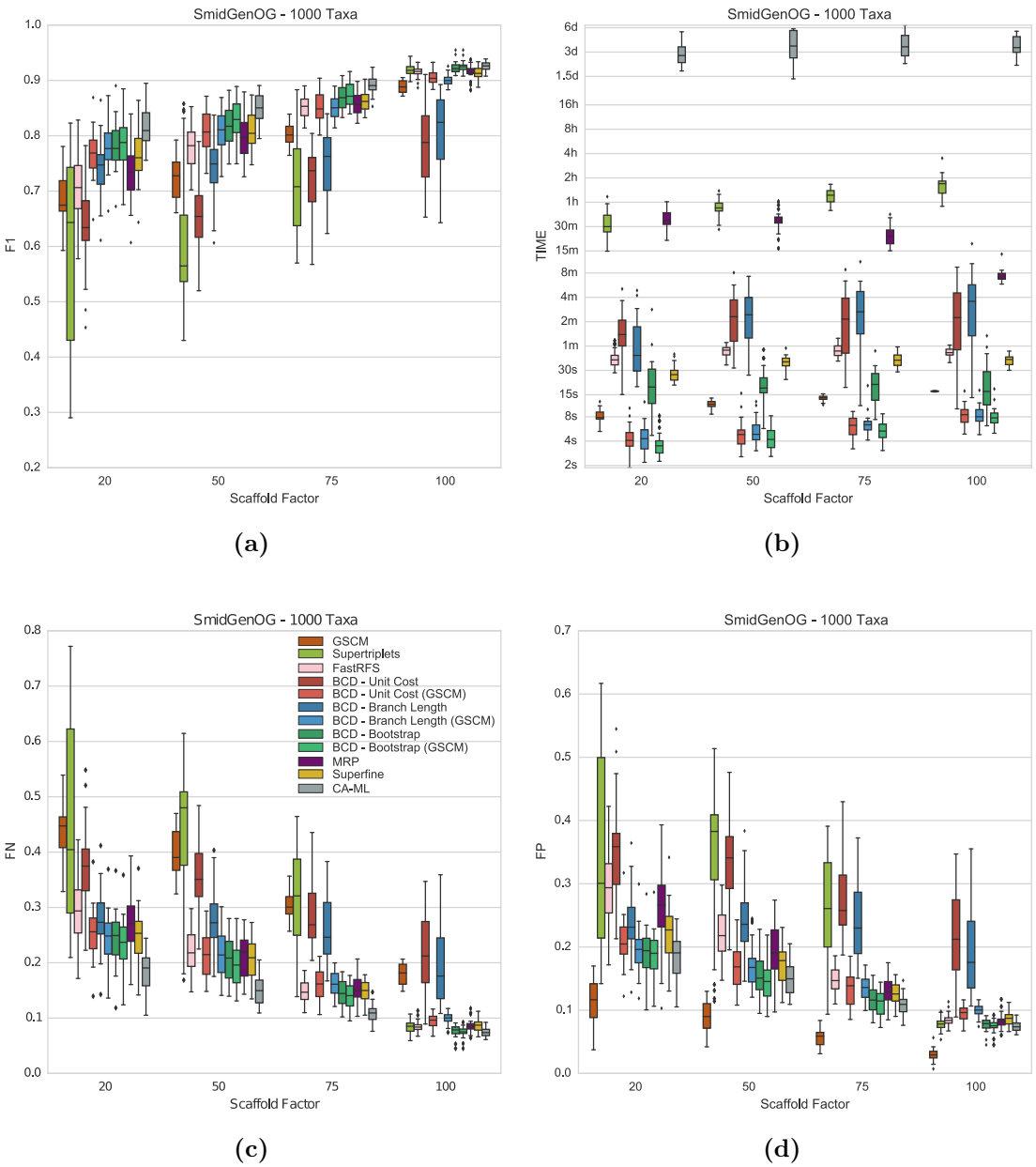


Figure A.26: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of the evaluated tree reconstruction methods on the simulated SMIDGenOG (1000 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

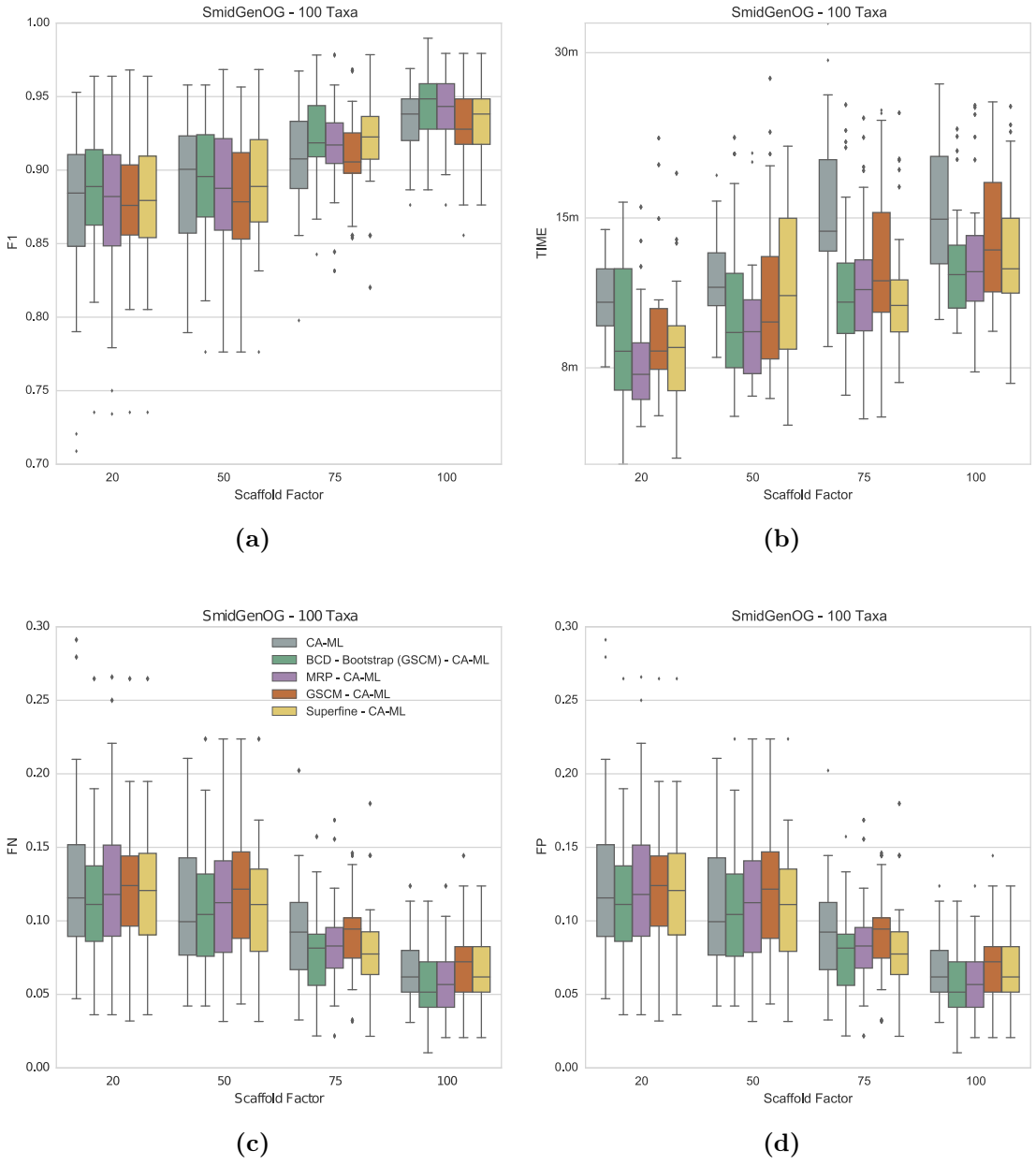


Figure A.27: Performance of different supertrees methods as a preprocessing step for a combined analysis with RAxML regarding F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) on the simulated SMIDGenOG (100 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

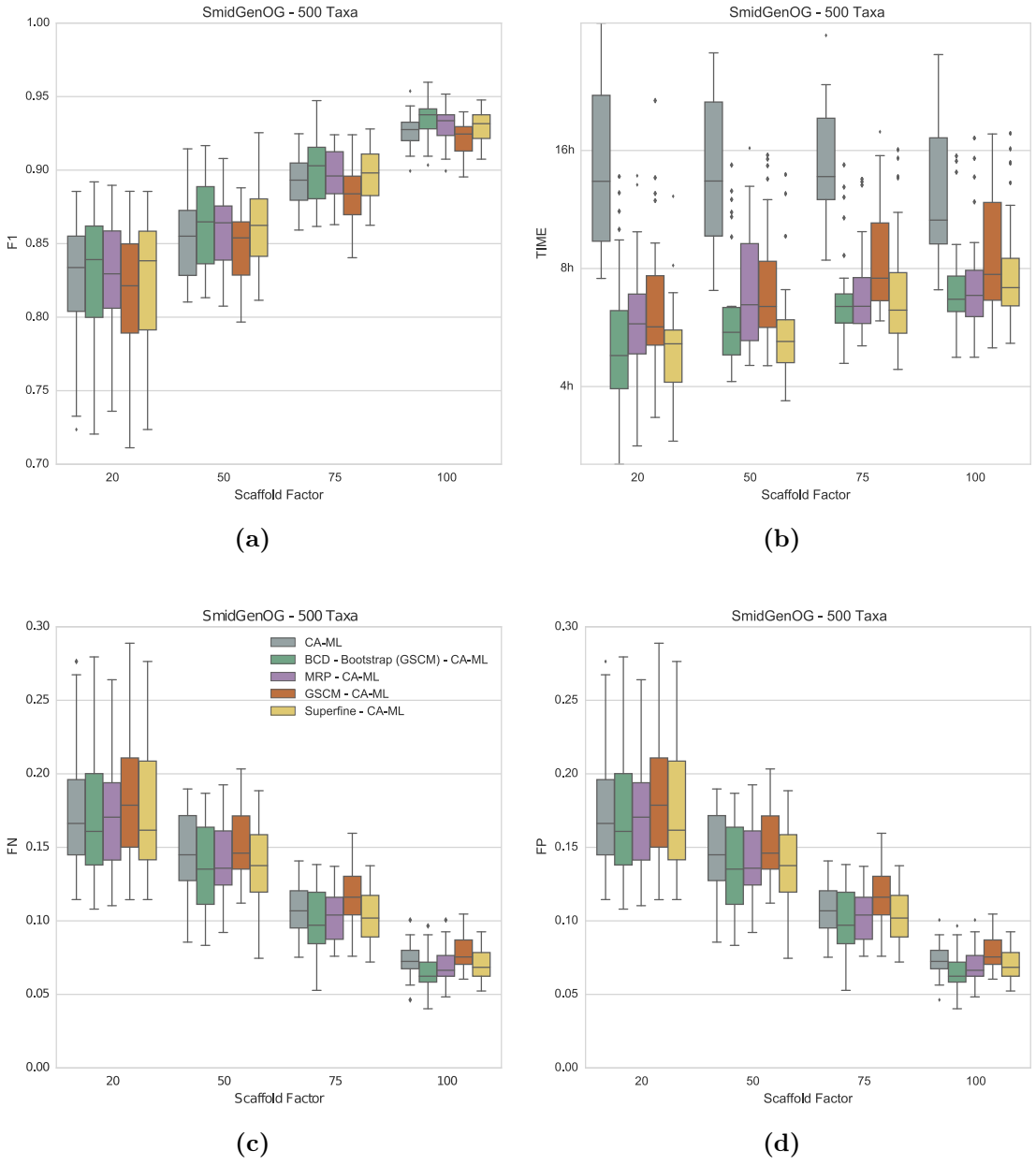


Figure A.28: Performance of different supertrees methods as a preprocessing step for a combined analysis with RAxML regarding F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) on the simulated SMIDGenOG (500 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

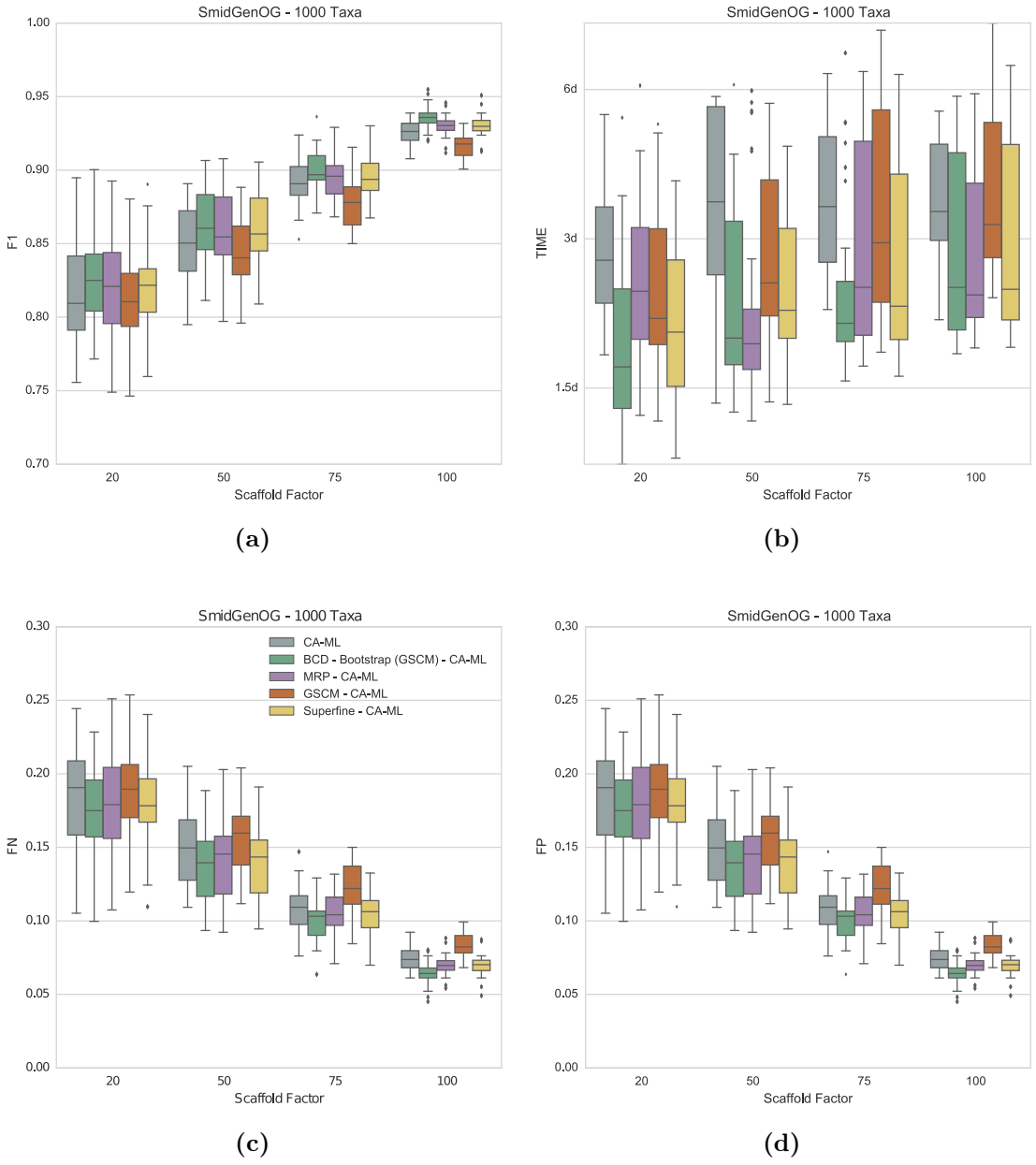


Figure A.29: Performance of different supertrees methods as a preprocessing step for a combined analysis with RAxML regarding F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) on the simulated SMIDGenOG (1000 Taxa) dataset. Error bars showing the standard mean error. Running times are shown in seconds on a logarithmic scale. Each dash on the ordinal x-axis shows the results for a different scaffold factor.

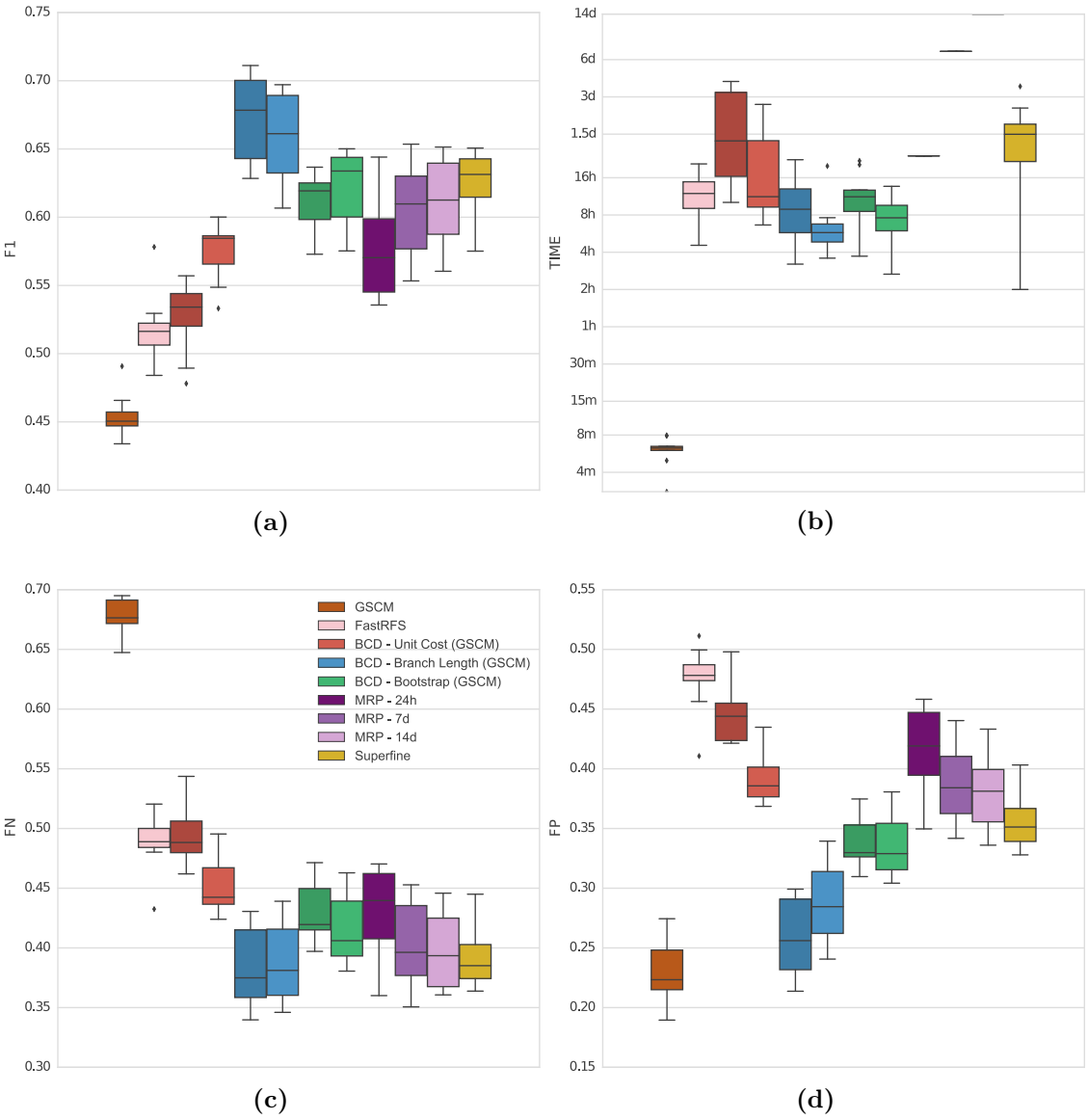


Figure A.30: F_1-score (a), running times (b), *FN* rate (c) and *FP* rate (d) of MRP, Superfine, GSCM, FastRFS and BCD on the simulated SMIDGenOG (5500 Taxa) dataset. Error bars showing the standard mean error. MRP did not finish on this dataset in reasonable time. We reported the results MRP reached after 1 day, 7 days and 14 days Running times are shown on a logarithmic scale.

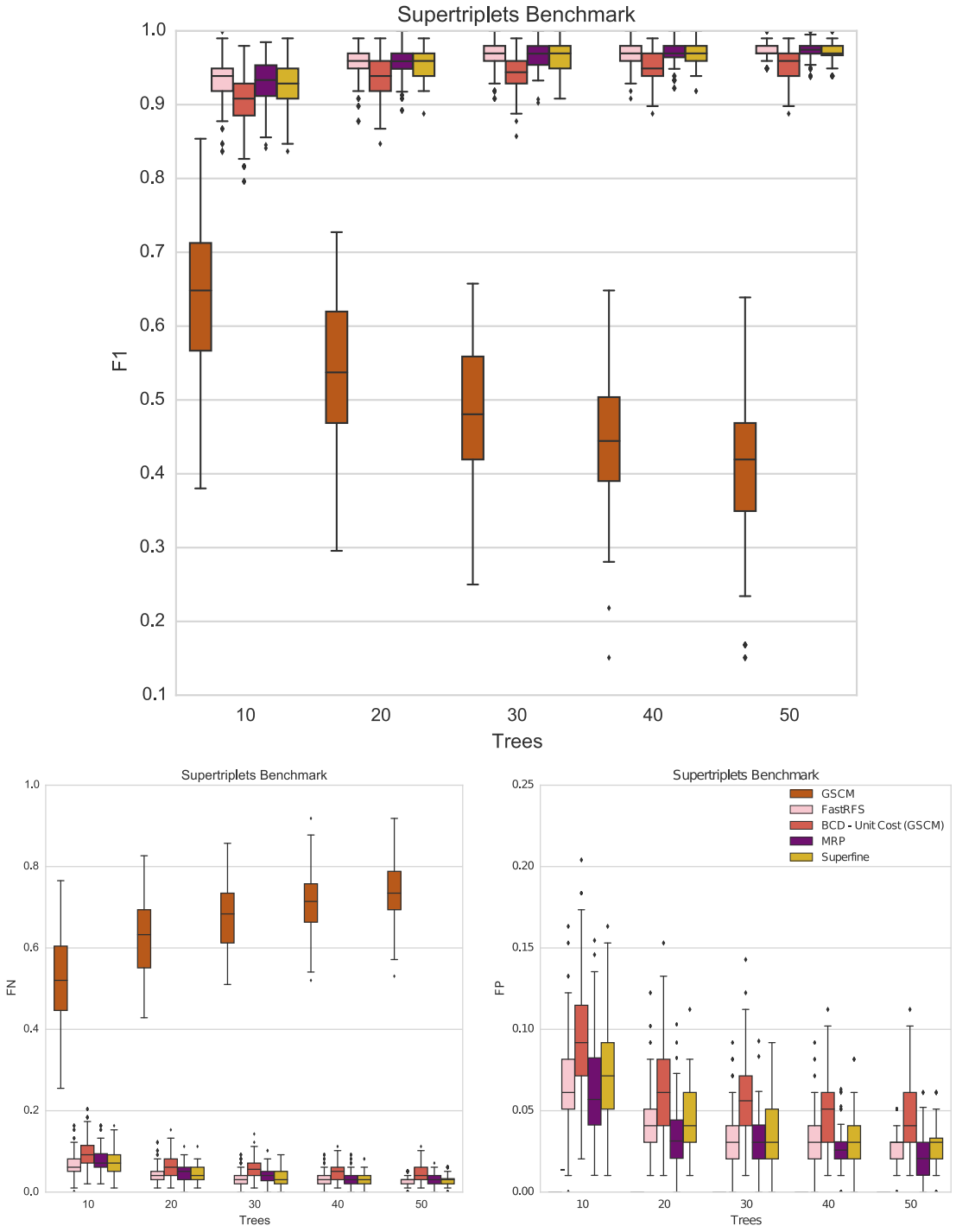


Figure A.31: F_1-score (above), *FN* rate (left) and *FP* rate (right) of MRP, Superfine, GSCM, FastRFS and BCD (Unit Weight only) on the simulated Supertriplet Benchmark (25% data deletion rate in the source trees) dataset. Error bars showing the standard mean error. Each dash on the ordinal x-axis shows the results for a different number of source trees.

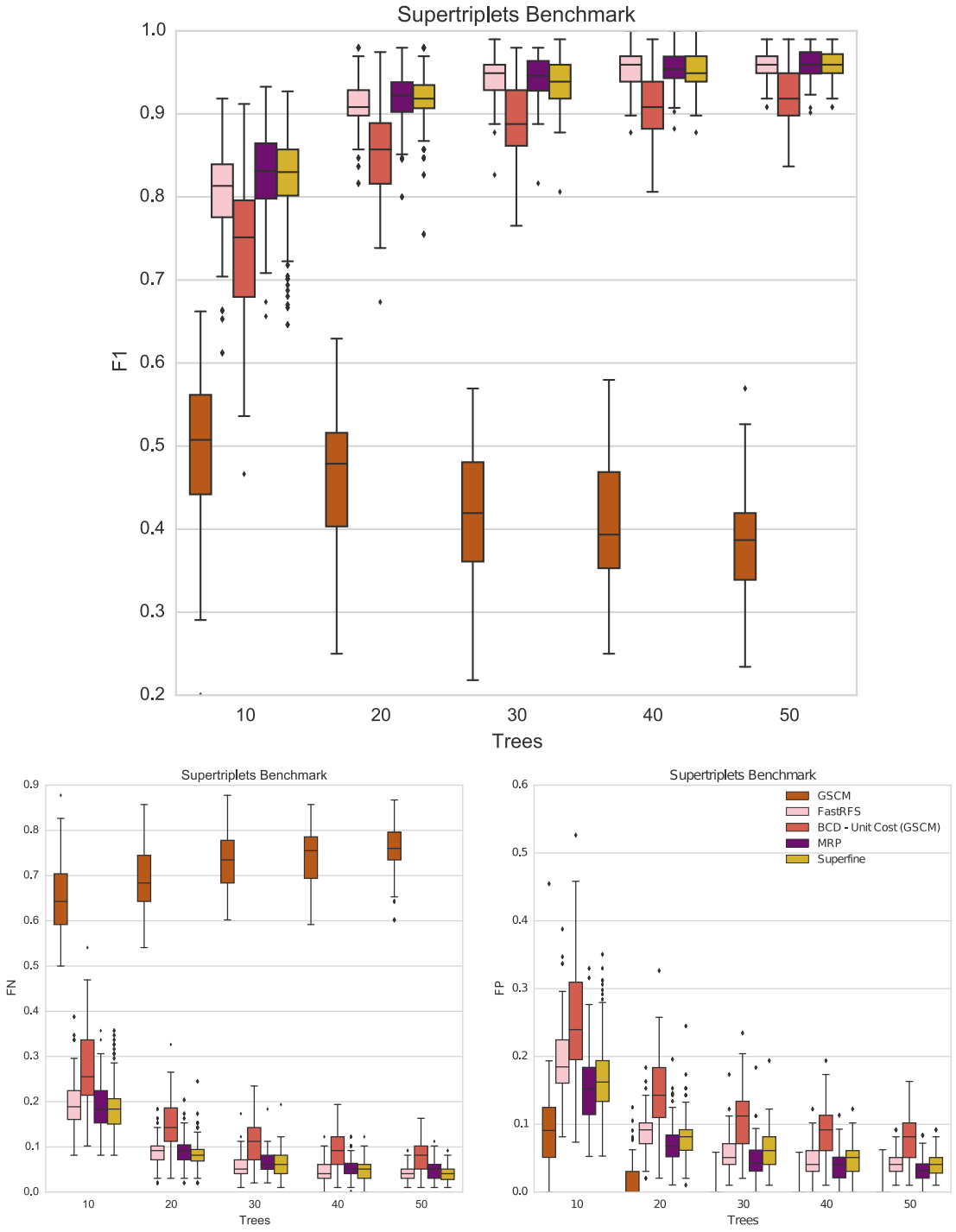


Figure A.32: F_1-score (above), *FN* rate (left) and *FP* rate (right) of MRP, Superfine, GSCM, FastRFS and BCD (Unit Weight only) on the simulated Supertriplet Benchmark (50% data deletion rate in the source trees) dataset. Error bars showing the standard mean error. Each dash on the ordinal x-axis shows the results for a different number of source trees.

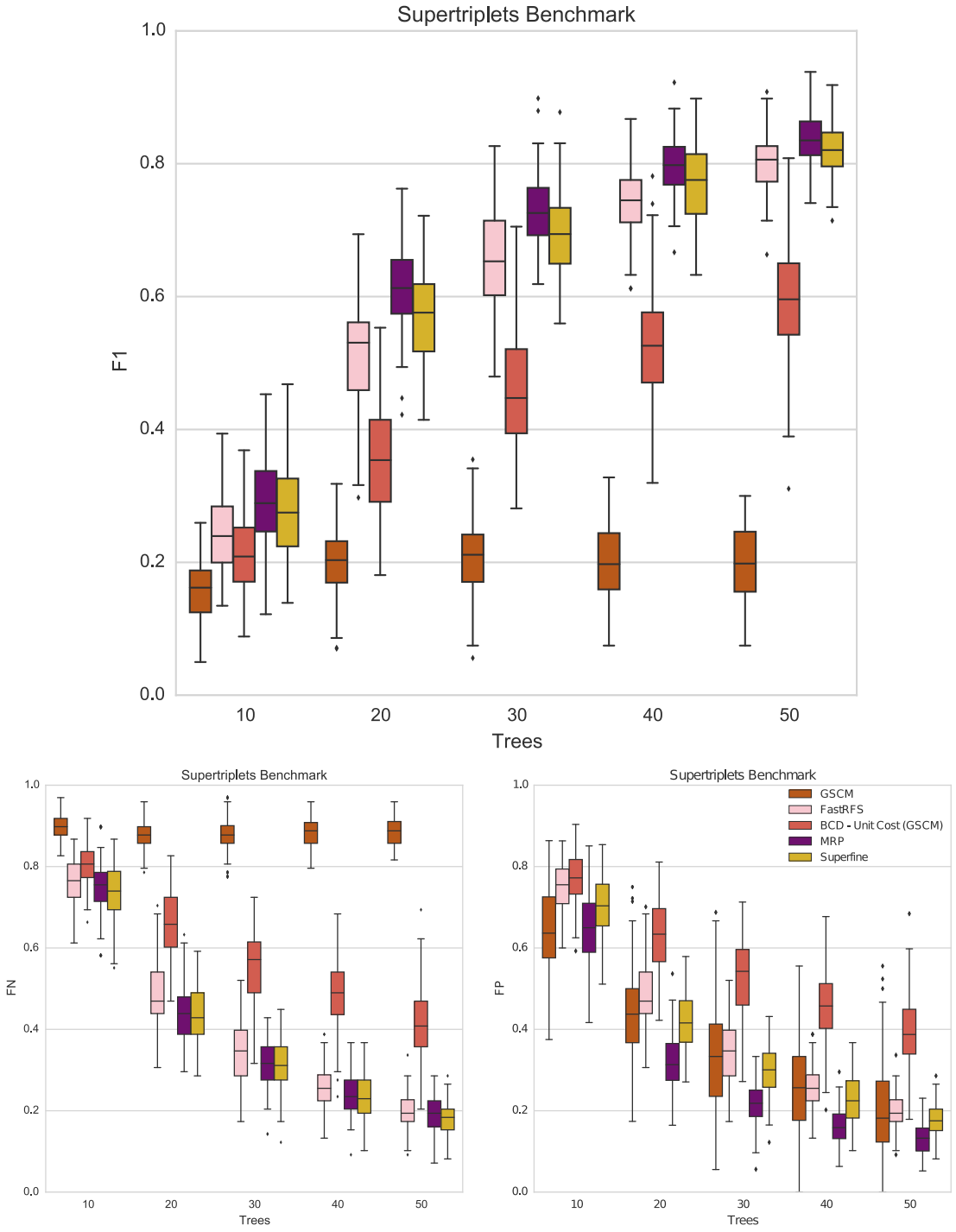


Figure A.33: F_1-score (above), *FN* rate (left) and *FP* rate (right) of MRP, Superfine, GSCM, FastRFS and BCD (Unit Weight only) on the simulated Supertriplet Benchmark (75% data deletion rate in the source trees) dataset. Error bars showing the standard mean error. Each dash on the ordinal x-axis shows the results for a different number of source trees.

Availability

All presented methods regarding the GSCM are implemented in Java as a command line tool which is available online[1]. The source code is available on GitHub[2]. The BCD algorithm (including the Beam Search and GSCM preprocessing) has been implemented as Java command line tool which is available online[3]. The software is open source and the code is available on GitHub[4]. All Datasets used for the evaluations are also available online[5].

Test environment

To calculate SuperFine and GSCM trees we use the `ReUP`[6] (version 1.0) implementation by Swenson *et al.* [173]. FastRFS was taken from Vachaspati and Warnow [178][7]. We have done the MRP and parsimony score calculations using `PAUP*` (version 4.0b10) [174]. ML trees and log-likelihood scores were calculated using `RAxML` (version 7.7.5) [161]. All running times were measured on an Intel XEON E5-2630 CPU at 2.3 GHz with 64 GB of memory.

[1] `https://bio.informatik.uni-jena.de/software/gscm/`
[2] `https://github.com/boecker-lab/gscm-supertrees`
[3] `https://bio.informatik.uni-jena.de/software/bcd/`
[4] `https://github.com/boecker-lab/bcd-supertrees`
[5] `https://bio.informatik.uni-jena.de/data/`
[6] `https://github.com/dtneves/SuperFine`
[7] `https://github.com/pranjalv123/FastRFS`

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.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts haben mich folgende Personen unterstützt: Sebastian Böcker und Franziska Hufsky.

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

Jena, den 31. Mai 2018

Markus Fleischauer