# FlipCut Supertrees: Towards Matrix Representation Accuracy in Polynomial Time

Malte Brinkmeyer, Thasso Griebel, and Sebastian Böcker

Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany, `sebastian.boecker@uni-jena.de`

**Abstract.** In computational phylogenetics, supertree methods provide a way to reconstruct larger clades of the *Tree of Life*. The supertree problem can be formalized in different ways, to cope with contradictory information in the input. In particular, there exist methods based on encoding the input trees in a matrix, and methods based on finding minimum cuts in some graph. Matrix representation methods compute supertrees of superior quality, but the underlying optimization problems are computationally hard. In contrast, graph-based methods have polynomial running time, but supertrees are inferior in quality.
In this paper, we present a novel approach for the computation of supertrees called FlipCut supertree. Our method combines the computation of minimum cuts from graph-based methods with a matrix representation method, namely Minimum Flip Supertrees. Here, the input trees are encoded in a 0/1/?-matrix. We present a heuristic to search for a minimum set of 0/1-flips such that the resulting matrix admits a directed perfect phylogeny. We then extend our approach by using edge weights to weight the columns of the 0/1/?-matrix.
In our evaluation, we show that our method is extremely swift in practice, and orders of magnitude faster than the runner up. Concerning supertree quality, our method is sometimes on par with the "gold standard" Matrix Representation with Parsimony.[1]

Preprint of: Malte Brinkmeyer, Thasso Griebel, and Sebastian Böcker.
FlipCut Supertrees: Towards Matrix Representation Accuracy in Polynomial Time.
*Algorithmica*, 2012. In press, doi:10.1007/s00453-012-9698-3.
The final publication is available at springerlink.com.

## 1 Introduction

When studying the relationship and ancestry of current organisms, discovered relations are usually represented as phylogenetic trees: These are rooted trees where each leaf corresponds to a group of organisms, called *taxon*. Inner vertices represent hypothetical last common ancestors of the organisms located at the leaves of its subtree. Supertree methods assemble phylogenetic trees with non-identical but overlapping taxon sets, into a larger supertree that contains all taxa of every input tree and describes the evolutionary relationship of these taxa. Constructing a supertree is easy if no contradictory information is encoded in the input trees [1]. The major problem of supertree methods is dealing with incompatible data in a reasonable way. It is understood that incompatible input trees are the rule rather than the exception in application.

Current supertree methods can roughly be subdivided into two major families: matrix representation (MR) methods, and graph-based methods with polynomial running time. The former encode inner vertices of all input trees as partial binary characters in a matrix, which is then analyzed using an optimization or agreement criterion to yield the supertree.

[1] A preliminary version of this paper appeared under the title "FlipCut Supertrees: Towards Matrix Representation Accuracy in Polynomial Time" in the Proceedings of the 17th International Computing and Combinatorics Conference, COCOON 2011, in: LNCS, vol. 6842, Springer, pp. 37–48.

In 1992, Baum [2] and Ragan [28] independently proposed the matrix representation with parsimony (MRP) method as the first matrix representation method. MRP is by far the most widely used supertree method today, and constructed supertrees are of comparatively high quality. Other variants have been proposed using different optimization criteria, such as matrix representation with flipping (MRF) [10] and matrix representation with compatibility [33]. All MR methods have in common that the underlying optimization problems are NP-hard [10,12,14,33]. Heuristic search strategies have to be used, but still, running times of MR methods can be prohibitive for large datasets. Weighted MRP [31] appears to outperform MRP with regards to supertree quality, but requires even higher running times [38]. Recently, Ranwez *et al.* [30] presented *SuperTriplets*, a local search heuristic based on triplet dissimilarity and triplet matrix encoding.

A particular matrix representation supertree method is "matrix representation with flipping": Here, the rooted input trees are encoded in a matrix with entries '0', '1', and '?' [10]. Utilizing the parsimony principle, MRF seeks the minimum number of "flips" $0 \to 1$ or $1 \to 0$ in the input matrix that make the resulting matrix consistent with a phylogenetic tree, where '?'-entries can be resolved arbitrarily. Evaluations indicate that MRF is on par with the "gold standard" MRP [9].

Graph-based methods make use of a graph to encode the topological information given by the input trees. This graph is used as a guiding structure to build the supertree top-down from the root to the leaves. The first graph-based supertree method was the Build algorithm [1] and its phylogenetic variant OneTree [8]. These algorithms are only applicable to non-conflicting input trees, and, thus only of limited use in practice [3]. This led to the development of the MinCut (MC) supertree algorithm [35] and a modified version, Modified MinCut (MMC) supertrees [25]. In fact, MinCut supertrees have already been been suggested by Gasieniec *et al.* [15, 16] back in 1997. MC and MMC construct a supertree even if the input trees are conflicting. All these methods share the advantage of polynomial running time, what results in swift computations in applications. On the downside, supertrees constructed by both MC and MMC are consistently of inferior quality compared to those constructed using MR methods [7].

Another graph-based method is *PhySIC* [29], a so-called veto supertree method. A drawback of veto methods is that they tend to produce unresolved supertrees in case of highly conflicting and/or poorly overlapping input trees. *PhySIC_IST* [34] tries to overcome this drawback by computing non-plenary supertrees: The supertree does not necessarily contain all taxa from the input trees. The Build With Distances algorithm (BWD) [40] is the first graph-based method that uses branch length information from the input trees to build the supertree. It also generalizes the Build algorithm but uses branch lengths to find better vertex partitions in the Build graph. Simulations indicate BWD supertrees are of much better quality than MC and MMC supertrees, but results are not on par with MRP [7].

To build even larger portions of the tree of life, a promising approach is the use of supertree methods as part of divide-and-conquer meta techniques, as pioneered by the disk-covering method [22, 23, 32]. Here, we break down a large phylogenetic problem into smaller subproblems that are computationally easier to solve, because of the lower number of taxa and the smaller evolutionary distance between them. The subproblem results are combined using a supertree method. The supertree can then be improved upon using local search heuristics and the complete dataset. By using preferably fast (polynomial-time) and accurate supertree methods "a divide-and-conquer strategy promises gains in both accuracy and speed compared to a conventional phylogenetic analysis" [4].

In this paper, we concentrate on the matrix representation with flipping framework. Recall that the problem is NP-hard [10], and only little algorithmic progress has been made towards its solution. We can test whether an MRF supertree instance admits a perfect phylogeny *without flipping* in time $O(mn \log^2(m+n))$ [26]. The MRF supertree problem is W[2]-hard and has no constant factor approximation unless P = NP [5]. Chen *et al.* [9] present a heuristic for MRF supertrees based on branch swapping. Chimani *et al.* [11] introduce an Integer Linear Program to find exact solutions, which is limited to relatively small and "simple" instances that do not require too many flips.

*Our contributions.* Here, we present a novel algorithm, named FlipCut, based on minimizing the number of 0/1-flips in the matrix representation. Our algorithm constructs the phylogenetic tree top-down, minimizing in each step the number of required flips. Running time of our algorithm is comparable to that of the MinCut algorithm: For $n$ taxa and $m$ internal nodes in the input trees, running time is $O(mn^3)$. We show that our method usually outperforms all other polynomial supertree methods with regards to supertree quality. In contrast to MinCut supertrees, our results are interpretable in the sense that we try to minimize a global objective function, namely the number of flips in the input matrix. We argue that this will allow us to further improve the quality of supertrees constructed using the FlipCut algorithm in the future.

## 2 Preliminaries

Let $n$ be the number of taxa in our study; for brevity, we assume that our set of taxa equals $\{1,\dots,n\}$. In this paper, we assume all trees to be *rooted phylogenetic trees*: The leaves of the trees are (labeled with) taxa from $\{1,\dots,n\}$, no taxon appears twice in a a tree, and there exist no vertices with out-degree one. If there are unrooted trees in the input set, each such tree has to be rooted using an outgroup. In this case, branch lengths (see Sec. 5) of edges incident to the root can be ignored. We are given a set of input trees $T_1,\dots,T_l$ with leaf set $\mathcal{L}(T_i) \subseteq \{1,\dots,n\}$. We assume $\bigcup_i \mathcal{L}(T_i) = \{1,\dots,n\}$. We search for a *supertree* $T$ of these input trees, that is, a tree with leaf set $\mathcal{L}(T) = \{1,\dots,n\}$. For $Y \subseteq \mathcal{L}(T)$ we define the *induced subtree* $T|_Y$ of $T$ where all internal vertices with degree two are contracted. Some tree $T$ *refines* $T'$ if $T'$ can be reached from $T$ by contracting internal edges. We say that a supertree $T$ of $T_1,\dots,T_l$ is a *parent tree* if $T|_{\mathcal{L}(T_i)}$ refines $T_i$, for all $i = 1,\dots,l$. In this case, $T_1,\dots,T_l$ are called *compatible.*

To cope with incompatibilities in the input, we employ the framework of Flip Supertrees: We encode the input trees in a matrix $M$ with elements in $\{0,1,?\}$, where rows correspond to taxa. Each inner vertex (except the root) in each input tree is encoded in one column of the matrix: Entry '1' indicates that the corresponding taxon is a leaf of the subtree rooted in the inner node, whereas all other taxa in the tree are encoded '0'. The state of taxa that are not part of the input tree is represented by a question mark ('?'). Columns of the matrix are called *characters*, and we assume that the set of characters equals $\{1,\dots,m\}$. See Sec. 3 for details.

The classical *perfect phylogeny* model [41] assumes that the matrix $M$ is binary, and that there exists an ancestral species that possesses none of the characters, corresponding to a row of zeros. This is sometimes referred to as *directed* perfect phylogeny. Further it is assumed that each transition from '0' to '1' happens at most once in the tree: An invented character never disappears and is never invented twice. According to the perfect phylogeny model, $M$ *admits a perfect phylogeny* if there is a rooted tree with $n$ leaves corresponding to the $n$ taxa, where for each character $u$, there is an inner node $w$ of the tree such that $M[t, u] = 1$ holds if and only if taxon $t$ is a leaf of the subtree below $w$, for all $t$. Given an arbitrary binary matrix $M$, we may ask whether $M$ admits a perfect phylogeny. Gusfield [18] shows how to check if a matrix $M$ admits a perfect phylogeny and, if possible, constructs the corresponding phylogenetic tree in time $\Theta(mn)$. There exist several characterizations for matrices that admit a perfect phylogeny, see for example [26]. One particularly important characterization is that any two characters (columns) of the matrix must be *compatible*: Let $A$ be the set of all taxa with entry '1' in the first character column, and let $B$ be the set of all taxa with entry '1' in the second character column, then $A \subseteq B$ or $A \supseteq B$ or $A \cap B = \emptyset$ must hold.

We now ask whether a matrix with '?'-entries allows for a perfect phylogeny, where '?'-entries can be arbitrarily resolved to '0' or '1'. Interestingly, this can also be decided in $\tilde{O}(mn)$ time [26]. (As usual, the $\tilde{O}(\cdot)$ notation suppresses all poly-log factors, see also Lemma 2 below.) To resolve incompatibilities among the input trees, the Flip Supertrees model assumes that the matrix $M$ is perturbed. We search for a perfect phylogeny matrix $M^*$ such that the number of entries where one matrix $M, M^*$ contains a '0' and the other matrix a '1', is minimal. This is the number of *flips* required to correct the input matrix $M$, also referred to as the *cost* of the instance.

**Minimum Flip Supertree (MFST) problem.** Given a matrix $M$ with entries in $\{0, 1, ?\}$ and an integer $k \geq 0$, decide whether $M$ allows for a perfect phylogeny with at most $k$ flips, where '?'-entries can be arbitrarily resolved to '0' or '1'.

Unfortunately, this is an NP-complete problem, even for an input matrix without '?'-entries [10]. When '?'-entries can be present in the input matrix, the problem is W[2]-hard and has no constant factor approximation unless P = NP [5].

In this paper, we deal with three types of graph-theoretical objects: namely, phylogenetic trees, graphs that we search for minimum cuts, and networks that we search for maximum flows. For readability, vertices of a tree will be called *nodes*, whereas directed edges of a network will be referred to as *arcs*.

To evaluate the quality of our supertrees, we use different measures: Each internal node of a rooted tree $T$ induces a cluster $Y \subseteq \mathcal{L}(T)$. The *Robinson-Foulds* (RF) symmetric distance between two trees $T, T'$ is the number of clusters induced by one tree but not the other, divided by the number of clusters induced by both trees. Another score between trees $T, T'$ is the *maximum agreement subtree* (MAST). This is a subset of leaves $Y \subseteq \mathcal{L}(T) = \mathcal{L}(T')$ of maximum cardinality such that $T|_Y = T'|_Y$ holds. The MAST distance of $T, T'$ then equals $1 - |Y| / |\mathcal{L}(T)|$. In both cases, we are using the normalized variant.

## 3 Matrix representation of a phylogenetic tree

The MinCut algorithm [35] as well as the Modified MinCut algorithm [25] construct supertrees by resolving conflicts in the input trees in a recursive top-down procedure. This has been adapted from the Build algorithm [1] that returns a supertree only if the input trees are compatible. A related algorithm was given by Pe'er *et al.* [26]. This algorithm tests whether an MFST instance $M$ allows for a perfect phylogeny without flipping, by resolving all '?'-entries. We will see below that this problem is basically equivalent to testing whether a set of input trees is compatible.

We can encode a tree $T$ with taxa set $\{1, \ldots, n\}$ in the input trees in a matrix $M(T)$ with elements in $\{0, 1\}$: Each row of the matrix corresponds to one taxon. For simplicity we assume that there is some natural ordering of the taxa and, hence, the rows. Each inner vertex except the root is encoded in one column of the matrix $M(T)$: Entry '1' indicates that the corresponding taxon is a leaf of the subtree rooted in the inner node, whereas all other taxa are encoded '0'. Compare to Section 17.3.5 in [19]. It is easy to see that for $m$ non-root inner vertices in $T$, the matrix representation $M(T)$ has size $m \times n$ and can be computed in $O(mn)$ time, using a tree traversal and lists of taxa. The correspondence between tree compatibility and their matrix representation follows from the next lemma:

**Lemma 1.** *Let $T$ and $T'$ be two trees with $\mathcal{L}(T) = \mathcal{L}(T')$. Then, $T'$ can be obtained from $T$ by contracting one of its interior edges if and only if $M(T')$ can be obtained from $M(T)$ by deleting one of its columns (and potentially reordering the remaining ones).*

We omit the simple proof. Hence, $T$ refines $T'$ if and only if $M(T')$ can be obtained from $M(T)$ by column deletion.

As indicated above, we can generalize the matrix encoding for a set of trees $\mathcal{T} = \{T_1, \ldots, T_l\}$ with taxa sets $\mathcal{L}(T_i) \subseteq \{1, \ldots, n\}$: We encode the input trees in a matrix $M(\mathcal{T})$ with elements in $\{0, 1, ?\}$. Again, each row of the matrix corresponds to one taxon. Each inner vertex except the root in each input tree is encoded in one column of the matrix $M(\mathcal{T})$: Entry '1' indicates that the corresponding taxon is a leaf of the subtree rooted in the inner node, whereas all other taxa of the input tree are encoded '0'. The state of taxa that are not part of the input tree is unknown, and represented by a question mark ('?'). Recall that columns of the matrix are called *characters*, and that we assume that the set of characters equals $\{1, \ldots, m\}$. Clearly, $m \leq l(n-2)$. In detail, $m$ is the total number of non-root inner vertices in $T_1, \ldots, T_l$. From the construction of $M(\mathcal{T})$, we infer that each column of the matrix contains a least one '0'-entry and at least two '1'-entries. Again, $M(\mathcal{T})$ has size $m \times n$ and can be computed in $O(mn)$ time, using a tree traversal and lists of taxa.

Now, it is straightforward to show that $\mathcal{T}$ has a parent tree if and only if $M(\mathcal{T})$ is an incomplete (directed) perfect phylogeny or, equivalently, an instance of the MFST problem that does not require flipping. So, we can use the algorithm of Pe'er *et al.* [26] to decide whether a set of input trees is compatible and, if so, construct the parent tree. This appears to be the fastest algorithm for computing the parent tree of a set of (unrestricted) input trees. For restricted cases such as binary input trees, faster algorithms are available [21].

**Lemma 2.** *Given a collection of input trees $\mathcal{T} = \{T_1, \ldots, T_l\}$ with taxa sets $\mathcal{L}(T_i) \subseteq \{1, \ldots, n\}$ for $i = 1, \ldots, l$. Then, we can decide whether $\mathcal{T}$ is compatible and, if so, construct a parent tree of $\mathcal{T}$ in time $O\left(mn \cdot \log^2(m+n)\right)$, where $m$ is the total number of non-root inner vertices in $T_1, \ldots, T_l$.*

So, we have reduced the problem of finding a parent tree for a given set of trees, to the MFST problem without flipping or, equivalently, the (directed) incomplete perfect phylogeny problem. But the converse is also true: An input matrix $M$ with $m$ columns can be transformed into $m$ input trees, where each column $c$ is transformed into a tree containing all taxa $t$ satisfying $M[t, c] \neq ?$, having a single non-trivial clade with those taxa $t$ satisfying $M[t, c] = 1$. Hence, these two problems are basically equivalent.

## 4 The FlipCut algorithm

We now show how to apply the idea of finding minimum cuts to the algorithm of Pe'er *et al.* [26]. For a subset $S \subseteq \{1, \ldots, n\}$ of taxa and a subset $D \subseteq \{1, \ldots, m\}$ of characters,

$$M = \left(\begin{array}{cccccc} 1&1&1&0&0&? \\ 1&1&1&1&0&0 \\ 0&0&1&1&1&? \\ 0&0&0&1&1&1 \\ 0&?&1&0&1&1 \end{array}\right) \begin{array}{c} \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \\ \text{e} \end{array}$$

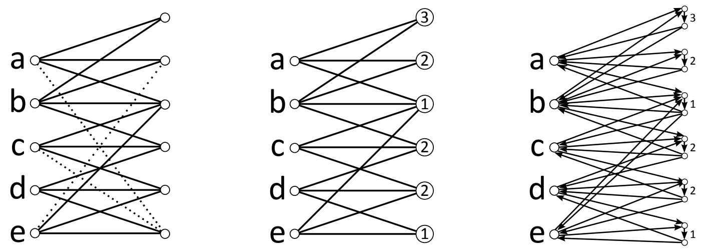


**Fig. 1.** Workflow of the FlipCut algorithm. From left to right: The input matrix $M$ and its graph representation, dotted lines correspond to '?'-entries; the corresponding FlipCut graph $G$ with vertex deletion costs; and the network $N$.

the *FlipCut graph* $G(S, D)$ 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, as all '?'-entries can be resolved to '1' without flipping [26]. The resulting graph is denoted $G(S, D)$.

The algorithm of Pe'er *et al.* proceeds as follows: We start with $S \leftarrow \{1, \ldots, n\}$ and $D \leftarrow \{1, \ldots, m\}$. We then construct the FlipCut graph $G(S, D)$. If this graph is connected, the algorithm terminates, as there is no perfect phylogeny resolving $M$. Otherwise, we recursively repeat for each connected component $S', D'$ of the FlipCut graph with $|S'| > 1$. In case the algorithm does not terminate early, then the sets $S'$ of taxa computed during the course of the algorithm, define the rooted phylogenetic tree.

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

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

Clearly, this problem is closely related to finding minimal cuts in an undirected graph. For the later problem, numerous efficient algorithms have been developed in the last years [6, 24]. Unfortunately, there exist two important differences here: First, we are not searching for an arbitrary cut in the graph $G(S, D)$ but instead, require that the set of taxa vertices is partitioned. Second, these algorithms do not allow us to delete vertices. We conjecture that the first modification is relatively easy to overcome. However, it is not obvious how to include vertex deletions in these algorithms.

To this end, we drop back to an older approach for finding minimum cuts: We fix one taxon vertex $s$, and for all other taxa vertices $t$ we search for a minimum $s$-$t$-cut, allowing vertex deletions. Among these cuts, the cut with minimal weight is the solution to the above problem. To find a minimum $s$-$t$-cut with vertex deletions, we transform $G(S, D)$

```
1: procedure FlipCut(set of taxa $S \subseteq \{1,\dots,n\}$, set of characters $D \subseteq \{1,\dots,m\}$)
2:     Output taxa set $S$
3:     if $|S'| = 1$ then return
4:     Construct $G := G(S,D)$ from $M$
5:     if $G$ is connected then
6:         Construct weighted network $N$ from $G$
7:         for all $t = 2,\dots,n$ do
8:             Find maximum 1-$t$-flow in $N$
9:             if maximum 1-$t$-flow is lighter than current minimum cut then
10:                Construct cut-of-the-phase in $G$ from maximum flow in $N$
11:                Store cut-of-the-phase as the current minimum cut
12:            end if
13:        end for
14:        Remove current minimum cut from $G$
15:    end if
16:    for all connected components of $G$ with vertex set $S', D'$ of $G$ do
17:        FlipCut($S', D'$)
18:    end for
19: end procedure
```

**Fig. 2.** Algorithm FlipCut with input $S, D$ and constant input matrix $M$ over $\{0,1,?\}$.

into a directed network $H(S,D)$ with capacities: Each taxa vertex $t$ is also a vertex in the network, each character vertex $c$ is transformed to 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 unit capacity. By the generalized min-cut max-flow theorem, finding a minimum cut in $G(S,D)$ is equivalent to computing a maximum flow in the network $H(S,D)$ [13]. Note that for all taxa $s,t$, the maximum $s$-$t$-flow in $H(S,D)$ equals the maximum $t$-$s$-flow. See Fig. 1 for an example. We reach:

**Lemma 3.** *Let $S \subseteq \{1,\dots,n\}$, $D \subseteq \{1,\dots,m\}$, and $M \in \{0,1,?\}^{m\times n}$. We construct the network $H := H(S,D)$ for the input matrix $M$. The minimum number of 0/1 flips required in $M$ to make the induced* FlipCut *graph $G(S,D)$ disconnected, equals the minimum cost of a minimum 1-$t$-cut in the network $H$, over all $t = 2,\dots,n$.*

We now proceed in a recursive top-down procedure to construct the supertree, similar to [25, 26, 35]. The pseudocode of our algorithm is depicted in Fig. 2; we initially call the procedure as FlipCut($\{1,\dots,n\}, \{1,\dots,m\}$). The subsets $S \subseteq \{1,\dots,n\}$ that are output during the course of the algorithm, form a hierarchy which can be transformed into the desired supertree.

As the algorithm reproduces that of Pe'er *et al.* in case the input trees are compatible or, equivalently, in case the input matrix allows for a perfect phylogeny without flipping, we infer:

**Lemma 4.** *In case the input matrix $M$ allows for a perfect phylogeny without flipping, then the* FlipCut *algorithm returns the perfect phylogeny tree.*

What is the running time of the above algorithm? At most $n-1$ minimum cuts have to be computed in total, as this is the number of inner nodes in the resulting phylogenetic tree. We reach a running time of $O(n \cdot T(m,n))$ where $T(m,n)$ is the time required for computing all maximum 1-$t$-flows in the networks $H(S,D)$ with at most $m$ character vertices and $n$

taxa vertices. The running time is dominated by the algorithm we use for constructing maximum flows. For a network $H = (V, E)$, Hao and Orlin [20] compute maximum flows from one source to all other vertices in $O\big(|V| \cdot |E| \cdot \log(|V|^2 / |E|)\big)$ time, using the maximum flow algorithm of Goldberg and Tarjan. For a bipartite graph with vertex set $V_1 \cup V_2$ and $|V_1| \leq |V_2|$, running time can be improved to $O\big(|V_1| \cdot |E| \cdot \log(|V_1|^2 / |E|)\big)$ [20]. Our networks $H(S, D)$ are bipartite and have $O(n + m)$ vertices and $O(mn)$ edges, and we may assume $n \leq m$. So, a minimum cut with vertex deletions in $G(S, D)$ can be computed in $O(mn^2)$ time. We infer:

**Lemma 5.** *Given an input matrix $M$ over $\{0, 1, ?\}$ for $n$ taxa and $m$ characters, the* FlipCut *algorithm computes a supertree in $O(mn^3)$ time.*

As presented here, the FlipCut algorithm may compute different solutions for the same input: This is because there can be several co-optimal minimum cuts, and our algorithm arbitrarily chooses one of these cuts. We can solve this by removing all edges and vertices that are part of *at least one* minimum cut, similar to the MinCut algorithm [35]. We do not have to actually enumerate all such minimum cuts, what can be a hard problem, to find these edges and vertices [27]. In the following, we ignore this modification: We weight all edges with real numbers, so the existence of several minimum cuts of identical weight is practically impossible.

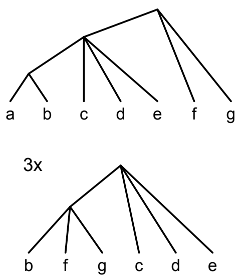
Steel *et al.* [37] list five desirable properties of a supertree method, and it is easy to see that the FlipCut algorithm satisfies three of them: If the input trees are compatible then the supertree is a parent tree (Lemma 4); the supertree can be computed in polynomial time (Lemma 5); and no species is missing from the supertree. With the modification indicated in the previous paragraph, we can also ensure that changing the order of input trees does not change the resulting supertree; and that relabeling the input species results in the same supertree with correspondingly relabeled species. We omit the simple technical details. Hence, the modified FlipCut algorithm satisfies all five desirable properties from [37].

## 5 Using branch lengths

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

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

The "Edge & Level" weighting does not cover all the information encoded in the input trees: As suggested by Willson [40], if taxa $t, t'$ have a last common ancestor much more recent than $t, t''$ then, even if this is observed in different input trees, we can still infer that $t, t'$ are more closely related to each other than to $t''$. In the future, we will investigate how to introduce this concept into the FlipCut framework, as better weightings will surely improve the performance of our method.



$$M = \begin{array}{c|c} \begin{pmatrix} 1\,1\,?\,?\,? \\ 1\,1\,1\,1\,1 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,1 \\ 0\,0\,1\,1\,1 \end{pmatrix} & \begin{matrix} \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \\ \text{e} \\ \text{f} \\ \text{g} \end{matrix} \end{array} \qquad M^* = \begin{array}{c|c} \begin{pmatrix} 1\,1\,0\,0\,0 \\ \mathbf{0}\,\mathbf{0}\,1\,1\,1 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,1\,1\,1 \\ 0\,0\,1\,1\,1 \end{pmatrix} & \begin{matrix} \text{a} \\ \text{b} \\ \text{c} \\ \text{d} \\ \text{e} \\ \text{f} \\ \text{g} \end{matrix} \end{array}$$

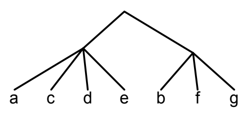


**Fig. 3.** The undisputed sibling problem. Left: Four input trees, the lower tree appears three times in the input. Taxon ‘a’ appears only in the upper tree, and is a sibling of ‘b’. Middle left: The resulting matrix representation $M$. Middle right: The optimal MRF solution $M^*$ with two flips. Right: The solution tree for $M^*$. Taxa ‘a’ and ‘b’ are no siblings in this tree.

## 6 The undisputed sibling problem

Given a set of input trees, assume that some taxon $x$ appears as a sibling of another taxon $y$ in all the input trees in which it is present at all. In other words, for all trees where $x$ is present, we also find $y$, and both are siblings. We call such $x$ an *undisputed sibling.* Then, it is reasonable to assume that $x$ is also a sibling of $y$ in the supertree, possibly accompanied by other siblings. Unfortunately, Flip Supertrees does not necessarily enforce this: Minimizing the number of flips, it is sometimes cheaper to separate $x$ and $y$. This is a seemingly rare but still undesirable effect of this objective function, see Fig. 3 for an example.

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

To counter the above effect, we use a data reduction rule that is applied to all input trees before we compute FlipCut supertrees: If there is an undisputed sibling $x$ of $y$, then remove $x$ from all input trees. We repeat this until we find no more undisputed siblings. Note that by removing an undisputed sibling, we might produce new undisputed siblings. After we have computed the supertree, we re-insert all undisputed siblings in reverse order. If $y$ has more than one undisputed sibling at the same time, we re-insert all siblings in one node, resulting in a polytomy in the supertree.

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

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

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

## 7 Experiments

We want to evaluate the performance of the FlipCut supertree method in comparison to Matrix Representation with Parsimony (MRP), Matrix Representation with Flipping (MRF), Build With Distances (BWD), *PhySIC_IST*, and *SuperTriplets*. Recall that MC and MMC supertrees are of comparatively low quality, and consistently worse than BWD [7], so we excluded these two methods from our study. We use simulated data in our evaluation since here, the *true tree* (or model tree) is known. Thus, results of different methods can be compared at an absolute scale. Our evaluation study proceeds in the usual fashion: A model tree is generated, and gene sequences are evolved along the branches. Sequences at the taxa of the model tree are used as datasets from which source trees for a supertree method are inferred. Finally, the resulting supertree is compared to the model tree using distance or similarity measures.

For our simulations, we used a dataset[2] that was generated using the SMIDGen protocol described in [38]. Compared to previous protocols, this protocol better reflects data collection processes used by systematists when gathering empirical data. This includes creation of densely-sampled clade-based trees as well as sparsely-sampled scaffold trees. Model trees having either 100, 500 or 1000 taxa were generated with 30 replicates for the 100 and 500 taxon case, and ten replicates for the 1000 taxa case. We defer the details to the appendix.

For the simulation study, we know that all branch lengths are computed under the same model of sequence evolution, so a normalization as proposed in Sec. 5 is not necessary. This can be seen as an optimal condition for the BWD and FlipCut algorithm. We defer the evaluation on whether branch length normalization changes the quality of reconstructed supertrees, to a later paper.

We implemented the FlipCut algorithm in Java as part of the EPoS framework [17]. In order to to illustrate the influence of branch-length to our approach, we use two different weighting schemes for edges and character vertices in the graph model: First, unit costs, where branch lengths are ignored. Here, the cost of deleting an edge is one, and the cost of deleting a character vertex $c$ is just the number of zeros in the corresponding column in matrix $M$. Second, "Edge & Level", where we make use of branch lengths. We multiply deletion costs for character vertex $c$ and all edges incident to $c$ by $w(c) = l(e) \cdot \mathit{depth}(v)$. Here, $l(e)$ is the length of branch $e = (u, v)$, $\mathit{depth}(v)$ is the number of edges on the path from the root to $v$, and $c$ corresponds to $v$.

MRP supertrees were computed using PAUP* 4.0b10 [39] with TBR branch swapping strategy, random addition of sequences, no limit on the maximal number of trees in memory, and 100 replicates. MRF supertrees were generated using the implementation provided by Duhong Chen[3], also with the TBR branch swapping strategy. For 100 taxa model trees, we used 30 replicates for the search, and in case of 500 and 1000 taxa model trees only ten replicates, because on our cluster the implementation failed with more replicates.

[2] `http://www.cs.utexas.edu/~phylo/datasets/supertrees.html`
[3] `http://genome.cs.iastate.edu/CBL/download/`

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

**Table 1.** Running times (min:sec) of the different algorithms. *MRP #TO* is the number of timeouts of MRP where computation was stopped after one hour, and *MRP avg** is the average running time of those runs that stopped before the time limit. *ST* is the SuperTriplets method. BWD computed four supertrees for different distance models within the measured time.

BWD supertrees were constructed using the implementation by Stephen J. Willson.[4] For the *PhySIC_IST* supertrees [34] we used the implementation provided by the authors.[5] *PhySIC_IST* offers a parameter to tune the method from "veto" to "voting-like". In our simulation, we used 0.5 and 1 as parameter settings. We computed SuperTriplets supertrees [30] using the implementation provided by the authors.[6]

All computations were performed on a Linux cluster of AMD Opteron-2378, 2.4 GHz CPUs, with 16 GB of memory.

## 8 Results

We first consider running times of MRP, MRF, BWD, *PhySIC_IST*, SuperTriplets, and the FlipCut algorithm presented here, see Table 1. For each instance, we use a running time limit of one hour in case of 100 and 500 taxon model trees, and two hours in case of 1000 taxon model trees. Entries '–' indicate that *no instance* was finished within the time limit: Regarding MRP, PAUP* returns a consensus of the current trees in memory if the time limit is exceeded. In contrast, the MRF implementation returns no tree. *PhySIC_IST* crashes for model tree sizes of 500 and 1000 taxa, and the SuperTriplets implementation crashes for model tree sizes of 1000 taxa.

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

[4] `http://www.public.iastate.edu/~swillson/software.html`
[5] `http://www.atgc-montpellier.fr/physic_ist/` with option '-c'
[6] `http://www.supertriplets.univ-montp2.fr/download.php`

Next, we investigate the accuracy of reconstructed supertrees. We use the Robinson-Foulds (RF) distance and the MAST distance. Results are shown in Fig. 4 and 5. Recall that *PhySIC_IST* usually computes non-plenary supertrees: Here, we first restrict the model tree to the taxon set of the supertree. This favors *PhySIC_IST* for all distance measures but the MAST distance, so *PhySIC_IST* results must be interpreted with some caution.

For the RF distance, we see that for all model tree sizes, MRP supertrees are of the best quality. For 100 taxa model trees, MRP, the *PhySIC_IST* variants, and MRF show the best performance, followed by FlipCut "Edge & Level". For 500 taxa model trees, performance from best to worst is: MRP, MRF, FlipCut "Edge & Level", BWD, and FlipCut unit costs. SuperTriplets performs good for small and large scaffold density. For 100 taxa model trees, performance from best to worst is: MRP, FlipCut "Edge & Level", and FlipCut unit costs.

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

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

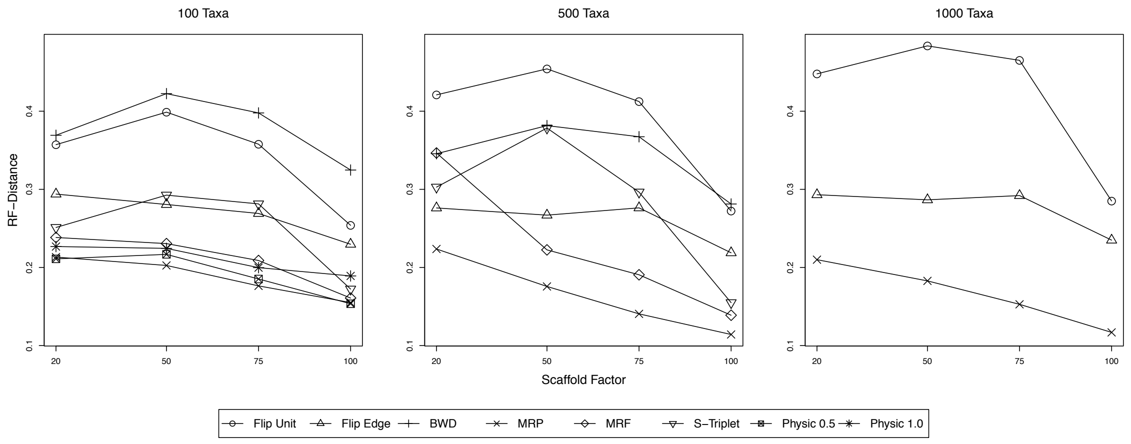


**Fig. 4.** Simulation results, quality of reconstructed supertrees. We plot the Robinson-Foulds distance between the calculated supertree to the true model tree, averaged over all simulation replicates. From left to right, model trees with 100, 500, and 1000 taxa.

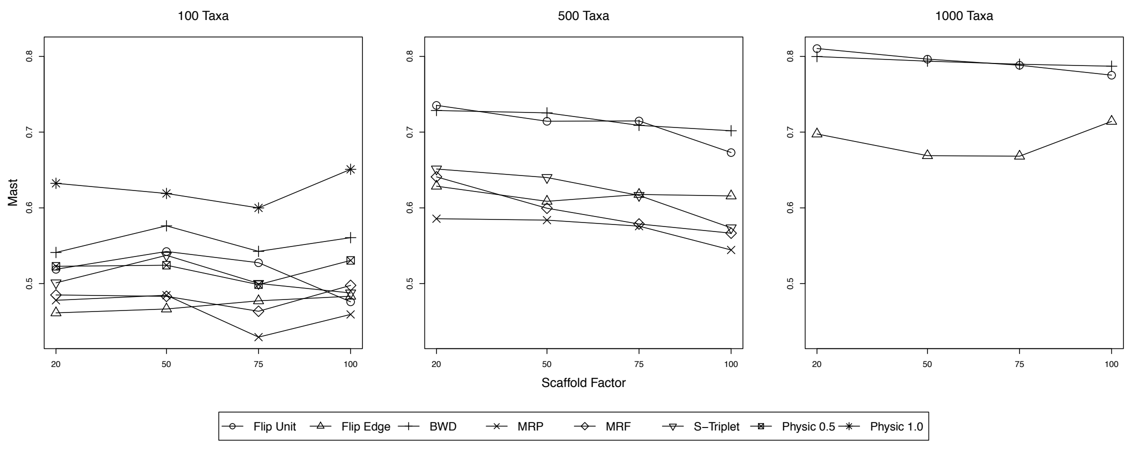


**Fig. 5.** Simulation results, quality of reconstructed supertrees. We plot the MAST distance between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, model trees with 100, 500, and 1000 taxa.

We also investigated the resolution as well as the triplet type I and II errors of the constructed supertrees. Results can be found in the appendix.

## 9 Conclusion

We have presented a novel supertree method named FlipCut supertrees. Our method combines the intuition behind both Minimum Flip and MinCut supertrees. In every step of the algorithm, we seek a cut in the FlipCut graph of minimum cost, and greedily proceed with the resulting subgraphs. We have presented a preprocessing method, ensuring that undisputed siblings will be present in the constructed supertree. The FlipCut supertree method has polynomial running time and is extremely swift in practice. Regarding supertree quality, performance of the FlipCut algorithm is sometimes even on par with MRP.

Besides the much better performance in simulations, FlipCut supertrees offer a fundamental advantage over MinCut supertrees: We have defined a global objective function (the number of flips in the input matrix) that we want to minimize. Using this objective function has been shown to result in supertrees of good quality [9]. Besides the theoretical amenity of such an objective function, this also has practical implications: We can compare the quality of different supertrees based on our objective function; and we can also compare the quality of *partial solutions*. This allows us to compare solutions that are build by a randomized version of our algorithm; and, we can do a beam search where we keep several sub-optimal solutions “alive” when building the trees. We conjecture that these modifications will further improve the quality of FlipCut supertrees, as it will address the problem of greedily separating a single taxon early. Finally, we want to evaluate methods for weighting the edges in the FlipCut graph. This includes using bootstrap values as character weights, as well as inferring ancestry using branch lengths as introduced by Willson [40]. The “Edge & Level” weighting presented here, is merely meant to demonstrate the impact of weighting edges on the quality of constructed supertrees. As a nice side effect, we stress that the weighted version of the FlipCut algorithm is consistently faster than its unweighted counterpart.

We note that the Undisputed Sibling preprocessing is not limited to the FlipCut algorithm; in the future, we want to evaluate whether using this preprocessing in conjunction with other supertree methods can improve their performance, both with respect to running times and accuracy. In addition, it is quite obvious that the preprocessing can be executed faster than $O(ln^2)$ time for $l$ trees and $n$ taxa.

*Acknowledgment.* Additional implementation by Markus Fleischhauer.

## References

1. A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J Comput*, 10(3):405–421, 1981.
2. B. R. Baum. Combining trees as a way of combining data sets for phylogenetic inference, and the desirability of combining gene trees. *Taxon*, 41(1):3–10, 1992.
3. O. R. P. Bininda-Emonds, editor. *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 4 of *Computational Biology Series.* Kluwer Academic, 2004.
4. O. R. P. Bininda-Emonds. Supertree construction in the genomic age. *Methods Enzymol*, 395:745–757, 2005.
5. S. Böcker, B. Bui, F. Nicolas, and A. Truss. Intractability of the minimum flip supertree problem and its variants. Technical report, Cornell University Library, 2011. arXiv:1112.4536v1.
6. M. Brinkmeier. A simple and fast min-cut algorithm. *Theory Comput Syst*, 41(2):369–380, 2007.
7. M. Brinkmeyer, T. Griebel, and S. Böcker. Polynomial supertree methods revisited. *Adv Bioinformatics*, 2011(Article ID 524182):21 pages, 2011.
8. D. Bryant and M. A. Steel. Extension operations on sets of leaf-labelled trees. *Adv Appl Math*, 16(4):425–453, 1995.
9. D. Chen, O. Eulenstein, D. Fernández-Baca, and J. G. Burleigh. Improved heuristics for minimum-flip supertree construction. *Evol Bioinform Online*, 2:347–356, 2006.
10. D. Chen, O. Eulenstein, D. Fernández-Baca, and M. Sanderson. Minimum-flip supertrees: Complexity and algorithms. *IEEE/ACM Trans Comput Biology Bioinform*, 3(2):165–173, 2006.
11. M. Chimani, S. Rahmann, and S. Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proc. of ACM Conf. on Bioinformatics and Computational Biology (ACM-BCB 2010)*, pages 147–153. ACM press, New York, 2010.
12. W. Day, D. Johnson, and D. Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Math Biosci*, 81(1):33–42, 1986.
13. L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, New Jersey, 1962.
14. L. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Adv Appl Math*, 3(1):43–49, 1982.
15. L. Gasieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of computing evolutionary trees. In *Proc. of Conference Computing and Combinatorics (COCOON 1997)*, volume 1276 of *Lect Notes Comput Sci*, pages 134–145. Springer, Berlin, 1997.
16. L. Gasieniec, J. Jansson, A. Lingas, and A. Östlin. On the complexity of constructing evolutionary trees. *J Comb Opt*, 3:183–197, 1999.
17. T. Griebel, M. Brinkmeyer, and S. Böcker. EPoS: a modular software framework for phylogenetic analysis. *Bioinformatics*, 24(20):2399–2400, 2008.
18. D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.
19. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
20. J. X. Hao and J. B. Orlin. A faster algorithm for finding the minimum cut in a directed graph. *J Algorithms*, 17(3):424 – 446, 1994.
21. M. R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees with applications to computational evolutionary biology. *Algorithmica*, 24:1– 13, 1999.
22. D. H. Huson, S. M. Nettles, and T. J. Warnow. Disk-Covering, a fast-converging method for phylogenetic tree reconstruction. *J Comput Biol*, 6(3-4):369–386, 1999.
23. D. H. Huson, L. Vawter, and T. J. Warnow. Solving large scale phylogenetic problems using DCM2. In *Proc. of Intelligent Systems for Molecular Biology (ISMB 1999)*, pages 118–129, 1999.
24. D. R. Karger. Minimum cuts in near-linear time. *J ACM*, 47(1):46–76, 2000.

25. R. D. M. Page. Modified mincut supertrees. In *Proc. of Workshop on Algorithms in Bioinformatics (WABI 2002)*, volume 2452 of *Lect Notes Comput Sci*, pages 537–552. Springer, Berlin, 2002.
26. I. Pe'er, T. Pupko, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM J Comput*, 33(3):590–607, 2004.
27. J.-C. Picard and M. Queyranne. On the structure of all minimum cuts in a network and applications. *Math Prog Study*, 13:8–16, 1980.
28. M. A. Ragan. Phylogenetic inference based on matrix representation of trees. *Mol Phylogenet Evol*, 1(1):53–58, 1992.
29. V. Ranwez, V. Berry, A. Criscuolo, P.-H. Fabre, S. Guillemot, C. Scornavacca, and E. J. P. Douzery. PhySIC: a veto supertree method with desirable properties. *Syst Biol*, 56(5):798–817, 2007.
30. V. Ranwez, A. Criscuolo, and E. J. P. Douzery. SuperTriplets: a triplet-based supertree approach to phylogenomics. *Bioinformatics*, 26(12):i115–i123, 2010.
31. F. Ronquist. Matrix representation of trees, redundancy, and weighting. *Syst Biol*, 45(2):247–253, 1996.
32. U. Roshan, B. Moret, T. Warnow, and T. Williams. Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proc. of IEEE Computational Systems Bioinformatics Conference (CSB 2004)*, pages 98–109, 2004.
33. H. Ross and A. Rodrigo. An assessment of matrix representation with compatibility in supertree construction. In O. R. Bininda-Emonds, editor, *Phylogenetic supertrees: Combining information to reveal the Tree of Life*, volume 4 of *Computational Biology Book Series*, pages 35–63. Kluwer Academic, 2004.
34. C. Scornavacca, V. Berry, V. Lefort, E. J. P. Douzery, and V. Ranwez. PhySIC_IST: cleaning source trees to infer more informative supertrees. *BMC Bioinformatics*, 9:413, 2008.
35. C. Semple and M. Steel. A supertree method for rooted trees. *Discrete Appl Math*, 105(1-3):147–158, 2000.
36. A. Stamatakis. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.
37. M. A. Steel, A. W. Dress, and S. Böcker. Simple but fundamental limitations on supertree and consensus tree methods. *Syst Biol*, 49(2):363–368, 2000.
38. M. S. Swenson, F. Barbancon, T. Warnow, and C. R. Linder. A simulation study comparing supertree and combined analysis methods using SMIDGen. *Algorithms Mol Biol*, 5(1):8, 2010.
39. D. L. Swofford. *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods) 4.0 Beta*. Sinauer Associates, 2002.
40. S. J. Willson. Constructing rooted supertrees using distances. *Bull Math Biol*, 66(6):1755–1783, 2004.
41. E. O. Wilson. A consistency test for phylogenies based on contemporaneous species. *Syst Zool*, 14(3):214–220, 1965.

## Appendix

## A Generating simulated datasets using SMIDGen

Basically, a set of source trees for the supertree methods consists of one scaffold tree and several clade-based trees. We describe only in brief how these sets are produced, see [38] for details. First, model trees are generated under a pure birth process. Branch lengths are perturbed from the ideal, ultrametric situation (no molecular clock). For each model tree, DNA sequence data sets are simulated under the GTR+$\Gamma$+I model, which differ in the taxa and genes used and whether they are scaffold or clade-based.

For the scaffold data sets, genes appearing at the root of a model tree are evolved along the tree without going extinct. Five of these so called universal genes are evolved for each model tree. A subset of taxa from the model tree is selected uniformly at random with a fixed probability $p$, called the "scaffold factor". This results in scaffold data sets having $p \times n$ taxa on average, where $n$ is the number of taxa in the model tree. The scaffold data sets are generated using scaffold factors of either 0.2, 0.5, 0.75 or 1. For each of the resulting scaffold data set, maximum likelihood trees are inferred using RAxML [36] in its GTRMIX default setting.

Genes for inferring clade-based source trees do not occupy the entire tree. For each of these 100 genes, called non-universal genes, a single birth node as well as lineages for which the given gene is lost, are determined using the process described in [38]. To choose clades for each clade-based data set, the same process is used, whereby the selection is restricted by setting bounds on the number of extant taxa in a clade to avoid selection of either very small or very large clades. For each 100-taxon model tree, five clades are selected with a clade size of at least 20. For each 500-taxon model tree, 15 clades with a clade size of at least 30, and for each 1000-taxon model tree 25 clades ranging in size between 30 and 500 are selected. For each of the clade chosen, the three non-universal genes are selected that cover the largest number of taxa in the clade. After these non-universal genes are chosen, the taxa in the clade are restricted to those that have all three of the genes. As this process could produce data sets with small numbers of taxa, any clade-based data set with fewer than ten taxa is excluded.

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

## B Results for Triplet distance (Type I and Type II error)

Beside the RF distance and the MAST score, we also considered *triplet-based type I* and *type II* errors. We iterate over all subsets $Y \subseteq \mathcal{L}(T) = \mathcal{L}(T')$ of cardinality $|Y| = 3$, and look at the induced subtrees $T|_Y$ and $T'|_Y$. According to [25], triplets of $T, T'$ can be partitioned into five sets: $S(T,T')$ and $D(T,T')$, the triplets resolved in $T$ and $T'$ that have the same and different topologies, respectively; $R_1(T,T')$, the triplets resolved in $T$ but not resolved in $T'$; $R_2(T,T')$, the triplets not resolved in $T$ but resolved in $T'$; $X(T,T')$ the triplets unresolved in $T$ and $T'$. Given these five triplet sets and the according triplet rates $s, d, r_1, r_2, x$, the type I error is defined as $et_I = (d + r_2)/(d + s + r_2)$. This corresponds to the proportion of triplets that are in $T'$ but not in $T$ and is sometimes called false positive. Accordingly, the type II error (sometimes called false negative) corresponds to the proportion of triplets that are not in $T'$ but in $T$, and is defined as $et_{II} = (d + r_1)/(d + s + r_1)$. We stress out that the type II error rate is closely related to the triplet-fit similarity from [25], which is defined as $1 - et_{II}$. Results concerning the type I and type II error can be found in Fig. 6 and 7.

## C Results for Resolution

Resolution was measured as the number of clades in the inferred supertree relative to the total number of clades on a fully binary tree of the same size ($n - 2$ for an unrooted tree, where $n$ is the number of taxa). Resolution varies between 0 and 1, where 0 indicates a unresolved bush and 1 indicates a binary supertree. Results concerning the resolution can be found in Fig. 8

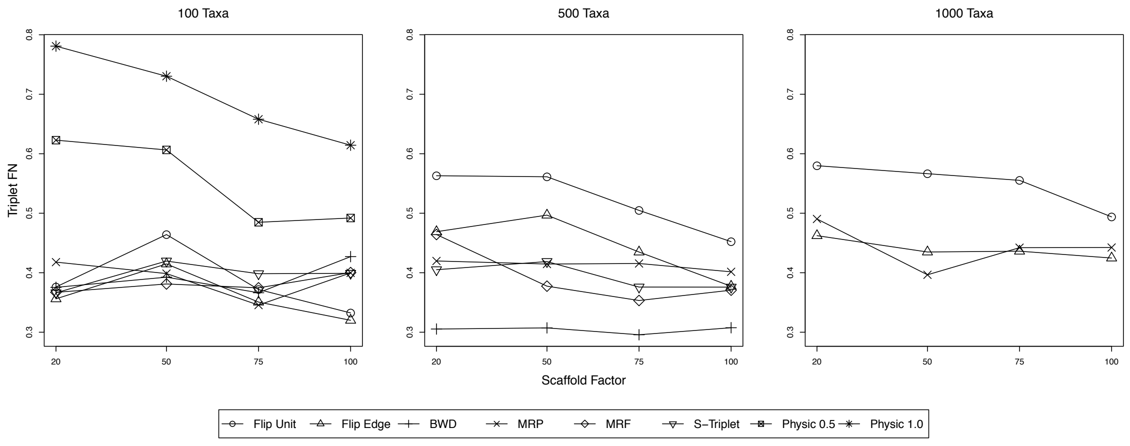


**Fig. 6.** Simulation results, quality of reconstructed supertrees. We plot the triplet-based type II error between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, the figure shows type II errors between the supertrees and 100, 500 and 1000 taxon model trees.

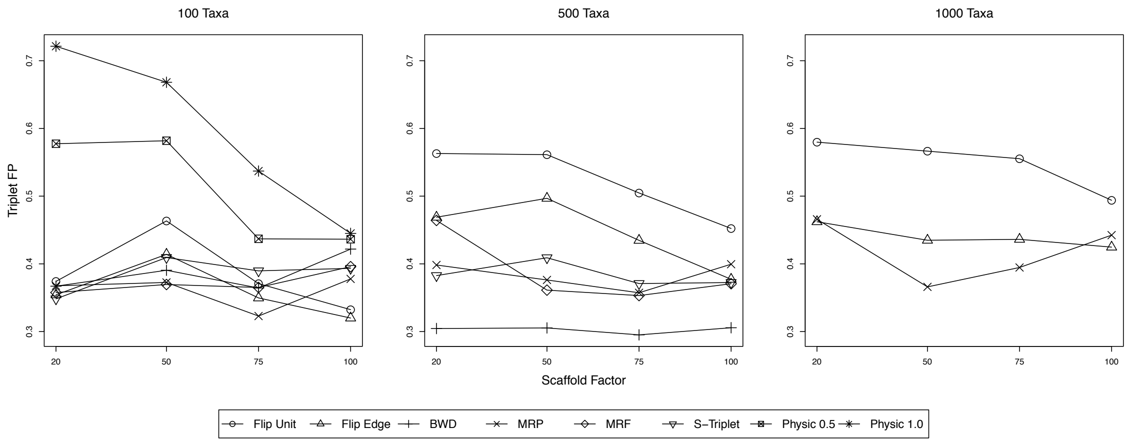


**Fig. 7.** Simulation results, quality of reconstructed supertrees. We plot the triplet-based type I error between the calculated supertree and the true model tree, averaged over all simulation replicates. From left to right, the figure shows type I errors between the supertrees and 100, 500 and 1000 taxon model trees.

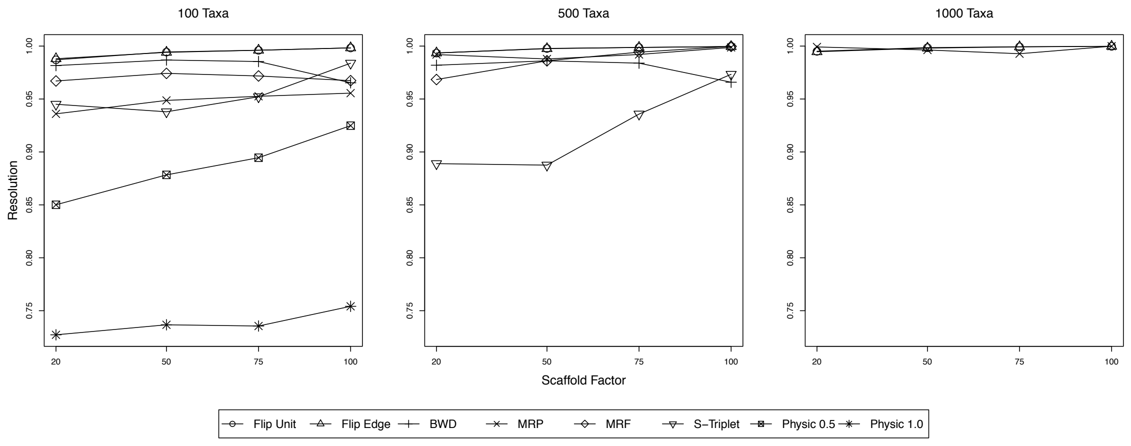


**Fig. 8.** Simulation results, resolution of reconstructed supertrees. We plot the resolution of supertrees averaged over all simulation replicates. From left to right, the figure shows resolution of supertrees belonging to 100, 500 and 1000 taxon model trees.