# Using Max Cut to Enhance Rooted Trees Consistency

Sagi Snir and Satish Rao

**Abstract**—Supertree methods are used to construct a large tree over a large set of taxa from a set of small trees over overlapping subsets of the complete taxa set. Since accurate reconstruction methods are currently limited to a maximum of a few dozen taxa, the use of a supertree method in order to construct the tree of life is inevitable. Supertree methods are broadly divided according to the input trees: When the input trees are unrooted, the basic reconstruction unit is a quartet tree. In this case, the basic decision problem of whether there exists a tree that agrees with all quartets is NP-complete. On the other hand, when the input trees are rooted, the basic reconstruction unit is a rooted triplet and the above decision problem has a polynomial time algorithm. However, when there is no tree which agrees with all triplets, it would be desirable to find the tree that agrees with the maximum number of triplets. However, this optimization problem was shown to be NP-hard. Current heuristic approaches perform min cut on a graph representing the triplets inconsistency and return a tree that is guaranteed to satisfy some required properties. In this work, we present a different heuristic approach that guarantees the properties provided by the current methods and give experimental evidence that it significantly outperforms currently used methods. This method is based on a divide and conquer approach, where the min cut in the divide step is replaced by a max cut in a variant of the same graph. The latter is achieved by a lightweight semidefinite programming-like heuristic that leads to very fast running times.

**Index Terms**—Phylogenetic trees, supertrees, rooted triplets, semidefinite programming.

## 1 Introduction

The study of evolution and the construction of phylogenetic (evolutionary) trees are classical subjects in biology. DNA sequences from a variety of organisms are rapidly accumulating, providing large amounts of data to a number of sequence-based approaches for phylogenetic trees reconstruction. The goal behind the "tree of life" project is to construct the tree representing the evolutionary history of over a million and a half different species. This task cannot be achieved by today's substitution-based phylogenetic reconstruction methods. Therefore, the need to design methods capable of amalgamating data taken from different sources is emerging.

The *supertree reconstruction problem* (or, for short, the supertree problem) (to be defined rigorously later) is as follows: Given a set of phylogenetic trees over overlapping nonidentical sets of taxa, find a tree over the union of the given taxa sets that *represents the best* the input subtrees. This output tree is denoted as the *supertree*.

We distinguish between *rooted* and *unrooted* trees (see Fig. 1). In unrooted trees, the decision problem of whether there exists a supertree that satisfies all the input subtrees was shown to be NP-hard by [25]. However, in the same paper, it is shown that, if the trees are rooted, then this problem is solvable in polynomial time by an algorithm of Aho et al. [2], devised originally in the setting of relational databases. Rooted trees have other attractive properties. Steel et al. [23] list three basic properties that are required from a supertree method:

- The method can be applied to any unordered set of input trees.
- If we rename all the species and then apply the method to the new input trees, we get the same old output tree under the renaming performed.
- If the input trees are consistent, the supertree returned should resolve all the input trees.

In the same work, it is shown that, if the input trees are unrooted, there is no supertree method that can satisfy these three requirements simultaneously. However, when the setting is changed to a rooted setting, these requirements are achievable. Moreover, there exists a polynomial time algorithm for that task. They suggested that the rooted setting was perhaps superior to the unrooted one in the context of supertree construction.

In this work, we focus on triplets (see Fig. 2). Every rooted tree can be represented by a set of rooted triplets, which are the smallest rooted trees. Given a rooted tree, we can construct all the triplets that define this tree (note that this set of triplets is not unique) and apply Aho et al.'s algorithm on that set. The algorithm (to be described in the sequel) uses a simple recursive divide and conquer approach to split between the taxa at every stage in the recursion while building the tree.

The other direction, however, of constructing a tree from input triplets is more complicated. As biological data suffers from noise, it is very likely that the set of input triplets will be inconsistent, i.e., there will be no tree $T$ that represents all the triplets. In that case, Aho et al.'s algorithm above will halt and report the inconsistency. Therefore, it is

- *S. Snir is with the Department of Mathematics, University of California, Berkeley, CA 94720. E-mail: ssagi@math.berkeley.edu.*
- *S. Rao is with the Department of Computer Science, University of California, Berkeley, CA 94720. E-mail: satishr@cs.berkeley.edu.*

*Manuscript received 21 Dec. 2005; revised 24 May 2006; accepted 15 June 2006; published online 31 Oct. 2006.*
*For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0149-1205.*

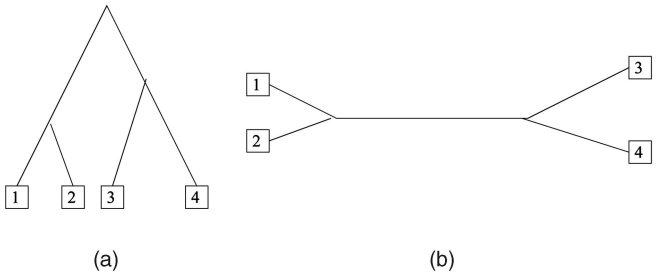


Fig. 1. (a) A rooted and (b) an unrooted version of the same tree.

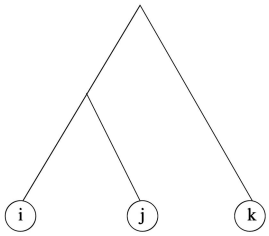


Fig. 2. The rooted triplet $i, j|k$. The least common ancestor of $i$ and $j$ is a descendant of the least common ancestor of $i$ (or $j$) and $k$.

desirable to find a tree that maximizes (minimizes) the number of satisfied (violated) triplets. Unfortunately, this turns to be an NP-hard problem, as was shown by [5].

The min-cut algorithm of [24] was the first to cope with this problem of inconsistent input triplets. Their algorithm handles rooted subtrees of arbitrary size, however, it also specializes in rooted triplets. Page [19] extended this idea to maintain all subtrees that are not in disagreement with any input subtree. The algorithm, denoted *modified min cut* (MMC), applies the min cut criterion but on a somewhat different graph than that of [24].

Briefly, both MC and MMC proceed recursively by finding a minimum cut in a graph built from the current set of triplets. The triplets in this minimum cut will then be discarded so that the remaining triplets can be partitioned into subtrees that are combined. The discarded triplets correspond to the unsatisfied ones. The algorithms are a bit narrow in their view in that they proceed cautiously in order to violate a minimum number of triplets at each step. At each step, there are also triplets that are satisfied. They do not include this phenomena in their divide step.

In this paper, we extend the above approach and apply a more inclusive and less greedy criterion in the divide step. In short, instead of minimizing the number of triplets that are violated at every step, we aim at maximizing the ratio between satisfied to violated triplets. Unfortunately, maximizing this ratio is, itself, an NP-complete problem. Thus, we develop a very fast heuristic that is motivated by semidefinite programming and that works quite well for this application. This approach leads to much better practical results in all the criteria without interfering with any of the properties guaranteed by these algorithms. Indeed, our methods are significantly better and even faster than implementations of MMC.

We also compare our method to a more accurate but much slower method, matrix representation with parsimony (MRP) [3], [21]. Our tests show that our method also outperforms this more accurate method. Unfortunately, the computational requirements of the MRP approach do not allow us to obtain performance numbers for MRP for even moderate size. As an example, MRP ran for more than 11 hours in an experiment of 2,000 triplets over 300 taxa while our approach took a bit more than six minutes (383 seconds) for 50,000 triplets on 2,000 taxa. Moreover, the implementation effort was quite modest.

We also report on results on two real data sets that come in the form of subtrees over larger sets of taxa than triplets. The results obtained highlight the difference between the data sets.

We comment that the use of semidefinite programming in the context of phylogenetic reconstruction is not new. Ben-Dor et al. [4] used semidefinite programming to construct a tree over a given set of quartets. As was mentioned above, the unrooted setting (and, hence, the use of quartets) is more complicated and does not render an immediate divide and conquer approach. Hence, their application embedded all the taxa set on the $n$-dimensional sphere in a single step by a heavy SDP machinery and then employed a cherry-picking strategy to construct the tree. The heavy use of the SDP mechanism enforced severe limitations on the size of the problems handled and allowed them to build trees over 15 taxa. In contrast, our heuristic efficiently handles problem sizes of several thousands.

In this contribution, we extend a previous work of ours [17] by providing more proofs to the arguments throughout the paper, elaborating more on our SDP-like approach and the properties of the other methods. The experimental part was also extended substantially.

The remainder of this paper is organized as follows: In Section 2, we present the notations used and define the maximum triplet consistency method. In Section 3, we survey some existing methods for rooted supertrees and discuss some of their properties. Section 4 describes the pitfalls of current triplet-based supertree methods, outlines our contribution, the algorithmic challenges involved, and their resolutions. Section 5 describes experimental results comparing our method with two other representative methods on two types of synthetic data and Section 6 reports on results on two instances of real data. We conclude in Section 7 with discussion and future research directions.

## 2 Preliminaries

### 2.1 Phylogenetic Trees

For a tree $T = (V, E)$, we denote by $\mathcal{L}(T)$ the set of leaves of $T$. A phylogenetic tree $T$ over a set of taxa $\mathcal{X}$ is a tree for which there is a bijection between $\mathcal{X}$ and $\mathcal{L}(T)$. Henceforth, we will identify the taxa set with the leaves they are mapped to. A tree $T$ is said to be *rooted* if the set of edges $E$ is directed and there is a single distinguished internal vertex $r$ with in-degree 0. Let $u$ and $v$ be two vertices in a rooted tree $T$. We say that $u$ is a *descendant* of $v$ and $v$ is an *ancestor* of $u$ if there is a (directed) path from $v$ to $u$. For $u$, $v \in V$, the *least common ancestor* of $u$ and $v$, or $lca(u, v)$, is a vertex $w$ that is an ancestor of both $u$ and $v$ and there is no

descendant of $w$, $w'$ that is also an ancestor of both $u$ and $v$. From now on, we will restrict our attention to phylogenetic trees solely. Let $T$ be a tree and $A \subseteq \mathcal{L}(T)$. We denote by $T|_A$ the tree induced by the subset of leaves $A$ where all internal vertices with degree two are contracted. When $T$ is rooted, the contraction is done at vertices with out-degree one. For two trees $T$ and $T'$, we say that $T$ *satisfies* $T'$, and $T'$ is *satisfied* by $T$, if $\mathcal{L}(T') \subseteq \mathcal{L}(T)$ and $T|_{\mathcal{L}(T')} = T'$. Otherwise, $T'$ is *violated* by $T$. In [6], it is shown that this requirement is equivalent to the condition that, for every $u, v, w \in \mathcal{L}(T')$, $LCA(u, v)$ is a descendant of $LCA(u, w)$ in $T$ if and only if $LCA(u, v)$ is a descendant of $LCA(u, w)$ in $T'$. This observation forms the basis of the triplet approach in supertree construction.

### 2.2 The Supertree Problem

For a set of trees $\mathcal{T} = \{T_1, \ldots, T_k\}$ with possibly overlapping leaves, we say that $\mathcal{T}$ is *consistent* if there exists a tree $T^*$ over the set of leaves $\bigcup_i \mathcal{L}(T_i)$ that satisfies every tree $T_i \in \mathcal{T}$. Otherwise, $\mathcal{T}$ is *inconsistent*. When $\mathcal{T}$ is inconsistent, it is desirable to find a tree $T^*$ over $\bigcup_i \mathcal{L}(T_i)$ that minimizes some objective function. $T^*$ is denoted a *supertree* and the problem of finding $T^*$ is the *supertree problem*.

A *rooted triplet*, or, for short, a triplet, is a rooted tree over three leaves $u$, $v$, $w$. We write a rooted triplet over $u$, $v$, $w$ as $u, v|w$ if $LCA(u, v)$ is a descendant of $LCA(u, w)$. The *triplet score* between a tree $T_i$ and $T^*$ is the sum of $u, v, w \in \mathcal{L}(T_i)$ such that $T_i|_{\{u,v,w\}} = T^*|_{\{u,v,w\}}$. The triplet score between $\mathcal{T}$ and $T^*$ is the sum of the triplet scores between $T^*$ and every $T_i \in \mathcal{T}$. When $\mathcal{T}$ is a set of rooted triplets, this score is just the number of triplets satisfied by $T^*$. We refer to this problem as the *maximum triplet consistency* (MTC) problem.

### 2.3 Tree Metrics

An edge in a phylogenetic tree induces a partition on the set of leaves. We identify an edge by the partition it induces. The Robinson-Foulds (RF) symmetric distance between two trees is the number of edges (partitions) existing in exactly one tree. Another score between a set of trees is the *maximum agreement subtree* (MAST). This is defined as the largest set of leaves $A$ common to all $T_i \in T$ such that $T_i|_A = T_j|_A$ for every $T_i$, $T_j \in \mathcal{T}$.

## 3 Supertree Methods

### 3.1 Triplets Methods

We now describe minimum cut algorithms for triplet-based reconstruction. The first algorithm solves the problem when all the triples are consistent and was developed in the context of relational databases by Aho et al. [2]. Steel presented the algorithm for use in phylogenetics in [25]. The algorithm uses two generic partitioning rules of which only one is used in our case. It proceeds recursively on the set of taxa by applying the partitioning rule to produce a tree. The algorithm is described below:

Aho et al. $(V, \mathcal{T})$

1. Let $V$ be the set of taxa.
2. If $\mathcal{T} = \emptyset$ return a tree of depth 1 with all $V$ as sister taxa.
3. Build the connectivity graph $G_C = (V', E')$ as follows:
   — $V' \leftarrow V$,
   — for every triplet $i, j|k \in \mathcal{T}$, introduce the edge $(i, j) \in E'$.
4. Let $c$ be the number of connected components in $G_C$.
5. If $c = 1$, return NULL, no tree is consistent with all triplets.
6. Else
   — Create an internal vertex $u$ in $T$.
   — For every connected component $C_i$ in $G$,
   • $T_i \leftarrow$ Aho et al. $(V(C_i), \{(i, j|k) \in \mathcal{T} : i, j, k \in V(C_i)\})$.
   • Make $T_i$ a child of $u$.
7. Return $T_u$.

It is clear that the algorithm finds a tree $T$ over $\mathcal{X}$ that satisfies all the input triplets if such a tree exists. The algorithm runs in time $O(|\mathcal{T}| \cdot n)$ and a later improvement by Henziger et al. [13] reduced it to $min\{O(|\mathcal{T}| \cdot n^{0.5}), O(|\mathcal{T}| + n^2 \log n)\}$.

In most of the cases, due to noise in the data, the input triplets are inconsistent and, therefore, the above algorithm will fail. This turns the problem into an optimization problem as follows:

*Problem 1.* Maximum Triplet Consistency (MTC)
**Input:** A set of rooted triplets
**Output:** A rooted tree $T$ maximizes the number of satisfied triplets.

**Proposition 1 ([5]).** *The MTC problem is NP-hard.*

In cases of inconsistent triplets, the Aho et al. algorithm will find it and halt. Still, the following lemma shows that running the algorithm to this point does not affect the optimality:

**Lemma 1.** *For any set of input triplets $\mathcal{T}$ (inconsistent or not), if the connectivity graph $G_C$ (constructed at Step 3 of Aho et al.'s algorithm) is not connected, then any optimal tree $T^*$ will have a subtree for (the taxa at) every component in $G_C$.*

**Proof.** Seeking contradiction, assume there is an optimal tree $T^*$ that does not have disjoint subtrees for every connected component in $G_c$.

We partition $\mathcal{T}$ into two disjoint sets:

- $\mathcal{T}_{intra}$: The set of triplets $\{(i, j|k)\}$ such that $i$, $j$, $k$ are in the same component in $G_c$.
- $\mathcal{T}_{inter}$: $\mathcal{T} \setminus \mathcal{T}_{intra}$, the set of triplets $\{(i, j|k)\}$ such that $i, j, k$ are in different components in $G_c$.

By definition, $T^*$ violates at least one triplet from $\mathcal{T}_{inter}$. Let $k$ be the number of components in $G_c$ and let $C_i$ be the $i$th component. Let $T'_i$ be $T^*|_{V(C_i)}$, the subtree of $T^*$ induced by the leafs of component $i$. We construct the tree $T'$ by hanging all subtrees $T'_i$, $1 \leq i \leq k$, from a common root $r$. Then, $T'$ satisfies the same number of triplets from $\mathcal{T}_{intra}$ as $T^*$, but all the triplets from $\mathcal{T}_{inter}$, a contradiction. □

Relying on the above lemma, the min-cut algorithm of Semple and Steel [24] is based on the following idea: Apply the Aho et al. algorithm as long as possible. If, at a certain

stage in the recursion of the algorithm, Step 3 yields a connected graph (i.e., a single connected component), perform the following modification to the algorithm:

- Convert the connectivity graph $G$ of Step 3 into an edge weighted graph $G' = (V, E, w)$, where, for $e = (i, j)$, $w(e) = |\{(i, j|k) \in \mathcal{T} : k \in X\}|$.
- Compute min-cut on the graph $G'$.
- Apply the Aho et al. algorithm on the two subproblems induced by the two resulting components.

Although [24] did not prove any bound for the quality of their solution, they did claim that subtrees that are shared by all input subtrees are resolved by the output supertree. An elegant extension of this idea was presented by Page [19] to maintain all subtrees that are not in disagreement with any input subtree. The algorithm, denoted *modified min cut* (MMC), applies the min cut criterion but on a somewhat different graph than that of [24].

### 3.2 Character-Based Supertree Methods

We now describe a family of supertree methods that is based on solving a more general problem. The family of character-based supertree methods contains these two methods: Matrix Representation using Parsimony (MRP) and Matrix Representation using Flipping (MRF). MRP [3], [21] is the most widely used supertree method by practitioners. It has been found to have good performance [9]. In this method, the input subtrees are encoded in a {0, 1} matrix in the following way: As every edge $e$ in an input subtree $T_i$ induces a partition on $\mathcal{L}(T_i)$, $(A, B)$, $e$ is encoded as binary character $C$, where, for each $s \in A$, $C(s) = 0$ and, for $s \in B$, $C(s) = 1$. For $s \in \mathcal{X} \setminus \mathcal{L}(T_i)$, $C(s) = ?$, indicating a *missing state*. The method tries to find the maximum parsimonious tree with respect to that matrix. It is not confined to a rooted setting and, hence, loses some amount of its power when the input trees are rooted. However, to code for rooted trees, the following idea is employed: Augment the taxa set with an artificial species $r$. Now, let $T_i$ be some input subtree with root $r_i$ and let $e$ be an edge in $T_i$ inducing the partition $(A, B)$ over $\mathcal{L}(T_i)$. Without loss of generality, assume that $T|_A$ contains $r_i$, that is, the root $r_i$ is in the subtree of $T_i$ induced by $A$. Then, we set $C(r) = 0$, for all $s \in A$, $C(s) = 0$ and, for $s \in B$, $C(s) = 1$.

When the input $\mathcal{T}$ is a set of triplets given as input to the MTC problem, the above is a reduction from MTC to the MRP problem in the following sense:

**Observation 1.** *Let $T$ be a tree over $\mathcal{X}$ and $M$ an $n \times m$ matrix as defined above. Then, for every character $C$ in $M$, the parsimony score of $C$ with respect to $T$ is 1 if the triplet coded for $C$ is satisfied by $T$. Otherwise, it is 2.*

**Proof.** Let $T_i = i, j|k$ be the triplet corresponding to character $C_i$ in the matrix. We have $C_i(i) = C_i(j) = 0$ and $C_i(k) = C_i(r) = 1$. Assume $T_i$ is satisfied by $T$, then there is an edge $e$ separating leaves $k$ and $r$ from $i$ and $j$. Then, we set the state at all vertices at the side of $i$ and $j$ to 0 and the rest to 1 (see Fig. 3a). So, there is exactly one change along the edge $e$ in the whole tree. Conversely, assume $T_i$ is violated by $T$. So, there is no such edge $e$ and it is easy to see that any {0, 1} labeling of internal nodes will yield at least two changes along some edges and there is always a labeling yielding exactly two changes (see Fig. 3b). □

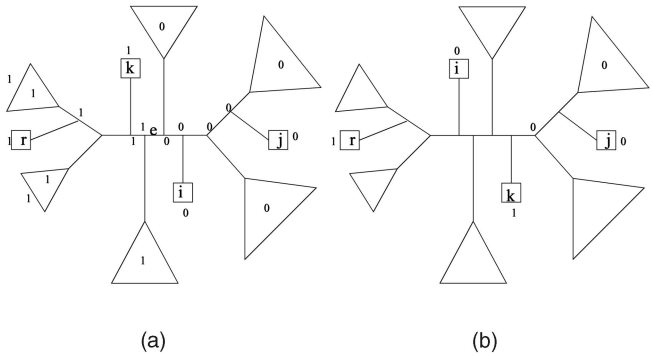


Fig. 3. (a) All nodes left of edge $e$ can be labeled with 1 and nodes to the right of $e$ with 0, yielding a single change along $e$. (b) Given $C_i(i) = C_i(j) = 0$ and $C_i(k) = C_i(r) = 1$, no labeling can yield a single change on the tree.

**Corollary 1.** *The parsimony score of $M$ (or, alternatively, the MRP score) is $m$ plus the number of triplets violated by $T$.*

Since the MTC is NP-complete, Corollary 1 implies that solving the MRP problem, even for rooted triplets, is NP-hard and, therefore, some heuristics need to be employed. In practice, this leads to the somewhat confounding results that, even when given a consistent set of triples, MRP methods do not obtain a consistent solution as do the triplet methods.

The other character-based supertree method is the Matrix Representation using Flipping (MRF) [8], [9]. MRF starts with the same matrix as MRP; however, its objective function is somewhat different: It seeks the minimum number of states flipping at the characters in order to make all characters compatible. In that case, the supertree is convex on all characters. This problem is somewhat of a restricted version of the "big convex recoloring" problem introduced in [18], as the characters are restricted to two states only. By similar lines to Observation 1, the following observation is derived:

**Observation 2.** *Given a set of triplets $\mathcal{T}$ and letting $M$ be the partial matrix representing $\mathcal{T}$, the MRF score for $M$ is exactly the MTC score for $\mathcal{T}$.*

It was found that MRP and MRF perform similarly in experiments [9]. Since MRF runs much slower, we compare our method just to MRP.

## 4 The Max Cut Triplets Algorithm

In this section, we describe our algorithm, MAX CUT triplets (MXC). We start with an example that demonstrates the locality of the min cut approach and motivates a more inclusive one. Next, we describe our algorithm and the SDP-like heuristic to cope with such cases.

### 4.1 Min Cut Flaw

Consider the set of triplets

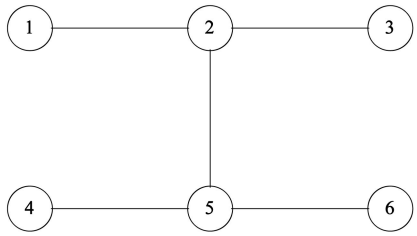


Fig. 4. A connectivity graph induced by the set of triplets $\{(1,2|4),(2,3|5),(2,5|3),(4,5|2),(5,6|3)\}$.

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

It can easily be shown that their connectivity graph contains a single component (depicted in Fig. 4).

This implies that there is no tree that satisfies all these triplets simultaneously and some criterion needs to be applied in order to partition the connectivity graph and continue in the recursion. Moreover, as every subset of leaves appears in at most one input tree, the weight of every edge in the connectivity graph is one. Therefore, a naive application of the min cut criterion is indifferent to where to partition the graph and a pathological example is the removal of the edge $(1,2)$ violating the triplet $(1,2|4)$, then the edge $(4,5)$ violating the triplet $(4,5|2)$, then $(5,6)$ violating $(5,6|3)$, and, eventually, $(3,2)$ violating $(2,3|5)$. The resulting tree is depicted in Fig. 6a. This tree satisfies only the $(2,5|3)$ triplet and violates four out of the five input triplets.

### 4.2 Our Approach

Our algorithm proceeds along the divide and conquer strategy of Aho et al.'s algorithm identically to MC and MMC. The algorithm relies on the connectivity graph computed by Aho et al. For a triplet $i,j|k$, the original edge $(i,j)$ is denoted as the *bad edge*. However, the algorithm also augments the graph with two *good edges*, $(i,k)$ and $(j,k)$. For the specific input above, the resulting graph is depicted in Fig. 5.

Now, we notice the following observation:

**Observation 3.** *For every triplet $i,j|k$ such that a good edge $(i,k)$ or $(j,k)$ is in the cut but $(i,j)$ is not, the triplet $i,j|k$ is satisfied by the algorithm. Moreover, only triplets $i,j|k$ such that no edge is in the cut will proceed to subsequent steps.*

Therefore, instead of minimizing the number of violated triplets, it is more plausible to also consider what triplets are satisfied at each step. This can lead to possibly more violated triplets at a specific step, but an increase in satisfied ones. Therefore, the algorithm should maximize the ratio between good and bad edges in every divide step of the algorithm. We remark here that all triplet-based algorithms do that maximization implicitly in cases when the triplets are consistent. At that time, the ratio is infinity.

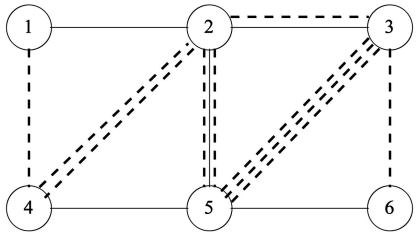


Fig. 5. The graph created by the Max Cut triplets algorithm for the set of triplets $\{(1,2|4),(2,3|5),(2,5|3),(4,5|2),(5,6|3)\}$. Good edges are drawn by dashed lines.

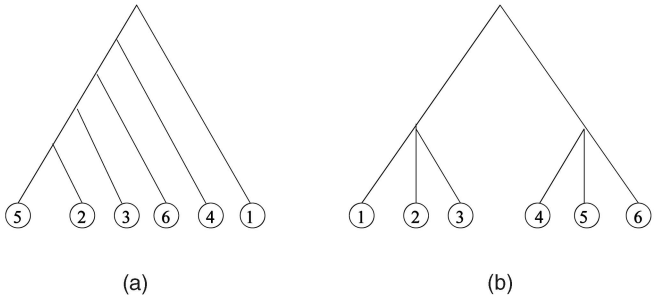


Fig. 6. The resulting trees of (a) min cut and (b) max cut.

It can easily be seen that the cut maximizing the ratio between good and bad edges in Fig. 5 is the cut $(\{1,2,3\},\{4,5,6\})$. This cut violates the triplet $2,5|3$ and, at the same time, satisfies the rest of the triplets. The resulting tree appears in Fig. 6b. A by-product of this approach is that the recursive step is initiated only three times here (for every internal node) in contrast to five in the min cut algorithm and the running time decreases as well.

The above example illustrates the motivation behind our algorithm. The complete algorithm is as follows:

MAX CUT triplets $(V,\mathcal{T})$

1. Let $V$ be the set of taxa.
2. If $\mathcal{T}=\emptyset$ return a tree of depth 1 with all $V$ sister taxa.
3. For every triplet $i,j|k\in\mathcal{T}$, introduce the edge $(i,j)\in E$.
4. Let $c$ be the number of connected components in $G=(V,E)$.
5. If $c=1$
   — Denote the edges created in Step 3 as *bad edges*.
   — For every triplet $i,j|k\in\mathcal{T}$, add two *good edges* $(i,k)$ and $(j,k)$ to $E$.
   — Find a cut $(C,\bar{C})$ in $G$ such that the ratio of good edges versus bad edges in the cut is maximized.
6. Create an internal vertex $u$.
7. For every connected component $C_i$ in $G$,
   — $T_i \leftarrow$ MAX CUT triplets $(V(C_i),\{(i,j|k)\in\mathcal{T}:\ i,j,k\in V(C_i)\})$.
   — Make $T_i$ a child of $u$.
8. Return $T_u$.

We observe that the algorithm terminates since, at any divide step, a nontrivial cut is produced, identically to the other algorithms.

### 4.3 Heuristic for Optimizing the Ratio

We notice that Step 5 tries to find a cut that maximizes the ratio between good and bad edges. Unfortunately, in general, finding such a cut is NP-hard as well.

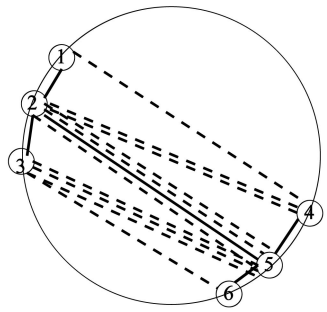


Fig. 7. The resulting embedding of the graph in Fig. 5. Bad edges are the wide solid lines and good edges are the wide dashed lines.

We now describe our heuristic for coping with this task. Since this heuristic is based on an SDP approach, we first give a short introduction to it (see [27] for a broader explanation). A symmetric matrix $X$ is positive semidefinite if it is a product of $V^TV$, where $V$ is an $n \times r$ real matrix.

The input to the SDP problem is a $n \times n$ symmetric matrix $C = [c_{i,j}]$, a set of $m$ $n \times n$ symmetric matrices $A^{(k)} = [a^{(k)}_{i,j}]$, and a real $m$-dimensional vector $b$.

The problem itself is defined as follows:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}x_{i,j} \\ \text{subject to} & \\ \forall k \in \{1,\ldots,m\} & \sum_{i=1}^{n}\sum_{j=1}^{n} a^{(k)}_{i,j}x_{i,j} \leq b^{(k)}, \end{array}$$

and the matrix $X = [x_{i,j}]$ is positive semidefinite.

SDP became very popular for applications in combinatorial optimization as it can be solved accurately and efficiently. In these applications, the matrix $V$ defined above represents the $n$ vertices, where each vertex is represented by an $r$-dimensional vector.

For problems related to our particular case, several semidefinite programming-based approximation algorithms have been suggested [12], [1]. Motivated by these algorithms we formulated the following problem:

Embed the vertices of the connectivity graph on a sphere in the Euclidean space so as to optimize the function:

$$\sum_{\text{good edges } e=(i,j)} d(i,j) - 3 \sum_{\text{bad edges } e=(i,j)} d(i,j), \qquad (1)$$

where $d(i,j)$ is the Euclidean distance between points $i$ and $j$.

Once we obtain the embedding, we produce a cut as follows: Partition the vertices into two sets by selecting a random hyperplane to partition the points and producing the corresponding cut.

The intuition behind that approach is that, for a good edge to be in the cut, we want its two endpoints to be as far apart as possible. For bad edges, the converse argument applies and, therefore, it is preceded with a minus sign. An ideal case is when we have a cut that cuts all good edges but no bad edges. A good embedding for such a cut will put all vertices at two antipodean points on the sphere, where, for vertices $i$, $j$ connected by a bad edge, $i$, $j$ will be embedded in the same point and, for $i$, $j$ connected by a good edge, $i, j$ will be embedded in different points.

A schematic illustration of the embedding resulted from the graph in Fig. 5 is shown in Fig. 7.

The embedding can be computed by semidefinite programming packages for embedding into $n$ dimensions. Although such a procedure runs in time polynomial in the number of taxa, we, however, found that replacing the heavy SDP mechanism by the following local heuristic provides good enough results for our purpose. We emphasize that this heuristic replaces only the embedding done by the SDP. The rest remains the same. We locate the points onto a three-dimensional sphere and then locally move a point toward its center of mass. Specifically, let $u$ and $v$ be nodes in the graph connected by an edge $(u,v) \in E$ with weight $w(u,v)$. Let $\bar{u}$ and $\bar{v}$ be the points in the sphere representing $u$ and $v$ and let $d(\bar{u},\bar{v}) = |\bar{u} - \bar{v}|$ be their Euclidean distance. Then, we define the *weighted distance* between $u$ and $v$,

$$wd(u,v) = w(u,v) \cdot d(\bar{u},\bar{v}).$$

For a node $v$, the *center of mass* of $v$,

$$cm(v) = \sum_{u:(u,v)\in E} -wd(u,v).$$

Our algorithm is as follows:

1. Locate the nodes randomly on the three-dimensional sphere.
2. Do a constant number of times:
   — For every $v \in V$, move $v$ to $cm(v)$.

The intuition behind this heuristic is identical to the one of the SDP: We want to locate nodes separated by bad edges close to each other and conversely for good edges. The constant we used in our program is 100 and we believe it can be reduced.

Finally, after generating the random cut, we used a version of the Fiduccia-Matheyes bisection improvement algorithm [15], [11] to improve the resulting cut. The *improvement potential* of a node is the change in the value of the cut incurs by swapping the node to the other side of the cut. In this step, the node with the best improvement potential, but such that it was not selected before, is swapped. The running time of this step is $O(n^2)$ and it recurs as long as the value of the cut increases. Although we have no bound on the number of such iterations, we found that a small number suffices (i.e., $< 20$ for $\#taxa = 1,000$).

## 5 Simulation Results

In order to test our method, we conducted experiments that compared our method versus two previously mentioned methods. Further results on the comparison between MC, MMC, MRP, and MRF are reported in [9]. As in [9], we report that MRP is substantially more accurate than MMC, but at the cost of a much longer running time. Since, by [9], MRF performs similarly to MRP but with longer running times, here we only test MRP and MMC versus our method, MXC. In contrast to [9], we found that the Robinson-Foulds measures does distinguish between the different methods and, therefore, included it.

We generated our triplets from some model tree. In particular, we generated triplets from a given model tree and output some percentage of incorrect triplets. The reasons to introduce contradicting triplets are two: When all the triplets agree, no cut is performed and the methods can't be tested. Second, we want to test how the method react in terms of reconstructing the model tree, to a small versus large perturbation in the input.

The properties we wanted to measure are the accuracy of the supertree returned in terms of the number of triplets satisfied and the resemblance of that tree to the model tree. Therefore, we measured the following:

- **Triplet fit:** This measures how many (in percentage) of the input triplets are satisfied by the tree returned by the method. This is actually the main criterion as this is what the method tries to optimize.
- **Maximum Agreement Subtree (MAST) fit:** The MAST score between the model tree and the reconstructed tree.
- **Robinson-Foulds (RF) fit**: The RF topological distance between the model tree and the reconstructed tree. Since MRP returns a fully resolved tree, whereas a triplet-based method (seeks to) return the minimally resolved tree that satisfies most of the triplets, we measured a more MRP-favorable measure, which is the number of common edges between the two trees. This is simply derived from the latter by $\frac{E_1+E_2-RF}{2}$, where $E_i$ is the number of edges in tree $i$ and $RF$ is the Robinson-Foulds distance. The two numbers separated by a slash (in the RF columns) in the tables represent the percentage of common edges out of the number of edges in the model tree and the percentage of common edges out of the number of edges in the inferred tree.[1]
- The running time in seconds.

The experiments we conducted are divided into two types of distributions, differing by the way in which the species of any triplet was chosen.

1. **Uniform distribution:** The species are chosen according to a uniform distribution from the species set.
2. **Geometric distribution**: Triplets over species with diameter (the maximum number of tree edges between any pair of taxa) $d$ were chosen with probability $\frac{1}{d}$. This introduces locality into the process of triplet generation where the input contains more triplets of related species than unrelated.

As a first and independent test, we wanted to check whether a set of randomly chosen triplets, even without providing incorrect triplets, contains sufficient information to reconstruct the model tree. This was tested by giving 100 percent accurate triplets to the Aho et al. algorithm and resulted in very high MAST and RF scores. For example, for a set of 1,000 triplets drawn from a 100 taxa tree, we obtained a MAST score of 84 and an RF scores 80.4/95. Of course, we can't expect to achieve better scores in the MAST and RF scores in the presence of incorrect triplets.

1. Note that while, for MRP, these two numbers are always the same, for MXC and MMC, the first is not greater than the second.

### 5.1 Uniform Distribution Results

A sample of our results in terms of satisfied triplets, MAST score, RF score, and running times is reported in Table 1. We note that the gap in performances and running time between MMC and MRP in these experiments is as reported in [9], while MRP and our procedure perform in a similar fashion (with respect to the triplet fit measure). Indeed, the latter methods typically output a tree that is *better* than the model tree. We also observe that our method is usually a bit better than MRP in the triplet fit score and, as the problem size grows, our advantage increases. For example, for the largest problem where we could get scores for both, we obtain a score of 75.7, while MRP achieves 67.3. Moreover, the running time of MRP (and even MMC) became prohibitive far before the limits of our methods. It appears that MRP outperforms our method in terms of MAST fit. However, the slight discrepancy between the two measures can be evidenced in the cases when the triplets were consistent (100 percent triplet fit for our method and MMC), yet a higher MAST fit was obtained by MRP, although the tree it returned did not satisfy all the triplets. As to the RF fit, it appears that RF is a very stringent measure, sensitive to even a small percentage of contradicting triplets or to insufficient information (too few triplets with respect to the number of taxa). This explains the small numbers (mostly zeros) when the percentage of correct triplets is 50 or 70 or when this percentage is high but not enough triplets (see, e.g., the line of 150 taxa, 1,000 triplets, and 90 percent correct). On the other hand, when the input data permitted (by both enough triplets and high enough percentage of correct triplets), our method outperforms MRP in the two RF measures (see, e.g., the line of 50 taxa, 1,000 triplets, and 90 percent correct).

MXC is much faster than MRP and a fair bit faster than MMC. For example, MRP ran for more than two and a half hours to solve a problem with 400 triplets over 200 taxa, where MXC ran 9 seconds. This is a factor of a thousand better. MMC also took more than 25 minutes to solve a problem with 10,000 triplets and 400 taxa, where MXC took 80 seconds. This can be attributed to the fact that, as was shown in the example in Section 4, the min cut approach produces more unbalanced trees, resulting in employing the min cut algorithm more times. The difference in running times between MMC and MXC can possibly be explained by the fact that the best-known deterministic algorithm for min cut runs in $O(nm + n^2 \log n)$ [26]. Although we don't have a bound for the Fiduccia-Matheyes bisection improvement, it seems that the embedding step with time $O(m)$ is the dominant. This difference became more evident for the larger inputs.

The largest experiment we performed in this distribution was with 50,000 triplets on 2,000 taxa. The running time on this data was a bit more than 6 minutes (383 seconds).

### 5.2 Geometric Distribution Results

In this type of experiment, we gave preferences to "close" over "far" triplets. That means that a triplet with diameter $d$ was chosen with probability $\frac{1}{d}$. We denote that as the

TABLE 1
Data from Experiments of Uniform Distribution of Triplet Selection

| #n | #t | %c | % Triplet fit | | | MAST fit | | | Robinson-Foulds fit | | | Running Time | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | MRP | MXC | MMC | MRP | MXC | MMC | MRP | MXC | MMC | MRP | MXC | MMC |
| 50 | 400 | 70 | 81 | 81.5 | 40 | 17 | 14 | 8 | 0.0/0.0 | 0.0/0.0 | 0.0/0.0 | 13 | 1 | 4 |
| 50 | 1000 | 70 | 79 | 80.9 | 38.4 | 18 | 22 | 7 | 6.2/6.2, | 8.3/15.4 | 0.0/0.0 | 18 | 2 | 14 |
| 50 | 1000 | 90 | 99 | 100 | 100 | 27 | 26 | 26 | 18.8/18.8, | 27.1/48.1, | 0.0/0.0 | 12 | 1 | 6 |
| 100 | 100 | 50 | 96 | 100 | 100 | 11 | 8 | 8 | 0/0 | 0/0 | 0/0 | 74 | 1 | 1 |
| 100 | 400 | 70 | 70 | 77.7 | 52.7 | 14 | 12 | 8 | 1/1 | 0/0 | 0/0 | 152 | 3 | 17 |
| 100 | 1000 | 70 | 72.3 | 73.1 | 33.6 | 23 | 18 | 8 | 2.0/2.0 | 2.0/5.1 | 0/0 | 642 | 4 | 16 |
| 100 | 2000 | 50 | 49.6 | 54.5 | 34.1 | 19 | 18 | 8 | 0/0 | 0/0 | 0/0 | 402 | 7 | 63 |
| 150 | 100 | 70 | 96 | 100 | 100 | 14 | 6 | 6 | 0/0 | 0/0 | 0/0 | 290 | 1 | 1 |
| 150 | 1000 | 70 | 70 | 73.7 | 43.7 | 16 | 17 | 8 | 0.7/0.7 | 0/0 | 0/0 | 1712 | 4 | 35 |
| 150 | 1000 | 90 | 89.2 | 90.7 | 42.3 | 25 | 16 | 8 | 1.4/1.4 | 2.7/10.8 | 0/0 | 2628 | 5 | 29 |
| 150 | 2000 | 90 | 87.9 | 90.3 | 34.8 | 31 | 24 | 8 | 2.0/2.0 | 6.1/25.0 | 0/0 | 4790 | 17 | 108 |
| 200 | 400 | 70 | 81.2 | 81.5 | 65.7 | 17 | 10 | 9 | 0/0 | 0/0 | 0/0 | 9047 | 9 | 25 |
| 200 | 1000 | 70 | 67.3 | 75.7 | 44.1 | 15 | 17 | 9 | 0/0 | 0/0 | 0/0 | 8581 | 23 | 200 |
| 200 | 4000 | 50 | - | 54.2 | 34.5 | - | 25 | 20 | - | 0/0 | 0/0 | - | 53 | 546 |
| 400 | 10000 | 50 | - | 53.5 | 36.1 | - | 31 | 10 | - | 0/0 | 0/0 | - | 80 | 1603 |
| 400 | 50000 | 50 | - | 50.6 | - | - | 55 | - | - | 0.5/1.6 | - | - | 320 | - |
| 600 | 4000 | 50 | - | 62.4 | - | - | 17 | - | - | 0/0 | - | - | 38 | - |
| 600 | 50000 | 50 | - | 51 | - | - | 56 | - | - | 0/0 | - | - | 961 | - |
| 800 | 20000 | 50 | - | 53.5 | - | - | 34 | - | - | 0/0 | - | - | 132 | - |
| 1000 | 50000 | 50 | - | 51.4 | - | - | 49 | - | - | 0/0 | - | - | 339 | - |
| 2000 | 50000 | 50 | - | 53.5 | - | - | - | - | - | 0/0 | - | - | 383 | - |

*The first three columns on the left indicate the number of taxa in the model tree (#n), the number of triplets drawn from the model tree (#t), and the percentage of correct triplets in the input (%c), respectively. "-" denotes that the problem took too long.*

geometric distribution. The reasoning for using that distribution is that, in general, we are less certain about the order of speciation of distantly related species than of more closely related species and, therefore, we "weigh" these distant triplets less. We believe this type of distribution is more realistic than the uniform distribution.

Table 2 depicts a sample of our experiments under the geometric distribution. It can be seen that both MRP and our method maintain the same superiority in terms of triplet fit score over MMC with an average of 2 percent advantage to MXC over MRP. MXC is still much faster than MRP. However, sometimes, in cases of (unrealistically) high error rate in this distribution, we needed to parameterize our heuristic to obtain better cuts. This means seeking a higher ratio between good and bad edges than the hard-coded ratio of 3. Normally, the ratio of 5 sufficed and at most a ratio of 7 was needed. This resulted in an increase in the running times (e.g., the line with 150 species, 2,000 triplets, and 50 percent correctness).

The MAST scores obtained under this distribution strictly surpass their equivalent in the uniform distribution (see, e.g., the line with 200 species, 1,000 triplets, and 70 percent correctness, or 150, 1,000, 50 percent). A somewhat interesting phenomenon is that, with this distribution, our method strictly outperforms MRP in terms of MAST score, in contrast to the uniform distribution, where it appears MRP has some advantage. It is notable that, even under this distribution, MRP has a better MAST

TABLE 2
Statistics on Experiments of Geometric Distribution of Triplet Selection

| #n | #t | %c | % Triplet fit | | | MAST fit | | | Robinson-Foulds fit | | | Running Time | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | MRP | MXC | MMC | MRP | MXC | MMC | MRP | MXC | MMC | MRP | MXC | MMC |
| 50 | 400 | 50 | 65.7 | 62 | 50.2 | 11 | 14 | 9 | 0/0 | 8.5/9.3 | 0/0 | 25 | 58 | 12 |
| 50 | 1000 | 70 | 65.3 | 70.3 | 43 | 21 | 29 | 1 | 21.3/21.3 | 55.3/59.1 | 8.5/11.4 | 44 | 7 | 26 |
| 50 | 2000 | 90 | 88.2 | 89.3 | 65.6 | 34 | 36 | 26 | 76.6/76.6 | 91.5/93.5 | 38.3/42.9 | 46 | 22 | 48 |
| 100 | 400 | 50 | 74.2 | 74 | 53 | 15 | 21 | 9 | 3.1/3.1 | 3.1/6 | 0/0 | 499 | 45 | 30 |
| 100 | 1000 | 50 | 62 | 63.4 | 38.1 | 17 | 24 | 12 | 1/1 | 7.2/8.3 | 0/0 | 806 | 30 | 78 |
| 100 | 1000 | 70 | 66.6 | 69.8 | 46.5 | 29 | 28 | 12 | 3.1/3.1 | 5.2/7.9 | 0/0 | 775 | 18 | 111 |
| 100 | 2000 | 90 | 82.7 | 85.5 | 49 | 39 | 45 | 21 | 34.0/34.0 | 43.3/55.3 | 5.2/6.6 | 411 | 10 | 53 |
| 150 | 1000 | 50 | 64.8 | 66.9 | 41.5 | 16 | 33 | 15 | 2/2 | 6.1/9.2 | 1.4/4.9 | 2564 | 65 | 43 |
| 150 | 2000 | 50 | 61.6 | 60 | 35 | 21 | 22 | 13 | 2/2 | 1.4/1.5 | 0.7/1.5 | 7553 | 1596 | 243 |
| 150 | 400 | 90 | 80.5 | 83.5 | 71.75 | 24 | 20 | 18 | 2/2 | 8.8/26.5 | 4.8/13.2 | 1114 | 4 | 13 |
| 200 | 1000 | 70 | 68.3 | 71.3 | 44.1 | 22 | 25 | 13 | 2.5/2.5 | 3.0/9.1 | 1.0/4.3 | 6317 | 11 | 66 |
| 200 | 400 | 90 | 83 | 86 | 73 | 19 | 19 | 15 | 2.0/2.0 | 3.0/10.7 | 1.5/5 | 3053 | 3 | 13 |
| 200 | 2000 | 90 | 83.9 | 86.35 | 38.65 | 84 | 93 | 15 | 7.6/7.6 | 13.7/31.8 | 1.0/3.3 | 10980 | 17 | 76 |
| 250 | 100 | 70 | 98 | 100 | 100 | 17 | 6 | 6 | 0/0 | 1.1/5.3 | 1.1/5.3 | 2036 | 2 | 3 |
| 250 | 2000 | 70 | 66.1 | 71.7 | 42 | 33 | 52 | 18 | 0.4/0.4 | 2.4/6.4 | 0.4/1.8 | 25114 | 29 | 92 |
| 250 | 1000 | 90 | 75.8 | 77.4 | 57.6 | 30 | 24 | 19 | 3.6/3.6 | 8.1/20.2 | 0.4/1.5 | 13626 | 38 | 217 |
| 300 | 2000 | 70 | 65 | 67.3 | 44.8 | 32 | 29 | 16 | 2.0/2.0 | 2.7/7.9 | 0.7/2.9 | 40176 | 38 | 302 |
| 300 | 400 | 90 | 88.5 | 97.2 | 91.7 | 25 | 22 | 20 | 1.0/1.0 | 1.0/4.5 | 0.7/2.5 | 27183 | 2 | 14 |
| 300 | 2000 | 90 | 71 | 74.9 | 49.8 | 37 | 46 | 19 | 5.7/5.7 | 9.1/24.1 | 1.0/3.9 | 35555 | 18 | 153 |

score when the triplets are sparse, so MXC and MMC satisfy all the triplets but leave an unresolved tree, as implied by the Aho et al. algorithm (e.g., the line with 250 species, 100 triplets, and 70 percent correctness).

As to the RF measure, it can be evidenced that we get much bigger numbers than with the uniform distribution for all methods. The superiority of MXC over MRP in the RF score is even bigger than was observed in the uniform distribution.

The above results lead to the assumption that perhaps this distribution of triplets along with the MXC algorithm is better for learning the true tree as compared to the uniform distribution of triplets. This also complies with other methods that rely on amalgamating small diameter subtrees (see, e.g., [16], [14], [10]).

## 6 Experiment on Real Data

Although our method was designed to handle input in the form of triplet trees, we wanted to test its behavior on real data. Real data normally comes in the form of trees over more than three species. When applied to a triplet based-method, a separate task is to generate a "representative set" of triplets that will lead to the best construction of a tree. In this paper, we used a straightforward way and generated *all* the triplets induced by *any* subtree. Of course, this approach is biased and gives more representation to higher ancestral vertices over more recent ancestral vertices. As will be shown in the sequel, this phenomenon is more dominant as the size of the subtrees increases. The real data we used for our experiments is composed of two data sets:

- The first data set contains 158 source trees with 267 marsupial species from 107 published studies [7]. The source trees were based on a wide range of data types, including molecular sequences, DNA

hybridization, karyotypes, and immunological, morphological, and behavioral data. The average number of species per tree is about 16.4, however, quite a few trees are over 50 taxa and one is over the complete set (267 taxa). The number of triplets generated is 2,380,724. Although the true tree is assumed to be known and is found in **TREEBASE** [20], we compared the results we obtained to either the set of triplets (triplet fit) or to the input subtrees (MAST and RF).

The results on this data set show a strict advantage to MRP. The triplet fit achieved by MRP was 98.2 percent versus 96 percent by MXC. Both the MAST score and the number of common edges between the supertree and each of the subtrees were normalized by the number of species in the subtree. The final scores are the average of the latter over all subtrees. For these measures, MRP achieved an average of 0.71 in the MAST score per species versus 0.66 of MXC. The average number of common edges per species obtained by MRP is 0.54 versus 0.44 by MXC. Additionally, even the running times for constructing the supertree were very similar, with a little advantage to MXC.

- The second data set is the RCBL 500 data set [22] of 500 taxa. This input is of 199 source trees each over at most 10 taxa. The number of triplets generated is 7,582. On this data set, the results show a strict advantage to our method. The triplet fit achieved by MRP was 78.1 percent versus 86.3 percent by MXC. The average number of common edges per species in the input trees was 0.57 by MRP versus 0.59 by MXC. MRP achieved an average MAST score per species of 0.72 versus 0.77 by MXC. The running times for constructing the trees were a few hours by MRP versus a few seconds by MXC.

The above results suggest that, for a triplet-based reconstruction, naively taking all the triplets defined by each subtree might not represent each subtree well. Big input subtrees generate many triplets and, therefore, dominate over small trees. Moreover, when the input is composed from big trees, the number of triplets generated becomes exceedingly large. This in turn causes to prohibitive running times.

We also see that the type of the input subtrees has a crucial influence on the reconstruction methods. While MRP copes well with big subtrees without generating too much data, a straightforward approach for triplet selection is not enough. On the other hand, it seems that, for small input subtrees, MRP suffers from the same shortcomings demonstrated by the simulations on geometric distribution of low MAST score and RF score.

## 7 Conclusion and Further Work

In this work, we described a novel heuristic that extends the min cut criterion for bipartitioning the taxa set in the presence of inconsistent rooted triplets. The approach relies on adding edges to the traditional connectivity graph and distinguishing between good and bad edges. The heuristic seeks a cut in the graph that maximizes the ratio between good and bad edges at every step. We showed experimentally that, although optimizing this criterion implies solving an NP-hard problem, a simple heuristic suffices to provide good performances. This in turn yields a very fast algorithm that outperforms even the theoretically efficient algorithms of min cut in terms of running time. Moreover, we showed that, for the type of inputs we studied here, the performance of triplet-based methods can exceed those of the heavier character-based methods.

We want to emphasize that there are few major questions to be answered in this direction:

- While triplet-based methods try to maximize the number of satisfied triplets, this does not necessarily go along with optimizing the other measures such as MAST or RF distance. We would like to study the difference between these measures and to come to conclusions about how they measure tree similarity.
- We saw that the difference in the performance between MRP and MXC on real data changes with the type of input subtrees. In particular, for one input, MRP is still superior over naively supplying all triplets to MXC. It is of interest to seek a general method for selecting triplets for any size of input subtrees.

## Acknowledgments

The authors would very much like to thank David Fernandez Baca and Duhong Chen for their help with the many technicalities involved and Usman Roshan for providing the rcbL data. They also thank Benny Chor, Oliver Eulenstein, Rod Page, Mauricio Resende, and Mike Steel for helpful discussions. Finally, they thank the two anonymous referees who gave very helpful comments on the manuscript. This research was supported by US National Institutes of Health Grant R01-HG02362-02. Satish Rao was partially supported by US National Science Foundation Award 0331494.

## References

[1] S. Arora, S. Rao, and U. Vazirani, "Expander Flows, Geometric Embeddings and Graph Partitioning," *Proc. Symp. Foundations of Computer Science*, 2004.

[2] 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. Computing*, vol. 10, no. 3, pp. 405-421, 1981.

[3] B.R. Baum, "Combining Trees as a Way of Combining Data Sets for Phylogenetic Inference," *Taxon*, vol. 41 pp. 3-10, 1992.

[4] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg, "Constructing Phylogenies from Quartets: Elucidation of Eutherian Superordinal Relationships," *J. Computational Biology*, vol. 5, no. 3, pp. 377-390, 1998.

[5] D. Bryant, "Hunting for Trees, Building Trees and Comparing Trees: Theory and Method in Phylogenetic Analysis," PhD thesis, Dept. of Math., Univ. of Canterbury, New Zealand, 1997.

[6] D.J. Bryant and M.A. Steel, "Extension Operations on Sets of Leaf-Labelled Trees," *Advances in Applied Math.*, vol. 16, no. 4, pp. 425-453, 2000.

[7] M. Cardillo, O.R.P. Bininda-Emonds, E. Boakes, and A. Purvis, "A Species-Level Phylogenetic Supertree of Marsupials," *J. Zoology*, vol. 264, no. 1, pp. 11-31, 2004.

[8] D. Chen, O. Eulenstein, D. Fernandez-Baca, and M. Sanderson, "Supertrees by Flipping," *Proc. Int'l Computing and Combinatorics Conf. (COCOON),* 2002.
[9] O. Eulenstein, D. Chen, J.G. Burleigh, D. Fernandez-Baca, and M.J. Sanderson, "Performance of Flip Supertrees with a Heuristic Algorithm," *Systematic Biology,* vol. 53, no. 2, pp. 299-308, 2004.
[10] P. Erdos, M. Steel, L. Szekely, and T. Warnow, "A Few Logs Suffice to Build (Almost) All Trees (I)," *Random Structures and Algorithms,* vol. 14, pp. 153-184, 1999.
[11] C.M. Fiduccia and R.M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. Design Automation Conf.,* pp. 175-181, 1982.
[12] M.X. Goemans and D.P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *J. ACM,* vol. 42, no. 6, pp. 1115-1145, Nov. 1995.
[13] M.R. Henzinger, V. King, and T. Warnow, "Constructing a Tree from Homeomorphic Subtrees, with Applications to Computational Evolutionary Biology," *Proc. ACM-SIAM Symp. Discrete Algorithms (SODA),* pp. 333-340, 1996.
[14] D. Huson, S. Nettles, and T. Warnow, "Disk-Covering, a Fast Converging Method for Phylogenetic Tree Reconstruction," *J. Computational Biology,* no. 6, pp. 369-386, 1999.
[15] B.W. Kernighan and S. Lin, "An Encient Heuristic Procedure for Partitioning Graphs," *The Bell Systems Technical J.,* vo. 29, no. 2, pp. 291-307, 1970.
[16] E. Mossel, "Distorted Metrics on Trees and Phylogenetic Forests," submitted, 2005.
[17] S. Moran, S. Rao, and S. Snir, "Using Semi-Definite Programming to Enhance Supertree Resolvability," *Proc. Fifth Int'l Workshop Algorithms in Bioinformatics (WABI),* pp. 89-103, Oct. 2005.
[18] S. Moran and S. Snir, "Convex Recoloring of Strings and Trees: Definitions, Hardness Results and Algorithms," *J. Computer and System Sciences,* 2005.
[19] R.D.M. Page, "Modified Mincut Supertrees," *Proc. Int'l Workshop Algorithms in Bioinformatics (WABI),* R. Guigo and D. Gusfield, eds., 2002.
[20] W. Piel, M. Donoghue, and M. Sanderson, "TreeBASE: A Database of Phylogenetic Knowledge," Nat'l Inst. for Environmental Studies, Tsukuba, Japan, 2002, http://www.treebase.org.
[21] M.A. Ragan, "Matrix Representation in Reconstructing Phylogenetic-Relationships among the Eukaryotes," *Biosystems,* vol. 28, pp. 47-55, 1992.
[22] K. Rice, M. Donoghue, and R. Olmstead, "Analyzing Large Datasets: *rbcl* 500 Revisited," *Systematic Biology,* vol. 46, no. 4, pp. 554-563, 1997.
[23] M. Steel, A. Dress, and S. Boker, "Simple but Fundamental Limitations on Supertree and Consensus Tree Methods," *Systematic Biology,* vol. 49, pp. 363-368, 2000.
[24] C. Semple and M. Steel, "A Supertree Method for Rooted Trees," *Discrete Applied Math.,* vol. 103, pp. 147-158, 2000.
[25] M. Steel, "The Complexity of Reconstructing Trees from Qualitative Characters and Subtress," *J. Classification,* vol. 9, no. 1, pp. 91-116, 1992.
[26] M. Stoer and F. Wagner, "A Simple Min-Cut Algorithm," *J. ACM,* vol. 44, pp. 585-591, 1997.
[27] L. Vandenberghe and S. Boyd, "Semidefinite Programming," *SIAM Rev.,* vol 38, no. 1, pp. 49-95, 1996.

**Sagi Snir** received the BA degree from Bar Ilan University at Israel majoring in computer science and economics. He received the MSc and PhD degrees in computer science from the Technion, Israel. He is currently a postdoctoral researcher at the University of California at Berkeley. Before working on the PhD, he worked in various information technologies companies, including IBM Haifa Research Lab. His main research interest is computational biology and, in particular, phylogenetics.

**Satish Rao** received the BS degree from the Massachusetts Institute of Technology (MIT) in electrical engineering, computer science, and physics. He received the SM and PhD (1989) degrees in computer science from MIT. He is a professor of computer science at the University of California at Berkeley. He then held a postdoctoral position at Harvard College for a year before becoming a research scientist at the NEC Research Institute in 1991. In January 1999, he joined Akamai as a senior research scientist.