The Complexity of Inferring a Minimally Resolved Phylogenetic Supertree

Jesper Jansson[1], Richard S. Lemence[1], and Andrzej Lingas[2]

[1] Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan
Jesper.Jansson@ocha.ac.jp, rslemence@gmail.com
Funded by the Special Coordination Funds for Promoting Science and Technology
[2] Department of Computer Science, Lund University, 22100 Lund, Sweden
Andrzej.Lingas@cs.lth.se

Abstract. A recursive algorithm by Aho, Sagiv, Szymanski, and Ullman [1] forms the basis for many modern rooted supertree methods employed in Phylogenetics. However, as observed by Bryant [4], the tree output by the algorithm of Aho *et al.* is not always minimal; there may exist other trees which contain fewer nodes yet are still consistent with the input. In this paper, we prove strong polynomial-time inapproximability results for the problem of inferring a minimally resolved supertree from a given consistent set of rooted triplets (MinRS). We also present an exponential-time algorithm for solving MinRS exactly which is based on tree separators. It runs in $2^{O(n \log k)}$ time when every node is required to have at most k children which are internal nodes and where n is the cardinality of the leaf label set of the input trees.

Keywords: Phylogenetic tree; rooted triplet; minimally resolved supertree; NP-hardness; tree separator.

1 Introduction

Phylogenetic trees are leaf-labeled trees which are used to represent tree-like evolutionary history. To infer a reliable phylogenetic tree containing a large number of leaves is often a difficult task because of the computational complexity of the underlying optimization problems. One approach which has become increasingly popular in recent years is the divide-and-conquer-based *supertree approach* [2,6,9,12], which first uses a computationally intense method to reconstruct highly accurate trees for small, partially overlapping subsets of the leaf label set, and then applies a combinatorial algorithm to merge the small trees into one large tree called a *supertree*.

One of the common criticisms of supertrees is that they make statements about evolutionary relationships among leaves that are not always directly supported by any one of the input trees, which can create false groupings in the form of "spurious novel clades" [2]. Therefore, a natural idea is to try to avoid making more such statements than necessary to obtain a supertree, and thus to introduce as little unsupported branching information as possible. For this reason, minimally resolved supertrees, i.e., trees that contain as few internal nodes as possible while still being consistent with the input, are important in Bioinformatics.

Several alternative supertree methods whose formal definitions differ have been developed; see, e.g., the survey paper by Bininda-Emonds [2] for an extensive list of references. A classic algorithm by Aho, Sagiv, Szymanski, and Ullman [1] named BUILD (see Section 2.2 below for a short description) can be used to merge a given set of *non-conflicting* rooted phylogenetic trees. Due to its simplicity and efficiency, it is used as a starting point of many rooted supertree methods such as the ones presented in [8,14,15,16] for combining a set of *conflicting* trees. These methods imitate the behavior of BUILD until a conflict occurs, at which point the so-called auxiliary graph consists of a single connected component which is then split into smaller components by removing some edges (different methods use different rules to do this), and then the method is executed recursively on each newly created component. Typically, such methods will all yield the same output as BUILD in the ideal case where the input set of trees contains no conflicts; hence, to understand these methods, it is important to fully understand the properties of the trees which are output by BUILD.

A surprising fact about BUILD is that it does not always produce a tree with the minimum possible number of internal nodes. This was first observed by Bryant in [4]. We generalize Bryant's example in Section 2.3 below to show that BUILD may in fact output a tree containing $\Omega(n)$ times more internal nodes than necessary, where n is the cardinality of the leaf label set of the input trees.

A binary phylogenetic tree with exactly three leaves is called a *rooted triplet*. Rooted triplets are a special case of phylogenetic trees, so hardness results concerning the computational complexity of inferring supertrees from rooted triplets will directly carry over to the corresponding problems for general inputs. Moreover, as explained in [9], the branching information contained in any rooted, binary phylogenetic tree with m leaves can be represented by a set of $O(m)$ rooted triplets (one rooted triplet per edge in the tree). From here on, we therefore focus on inputs which consist of rooted triplets.

1.1 Our Results and Organization of the Paper

We study the computational complexity of the problem of inferring a minimally resolved supertree from a given consistent set of rooted triplets over a leaf label set of cardinality n, called MinRS for short.

The paper is organized as follows. Section 2 defines the MinRS problem formally and surveys previous work. Section 3 proves two strong negative results: (1) the decision version of MinRS is NP-hard for any fixed number of internal nodes larger than or equal to 4; and (2) MinRS cannot be approximated within $n^{1-\epsilon}$ for any constant $0 < \epsilon < 1$ in polynomial time, unless P = NP. Then, Section 4.1 describes a simple algorithm for the decision version of MinRS which runs in $O^*(f(q) \cdot q^n)$ time, where q is the allowed number of internal nodes and $f(q)$ is the number of rooted, unlabeled trees with q nodes. Section 4.2 presents a more sophisticated exponential-time exact algorithm based on tree separators that runs in $2^{O(n \log k)}$ time, where every node is required to have at most k children which are internal nodes. Section 5 contains some final remarks.

2 Preliminaries

2.1 Basic Definitions

We will use the following definitions and notation.

To simplify the presentation, any node in a tree is considered to be an ancestor as well as a descendant of itself. For any nodes u, v in a tree, in case u is a descendant of v and $u \neq v$ then we write $u \prec v$ and call u a *proper* descendant of v. The *lowest common ancestor of* u *and* v, denoted by $lca(u, v)$, is the node w such that both u and v are descendants of w and $w \prec x$ holds for every other node x which is an ancestor of both u and v. The set of leaves in a tree T is denoted by $\Lambda(T)$.

A *phylogenetic tree* is a rooted, unordered tree whose leaves are distinctly labeled. Since the leaves in a phylogenetic tree are uniquely labeled, we will refer to them by referring to their labels. A *rooted triplet* is a phylogenetic tree with exactly three leaves in which every internal node has exactly two children, and we let $xy|z$ denote the rooted triplet having leaf label set $\{x, y, z\}$ that satisfies $lca(x, y) \prec lca(x, z) = lca(y, z)$.

Let T be a phylogenetic tree. For any $\{x, y, z\} \subseteq \Lambda(T)$, if the relation $lca(x, y) \prec lca(x, z) = lca(y, z)$ holds in T then the rooted triplet $xy|z$ is said to be *consistent with* T. A given set $\mathcal{R}$ of rooted triplets and a given phylogenetic tree T are *consistent* if every $t \in \mathcal{R}$ is consistent with T. Lastly, any given set $\mathcal{R}$ of rooted triplets is called *consistent* if there exists a tree which is consistent with $\mathcal{R}$ (otherwise, $\mathcal{R}$ is called *inconsistent*).

When $\mathcal{R}$ is given, we denote the set of all leaf labels which occur in $\mathcal{R}$ by L, i.e., we define $L = \bigcup_{t \in \mathcal{R}} \Lambda(t)$. Throughout the paper, we use the notation $n = |L|$ and $k = |\mathcal{R}|$. Given an input set $\mathcal{R}$ of rooted triplets, it is possible to efficiently check whether $\mathcal{R}$ is consistent and if so, to construct a phylogenetic tree consistent with $\mathcal{R}$, by a classic algorithm of Aho, Sagiv, Szymanski, and Ullman [1] named BUILD. The algorithm is described in Section 2.2 below.

Finally, for any consistent set $\mathcal{R}$ of rooted triplets, we say that a phylogenetic tree which is consistent with $\mathcal{R}$ and contains as few internal nodes as possible is a *minimally resolved* supertree for $\mathcal{R}$.

2.2 The Algorithm of Aho *et al.* [1] (BUILD)

In this subsection, we briefly review the algorithm of Aho, Sagiv, Szymanski, and Ullman [1]. The algorithm, referred to as BUILD, constructs a phylogenetic tree consistent with an input set $\mathcal{R}$ of rooted triplets over a leaf label set L, if such a tree exists. In case such a tree does not exist, the algorithm outputs *null*.

BUILD is a top-down, recursive algorithm. The main idea of the algorithm is to first partition the leaf set L into *blocks* according to the rooted triplets in $\mathcal{R}$. Then, the algorithm outputs a tree consisting of a root node whose children are the roots of the trees obtained by recursing on each block. The base case of the recursion is when the leaf set consists of a single leaf.

To perform the partitioning into blocks for any subset $L' \subseteq L$ with $|L'| > 1$, BUILD uses an *auxiliary graph* $\mathcal{G}(L')$. The auxiliary graph for any $L' \subseteq L$ is defined as $\mathcal{G}(L') = (L', E)$, where E contains the edge $\{x, y\}$ if and only if there is some rooted triplet of the form $xy|z$ in $\mathcal{R}$ with $x, y, z \in L'$. After constructing $\mathcal{G}(L')$, the algorithm computes the connected components in $\mathcal{G}(L')$ and lets each such connected component define one block of L'. If, at any point of its execution, $|L'| > 1$ yet L' contains just one block, then BUILD terminates and outputs *null*. This approach is motivated by Proposition 1 below together with the key observation that for any rooted triplet $xy|z$ consistent with a phylogenetic tree T, the leaves labeled by x and y cannot descend from two different children of the root of T, i.e., x and y must belong to the same block. (For a formal proof of correctness, see [1].)

Proposition 1 (Aho, Sagiv, Szymanski, and Ullman [1]). *If $\mathcal{G}(L)$ has only one connected component and $|L| > 1$ then $\mathcal{R}$ is not consistent with any phylogenetic tree.*

The running time of the original implementation of BUILD [1] was $O(nk)$, where $n = |L|$ and $k = |\mathcal{R}|$. Henzinger *et al.* [9] later presented a faster implementation of this algorithm, and replacing the dynamic graph connectivity data structure used by [9] by a more recent one [10] further reduces the complexity of the algorithm to $\min\{O(n + k \log^2 n),\ O(k + n^2 \log n)\}$ time [11].

2.3 Definition of MinRS

Bryant [4] noted that the BUILD algorithm of Aho *et al.* [1] does not always produce a minimally resolved supertree consistent with a given set of rooted triplets. In the example provided in Section 2.5.2 of [4], Bryant considered the set $\mathcal{R} = \{bc|a,\ bd|a,\ ef|a,\ eg|a\}$. As demonstrated in Figure 13 in [4], BUILD will construct a tree consistent with $\mathcal{R}$ which contains *three* internal nodes (a root node along with two internal nodes which are the parents of the leaves b, c, d and e, f, g, respectively), whereas the optimal solution is a tree containing *two* internal nodes (a root node and an internal node to which the leaves b, c, d, e, f, g are directly attached).

We can simplify Bryant's example to $\{bc|a,\ ef|a\}$. Then, if we extend the example as follows:

$$\mathcal{R} = \{x_1x_2|x_0,\ x_3x_4|x_0,\ \ldots,\ x_{2i-1}x_{2i}|x_0)\},$$

we obtain a consistent set of rooted triplets for which BUILD will construct a tree having $i+1$ internal nodes. However, the tree consisting of a root node with two children c_1 and c_2, where c_1 is a leaf labeled by x_0 and c_2 is an internal node with $2i$ children which are leaves labeled by $x_1, x_2, \ldots, x_{2i}$, contains exactly two internal nodes and is also consistent with $\mathcal{R}$. This shows that asymptotically, the BUILD algorithm of Aho *et al.* may produce a tree with $\Omega(n)$ times more internal nodes than the minimally resolved supertree, where n is the cardinality of the leaf label set.

From this observation, a natural question arises: When a consistent set of rooted triplets $\mathcal{R}$ is given, how efficiently can one construct a *minimally resolved* supertree consistent with $\mathcal{R}$? Formally, we define:

The Minimally Resolved Supertree Consistent with Rooted Triplets Problem (MinRS)

Instance: A set $\mathcal{R}$ of rooted triplets with leaf set L.
Output: A rooted, unordered tree whose leaves are distinctly labeled by L which has as few internal nodes as possible and which is consistent with every rooted triplet in $\mathcal{R}$, if such a tree exists; otherwise, *null*.

2.4 Related Work

Besides Bryant [4], other authors such as Henzinger, King, and Warnow [9] have also previously considered the problem of inferring a minimally resolved supertree from a set of rooted triplets. Unfortunately, Henzinger *et al.* [9] incorrectly assumed that the BUILD algorithm of Aho *et al.* [1] always constructs a minimally resolved supertree. According to the proof of Theorem 4 in [9], the tree constructed by Algorithm A' of Henzinger *et al.* [9] is identical to the tree constructed by the BUILD algorithm; therefore, our example from Section 2.3 above also implies that Algorithm A' may output a tree with $\Omega(n)$ times more internal nodes than a minimally resolved supertree. This means that the minimality claim in Theorem 4 in [9] is not correct.

A *fan triplet* is a a phylogenetic tree consisting of a root node to which three leaves are directly attached. For any phylogenetic tree T and any $\{x, y, z\} \subseteq \Lambda(T)$, if $lca(x, y) = lca(x, z) = lca(y, z)$ holds in T then the fan triplet with leaves x, y, z is *consistent with* T. In Section 2.6.3 of [4], Bryant studied a kind of "dual" problem to MinRS called Most Resolved Compatible Tree, in which the input is a consistent set $\mathcal{R}$ of (rooted and fan) triplets on a leaf set L, and the objective is to construct a phylogenetic tree leaf-labeled by L with *the largest possible* number of internal edges which is consistent with $\mathcal{R}$. (Here, trees containing an internal node with a single child are not allowed.) Note that if $\mathcal{R}$ contains rooted triplets only then the problem is trivial since any binary tree which is consistent with $\mathcal{R}$ will give an optimal solution. However, in the general case, Most Resolved Compatible Tree is NP-hard [4].

For a recent survey of other optimization problems related to rooted triplets consistency (for example, computing a maximum cardinality subset $\mathcal{R}'$ of an inconsistent set $\mathcal{R}$ of rooted triplets such that $\mathcal{R}'$ is consistent), see Section 2 in [5].

3 Polynomial-Time Inapproximability of MinRS

In this section, we establish a strong polynomial-time inapproximability result for MinRS, namely that MinRS cannot be approximated within $n^{1-\epsilon}$ for any constant $0 < \epsilon < 1$ in polynomial time, unless P = NP. We will obtain this result by reducing the CHROMATIC NUMBER problem to MINRS.

First, recall that for any undirected graph $G = (V, E)$ and any positive integer K, a *K-coloring of G* is a partition of V into (possibly empty) disjoint subsets $V_1, V_2, \ldots, V_K$ called *color classes* such that for any $\{v, w\} \in E$, it holds that v and w belong to different color classes. A graph G is called *K-colorable* if there exists a K-coloring of G. The CHROMATIC NUMBER problem is defined as:

CHROMATIC NUMBER

Instance: An undirected graph $G = (V, E)$.
Output: The smallest integer K such that G is K-colorable.

Zuckerman [17] proved that CHROMATIC NUMBER is NP-hard to approximate within $|V|^{1-\epsilon}$ for every $0 < \epsilon < 1$. Moreover, the decision version of the problem, i.e., to determine if an undirected graph G is K-colorable for a particular value of K, is easily solvable in polynomial time when $K = 2$ but known to be NP-hard for any fixed positive integer $K \geq 3$; see, e.g., [7].

We now describe the reduction. Let $G = (V, E)$ be any given instance of CHROMATIC NUMBER. Without loss of generality, we assume that V contains at least two vertices and that G is connected. Construct an instance of MINRS as follows. Let $L = \{v_1, v_2 : v \in V\}$ be a set of $2|V|$ new leaf labels and define $\mathcal{R} = \{v_1v_2|w_1,\ v_1v_2|w_2,\ w_1w_2|v_1,\ w_1w_2|v_2 : \{v, w\} \in E\}$. Clearly, the reduction can be carried out in polynomial time.

Then we have:

Lemma 1. *If G is K-colorable then there exists a tree which is consistent with $\mathcal{R}$ and contains $K + 1$ internal nodes.*

Proof. Since $G = (V, E)$ is K-colorable, we can partition the vertex set V of G into K disjoint color classes $V_1, V_2, \ldots, V_K$. Order the color classes so that $V_1, V_2, \ldots, V_j$ are non-empty and $V_{j+1} = \cdots = V_K = \emptyset$, where $j \leq K$. Define a tree T having exactly $K + 1$ internal nodes as follows. (See Fig. 1 for an illustration.)

- Let the root of T be one end of a path of length $K - j$ and let a_0 be the other end of the path. Let a_0 have j children $a_1, a_2, \ldots, a_j$.
- For each $i \in \{1, 2, \ldots, j\}$ and each $v \in V_i$, attach two leaves labeled by v_1 and v_2 to the node a_i.

Consider any rooted triplet in $\mathcal{R}$. It is of the form $v_1v_2|w_1$, where $\{v, w\} \in E$; furthermore, since $\{v, w\} \in E$, both of the vertices v and w cannot belong to the same color class V_i. Thus, the parent of the leaves v_1 and v_2 in T is different from the parent of the leaves w_1 and w_2, and so $v_1v_2|w_1$ is consistent with T. Therefore, $\mathcal{R}$ is consistent with T. $\square$

Lemma 2. *If there exists a tree which is consistent with $\mathcal{R}$ and contains $K + 1$ internal nodes then G is K-colorable.*

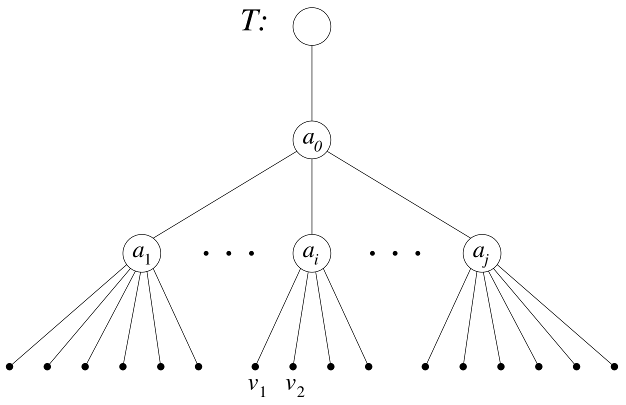

Fig. 1. Illustrating the proof of Lemma 1. In this example, there is one empty color class (i.e., $K - j = 1$), so the path from the root of T to the internal node a_0 has length 1. For each vertex $v \in V$, where v belongs to the color class V_i, the leaves v_1, v_2 in T are directly attached to the internal node a_i.

Proof. Let T be a tree with $K + 1$ internal nodes which is leaf-labeled by L and consistent with $\mathcal{R}$. Let c_0 be the root of T and denote the other internal nodes of T by $c_1, c_2, \ldots, c_K$ arbitrarily. For every $i \in \{0, 1, \ldots, K\}$, associate a (possibly empty) subset $C_i \subseteq V$ with the internal node c_i, defined as follows: for each $v \in V$, if $lca(v_1, v_2) = c_i$ in T then let $v \in C_i$. It follows directly that for any $i, j \in \{0, 1, \ldots, K\}$ with $i \neq j$, the subsets C_i and C_j are disjoint.

Observe that every $v \in V$ belongs to at least one edge in E of the form $\{v, w\}$ (otherwise, the graph G would not be connected), and thus, by the construction of $\mathcal{R}$, the rooted triplets $v_1v_2|w_1$, $v_1v_2|w_2$, $w_1w_2|v_1$, and $w_1w_2|v_2$ belong to $\mathcal{R}$. Since $v_1v_2|w_1$ is consistent with T, it holds that $lca(v_1, v_2)$ is a proper descendant of $lca(v_1, w_1)$, i.e., $lca(v_1, v_2)$ cannot be the root of T. We have just shown that $C_0 = \emptyset$.

Next, we claim that for any two vertices $v, w \in V$, if $\{v, w\} \in E$ then v and w cannot belong to the same subset C_i. For the purpose of obtaining a contradiction, suppose that $v, w \in C_i$. Then $lca(v_1, v_2)$ and $lca(w_1, w_2)$ are the same node in T according to the definition of C_i. By transitivity, at least one of $lca(v_1, w_1)$, $lca(v_1, w_2)$, $lca(v_2, w_1)$, and $lca(v_2, w_2)$ is also equal to this node. However, since T is consistent with the rooted triplets $v_1v_2|w_1$, $v_1v_2|w_2$, $w_1w_2|v_1$, and $w_1w_2|v_2$, it follows from the definition of "consistent with" that the node $lca(v_1, v_2)$ is a proper descendant of (and hence different from) $lca(v_1, w_1)$ as well as of $lca(v_1, w_2)$, and in the same way that $lca(w_1, w_2)$ is a proper descendant of $lca(v_2, w_1)$ and of $lca(v_2, w_2)$. This yields a contradiction, so the claim must hold.

Thus, the partition of V into disjoint subsets $C_1, C_2, \ldots, C_K$ gives a K-coloring of G, and so G is K-colorable. $\square$

Theorem 1. MinRS *cannot be approximated within* $n^{1-\epsilon}$ *for any constant* $0 < \epsilon < 1$ *in polynomial time, unless* $P = NP$.

Proof. Follows from Lemmas 1 and 2 together with the fact that Chromatic Number is NP-hard to approximate within $|V|^{1-\epsilon}$ for every $0 < \epsilon < 1$ [17]. □

Since the decision version of Chromatic Number is NP-hard for any fixed positive integer $K \geq 3$ (see, e.g., [7]), using the above reduction and applying Lemmas 1 and 2 also yields:

Corollary 1. *The decision version of* MinRS *is NP-hard for any fixed positive integer $q \geq 4$, where q is the allowed number of internal nodes.*

4 Exact Algorithms for MinRS

4.1 A Brute-Force Algorithm

We can solve the decision version of MinRS with a simple brute-force algorithm as follows.

- Let q be the allowed number of internal nodes.
- Generate all possible trees having q nodes and for each one try all q^n ways of attaching the n leaves in L to the q different nodes. For each obtained tree, check if it is consistent with $\mathcal{R}$ in polynomial time. If at least one such tree exists then output "yes"; otherwise, output "no".

This yields:

Theorem 2. *For any given positive integer q, the decision version of* MinRS *can be solved in $O^*(f(q) \cdot q^n)$ time, where $f(q)$ is the number of rooted, unlabeled trees with q nodes.*

It is known that $f(q) \sim c \cdot d^q \cdot q^{-3/2}$, where $c = 0.439924\ldots$ and $d = 2.955765\ldots$ [13]. Thus, the algorithm runs in exponential time for $q = O(1)$.

4.2 An Exponential-Time Algorithm for a Restricted Case of MinRS

The derivation of our main result in this section relies on the following variant of the tree separator theorem. A *non-leaf child* of a node v in a tree is a child of v which is an internal node, and for any rooted tree T and node v in T, the notation T_v means the subtree of T rooted at v.

Lemma 3. *Let T be a rooted tree with n leaves. There is a node v such that the subtree T_v contains strictly greater than $\frac{n}{2}$ leaves but for each non-leaf child w of v, the subtree T_w has at most $\frac{n}{2}$ leaves.*

Proof. We start from the root of T and perform the following procedure. If the root satisfies the condition then we set v to it and stop. Otherwise, we set v to the child of the root with the largest number of leaves and iterate the procedure for T_v.

Note that whenever a new iteration is applied to T_v then T_v has to have more than $\frac{n}{2}$ leaves. It follows from the finiteness of T that eventually a node v satisfying the condition will be found. □

We now use Lemma 3 to design a $2^{O(n \log k)}$-time procedure for the variant of MINRS where every internal node is allowed to have at most k non-leaf children.

The procedure is recursive. We enumerate all partitions of the leaf set L corresponding to the condition in Lemma 3. Then, we recursively apply the procedure on the resulting leaf subsets, possibly augmented by a dummy leaf, modifying the rooted triplets accordingly.

A partition Q corresponding to the condition in Lemma 3 has two levels. See Fig. 2. Firstly, it splits the set L of leaves into a set L' corresponding to T_v of size strictly greater than $\frac{n}{2}$ and its complement $L \setminus L'$ corresponding to $T \setminus T_v$. Secondly, Q splits L' into $k' \leq k$ sets $L'_1, \ldots, L'_{k'}$ corresponding to the non-leaf children of v in the condition, each of the sets of size at most $\frac{n}{2}$, and a number of singletons corresponding to the leaves pending on v.

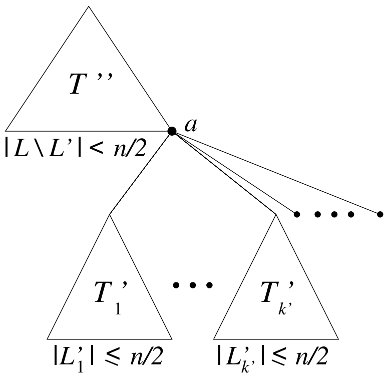

Fig. 2. According to Lemma 3, T has a node whose removal would divide the leaf set L into a subset L' containing strictly more than $\frac{n}{2}$ leaves and its complement $L \setminus L'$, and L' would be further partitioned into subsets $L'_1, \ldots, L'_{k'}$ of at most $\frac{n}{2}$ leaves each as well as a number of singletons.

If there is a rooted triplet $xy|z$ where $x, z \in L'$ and $y \in L \setminus L'$ then we can disregard Q. On the other hand, if $x, y \in L'$ and $z \in L \setminus L'$ then $xy|z$ is satisfied by Q and the triplet can be disregarded. As for the sets $L'_1, \ldots, L'_{k'}$ and the remaining leaf singletons, for each rooted triplet of the form $xy|z$ where $x, y, z \in L'$, if x, y are not in the same set L'_l then we can also disregard Q.

Otherwise, we augment $L \setminus L'$ by a dummy leaf a and for each rooted triplet of the form $xy|z$ where $x,\ z \in L \setminus L'$ and $y \in L'$, we form the rooted triplet $xa|z$. Analogously, for each rooted triplet of the form $xy|z$ where $x,\ y \in L \setminus L'$ and $z \in L'$, we form the rooted triplet $xy|a$.

For each such remaining partition Q, we run our procedure recursively on $L \setminus L' \cup \{a\}$ with the original set $\mathcal{R}$ of rooted triplets restricted to $L \setminus L'$ and the set of additional rooted triplets containing the dummy leaf a. Let T'' be the tree returned by the procedure.

We run also our procedure on each of the sets $L'_1, \ldots, L'_{k'}$ obtaining trees $T'_1, T'_2, \ldots, T'_{k'}$. Then, we make the trees as well as the singleton leaves children of the leaf a in the tree T'', and put the resulting tree on a candidate list.

Finally, we return the tree on the candidate list which has the smallest number of internal nodes.

The correctness of our procedure follows from Lemma 3 and the fact that we can join T'' with $T'_1, T'_2, \ldots, T'_{k'}$ in the described way at the dummy leaf a. The additional rooted triplets with a representing any leaf in L' satisfied by T'' make the join possible.

Let us estimate the time complexity $T(n)$ of our procedure in terms of the number of leaves n. Note that the number of rooted triplets is $O(n^3)$. The number of partitions considered and the time needed to generate them are trivially $O((k+2)^n) = 2^{O(n \log k)}$. Each of the at most $k+1$ recursive calls is applied to a set of leaves of size at most $\frac{n}{2}$. Thus, the total time complexity of processing the partitions is $2^{O(n \log k)}((k+1)T(\frac{n}{2}) + n^{O(1)})$. By the inequality $2^{\alpha n \log k} 2^{c(\frac{n}{2}) \log k} \leq 2^{cn \log k}$ for $c \geq 2\alpha$, $k \geq 2$, and $n \geq 3$, we obtain $T(n) = 2^{O(n \log k)}$.

Theorem 3. *The problem of constructing a minimally resolved tree consistent with a set $\mathcal{R}$ of rooted triplets on a leaf set L under the restriction that each node has at most k non-leaf children, where $k \geq 2$, is solvable in $2^{O(n \log k)}$ time.*

Any tree with n leaves which is consistent with $\mathcal{R}$ can easily be converted into a tree consistent with $\mathcal{R}$ where each node has at most k non-leaf children by increasing the number of internal nodes by an $O(\log_k n)$ multiplicative factor. (Simply connect each internal node v to its non-leaf children C_v via a k-ary tree of depth $O(\log_k n)$ having C_v as leaves.) Hence, we obtain the following corollary.

Corollary 2. MinRS *can be approximated within a $O(\log_k n)$ factor in time $2^{O(n \log k)}$.*

5 Concluding Remarks

Recall that at each recursion level, the BUILD algorithm of Aho *et al.* [1] partitions the leaf set into blocks by computing the connected components in the auxiliary graph, and then represents each block by one node in the tree. A simple idea to reduce the number of internal nodes in the tree produced by BUILD is to merge blocks while ensuring that no rooted triplets are violated as follows:

> Proceed as in BUILD, but after computing the blocks (i.e., the connected components in $\mathcal{G}(L')$), construct an undirected graph H whose vertices are the blocks and where $\{A, B\}$ is an edge in H if and only if $\mathcal{R}$ contains some rooted triplet of the form $xy|z$ where either $x, y \in A$ and $z \in B$, or $x, y \in B$ and $z \in A$. Compute a minimum coloring of H and merge all blocks whose vertices in H received the same color. Then, continue the execution of BUILD.

The above step can be implemented in $O^*(2^j)$ time by applying an exact algorithm for Chromatic Number [3], where j is the number of vertices in H. Summation over all recursive calls yields a total running time of $O^*(2^n)$. An interesting question is if this method minimizes the number of internal nodes, i.e., whether or not it always gives an optimal solution for MinRS.

Acknowledgments

The authors would like to thank Sylvain Guillemot and Wing-Kin Sung for some helpful comments.

References

1. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. SIAM Journal on Computing 10(3), 405–421 (1981)
2. Bininda-Emonds, O.R.P.: The evolution of supertrees. TRENDS in Ecology and Evolution 19(6), 315–322 (2004)
3. Björklund, A., Husfeldt, T.: Exact graph coloring using inclusion-exclusion. In: Kao, M.-Y. (ed.) Encyclopedia of Algorithms, p. 289. Springer Science+Business Media, LLC, Heidelberg (2008)
4. Bryant, D.: Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis. PhD thesis, University of Canterbury, Christchurch, New Zealand (1997)
5. Byrka, J., Guillemot, S., Jansson, J.: New results on optimizing rooted triplets consistency. Discrete Applied Mathematics 158(11), 1136–1147 (2010)
6. Chor, B., Hendy, M., Penny, D.: Analytic solutions for three taxon ML trees with variable rates across sites. Discrete Applied Mathematics 155(6-7), 750–758 (2007)
7. Garey, M., Johnson, D.: Computers and Intractability – A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
8. Gąsieniec, L., Jansson, J., Lingas, A., Östlin, A.: On the complexity of constructing evolutionary trees. Journal of Combinatorial Optimization 3(2-3), 183–197 (1999)
9. Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica 24(1), 1–13 (1999)
10. Holm, J., de Lichtenberg, K., Thorup, M.: Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. Journal of the ACM 48(4), 723–760 (2001)
11. Jansson, J., Ng, J.H.-K., Sadakane, K., Sung, W.-K.: Rooted maximum agreement supertrees. Algorithmica 43(4), 293–307 (2005)
12. Kearney, P.: Phylogenetics and the quartet method. In: Jiang, T., Xu, Y., Zhang, M.Q. (eds.) Current Topics in Computational Molecular Biology, pp. 111–133. The MIT Press, Massachusetts (2002)
13. Otter, R.: The number of trees. Annals of Mathematics 49(3), 583–599 (1948)

14. Page, R.D.M.: Modified mincut supertrees. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 537–552. Springer, Heidelberg (2002)
15. Semple, C., Steel, M.: A supertree method for rooted trees. Discrete Applied Mathematics 105(1-3), 147–158 (2000)
16. Snir, S., Rao, S.: Using Max Cut to enhance rooted trees consistency. IEEE/ACM Transactions on Computational Biology and Bioinformatics 3(4), 323–333 (2006)
17. Zuckerman, D.: Linear degree extractors and the inapproximability of Max Clique and Chromatic Number. Theory of Computing 3(1), 103–128 (2007)