SIAM J. COMPUT.
Vol. 41, No. 1, pp. 272–291

THE COMPLEXITY OF INFERRING A MINIMALLY RESOLVED PHYLOGENETIC SUPERTREE*

JESPER JANSSON†, RICHARD S. LEMENCE‡, AND ANDRZEJ LINGAS§

Abstract. A recursive algorithm by Aho et al. [*SIAM J. Comput.*, 10 (1981), pp. 405–421] forms the basis for many modern rooted supertree methods employed in Phylogenetics. However, as observed by Bryant [*Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis*, Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997], 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). Furthermore, we show that the decision version of MINRS is NP-hard for any fixed positive integer $q \geq 4$, where q is the number of allowed internal nodes, but linear-time solvable for $q \leq 3$. In contrast, MINRS becomes polynomial-time solvable for any q when restricted to caterpillars. We also present an exponential-time algorithm based on tree separators for solving MINRS exactly. It runs in $2^{O(n \log p)}$ time when every node may have at most p children that are internal nodes and where n is the cardinality of the leaf label set. Finally, we demonstrate that augmenting the algorithm of Aho et al. with an algorithm for optimal graph coloring to help merge certain blocks of leaves during the execution does not improve the output solution much in the worst case.

Key words. phylogenetic tree, rooted triplet, minimally resolved supertree, NP-hardness, graph coloring, tree separator

AMS subject classifications. 68Q17, 05C05, 05C85

DOI. 10.1137/100811489

1. Introduction.

Phylogenetic trees are leaf-labeled trees that 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, 16, 19, 20], 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*.

Many alternative supertree methods whose properties differ have been developed; see, e.g., the survey paper by Bininda-Emonds [2], the introduction of [19], or Chapter 3.3 in [20] for an extensive list of references. A classic algorithm by Aho et al. [1] named BUILD (see section 2.2 for a short description) can be used to merge a given set of *nonconflicting* rooted phylogenetic trees. Due to its simplicity and efficiency,

*Received by the editors October 12, 2010; accepted for publication (in revised form) December 22, 2011; published electronically February 16, 2012. A preliminary version of this article appeared in [13].

http://www.siam.org/journals/sicomp/41-1/81148.html

†Corresponding author. Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan (Jesper.Jansson@ocha.ac.jp). This author's work was funded by the Special Coordination Funds for Promoting Science and Technology (Japan) and KAKENHI grant 23700011.

‡Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan and Institute of Mathematics, College of Science, University of the Philippines, Diliman, Quezon City, 1101 Philippines (rslemence@gmail.com). This author's work was funded by the Special Coordination Funds for Promoting Science and Technology (Japan).

§Department of Computer Science, Lund University, 22100 Lund, Sweden (Andrzej.Lingas@cs.lth.se).

it is used as a starting point in several rooted supertree methods such as the one presented by Gąsieniec et al. in [8] for combining a set of *conflicting* trees.[1] Similar ideas were employed in later methods such as [18, 22, 23], and although they use slightly different rules for how to split a connected component of the auxiliary graph, typically, these methods 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 constructed 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 to show that BUILD can, 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.

One of the common criticisms of supertrees is that they may suggest evolutionary relationships among leaves that are not directly supported by any one of the input trees, which can create false groupings in the form of "spurious novel clades" [2]. See also the discussion in [19]. Therefore, it is reasonable to try to avoid making more such statements about the evolutionary history 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 all of the input, are important in Bioinformatics. Furthermore, a minimally resolved supertree gives a simpler overview of the observed data than a supertree containing many internal nodes, which in itself is a desirable property as it makes it easier for researchers to represent relationships among nodes and to organize the data. However, the computational aspects of inferring minimally resolved supertrees have been overlooked until now.

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. We present several new negative and positive results.

The paper is organized as follows. Section 2 defines the MinRS problem formally, surveys previous work, and exhibits an example for which the BUILD algorithm of Aho et al. [1] produces an extremely poor solution to MinRS. 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. Next, section 4 presents some exact algorithms for MinRS. More precisely, section 4.1 gives a recursive, polynomial-time algorithm for solving MinRS when the

[1] The method, named "Heuristic 2" in [8], imitates the behavior of BUILD until a conflict occurs, at which point the so-called auxiliary graph consists of a single connected component which is subsequently forced to split into smaller components by deleting the edges in a minimum weight edge cut. Then, the method applies itself recursively to each newly created component.

output tree is required to be a *caterpillar*. Then, section 4.2 uses the algorithm from section 4.1 to solve MinRS in linear time when $q = 2$ or $q = 3$, where q is the number of allowed internal nodes in the output tree. Section 4.3 describes a simple algorithm for the decision version of MinRS which runs in $O^*(f(q) \cdot q^n)$ time, where $f(q)$ is the number of rooted, unlabeled trees with q nodes. Section 4.4 presents a more sophisticated exponential-time exact algorithm based on tree separators that runs in $2^{O(n \log p)}$ time, where every node is restricted to have at most p children which are internal nodes. Finally, section 5 disproves a conjecture from the conference version of this paper [13] concerning the optimality of BUILD when augmented with an algorithm for optimal graph coloring and section 6 contains some concluding remarks.

2. Preliminaries.

2.1. Basic definitions. We will use the following definitions and notation.

To simplify the presentation, every node in a tree is considered to be both an ancestor and 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 $lca(x, y) \prec lca(x, z) = lca(y, z)$. See Figure 1 (a).

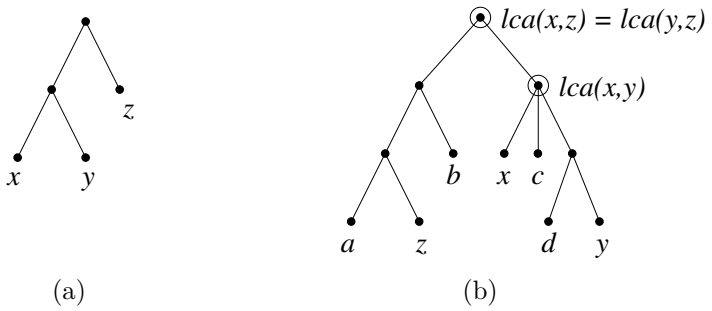

FIG. 1. (a) *The rooted triplet* $xy|z$. (b) *A phylogenetic tree consistent with* $xy|z$.

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 $xy|z$ and T are said to be *consistent* with each other. See Figure 1 (b). 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, construct a phylogenetic tree consistent with $\mathcal{R}$, by a classic algorithm of Aho et al. [1] named BUILD. The algorithm is described in section 2.2.

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 et al. [1]. It was initially invented to solve a problem from the theory of relational databases, and a decade later, it was realized that the algorithm can be applied to Phylogenetics as well [24].

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 *fail*.

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. When recursing on a block B, only those rooted triplets in $\mathcal{R}$ whose three leaves belong to B are considered. 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 *fail*. This approach is motivated by Proposition 2.1 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 2.1 (Aho et al. [1]). *If $\mathcal{G}(L)$ has only one connected component and $|L| > 1$, then $\mathcal{R}$ is not consistent with any phylogenetic tree.*

An example of BUILD's execution can be found in section 2.3 below. Also see the beginning of section 5.2 for another example.

The running time of the original implementation of BUILD [1] was $O(nk)$, where $n = |L|$ and $k = |\mathcal{R}|$. Henzinger, King, and Warnow [9] later presented a faster implementation of BUILD that avoids recomputing the connected components of the auxiliary graphs from scratch on each recursion level by employing special dynamic data structures for keeping track of the connected components in a graph under edge deletions. Replacing the data structures used in [9] by a more recent one [10] further reduces the complexity of the BUILD algorithm to $\min\{O(n+k\log^2 n), O(k+n^2\log n)\}$ time [14].

2.3. A bad example for BUILD. 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 Chapter 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 minimally resolved supertree contains *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 $\{bc|a, ef|a\}$. After that, if we extend the example as follows:

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

where i is any positive integer, we then obtain a consistent set of rooted triplets for which BUILD first partitions the leaves into the blocks $\{x_0\}$, $\{x_1, x_2\}$, $\{x_3, x_4\}$, $\ldots$, $\{x_{2i-1}, x_{2i}\}$ and subsequently outputs a tree having $i + 1$ internal nodes, as shown in Figure 2 (a). However, the tree in Figure 2 (b) consisting of a root node with two children, one being a leaf labeled x_0 and the other being an internal node with $2i$ children that are leaves labeled $x_1, x_2, \ldots, x_{2i}$, contains exactly two internal nodes and is also consistent with $\mathcal{R}_e$.

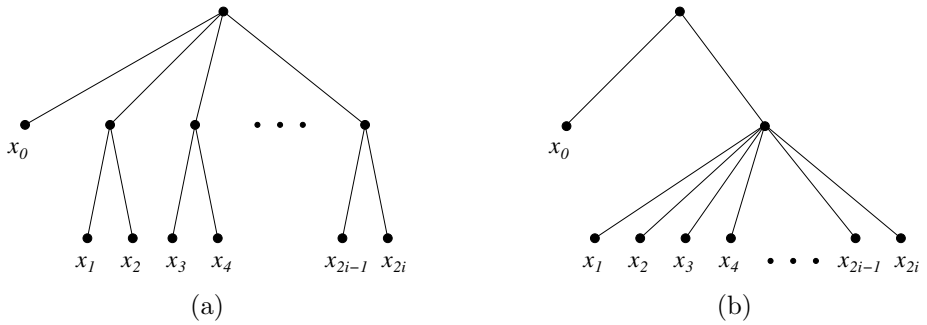

FIG. 2. (a) *The tree constructed by BUILD on input $\mathcal{R}_e$ has $i + 1$ internal nodes.* (b) *A tree consistent with $\mathcal{R}_e$ having 2 internal nodes.*

This proves the following theorem.

THEOREM 2.2. *The BUILD algorithm of Aho et al.* [1] *may produce a tree with $\Omega(n)$ times more internal nodes than a minimally resolved supertree, where n is the cardinality of the leaf label set.*

2.4. Definition of MinRS. From the observation in section 2.3, 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 following problem.

THE MINIMALLY RESOLVED SUPERTREE CONSISTENT WITH ROOTED TRIPLETS PROBLEM (MINRS)

Instance: A set $\mathcal{R}$ of rooted triplets over a leaf label 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, *fail*.

We will also address the *decision version of* MinRS, in which the input consists of a set $\mathcal{R}$ of rooted triplets with a leaf set L together with a positive integer q, and the objective is to determine whether there exists a phylogenetic tree leaf-labeled by L having q internal nodes that is consistent with $\mathcal{R}$.

2.5. 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, King, and Warnow [9] incorrectly assumed that the BUILD algorithm of Aho et al. [1] always constructs a minimally resolved supertree. Thus, the claim on page 7 in [9] which says:

> "If we use the version of the batch deletion algorithm which discovers *all* newly created components, the consensus tree with the minimal number of nodes (the *minimal tree*) is returned."

is false. In particular, according to the proof of Theorem 4 in [9], the tree constructed by Algorithm A' of Henzinger, King, and Warnow [9] is identical to the tree constructed by the BUILD algorithm; therefore, the minimality claim in Theorem 4 in [9] is not correct and our example from section 2.3 implies that Algorithm A' may output a tree with $\Omega(n)$ times more internal nodes than a minimally resolved supertree.

In another related paper, Semple [21] presented an algorithm named ALL-MIN-TREES that enumerates all minor-minimal phylogenetic trees consistent with a given set $\mathcal{R}$ of rooted triplets. (If T is a phylogenetic tree consistent with $\mathcal{R}$ and it is not possible to obtain a tree consistent with $\mathcal{R}$ by contracting an internal edge of T, then T is called *minor-minimal with respect to* $\mathcal{R}$.) For example, according to Proposition 4.1 in [21], the tree output by the BUILD algorithm [1] is always minor-minimal with respect to $\mathcal{R}$, which also proves that a minor-minimal tree is not necessarily a minimally resolved supertree. On the other hand, by definition, any minimally resolved supertree for $\mathcal{R}$ must be minor-minimal, so one way to solve MinRS is by running ALL-MIN-TREES to find all minor-minimal trees and then selecting a tree with the smallest number of internal nodes. The worst-case running time of this particular approach to solving MinRS depends on that of Semple's algorithm, which is $\Omega(2^n)$ according to Example 4.7 in [21]. Actually, the lower bound on the worst-case running time of ALL-MIN-TREES can be strengthened to $\Omega((\frac{n}{2})^{n/2})$, i.e., a self-exponential function in $n/2$. To see this, modify Example 4.7 in [21] to $L = \{a_1, a_2, \ldots, a_x, g_1, g_2, \ldots, g_y\}$, where x and y are two positive integers satisfying $x + y = n$, and let $\mathcal{R}_1 = \{a_i a_{i+1} | a_{i+2} : 1 \le i \le x-2\}$, $\mathcal{R}_2 = \{a_1 a_2 | g_i : 1 \le i \le y\}$, and $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Then, by an argument analogous to the one in Example 4.7 of [21], there are at least $(x-2)^y = (x-2)^{n-x}$ minor-minimal trees consistent with $\mathcal{R}$, and taking $x = \frac{n}{3} + 2$ gives

$$(x-2)^{n-x} = \left(\frac{n}{3}\right)^{(2n/3)-2} = \frac{(n/3)^{2n/3} \cdot (3/2)^{2n/3}}{(n/3)^2 \cdot (3/2)^{2n/3}}$$
$$= \left(\frac{n}{2}\right)^{n/2} \cdot \frac{(n/2)^{n/6}}{(n/3)^2 \cdot (3/2)^{2n/3}} > \left(\frac{n}{2}\right)^{n/2} \cdot 1 = \left(\frac{n}{2}\right)^{n/2}$$

for large enough n.

Corollary 2.3. *The algorithm ALL-MIN-TREES in* [21] *solves* MinRS*, but its worst-case running time is* $\Omega((\frac{n}{2})^{n/2})$*, where* n *is the number of leaf labels in* $\mathcal{R}$*.*

A kind of "dual" problem to MinRS called Most Resolved Compatible Tree was studied by Bryant in Chapter 2.6.3 in [4]. It is defined next. A *fan triplet* is 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 the Most Resolved Compatible Tree problem, 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], and this result holds even if $\mathcal{R}$ consists entirely of fan triplets.

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 follows:

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

Zuckerman [25] 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 specific 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.
- 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.

The given graph G and the constructed set $\mathcal{R}$ are related as stated in the next two lemmas.

Lemma 3.1. *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 nonempty 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 Figure 3 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 $a_1, a_2, \ldots, a_j$. Then, 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

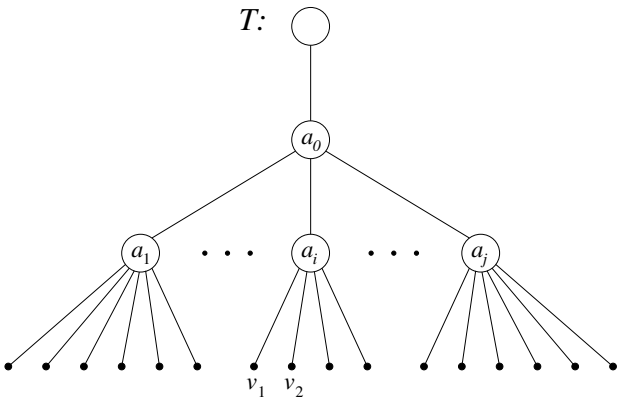

FIG. 3. *Illustrating the proof of Lemma* 3.1. *In this example, there is one empty color class (i.e.,* $K - j = 1$*), so the path from the root of* T *to the internal node* a_0 *has length* 1. *For each vertex* $v \in V$*, where* v *belongs to the color class* V_i*, the leaves* v_1*,* v_2 *in* T *are directly attached to the internal node* a_i*.*

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. □

LEMMA 3.2. *If there exists a tree which is consistent with* $\mathcal{R}$ *and contains* $K+1$ *internal nodes, then* G *is* K*-colorable.*

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 $lca(v_1, w_1)$, $lca(v_1, w_2)$, $lca(v_2, w_1)$, or $lca(v_2, w_2)$ 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. □

THEOREM 3.3. MINRS *cannot be approximated within* $n^{1-\epsilon}$ *for any constant* $0 < \epsilon < 1$ *in polynomial time, unless* $P = NP$.

Proof. The proof follows from Lemmas 3.1 and 3.2 together with the fact that

CHROMATIC NUMBER is NP-hard to approximate within $|V|^{1-\epsilon}$ for every $0 < \epsilon < 1$ [25]. □

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 3.1 and 3.2 also yields the following corollary.

COROLLARY 3.4. *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. This section presents several algorithms for solving MINRS exactly. First, section 4.1 gives an algorithm for MINRS restricted to caterpillars. Then, section 4.2 shows how to solve MINRS in linear time when $q = 2$ or $q = 3$. Section 4.3 describes a brute-force algorithm, and section 4.4 shows that MINRS under the restriction that each node has at most p nonleaf children, where $p \geq 2$, is solvable in $2^{O(n \log p)}$ time.

Let $\mathcal{R}$ be a given set of rooted triplets with leaf set L. For any $x \in L$, x is said to *have a lower occurrence in* $\mathcal{R}$ if there exist $b, c \in L$ such that $xb|c \in \mathcal{R}$. If there exist $a, b \in L$ such that $ab|x \in \mathcal{R}$, then x is said to *have an upper occurrence in* $\mathcal{R}$.

4.1. MINRS restricted to caterpillars. A *caterpillar* is a phylogenetic tree in which every node has at most one child that is an internal node. (For an example of a caterpillar, see, e.g., Figure 6 (a).) This subsection gives an algorithm for determining if $\mathcal{R}$ is consistent with a caterpillar, and if so, constructing such a tree having the smallest possible number of internal nodes. To describe the algorithm, we introduce some additional notation.

Define X as the subset of leaves in L that have upper occurrences but no lower occurrences in $\mathcal{R}$. Let $\mathcal{R}'$ be the subset of rooted triplets in $\mathcal{R}$ that do not contain any leaves from X, and let L' be the set of leaves appearing in $\mathcal{R}'$. Finally, define $Y = L \setminus (X \cup L')$ (note that Y may be empty). Intuitively, after attaching all leaves in X directly to the root node of the output tree, we may safely remove all rooted triplets that contain leaves from X, obtaining a smaller instance $\mathcal{R}'$ of MINRS which can be solved recursively. The leaves in Y only appear in rooted triplets that will be removed so we have to be careful to attach them to an internal node below the root in the output tree. See Figure 4 for an illustration.

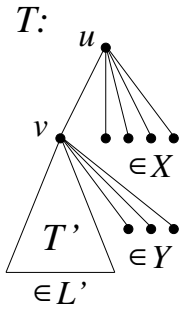

FIG. 4. *The strategy of Algorithm* `MinRS_caterpillar` *is to attach the leaves of X and Y to the root and a child of the root, respectively, and recursively compute a caterpillar T' for L'.*

The next elementary lemma characterizes three key properties of caterpillar consistency. The lemma leads to a natural top-down algorithm for recursively building a caterpillar, named `MinRS_caterpillar` and listed in Figure 5.

Algorithm `MinRS_caterpillar`($\mathcal{R}$)

Input: A nonempty set $\mathcal{R}$ of rooted triplets.
Output: A caterpillar consistent with $\mathcal{R}$ having the smallest possible number of internal nodes, if such a tree exists; otherwise, *fail*.

1. Compute the sets X, $\mathcal{R}'$, L', and Y.
2. **if** $X = \emptyset$ **then**
   Let $T :=$ *fail*.
3. **else do begin**
3.1 Create a root node u of T and create an internal node v that is a child of u.
   Attach each leaf $x \in X$ as a child of u and each $y \in Y$ as a child of v.
3.2 **if** $\mathcal{R}' \neq \emptyset$ **then**
   Let $T' :=$ `MinRS_caterpillar`($\mathcal{R}'$).
   If $T' =$ *fail*, then let $T :=$ *fail*; otherwise, merge the root of T' and v into one node of T (e.g., delete the root of T' while letting all its children become children of v instead).
4. **return** T.

FIG. 5. *The pseudocode for Algorithm* `MinRS_caterpillar`.

LEMMA 4.1. *Suppose that $\mathcal{R} \neq \emptyset$. The following holds:*

1. *If $X = \emptyset$, then $\mathcal{R}$ is not consistent with any caterpillar.*
2. *If $X \neq \emptyset$ and $\mathcal{R}' = \emptyset$, then $\mathcal{R}$ is consistent with a caterpillar containing two internal nodes.*
3. *If $X \neq \emptyset$ and $\mathcal{R}' \neq \emptyset$, then $\mathcal{R}$ is consistent with a caterpillar if and only if $\mathcal{R}'$ is consistent with a caterpillar.*

Proof.

1. Suppose there exists a caterpillar T consistent with $\mathcal{R}$. We shall prove that $X \neq \emptyset$. Consider any leaf $w \in L$ at minimum distance from the root of T. Then $\mathcal{R}$ cannot contain any rooted triplets of the form $wa|b$ with $a, b \in L$, i.e., w has no lower occurrences in $\mathcal{R}$. By definition, w belongs to X.
2. In this case, Y cannot be empty. Construct a caterpillar with two internal nodes u, v, where u is the root and v a child of u, and attach each $x \in X$ to u and each $y \in Y$ to v. Obviously, the resulting tree is consistent with $\mathcal{R}$.
3. First, suppose that $\mathcal{R}$ is consistent with a caterpillar T^*. Since $\mathcal{R}' \subseteq \mathcal{R}$, all rooted triplets in $\mathcal{R}'$ are also consistent with T^*. Taking the subtree of T^* induced by leaves appearing in $\mathcal{R}'$ and then contracting each edge incident to an internal node with outdegree 1 yields a caterpillar consistent with $\mathcal{R}'$.
   Next, suppose that $\mathcal{R}'$ is consistent with a caterpillar T'. Attach every $y \in Y$ as a child of the root of T', and denote the resulting caterpillar by T''. Let T be a caterpillar such that the children of the root node of T are the following: (1) the root of T''; and (2) every element in X. Then T is consistent with all rooted triplets in $\mathcal{R}'$ as well as with every rooted triplet from $\mathcal{R}$ that involves a leaf from X, i.e., T is consistent with $\mathcal{R}$. □

The optimality and time complexity of Algorithm `MinRS_caterpillar` are addressed by the next theorem.

THEOREM 4.2. *Algorithm* `MinRS_caterpillar` *solves the* MINRS *problem restricted to caterpillars. It can be implemented to run in $O((n + k) \cdot q)$ time when a*

solution having q internal nodes exists, and in $O((n+k)\cdot n)$ time when no solution exists.

Proof. By Lemma 4.1, the algorithm correctly returns a caterpillar consistent with $\mathcal{R}$ whenever such a tree exists and *fail* otherwise. It remains to show the minimality of the returned solutions for the non*fail* case.

Our proof is by induction over the number of internal nodes in an optimal solution. Let $H(i)$ be the statement that Algorithm `MinRS_caterpillar` always returns a minimal solution when a minimal solution contains exactly i internal nodes. The base case $H(2)$ (i.e., two internal nodes) holds according to Lemma 4.1.2. Next, assume that $H(i)$ is true for some positive integer $i \geq 2$. Let $\mathcal{R}^*$ be any set of rooted triplets for which an optimal caterpillar T^* contains $i+1$ internal nodes. We shall show that the caterpillar output by Algorithm `MinRS_caterpillar` is as good as T^*. Consider the set X^* of leaves that have upper occurrences but no lower occurrences in $\mathcal{R}^*$. Every leaf x that is a child of the root u^* of T^* must belong to X^* (otherwise, if $x \notin X^*$, x would have some lower occurrence in $\mathcal{R}^*$ and then x could not be a child of u^*). On the other hand, not all leaves belonging to X^* must be children of u^*. Let $T^{\#}$ be the caterpillar obtained by removing all leaves in X^* from T^* and reattaching them as children of u^*. Note that $T^{\#}$ is still a caterpillar consistent with $\mathcal{R}^*$, and that $T^{\#}$ still has $i+1$ internal nodes. Furthermore, the subtree rooted at the internal node $v^{\#}$ that is a child of the root of $T^{\#}$ has i internal nodes and is consistent with $(\mathcal{R}^*)'$, where $(\mathcal{R}^*)'$ is defined as the set of all rooted triplets from $\mathcal{R}^*$ that do not involve leaves from X^*; this subtree must be optimal because if there were such a tree with fewer internal nodes, then it would directly yield a solution for $\mathcal{R}^*$ having fewer than $i+1$ internal nodes, contradicting the minimality of T^*. Let T' be the output of Algorithm `MinRS_caterpillar` on input $(\mathcal{R}^*)'$. By assumption $H(i)$, T' has the fewest possible number of internal nodes, i.e., i. If we replace the subtree rooted at $v^{\#}$ in $T^{\#}$ by T', we obtain another caterpillar that is consistent with $\mathcal{R}^*$ with $i+1$ internal nodes, and the result is precisely the output of Algorithm `MinRS_caterpillar` on input $\mathcal{R}^*$. Therefore, $H(i+1)$ holds.

To implement the algorithm so that any level of the recursion uses $O(n+k)$ time, use two bit vectors `Lower` and `Upper`, each of size n, to indicate lower and upper occurrences of the leaves in L. Initially, set all bits in `Lower` and `Upper` to 0. Next, for each $ab|c \in \mathcal{R}$, set `Lower`$[a] = 1$, `Lower`$[b] = 1$, and `Upper`$[c] = 1$. Then, it is straightforward to identify the sets X, $\mathcal{R}'$, L', and Y in linear time. ∎

For any given $q \geq 2$, if at most q internal nodes are permitted, the algorithm can be modified to run in $O((n+k)\cdot q)$ time by terminating its execution and returning *fail* if it ever reaches q recursion levels.

4.2. MinRS with $q = 2$ or $q = 3$. Corollary 3.4 states that MinRS (in its unrestricted form) is NP-hard for every fixed $q \geq 4$, where q is the allowed number of internal nodes. We now turn our attention to MinRS with $q \in \{2, 3\}$. We have seen that when restricted to caterpillars, the problem becomes solvable in polynomial time for any q (Theorem 4.2). Noting that a tree with exactly two internal nodes is always a caterpillar, Theorem 4.2 immediately gives us the next corollary.

Corollary 4.3. MinRS *restricted to $q = 2$ can be solved in $O(n+k)$ time.*

For $q = 3$, there are two possible tree topologies, as shown in Figure 6. (Observe that one of them is also a caterpillar.) To solve MinRS for $q = 3$, it therefore suffices to check if there exists a tree of one of the two types that is consistent with $\mathcal{R}$; if so, we output it and otherwise we output *fail*. In this subsection, we refer to the structures in Figure 6 (a) and (b) as *type* $3A$ and *type* $3B$, respectively.

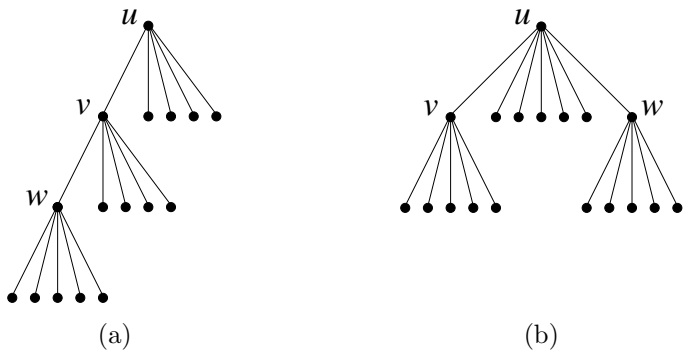

FIG. 6. *The two different types of trees having three internal nodes:* (a) *type* 3*A tree;* (b) *type* 3*B tree. Note that a type* 3*A tree is a caterpillar.*

Algorithm `MinRS_type_3B_tree`

1. Compute the sets X and $\mathcal{R}'$.
2. Construct the auxiliary graph $\mathcal{G}(L)$ as in the BUILD algorithm [1] described in section 2.2. Let $\mathcal{C}$ be the set of all *nonsingleton* connected components of $\mathcal{G}(L)$. (Note that $\{x\}$ is a singleton connected component in $\mathcal{G}(L)$ if and only if $x \in X$.)
3. Construct an undirected graph H with vertex set $\mathcal{C}$ and where $\{I, J\}$ is an edge in H if and only if $\mathcal{R}'$ contains some rooted triplet of the form $xy|z$ where either $x, y \in I$ and $z \in J$, or $x, y \in J$ and $z \in I$.[2]
4. Try to compute a 2-coloring of H.
5. If no 2-coloring exists, then let T = *fail*. Otherwise, let V_1 and V_2 be the two color classes of H and define $M_i = \{z \in L \setminus X : z$ belongs to some connected component in color class $V_i\}$ for $i \in \{1, 2\}$. Create a root node u of T with two children v, w which are internal nodes, then attach the leaves in L to T by letting each $x \in X$ be a child of u, each $z \in M_1$ be a child of v, and each $z \in M_2$ a child of w.
6. Return T.

FIG. 7. *The pseudocode for Algorithm* `MinRS_type_3B_tree`.

4.2.1. Constructing a tree of type 3A. Type 3A trees are caterpillars with exactly 3 internal nodes; therefore, this case is also covered by Theorem 4.2.

LEMMA 4.4. *It is possible to check if there exists a type* 3*A tree consistent with* $\mathcal{R}$ *(and if so, construct it) in* $O(n + k)$ *time.*

4.2.2. Constructing a tree of type 3B. In the same way as in section 4.1, define X to be the subset of leaves in L that have upper occurrences but no lower occurrences in $\mathcal{R}$. Let $\mathcal{R}'$ be the subset of rooted triplets in $\mathcal{R}$ that do not contain any leaves from X. Build a tree T of type 3B as in Figure 6 (b) according to Algorithm `MinRS_type_3B_tree` listed in Figure 7. Here, the basic idea is to attach all leaves in X directly to the root of the output tree and to determine where all other leaves in $L \setminus X$ should be attached by using a 2-coloring of a graph whose edges rep-

resent pairs of leaves that are required to have different parents in the output tree. The next lemma shows the correctness of this approach.

LEMMA 4.5. *If $\mathcal{R}$ is consistent with a tree of type 3B, then the tree T output by Algorithm* `MinRS_type_3B_tree` *is also a type 3B tree consistent with $\mathcal{R}$.*

Proof. We first show that the graph H is 2-colorable. By the lemma statement, there exists at least one tree T^* of type 3B consistent with $\mathcal{R}$ (and thus consistent with $\mathcal{R}'$). Consider any $ab|c \in \mathcal{R}'$. Leaves a and b cannot be children of the root of T^*. In addition, we have $c \notin X$, and by the definition of X, c has at least one lower occurrence in $\mathcal{R}$. It follows that c cannot be a child of the root of T^* either. In the same way, for any $ab|x \in \mathcal{R}$ with $x \in X$, it holds that a and b cannot be children of the root of T^*. Therefore, $L \setminus X$ can be partitioned into two disjoint subsets M_{v^*} and M_{w^*}, each consisting of the children of one nonroot, internal node of T^*. Now, the partition $\{M_{v^*}, M_{w^*}\}$ induces a valid 2-coloring of H because for each edge $\{I, J\}$ in H, there exists some $ab|c \in \mathcal{R}'$ with either $a, b \in I$, $c \in J$ or $a, b \in J$, $c \in I$; moreover, T^* is consistent with $ab|c$, which means that either $a, b \in M_{v^*}$ and $c \in M_{w^*}$ or $a, b \in M_{w^*}$ and $c \in M_{v^*}$, so I and J will never receive the same color.

Since H is 2-colorable, the above procedure will find a valid 2-coloring and partition $L \setminus X$ into M_1 and M_2 accordingly. Finally, we show that each $ab|c \in \mathcal{R}$ is consistent with T. Leaves a and b belong to the same connected component of $\mathcal{G}(L)$ and are therefore always placed together in one of M_1 and M_2. There are two cases:

- $c \in X$: Then c is a child of u in T while a and b are children of one of v and w in T. Hence, $ab|c$ is consistent with T.
- $c \in L \setminus X$: Then H contains an edge $\{I, J\}$ with $a, b \in I$ and $c \in J$, or $a, b \in J$ and $c \in I$. Because of the 2-coloring, components I and J belong to different color classes so either $a, b \in M_1$ and $c \in M_2$, i.e., a and b are children of v in T and c is a child of w, or vice versa. Hence, $ab|c$ is consistent with T. ∎

The set X can be computed in $O(n + k)$ time by using two bit vectors of size n. After that, it is easy to obtain $\mathcal{R}'$ in $O(n + k)$ time. By the definition of $\mathcal{G}(L')$, the number of edges in $\mathcal{G}(L')$ is at most k and computing the connected components of $\mathcal{G}(L')$, constructing the graph H, and 2-coloring H can all be done in linear time using standard breadth-first search techniques. Thus, we have the following lemma.

LEMMA 4.6. *It is possible to check if there exists a type 3B tree consistent with $\mathcal{R}$ (and if so, construct it) in $O(n + k)$ time.*

Now, we can combine Lemmas 4.4 and 4.6.

THEOREM 4.7. MinRS *restricted to $q = 3$ can be solved in $O(n + k)$ time.*

4.3. A brute-force algorithm. The decision version of MinRS can be solved with a naive 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 the following theorem.

THEOREM 4.8. *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.*

[2]By the construction, H might contain loops of the form $\{I, I\}$. In this case, H will never be 2-colorable.

It is known that $f(q) \sim c \cdot d^q \cdot q^{-3/2}$, where $c = 0.439924\ldots$ and $d = 2.955765\ldots$ [17]. Thus, the algorithm runs in exponential time for $q = O(1)$.

4.4. An exponential-time algorithm for a restricted case of MinRS. A *leaf child* of a node v in a tree is a child of v which is a leaf, and a *nonleaf child* of v is a child of v which is an internal node. For any rooted tree T and node v in T, the notation T_v means the subtree of T rooted at v. Every leaf that belongs to T_v is called a *leaf descendant* of v. This subsection develops an exact algorithm for MinRS whose running time depends exponentially on n and a parameter p which specifies an upper bound on the number of nonleaf children that every node may have. The derivation of our main result relies on the following variant of the tree separator theorem.

Lemma 4.9. *Let T be a rooted tree with n leaves. There exists a node v such that the subtree T_v contains strictly greater than $\frac{n}{2}$ leaves but for each child w of v, the subtree T_w has at most $\frac{n}{2}$ leaves.*

Proof. Start from the root of T and perform the following procedure: If the root satisfies the condition, then set v to it and stop; otherwise, set v to the child of the root with the largest number of leaf descendants and iterate the procedure for T_v.

Note that whenever a new iteration is applied to T_v, T_v must 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 4.9 to design a $2^{O(n \log p)}$-time procedure for the variant of MinRS where every internal node is allowed to have at most p nonleaf children.

The procedure is recursive. We enumerate all partitions of the leaf set L for which there exists a tree with a node v satisfying the condition in Lemma 4.9. Then, we apply the procedure recursively on the resulting leaf subsets, possibly augmented by a dummy leaf, modifying the rooted triplets accordingly.

A partition Q induced by a node v that satisfies the condition in Lemma 4.9 has two levels. See Figure 8. First, Q splits the set L of leaves into a set L' (corresponding to the leaves in T_v) of size strictly greater than $\frac{n}{2}$ and its complement $L \setminus L'$ (corresponding to the leaves in $T \setminus T_v$) of size strictly less than $\frac{n}{2}$. Second, Q further splits L' into $p' \le p$ sets $L'_1, \ldots, L'_{p'}$ (corresponding to the p' sets of leaf descendants of the p' nonleaf children of v), each of size at most $\frac{n}{2}$, plus a number of singletons (corresponding to the leaf children of v).

Let Q be a candidate partition of L as defined above. If there is a rooted triplet of the form $xy|z$ 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 automatically satisfied by Q and the rooted triplet can be disregarded. As for the subsets $L'_1, \ldots, L'_{p'}$ and the remaining singleton leaves, for each rooted triplet of the form $xy|z$ where $x, y, z \in L'$, if x, y are not in the same subset L'_l, then we can also disregard Q.

Next, augment $L \setminus L'$ by a dummy leaf a that represents L', i.e., all leaf descendants of the node v. For each rooted triplet of the form $xy|z$ where $x, z \in L \setminus L'$ and $y \in L'$, create the rooted triplet $xa|z$, and for each rooted triplet of the form $xy|z$ where $x, y \in L \setminus L'$ and $z \in L'$, create the rooted triplet $xy|a$. Run the 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 newly created rooted triplets involving the dummy leaf a. Let T'' be the tree returned by the procedure.

Also run the procedure recursively on each of the subsets $L'_1, \ldots, L'_{p'}$, obtaining trees $T'_1, \ldots, T'_{p'}$. Then, let $T'_1, \ldots, T'_{p'}$ as well as the singleton leaves become children of the leaf a in the tree T'', and put the resulting tree on a list of potential solutions.

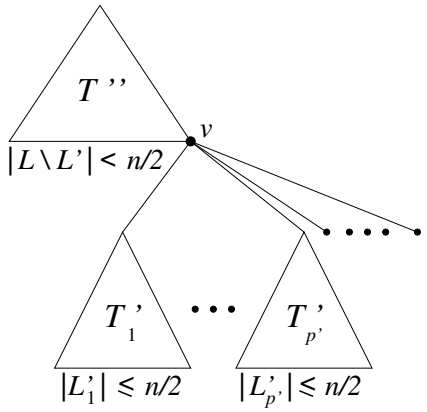

FIG. 8. *According to Lemma* 4.9, *T contains a node v whose removal divides the leaf set L into a subset L' with $|L'| > \frac{n}{2}$ and its complement $L \setminus L'$ with $|L \setminus L'| < \frac{n}{2}$. In addition, removing v partitions L' into subsets $L'_1, \ldots, L'_{p'}$ with at most $\frac{n}{2}$ leaves each as well as a number of singletons.*

Finally, after all valid candidate partitions of L have been considered, return the tree on the list (if any) with the minimum number of internal nodes.

The correctness of our procedure follows from Lemma 4.9 and the fact that we can join T'' with $T'_1, \ldots, T'_{p'}$ at the dummy leaf a in the described way.

Let us estimate the time complexity $T(n)$ of our procedure. The number of partitions considered and the time needed to generate them are trivially $O((p+2)^n) = 2^{O(n \log p)}$ since each of the n elements of L can belong to either $L \setminus L'$, one of the at most p subsets of the form L'_l, or the set of all singleton leaves in L'. Furthermore, each of the at most $p+1$ 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 $2^{O(n \log p)} \cdot ((p+1) \cdot T(\frac{n}{2}) + n^{O(1)})$. By the inequality $2^{\alpha n \log p} \cdot (2^{c(n/2) \log p} + n^{O(1)}) \leq 2^{cn \log p}$ for $c > 2\alpha$, $p \geq 2$, and sufficiently large n, we obtain $T(n) = 2^{O(n \log p)}$.

THEOREM 4.10. *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 p nonleaf children, where $p \geq 2$, is solvable in $2^{O(n \log p)}$ 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 p nonleaf children by increasing the number of internal nodes by an $O(\log_p n)$ multiplicative factor. (Simply connect each internal node v to its nonleaf children C_v via a p-ary tree of depth $O(\log_p n)$ having C_v as leaves.) Hence, we obtain the following corollary.

COROLLARY 4.11. MINRS *can be approximated within a ratio of $O(\log_p n)$ in $2^{O(n \log p)}$ time.*

5. A counterexample to a conjecture. Here, we further explore the relationship between MINRS and CHROMATIC NUMBER and give a counterexample to a conjecture from the conference version of this paper [13].

5.1. The idea. 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

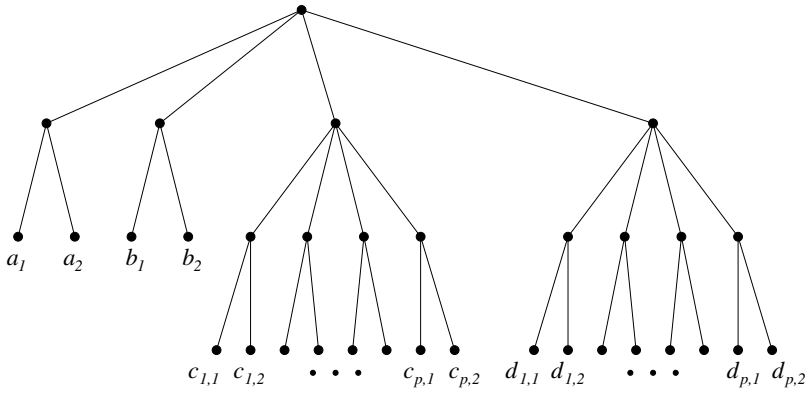

FIG. 9. *The tree constructed by BUILD on input $\mathcal{R}_x$ has $2p+5$ internal nodes.*

is to merge blocks while ensuring that no rooted triplets are violated as follows:[3]

> 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 open question from section 5 in [13] was if the BUILD algorithm, modified as above, minimizes the number of internal nodes in the output tree, i.e., whether or not it always gives an optimal solution for MINRS. We now show that this is not the case.

5.2. The counterexample. Let p be any fixed positive integer with $p \geq 2$. Define the following four leaf sets:

$$\begin{aligned}
L_A &= \{a_1, a_2\},\\
L_B &= \{b_1, b_2\},\\
L_C &= \{c_{1,1}, c_{1,2}, c_{2,1}, c_{2,2}, \ldots, c_{p,1}, c_{p,2}\},\\
L_D &= \{d_{1,1}, d_{1,2}, d_{2,1}, d_{2,2}, \ldots, d_{p,1}, d_{p,2}\},
\end{aligned}$$

and let $L = L_A \cup L_B \cup L_C \cup L_D$. Observe that $|L| = 4p+4$. Next, define the following sets of rooted triplets with leaf set L:

$$\begin{aligned}
\mathcal{R}_1 &= \{a_1a_2|b_1,\ b_1b_2|a_1\},\\
\mathcal{R}_2 &= \{c_{i,1}c_{i,2}|c_{j,1} : 1 \leq i, j \leq p,\ i \neq j\},\\
\mathcal{R}_3 &= \{c_{i,1}c_{(i+1),1}|a_1 : 1 \leq i \leq p-1\},\\
\mathcal{R}_4 &= \{d_{i,1}d_{i,2}|d_{j,1} : 1 \leq i, j \leq p,\ i \neq j\},\\
\mathcal{R}_5 &= \{d_{i,1}d_{(i+1),1}|b_1 : 1 \leq i \leq p-1\},
\end{aligned}$$

and take $\mathcal{R}_x = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \cup \mathcal{R}_5$.

First, suppose we run the original version of the BUILD algorithm on input $\mathcal{R}_x$. On the first level of recursion, the auxiliary graph $\mathcal{G}(L)$ is constructed using all rooted

[3]This is an extension of the technique used to construct type 3B trees in section 4.2.

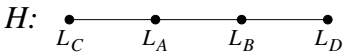

FIG. 10. *The graph H whose vertices represent the connected components in $\mathcal{G}(L)$ and whose edges indicate blocks that cannot be immediately merged.*

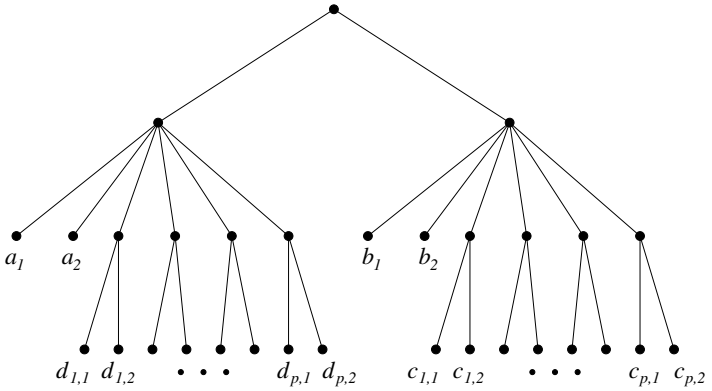

FIG. 11. *The tree output by the modified version of BUILD on input $\mathcal{R}_x$ has $2p + 3$ internal nodes.*

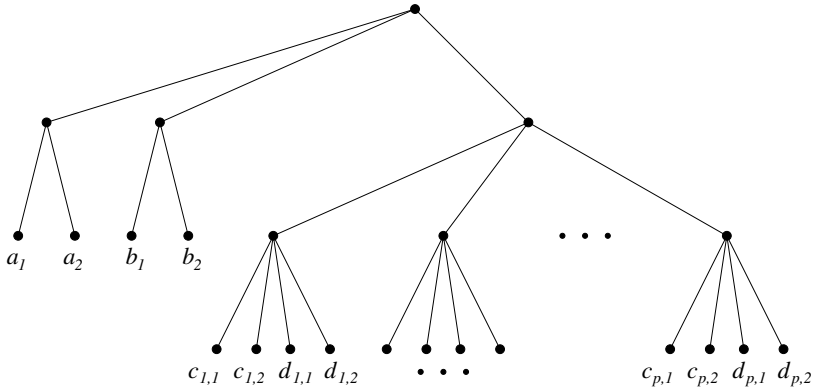

FIG. 12. *A tree consistent with $\mathcal{R}_x$ having only $p + 4$ internal nodes.*

triplets in $\mathcal{R}_x$ and it is easy to see that $\mathcal{G}(L)$ contains four connected components, with vertex sets L_A, L_B, L_C, and L_D, respectively. On the second level of recursion, $\mathcal{G}(L_A)$ is a graph with two vertices and no edges, and similarly for $\mathcal{G}(L_B)$; however, because of $\mathcal{R}_2$ and $\mathcal{R}_4$, each of $\mathcal{G}(L_C)$ and $\mathcal{G}(L_D)$ contains p connected components consisting of two vertices each. The resulting tree has exactly $2p + 5$ internal nodes and is displayed in Figure 9.

Next, consider the modified version of BUILD described in section 5.1. After identifying the blocks L_A, L_B, L_C, and L_D on the first recursion level, it constructs the undirected graph H shown in Figure 10 with vertex set $\{L_A, L_B, L_C, L_D\}$ and the three edges $\{L_C, L_A\}$, $\{L_A, L_B\}$, and $\{L_B, L_D\}$ due to $\mathcal{R}_3$, $\mathcal{R}_1$, and $\mathcal{R}_5$. H is a path graph, and any minimum coloring of H partitions the vertex set into two color classes: $\{L_A, L_D\}$ and $\{L_B, L_C\}$. Hence, the algorithm merges L_A with L_D, and L_B with L_C. On the next level of recursion, the auxiliary graph for $L_A \cup L_D$ contains

two trivial connected components for a_1 and a_2, along with p connected components induced by $\mathcal{R}_4$ which contribute p internal nodes to the final tree. (The case $L_B \cup L_C$ is symmetric.) In summary, the tree output by the modified BUILD algorithm, shown in Figure 11, contains $2p + 3$ internal nodes. We note that the modified version of BUILD is not much better than the original BUILD algorithm for input $\mathcal{R}_x$.

The main result of this section is as follows.

THEOREM 5.1. *The modified version of the BUILD algorithm described in section* 5.1 *does not return an optimal solution for* MINRS *on input* $\mathcal{R}_x$.

Proof. We show that there exists a better solution to MINRS on input $\mathcal{R}_x$ which will not be found by taking a minimum coloring of H on the first recursion level. Proceed as in the modified version of BUILD, but use *three* colors to color the graph H by partitioning its vertex set into three color classes $\{L_A\}$, $\{L_B\}$, and $\{L_C, L_D\}$, i.e., merge L_C and L_D before recursing. Then, for the second recursion level, there exists a tree with $p+1$ internal nodes which is a solution to the subproblem defined by $\mathcal{R}_2 \cup \mathcal{R}_4$ (a root node with p children, each of which is the parent of four leaves). This yields the tree in Figure 12, which is consistent with $\mathcal{R}_x$ and has $p + 4$ internal nodes. ∎

6. Concluding remarks. In this paper, we have introduced the MINRS problem and studied its computational complexity. Significantly, MINRS cannot be approximated within $n^{1-\epsilon}$ for any constant $0 < \epsilon < 1$ in polynomial time, unless P = NP (Theorem 3.3 in section 3). The following table summarizes the computational complexity of the decision version of MINRS for any fixed positive integer q, where q is the allowed number of internal nodes.

q	Computational complexity of MINRS	Reference
2	Solvable in linear time	Corollary 4.3 (section 4.2)
3	Solvable in linear time	Theorem 4.7 (section 4.2)
≥ 4	NP-hard	Corollary 3.4 (section 3)

In addition, caterpillars form a special class of phylogenetic trees for which MINRS is solvable in polynomial time for any q (Theorem 4.2 in section 4.1). The algorithm in section 4.1 for constructing caterpillars was employed as a subroutine to solve MINRS with $q = 3$ in polynomial time in section 4.2, and it may be useful again for designing heuristics for MINRS in the future.

Several important questions remain to be answered.

- The major open problem is: Can MINRS be solved exactly in $O^*(2^n)$ time? Section 3 showed that there is a simple reduction from CHROMATIC NUMBER to MINRS. Conversely, we do not know if MINRS admits a direct reduction to CHROMATIC NUMBER that would allow the $O^*(2^n)$-time algorithm for CHROMATIC NUMBER [3] to be used efficiently. The modification to the BUILD algorithm based on minimum graph coloring described in section 5.1 results in an $O^*(2^n)$-time algorithm, but unfortunately it does not always give an optimal solution. Is there some better way to apply exact algorithms for CHROMATIC NUMBER to solve MINRS?
- We have generalized Bryant's counterexample (Chapter 2.5.2 in [4]) to show that the BUILD algorithm may be suboptimal by a linear factor in the input size when applied to MINRS (Theorem 2.2 in section 2.3). In practice, how often does BUILD not construct a minimally resolved tree? Are these cases common or rare? MINRS is NP-hard even to approximate efficiently (Theorem 3.3 in section 3), so is it possible to avoid the MINRS problem

altogether in real applications by requiring the input set $\mathcal{R}$ to contain a prespecified number of rooted triplets? For example, certain NP-hard problems related to rooted triplet consistency for phylogenetic networks become solvable in polynomial time when restricted to so-called *dense* input sets [12, 15], meaning that $\mathcal{R}$ contains at least one rooted triplet for each cardinality-three subset of L. In fact, since any tree consistent with a dense set of rooted triplets must be binary [15], MinRS restricted to dense inputs is trivially solvable by BUILD. On the other hand, due to errors in the data, increasing the number of rooted triplets in the input too much may introduce an impractically high probability of obtaining one or more "problematic" rooted triplets that lead to inconsistent sets while being difficult to identify. Some experimental validation for how many rooted triplets are needed to achieve a good balance between being able to solve/approximate MinRS efficiently while keeping the probability of obtaining inconsistent sets of rooted triplets sufficiently low would be useful here.

- How far from optimal is the solution output by the modified version of BUILD described in section 5.1 in the worst case? As a straightforward lower bound, we have seen that the modified BUILD constructs a tree having $2p+3$ internal nodes when given the set $\mathcal{R}_x$ defined in section 5.2, while there exists a tree consistent with $\mathcal{R}_x$ having $p+4$ internal nodes according to the proof of Theorem 5.1. Asymptotically, the ratio $(2p+3)/(p+4) \to 2$ as $p \to \infty$. Can $\mathcal{R}_x$ be refined to yield instances on which the modified BUILD performs even worse, or is this bound tight?
- Lastly, p. 302 of [11] states that it is unknown whether the output of BUILD is consistent with the minimum number of rooted triplets, i.e., consistent with a smallest possible superset of rooted triplets $\mathcal{R}'$ such that $\mathcal{R} \subseteq \mathcal{R}'$. However, Bryant's example $\mathcal{R} = \{bc|a, bd|a, ef|a, eg|a\}$ from Chapter 2.5.2 in [4] already established that this is not always the case. We wonder: What can be said about the computational complexity and polynomial-time approximability of the problem of inferring a tree that is consistent with all of the rooted triplets in an input set $\mathcal{R}$ and as few additional rooted triplets as possible?

Acknowledgments. The authors would like to thank Sylvain Guillemot, Charles Semple, Wing-Kin Sung, and Kei Yura for their helpful comments.

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 (1981), pp. 405–421.

[2] O. R. P. Bininda-Emonds, *The evolution of supertrees*, TRENDS in Ecology and Evolution, 19 (2004), pp. 315–322.

[3] A. Björklund and T. Husfeldt, *Exact graph coloring using inclusion-exclusion*, in Encyclopedia of Algorithms, M.-Y. Kao, ed., Springer Science+Business Media, New York, 2008, p. 289.

[4] D. Bryant, *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis*, Ph.D. thesis, University of Canterbury, Christchurch, New Zealand, 1997.

[5] J. Byrka, S. Guillemot, and J. Jansson, *New results on optimizing rooted triplets consistency*, Discrete Appl. Math., 158 (2010), pp. 1136–1147.

[6] B. Chor, M. Hendy, and D. Penny, *Analytic solutions for three taxon ML trees and variable rates across sites*, Discrete Appl. Math., 155 (2007), pp. 750–758.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1979.
[8] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin, *On the complexity of constructing evolutionary trees*, J. Comb. Optim., 3 (1999), pp. 183–197.
[9] M. R. Henzinger, V. King, and T. Warnow, *Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology*, Algorithmica, 24 (1999), pp. 1–13.
[10] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree,* 2*-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760.
[11] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic Networks: Concepts, Algorithms and Applications*, Cambridge University Press, Cambridge, UK, 2010.
[12] L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, F. Hagen, and T. Boekhout, *Constructing level-*2 *phylogenetic networks from triplets*, IEEE/ACM Trans. Comput. Biology Bioinform., 6 (2009), pp. 667–681.
[13] J. Jansson, R. S. Lemence, and A. Lingas, *The complexity of inferring a minimally resolved phylogenetic supertree*, in Proceedings of the 10th International Workshop on Algorithms in Bioinformatics, WABI 2010, Lecture Notes in Comput. Sci. 6293, Springer-Verlag, New York, 2010, pp. 262–273.
[14] J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung, *Rooted maximum agreement supertrees*, Algorithmica, 43 (2005), pp. 293–307.
[15] J. Jansson, N. B. Nguyen, and W.-K. Sung, *Algorithms for combining rooted triplets into a galled phylogenetic network*, SIAM J. Comput., 35 (2006), pp. 1098–1121.
[16] P. Kearney, *Phylogenetics and the quartet method*, in Current Topics in Computational Molecular Biology, T. Jiang, Y. Xu, and M. Q. Zhang, eds., MIT Press, Cambridge, MA, 2002, pp. 111–133.
[17] R. Otter, *The number of trees*, Ann. of Math. (2), 49 (1948), pp. 583–599.
[18] R. D. M. Page, *Modified mincut supertrees*, in Proceedings of the 2nd International Workshop on Algorithms in Bioinformatics, WABI 2002, Lecture Notes in Comput. Sci. 2452, Springer-Verlag, New York, 2002, pp. 537–552.
[19] 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*, Systematic Biology, 56 (2007), pp. 798–817.
[20] C. Scornavacca, *Supertree Methods for Phylogenomics*, Ph.D. thesis, University of Montpellier II, Montpellier, France, 2009.
[21] C. Semple, *Reconstructing minimal rooted trees*, Discrete Appl. Math., 127 (2003), pp. 489–503.
[22] C. Semple and M. Steel, *A supertree method for rooted trees*, Discrete Appl. Math., 105 (2000), pp. 147–158.
[23] S. Snir and S. Rao, *Using Max Cut to enhance rooted trees consistency*, IEEE/ACM Trans. Comput. Biology Bioinform., 3 (2006), pp. 323–333.
[24] M. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.
[25] D. Zuckerman, *Linear degree extractors and the inapproximability of Max Clique and Chromatic Number*, Theory Comput., 3 (2007), pp. 103–128.