AIMS Medical Science

http://www.aimspress.com/journal/medicalScience/

AIMS Medical Science, 5(2): 181–203
DOI:10.3934/medsci.2018.2.181
Received: 15 May 2017
Accepted: 02 March 2018
Published: 10 May 2018

Research article

Minimal Phylogenetic Supertrees and Local Consensus Trees

Jesper Jansson[1], **Ramesh Rajaby**[2], **and Wing-Kin Sung**[3,4,*]

[1] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong;

[2] NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore, 28 Medical Drive, Singapore 117456;

[3] School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417;

[4] Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

* **Correspondence:** ksung@comp.nus.edu.sg

Abstract: The problem of constructing a *minimally resolved phylogenetic supertree* (i.e., a rooted tree having the smallest possible number of internal nodes) that contains all of the rooted triplets from a consistent set $\mathcal{R}$ is known to be NP-hard. In this article, we prove that constructing a phylogenetic tree consistent with $\mathcal{R}$ that contains the minimum number of additional rooted triplets is also NP-hard, and develop exact, exponential-time algorithms for both problems. The new algorithms are applied to construct two variants of the *local consensus tree*; for any set $\mathcal{S}$ of phylogenetic trees over some leaf label set L, this gives a minimal phylogenetic tree over L that contains every rooted triplet present in all trees in $\mathcal{S}$, where "minimal" means either having the smallest possible number of internal nodes or the smallest possible number of rooted triplets. (The second variant generalizes the *RV-II tree*, introduced by Kannan *et al.* in 1998.) We also measure the running times and memory usage in practice of the new algorithms for various inputs. Finally, we use our implementations to experimentally investigate the non-optimality of Aho *et al.*'s well-known BUILD algorithm from 1981 when applied to the local consensus tree problems considered here.

Keywords: Phylogenetic tree; rooted triplet; local consensus; minimal supertree; algorithms; computational complexity

1. Introduction

Phylogenetic trees are used to describe evolutionary relationships between species [12]. Numerous methods for reconstructing and comparing phylogenetic trees have been developed, fine-tuned

to different applications and different types of input data [12, 30]. The *supertree approach* is a relatively new divide-and-conquer-based technique for reconstructing phylogenetic trees that may be useful when dealing with very large datasets [5]. The general idea behind it is to first infer a set of highly accurate trees for overlapping subsets of the species (e.g., using a computationally expensive method such as maximum likelihood [10, 12]) and then combine all the trees into one tree according to some well-defined rule. An example of a famous phylogenetic supertree for more than 4500 species can be found in [6]; see also [5, 16] for references to many other supertrees in the biological literature. One class of supertree methods consists of the BUILD algorithm [2] and its various extensions [11, 14, 15, 19, 25–28, 31] for combining a set of *rooted triplets* (binary phylogenetic trees with three leaves each), e.g., inferred by the method in [10].

A *consensus tree* [1, 8, 22] can be regarded as the special case of a phylogenetic supertree where all the trees that are to be combined have the same leaf label set. Such inputs arise when a collection of alternative datasets, each covering all the species, is available, or when applying bootstrapping or different tree reconstruction algorithms to the same basic dataset [12]. A consensus tree can also measure the similarity between two identically leaf-labeled trees or identify parts of trees that are similar. Many different types of consensus trees, whose formal definitions of how to handle conflicts differ, have been proposed in the last 45 years. See the surveys in [8], Chapter 30 in [12], and Chapter 8.4 in [30] for more details about different consensus trees and their advantages and disadvantages, and [20, 21] for some recent algorithmic results.

In situations where more than one phylogenetic tree can explain some given experimental data equally well, it is natural to select a "minimal" tree that supports the data while making as few extra statements about the evolutionary history as possible. A *minimally resolved phylogenetic supertree* [18] is a supertree that is consistent with all of the input and that has the minimum number of internal nodes. By minimizing the number of internal nodes, the risk of creating false groupings called "spurious novel clades" [5] is reduced. Furthermore, such a tree gives a simpler overview of the data than a tree with many internal nodes and can in general be stored in less memory. This makes it easier for scientists to exchange information. Another way to define "minimal" above, giving what we call a *minimally rooted-triplet-inducing phylogenetic supertree*, instead requires that the supertree contains the minimum number of rooted triplets. This interpretation of minimal was previously considered in the definition of the *RV-II local consensus tree* in [22].

There are a few misunderstandings about minimal phylogenetic supertrees in the literature. The goal of this article is to correct these issues, to further develop the underlying mathematical framework, and to design new supertree algorithms that can also be applied to the construction of consensus trees. The algorithms presented here have been implemented and are publicly available.

1.1. Problem Definitions

A *rooted phylogenetic tree* is a rooted, unordered, leaf-labeled tree in which all leaf labels are different and every internal node has at least two children. For example, T_1 and T_2 in figure 1 are two rooted phylogenetic trees. In this article, rooted phylogenetic trees are referred to as "trees" and every leaf in a tree is identified with its unique label.

Let T be a tree. The set of all nodes in T, the set of internal nodes in T, and the set of leaves in T are denoted by $V(T)$, $\mu(T)$, and $\Lambda(T)$, respectively. For any $u, v \in V(T)$, if u is a descendant of v and $u \neq v$ then we write $u \prec v$. The *lowest common ancestor of u and v*, or $lca(u, v)$ for short, is the node w such

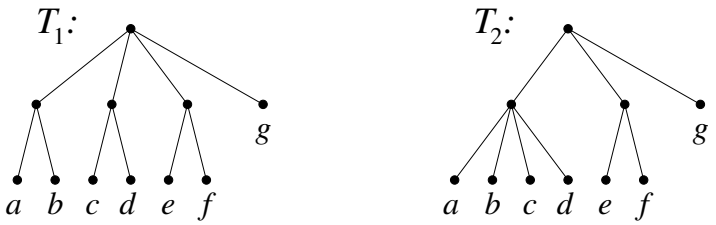


Figure 1. An example. Let $\mathcal{S} = \{T_1, T_2\}$ as above with $\Lambda(T_1) = \Lambda(T_2) = \{a, b, c, d, e, f, g\}$. Then $r(T_1) \cap r(T_2) = \{ab|e,\ ab|f,\ ab|g,\ cd|e,\ cd|f,\ cd|g,\ ef|a,\ ef|b,\ ef|c,\ ef|d,\ ef|g\}$ and T_2 is an optimal solution to MinRLC. On the other hand, $|r(T_1)| = 15$ while $|r(T_2)| = 23$, so T_2 cannot be an optimal solution to MinILC.

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.

A *rooted triplet* is a binary tree with exactly three leaves. We use the notation $xy|z$ to refer to the rooted triplet with leaf label set $\{x, y, z\}$ such that $lca(x, y) \prec lca(x, z) = lca(y, z)$. Let T be a tree. For any $x, y, z \in \Lambda(T)$, if $lca(x, y) \prec lca(x, z) = lca(y, z)$ holds in T then the rooted triplet $xy|z$ and T are said to be *consistent* with each other. For example, $ab|c$ is consistent with T_1 but not with T_2 in figure 1. Observe that for any $\{x, y, z\} \subseteq \Lambda(T)$, exactly zero or one of the three rooted triplets $xy|z$, $xz|y$, and $yz|x$ is consistent with T. The set of all rooted triplets that are consistent with T is denoted by $r(T)$. For any set $\mathcal{R}$ of rooted triplets, if $\mathcal{R} \subseteq r(T)$ then $\mathcal{R}$ and T are *consistent* with each other. Finally, a set $\mathcal{R}$ of rooted triplets is *consistent* if there exists a tree that is consistent with $\mathcal{R}$.

Next, we give the definitions of the *minimally resolved phylogenetic supertree consistent with rooted triplets problem* (MinRS) (studied in [18]) and the *minimally rooted-triplet-inducing phylogenetic supertree consistent with rooted triplets problem* (MinIS). In both problems, the input is a consistent set $\mathcal{R}$ of rooted triplets*, and the output is a tree T satisfying $\Lambda(T) = \bigcup_{t \in \mathcal{R}} \Lambda(t)$ and $\mathcal{R} \subseteq r(T)$. The objectives are to minimize the value of $|\mu(T)|$ (for MinRS) and to minimize the value of $|r(T)|$ (for MinIS), respectively.

In the *minimally resolved local consensus tree problem* (MinRLC) and the *minimally rooted-triplet-inducing local consensus tree problem* (MinILC) (introduced in [22] for the special case $k = 2$), the input is a set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$ for some leaf label set L, and the output is a tree T satisfying $\Lambda(T) = L$ and $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$. The objectives in MinRLC and MinILC are, respectively, to minimize the value of $|\mu(T)|$ and to minimize the value of $|r(T)|$.

Note that MinRLC and MinILC admit polynomial-time reductions to MinRS and MinIS, respectively, by setting $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$.

See figure 1 for a simple example showing that MinRLC and MinILC are indeed different problems, and consequently, that MinRS is different from MinIS. From here on, we will use *Newick notation* to describe trees compactly. E.g., in figure 1, we have $T_1 = ((a, b), (c, d), (e, f), g)$; and $T_2 = ((a, b, c, d), (e, f), g)$;. (For details about Newick notation, the reader is referred to `http://evolution.genetics.washington.edu/phylip/newicktree.html`.)

Throughout the article, the size of the input to MinRS/MinIS is expressed in terms of $k = |\mathcal{R}|$ and

*This article assumes without loss of generality that the input $\mathcal{R}$ to MinRS/MinIS is consistent. The reason is that given an arbitrary $\mathcal{R}$, one can check whether $\mathcal{R}$ is consistent or not in polynomial time using the BUILD algorithm [2] described below.

$n = |L|$, where $L = \bigcup_{t \in \mathcal{R}} \Lambda(t)$. For MinRLC/MinILC, $k = |\mathcal{S}|$ and $n = |L|$, where $L = \Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.

1.2. Previous Work

Here, we give an overview of some relevant results from the literature.

BUILD: Aho *et al.* [2] presented a polynomial-time algorithm called BUILD for determining if an input set $\mathcal{R}$ of rooted triplets is consistent, and if so, constructing a tree T with $\Lambda(T) = \bigcup_{t \in \mathcal{R}} \Lambda(t)$ and $\mathcal{R} \subseteq r(T)$. (When the input $\mathcal{R}$ is *not* consistent, one can for example look for a tree T that maximizes $|r(T) \cap \mathcal{R}|$; cf. Section 2 in [9] for a survey on this problem variant.) BUILD is summarized in Section 2.1 below. Henzinger *et al.* [16] gave a faster implementation of BUILD, and substituting the data structure for dynamic graph connectivity used in the proof of Theorem 1 in [16] by the one in [32] yields a time complexity of $\min\{O(n + k\frac{\log^2 n}{\log\log n}),\ O(k + n^2 \log n)\}$, where $k = |\mathcal{R}|$ and $n = |\bigcup_{t \in \mathcal{R}} \Lambda(t)|$.

Importantly, BUILD does not solve MinRS and MinIS. This was first observed by Bryant [7, Section 2.5.2], who gave the following counterexample: $\mathcal{R} = \{bc|a,\ bd|a,\ ef|a,\ eg|a\}$. Given $\mathcal{R}$ as input, BUILD constructs the tree $T_B = (a,\ (b, c, d),\ (e, f, g))$;, which has three internal nodes and 24 rooted triplets. In contrast, the optimal solution to both MinRS and MinIS is the tree $T_O = (a,\ (b, c, d, e, f, g))$;, which has two internal nodes and 15 rooted triplets. As pointed out in [18], the claim by Henzinger *et al.* in [16] that their Algorithm A' always constructs a minimal tree is therefore false. In another highly cited article, Ng and Wormald [25] presented an extension of BUILD named OneTree to so-called *fans*; however, Note 2 in Section 4 of [25] incorrectly states that OneTree outputs a tree with the minimum number of nodes. Finally, the authors of the textbook [17] seem to have been unaware of Bryant's example, as p. 302 of [17] says it is not known if the tree output by BUILD always contains the minimum number of rooted triplets.

MinRS: For MinRS, the following strong negative result is known: MinRS cannot be approximated within $n^{1-\epsilon}$ for any constant $\epsilon > 0$ in polynomial time, unless P = NP [18]. An algorithm named AllMinTrees in [26] outputs all minor-minimal trees consistent with $\mathcal{R}$, where a tree T is *minor-minimal* if it is consistent with $\mathcal{R}$ and it is not possible to obtain a tree consistent with $\mathcal{R}$ by contracting any edges of T, and this algorithm can be used to solve MinRS. However, it runs in $\Omega((\frac{n}{2})^{n/2})$ time [18], which is self-exponential in $n/2$. Some special cases of MinRS can be solved in polynomial time; e.g., if the output tree has at most three internal nodes or if it is a *caterpillar* (a tree in which every node has at most one child that is an internal node) [18]. Also, for any positive integer p, if every node in the output tree has at most p children which are internal nodes then MinRS can be solved in $p^{O(n)}$ time [18].

MinIS: To determine the computational complexity of MinIS was listed as an open problem in Section 6 in [18]. As far as we know, it has not been studied previously.

MinRLC, MinILC, and local consensus trees: Finding a tree T that satisfies $\bigcap_{i=1}^{k} r(T_i) \subseteq r(T)$ is trivial since one can just select $T = T_1$, but imposing additional conditions makes the output more informative and meaningful. MinRLC and MinILC provide two natural ways to do it. The MinILC problem originates from Kannan *et al.* [22], who gave several alternative definitions of a "local consensus tree". They called a tree T an *RV-II* ("relaxed version II") tree of two trees T_1 and T_2 with identical leaf label sets if $r(T_1) \cap r(T_2) \subseteq r(T)$ and $|r(T)|$ is minimized. Thus, an RV-II tree is a solution to MinILC when $k = 2$. In Section 5.4 of [22], the authors suggested that applying BUILD to the set

Problem	Positive result	Negative result
MinRS	$O(2.733^n)$ time	NP-hard
	(Theorem 1, Section 3)	(Theorem 3.3 in [18])
MinRLC	$O(kn^3 + 2.733^n)$ time	NP-hard
	(Corollary 1, Section 3)	(Theorem 3, Section 5)
MinIS	$O^*(4^n)$ time	NP-hard
	(Theorem 2, Section 4)	(Corollary 4, Section 6)
MinILC	$O(kn^3 + 4^n \cdot poly(n))$ time	NP-hard
	(Corollary 2, Section 3)	(Theorem 4, Section 6)

Table 1. Overview of the computational complexity of the four studied problems.

$r(T_1) \cap r(T_2)$ always produces an RV-II tree, but this is not correct. A counterexample, analogous to the one for MinRS and MinIS above, is obtained by letting T_1 and T_2 be the two trees T_B and T_O, which gives $r(T_1) \cap r(T_2) = \{bc|a, bd|a, cd|a, ef|a, eg|a, fg|a\}$. Then, BUILD's output is T_B but this cannot be a solution to MinRLC or MinILC because T_O has fewer internal nodes and fewer rooted triplets than T_B. This shows that one cannot solve MinRLC/MinILC by taking $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ and applying BUILD directly.

In the consensus tree survey by Bryant [8], "the local consensus tree" is defined as the output of BUILD when given $\bigcap_{i=1}^{k} r(T_i)$ as input. The algorithm in Section 5.4.1 of [22] constructs such a tree in $O(n^2)$ time for the case $k = 2$, while the $O(kn^3)$-time algorithm in Theorem 7 in [16] by Henzinger *et al.* can be used for unbounded k. The advantages of Bryant's BUILD-based local consensus tree are that it is unique and can be computed efficiently. The disadvantages are that it does not minimize the number of nodes or induced rooted triplets and that it is defined in terms of the output of an algorithm and not axiomatically.

1.3. New Results and Organization of the Article

Section 2 reviews the BUILD algorithm from [2] and an enlightening result by Semple in [26] that characterizes all trees consistent with the input $\mathcal{R}$. Based on Semple's characterization, Section 3 gives an $O^*((1 + \sqrt{3})^n) = O(2.733^n)$-time algorithm[†] for MinRS and an $O(kn^3 + 2.733^n)$-time algorithm for MinRLC. Section 4 then describes an $O^*(4^n)$-time algorithm for MinIS and an $O(kn^3 + 4^n \cdot poly(n))$-time algorithm for MinILC. All four problems are NP-hard; MinRS was shown to be NP-hard in [18], and we complement this result by establishing the NP-hardness of MinRLC in Section 5 and the NP-hardness of MinIS and MinILC in Section 6. See Table 1 for a summary of the theoretical results.

Next, Section 7 presents a publicly available implementation of our algorithms and experimental results demonstrating how the running times and memory usage increase as the inputs grow larger. Equipped with these implementations, in Section 8 we then investigate a fundamental question: Is Bryant's BUILD-based local consensus tree (see Section 1.2), which can be computed in polynomial time, a good approximation to MinRLC and MinILC? Our experiments indicate that although the former is not optimal most of the time even for small randomly generated inputs (e.g., for $k = 4$ and $n = 12$), the ratio between the number of internal nodes or the number of rooted triplets in the

[†]The notation $O^*(f(n))$ means $O(f(n) \cdot poly(n))$.

BUILD-based local consensus tree and in an optimal solution is not far from 1 on average for small inputs.

2. Preliminaries

2.1. Aho et al.*'s BUILD Algorithm [2]*

The BUILD algorithm [2] is a recursive, top-down algorithm that takes as input a set $\mathcal{R}$ of rooted triplets and a leaf label set L such that $\bigcup_{t\in\mathcal{R}} \Lambda(t) \subseteq L$ and outputs a tree T with $\Lambda(T) = L$ that is consistent with all of the rooted triplets in $\mathcal{R}$, if such a tree exists; otherwise, it outputs *fail*. The time complexity of BUILD is polynomial (q.v., Section 1.2).

A summary of how BUILD works is given here. It first partitions the leaf label set L into *blocks* according to the information contained in $\mathcal{R}$. More precisely, BUILD constructs an *auxiliary graph*, defined as the undirected graph $\mathcal{G}(L) = (L, E)$ where for any $x, y \in L$, the edge $\{x, y\}$ belongs to E if and only if $\mathcal{R}$ contains at least one rooted triplet of the form $xy|z$ with $z \in L$. It then computes the connected components in $\mathcal{G}(L)$ and assigns the leaf labels in each connected component to one block. (Henceforth, the set of leaf labels belonging to any connected component C in $\mathcal{G}(L)$ is denoted by $\Lambda(C)$, and for every $L' \subseteq L$, we define $\mathcal{R}|L' = \{t \in \mathcal{R} : \Lambda(t) \subseteq L'\}$.) Next, for each block $\Lambda(C)$, BUILD builds a tree T_C by calling itself recursively using $\mathcal{R}|\Lambda(C)$ together with $\Lambda(C)$ as input. Finally, BUILD returns a tree consisting of a newly created root node whose children are the roots of all the recursively constructed T_C-trees. The recursion's base case is when the leaf label set consists of one element x, in which case the algorithm just returns a tree with a single leaf labeled by x. If any auxiliary graph $\mathcal{G}(L')$ constructed during BUILD's execution has only one connected component and $|L'| > 1$ holds then the algorithm terminates and outputs *fail*. See, e.g., [2] for the correctness proof and further details.

Returning to the example in Section 1.2 where we had $\mathcal{R} = \{bc|a, bd|a, ef|a, eg|a\}$ and $L = \{a, b, c, d, e, f, g\}$, the blocks in the auxiliary graph $\mathcal{G}(L)$ are $\{a\}$, $\{b, c, d\}$, and $\{e, f, g\}$. The auxiliary graphs on the successive recursive levels contain no edges, so BUILD outputs the tree $(a, (b, c, d), (e, f, g))$;.

2.2. Semple's Characterization

In [26], Semple clarified the relationship between the auxiliary graph $\mathcal{G}(L)$ used in the BUILD algorithm and the trees consistent with $\mathcal{R}$. For any tree T, define $\pi(T)$ as the partition of $\Lambda(T)$ whose parts are the leaves in the different subtrees attached to the root of T. As an example, $\pi(T_1) = \{\{a, b\}, \{c, d\}, \{e, f\}, \{g\}\}$ in figure 1. With this notation, Semple's characterization can be expressed as:

Lemma 1. *(Corollary 3.3 in [26]) Let T be any tree that is consistent with $\mathcal{R}$. For each connected component C in $\mathcal{G}(L)$, $\Lambda(C) \subseteq B$ for some $B \in \pi(T)$.*

Lemma 1 implies that if T is any tree consistent with $\mathcal{R}$ then the partition $\pi(T)$ can be obtained by performing zero or more mergings of $\mathcal{G}(L)$'s connected components. Thus, every tree consistent with $\mathcal{R}$ can be recovered by trying all possible mergings of the connected components in $\mathcal{G}(L)$ at each recursion level.[‡]

[‡]This technique was actually used even earlier than [26]; the SUPERB algorithm in [11] outputs all binary trees consistent with $\mathcal{R}$ by

We remark that Lemma 1 is very useful. For example, it can be employed to prove the non-uniqueness of solutions to MinRLC and MinILC (and hence, MinRS and MinIS). To illustrate, consider the following instance: $\mathcal{S} = \{T_1, T_2, T_3\}$ with $T_1 = ((1,2,3,4,5,6),(7,8),(9,10),11);$, $T_2 = ((1,2,3,4,5,6,7,8),(9,10),11);$, and $T_3 = ((1,2,3,4,5,6,9,10),(7,8),11);$. The connected components in $\mathcal{G}(L)$, where $\mathcal{R} = \bigcap_{i=1}^{3} r(T_i)$, consist of $\{1,2,3,4,5,6\}$, $\{7,8\}$, $\{9,10\}$, and $\{11\}$. By Lemma 1, we only need to check a few possible candidate trees (corresponding to the different ways of merging these connected components), and we find that each of T_1, T_2, and T_3 is an optimal solution to MinILC since $|r(T_1)| = |r(T_2)| = |r(T_3)| = 93$. Furthermore, each of T_2 and T_3 is an optimal solution to MinRLC.

3. Exponential-Time Algorithms for MinRS and MinRLC

This section presents an exact $O(2.733^n)$-time algorithm for MinRS. As a consequence, MinRLC can be solved in $O(kn^3 + 2.733^n)$ time.

The main idea is to use Lemma 1 together with dynamic programming. For every $L' \subseteq L$, let $opt(L')$ be the number of internal nodes in an optimal solution to MinRS for $\mathcal{R}|L'$. Clearly, if $|L'| = 1$ then $opt(L') = 0$. To compute $opt(L')$ when $|L'| \geq 2$, observe that if T' is any optimal solution for $\mathcal{R}|L'$ then T' consists of a root node whose children are the roots of the optimal solutions for $\mathcal{R}|P_1$, $\mathcal{R}|P_2$, …, $\mathcal{R}|P_t$, where $\{P_1, P_2, \ldots, P_t\}$ is equal to $\pi(T')$. The partition $\pi(T')$ can be found by enumerating the partitions of L' and using dynamic programming to identify the best one; by Lemma 1, only partitions corresponding to the different ways of merging connected components in the auxiliary graph $\mathcal{G}(L')$ need to be considered.

The details are explained next. Suppose $L' \subseteq L$ is given. Let $C_{L'}$ be the set of connected components in $\mathcal{G}(L')$. For every subset $\mathcal{D} \subseteq C_{L'}$, define $Merge(\mathcal{D})$ as the set of all leaf labels belonging to components in $\mathcal{D}$, i.e., $Merge(\mathcal{D}) = \bigcup_{Q \in \mathcal{D}} \Lambda(Q)$. Also define $DP(\mathcal{D})$ for every $\mathcal{D} \subseteq C_{L'}$ to be the minimum value of $\sum_{\mathcal{X} \in \mathcal{Q}} opt(Merge(\mathcal{X}))$ taken over all possible true partitions $\mathcal{Q}$ of $\mathcal{D}$, where we say that a partition $\mathcal{Q}$ of a set X is a *true partition of* X if $|X| \geq 2$ and $\mathcal{Q} \neq \{X\}$ (i.e., if $|\mathcal{Q}| > 1$), or if $|X| = |\mathcal{Q}| = 1$. Then:

Lemma 2. *For every $L' \subseteq L$ with $|L'| \geq 2$, it holds that $opt(L') = DP(C_{L'}) + 1$.*

Proof. Let T' be any optimal tree for $\mathcal{R}|L'$. The children of the root of T' are the roots of the optimal solutions for $\mathcal{R}|P_1$, $\mathcal{R}|P_2$, …, $\mathcal{R}|P_t$, where each P_i equals $Merge(\mathcal{D})$ for some $\mathcal{D} \subseteq C_{L'}$ because of Lemma 1. By definition, $DP(C_{L'})$ is the minimum value of $\sum_{\mathcal{X} \in \mathcal{P}} opt(\mathcal{X})$ over all true partitions $\mathcal{P}$ of L' such that each $\mathcal{X} \in \mathcal{P}$ equals $Merge(\mathcal{D})$ for some $\mathcal{D} \subseteq C_{L'}$. Together with the common root node, this gives $opt(L') = DP(C_{L'}) + 1$. □

Lemma 3. *For every $\mathcal{D} \subseteq C_{L'}$ with $|\mathcal{D}| \geq 2$, $DP(\mathcal{D}) = \min_{\emptyset \neq \mathcal{X} \subseteq \mathcal{D}} \{opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(Merge(\mathcal{D} \setminus \mathcal{X}))\}\}$.*

Proof. $DP(\mathcal{D}) = \min\{\sum_{\mathcal{X} \in \mathcal{Q}} opt(Merge(\mathcal{X})) : \mathcal{Q}$ is a true partition of $\mathcal{D}\} =$
$\min\{opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(Merge(\mathcal{D} \setminus \mathcal{X}))\} : \mathcal{X} \in \mathcal{Q}, \mathcal{Q}$ is a true partition of $\mathcal{D}\} = \min\{opt(Merge(\mathcal{X})) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(Merge(\mathcal{D} \setminus \mathcal{X}))\} : \emptyset \neq \mathcal{X} \subsetneq \mathcal{D}\}$. □

considering all ways of merging the connected components of $\mathcal{G}(L)$ into exactly two connected components at each recursion level.

Algorithm `MinRS_exact`

Input: A consistent set $\mathcal{R}$ of rooted triplets over a leaf label set L.

Output: The number of internal nodes in a minimally resolved supertree consistent with $\mathcal{R}$ and leaf-labeled by L.

For every $x \in L$, initialize $opt(\{x\}) := 0$;
for $i := 2$ to n **do**
  for every cardinality-i subset L' of L **do**
    Construct $\mathcal{G}(L')$. Let $\mathcal{C}$ and $\mathcal{U}$ be the set of connected components and the set of singleton components, respectively, in $\mathcal{G}(L')$;
    Let $DP(\emptyset) := 0$. For every $X \in \mathcal{C} \setminus \mathcal{U}$, let $DP(\{X\}) := opt(\Lambda(X))$;
    for $j := 2$ to $|\mathcal{C}| - |\mathcal{U}|$ **do**
      for every cardinality-j subset $\mathcal{D}$ of $\mathcal{C} \setminus \mathcal{U}$ **do**
        $DP(\mathcal{D}) :=$
          $\min_{\emptyset \neq \mathcal{X} \subseteq \mathcal{D}} \{opt(\bigcup_{Q \in \mathcal{X}} \Lambda(Q)) + \min\{DP(\mathcal{D} \setminus \mathcal{X}), opt(\bigcup_{Q \in \mathcal{D} \setminus \mathcal{X}} \Lambda(Q))\}\}$;
      end for
    end for
    $opt(L') := DP(\mathcal{C} \setminus \mathcal{U}) + 1$;
  end for
end for
return $opt(L)$;

Figure 2. Algorithm `MinRS_exact`.

Lemmas 2 and 3 suggest the following strategy: Compute $opt(L')$ for all subsets L' of L in order of increasing cardinality by evaluating the formula in Lemma 2, while using dynamic programming to compute and store the relevant DP-values. To do this, for each L', we first construct $\mathcal{G}(L')$ in polynomial time. We then enumerate all subsets $\mathcal{D}$ of $\mathcal{C}_{L'}$ in a loop having $|\mathcal{C}_{L'}|$ iterations in which iteration j uses Lemma 3 to compute all $DP(\mathcal{D})$-values where $|\mathcal{D}| = j$. Each application of Lemma 3 takes $O^*(2^{|\mathcal{D}|})$ time, so this takes a total of $O^*(\sum_{j=2}^{|\mathcal{C}_{L'}|} \binom{|\mathcal{C}_{L'}|}{j} 2^j) = O^*((2+1)^{|\mathcal{C}_{L'}|}) = O^*(3^{|\mathcal{C}_{L'}|})$ time for each L' by binomial expansion. To obtain $opt(L)$, we iterate over all subsets L' of L of cardinality $i = 2, 3, \ldots, n$; iteration i computes $opt(L')$ for each L' with $|L'| = i$ in $O^*(3^{|\mathcal{C}_{L'}|})$ time as just described. The total running time becomes $O^*(\sum_{i=2}^{n} \binom{n}{i} 3^i) = O^*((3+1)^n) = O^*(4^n)$.

To reduce the time complexity, we will reduce the number of applications of Lemma 3 in the main loop that computes $opt(L')$ for any $L' \subseteq L$. We rely on the following simple observation, which essentially tells us that the singleton components of $\mathcal{G}(L')$ can be ignored.

Lemma 4. *Let $\mathcal{U}$ be the set of singleton components in $\mathcal{G}(L')$. $DP(\mathcal{C}_{L'}) = DP(\mathcal{C}_{L'} \setminus \mathcal{U})$.*

Proof. Consider any $x \in \mathcal{U}$. By the construction of $\mathcal{G}(L')$ and $\mathcal{U}$, there are no rooted triplets of the form $xy|z$ for any $y, z \in L'$ in the set $\mathcal{R}|L'$. Hence, there exists a minimally resolved supertree consistent with $\mathcal{R}|L'$ in which x is attached directly to the root. The lemma follows. □

The resulting algorithm, called `MinRS_exact`, is summarized in figure 2.

Theorem 1. *Algorithm* `MinRS_exact` *solves* MinRS *in $O^*((1+\sqrt{3})^n)$ time.*

Proof. First note that $C_{L'} \setminus \mathcal{U}$ contains no singleton components. Therefore, the number of connected components in $C_{L'} \setminus \mathcal{U}$ is at most $\frac{|L'|}{2}$, i.e., $|C_{L'}| - |\mathcal{U}| \leq \frac{|L'|}{2}$. Now, when computing $opt(L')$ for any subset L' of L, the number of applications of the formula in Lemma 3 is reduced since there no are subsets $\mathcal{D}$ of cardinality larger than $|C_{L'}| - |\mathcal{U}|$. More precisely, the time for each L' is reduced to $O^*(\sum_{j=2}^{|C_{L'}|-|\mathcal{U}|} \binom{|C_{L'}|-|\mathcal{U}|}{j} 2^j) = O^*((2+1)^{|C_{L'}|-|\mathcal{U}|}) = O^*(3^{|C_{L'}|-|\mathcal{U}|}) = O^*(\sqrt{3}^{|L'|})$. Finally, replacing $O^*(3^{|C_{L'}|})$ by $O^*(\sqrt{3}^{|L'|})$ in the analysis of computing $opt(L)$ above gives a total time complexity of $O^*(\sum_{i=2}^{n} \binom{n}{i} \sqrt{3}^i)$ $= O^*((\sqrt{3}+1)^n)$. □

Remark 1: The algorithm as presented here returns $opt(L)$. An optimal tree with this number of internal nodes can be obtained by standard traceback techniques.

Remark 2: For each $L' \subseteq L$, the algorithm needs to store the value of $opt(L')$ and there are $\Omega(2^n)$ such subsets. Therefore, the space complexity of the algorithm is also exponential in n.

Corollary 1. MinRLC *can be solved in* $O(kn^3 + (1+\sqrt{3})^n \cdot poly(n))$ *time.*

Proof. First construct $\mathcal{R} = \bigcap_{i=1}^{k} r(T_i)$ in $O(kn^3)$ time, e.g., by preprocessing each T_i in $O(n)$ time so that any query of the form $lca(x, y)$ in T_i with $x, y \in L$ can be answered in $O(1)$ time [4] and then, for every $L' \subseteq L$ with $|L'| = 3$, doing $3k$ *lca*-queries to see if L' induces the same rooted triplet in all of the k trees. Next, run `MinRS_exact` on $\mathcal{R}$. □

In Section 7, we shall refer to the algorithm in Corollary 1 as `MinRLC_exact`.

4. Exponential-Time Algorithms for MinIS and MinILC

We now describe an $O^*(4^n)$-time algorithm for MinIS based on the technique from Section 3. Applying it to MinILC yields an $O(kn^3 + 4^n \cdot poly(n))$-time algorithm for the latter.

Lemma 1 guarantees that every valid solution to MinIS can be discovered by trying all ways of merging connected components in the auxiliary graphs $\mathcal{G}(L')$. As in Section 3, we use dynamic programming to compute and store optimal values to subproblems but make the following modifications. First of all, redefine *opt* so that $opt(L')$ for every $L' \subseteq L$ means the value of $|r(T')|$ for an optimal solution T' to MinIS for $\mathcal{R}|L'$. Secondly, redefine $DP(\mathcal{D})$ for every $\mathcal{D} \subseteq C_{L'}$ to mean the minimum value of $\sum_{\mathcal{X} \in \mathcal{Q}} (opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})|)$, taken over all possible true partitions $\mathcal{Q}$ of $\mathcal{D}$. With the new definitions of *opt* and *DP*, the analogues of Lemmas 2 and 3 become:

Lemma 5. *For every* $L' \subseteq L$ *with* $|L'| \geq 2$*, it holds that* $opt(L') = DP(C_{L'})$.

Proof. $DP(C_{L'})$ counts the minimum number of rooted triplets in a tree consistent with $\mathcal{R}|L'$ among all partitions $\mathcal{Q}$ of $C_{L'}$. Hence, $opt(L') = DP(C_{L'})$. □

Lemma 6. *For every* $\mathcal{D} \subseteq C_{L'}$ *with* $|\mathcal{D}| \geq 2$, $DP(\mathcal{D}) = \min_{\emptyset \neq \mathcal{X} \subsetneq \mathcal{D}} \{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})|$
$+ \min\{DP(\mathcal{D} \setminus \mathcal{X}),\ opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D} \setminus \mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})|\}\}$.

Proof. $DP(\mathcal{D}) = \min\{\sum_{\mathcal{X} \in \mathcal{Q}} (opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})|) : \mathcal{Q}$ is a true partition of $\mathcal{D}\} =$
$\min\{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})| + \min\{DP(\mathcal{D} \setminus X),$
$opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D} \setminus \mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})| \} : \mathcal{X} \in \mathcal{Q}$, $\mathcal{Q}$ is a true partition of $\mathcal{D}\} =$

$\min\{opt(Merge(\mathcal{X})) + \binom{|Merge(\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{X})| + \min\{DP(\mathcal{D} \setminus X),\ opt(Merge(\mathcal{D} \setminus \mathcal{X})) + \binom{|Merge(\mathcal{D}\setminus\mathcal{X})|}{2} \cdot |L' \setminus Merge(\mathcal{D} \setminus \mathcal{X})|\} : \emptyset \neq \mathcal{X} \subsetneq \mathcal{D}\}$. □

The new algorithm, called `MinIS_exact`, is obtained by modifying Algorithm `MinRS_exact` as follows:

- Change the last part of Step 5 so that it assigns $DP(\{X\}) := opt(\Lambda(X)) + \binom{|\Lambda(X)|}{2} \cdot |L' \setminus \Lambda(X)|$, according to the new definition of DP.
- Change Step 8 so that it computes $DP(\mathcal{D})$ using Lemma 6 instead of Lemma 3.
- Change Step 11 so that it assigns $opt(L') := DP(C_{L'})$, in accordance with Lemma 5.
- Change Step 4 so that it always sets $\mathcal{U}$ to $\emptyset$.

The reason why we force $\mathcal{U} = \emptyset$ is that we do not have an analogue of Lemma 4 for MinIS that would allow us to ignore the singleton components. The algorithm therefore spends $O^*(\sum_{j=2}^{|C_{L'}|} \binom{|C_{L'}|}{j} 2^j) = O^*((2+1)^{|C_{L'}|}) = O^*(3^{|C_{L'}|})$ time for each L', just like the slower version of Algorithm `MinRS_exact` in Section 3, and the total running time is $O^*(\sum_{i=2}^{n} \binom{n}{i} 3^i) = O^*((3+1)^n) = O^*(4^n)$.

Theorem 2. *Algorithm* `MinIS_exact` *solves* MinIS *in* $O^*(4^n)$ *time.*

Corollary 2. MinILC *can be solved in* $O(kn^3 + 4^n \cdot poly(n))$ *time.*

Section 7 refers to the algorithm in Corollary 2 as `MinILC_exact`.

5. NP-Hardness of MinRLC

Section 3 in [18] proved that MinRS is NP-hard. It follows from the proof in [18] that MinRS remains NP-hard even if restricted to a particular special case which we now describe.

Suppose that $L_0 = \{v_1, v_2, \ldots, v_q\}$ is a set of elements. Define $L_0' = \{v_{1'}, v_{1''}, v_{2'}, v_{2''}, \ldots, v_{q'}, v_{q''}\}$, and for any integers i, j with $1 \leq i < j \leq q$, define $\mathcal{R}(v_i, v_j)$ as the set of four rooted triplets $\{v_{i'}v_{i''}|v_{j'}, v_{i'}v_{i''}|v_{j''}, v_{j'}v_{j''}|v_{i'}, v_{j'}v_{j''}|v_{i''}\}$ over L_0'. For any set S, let $\binom{S}{2}$ denote the set of all subsets of S of cardinality 2. According to Section 3 in [18], MinRS is NP-hard even if restricted to instances where $\mathcal{R}$ has the form $\mathcal{R} = \bigcup_{\{v_i,v_j\}\in Z} \mathcal{R}(v_i, v_j)$ for some set L_0 and some $Z \subseteq \binom{L_0}{2}$.

Theorem 3. MinRLC *is NP-hard.*

Proof. We reduce from the above variant of MinRS. Let $\mathcal{R}$ be any given instance of the problem. Let P be the set of pairs of indices that form rooted triplets in $\mathcal{R}$, i.e., $P = \{\{i, j\} : v_{i'}v_{i''}|v_{j'}, v_{i'}v_{i''}|v_{j''}, v_{j'}v_{j''}|v_{i'}, v_{j'}v_{j''}|v_{i''} \in \mathcal{R}\}$, and let $Q = \binom{\{1,2,\ldots,q\}}{2} \setminus P$.

Define a tree $T_0 = ((v_{1'}, v_{1''}), (v_{2'}, v_{2''}), \ldots, (v_{q'}, v_{q''}))$; and for every $f = \{x, y\} \in Q$, define a tree T_f by taking a copy of T_0 and merging the two subtrees $(v_{x'}, v_{x''})$ and $(v_{y'}, v_{y''})$ so that $T_f = ((v_{x'}, v_{x''}, v_{y'}, v_{y''}), (v_{1'}, v_{1''}), (v_{2'}, v_{2''}), \ldots, (v_{n'}, v_{n''}))$;. Let $\mathcal{S} = \{T_0\} \cup \{T_f : f \in Q\}$. Note that $\mathcal{R} = \bigcap_{T_i\in\mathcal{S}} r(T_i)$. This is because for any $\{x, y\} \in P$, the four rooted triplets $v_{x'}v_{x''}|v_{y'}$, $v_{x'}v_{x''}|v_{y''}$, $v_{y'}v_{y''}|v_{x'}$, $v_{y'}v_{y''}|v_{x''}$ appear in $\mathcal{R}$ as well as in $r(T_i)$ for every $T_i \in \mathcal{S}$. On the other hand, for any $\{x, y\} \in Q$, $v_{x'}v_{x''}|v_{y'}$, $v_{x'}v_{x''}|v_{y''}$, $v_{y'}v_{y''}|v_{x'}$, $v_{y'}v_{y''}|v_{x''}$ do not appear in $\mathcal{R}$ or in $r(T_{\{x,y\}})$. Thus, there exists a tree T with $\bigcap_{T_i\in\mathcal{S}} r(T_i) \subseteq r(T)$ having x internal nodes if and only if there exists a tree T' with $\mathcal{R} \subseteq r(T')$ having x internal nodes. □

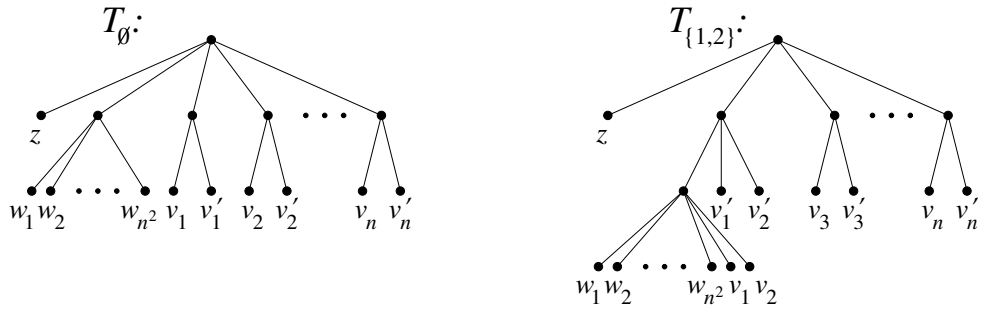


Figure 3. The tree $T_\emptyset$ defined in the reduction from MAXIMUM CLIQUE and a tree $T_{\{1,2\}}$.

6. NP-Hardness of MINILC and MINIS

To prove the NP-hardness of MINILC, we give a polynomial-time reduction from the MAXIMUM CLIQUE problem, which is NP-hard [13]. MAXIMUM CLIQUE takes as input an undirected graph $G = (V, E)$ and asks for a largest clique in G, where $X \subseteq V$ is a *clique in G* if every two vertices belonging to X are adjacent in G.

The reduction is as follows. Let $n = |V|$ and write $V = \{1, 2, \ldots, n\}$. Create a set L of leaf labels such that $L = \{v_i, v_i' : i \in V\} \cup \{z, w_1, w_2, \ldots, w_{n^2}\}$. Let $T_\emptyset$ be the tree $(z, (w_1, w_2, \ldots, w_{n^2}), (v_1, v_1'), (v_2, v_2'), \ldots, (v_n, v_n'))$; with $\Lambda(T_\emptyset) = L$. For any nonempty subset $X = \{i_1, i_2, \ldots, i_p\} \subseteq V$, define T_X as the tree with $\Lambda(T_X) = L$ obtained by taking a copy of $T_\emptyset$, deleting the subtrees (v_i, v_i') for all $i \in X$, and replacing the subtree $(w_1, w_2, \ldots, w_{n^2})$ by $((w_1, w_2, \ldots, w_{n^2}, v_{i_1}, v_{i_2}, \ldots, v_{i_p}), v_{i_1}', v_{i_2}', \ldots, v_{i_p}')$. See figure 3 for an illustration. Finally, let $\mathcal{S} = \{T_\emptyset\} \cup \{T_{\{i\}} : i \in V\} \cup \{T_{\{i,j\}} : \{i, j\} \in E\}$.

Section 6.1 first states some general properties satisfied by the trees defined above. Then, Section 6.2 establishes some specific properties satisfied by any local consensus tree of $\mathcal{S}$. After that, we will prove that for any $X \subseteq V$, X is a maximum clique in G if and only if T_X is a local consensus tree of $\mathcal{S}$ with the smallest possible number of rooted triplets, giving the main result of this section.

6.1. General Properties

The following additional notation is used. For any tree H, $Child(H)$ is the set of children of the root of H. For any $u \in V(H)$, the subtree of H induced by u and all proper descendants of u is called *the subtree of H rooted at u* and is denoted by H^u. For any $L' \subseteq \Lambda(H)$, $H|L'$ is the tree obtained from H by deleting all nodes with no descendants in L' and their incident edges, and then contracting every edge between a node having one child and its child. Finally, for every positive integer n, define a function $f_n(k) = n^5 - \frac{k-1}{2}n^4 + 4kn^3 - \frac{6k^2-3k-3}{2}n^2 + (4k^2 - 4k - 1)n - \frac{7k^3-7k^2}{2}$. We immediately have:

Lemma 7. *For any tree H,* $|r(H)| = \sum_{u \in Child(H)} \left(|r(H^u)| + \binom{|\Lambda(H^u)|}{2} \cdot (|\Lambda(H)| - |\Lambda(H^u)|) \right)$.

Lemma 8. *Let X be any subset of the given V. Write* $k = |X|$. *Then* $|r(T_X)| = f_n(k)$.

Proof. By Lemma 7, the number of rooted triplets consistent with T_X is $\binom{n^2+k}{2} \cdot k + \binom{n^2+2k}{2} \cdot (2n - 2k + 1) + (n - k) \cdot (n^2 + 2n - 1)$. Expanding this expression yields the formula. □

Corollary 3. *For any fixed* $n \geq 8$, $f_n(k)$ *is strictly decreasing as k increases.*

Proof. By Lemma 8, $f_n(k+1) - f_n(k) = -\frac{1}{2}n^4 + 4n^3 - \frac{12k+3}{2}n^2 + 8kn - \frac{7}{2}(3k^2+k)$. Since $n \geq 8$, $-\frac{1}{2}n^4 + 4n^3 \leq 0$ holds. Also, $\frac{12k+3}{2}n > 8k$ for $n \geq 8$ and $-\frac{7}{2}(3k^2+k) \leq 0$. The corollary follows. □

Lemma 9. *Consider any $u \in Child(H)$ in a tree H. Suppose that $\Lambda(H^u) = \alpha \cup \beta$ for some $\alpha, \beta \neq \emptyset$ with $\alpha \cap \beta = \emptyset$. Let H' be the tree obtained from H by deleting H^u and its parent edge and attaching the roots of $H|\alpha$ and $H|\beta$ as children of the root of H. If $|\alpha| + |\beta| \leq \frac{2|\Lambda(H)|}{3}$ then $|r(H')| < |r(H)|$.*

Proof. Define $m = |\Lambda(H)|$. Lemma 7 gives $|r(H)| - |r(H')| = |r(H^u)| + \binom{|\alpha|+|\beta|}{2} \cdot (m - |\alpha| - |\beta|) - |r(H|\alpha)| - \binom{|\alpha|}{2} \cdot (m - |\alpha|) - |r(H|\beta)| - \binom{|\beta|}{2} \cdot (m - |\beta|)$. Noting that $|r(H^u)| \geq |r(H|\alpha)| + |r(H|\beta)|$, we have $|r(H)| - |r(H')| \geq \binom{|\alpha|+|\beta|}{2} \cdot (m - |\alpha| - |\beta|) - \binom{|\alpha|}{2} \cdot (m - |\alpha|) - \binom{|\beta|}{2} \cdot (m - |\beta|) = |\alpha| \cdot |\beta| \cdot (m + 1 - \frac{3}{2} \cdot (|\alpha| + |\beta|)) \geq |\alpha| \cdot |\beta| \cdot (m + 1 - m) = |\alpha| \cdot |\beta| > 0$. □

6.2. Properties of a Local Consensus Tree of $\mathcal{S}$

By the definition of $\mathcal{S}$, we have the next lemma.

Lemma 10. *The set $\bigcap_{T_i \in \mathcal{S}} r(T_i)$ consists of the following rooted triplets:*

- *$w_i w_j | z$ for all $1 \leq i < j \leq n^2$ and $v_i v'_i | z$ for all $1 \leq i \leq n$;*
- *$w_i w_j | v'_k$ for all $1 \leq i < j \leq n^2$ and $1 \leq k \leq n$;*
- *$v_i v'_i | v_j$, $v_i v'_i | v'_j$, $v_j v'_j | v_i$, and $v_j v'_j | v'_i$ for all $1 \leq i < j \leq n$ with $\{i, j\} \notin E$.*

Let T be any local consensus tree of $\mathcal{S}$, i.e., any tree T such that $\Lambda(T) = L$ and $\bigcap_{T_i \in \mathcal{S}} r(T_i) \subseteq r(T)$. According to Lemma 10, $r(T)$ contains $v_i v'_i | z$ for all $1 \leq i \leq n$, so the two leaves v_i and v'_i must belong to the same subtree attached to the root of T for all $1 \leq i \leq n$. Similarly, all leaves in $\{w_1, w_2, \ldots, w_{n^2}\}$ must belong to one subtree attached to the root of T. The *core of T*, denoted by γ_T, is the subtree of T rooted at the node $lca(w_1, w_2, \ldots, w_{n^2})$. The path from the root of T to the parent of γ_T is called the *core path of T*. For any node $u \in V(T)$, if u is a child of the core path of T that does not belong to the core path itself and $u \neq lca(w_1, w_2, \ldots, w_{n^2})$, the subtree of T rooted at u is called a *secondary subtree of T*. Note that the secondary subtrees of T are disjoint. Define $C_T = \{i : v_i \in \Lambda(\gamma_T)\}$.

Lemma 11. *Let T be a local consensus tree of $\mathcal{S}$. T has the following properties:*

1. *The core γ_T does not contain the leaf v'_i for any $1 \leq i \leq n$.*
2. *C_T forms a clique in G.*
3. *For any $i \in \{1, 2, \ldots n\}$, if $C_T \cup \{i\}$ is not a clique in G then v_i and v'_i belong to the same secondary subtree of T.*

Proof.

1. Suppose $v'_i \in \Lambda(\gamma_T)$. Let w_a, w_b be any two leaves such that $lca(\{w_a, w_b\}) = lca(\{w_1, w_2, \ldots, w_{n^2}\})$. Then, $w_a w_b | v'_i \notin r(T)$, contradicting Lemma 10.
2. Consider any $i, j \in C_T$ with $i \neq j$. By point 1., $v_i v_j | v'_i \in r(T)$, so $v_i v'_i | v_j \notin r(T)$. According to Lemma 10, $\{i, j\} \notin E$ does not hold, which means that $\{i, j\} \in E$.
3. Since $C_T \cup \{i\}$ is not a clique, there exists some $j \in C_T$ where $\{i, j\} \notin E$. By Lemma 10, $v_i v'_i | v_j \in r(T)$. Thus, v_i and v'_i are in the same subtree attached to the core path.

□

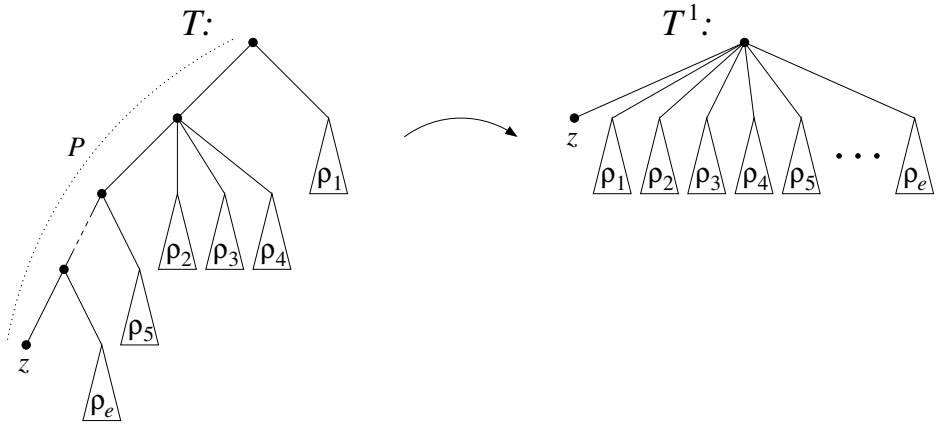


Figure 4. Illustrating the first part of the proof of Lemma 12. Any local consensus tree T of $\mathcal{S}$ can be transformed into a tree of the form shown in T^1 without losing any of the rooted triplets specified in Lemma 10. P is the path in T from the root to the leaf z, to which the subtrees $\rho_1, \rho_2, \ldots, \rho_e$ are attached. For each $i \in \{1, 2, \ldots, n\}$, the two leaves v_i and v'_i belong to the same ρ_j-subtree by Lemma 10. Also, all leaves in $\{w_1, w_2, \ldots, w_{n^2}\}$ are in a single ρ_j-subtree.

Observe that Lemma 11.1 implies $\Lambda(\gamma_T) = \{w_1, w_2, \ldots, w_{n^2}\} \cup \{v_p \mid p \in C_T\}$. Moreover, by Lemma 11.3, for any $i \in \{1, 2, \ldots n\}$, if v_i and v'_i belong to subtrees attached to different nodes on the core path then $C_T \cup \{i\}$ is a clique in G.

Lemma 12. *Let $n \geq 10$ and let T be a local consensus tree of $\mathcal{S}$. T can be transformed into a local consensus tree of $\mathcal{S}$ of the form T_X for some $X \subseteq V$ where X is a clique in G and $|r(T_X)| \leq |r(T)|$.*

Proof. We describe a sequence of transformations that can be applied to T without increasing the number of rooted triplets consistent with it. After each transformation, the resulting tree still contains all of the rooted triplets listed in Lemma 10, so it is still a local consensus tree of $\mathcal{S}$.

First, consider the leaf z in T. Let P be the path from the root of T to z, and let $\rho_1, \rho_2, \ldots, \rho_e$ (where possibly $e = 0$) be the subtrees of T attached to P whose roots do not belong to P themselves. Let T^1 be the tree formed by removing P and attaching z and the roots of $\rho_1, \rho_2, \ldots, \rho_e$ as children of the root. See figure 4. Then $|r(T^1)| \leq |r(T)|$, and T^1 has the property that z is a child of the root of T^1.

Secondly, transform T^1 to T^2 by contracting the core γ_{T^1}, i.e., by replacing γ_{T^1} by a single node to which all leaves in $\Lambda(\gamma_{T^1})$ are directly attached. See figure 5. Clearly, $|r(T^2)| \leq |r(T^1)|$.

Thirdly, suppose that for some $s \in \{1, 2, \ldots, n\}$, it holds that $s \notin C_{T^2}$ while $C_{T^2} \cup \{s\}$ is a clique in G. Let T^3 be the tree formed by removing the leaf v_s from its location in T^2 and attaching it to the root of γ_{T^2}, and moving the leaf v'_s so that it becomes a sibling of the root of γ_{T^2}. (If, as a result, any node has only one child left then contract its outgoing edge.) See figure 6. There are two types of rooted triplets involving v_s: (i) $xv_s|y$ and (ii) $xy|v_s$. For (i), T^3 has at most $(n^2 + n - 1) \cdot 2n$ more rooted triplets than T^2 of this form because there are at most $n^2 + n - 1$ choices of x by Lemma 11.1 and at most $2n$ choices of y. For (ii), there are at least $\binom{n^2}{2}$ rooted triplets in T^2 but not in T^3, corresponding to pairs of the form (w_i, w_j), and at most $\binom{2n}{2}$ such rooted triplets in T^3 that are not present in T^2. Similarly, there are two types of rooted triplets involving v'_s: (iii) $xv'_s|y$ and (iv) $xy|v'_s$. As above, T^3 has at most $(n^2 + n - 1) \cdot 2n$ more rooted triplets than T^2 of the form (iii) and at most $\binom{2n}{2}$ more rooted triplets of

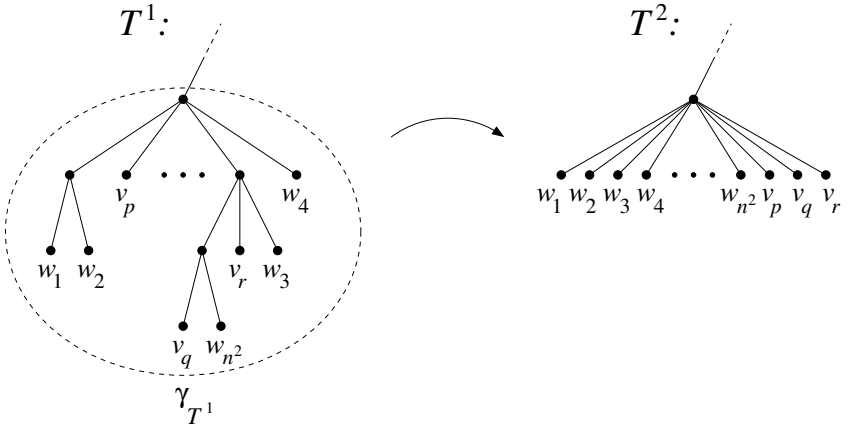


Figure 5. Transforming T^1 to T^2.

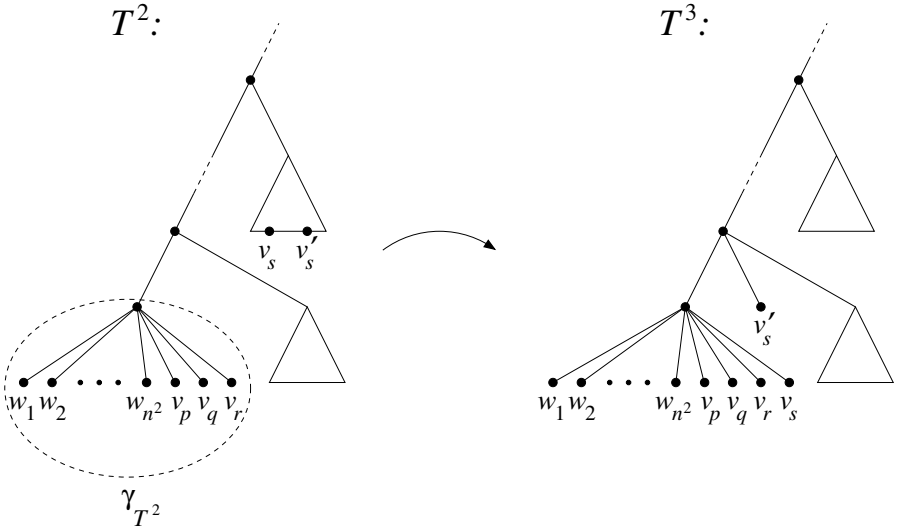


Figure 6. Transforming T^2 to T^3.

the form (iv). Hence, by transforming T^2 to T^3, the number of rooted triplets is reduced by at least $\binom{n^2}{2} - 2 \cdot \binom{2n}{2} - 2 \cdot (n^2 + n - 1) \cdot (2n)$, which is larger than 0 when $n \geq 10$. We repeat this step until T^3 has no leaf v_s such that $s \notin C_{T^3}$ and $C_{T^3} \cup \{s\}$ is a clique. This gives $|r(T^3)| \leq |r(T^2)|$.

Next, transform T^3 to a tree T^4 in which every secondary subtree contains at most two leaves and, furthermore, the leaves in any secondary subtree with precisely two leaves are of the form $\{v_q, v_q'\}$ where $C_{T^4} \cup \{q\}$ is not a clique in G. To do this, consider any secondary subtree σ of T^3. By the definition of T^3 in the previous paragraph, $C_{T^3} \cup \{q\}$ is not a clique in G for any $v_q \in \Lambda(\sigma)$. Recall from Lemma 11.3 that any two leaves of the form v_q and v_q' must belong to the same secondary subtree. While $|\Lambda(\sigma)| > 2$, extract any pair of leaves $\{v_q, v_q'\}$ from σ and create a new secondary subtree with the leaves $\{v_q, v_q'\}$ attached to the core path as a sibling of σ. (As above, if any node has only one child left after these operations then contract its outgoing edge.) See figure 7. Every secondary subtree σ satisfies $|\Lambda(\sigma)| \leq 2n \leq \frac{2n^2}{3} \leq \frac{2|\Lambda(H)|}{3}$, where H is the subtree rooted at the parent of the root of σ, so we get $|r(T^4)| \leq |r(T^3)|$ by Lemma 9.

Lastly, transform T^4 to a tree T^5 whose core path consists of a single edge (u_0, u_1), where u_0 is the

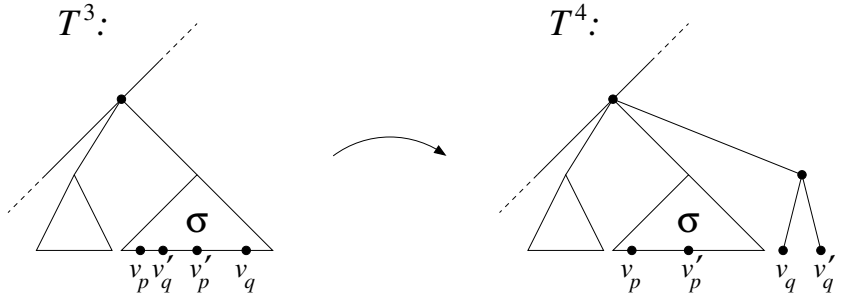


Figure 7. Transforming T^3 to T^4.

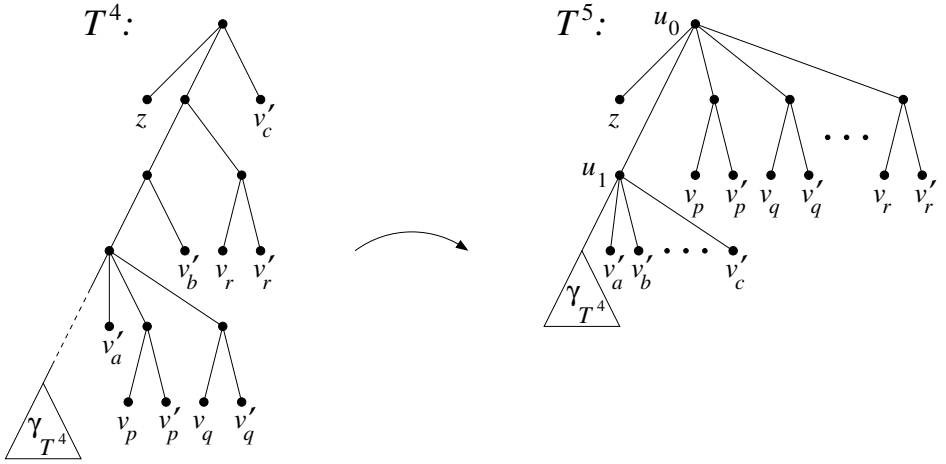


Figure 8. Transforming T^4 to T^5.

root of T^5, as follows. Attach every secondary subtree of T^4 having two leaves as a child of u_0, and attach every secondary subtree of T^4 having one leaf as a child of u_1. Attach z as a child of u_0 and the core γ_{T^4} as a child of u_1. See figure 8. Note that this will not destroy any of the rooted triplets in Lemma 10 and that $|r(T^5)| \leq |r(T^4)|$.

By the definition of T_X, T^5 is equal to T_X if we select $X = C_{T^5}$. Finally, C_{T^5} is a clique in G by Lemma 11.2. □

Lemma 13. *Let $n \geq 10$. $X \subseteq V$ is a maximum clique in G if and only if T_X is a local consensus tree of $\mathcal{S}$ that minimizes the number of rooted triplets.*

Proof. ($\rightarrow$) For the purpose of obtaining a contradiction, suppose there exists a local consensus tree T' of $\mathcal{S}$ with $|r(T')| < |r(T_X)|$. Apply Lemma 12 to T' to get a tree T_Q that is also a local consensus tree of $\mathcal{S}$ with $|r(T_Q)| \leq |r(T')|$ and where Q is a clique in G. Then $|Q| > |X|$ by Lemma 8 and Corollary 3, which is impossible.

($\leftarrow$) Suppose X' is a larger clique in G than X. Lemma 8 and Corollary 3 imply $|r(T_{X'})| < |r(T_X)|$, contradicting that T_X is a local consensus tree of $\mathcal{S}$ minimizing the number of rooted triplets. □

Now, assuming without loss of generality that $n \geq 10$ in the reduction from Maximum Clique above, Lemma 13 gives:

Theorem 4. MinILC *is NP-hard.*

Finally, by the reduction from MinILC to MinIS mentioned in Section 1.1:

Corollary 4. MinIS *is NP-hard.*

7. Implementations

We have implemented the algorithms from Sections 3 and 4 above in C++. The source code can be downloaded from:

`https://github.com/Mesh89/FACT2`

(This package also contains an algorithm for computing another type of consensus tree called the *frequency difference consensus tree*, described in detail in [20].) Subsets, as well as components, were represented as bitsets and implemented as unsigned 32-bit integers. This means that the current implementation can deal with up to 32 leaves; it can be extended by using 64-bit integers or a proper bitset, in which case the memory usage will increase accordingly.

Since the computational complexity of the new algorithms is exponential, we do not expect them to be efficient for large inputs. Nevertheless, it would be useful to know how far they can be pushed. We therefore conducted several experiments to test their running times and memory usage for various inputs. Sections 7.1 and 7.2 below describe the experiments and the results, respectively. In short, the algorithms can be considered "practical" for $n \leq 16$.

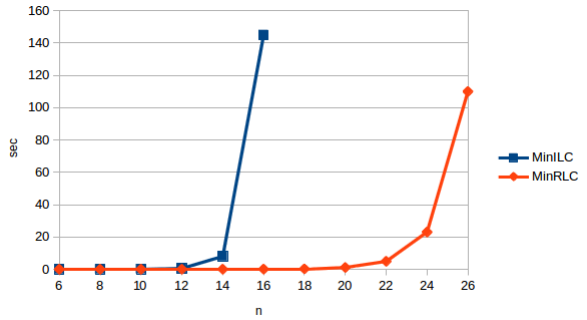
7.1. Experimental Setup

The experiments were done on a computer equipped with an Intel i7-2600 processor (3.4 GHz) and 8 Gb of memory, running Ubuntu 16.04. The source code was compiled using `g++`, version 5.4.0. Running times were measured using the `time` command, and the memory usage was measured using Valgrind [24] and its heap profiling tool Massif.

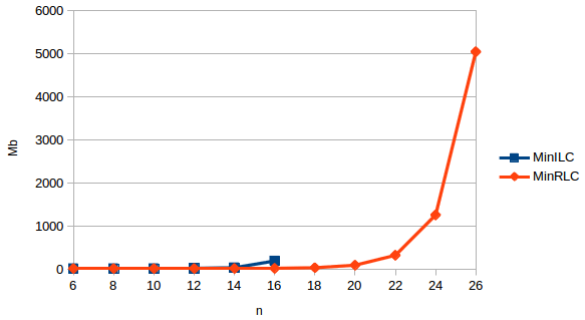
To generate simulated inputs, we used two different approaches:

- Approach 1 aimed at generating a set of *unrelated* trees. For each specified value of (k, n), we repeated the following procedure k times to obtain a set of k non-binary trees with leaf label set $\{1, 2, \ldots, n\}$: First, generate a binary tree in the *uniform model*.[§] Then, for each internal node, contract it with probability 0.2.
- Approach 2 aimed at generating a set of *related* trees. For each specified value of (k, n), we first generated a single master tree T_0 using the method described in Approach 1. We then created a set of k non-binary trees by doing the following k times: Take a copy of T_0 and for each non-root node v, with probability 0.1 move the subtree rooted at v by selecting a node u uniformly at random that is not a descendant of v and letting v become a child of u instead (nodes that end up with a single child are contracted to preserve the property that every internal node has at least two children).

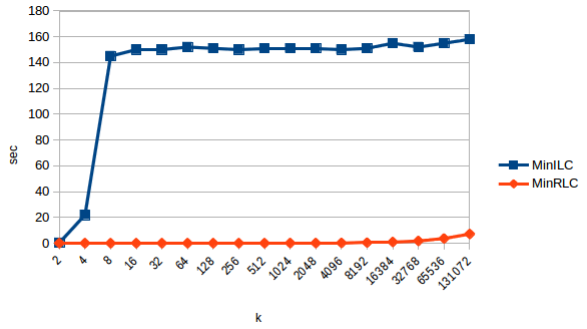
[§]The *uniform model* [23] starts with a tree consisting of a single leaf labeled by 1, and then for i from 2 to n, inserts a leaf labeled by i by selecting an existing edge f in the tree uniformly at random (here, the root is regarded to be attached to an imaginary parent node via an edge that belongs to the tree as well), breaking f into two edges by inserting a new internal node x into f, and creating an edge from x to a leaf labeled by i.



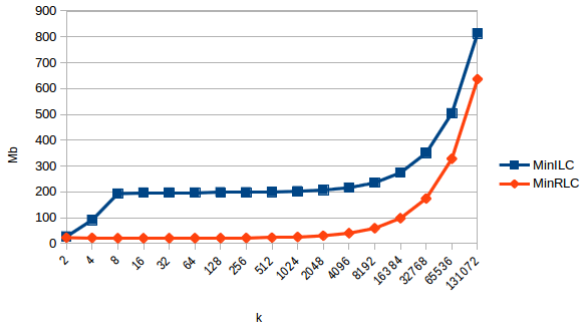
(a) Running time, unrelated trees, $k = 8$.



(b) Memory usage, unrelated trees, $k = 8$.



(c) Running time, unrelated trees, $n = 16$.



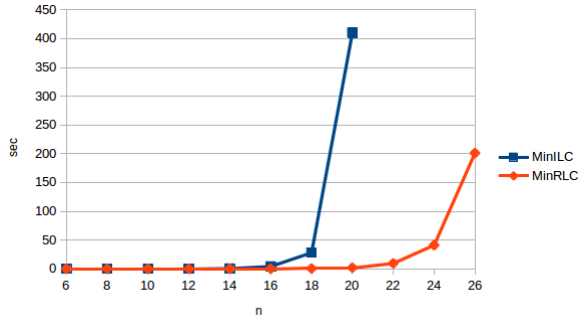
(d) Memory usage, unrelated trees, $n = 16$.

Figure 9. Plots of the average running times of `MinRLC_exact` and `MinILC_exact` in seconds (left) and their memory usage (right) as a function of increasing n (with fixed $k = 8$) and increasing k (with fixed $n = 16$) on unrelated trees.

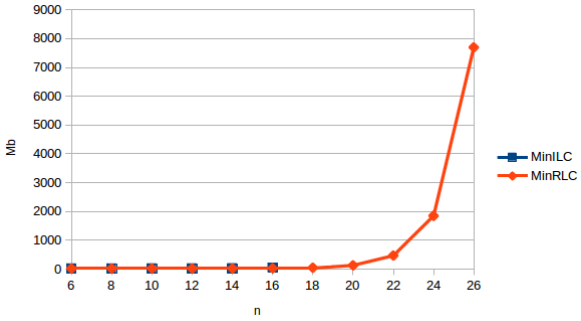
In the experiments, for various values of (k, n), we generated 50 independent inputs at random following Approach 1, and measured the average running times of Algorithms `MinRLC_exact` and `MinILC_exact` for these inputs. The average memory usage was also measured, but taken over 20 runs only because of the slowdown caused by Valgrind. The experiments were repeated for inputs generated according to Approach 2.

To test the algorithms on real datasets, we also downloaded a set of trees from the 10kTrees website [3], Primates, for varying k and n. In order to obtain trees with a specified value of n, we manually selected n species from the Colobinae subcategory. The 10kTrees website produces a number of trees between 10 and $10,000$ so we tested the algorithms on a more limited range of values of k here (i.e., $k \leq 10,000$). To obtain instances with $k = 8$, we generated 10 trees and subsequently removed two of them selected at random.

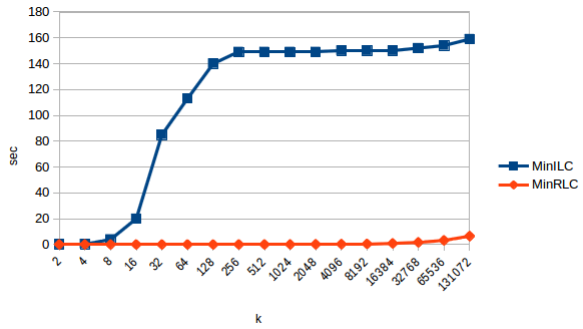
The results are reported in the next subsection.



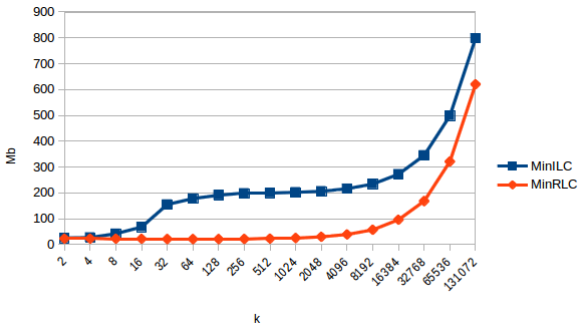
(a) Running time, related trees, $k = 8$.



(b) Memory usage, related trees, $k = 8$.

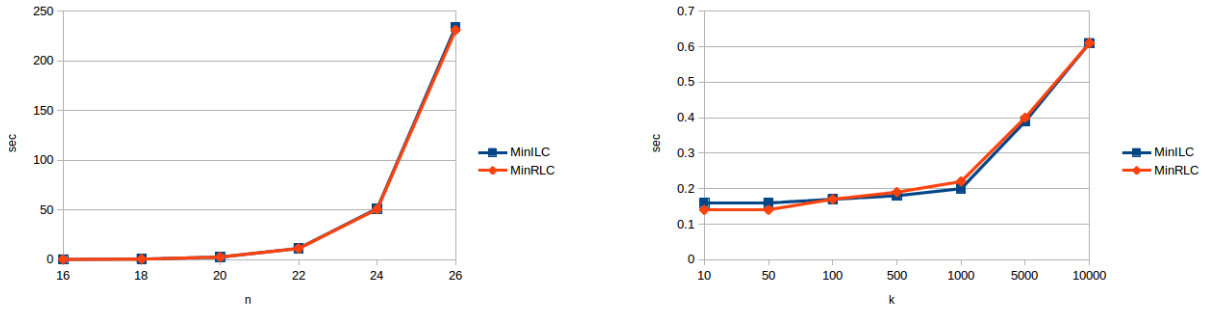


(c) Running time, related trees, $n = 16$.



(d) Memory usage, related trees, $n = 16$.

Figure 10. Plots of the average running times of `MinRLC_exact` and `MinILC_exact` in seconds (left) and their memory usage (right) as a function of increasing n (with fixed $k = 8$) and increasing k (with fixed $n = 16$) on related trees.



(a) 10kTrees website trees, $k = 8$. (b) 10kTrees website trees, $n = 16$.

Figure 11. Plots of the average running times of `MinRLC_exact` and `MinILC_exact` in seconds as a function of increasing n (with fixed $k = 8$) and increasing k (with fixed $n = 16$).

7.2. Experimental Results

The plots in figures 9 and 10 show how the average running times (in seconds) and the average memory usage (in Mb) of Algorithms `MinRLC_exact` and `MinILC_exact` increase for fixed k and increasing n, and for fixed n and increasing k. In figure 9, inputs were generated according to Approach 1 ("unrelated trees"), while in figure 10, they were generated according to Approach 2 ("related trees"). As a fixed value of k, we selected $k = 8$ because using a larger number of trees resulted in $\bigcap_{i=1}^{k} r(T_i)$ being almost empty most of the time, leading to a trivial scenario. As a fixed value of n, we selected $n = 16$ because it fell safely within the practical limit of our algorithms yet it was large enough to yield informative results. In general, `MinRLC_exact` is more efficient than `MinILC_exact`.

The behavior of `MinRLC_exact` is as one might expect. Its average running time is essentially independent of k, with a slight increase when k gets huge due to the added cost of parsing the input. Its average memory usage also appears to be linear in the size of the input. On the other hand, as n increases, the average running time follows a typical exponential-growth pattern.

The performance of `MinILC_exact` is more curious and worthy of some additional comments. Firstly, figures 9 (a), (b) and figures 10 (a), (b) show that its running time becomes an issue earlier than its memory usage does. (In fact, in figures 9 (b) and 10 (b), we had to end the experiments at $n = 16$ because $n = 18$ required hours to finish a single run.) Secondly, observe `MinILC_exacts`'s behavior in figure 9 (c) and figure 10 (c). In both cases, the average running time increases rapidly as more trees are added and stabilizes after a certain value. However, the average running time grows much faster for unrelated trees than related trees. In contrast, `MinRLC_exact` seems to do better on unrelated trees than related trees; compare, e.g., figure 9 (a) to figure 10 (a). Our hypothesis is that this is due to the different impact that the singleton components have on the two algorithms: `MinRLC_exact` is allowed to ignore singleton components but `MinILC_exact` cannot. When there are few shared rooted triplets, most components in the auxiliary graph will be singleton components, meaning that `MinRLC_exact` will perform very well while `MinILC_exact` will perform poorly as it has to consider a larger number of connected components. In other words, `MinRLC_exact` prefers inputs consisting of dissimilar trees whereas `MinILC_exact` prefers inputs where the trees are similar. This hypothesis is reinforced by the results for the trees from the 10kTrees website; these trees are expected to be very similar as they are highly correlated [3]. Figure 11 shows that in this case, the running times of `MinILC_exact` and `MinRLC_exact` are roughly the same.

		n									
		6	8	10	12	14	16	18	20	22	24
k	2	1.00	1.00	1.00	0.96	0.94	0.96	0.92	0.94	0.84	0.80
	4	1.00	1.00	0.98	0.96	0.90	0.94	0.86	0.74	0.72	0.68
	8	1.00	0.98	0.92	0.94	0.92	0.68	0.58	0.56	0.52	0.32
	16	1.00	1.00	0.98	0.90	0.74	0.92	0.68	0.30	0.46	0.30

Table 2. The suboptimality of the BUILD-based local consensus tree as a solution to MinRLC. For different values of k (rows) and n (columns), the table shows the number of times that the BUILD-based local consensus tree was an optimal solution to MinRLC divided by the number of cases tested.

8. An Experimental Investigation of the Non-Minimality of BUILD

In this section, we use the implementations from Section 7 to experimentally investigate whether applying the BUILD algorithm [2] to the set $\bigcap_{i=1}^{k} r(T_i)$, i.e., computing Bryant's BUILD-based local consensus tree [8], yields close approximations to MinRLC and MinILC in practice. This would be significant because it would provide a good polynomial-time heuristic for MinRLC and MinILC, which are NP-hard according to Theorems 3 and 4.

We generated sets of "related trees" of various sizes using Approach 2 in Section 7.1, and for each such input set, we extracted the shared rooted triplets and fed them to BUILD. We then counted the number of internal nodes and rooted triplets in BUILD's output as well as in the optimal solutions to MinRLC and MinILC, obtained by running our algorithms `MinRLC_exact` and `MinILC_exact`.

Tables 2 and 3 report the fraction of times that the solution reported by BUILD was optimal for MinRLC and MinILC, respectively, over 50 runs. Unfortunately, even for small values of n, BUILD often returns a non-optimal solution. For example, when $k = 4$ and $n = 12$, the BUILD-based solution was suboptimal compared to MinILC in more than half of the cases. Next, Tables 4 and 5 show *how* wrong BUILD usually is, which may be more relevant. On the positive side, our experiments indicate that the ratio between the number of internal nodes or the number of rooted triplets in the BUILD-based local consensus tree and in an optimal solution is fairly close to 1 on average; in the example above with $k = 4$ and $n = 12$, the BUILD-based solution was only 8% worse than the optimal solution to MinILC on average. However, the errors seem to increase with n and it may be the case that BUILD becomes less and less optimal as n grows larger, thus motivating the need for either faster optimal algorithms or better polynomial-time heuristics.

9. Concluding Remarks

The main open problem is to obtain faster exponential-time algorithms than the ones presented here. In particular, can MinRS be solved in $O^*(2^n)$ time?

According to the experiments in Section 7, the implemented algorithms have different bottlenecks. For MinRLC, the memory usage of our algorithm becomes problematic before the running time does, hence requiring a more memory-efficient solution, but for MinILC, developing a faster solution is more critical.

		n					
		6	8	10	12	14	16
k	2	0.94	0.88	0.74	0.72	0.50	0.62
	4	0.88	0.78	0.72	0.46	0.34	0.32
	8	0.94	0.78	0.78	0.72	0.56	0.58
	16	1.00	0.96	0.74	0.72	0.78	0.56

Table 3. The fraction of times that the BUILD-based local consensus tree was an optimal solution to MinILC, analogous to Table 2.

		n									
		6	8	10	12	14	16	18	20	22	24
k	2	1.00	1.00	1.00	1.01	1.01	1.00	1.01	1.00	1.01	1.01
	4	1.00	1.00	1.00	1.01	1.02	1.01	1.02	1.03	1.03	1.03
	8	1.00	1.01	1.02	1.03	1.02	1.07	1.09	1.10	1.10	1.16
	16	1.00	1 .00	1.01	1.03	1.08	1.03	1.15	1.24	1.19	1.32

Table 4. The average ratio between the number of internal nodes in the BUILD-based local consensus tree and in an optimal solution to MinRLC.

		n					
		6	8	10	12	14	16
k	2	1.00	1.01	1.02	1.01	1.04	1.02
	4	1.01	1.02	1.04	1.08	1.08	1.10
	8	1.01	1.04	1.03	1.07	1.09	1.13
	16	1.00	1.01	1.08	1.10	1.10	1.22

Table 5. The average ratio between the number of rooted triplets in the BUILD-based local consensus tree and in an optimal solution to MinILC, analogous to Table 4.

Another task is to extend the algorithms in this article to unrooted phylogenetic trees. This would be interesting because many existing methods for inferring phylogenetic trees produce unrooted trees [12]. The unrooted case appears to be much harder than the rooted case, as the basic problem of determining the consistency of an input set of rooted triplets is solvable in polynomial time (see Section 1.2), while the corresponding problem for *unrooted quartets* (unrooted, distinctly leaf-labeled trees with exactly four leaves each and in which every internal node has at least three neighbors) is already NP-hard [29].

Acknowledgments

The authors would like to thank Andrzej Lingas and Richard S. Lemence for some inspiring discussions. J.J. was partially funded by The Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1.Adams III EN (1972) Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology* 21: 390–397.

2.Aho AV, Sagiv Y, Szymanski TG, et al. (1981) Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM J Comput* 10: 405–421.

3.Arnold C, Matthews LJ, Nunn CL (2010) The 10kTrees website: A new online resource for primate phylogeny. *Evolutionary Anthropology* 19: 114–118.

4.Bender MA, Farach-Colton M (2000) The LCA problem revisited. In *Proceedings of the 4thLatin American Symposium on Theoretical Informatics* (LATIN 2000), volume 1776 of *LNCS*, pages 88–94. Springer-Verlag.

5.Bininda-Emonds ORP (2004) The evolution of supertrees. *TRENDS Ecol Evolution* 19: 315–322.

6.Bininda-Emonds ORP, Cardillo M, Jones KE, et al. (2007) The delayed rise of present-day mammals. *Nature* 446: 507–512.

7.Bryant D (1997) *Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis*. PhD thesis, University of Canterbury, Christchurch, New Zealand.

8.Bryant D (2003) A classification of consensus methods for phylogenetics. In Janowitz MF, Lapointe FJ, McMorris FR, et al., editors, *Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society.

9.Byrka J, Guillemot S, Jansson J (2010) New results on optimizing rooted triplets consistency. *Discrete Applied Mathematics* 158: 1136–1147.

10.Chor B, Hendy M, Penny D (2007) Analytic solutions for three taxon ML trees with variable rates across sites. *Discrete Applied Mathematics* 155: 750–758.

11.Constantinescu M, Sankoff D (1995) An efficient algorithm for supertrees. *J Classification* 12: 101–112.

12.Felsenstein J (2004) *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts.

13.Garey M, Johnson D (1979) *Computers and Intractability – A Guide to the Theory of NP-Completeness*. Freeman WH and Company, New York.

14.Gąsieniec L, Jansson J, Lingas A, et al. (1999) On the complexity of constructing evolutionary trees. *J Combinatorial Optimization* 3: 183–197.

15.He YJ, Huynh TND, Jansson J, et al. (2006) Inferring phylogenetic relationships avoiding forbidden rooted triplets. *J Bioinformatics Comput Bio* 4: 59–74.

16.Henzinger MR, King V, Warnow T. (1999) Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. *Algorithmica* 24: 1–13.

17.Huson DH, Rupp R, Scornavacca C (2010) *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, Cambridge, U.K.

18.Jansson J, Lemence RS, Lingas A (2012) The complexity of inferring a minimally resolved phylogenetic supertree. *SIAM J Comput* 41: 272–291.

19.Jansson J, Lingas A, Rajaby R, et al. (2017) Determining the consistency of resolved triplets and fan triplets. In *Proceedings of the 21st Annual International Conference on Research in Computational Molecular Biology* (RECOMB 2017), volume 10229 of *LNCS*, pages 82–98. Springer-Verlag.

20.Jansson J, Rajaby R, Shen C, et al. (to appear). Algorithms for the majority rule (+) consensus tree and the frequency difference consensus tree. *IEEE/ACM Transactions Computational Bio Bioinformatics*.

21.Jansson J, Shen C, Sung WK (2016) Improved algorithms for constructing consensus trees. *J ACM* 63.

22.Kannan S, Warnow T, Yooseph S (1998) Computing the local consensus of trees. *SIAM J Computing* 27: 1695–1724.

23.McKenzie A, Steel M (2000) Distributions of cherries for two models of trees. *Mathematical Biosciences* 164: 81–92.

24.Nethercote N, Seward J (2007) Valgrind: a framework for heavyweight dynamic binary instrumentation. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation* (PLDI 2007), pages 89–100. ACM.

25.Ng MP, Wormald NC (1996) Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics* 69: 19–31.

26.Semple C (2003) Reconstructing minimal rooted trees. *Discrete Applied Mathematics* 127: 489–503.

27.Semple C, Daniel P, Hordijk W, et al. (2004) Supertree algorithms for ancestral divergence dates and nested taxa. *Bioinformatics* 20: 2355–2360.

28.Snir S, Rao S (2006) Using Max Cut to enhance rooted trees consistency. *IEEE/ACM Transactions Comput Bio Bioinformatics* 3: 323–333.

29.Steel M (1992) The complexity of reconstructing trees from qualitative characters and subtrees. *J Classification* 9: 91–116.

30.Sung WK (2010) *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC, Boca Raton, Florida.

31.Willson SJ. (2004) Constructing rooted supertrees using distances. *Bulletin Mathematical Bio* 66: 1755–1783.

32.Wulff-Nilsen C (2013) Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2013), pages 1757–1769. SIAM.

AIMS Press