Algorithms for Reticulate Networks of Multiple Phylogenetic Trees

Zhi-Zhong Chen and Lusheng Wang

Abstract—A reticulate network N of multiple phylogenetic trees may have nodes with two or more parents (called *reticulation nodes*). There are two ways to define the *reticulation number* of N. One way is to define it as the number of reticulation nodes in N [13]; in this case, a reticulate network with the smallest reticulation number is called an *optimal type-I reticulate network* of the trees. The better way is to define it as the total number of parents of reticulation nodes in N minus the number of reticulation nodes in N [18]; in this case, a reticulate network with the smallest reticulation number is called an *optimal type-II reticulate network* of the trees. In this paper, we first present a fast fixed-parameter algorithm for constructing one or all optimal type-I reticulate networks of multiple phylogenetic trees. We then use the algorithm together with other ideas to obtain an algorithm for estimating a lower bound on the reticulation number of an optimal type-II reticulate network of the input trees. To our knowledge, these are the first fixed-parameter algorithms for the problems. We have implemented the algorithms in ANSI C, obtaining programs *CMPT* and *MaafB*. Our experimental data show that *CMPT* can construct optimal type-I reticulate networks rapidly and *MaafB* can compute better lower bounds for optimal type-II reticulate networks within shorter time than the previously best program *PIRN* designed by Wu [18].

Index Terms—Phylogenetic trees, reticulate networks, lower bounds of reticulate numbers.

1 Introduction

When studying the evolutionary history of a set of existing species, one can obtain a phylogenetic tree of the species with high confidence by looking at a segment of sequences or a set of genes. When looking at another segment of sequences, a different phylogenetic tree can be obtained with high confidence, too. This indicates that reticulation events may occur. When reticulation events occur, the evolutionary history of a set of existing species can be represented by a reticulate network in which there may exist nodes with two or more parents (called *reticulation nodes*).

Thus, we have the following problem: given a set of rooted phylogenetic trees on a set of species that correctly represent the evolution of different parts of their genomes, we want to construct a reticulate network with the smallest number of reticulation events needed to explain the evolution of the species under consideration.

There are two ways to define the *reticulation number* of a reticulate network N. One way is to define it as the number of reticulation nodes in N [13]; in this case, a reticulate network with the smallest reticulation number is called an *optimal type-I reticulate network*. The other is to define it as the total number of parents of reticulation nodes in N minus the number of reticulation nodes in N [18]; in this case, a reticulate network with the smallest reticulation number is called an *optimal type-II reticulate network*. The two definitions coincide in the special case when the number of given phylogenetic trees is 2. This special case has been studied extensively in the literature [5], [6], [9], [15], [17]. It is known that even this special case is NP-hard [2], [3], [8].

It is worth mentioning that optimal type-II reticulate networks are obviously a much better model for explaining reticulation events than optimal type-I reticulate networks. Nevertheless, optimal type-I reticulate networks are interesting to us because, as will be shown in this paper, it is much easier to construct them and their reticulation numbers can be used to obtain lower bounds on those of optimal type-II reticulate networks.

In this paper, we design fast fixed-parameter algorithms for constructing optimal type-I reticulate networks of multiple phylogenetic trees and for estimating lower bounds on the reticulation numbers of optimal type-II reticulate networks of multiple phylogenetic trees. It is widely known that reticulate networks are closely related to acyclic agreement forests (AAFs). Indeed, one can obtain an acyclic agreement forest of a set of phylogenetic trees from a reticulate network of the trees by deleting all the edges entering the reticulation nodes in the network. Moreover, a maximum acyclic agreement forest (MAAF) of a set of phylogenetic trees corresponds to an optimal type-I reticulate network of the trees. So, we first design a fixed-parameter algorithm for computing an MAAF of two or more given phylogenetic trees. The algorithm runs very fast in practice. After finding an MAAF of the given trees, our algorithm can then easily construct an optimal type-I reticulate network. Within the same time bound, our algorithm can also enumerate all MAAFs of the given trees and construct one optimal type-I reticulate network for each MAAF. To our knowledge, this is the first fixed-parameter algorithm for the problem. We have implemented the algorithm in ANSI C, obtaining a program (called *CMPT*) that can construct optimal type-I reticulate networks rapidly.

- *Z.-Z. Chen is with the Division of Information System Design, Tokyo Denki University, Ishizaka, Hatoyama, Saitama 359-0394, Japan. E-mail: zzchen@mail.dendai.ac.jp.*
- *L. Wang is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong. E-mail: cswangl@cityu.edu.hk.*

Manuscript received 11 Oct. 2010; revised 10 Aug. 2011; accepted 26 Sept. 2011; published online 17 Oct. 2011.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2010-10-0234.
Digital Object Identifier no. 10.1109/TCBB.2011.137.

We further use our algorithm (for enumerating all MAAFs of a set of given phylogenetic trees) together with other ideas to obtain an algorithm for estimating a lower bound on the reticulation number of an optimal type-II reticulate network of the given trees. We have implemented the algorithm in ANSI C, obtaining a program (called *MaafB*) that can compute better lower bounds within shorter time than the previously best program *PIRN* by Wu [18].

The programs *CMPT* and *MaafB* are available for noncommercial use, at http://rnc.r.dendai.ac.jp/~chen/mtree/mtree.html or http://www.cs.cityu.edu.hk/~lwang/software/mtree/mtree.html.

2 Preliminaries

Throughout this paper, a *rooted forest* always means a directed acyclic graph in which every node has in-degree at most 1 and out-degree at most 2.

Let F be a rooted forest. F is a *rooted tree* if it has only one root. F is a *rooted binary tree* if it is a rooted tree and the out-degree of every nonleaf node in F is 2. For convenience, we view each node u of F as an ancestor and descendant of u itself. A node u is *lower than* another node $v \neq u$ in F if u is a descendant of v in F. The *lowest common ancestor* (LCA) of a set U of nodes in F is the lowest node v in F such that for every node $u \in U$, v is an ancestor of u in F.

A node v of F is *unifurcate* if it has only one child in F. If a root v of F is unifurcate, then *contracting v in F* is the operation that modifies F by deleting v. If a nonroot node v of F is unifurcate, then *contracting v in F* is the operation that modifies F by first adding an edge from the parent of v to the child of v and then deleting v.

For a node v of F, the *subtree of F rooted at v* is the subgraph of F whose nodes are the descendants of v in F and whose edges are those edges connecting two descendants of v in F. If v is a root of F, then the subtree of F rooted at v is a *component tree* of F. If v is a nonroot node of F with parent p and sibling u, then *detaching the subtree of F rooted at v* is the operation that modifies F by first deleting the edge (p, v) and then contracting p. A *detaching operation* on F is the operation of detaching the subtree of F rooted at a nonroot node.

The *disconnectivity* of F, denoted by $|F|$, is the number of component trees in F minus 1. For convenience, we use $\mathcal{L}(F)$ to denote the family of all sets S such that S is the leaf set of a component tree of F.

Hereafter, we will use F (subscripted or not) to denote a rooted forest, use Γ (subscripted or not) to denote a component tree of a rooted forest, and use T (subscripted or not) to denote a rooted binary tree.

2.1 Phylogenetic Trees

Let X be a set of existing species. A *phylogenetic tree* on X is a rooted binary tree whose leaf set is X. Let T be a phylogenetic tree on X. For a subset Y of X, let T_Y denote the smallest subtree of T whose leaf set is Y, and $\overline{T_Y}$ be the phylogenetic tree on Y obtained from T_Y by repeatedly contracting a unifurcate node until none exists. If we start with T and apply a sequence of m detaching operations on T, we obtain a forest F with $|F| = m$. Note that $\mathcal{L}(F)$ is a partition of X. Moreover, for each set Y in $\mathcal{L}(F)$, $\overline{T_Y}$ is a component tree of F. Thus, we can identify F with $\mathcal{L}(F)$. The next fact is trivial.

Fact 1. *For every forest F obtained by performing zero or more detaching operations on T, every two sets Y and Z in $\mathcal{L}(F)$ satisfy that T_Y and T_Z are node disjoint.*

We say that a partition $\mathcal{P}$ of X is *valid* for T if we can perform zero or more detaching operations on T to obtain a forest F such that $\mathcal{P}$ is the same as $\mathcal{L}(F)$. Roughly speaking, the next fact states that the reverse of Fact 1 is also true.

Fact 2. *Suppose that $\mathcal{P}$ is a partition of X such that for every two sets Y and Z in $\mathcal{P}$, T_Y and T_Z are node disjoint. Then, $\mathcal{P}$ is valid for T.*

Let F_1 and F_2 be two forests each obtained by performing zero or more detaching operations on T. If $F_1 \neq F_2$ and for every set $Y_1 \in \mathcal{L}(F_1)$, there is a set $Y_2 \in \mathcal{L}(F_2)$ with $Y_1 \subseteq Y_2$, then we say that F_1 is *finer than* F_2 and F_2 is *coarser than* F_1. From Facts 1 and 2, one can easily see the next fact.

Fact 3. *Suppose that F_1 and F_2 are two rooted forests such that 1) each of F_1 and F_2 is obtained by performing zero or more detaching operations on T and 2) F_2 is finer than F_1. Then, we can obtain F_2 from F_1 by performing $|F_2| - |F_1|$ detaching operations.*

Roughly speaking, Fact 3 states that if a forest F is obtained from T by performing two or more detaching operations, then the order of performing the operations is not important.

2.2 Reticulate Networks

Let X be a set of existing species. A *reticulate network* on X is a directed acyclic graph N in which the set of nodes of out-degree 0 (still called the *leaves*) is X, each nonleaf node has out-degree 2, there is exactly one node of in-degree 0 (called the *root*), and each nonroot node has in-degree larger than 0. Note that the in-degree of a nonroot node in N may be larger than 1. A node of in-degree larger than 1 in N is called a *reticulation node* of N. Intuitively speaking, a reticulation node corresponds to a reticulation event. The *reticulation number* of a reticulation node in N is its in-degree in N minus 1. There are two ways to define the *reticulation number* of N. One way is to define it as the number of reticulation nodes in N and the other way is to define it as the total reticulation number of reticulation nodes in N. For convenience, we use $R_1(N)$ (resp., $R_2(N)$) to denote the reticulation number of N defined in the first (resp., second) way.

A reticulate network N on X *displays* a phylogenetic tree T on X if N has a subgraph M such that M is a rooted tree, the root of M has exactly two children in M, and modifying M by contracting its unifurcate nodes yields T. We refer to M as an *embedding* of T in N. For example, the network N in Fig. 4 displays the tree T_4 in Fig. 1. A *reticulate network* of two or more phylogenetic trees $T_1, \ldots, T_k$ on X is a reticulate network N on X such that N displays all of $T_1, \ldots, T_k$. For example, the network N in Fig. 4 is a reticulate network of the four trees $T_1, \ldots, T_4$ in Fig. 1. For convenience, we hereafter use $R_1(T_1, \ldots, T_k)$ (resp., $R_2(T_1, \ldots, T_k)$) to denote $\min_N R_1(N)$ (resp., $\min_N R_2(N)$), where N ranges over all reticulate networks of $T_1, \ldots, T_k$.

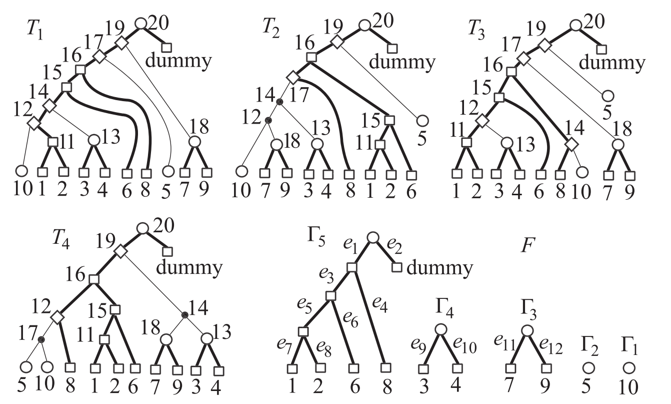

Fig. 1. Four phylogenetic trees $T_1, \dots, T_4$ and their MAAF F, where preserved nodes that are roots of F are emphasized with hollow circles, other preserved nodes are emphasized with rectangles, contracted nodes are emphasized with diamonds, and dangling nodes are emphasized with small filled circles.

An *optimal type-I* (resp., *type-II*) *reticulate network* of $T_1, \dots, T_k$ is a reticulate network N of $T_1, \dots, T_k$ such that $R_1(N) = R_1(T_1, \dots, T_k)$ (resp., $R_2(N) = R_2(T_1, \dots, T_k)$).

We are now ready to define the *general minimum type-I* (resp., *type-II*) *reticulate network problem*:

- **Input.** Two or more phylogenetic trees $T_1, \dots, T_k$ on the same set X of species.
- **Output.** An optimal type-I (resp., type-II) reticulate network of $T_1, \dots, T_k$.

As in [18], when we consider the general minimum type-I or II reticulate network problem, we always assume that each given phylogenetic tree has been modified by first introducing a new root and a dummy leaf and then letting the old root and the dummy leaf be the children of the new root.

2.3 Agreement Forests (AFs)

Throughout this section, let $T_1, \dots, T_k$ be two or more phylogenetic trees on the same set X of species. If we can apply a sequence of m detaching operations on each of $T_1, \dots, T_k$ so that they become the same forest F, then we refer to F as an *agreement forest* of $T_1, \dots, T_k$. A *maximum agreement forest* (MAF) of $T_1, \dots, T_k$ is an agreement forest of $T_1, \dots, T_k$ whose disconnectivity is minimized over all agreement forests of $T_1, \dots, T_k$.

Let F be an agreement forest of $T_1, \dots, T_k$. Obviously, for each $i \in \{1, \dots, k\}$, the leaves of T_i one-to-one correspond to the leaves of F. For convenience, we hereafter identify each leaf v of F with the leaf of T_i corresponding to v. Similarly, for each $i \in \{1, \dots, k\}$, the nonleaf nodes of F correspond to distinct nonleaf nodes of T_i. More precisely, a nonleaf node u of F corresponds to the LCA of $\{v_1, \dots, v_\ell\}$ in T_i, where $v_1, \dots, v_\ell$ are the leaf descendants of u in F. Again for convenience, we hereafter identify each nonleaf node u of F with the nonleaf node of T_i corresponding to u. With these correspondences, we can use F, $T_1, \dots, T_k$ to construct a directed graph G_F as follows:

- The nodes of G_F are the roots of F.
- For every two roots r_1 and r_2 of F, there is an edge from r_1 to r_2 in G_F if and only if there is an $i \in \{1, \dots, k\}$ such that r_1 is an ancestor of r_2 in T_i.

We refer to G_F as the *decision graph associated with* F. If G_F is acyclic, then F is an *acyclic agreement forest* of $T_1, \dots, T_k$; otherwise, F is a *cyclic agreement forest* (CAF) of $T_1, \dots, T_k$. If F is an AAF of $T_1, \dots, T_k$ and its disconnectivity is minimized over all AAFs of $T_1, \dots, T_k$, then F is a *maximum acyclic agreement forest* of $T_1, \dots, T_k$. Note that our definition of an AAF is the same as those in [4], [18] but is different from that in [16].

Fig. 1 depicts four example phylogenetic trees $T_1, \dots, T_4$ and an MAAF F of the trees. The component trees of F are $\Gamma_1, \dots, \Gamma_5$.

With respect to an AF F of $T_1, \dots, T_k$, we classify the nodes of each T_i with $1 \le i \le k$ into three types: *preserved nodes*, *contracted nodes*, and *dangling nodes*. A node of T_i is *preserved* with respect to F if it is also a node in F. A node y of T_i is *contracted* with respect to F, if it is not preserved with respect to F and there is an edge (u, v) in F such that the path from u to v in T_i contains y. For convenience, we refer to (u, v) as the *supporting edge* of y in F. For example, in Fig. 1, if y is the contracted node 14 in T_1, then its supporting edge in F is e_5. A node of T_i is *dangling* with respect to F if it is neither preserved nor contracted with respect to F. For example, in Fig. 1, nodes 12 and 14 of T_2 are dangling with respect to F, and so are nodes 14 and 17 of T_4.

Lemma 4. *Suppose that C is a cycle of G_F and $r_1, \dots, r_\ell$ are the nodes of C. Then, each $r_j \in \{r_1, \dots, r_\ell\}$ has two children u_j and u'_j in F. Moreover, for every nonroot node v of F not contained in $\{u_1, u'_1, \dots, u_\ell, u'_\ell\}$, C remains to be a cycle in G_F after F is modified by detaching the subtree of F rooted at v.*

Proof. The lemma is almost obvious and its proof is given in Section 1 of the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TCBB.2011.137. □

Lemma 5. *The dummy leaf alone does not form a component tree of an MAAF of $T_1, \dots, T_k$.*

Proof. Roughly speaking, if the dummy leaf alone formed a component tree of an MAAF F of $T_1, \dots, T_k$, then we would be able to merge this component with another component to obtain an AAF F' of $T_1, \dots, T_k$ with $|F'| < |F|$. The details are given in Section 1, available in the online supplemental material. □

By Lemma 5, the root of each T_i is a preserved node with respect to every AAF of $T_1, \dots, T_k$.

3 Computing MAFs, MAAFs, and Optimal Type-I Networks

In this section, we first describe how to compute all MAAFs of two or more given phylogenetic trees. We then explain how to contruct an optimal type-I network of two or more given phylogenetic trees from an MAAF of the trees. We omit the details of computing one MAAF, one MAF, or all MAFs, because they are similar to the computation of all MAAFs.

Whidden et al. [16] give an algorithm for computing an MAF of two given phylogenetic trees T_1 and T_2 on the same set X of species in $O(3^{d'}|X|)$ time, where d' is the disconnectivity of an MAF of T_1 and T_2. The basic idea behind their algorithm is as follows: initially, we set $F_1 = T_1$ and $F_2 = T_2$. We then repeatedly modify F_1 and F_2 (until F_1 becomes an AF of T_1 and T_2) as follows: we find two arbitrary sibling leaves u and v in F_2. If u and v are also siblings in F_1, then we modify F_1 and F_2 by introducing a new species x and replacing the identical subtrees of F_1 and F_2 rooted at the parent of u and v each with a single leaf labeled x. The point is that a detaching operation on the modified F_1 (resp., F_2) naturally corresponds to a detaching operation on the original F_1 (resp., F_2). On the other hand, if u and v are not siblings in F_1, then we have two cases depending on whether u and v are in the same component tree of F_1 or not. First, consider the case when u and v are in different component trees of F_1. In this case, in order to transform F_1 and F_2 into an AF of T_1 and T_2, we have two choices to modify them. One choice is to detach u from both F_1 and F_2 and the other is to detach v from both F_1 and F_2. Next, consider the case when u and v are in the same component trees of F_1. In this case, in order to transform F_1 and F_2 into an AF of T_1 and T_2, we have three choices to modify them. The first choice is to detach u from both F_1 and F_2. The second choice is to detach v from both F_1 and F_2. The third choice is to detach all those subtrees H from F_1 such that the root w of H does not appear in the (not necessarily directed) path between u and v in F_1 but the parent of w in F_1 does. Note that we always have $|F_1| \geq |F_2|$.

It is easy to modify the algorithm so that instead of computing only one MAF, it enumerates all MAFs of T_1 and T_2 within the same time bound. The idea is to simply let the algorithm continue to find other MAFs of T_1 and T_2 even after it finds an MAF of T_1 and T_2. Indeed, Chen and Wang [5] have implemented the algorithm in C to obtain a program (called *HybridNet*) that can enumerate all MAFs of T_1 and T_2 rapidly.

Obviously, an MAAF of T_1 and T_2 is an AF of T_1 and T_2 but is not necessarily an MAF of T_1 and T_2. So, in order to enumerate all MAAFs of T_1 and T_2, it is not sufficient to enumerate all MAFs of T_1 and T_2 and test if each enumerated MAF is cyclic or not. To obtain an algorithm for enumerating all MAAFs of two (or more) phylogenetic trees, our idea is to first extend Whidden et al.'s algorithm so that it solves the following *generalized agreement forest* (GAF) problem:

- **Input.** (T_1, T_2, F_1, b), where T_1 and T_2 are two phylogenetic trees on the same set X, F_1 is a forest obtained from T_1 by performing zero or more detaching operations on T_1, and b is a nonnegative integer.
- **Output.** A sequence of AFs of T_1 and T_2 including all AFs F of T_1 and T_2 such that 1) F can be obtained by performing at most b detaching operations on F_1 (or equivalently, at most $|F_1| + b$ detaching operations on T_2) and 2) no AF of T_1 and T_2 is finer than F_1 and coarser than F.

Lemma 6. *There is an algorithm for the GAF problem which on input (T_1, T_2, F_1, b), runs in $O(3^b|X|)$ time and outputs at most 3^h AFs F of T_1 and T_2 with $|F| = |F_1| + h$ for every integer $0 \leq h \leq b$.*

Proof. The algorithm and its analysis are detailed in Section 2, available in the online supplemental material. □

For convenience, we refer to the algorithm for the GAF problem guaranteed by Lemma 6 as the *GAF algorithm*.

We note that if we change the output of the GAF problem to be all AFs F of T_1 and T_2 such that F can be obtained by performing at most b detaching operations on F_1, then we cannot have an $O(3^b|X|)$-time algorithm for the GAF problem. To see this, suppose that we have performed $b' < b$ detaching operations on F_1 so that it is already an AF of T_1 and T_2. Then, F_1 remains to be an AF of T_1 and T_2 even if we further perform one or more detaching operations on it. However, since F_1 has $2(|X| - |F_1| - 1)$ nonroot nodes, there are $2(|X| - |F_1| - 1)$ ways to perform just one more detaching operation on F_1. So, there can exist $O(|X|^{b-b'})$ ways to perform $b - b'$ more detaching operations on F_1.

Obviously, to enumerate all MAAFs of T_1 and T_2, it suffices to first set $b = 0$ and then proceed as follows:

1. Simulate the GAF algorithm on input (T_1, T_2, T_1, b). During the simulation, whenever an AF F of T_1 and T_2 is enumerated, perform one of the following steps depending on whether F is acyclic or not:
    a. If F is acyclic, output it.
    b. If F is cyclic, then output all AAFs F' of T_1 and T_2 such that F' can be obtained from F by performing $b - |F|$ detaching operations on F.
2. If at least one AAF of T_1 and T_2 was outputted in Step 1a or 1b, then stop; otherwise, increase b by 1 and go to Step 1.

Note that Step 1b is nontrivial. We here do not detail how to perform Step 1b, because later in Lemma 12, we will show how to solve the following more general problem: given a CAF F of two or more phylogenetic trees $T_1, \ldots, T_k$ together with a positive integer b less than or equal to the disconnectivity of an MAAF of $T_1, \ldots, T_k$, enumerate all AAFs F' of $T_1, \ldots, T_k$ such that F' can be obtained from F by performing $b - |F|$ detaching operations on F.

We next extend the above approach so that it works for multiple phylogenetic trees. Let $T_1, \ldots, T_k$ be two or more phylogenetic trees on the same set X of species. To compute all MAAFs of $T_1, \ldots, T_k$, our idea is roughly as follows: since we do not know how large the disconnectivity d of an MAAF of $T_1, \ldots, T_k$ is, we try $d = 0, 1, 2, \ldots$ (in this order). When trying $d = b$, we want to compute a sequence $\mathcal{S}$ of AFs of $T_1, \ldots, T_k$ including all AFs F of $T_1, \ldots, T_k$ such that

- F can be obtained by performing at most b detaching operations on T_1 (or equivalently, at most b detaching operations on each T_i with $2 \leq i \leq k$) and
- no AF of $T_1, \ldots, T_k$ is coarser than F.

Roughly speaking, we can compute $\mathcal{S}$ as follows: first, we simulate the GAF algorithm on input (T_1, T_2, T_1, b) to obtain a sequence $\mathcal{S}_1$ of AFs of T_1 and T_2. For each $F_1 \in \mathcal{S}_1$, we then simulate the GAF algorithm on input $(T_1, T_3, F_1, b - |F_1|)$ to obtain a sequence $\mathcal{S}_2(F_1)$ of AFs of T_1 and T_3. Let

```
Input:  $(F_1, i, b)$, where $F_1$ is a forest obtained by
        performing zero or more detaching operations on $T_1$,
        $i \in \{2, \ldots, k\}$, and $b$ is a nonnegative integer.
Output: A sequence of AFs of $T_1$, $T_i$, ..., $T_k$ including all
        AFs $F$ of $T_1$, $T_i$, ..., $T_k$ such that (1) $F$ can be
        obtained by performing at most $b$ detaching operations on
        $F_1$ (or equivalently, at most $|F_1| + b$ detaching
        operations on each of $T_i, \ldots, T_k$) and (2) no AF of
        $T_1$, $T_i$, ..., $T_k$ is finer than $F_1$ and coarser than $F$.
1. Initialize $j = i$.
2. While $j \leq k$ and $F_1$ is an AF of $T_1$ and $T_j$,
   increase $j$ by 1.
3. If $j > k$, then output $F_1$ and return.
4. If $b = 0$, then return.
5. Simulate the GAF algorithm on input $(T_1, T_j, F_1, b)$ until it
   finds an AF $F'$ of $T_1$ and $T_j$ or returns.
6. If the GAF algorithm returns, then return.
7. If the GAF algorithm finds an AF $F'$ of $T_1$ and $T_j$, then
   perform Step 7.1 or 7.2 depending on the value of $j$:
   7.1. If $j = k$, then output $F'$.
   7.2. If $j < k$, then recursively call the algorithm on
        input $(F', j+1, b-h)$, where $h$ is the number
        of detaching operations performed on $F_1$ to
        obtain $F'$.
8. Continue to simulate the GAF algorithm on input $(T_1, T_j, F_1, b)$
   until it finds the next AF $F'$ of $T_1$ and $T_j$ or returns.
9. Go to Step 6.
```

Fig. 2. A subroutine for enumerating AFs of $T_1, \ldots, T_k$.

$\mathcal{S}_2 = \bigcup_{F_1 \in \mathcal{S}_1} \mathcal{S}_2(F_1)$. Note that each $F \in \mathcal{S}_2$ is an AF of $T_1, \ldots, T_3$ with $|F| \leq b$. Proceeding in this way for $i = 3, 4, \ldots, k$, we simulate the GAF algorithm on input $(T_1, T_{i+1}, F_1, b - |F_1|)$ for each $F_1 \in \mathcal{S}_{i-1}$ to obtain a sequence $\mathcal{S}_i(F_1)$ of AFs of T_1 and T_{i+1}. Let $\mathcal{S}_i = \bigcup_{F_1 \in \mathcal{S}_{i-1}} \mathcal{S}_i(F_1)$. Note that each $F \in \mathcal{S}_i$ is an AF of $T_1, \ldots, T_{i+1}$ with $|F| \leq b$. The crucial point is that $\mathcal{S}_{k-1}$ is the required $\mathcal{S}$.

To make the above rough idea precise, we first modify the GAF algorithm into the subroutine in Fig. 2. Basically, $\mathcal{S}$ can be obtained by calling the subroutine on input $(T_1, 2, b)$.

Lemma 7. *The subroutine in Fig. 2 is correct.*

Proof. We prove the lemma by induction on $k - i$.

Basis. In the base case, $k = i$ and in turn the subroutine is correct by Lemma 6.

Inductive step. Assume that $k > i$. Obviously, each output of the subroutine is an AF of $T_1, T_i, \ldots, T_k$. Suppose that F is an AF of $T_1, T_i, \ldots, T_k$ such that 1) F can be obtained by performing at most b detaching operations on F_1 and 2) no AF of $T_1, T_i, \ldots, T_k$ is finer than F_1 and coarser than F. If F_1 is an AF of T_1 and T_i, then no AF of $T_1, T_{i+1}, \ldots, T_k$ is finer than F_1 and coarser than F and the output of the subroutine on input (F_1, i, b) is the same as the output of the subroutine on input $(F_1, i+1, b)$, implying that the subroutine can output F by the inductive hypothesis. So, assume that F_1 is not an AF of T_1 and T_i. There are two cases depending on whether T_1 and T_i have an AF that is finer than F_1 and coarser than F.

First consider the case where no AF of T_1 and T_i is finer than F_1 and coarser than F. In this case, by Lemma 6, the GAF algorithm on input (T_1, T_i, F_1, b) can enumerate F and hence there is a recursive call on input $(F, i+1, b-h)$ in Step 7.2 of the subroutine, where $h = |F| - |F_1|$. The recursive call on input $(F, i+1, b-h)$ will output F in Step 3.

Next consider the case where T_1 and T_i have an AF that is finer than F_1 and coarser than F. Let $\mathcal{A}$ be the set of all AFs of T_1 and T_i that are finer than F_1 and coarser than F. Among the AFs in $\mathcal{A}$, we can choose an F' such that for every $F'' \in \mathcal{A} - \{F'\}$, F' is not finer than F''. By the choice of F', no AF of T_1 and T_i is finer than F_1 and coarser than F'. So, by Fact 3, F' can be obtained from F_1 by performing $|F'| - |F_1|$ detaching operations. Thus, by Lemma 6, the GAF algorithm can enumerate F' on input (T_1, T_i, F_1, b) and hence there is a recursive call on input $(F', i+1, b-h)$ in Step 7.2 of the subroutine, where $h = |F'| - |F_1|$. Again, by Fact 3, F can be obtained from F' by performing $|F| - |F'| \leq b - h$ detaching operations. Moreover, no AF of $T_1, T_{i+1}, \ldots, T_k$ is finer than F' and coarser than F, because no AF of $T_1, T_i, \ldots, T_k$ is finer than F_1 and coarser than F. Thus, by the inductive hypothesis, the recursive call on input $(F', i+1, b-h)$ in Step 7.2 of the subroutine will output F. Therefore, the subroutine is correct. □

Lemma 8. *Given an input (F_1, i, b), the subroutine in Fig. 2 outputs at most 6^h AFs F of $T_1, T_i, \ldots, T_k$ with $|F| = |F_1| + h$ for every $0 \leq h \leq b$.*

Proof. By induction on b. The details are given in Section 1, available in the online supplemental material. □

Lemma 9. *Given an input $(F_1, 2, b)$, the subroutine in Fig. 2 outputs at most $6^h 2^{1-d'_{1,2}}$ AFs F of $T_1, \ldots, T_k$ with $|F| = h$ for every $d'_{1,2} \leq h \leq b$, where $d'_{1,2}$ is the disconnectivity of an MAF of T_1 and T_2.*

Proof. The point is that the analysis in the proof of Lemma 8 can be tightened when $i = 2$. The details are given in Section 1, available in the online supplemental material. □

Lemma 10. *Given an input (F_1, i, b), the subroutine in Fig. 2 takes $O((k - i + 1)6^b|X|)$ time.*

Proof. By induction on b. The details are given in Section 1, available in the online supplemental material. □

Lemma 11. *Given an input $(F_1, 2, b)$, the subroutine in Fig. 2 takes $O(k6^b 2^{-d'_{1,2}}|X|)$ time.*

Proof. The point is that the analysis in the proof of Lemma 10 can be tightened when $i = 2$. The details are given in Section 1, available in the online supplemental material. □

We next use the subroutine in Fig. 2 to solve the following *generalized acyclic agreement forest* (GAAF) problem:

```
Input: An instance $(T_1, \ldots, T_k, b)$ of the GAAF problem.
Output: All AAFs $F$ of $T_1, \ldots, T_k$ with $|F| = b$.
1. Simulate the subroutine in Figure 2 on input $(T_1, 2, b)$ until it finds an AF $F'$ of $T_1, \ldots, T_k$ or returns.
2. If the subroutine returns, then return.
3. If the subroutine finds an AF $F'$ of $T_1, \ldots, T_k$ such that $|F'| \le b$ and the dummy leaf is not a root in $F'$, then perform the following steps:
   3.1. Construct the decision graph $G_{F'}$ associated with $F'$.
   3.2. If $G_{F'}$ is acyclic, then output $F'$.
   3.3. If $G_{F'}$ is cyclic and $|F'| < b$, then output all AAFs $F''$ of $T_1, \ldots, T_k$ with $|F''| = b$ such that $F''$ is finer than $F'$.
4. Continue to simulate the subroutine on input $(T_1, 2, b)$ until it finds the next AF $F'$ of $T_1, \ldots, T_k$ or returns.
5. Go to Step 2.
```

Fig. 3. An algorithm for the GAAF problem.

- **Input.** $(T_1, \ldots, T_k, b)$, where $T_1, \ldots, T_k$ are two or more phylogenetic trees on the same set X and b is a lower bound on the disconnectivity of an MAAF of $T_1, \ldots, T_k$.
- **Output.** All AAFs F of $T_1, \ldots, T_k$ with $|F| = b$.

Roughly speaking, we can solve the GAAF problem as follows: given an input $(T_1, \ldots, T_k, b)$, we simulate the subroutine in Fig. 2 on input $(T_1, 2, b)$. During the simulation, whenever an AF F of $T_1, \ldots, T_k$ is enumerated, we check if F is acyclic or not. If F is acyclic, then we can simply output it. Otherwise, we check if the dummy leaf is a root of F or not. If the dummy leaf is a root of F, then by Lemma 5, we can simply discard F. Otherwise, we output all AAFs F' of $T_1, \ldots, T_k$ such that F' can be obtained from F by performing $b - |F|$ detaching operations on F.

Based on the above rough idea, we present an algorithm for the GAAF problem in Fig. 3. We refer to this algorithm as the *GAAF algorithm*. Note that in Step 3.3, we need to output all AAFs F'' of $T_1, \ldots, T_k$ with $|F''| = b$ such that F'' is finer than F'. Lemma 12 shows that we can do this in $O(b^{b-|F'|+2})$ time.

Lemma 12. *Suppose that for every node v of F' and every $i \in \{1, \ldots, k\}$, we have precomputed the node of T_i corresponding to v. Further assume that for every $i \in \{1, \ldots, k\}$, we have preprocessed T_i so that given a pair (u, v) of nodes of T_i, we can compute the LCA of $\{u, v\}$ in T_i in $O(1)$ time. Then, Step 3.3 of the GAAF algorithm takes $O((b-1)^{b-|F'|+2})$ time.*

Proof. The idea is to use Lemma 4. The details are given in Section 1, available in the online supplemental material. □

Lemma 13. *The GAAF algorithm is correct.*

Proof. Clearly, each output of the algorithm on input $(T_1, \ldots, T_k, b)$ is an AAF of $T_1, \ldots, T_k$ with disconnectivity b. Let F be an arbitrary AAF of $T_1, \ldots, T_k$ with $|F| = b$. If no AF of $T_1, \ldots, T_k$ is coarser than F, then the subroutine in Fig. 2 on input $(T_1, 2, b)$ can find F and so the algorithm can output F in Step 3.2. So, assume that some AF of $T_1, \ldots, T_k$ is coarser than F. Then, there must exist an AF F' of $T_1, \ldots, T_k$ such that F' is coarser than F and no AF of $T_1, \ldots, T_k$ is coarser than F'. Now, the subroutine in Fig. 2 on input $(T_1, 2, b)$ can find F' and output F in Step 3.3. Thus, the algorithm is correct. □

Lemma 14. *Let $(T_1, \ldots, T_k, b)$ be an input to the GAAF algorithm. Suppose that for every $i \in \{1, \ldots, k\}$, we have preprocessed T_i so that given a pair (u, v) of nodes of T_i, we can compute the LCA of $\{u, v\}$ in T_i in $O(1)$ time. Then, the GAAF algorithm on input $(T_1, \ldots, T_k, b)$ takes $O(k|X|6^b 2^{-d'_{1,2}} + (b-1)^{b-d'+2} 6^{d'} 2^{-d'_{1,2}})$ time, where d' is the disconnectivity of an MAF of $T_1, \ldots, T_k$ and $d'_{1,2}$ is the disconnectivity of an MAF of T_1 and T_2.*

Proof. The proof employs Lemmas 11 and 9. The details are given in Section 1, available in the online supplemental material. □

Now, to compute all MAAFs of $T_1, \ldots, T_k$, it suffices to perform the following steps 1 through 4:

1. Rearrange $T_1, \ldots, T_k$ so that the disconnectivity of an MAF of T_1 and T_2 is the maximum disconnectivity of an MAF of two trees among $T_1, \ldots, T_k$.
2. For every $i \in \{1, \ldots, k\}$, preprocess T_i so that given a pair (u, v) of nodes of T_i, we can compute the LCA of $\{u, v\}$ in T_i in $O(1)$ time.
3. Call the GAAF algorithm on input $(T_1, \ldots, T_k, d')$, where d' is the disconnectivity of an MAF of T_1 and T_2.
4. If at least one AAF is output in Step 3, then return; otherwise, increase d' by 1 and go to Step 3.

We are now ready to prove the following theorem:

Theorem 1. *Given two or more phylogenetic trees $T_1, \ldots, T_k$ on the same set X of species, we can compute all MAAFs of $T_1, \ldots, T_k$ in $O(k^2|X|3^{d''} + k|X|6^d 2^{-d''} + (d-1)^{d-d'+2} 6^{d'} 2^{-d''})$ time, where d (resp., d') is the disconnectivity of an MAAF (resp., MAF) of $T_1, \ldots, T_k$ and d'' is the maximum disconnectivity of an MAF of two trees among $T_1, \ldots, T_k$.*

Proof. By Lemma 13, performing Steps 1 through 4 in the above gives us all MAAFs of $T_1, \ldots, T_k$. We next estimate the time needed to perform the steps. Using Whidden et al.'s algorithm for computing an MAF of two given phylogenetic trees, we can perform Step 1 in $O(k^2 3^{d''}|X|)$ time. For each $i \in \{1, \ldots, k\}$, we can use the algorithm in [12] to preprocess T_i in $O(|X|)$ time so that given a pair (u, v) of nodes in T_i, we can compute the LCA of $\{u, v\}$ in T_i in $O(1)$ time. Thus, Step 2 takes $O(k|X|)$ total time for $T_1, \ldots, T_k$. Therefore, by Lemma 14, the four steps take $O(k^2|X|3^{d''} + k|X|6^d 2^{-d''} + (d-1)^{d-d'+2} 6^{d'} 2^{-d''})$ total time. □

Let F be an MAAF of $T_1, \ldots, T_k$. We want to use F to construct a reticulate network N of $T_1, \ldots, T_k$ with $R_1(N) = |F|$. In the special case where $k = 2$, there are known algorithms for this purpose [5], [14]. It is quite easy to modify these algorithms so that they work for our purpose even in the general case (i.e., the case where $k \geq 2$). In particular, Section 3, available in the online supplemental material, which is available online, describes how to modify the algorithm detailed in the supplementary material of [5]. If one wants to compute the extended Newick representation of N from F, Section 4, available in the online supplemental material, details how to do this.

It remains to show that a reticulate network N of $T_1, \ldots, T_k$ with $R_1(N) = |F|$ is an optimal type-I reticulate network of $T_1, \ldots, T_k$. To this end, it suffices to claim that for every reticulate network M of $T_1, \ldots, T_k$, $R_1(M) \geq |F|$. This claim is easy to prove. Indeed, it follows from the last inequality in Statement 2 in Lemma 15 immediately.

4 Better Lower Bound on $R_2(T_1, \ldots, T_k)$

Throughout this section, fix two or more phylogenetic trees $T_1, \ldots, T_k$ on the same set X of species. Our goal is to compute a lower bound on $R_2(T_1, \ldots, T_k)$.

Previously, a lower bound, called the *RH bound*, on $R_2(T_1, \ldots, T_k)$ was given by Wu [18]. Given $R_2(T_i, T_j)$ for all pairs (i, j) with $1 \leq i < j \leq k$, the RH bound on $R_2(T_1, \ldots, T_k)$ is inferred via integer linear programming. In this section, we will use our algorithm for enumerating all MAAFs of $T_1, \ldots, T_k$ to compute a lower bound on $R_2(T_1, \ldots, T_k)$ that is better than the RH bound in a significant number of cases.

Consider a reticulate network N of $T_1, \ldots, T_k$. Let F be the forest obtained from N by removing all edges entering the reticulation nodes of N. We refer to F as the *forest associated with* N and use $\mathcal{F}(N)$ to denote it. For each root v of $\mathcal{F}(N)$ that is not the root of N, the reticulation number of v in N is at least 1. Thus, $R_2(N) \geq |\mathcal{F}(N)|$, where we recall that $|\mathcal{F}(N)|$ denotes the disconnectivity of $\mathcal{F}(N)$. For an example, see Fig. 4.

Suppose that we obtain a forest F' by modifying $\mathcal{F}(N)$ by performing the next two steps:

- Step 1. Delete those nodes v such that neither v nor its descendants in $\mathcal{F}(N)$ are in X.
- Step 2. Contract all unifurcate nodes in $\mathcal{F}(N)$.

Obviously, F' is an AAF of $T_1, \ldots, T_k$. We refer to F' as the *AAF associated with* N and use $\mathcal{A}(N)$ to denote it. Each component tree Γ of $\mathcal{A}(N)$ is a modification of a component tree Γ' of $\mathcal{F}(N)$. We call Γ' the *tree in* $\mathcal{F}(N)$ *corresponding to* Γ. Fig. 4 shows an example, where the component tree of $\mathcal{F}(N)$ rooted at node 26 corresponds to the component tree of $\mathcal{A}(N)$ rooted at node 26.

$|\mathcal{A}(N)|$ may be smaller than $|\mathcal{F}(N)|$. This can happen only when at least one node is deleted in Step 1 in the above. If $|\mathcal{A}(N)| < |\mathcal{F}(N)|$, we say that N is *unusual*; otherwise, we say that N is *usual*. For example, the network N in Fig. 4 is unusual.

Lemma 15. *For every reticulate network N of $T_1, \ldots, T_k$, the following statements hold:*

1. *The roots of $\mathcal{F}(N)$ are exactly the reticulation nodes in N plus the root of N.*
2. $R_2(N) \geq |\mathcal{F}(N)| = R_1(N) \geq |\mathcal{A}(N)|$.

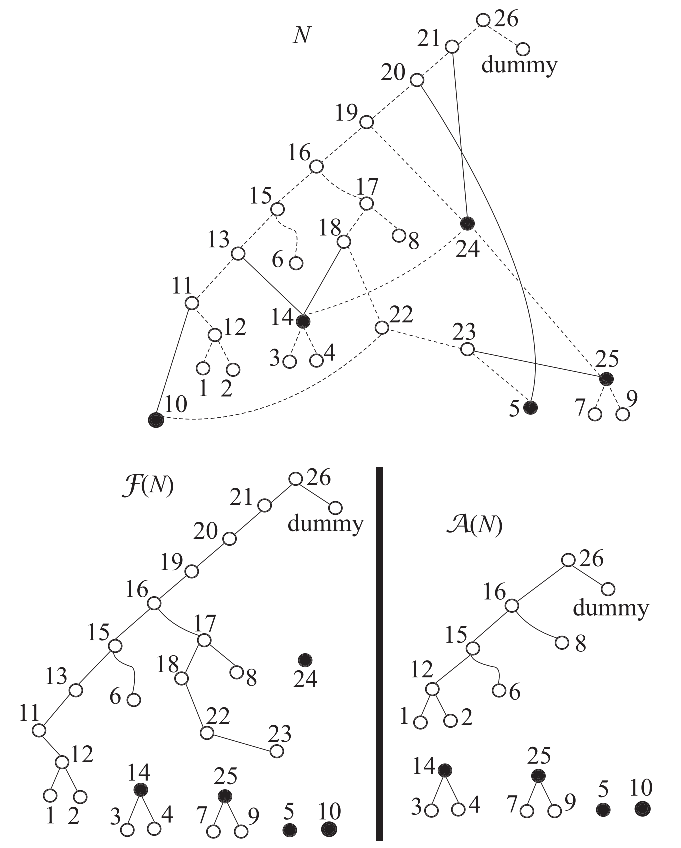

Fig. 4. A reticulate network N of the four trees $T_1, \ldots, T_4$ in Fig. 1, $\mathcal{F}(N)$, and $\mathcal{A}(N)$, where the broken edges in N show an embedding of T_4 in N.

3. *Let u, u', and v be three distinct nodes of $\mathcal{F}(N)$ such that u' is an ancestor of v in $\mathcal{F}(N)$ but u is not. Then, u' is a node of every path from u to v in N.*
4. *Let v be a node of a component tree Γ of $\mathcal{F}(N)$ and u be a node of N outside Γ. If there is a path P from u to v in N, then every ancestor u' of v in $\mathcal{F}(N)$ is a node of P.*
5. *Let u and v be two distinct nodes of $\mathcal{F}(N)$. If there is a path P from u to v in $\mathcal{F}(N)$, then P is the unique path from u to v in N.*
6. *Let u and v be two distinct nodes of a component tree Γ in $\mathcal{F}(N)$. If there is a path P from u to v in N, then u is an ancestor of v in Γ.*

Proof. Statement 1, the equality $R_1(N) = |\mathcal{F}(N)|$, and the inequality $|\mathcal{F}(N)| \geq |\mathcal{A}(N)|$ are clearly true. Inequality $R_2(N) \geq |\mathcal{F}(N)|$ holds because all roots of $\mathcal{F}(N)$ except one have in-degree at least 2 in N. Roughly speaking, Statements 3 through 6 hold because every path of N from a node outside a component tree Γ of $\mathcal{F}(N)$ to a node of Γ has to pass through the root of Γ. The detailed proofs are given in Section 1, available in the online supplemental material. □

By Statement 2 in Lemma 15, the disconnectivity of an MAAF of $T_1, \ldots, T_k$ is a lower bound on $R_2(T_1, \ldots, T_k)$. We refer to this bound as the *MAAF bound*. In the sequel, we show how to improve the MAAF bound by 1, obtaining a new bound called the *revised MAAF* (*rMAAF*) bound. Although the rMAAF bound can be larger than the MAAF

TABLE 1
Comparing the rMAAF and the RH Bounds on Simulated Data Sets from [18]

dataset			avg						rMaaf
$\lvert X\rvert$	r	k	Maaf	$+2$	$+1$	$=$	-1	-2	>Maaf
10	1	4	1.47	0%	3%	96%	1%	0%	17%
10	1	5	1.7	0%	4%	93%	3%	0%	23%
20	1	4	2.84	0%	7%	91%	2%	0%	30%
20	1	5	2.97	0%	6%	91%	3%	0%	32%
30	1	4	3.12	0%	3%	96%	1%	0%	19%
30	1	5	3.36	0%	4%	94%	2%	0%	28%
40	1	4	3.37	0%	5%	95%	0%	0%	33%
40	1	5	3.86	1%	4%	93%	2%	0%	37%
10	3	4	3.31	0%	8%	90%	2%	0%	64%
10	3	5	3.63	0%	13%	80%	7%	0%	63%
20	3	4	5.42	0%	16%	73%	11%	0%	64%
20	3	5	5.72	1%	18%	67%	14%	0%	72%
30	3	4	7.6	2%	23%	69%	5%	1%	65%
30	3	5	7.87	1%	20%	66%	13%	0%	84%
40	3	4	8.1	3%	29%	60%	8%	0%	73%
40	3	5	9.1	3%	23%	64%	10%	0%	76%
10	5	4	4.1	0%	13%	75%	12%	0%	65%
10	5	5	4.23	0%	8%	76%	16%	0%	66%
20	5	4	7.28	0%	20%	68%	12%	0%	85%
20	5	5	7.91	0%	20%	62%	18%	0%	86%
30	5	4	9.84	4%	26%	49%	20%	1%	88%
30	5	5	10.5	4%	28%	56%	11%	1%	92%
40	5	4	11.55	4%	29%	43%	22%	2%	87%
40	5	5	12.16	3%	25%	50%	20%	2%	96%

Column "$\lvert X\rvert$" shows the number of leaves in one input tree, column "r" shows the reticulation level, column "k" shows the number of trees, column "avg Maaf" shows the average MAAF bound, column "+2" (respectively, "+1," "= ," "–1," or "–2") shows the percentage of data sets for which the rMAAF bound is larger than the RH bound by 2 (respectively, 1, 0, -1, or -2), and column "rMaaf> Maaf" shows the percentage of data sets for which the rMAAF bound is larger than the MAAF bound.

TABLE 2
Comparing the rMAAF and the RH Bounds on Other Simulated Data Sets

dataset		avg	avg				max	rMaaf
k	m	rMaaf	RH	$+$	$=$	$-$	gap	>Maaf
7	1	13.65	10.35	100%	0%	0%	5	100%
7	2	12.95	12.8	15%	60%	25%	1	100%
10	1	15.35	10.1	100%	0%	0%	6	100%
10	2	16.25	16.15	10%	35%	55%	1	100%
15	1	16.6	10.1	100%	0%	0%	8	100%
15	2	16.7	16.9	30.5%	20%	49.5%	2	100%

Column "k" shows the number of trees, column "m" shows the method used to generate the trees, column "avg rMaaf" shows the average rMAAF bound, column "avg RH" shows the average RH bound outputted by PIRN within 1 hour, column "+" (respectively, "= " or "–") shows the percentage of data sets for which the rMAAF bound is larger than (respectively, equal to or smaller than) the RH bound, and column "max gap" shows the maximum gap between the rMAAF bound and the RH bound found by PIRN within 1 hour.

bound by only 1, we are interested in computing it for several reasons. First, the rMAAF bound can be computed almost as fast as the MAAF bound. Second, our experimental data show that the rMAAF bound is usually larger than the MAAF bound (see Tables 1 and 2). Third, our experimental data also show that the rMAAF bound is better than the *RH* bound in a significant number of cases (see Tables 1 and 2). Fourth, unlike the MAAF bound, the rMAAF bound is nontrivial and studying it gives us some insight into optimal type-II reticulate networks and hence may eventually lead to better lower bounds in the future.

Throughout the remainder of this section, let m be the MAAF bound on $R_2(T_1, \dots, T_k)$. We want to figure out when $R_2(T_1, \dots, T_k) \geq m + 1$.

There are two simple cases where $R_2(T_1, \dots, T_k) \geq m + 1$. In the case where at least one optimal type-II reticulate network N of $T_1, \dots, T_k$ is unusual, we have $R_2(T_1, \dots, T_k) \geq m + 1$. This is true because $R_2(T_1, \dots, T_k) = R_2(N) > |\mathcal{A}(N)| \geq m$, where the first inequality holds because N is unusual. Moreover, in the case where there is an optimal type-II reticulate network N of $T_1, \dots, T_k$ such that $\mathcal{A}(N)$ is not an MAAF of $T_1, \dots, T_k$, we have $R_2(T_1, \dots, T_k) \geq m + 1$. This is true because $R_2(T_1, \dots, T_k) = R_2(N) \geq |\mathcal{A}(N)| > m$. Thus, to figure out when $R_2(T_1, \dots, T_k) \geq m + 1$, we can concentrate on those optimal type-II reticulate networks N of $T_1, \dots, T_k$ such that N is usual and $\mathcal{A}(N)$ is an MAAF of $T_1, \dots, T_k$. We refer to such an N as a *doubly optimal* reticulate network of $T_1, \dots, T_k$. The next fact justifies this naming.

Fact 16. *Suppose that N is a reticulation network of $T_1, \dots, T_k$. Then, N is an optimal type-I reticulate network of $T_1, \dots, T_k$ if and only if N is usual and $\mathcal{A}(N)$ is an MAAF of $T_1, \dots, T_k$.*

Proof. The proof is quite easy and is detailed in Section 1, available in the online supplemental material. □

Lemma 17. *For every optimal type-II reticulate network N of $T_1, \dots, T_k$ such that $\mathcal{F}(N)$ has at most $m + 1$ component trees, the following statements hold:*

1. *$\mathcal{F}(N)$ has $m + 1$ component trees and so does $\mathcal{A}(N)$.*
2. *$\mathcal{A}(N)$ is an MAAF of $T_1, \dots, T_k$.*
3. *N is doubly optimal.*

Proof. The lemma is almost obvious from Statement 2 in Lemma 15. The proof is detailed in Section 1, available in the online supplemental material. □

We say that an MAAF F of $T_1, \dots, T_k$ is *good* if for every doubly optimal reticulate network N of $T_1, \dots, T_k$ such that $\mathcal{A}(N)$ is the same as F, $R_2(N) \geq m + 1$.

Theorem 2. *Assume that every MAAF of $T_1, \dots, T_k$ is good. Then, $R_2(T_1, \dots, T_k) \geq m + 1$.*

Proof. If there is an optimal type-II reticulate network N of $T_1, \dots, T_k$ such that $\mathcal{F}(N)$ has at least $m + 2$ component trees, then $R_2(T_1, \dots, T_k) \geq m + 1$. Otherwise, we can use Statements 2 and 3 in Lemma 17 to show that $R_2(T_1, \dots, T_k) \geq m + 1$. The details are given in Section 1, available in the online supplemental material. □

Based on Theorem 2, we will design an algorithm that enumerates all MAAFs of $T_1, \dots, T_k$ and checks if each of them is good. Moreover, if the algorithm finds out that each MAAF of $T_1, \dots, T_k$ is good, then it outputs $m + 1$ as a lower bound on $R_2(T_1, \dots, T_k)$; otherwise, it outputs m as a lower bound.

Throughout the remainder of this section, fix an MAAF F of $T_1, \dots, T_k$. See Fig. 1 for an example. Based on F, we define the following notations:

- Let $\Gamma_1, \dots, \Gamma_{m+1}$ denote the component trees of F. Without loss of generality, we assume that Γ_{m+1} contains the dummy leaf.

- For each $j \in \{1, \ldots, m+1\}$, let r_j denote the root of Γ_j. By Lemma 5, r_{m+1} is also the root of T_i for each $i \in \{1, \ldots, k\}$.
- For each $1 \leq j \leq m$ and each $1 \leq i \leq k$, let $p_{j,i}$ denote the lowest ancestor of r_j in T_i that is a contracted node, and let $e_{j,i}$ denote the supporting edge of $p_{j,i}$ in F. For example, in Fig. 1, $p_{1,1} = 12$, $e_{1,1} = e_5$, $p_{3,2} = 17$, and $e_{3,2} = e_4$. Note that each inner node of the path from $p_{j,i}$ to r_j in T_i is a dangling node.

We want an easily checkable necessary-and-sufficient condition for F to be good. Unfortunately, we are unable to find such a condition. In the following, we give easily checkable sufficient conditions for F to be good.

By a *reticulate F-network*, we mean a reticulate network N of $T_1, \ldots, T_k$ such that $\mathcal{A}(N)$ is the same as F. Consider an arbitrary doubly optimal reticulate F-network N. Since N is usual, $\mathcal{F}(N)$ has exactly $m+1$ roots. One root of $\mathcal{F}(N)$ has in-degree 0 in N, while each other root of $\mathcal{F}(N)$ has in-degree at least 2 in N. So, if $\mathcal{F}(N)$ has a root whose in-degree in N is at least 3, then $R_2(N) \geq m+1$, as desired. Thus, to find easily checkable sufficient conditions for F to be good, we will instead find easily checkable sufficient conditions which guarantee that for every doubly optimal reticulate F-network N, some root of $\mathcal{F}(N)$ has in-degree at least 3 in N.

Lemma 18. *For every doubly optimal reticulate F-network N, r_{m+1} is the root of both N and the tree in $\mathcal{F}(N)$ corresponding to Γ_{m+1}.*

Proof. Roughly speaking, if r_{m+1} were not the root of N, then we would be able to decrease $R_2(N)$ by modifying N by deleting all nodes from which we can reach r_{m+1}. The details are given in Section 1, available in the online supplemental material. □

Lemma 19. *Consider some $j \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, k\}$ such that $p_{j,i}$ is the parent of r_j in T_i. Then, for every doubly optimal reticulate F-network N and for every embedding E_i of T_i in N, no inner node of the path from $p_{j,i}$ to r'_j in E_i is a root in $\mathcal{F}(N)$, where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j.*

Proof. Fig. 5(1) helps understand the proof. For each $h \in \{1, \ldots, m\}$, let r'_h be the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_h. Let $Q_{j,i}$ be the path from $p_{j,i}$ to r'_j in E_i. Since r_{m+1} is the root of N (by Lemma 18), r_{m+1} cannot be an inner node of $Q_{j,i}$. Toward a contradiction, assume that there is an integer $h \in \{1, \ldots, m\}$ such that r'_h is an inner node of $Q_{j,i}$. Let $x \in X$ be a leaf descendant of r'_h in $\mathcal{F}(N)$. Since N is doubly optimal, x must exist. By Statement 4 in Lemma 15, the path from r_{m+1} to x in E_i must contain the path P from r'_h to x in $\mathcal{F}(N)$. Let P' be the subpath of $Q_{j,i}$ from r'_h to r'_j. Since P cannot pass through r'_j, P and P' must share a node t such that the edge leaving t in P is different from the edge leaving t in P'. So, t is an inner node of $Q_{j,i}$ and its out-degree in E_i is 2. On the other hand, since $(p_{j,i}, r_j)$ is an edge of T_i and E_i is an embedding of T_i in N, every inner node of $Q_{j,i}$ must have out-degree 1 in E_i. Therefore, we have a contradiction. This completes the proof. □

Lemma 20. *Consider some $j \in \{1, \ldots, m\}$, $i_1 \in \{1, \ldots, k\}$, and $i_2 \in \{1, \ldots, k\}$ such that $i_1 \neq i_2$, $e_{j,i_1} \neq e_{j,i_2}$, p_{j,i_1} is the parent of r_j in T_{i_1}. Then, for every doubly optimal reticulate F-network N, for every embedding E_{i_1} of T_{i_1} in N, and for every embedding E_{i_2} of T_{i_2} in N, r'_j is the only node shared by the path from p_{j,i_1} to r'_j in E_{i_1} and the path from p_{j,i_2} to r'_j in E_{i_2} (and hence the edge entering r'_j in E_{i_1} is different from the edge entering r'_j in E_{i_2}), where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j.*

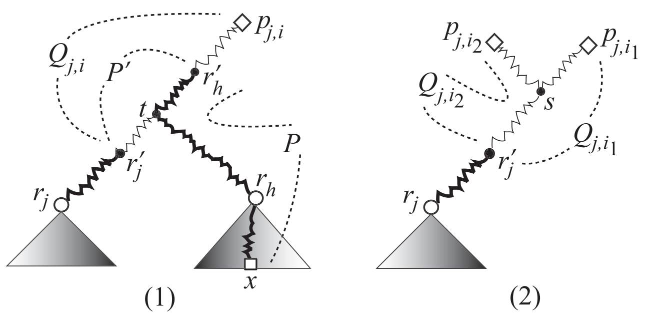

Fig. 5. (1) The paths P, P', and $Q_{j,i}$ (shown in zigzag lines or curves) in the proof of Lemma 19. (2) The paths Q_{j,i_1} and Q_{j,i_2} (shown in zigzag lines or curves) in the proof of Lemma 20.

Proof. Fig. 5(2) helps understand the proof. Let r'_j be the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j. Let Q_{j,i_1} (resp., Q_{j,i_2}) be the path from p_{j,i_1} (resp., p_{j,i_2}) to r'_j in E_{i_1} (resp., E_{i_2}). Note that r'_j is a node shared by Q_{j,i_1} and Q_{j,i_2}. We claim that no node of N other than r'_j can be shared by Q_{j,i_1} and Q_{j,i_2}. Toward a contradiction, assume that the claim is false. Then, starting at p_{j,i_1} and walking along Q_{j,i_1} toward r'_j, we can find the first node $s \neq r'_j$ shared by Q_{j,i_1} and Q_{j,i_2}. Since $e_{j,i_1} \neq e_{j,i_2}$, p_{j,i_1} and p_{j,i_2} are different nodes in N. Thus, s cannot be p_{j,i_1} or p_{j,i_2}. Hence, s is an inner node of both Q_{j,i_1} and Q_{j,i_2}. Consequently, by Lemma 19, s is not a root in $\mathcal{F}(N)$. Therefore, the in-degree of s in N is 1. However, by the choice of s, the edge of Q_{j,i_1} entering s and the edge of Q_{j,i_2} entering s must be different, implying that the in-degree of s in N is at least 2. So, we have a contradiction. This finishes the proof. □

For each $j \in \{1, \ldots, m\}$, we define two sets as follows:

- Let I_j denote the set of integers $i \in \{1, \ldots, k\}$ such that $p_{j,i}$ is the parent of r_j in T_i.
- Let $S_j = \{e_{j,i} | i \in I_j\}$.

For example, in Fig. 1, $I_1 = \{1, 3\}$, $S_1 = \{e_5, e_4\}$, $I_2 = \{1, 2, 3\}$, $S_2 = \{e_1\}$, $I_3 = \{1, 3\}$, $S_3 = \{e_1\}$, and $I_4 = \{1, 3\}$, $S_4 = \{e_5\}$.

By Lemma 20, the in-degree of r'_j in every doubly optimal reticulate F-network N is at least $|S_j|$, where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j. Thus, if there is a $j \in \{1, \ldots, m\}$ with $|S_j| \geq 3$, then F is good. This gives us an easily checkable sufficient condition for F to be good. Unfortunately, this condition is too strong that not so many MAAFs F satisfy it. For example, the MAAF F in Fig. 1 does not satisfy the condition. So, we next proceed to find a weaker sufficient condition. The idea is to expand the sets $S_1, \ldots, S_m$ based on the following sets:

- For each $j \in \{1, \ldots, m\}$, let $\overline{I_j}$ be the set of all $i \in \{1, \ldots, k\} - I_j$ with $e_{j,i} \notin S_j$.

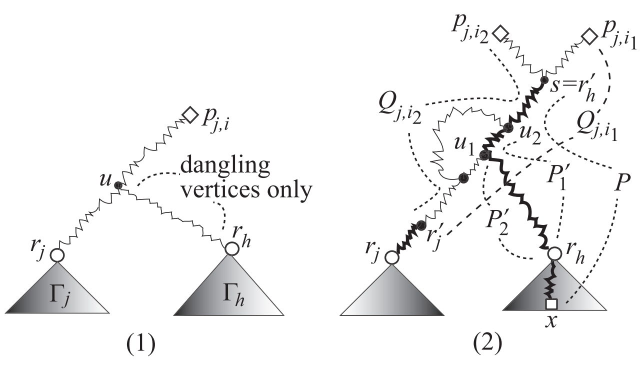

Fig. 6. (1) A portion of T_i witnessing that $h \in H_{j,i}$. (2) The paths Q_{j,i_1}, Q_{j,i_2}, P, P'_1, and P'_2 (shown in zigzag lines or curves) in the proof of Lemma 21.

- For each $j \in \{1,\dots,m\}$ and each $i \in \overline{I_j}$, let $H_{j,i}$ denote the set of all $h \in \{1,\dots,m\} - \{j\}$ such that r_h is a descendant of some inner node u of the path from $p_{j,i}$ to r_j in T_i and every inner node of the path from u to r_h in T_i is a dangling node in T_i. See Fig. 6(1) for an illustration.

For example, in Fig. 1, $\overline{I_1} = \emptyset$, $\overline{I_2} = \{4\}$, $\overline{I_3} = \{2\}$, $\overline{I_4} = \{2,4\}$, $H_{2,4} = \{1\}$, $H_{3,2} = \{1,4\}$, $H_{4,2} = \{1,3\}$, and $H_{4,4} = \{3\}$.

The intuition behind each $\overline{I_j}$ is as follows: we have used the trees T_i with $i \in I_j$ and the edges in S_j to obtain a lower bound (namely, $|S_j|$) on the in-degree of r'_j in every doubly optimal reticulate F-network N, where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j. In order to increase this lower bound by expanding S_j, we have to exclude the trees T_j with $i \in I_j$ and the edges in S_j from further consideration (to avoid double counting).

Lemma 21. *Consider some $j \in \{1,\dots,m\}$, $i_1 \in \{1,\dots,k\}$, and $i_2 \in \{1,\dots,k\}$ such that $i_1 \neq i_2$, $H_{j,i_1} \cap H_{j,i_2} = \emptyset$, and $e_{j,i_1} \neq e_{j,i_2}$. Then, for every doubly optimal reticulate F-network N, for every embedding E_{i_1} of T_{i_1} in N, and for every embedding E_{i_2} of T_{i_2} in N, r'_j is the only node shared by the path from p_{j,i_1} to r'_j in E_{i_1} and the path from p_{j,i_2} to r'_j in E_{i_2} (and hence the edge entering r'_j in E_{i_1} is different from the edge entering r'_j in E_{i_2}), where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j.*

Proof. Fig. 6(2) helps understand the proof. For each $h \in \{1,\dots,m\}$, let r'_h be the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_h. Note that $r'_1,\dots,r'_m$ are the reticulation nodes of N. Let Q_{j,i_1} (resp., Q_{j,i_2}) be the path from p_{j,i_1} (resp., p_{j,i_2}) to r'_j in E_{i_1} (resp., E_{i_2}). Obviously, r'_j is a node shared by Q_{j,i_1} and Q_{j,i_2}. We claim that no node of N other than r'_j can be shared by Q_{j,i_1} and Q_{j,i_2}. Toward a contradiction, assume that the claim is false. Then, starting at p_{j,i_1} and walking along Q_{j,i_1} toward r'_j, we can find the first node $s \neq r'_j$ shared by Q_{j,i_1} and Q_{j,i_2}. Since $e_{j,i_1} \neq e_{j,i_2}$, p_{j,i_1} and p_{j,i_2} are different nodes in N. Thus, s cannot be p_{j,i_1} or p_{j,i_2}. Hence, s is an inner node of both Q_{j,i_1} and Q_{j,i_2}. By the choice of s, the edge of Q_{j,i_1} entering s and the edge of Q_{j,i_2} entering s must be different, implying that the in-degree of s in N is at least 2. Thus, $s = r'_h$ for some $h \in \{1,\dots,m\} - \{j\}$.

Let $x \in X$ be a leaf descendant of r'_h in $\mathcal{F}(N)$. Since N is doubly optimal, x must exist. By Statement 4 in Lemma 15, the path from r_{m+1} to x in E_{i_1} (resp., E_{i_2}) must contain the path P from r'_h to x in $\mathcal{F}(N)$. For each $\ell \in \{1,2\}$, let u_ℓ be the node closest to r_h in P that is shared by Q_{j,i_ℓ} and P. Since P cannot pass through r'_j, u_ℓ is an inner node of the path from p_{j,i_ℓ} to r_j in T_{i_ℓ}. Consider the subpath P'_ℓ of P from u_ℓ to r_h. Every node of P'_ℓ with out-degree 2 in E_{i_ℓ} is a dangling node in T_{i_ℓ} because E_{i_ℓ} is an embedding of T_{i_ℓ} in N. Hence, $h \in H_{j,i_1} \cap H_{j,i_2}$, contradicting the assumption that $H_{j,i_1} \cap H_{j,i_2} = \emptyset$. This finishes the proof. □

Based on Lemma 21, we can expand S_j by first initializing $J_j = \emptyset$ and then performing the following step for $h = 1,2,\dots,|\overline{I_j}|$:

- Let i_h be the hth integer in $\overline{I_j}$. If $e_{j,i_h} \notin S_j$ and $H_{j,i_h} \cap J_j = \emptyset$, then add e_{j,i_h} to S_j and also add the elements of H_{j,i_h} to J_j.

The size of the final S_j depends on the ordering of the integers in $\overline{I_j}$. We want an ordering that maximizes the size of the final S_j. When $|\overline{I_j}|$ is small, we can try all possible orderings and find the best among them. Otherwise, we may just try a small number of random orderings and find the best among them. We make this rough idea more precise below.

For each $j \in \{1,\dots,m\}$ and each bijection $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$, we compute a set $S_{j,f}$ of edges in F and a subset $J_{j,f}$ of $\{1,\dots,m\}$ as follows: initially, $S_{j,f} = S_j$ and $J_{j,f} = \emptyset$. We then expand $S_{j,f}$ and $J_{j,f}$ by performing the following step:

- For $h = 1,2,\dots,|\overline{I_j}|$ (in this order), if $H_{j,f(h)} \cap J_{j,f} = \emptyset$ and $S_{j,f}$ does not contain $e_{j,f(h)}$, then add $e_{j,f(h)}$ to $S_{j,f}$ and add the elements of $H_{j,f(h)}$ to $J_{j,f}$.

Here, if $\overline{I_j} = \emptyset$, there is a unique bijection $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$ (namely, the empty mapping).

For example, in Fig. 1, if f_1 is the identity function, then $S_{1,f_1} = \{e_5, e_4\}$, $J_{1,f_1} = \emptyset$, $S_{2,f_1} = \{e_1, e_4\}$, $J_{2,f_1} = \{1\}$, $S_{3,f_1} = \{e_1, e_4\}$, $J_{3,f_1} = \{1,4\}$, $S_{4,f_1} = \{e_5, e_4\}$, and $J_{4,f_1} = \{1,3\}$. In the same example, if f_2 is the function with $f_2(1) = 4$, $f_2(2) = 3$, $f_2(3) = 2$, and $f_2(4) = 1$, then $S_{1,f_2} = \{e_5, e_4\}$, $J_{1,f_2} = \emptyset$, $S_{2,f_2} = \{e_1, e_4\}$, $J_{2,f_2} = \{1\}$, $S_{3,f_2} = \{e_1, e_4\}$, $J_{3,f_2} = \{1,4\}$, $S_{4,f_2} = \{e_5, e_1\}$, and $J_{4,f_2} = \{3\}$.

By Lemmas 20 and 21 and the construction of $S_{j,f}$, the in-degree of r'_j in every doubly optimal reticulate F-network N is at least $|S_{j,f}|$, where r'_j is the root of the tree in $\mathcal{F}(N)$ corresponding to Γ_j. So, if there are $j \in \{1,\dots,m\}$ and bijection $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$ such that $|S_{j,f}| \geq 3$, then F is good. This gives us a weaker sufficient condition for F to be good. As an example, the MAAF F in Fig. 1 still does not satisfy this weaker condition.

If j is an integer in $\{1,\dots,m\}$ such that $|\overline{I_j}|$ is small, then we can afford to compute $S_{j,f}$ for all bijections $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$. However, for each $j \in \{1,\dots,m\}$ such that $|\overline{I_j}|$ is large, it is too time consuming to compute $S_{j,f}$ for all bijections $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$. So, the above sufficient condition may not be polynomial-time checkable. A simple idea to get around this problem is to predetermine two small numbers b_1 (say, 5) and b_2 (say, 200). For each $j \in \{1,\dots,m\}$ with $|\overline{I_j}| \leq b_1$, we compute $S_{j,f}$ for all bijections $f : \{1,\dots,|\overline{I_j}|\} \to \overline{I_j}$. On the other hand, for each $j \in \{1,\dots,m\}$ with $|\overline{I_j}| > b_1$, we compute $S_{j,f}$ for b_2 random

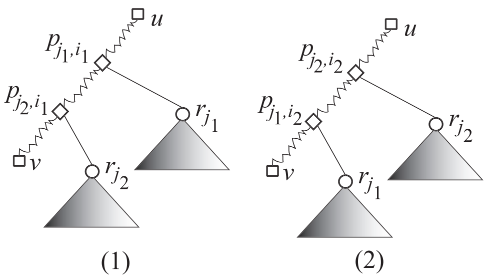

Fig. 7. (1) A portion of T_{i_1}, where (u,v) is an edge of F. (2) A portion of T_{i_2}, where (u,v) is the same edge of F as in (1).

bijections $f:\{1,\ldots,|\overline{I_j}|\}\to\overline{I_j}$. In this way, if we find a $j\in\{1,\ldots,m\}$ and an $f:\{1,\ldots,|\overline{I_j}|\}\to\overline{I_j}$ such that $|S_{j,f}|\geq 3$, then F is good. This gives us an easily checkable sufficient condition for F to be good.

Suppose that after checking the above easily checkable sufficient condition, we have not found an integer $j\in\{1,\ldots,m\}$ and a bijection $f:\{1,\ldots,|\overline{I_j}|\}\to\overline{I_j}$ such that $|S_{j,f}|\geq 3$. Then, we cannot conclude that F is good. However, it is too early to give up. Our idea is to look at the set J of those integers $j\in\{1,\ldots,m\}$ such that we have found at least one bijection $f:\{1,\ldots,|\overline{I_j}|\}\to\overline{I_j}$ with $|S_{j,f}|=2$.

Lemma 22. *Suppose that j_1 and j_2 are two integers in J and i_1 and i_2 are two integers in $\{1,\ldots,k\}$ satisfying the following condition C1 (see Fig. 7):*

- *$C1: e_{j_1,i_1}=e_{j_2,i_1}=e_{j_1,i_2}=e_{j_2,i_2}$, p_{j_1,i_1} and p_{j_2,i_1} are the parents of r_{j_1} and r_{j_2} in T_{i_1}, respectively, p_{j_1,i_2} and p_{j_2,i_2} are the parents of r_{j_1} and r_{j_2} in T_{i_2}, respectively, p_{j_1,i_1} is an ancestor of p_{j_2,i_1} in T_{i_1}, and p_{j_2,i_2} is an ancestor of p_{j_1,i_2} in T_{i_2}.*

Then, for every doubly optimal reticulate F-network N, the in-degree of the root of Γ'_{j_1} in N is at least 3 or the in-degree of the root of Γ'_{j_2} in N is at least 3, where Γ'_{j_1} (resp., Γ'_{j_2}) is the tree in $\mathcal{F}(N)$ corresponding to Γ_{j_1} (resp., Γ_{j_2}).

Proof. Let E_{i_1} (resp., E_{i_2}) be an arbitrary embedding of T_{i_1} (resp., T_{i_2}) in N. We claim that at least one of the following statements holds:

1. p_{j_1,i_1} in E_{i_1} and p_{j_1,i_2} in E_{i_2} are distinct nodes of N.
2. p_{j_2,i_1} in E_{i_1} and p_{j_2,i_2} in E_{i_2} are distinct nodes of N.

Toward a contradiction, assume that neither of the statements holds. Then, since p_{j_2,i_2} is an ancestor of p_{j_1,i_2} in T_{i_2}, there is a path P_1 from $p_{j_2,i_1}=p_{j_2,i_2}$ to $p_{j_1,i_1}=p_{j_1,i_2}$ in E_{i_2}. Moreover, since p_{j_1,i_1} is an ancestor of p_{j_2,i_1} in T_{i_1}, there is a path P_2 from p_{j_1,i_1} to p_{j_2,i_1} in E_{i_1}. Note that P_1 and P_2 are paths in N. However, the existence of P_1 and P_2 implies that there is a cycle in N, a contradiction. So, the claim holds.

By the claim, Statement 1 or 2 holds. We assume that Statement 1 holds; the other case is similar. Let Q_{j_1,i_1} (resp., Q_{j_1,i_2}) be the path from p_{j_1,i_1} (resp., p_{j_1,i_2}) to r'_{j_1} in E_{i_1} (resp., E_{i_2}), where r'_{j_1} is the root of Γ'_{j_1}. An argument similar to the proof of Lemma 20 shows that r'_{j_1} is the only node shared by Q_{j_1,i_1} and Q_{j_1,i_2}. Thus, the edge e_1 of Q_{j_1,i_1} entering r'_{j_1} is different from the edge e_2 of Q_{j_1,i_2} entering r'_{j_1}.

Since $j_1\in J$, we have already found a bijection $f:\{1,\ldots,|\overline{I_{j_1}}|\}\to\overline{I_{j_1}}$ with $|S_{j_1,f}|=2$. By Lemmas 20 and 21 and the construction of $S_{j_1,f}$, there are two distinct edges e_3 and e_4 entering r'_{j_1} in N. It is possible that $\{e_3,e_4\}\cap\{e_1,e_2\}\neq\emptyset$. However, at most one of e_1 and e_2 belongs to $\{e_3,e_4\}$, because $e_{j_1,i_1}=e_{j_1,i_2}$ but $S_{j_1,f}$ contains two different supporting edges. Hence, the in-degree of r'_{j_1} in N is at least 3. □

If there are two integers j_1 and j_2 in J and two integers i_1 and i_2 in $\{1,\ldots,k\}$ satisfying Condition C1 in Lemma 22, then F is good by Lemma 22. For example, in Fig. 1, if we predetermine $b_1=5$ and $b_2=100$, then $J=\{1,2,3,4\}$. Moreover, the two integers $j_1=2$ and $j_2=3$ in J and the two integers $i_1=1$ and $i_2=3$ in $\{1,\ldots,4\}$ satisfy Condition C1. Thus, the MAAF F in Fig. 1 is good.

Since it is easy to check whether there are two integers j_1 and j_2 in J and two integers i_1 and i_2 in $\{1,\ldots,k\}$ satisfying Condition C1 in Lemma 22, we have another easily checkable sufficient condition for F to be good.

By the above discussions, we now have an algorithm for deciding if a given MAAF F of $T_1,\ldots,T_k$ is good. It is depicted in Fig. 8.

5 Implementation

We have implemented our algorithms in ANSI C, obtaining programs *CMPT* and *MaafB* for comparing multiple phylogenetic trees and computing a lower bound on the reticulation number of an optimal type-II reticulate network of multiple phylogenetic trees, respectively. The programs are available at the website, where one can download executables that can run on a Windows XP (x86), Windows 7 (x64), Macintosh, or Linux machine. Section 4, available in the online supplemental material, details how to run the programs.

When running *MaafB*, the user can choose to compute the RH bound or not. If the user chooses not to compute the RH bound, then *MaafB* will output the rMAAF bound only. Otherwise, it will output the larger bound between the 2, implying that *MaafB* does not output a lower bound smaller than *PIRN*. To compute the RH bound, *MaafB* tests if the RH bound is larger than i for $i=\ell,\ell+1,\ldots$ (in this order), where ℓ is the rMAAF bound. Note that *PIRN* computes the RH bound by testing if the RH bound is larger than i for $i=b,\ b+1,\ldots$ (in this order), where b is the maximum disconnectivity of an MAAF of two of the input trees. Obviously, ℓ is at least as large as b. Indeed, ℓ is often larger than b. Thus, *MaafB* can often compute the RH bound faster than *PIRN*. It is worth noting that the downloadable version of *MaafB* uses the GLPK library to compute the RH bound and hence can be slow.

6 Experimental Results

To test the performance of *MaafB*, we have compared it with *PIRN* on both simulated data and biological data on a 2.66 GHz Mac-OS-X PC. To compute the RH bound, we use CPLEX (a commercial ILP solver that is now freely available from IBM for academic research).

```
Input: $T_1, \ldots, T_k$ and their MAAF $F = \{\Gamma_1, \ldots, \Gamma_{m+1}\}$, where $\Gamma_{m+1}$ is the component tree of $F$ containing the dummy leaf.
Output: "Yes" if $F$ is good, "no" otherwise.
1. Select two small integers $b_1$ (say, 5) and $b_2$ (say, 200).
2. For each $1 \leq i \leq k$, use $F$ to classify the nodes of $T_i$ into preserved nodes, contracted nodes, and dangling nodes.
3. For each $1 \leq j \leq m$ and each $1 \leq i \leq k$, perform Steps 3.1 through 3.3:
3.1. Find the lowest ancestor $p_{j,i}$ of the root $r_j$ of $\Gamma_j$ in $T_i$ that is a contracted node.
3.2. Find the edge $e_{j,i} = (u, v)$ in $F$ such that the path from $u$ to $v$ in $T_i$ contains $p_{j,i}$.
3.3. Compute the set $H_{j,i}$ of all $h \in \{1, \ldots, m\} - \{j\}$ such that the root $r_h$ of $\Gamma_h$ is a descendant of some inner node $u$ of the path from $p_{j,i}$ to $r_j$ in $T_i$ and every inner node of the path from $u$ to $r_h$ in $T_i$ is a dangling node.
4. For each $j \in \{1, \ldots, m\}$, compute $I_j = \{i \in \{1, \ldots, k\} \mid p_{j,i}$ is the parent of $r_j$ in $T_i\}$, $S_j = \{e_{j,i} \mid i \in I_j\}$, and $\overline{I_j} = \{i \in \{1, \ldots, k\} - I_j \mid e_{j,i} \notin S_j\}$.
5. Initialize $J = \emptyset$.
6. For every $j \in \{1, \ldots, m\}$ with $|\overline{I_j}| \leq b_1$ and for every bijection $f : \{1, \ldots, |\overline{I_j}|\} \to \overline{I_j}$, perform Steps 6.1~6.4:
6.1. Initialize $S_{j,f} = S_j$ and $J_{j,f} = \emptyset$.
6.2. For $h = 1, 2, \ldots, |\overline{I_j}|$ (in this order),
       if $H_{j,f(h)} \cap J_{j,f} = \emptyset$ and $e_{j,f(h)} \notin S_{j,f}$, then add $e_{j,f(h)}$ to $S_{j,f}$ and add the elements of $H_{j,f(h)}$ to $J_{j,f}$.
6.3. If $|S_{j,f}| \geq 3$, then output "yes" and halt.
6.4. If $|S_{j,f}| = 2$ and $j \notin J$, then add $j$ to $J$.
7. For every $j \in \{1, \ldots, m\}$ with $|\overline{I_j}| > b_1$, generate $b_2$ random bijections $f : \{1, \ldots, |\overline{I_j}|\} \to \overline{I_j}$, and perform Steps 6.1 through 6.4 for each generated bijection $f$.
8. If there are integers $j_1$ and $j_2$ in $J$ and integers $i_1$ and $i_2$ in $\{1, \ldots, k\}$ satisfying Condition C1, then output "yes" and halt.
9. Output "no" and halt.
```

Fig. 8. The algorithm for deciding if an MAAF is good.

6.1 Simulated Data

We use the same data sets as in [18] whose author generates a data set using a two-stage approach: first, simulate a reticulate network N, and then generate a fixed number of trees from N by deleting all but one randomly chosen edge entering each reticulation node in N. To simulate a reticulate network, Wu [18] uses a scheme similar to the coalescent simulation implemented in program ms due to Hudson [10] as follows: for a given number t of taxa, we start with t isolated lineages and simulate reticulation backward in time. At each step, there are two possible events: 1) lineage merging, which occurs at rate 1; and 2) lineage splitting, which occurs at rate r. We choose the next event according to relative probabilities of all feasible events. Lineage merging generates speciation events, while lineage splitting generates reticulation events. The parameter r dictates the level of reticulation in the simulated network: larger r will lead to more reticulation events in simulation.

Table 1 summarizes our experimental results on estimating a lower bound on the reticulation number of an optimal type-II reticulate network of multiple (four or five) trees. For each triple (t, r, k), 100 data sets are tested and the average running time for computing the rMAAF bound for one data set is shorter than 22 seconds and is less than half the average running time of *PIRN* for the same data set. The experimental results in Table 1 indicate that the rMAAF bound is often larger than the MAAF bound and is also larger than the RH bound for a significant fraction of data sets. This is particularly true when the disconnectivity of MAAFs of the trees becomes large.

6.2 Other Simulated Data

An *rSPR operation* on a phylogenetic tree T first detaches the subtree of T rooted at a nonroot node v and then reattaches the subtree to an edge (u, w) of T (by introducing a new node v', splitting edge (u, w) into two edges (u, v') and (v', w), and adding a new edge (v', v)). Beiko and Hamilton [1] have written a program for performing a given number of random rSPR operations on a given phylogenetic tree. They have also written a program for generating a random phylogenetic tree with a given number of leaves. Using their programs, we can generate multiple phylogenetic trees in several ways. To compare the rMAAF and the RH bounds, we generate multiple phylogenetic trees in two ways. In the first way, we generate a random phylognetic tree T_0 with 20 leaves and then use it to obtain k other trees by performing the following step:

- For $i = 1, 2, \ldots, k$, perform three random rSPR operations on T_0 to obtain T_i.

In the second way, we generate a random phylognetic tree T_1 with 20 leaves and then use it to obtain $k - 1$ other trees by performing the following step:

- For $i = 2, 3, \ldots, k$, perform three random rSPR operations on T_{i-1} to obtain T_i.

Roughly speaking, two of the trees $T_1, \ldots, T_k$ obtained in the first way are not so different while two of the trees $T_1, \ldots, T_k$ obtained in the second way can be quite different. For each $k \in \{7, 10, 15\}$ and each $j \in \{1, 2\}$, we generate 20 sets of multiple phylogenetic trees in the jth way and compare the rMAAF and the RH bounds on the sets. Table 2 summarizes the experimental results. As can be seen from the table, the rMAAF bounds are significantly larger than the RH bounds for the sets of multiple phylogenetic trees generated in the first way while the rMAAF bounds are usually not better than the RH bounds for the sets of multiple phylogenetic trees generated in the second way. Section 6 in the supplementary material, which is available online, contains more detailed comparison of the rMAAF and the RH bounds. Moreover, the data sets are available at the website.

TABLE 3
Comparing the rMAAF and the RH Bounds on the Poaceae Data Sets

dataset	$\|X\|$	Maaf	rMaaf	RH
rpoC2, waxy, ITS	11	6	6	7
ndhF, phyB, rbcL	22	9	10	10
ndhF, phyB, rbcL, rpoC2, ITS	14	9	10	11

Column "$|X|$" shows the number of leaves in one input tree, and column "Maaf" (respectively, "rMaaf" or "RH") shows the MAAF (resp., rMAAF or RH) bound of each set of trees.

6.3 Biological Data

We use the Poaceae data set from the Grass Phylogeny Working Group [7]. The data set contains sequences for six loci: internal transcribed spacer of ribosomal DNA (ITS); NADH dehydrogenase, subunit F (ndhF); phytochrome B (phyB); ribulose 1,5-biphosphate carboxylase/oxygenase, large subunit (rbcL); RNA polymerase II, subunit β'' (rpoC2); and granule bound starch synthase I (waxy). The Poaceae data set was previously analyzed by Schmidt [11], who generated the inferred rooted binary trees for these loci.

Table 3 summarizes our experimental results on estimating a lower bound on the reticulation number of an optimal type-II reticulate network of multiple (three to five) trees. As can be seen from the table, the lower bounds outputted by *MaafB* are the same as those outputted by *PIRN*, but the rMAAF bound is not better than the RH bound because each data set contains very few trees or very small trees.

Acknowledgments

The authors thank Y. Wu for the simulated data sets and the referees for very helpful comments. Zhi-Zhong Chen was supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 20500021. Lusheng Wang was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 121608].

References

[1] R.G. Beiko and N. Hamilton, "Phylogenetic Identification of Lateral Genetic Transfer Events," *BMC Evolutionary Biology,* vol. 6, pp. 159-169, 2006.

[2] M. Bordewich and C. Semple, "On the Computational Complexity of the Rooted Subtree Prune and Regraft Distance," *Annals of Combinatorics,* vol. 8, pp. 409-423, 2005.

[3] M. Bordewich and C. Semple, "Computing the Minimum Number of Hybridization Events for a Consistent Evolutionary History," *Discrete Applied Math.,* vol. 155, pp. 914-928, 2007.

[4] M. Bordewich and C. Semple, "Computing the Hybridization Number of Two Phylogenetic Trees Is Fixed-Parameter Tractable," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 4, no. 3, pp. 458-466, July 2007.

[5] Z.-Z. Chen and L. Wang, "HybridNet: A Tool for Constructing Hybridization Networks," *Bioinformatics,* vol. 26, pp. 2912-2913, http://rnc.r.dendai.ac.jp/~chen/notess2.pdf or http://www.cs.cityu.edu.hk/~lwang/software/Hn/notess2.pdf), 2010.

[6] L. Collins, S. Linz, and C. Semple, "Quantifying Hybridization in Realistic Time," *J. Computational Biology,* vol. 18, pp. 1305-1318, http://www.math.canterbury.ac.nz/~c.semple/software.shtml], 2011.

[7] N.P. Barker et al., "Phylogeny and Subfamilial Classification of the Grasses (Poaceae)," *Annals of the Missouri Botanical Garden,* vol. 88, pp. 373-457, 2001.

[8] J. Hein, T. Jing, L. Wang, and K. Zhang, "On the Complexity of Comparing Evolutionary Trees," *Discrete Applied Math.,* vol. 71, pp. 153-169, 1996.

[9] T. Hill, K.J. Nordström, M. Thollesson, T.M. Säfström, A.K. Vernersson, R. Fredriksson, and H.B. Schiöth, "SPRIT: Identifying Horizontal Gene Transfer in Rooted Phylogenetic Trees," *BMC Evolutionary Biology,* vol. 10, article 42, 2010.

[10] R. Hudson, "Generating Samples under the Wright-Fisher Neutral Model of Genetic Variation," *Bioinformatics,* vol. 18, pp. 337-338, 2002.

[11] H.A. Schmidt, "Phylogenetic Trees from Large Data Sets," PhD thesis, Heinrich-Heine-Universitat, Dusseldorf, 2003.

[12] B. Schieber and U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization," *SIAM J. Computing,* vol. 17, pp. 1253-1262, 1988.

[13] C. Semple, "Reticulate Evolution," http://www.lirmm.fr/MEP05/talk/20_Semple.pdf, 2005.

[14] C. Semple, "Hybridization Networks," *Reconstructing Evolution: New Mathematical and Computational Advances,* O. Gascuel and M. Steel, eds., pp. 277-314, Oxford Univ. Press, 2007.

[15] J. Wang and Y. Wu, "Fast Computation of the Exact Hybridization Number of Two Phylogenetic Trees," *Proc. Int'l Symp. Bioinformatics Research and Applications (ISBRA '10),* pp. 203-214, 2010.

[16] C. Whidden, R.G. Beiko, and N. Zeh, "Fast FPT Algorithms for Computing Rooted Agreement Forest: Theory and Experiments," *Proc. Int'l Symp. Experimental Algorithms,* pp. 141-153, 2010.

[17] Y. Wu, "A Practical Method for Exact Computation of Subtree Prune and Regraft Distance," *Bioinformatics,* vol. 25, pp. 190-196, 2009.

[18] Y. Wu, "Close Lower and Upper Bounds for the Minimum Reticulate Network of Multiple Phylogenetic Trees," *Bioinformatics,* vol. 26, pp. 140-148, 2010.

Zhi-Zhong Chen received the PhD degree from the University of Electro-Communications, Tokyo, Japan, in 1992. Currently, he is a professor in the Division of Information System Design, Tokyo Denki University. His research interests include algorithms, computational biology, and graph theory.

Lusheng Wang received the PhD degree from McMaster University, Hamilton, Ontario, Canada, in 1995. Currently, he is a professor in the Department of Computer Science, City University of Hong Kong. His research interests include algorithms, bioinformatics, and computational biology. He is a member of the IEEE.