SIAM J. COMPUT.
Vol. 35, No. 5, pp. 1098–1121

ALGORITHMS FOR COMBINING ROOTED TRIPLETS INTO A GALLED PHYLOGENETIC NETWORK*

JESPER JANSSON†, NGUYEN BAO NGUYEN†, AND WING-KIN SUNG†‡

Abstract. This paper considers the problem of determining whether a given set $\mathcal{T}$ of rooted triplets can be merged without conflicts into a galled phylogenetic network and, if so, constructing such a network. When the input $\mathcal{T}$ is dense, we solve the problem in $O(|\mathcal{T}|)$ time, which is optimal since the size of the input is $\Theta(|\mathcal{T}|)$. In comparison, the previously fastest algorithm for this problem runs in $O(|\mathcal{T}|^2)$ time. We also develop an optimal $O(|\mathcal{T}|)$-time algorithm for enumerating all simple phylogenetic networks leaf-labeled by L that are consistent with $\mathcal{T}$, where L is the set of leaf labels in $\mathcal{T}$, which is used by our main algorithm. Next, we prove that the problem becomes NP-hard if extended to nondense inputs, even for the special case of simple phylogenetic networks. We also show that for every positive integer n, there exists some set $\mathcal{T}$ of rooted triplets on n leaves such that any galled network can be consistent with at most $0.4883 \cdot |\mathcal{T}|$ of the rooted triplets in $\mathcal{T}$. On the other hand, we provide a polynomial-time approximation algorithm that always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in $\mathcal{T}$.

Key words. phylogenetic network, galled network, rooted triplet, SN-tree, NP-hardness, algorithm

AMS subject classifications. 68Q25, 68R10, 05C85, 92B99

DOI. 10.1137/S0097539704446529

1. Introduction.

A *rooted triplet* is a binary, rooted, unordered tree with three distinctly labeled leaves. Aho et al. [1] introduced the problem of determining whether a given set of rooted triplets can be combined without conflicts into a distinctly leaf-labeled tree which contains each of the given rooted triplets as an embedded subtree, and, if so, returning one. The original motivation for this problem came from an application in the theory of relational databases (see [1] for details), but it has since been studied further and generalized because of its applications to phylogenetic tree construction [2, 6, 7, 10, 13, 14, 17, 20, 21, 23, 25]. Here, we study an extension of the problem in which the objective is to determine if a given set $\mathcal{T}$ of rooted triplets can be merged into a more complex structure known as a *galled phylogenetic network*.

A *phylogenetic network* is a type of distinctly leaf-labeled, directed acyclic graph that can be used to model nontreelike evolution. A number of methods for inferring phylogenetic networks under various assumptions and using different kinds of data have been proposed recently [8, 12, 15, 19, 22, 24]. A *galled phylogenetic network*, or *galled network* for short, is an important, biologically motivated structural restriction of a phylogenetic network (see section 1.2) in which all cycles in the underlying undirected graph are node-disjoint.[1]

*Received by the editors November 12, 2004; accepted for publication (in revised form) October 2, 2005; published electronically March 3, 2006. An extended abstract of this article has appeared in *Proceedings of the* 16*th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, pp. 349–358.

http://www.siam.org/journals/sicomp/35-5/44652.html

†School of Computing, National University of Singapore, 3 Science Drive 2, 117543 Singapore (jansson@comp.nus.edu.sg, nguyenba@comp.nus.edu.sg, ksung@comp.nus.edu.sg). The first author was supported in part by Kyushu University and JSPS (Japan Society for the Promotion of Science).

‡Genome Institute of Singapore, 60 Biopolis Street, Genome, 138672 Singapore.

[1]Without the node-disjoint constraint, our problem becomes trivial to solve since then a solution always exists and, furthermore, can be obtained in polynomial time using a simple sorting network-based construction [15].

We present several new results for the problem of inferring a galled network consistent with a given set $\mathcal{T}$ of rooted triplets. Denote the set of leaf labels in $\mathcal{T}$ by L. If $\mathcal{T}$ contains at least one rooted triplet for each cardinality-three subset of L, then $\mathcal{T}$ is called *dense*. We first give an exact algorithm named *FastGalledNetwork* for dense inputs whose running time is $O(|\mathcal{T}|)$. In comparison, the previously fastest known algorithm for this case runs in $O(|\mathcal{T}|^2)$ time [15]. Since the size of the input is $\Theta(|\mathcal{T}|)$ when $\mathcal{T}$ is dense and any algorithm that solves the problem must look at the entire input, the asymptotic running time of our new algorithm is optimal. The improvement in running time is due to two observations: first, that the so-called SN-sets employed in [15] do not have to be explicitly computed but can be represented using a tree (the SN-tree) which we can construct in $O(|\mathcal{T}|)$ time, and second, that the SN-tree can be expanded into a galled network consistent with $\mathcal{T}$ (if one exists) in $O(|\mathcal{T}|)$ time by replacing each internal node of degree 3 or higher with a special kind of network found by applying an algorithm called *SimpleNetworks*.

Next, we show that the problem becomes NP-hard when $\mathcal{T}$ is not required to be dense by giving a polynomial-time reduction from Set Splitting. Finally, we consider approximation algorithms. We present an $O(|L| \cdot |\mathcal{T}|^3)$-time algorithm that always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in $\mathcal{T}$ for any $\mathcal{T}$. (Our approximation algorithm can also be applied in the dense case when the input cannot be combined into a galled network without conflicts.) On the negative side, we show that there exist inputs for which any galled network can be consistent with at most a factor of 0.4883 of the rooted triplets. It is interesting to note that for *trees*, the corresponding bounds are known to be tight [7]; that is, there is a polynomial-time approximation algorithm which always constructs a tree consistent with at least $\frac{1}{3} \cdot |\mathcal{T}|$ of the rooted triplets in $\mathcal{T}$, and there exist some inputs for which no tree can achieve a factor higher than $\frac{1}{3} \cdot |\mathcal{T}|$.

1.1. Definitions and notation. A *phylogenetic tree* is a binary, rooted, unordered tree whose leaves are distinctly labeled. A *phylogenetic network* is a generalization of a phylogenetic tree formally defined as a rooted, connected, directed acyclic graph in which (1) exactly one node has indegree 0 (the *root*), and all other nodes have indegree 1 or 2; (2) all nodes with indegree 2 (referred to as *hybrid nodes*) have outdegree 1, and all other nodes have outdegree 0 or 2; and (3) all nodes with outdegree 0 (the *leaves*) are distinctly labeled. For any phylogenetic network N, let $\mathcal{U}(N)$ be the undirected graph obtained from N by replacing each directed edge by an undirected edge. N is said to be a *galled phylogenetic network* (*galled network*, for short) if all cycles in $\mathcal{U}(N)$ are node-disjoint. Galled networks are also known in the literature as *topologies with independent recombination events* [24], *galled-trees* [8], *gt-networks* [19], and *level*-1 *phylogenetic networks* [4, 15].

A phylogenetic tree with exactly three leaves is called a *rooted triplet*. The unique rooted triplet on a leaf set $\{x, y, z\}$ in which the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z (or, equivalently, where the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of y and z) is denoted by $(\{x, y\}, z)$. For any phylogenetic network N, a rooted triplet $t = (\{x, y\}, z)$ is said to be *consistent with* N if t is an embedded subtree of N (i.e., if a lowest common ancestor of x and y in N is a proper descendant of a lowest common ancestor of x and z in N), and a set $\mathcal{T}$ of rooted triplets is said to be *consistent with* N if every rooted triplet in $\mathcal{T}$ is consistent with N.

Denote the set of leaves in any phylogenetic network N by $\Lambda(N)$ and for any set $\mathcal{T}$ of rooted triplets, define $\Lambda(\mathcal{T}) = \bigcup_{t_i \in \mathcal{T}} \Lambda(t_i)$. A set $\mathcal{T}$ of rooted triplets is *dense* if

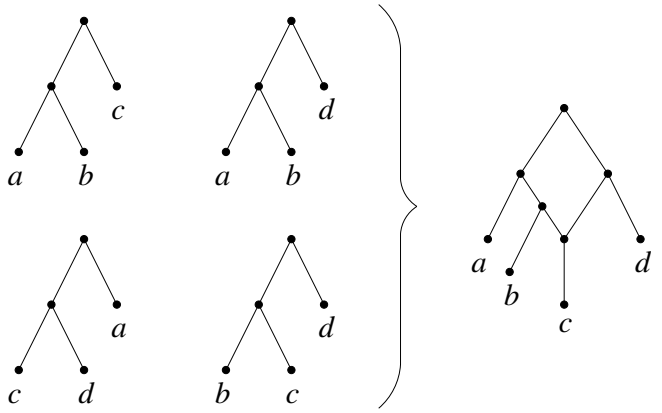

FIG. 1. *A dense set $\mathcal{T}$ of rooted triplets with leaf set $\{a, b, c, d\}$ and a galled phylogenetic network which is consistent with $\mathcal{T}$. Note that this solution is not unique.*

for each $\{x, y, z\} \subseteq \Lambda(\mathcal{T})$, at least one of $(\{x, y\}, z)$, $(\{x, z\}, y)$, and $(\{y, z\}, x)$ belongs to $\mathcal{T}$. If $\mathcal{T}$ is dense, then $|\mathcal{T}| = \Theta(|\Lambda(\mathcal{T})|^3)$. Furthermore, for any set $\mathcal{T}$ of rooted triplets and $L' \subseteq \Lambda(\mathcal{T})$, define $\mathcal{T} \,|\, L'$ as the subset of $\mathcal{T}$ consisting of all rooted triplets t with $\Lambda(t) \subseteq L'$. The problem we consider here is the following: Given a set $\mathcal{T}$ of rooted triplets, output a galled network N with $\Lambda(N) = \Lambda(\mathcal{T})$ such that N and $\mathcal{T}$ are consistent if such a network exists; otherwise, output *null*. See Figure 1 for an example. Throughout this paper, we write $L = \Lambda(\mathcal{T})$ and $n = |L|$.

To describe our algorithms, we need the following additional terminology. Let N be a phylogenetic network. We call nodes with indegree 2 *hybrid nodes* and their parent edges *hybrid edges*. Let h be a hybrid node in N. Every ancestor s of h such that h can be reached using two disjoint directed paths starting at the children of s is called a *split node of h*. If s is a split node of h, then any path starting at s and ending at h is called a *merge path of h*, and any path starting at a child of s and ending at a parent of h is called a *clipped merge path of h*. From the above, it follows that in a galled network, each split node is a split node of exactly one hybrid node, and each hybrid node has exactly one split node.

Let N be a galled network. For any node u in N, $N[u]$ denotes the subnetwork of N rooted at u, i.e., the minimal subgraph of N which includes all nodes and directed edges of N reachable from u. $N[u]$ is called a *side network* of N if there exists a merge path P in N such that u does not belong to P but u is a child of a node belonging to P. In this case, $N[u]$ is also said to be *attached to P*. N is called a *simple phylogenetic network* (or *simple network*) if N has exactly one hybrid node h, the root node of N is the split node of h, and every side network of N is a leaf. For example, the galled network on the right in Figure 1 is a simple network. For any simple network N, denote the leaf attached to the hybrid node in N by $hl(N)$.

1.2. Motivation. Phylogenetic networks are used by scientists to describe evolutionary relationships that do not fit the traditional models in which evolution is assumed to be treelike. Evolutionary events such as horizontal gene transfer or hybrid speciation (often referred to as *recombination events*) cannot be adequately represented in a single tree [8, 9, 19, 22, 24] but can be modeled in a phylogenetic network as internal nodes having more than one parent. Galled networks are an important type of phylogenetic network and have attracted special attention in the

literature [4, 8, 19, 24] due to their biological significance (see [8] for a discussion) and their simple, almost treelike, structure. When the number of recombination events is limited and most of them have occurred recently, a galled network may suffice to accurately describe the evolutionary process under study [8].

A challenge in the field of phylogenetics is to develop efficient and reliable methods for constructing and comparing phylogenetic networks. For example, to construct a meaningful phylogenetic network for a large subset of the human population (which may subsequently be used to help locate regions in the genome associated with some observable trait indicating a particular disease) in the future, efficient algorithms are crucial because the input can be expected to be very large. The motivation behind the rooted triplet approach taken in this paper is that a highly accurate tree for each cardinality-three subset of the leaf set can be obtained through maximum likelihood-based methods such as [3] or Sibley–Ahlquist-style DNA-DNA hybridization experiments (see [17]). Hence, the algorithms presented in [15] and here can be used as the merging step in a divide-and-conquer approach for constructing phylogenetic networks analogous to the quartet method paradigm for inferring unrooted phylogenetic trees [16, 18] and other supertree methods (see [10, 21] and the references therein). We consider dense input sets in particular since this case can be solved in polynomial time.

1.3. Related work. Aho et al. [1] gave an $O(|\mathcal{T}| \cdot n)$-time algorithm for determining whether a given set $\mathcal{T}$ of rooted triplets on n leaves is consistent with some rooted, distinctly leaf-labeled tree, and, if so, returning such a tree. Henzinger, King, and Warnow [10] improved its running time to $\min\{O(|\mathcal{T}| \cdot n^{0.5}), O(|\mathcal{T}| + n^2 \log n)\}$; in fact, replacing the deterministic algorithm for dynamic graph connectivity employed by Henzinger, King, and Warnow, with a more recent one due to Holm, de Lichtenberg, and Thorup [11] yields a running time of $\min\{O(|\mathcal{T}| \cdot \log^2 n), O(|\mathcal{T}| + n^2 \log n)\}$ [14]. Gąsieniec et al. [6] studied a variant of the problem for *ordered* trees. Ng and Wormald [21] considered the problem of constructing *all* rooted, unordered trees distinctly leaf-labeled by $\Lambda(\mathcal{T})$ that are consistent with $\mathcal{T}$.

If two or more of the rooted triplets are in conflict, i.e., contain contradicting branching information, the algorithm of Aho et al. returns a null tree. However, this is not very practical in certain applications. For example, in the context of constructing a phylogenetic tree from a set of rooted triplets, some errors may occur in the input when the rooted triplets are based on data obtained experimentally, yet a nonnull tree is still required. In this case, one can try to construct a tree consistent with the maximum number of rooted triplets in the input [2, 6, 7, 13, 25], or a tree with as many leaves from $\Lambda(\mathcal{T})$ as possible which is consistent with all input rooted triplets involving these leaves only [14]. Although the former problem is NP-hard [2, 13, 25], Gąsieniec et al. [7] showed that it has a polynomial-time approximation algorithm that outputs a distinctly leaf-labeled tree consistent with at least $\frac{1}{3}$ of the given rooted triplets, which is a tight bound in the sense that there exist inputs $\mathcal{T}$ such that any distinctly leaf-labeled tree can be consistent with at most $\frac{1}{3}$ of the rooted triplets in $\mathcal{T}$.[2]

The problem studied in this paper was introduced in [15]. The main result of [15] is an exact $O(|\mathcal{T}|^2)$-time algorithm for the dense case. Reference [15] also showed that if no restrictions are placed on the structure of the output phylogenetic network (i.e., if nongalled networks are allowed), then the problem always has a solution which

[2] On the other hand, if the optimal solution contains a large fraction of the input rooted triplets, another approximation presented in [7] (based on minimum cuts in the auxiliary graph introduced by Aho et al. [1]) gives a better approximation factor.

can be easily obtained from any given sorting network for n elements. Nakhleh et al. [19] gave an $O(n^2)$-time algorithm for the related problem of determining if two given phylogenetic trees T_1 and T_2 with identical leaf sets can be combined into a galled network containing both T_1 and T_2 as embedded subtrees and, if so, constructing one with the smallest possible number of hybrid nodes, where n is the number of leaves (in fact, this is equivalent to inferring a galled network consistent with the set of all rooted triplets which are embedded subtrees of T_1 or T_2). They also studied the case where T_1 and T_2 may contain errors but only one hybrid node is allowed. Huson et al. [12] considered a similar problem for constructing an *unrooted* phylogenetic network from a set of unrooted, distinctly leaf-labeled trees.

1.4. Organization of the paper. In section 2, we present a new algorithm called *SimpleNetworks* for computing all simple phylogenetic networks consistent with a given dense set $\mathcal{T}$ of rooted triplets in $O(n^3)$ time. This algorithm is used by our main algorithm *FastGalledNetwork* in section 3 to construct a galled network consistent with a given dense set $\mathcal{T}$ of rooted triplets, if one exists, in optimal $O(n^3)$ time. In section 4, we prove that the problem becomes NP-hard if we remove the requirement that $\mathcal{T}$ forms a dense set. Next, in section 5.1, we show that for every positive integer n, there exists some set $\mathcal{T}$ of rooted triplets with $|\Lambda(T)| = n$ such that any galled network can be consistent with at most $0.4883 \cdot |\mathcal{T}|$ of the rooted triplets in $\mathcal{T}$. On the other hand, we give an $O(n \cdot |\mathcal{T}|^3)$-time algorithm in section 5.2 that constructs a galled network guaranteed to be consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in $\mathcal{T}$ for any input $\mathcal{T}$.

2. Constructing all simple phylogenetic networks when $\mathcal{T}$ is dense. In this section, we describe an algorithm called *SimpleNetworks* for inferring all simple phylogenetic networks consistent with a given dense set $\mathcal{T}$ of rooted triplets in $O(n^3)$ time, where $L = \Lambda(\mathcal{T})$ and $n = |L|$. This algorithm is later used by our main algorithm in section 3. Below, for any $L' \subseteq L$, $\mathcal{G}(L')$ denotes *the auxiliary graph for* L' (originally defined by Aho et al. [1]), which is the undirected graph with vertex set L' and edge set $E(L')$, where for each $(\{i, j\}, k) \in \mathcal{T} \mid L'$, the edge $\{i, j\}$ is included in $E(L')$.

SimpleNetworks assumes that $n \geq 3$, $\mathcal{T}$ is dense, and $\mathcal{G}(L)$ is connected.

For any simple network N, define $A(N)$ and $B(N)$ to be the sets of leaves attached to the two clipped merge paths in N, where we require without loss of generality that $A(N)$ is nonempty. If both $A(N)$ and $B(N)$ are nonempty, then N is called *nonskew*; if $A(N)$ is nonempty and $B(N)$ is empty, then N is called *skew*.

Algorithm *SimpleNetworks* is listed in Figure 2. It calls two procedures named *Non-SkewSimpleNetworks* and *SkewSimpleNetworks* that find all valid nonskew simple networks and all valid skew simple networks, respectively. Then it returns their union. In the next two subsections, we show how to implement each of these two procedures to run in $O(n^3)$ time. We thus obtain Theorem 1.

THEOREM 1. *The set of all simple networks consistent with a given dense set of rooted triplets and leaf-labeled by L can be constructed in $O(n^3)$ time.*

The next two lemmas are used in sections 2.1 and 2.2. A *caterpillar tree* is a rooted tree such that every internal node has at most one child which is not a leaf (see, e.g., [2]). Recall that for any simple network N, we denote the leaf attached to the hybrid node in N by $hl(N)$.

LEMMA 1. *Suppose N is a simple phylogenetic network that is consistent with a set $\mathcal{T}$ of rooted triplets. Let N' be the graph obtained from N by deleting the root node of N, the hybrid node of N, and $hl(N)$ together with all their incident edges, and then, for every node with outdegree 1 and indegree less than 2, contracting its*

Algorithm *SimpleNetworks*
Input: A dense set $\mathcal{T}$ of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.
Output: The set of all simple phylogenetic networks with leaf set L which are consistent with $\mathcal{T}$.
1 Let $\mathcal{N}_1 =$ *Non-SkewSimpleNetworks*$(\mathcal{T})$.
2 Let $\mathcal{N}_2 =$ *SkewSimpleNetworks*$(\mathcal{T})$.
3 **return** $\mathcal{N}_1 \cup \mathcal{N}_2$.
End *SimpleNetworks*

FIG. 2. *Constructing all simple phylogenetic networks.*

outgoing edge. If N is nonskew, then N' consists of two binary caterpillar trees which are consistent with $\mathcal{T} \mid A(N)$ and $\mathcal{T} \mid B(N)$, respectively; if N is skew, then N' is a binary caterpillar tree which is consistent with $\mathcal{T} \mid A(N)$.

LEMMA 2. [15] *Let $\mathcal{T}$ be a dense set of rooted triplets and let L be the leaf set of $\mathcal{T}$. There is at most one rooted, unordered tree distinctly leaf-labeled by L which is consistent with $\mathcal{T}$. Furthermore, if such a tree exists, then it must be binary.*

2.1. Constructing all nonskew simple phylogenetic networks. Let U be an undirected, connected graph. Any partition (X, Y, Z) of the vertex set of U is called a *nonskew leaf partition in U* if $|X| \geq 1$, $|Y| = 1$, $|Z| \geq 1$, and in U the following holds: (1) X and Z form two cliques, and (2) there is no edge between a vertex in X and a vertex in Z. Any two nonskew leaf partitions of the form (X, Y, Z) and (Z, Y, X) are considered to be equivalent.

LEMMA 3. *Let $\mathcal{T}$ be a dense set of rooted triplets and let L be the leaf set of $\mathcal{T}$. If N is a nonskew simple phylogenetic network with leaf set L that is consistent with $\mathcal{T}$, then $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$.*

Proof. Consider any $a_i, a_j \in A(N)$ with $i \neq j$ and let b be an arbitrary element in $B(N)$. N is consistent with $\mathcal{T}$, implying that $(\{a_i, b\}, a_j) \notin \mathcal{T}$ and $(\{a_j, b\}, a_i) \notin \mathcal{T}$. Since $\mathcal{T}$ is dense, $(\{a_i, a_j\}, b)$ must belong to $\mathcal{T}$; i.e., there is an edge (a_i, a_j) in $\mathcal{G}(L)$. Hence, $A(N)$ forms a clique in $\mathcal{G}(L)$. In the same way, $B(N)$ forms a clique in $\mathcal{G}(L)$. Moreover, $\mathcal{T}$ cannot contain any rooted triplet of the form $(\{a, b\}, x)$ where $a \in A(N)$, $b \in B(N)$, $x \in L$, and thus there are no edges in $\mathcal{G}(L)$ between leaves in $A(N)$ and leaves in $B(N)$. However, $\mathcal{G}(L)$ is connected, which means that there must be at least one edge from $A(N)$ to the leaf $hl(N)$ and at least one edge from $B(N)$ to $hl(N)$. By definition, $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$. ∎

By Lemmas 1 and 3, if N is a nonskew simple network with leaf set L that is consistent with $\mathcal{T}$, then $(A(N), hl(N), B(N))$ forms a nonskew leaf partition in $\mathcal{G}(L)$ and $\mathcal{T} \mid A(N)$ and $\mathcal{T} \mid B(N)$ are consistent with two binary caterpillar trees. Algorithm *Non-SkewSimpleNetworks*, shown in Figure 3, uses these implications to efficiently construct all nonskew simple networks with leaf set L that are consistent with $\mathcal{T}$. The algorithm enumerates all nonskew leaf partitions in $\mathcal{G}(L)$, and for each such leaf partition P, tries to build binary caterpillar trees for subsets of L induced by P (Lemma 2 ensures that for any dense subset $\mathcal{T}'$ of $\mathcal{T}$, if $\mathcal{T}'$ is consistent with a caterpillar tree, then it is uniquely determined, and so the algorithm of Aho et al. [1] can find it) and if successful, then combines the caterpillar trees in accordance with Lemma 1 to obtain all possible valid simple networks. Lemmas 1 and 3 guarantee that this approach will discover every valid simple network. However, it may also yield some simple networks which are not consistent with $\mathcal{T}$; hence, before including any

Algorithm *Non-SkewSimpleNetworks*
Input: A dense set $\mathcal{T}$ of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.
Output: The set of all nonskew simple phylogenetic networks with leaf set L which are consistent with $\mathcal{T}$.
1 Set $\mathcal{N}_1 = \emptyset$.
2 Construct $\mathcal{G}(L)$ and compute all nonskew leaf partitions in $\mathcal{G}(L)$.
3 for every nonskew leaf partition (X, Y, Z) in $\mathcal{G}(L)$ **do**
3.1 Let $T_X = \mathit{BuildTree}(\mathcal{T} \mid X)$ and $T_Z = \mathit{BuildTree}(\mathcal{T} \mid Z)$.
3.2 **if** T_X and T_Z are binary caterpillar trees **then**
Make the roots of T_X and T_Z children of a new root node, create a new hybrid node H with a child labeled by the leaf in Y, and construct at most four nonskew simple networks by attaching H to one of T_X's bottommost leaves' parent edges and one of T_Z's bottommost leaves' parent edges in all possible ways.
For each obtained network N, if N is consistent with $\mathcal{T}$ then let $\mathcal{N}_1 = \mathcal{N}_1 \cup \{N\}$.
endfor
4 return $\mathcal{N}_1$.
End *Non-SkewSimpleNetworks*

FIG. 3. *Constructing all nonskew simple phylogenetic networks.*

constructed network N in the final solution set $\mathcal{N}_1$, *Non-SkewSimpleNetworks* verifies if N is consistent with $\mathcal{T}$.

For any $L' \subseteq L$ with $|L'| \geq 3$, $\mathit{BuildTree}(\mathcal{T} \mid L')$ refers to the fast implementation of the algorithm of Aho et al. applied to $\mathcal{T} \mid L'$ (we may assume it returns *null* if it fails). For $|L'| < 3$, the set $\mathcal{T} \mid L'$ is empty and we simply let $\mathit{BuildTree}(\mathcal{T} \mid L')$ return a tree with the one or two leaves in L'. The running time of $\mathit{BuildTree}(\mathcal{T} \mid L')$ is $\min\{O(|\mathcal{T}| \cdot \log^2 n), O(|\mathcal{T}| + n^2 \log n)\}$ (see section 1.3).

We now derive an upper bound on the running time of *Non-SkewSimpleNetworks*.

LEMMA 4. *For any undirected, connected graph U with n vertices, all nonskew leaf partitions in U can be computed in $O(n^3)$ time.*

Proof. To find all nonskew leaf partitions in U, test each of the n vertices to see if its removal divides U into two disjoint, nonempty cliques. Each test can be done in $O(n^2)$ time by depth-first search; thus this takes a total of $O(n^3)$ time. □

LEMMA 5. *Any undirected, connected graph U has at most two nonskew leaf partitions.*

Proof. First observe that for any two different nonskew leaf partitions $(X, \{h\}, Z)$ and $(X', \{h'\}, Z')$ in U, we have $h \neq h'$. Moreover, h and h' are neighbors in U (otherwise, for any two neighbors x', z' of h' such that $x' \in X'$ and $z' \in Z'$, we have $x' \neq h$ and $z' \neq h$, and then all of h', x', z' must belong to one of X and Z while there is no edge between x' and z', which is a contradiction).

Now, suppose $(X, \{h\}, Z)$ is a nonskew leaf partition in U and consider any other nonskew leaf partition $(X', \{h'\}, Z')$ in U. Either $h' \in X$ or $h' \in Z$. If $h' \in X$, then h can have no neighbors in X other than h' (otherwise, there would be an edge between X' and Z') and, furthermore, U cannot have any nonskew leaf partition of the form $(X'', \{h''\}, Z'')$ where $h'' \in Z$ because h' and h'' are not neighbors. This also holds in the case $h' \in Z$. This proves that U has at most two nonskew leaf partitions. □

Algorithm *SkewSimpleNetworks*
Input: A dense set $\mathcal{T}$ of rooted triplets with a leaf set L such that $\mathcal{G}(L)$ consists of one connected component.
Output: The set of all skew simple phylogenetic networks with leaf set L which are consistent with $\mathcal{T}$.
1 Set $\mathcal{N}_2 = \emptyset$.
2 Construct $\mathcal{D}$.
3 **for** every $x \in L$ **do**
3.1 Let $Q = BuildCaterpillar(\mathcal{D} \mid L')$, where $L' = L \setminus \{x\}$.
3.2 **if** $Q \neq null$ **then**
Make the root of Q a child of a new root node r, create a new hybrid node H with a child labeled by x, add an edge from r to H, and construct two skew simple networks by attaching H to each one of Q's two bottommost edges. For each obtained network N, if N is consistent with all rooted triplets in $\mathcal{T}$ involving x then let $\mathcal{N}_2 = \mathcal{N}_2 \cup \{N\}$.
endfor
4 **return** $\mathcal{N}_2$.
End *SkewSimpleNetworks*

FIG. 4. *Constructing all skew simple phylogenetic networks.*

Next, if N is any galled network with n leaves, then the total number of nodes in N is $O(n)$ by Lemma 3 in [4]. By traversing the $O(n)$ nodes in N in a bottom-up order while keeping track of each node's $O(n)$ descendants and updating a table containing all $O(n^2)$ node pairs' lowest common ancestors, we obtain the following.

LEMMA 6. *Let N be a galled network with n leaves. After $O(n^2)$ time preprocessing, we can check if any given rooted triplet is consistent with N in $O(1)$ time.*

THEOREM 2. *The time complexity of Algorithm Non-SkewSimpleNetworks is $O(n^3)$.*

Proof. Steps 1 and 4 require $O(1)$ time. Step 2 can be performed in $O(n^3)$ time by a single scan of $\mathcal{T}$ and by applying Lemma 4. Moreover, $\mathcal{G}(L)$ has at most two nonskew leaf partitions according to Lemma 5. Therefore, steps 3.1 and 3.2 are carried out at most two times each. Step 3.1 takes $O(|\mathcal{T}| + n^2 \log n) = O(n^3)$ time with the fast implementation of *BuildTree* by Henzinger, King, and Warnow [10]. Every time step 3.2 is performed, the algorithm constructs at most four nonskew simple networks and tests each of them for inclusion in $\mathcal{N}$, which after $O(n^2)$ time preprocessing takes $O(n^3)$ time using Lemma 6 since $|\mathcal{T}| = O(n^3)$. Hence, the total running time is $O(n^3)$. ∎

2.2. Constructing all skew simple phylogenetic networks. To obtain all skew simple networks with leaf set L consistent with $\mathcal{T}$, Algorithm *SkewSimpleNetworks* in Figure 4 tries all ways to remove one leaf x from L and construct a binary caterpillar tree consistent with all rooted triplets not involving x using a procedure named *BuildCaterpillar*. For each such caterpillar Q, it forms two candidate skew simple networks by letting the root of Q be a child of a new split node with a hybrid node H such that H has a child labeled by x and H is attached to one of Q's two bottommost edges. (By Lemma 1, every skew simple network with leaf set L that is consistent with $\mathcal{T}$ must have this structure.) Then, each candidate skew simple network is checked to see if it is consistent with all rooted triplets in $\mathcal{T}$ involving x (by the above, it is always consistent with the rest); if yes, then it is included in the solution set $\mathcal{N}_2$.

The procedure *BuildCaterpillar* uses a graph $\mathcal{D}$, defined as follows. Given a set $\mathcal{T}$ of rooted triplets with leaf set L, let $\mathcal{D}$ be the directed graph with vertex set L such that there is a directed edge (x, y) if and only if $\mathcal{T}$ contains at least one rooted triplet of the form $(\{y, z\}, x)$, where $z \in L$. For any $L' \subseteq L$, let $\mathcal{D} \mid L'$ be the subgraph of $\mathcal{D}$ in which all vertices not in L' and their incident edges have been deleted. $\mathcal{D} \mid L'$ is acyclic if and only if there exists a binary caterpillar tree consistent with $\mathcal{T} \mid L'$.

For any $L' \subseteq L$, *BuildCaterpillar*$(\mathcal{D} \mid L')$ returns a binary caterpillar tree with leaf set L' which is consistent with $\mathcal{T} \mid L'$ if such a tree exists, and *null* otherwise, by the following method. If $\mathcal{D} \mid L'$ has a cycle, then return *null*. Else, do a topological sort of $\mathcal{D} \mid L'$ to find a linear ordering $\mathcal{O}$ of L' and return a binary caterpillar tree whose leaves are labeled in order of increasing distance from the root according to $\mathcal{O}$. Since $\mathcal{T}$ is dense, $\mathcal{O}$ is uniquely determined except for its last two elements which may be interchanged arbitrarily.

THEOREM 3. *The time complexity of Algorithm SkewSimpleNetworks is* $O(n^3)$.

Proof. Steps 1 and 4 require $O(1)$ time, and step 2 can be performed in $O(n^3)$ time by a single scan of $\mathcal{T}$. Steps 3.1 and 3.2 are carried out n times. Each call to *BuildCaterpillar* in step 3.1 takes $O(n^2)$ time since a topological sort can be done in $O(n^2)$ time. In step 3.2, to construct two networks and test them against the $O(n^2)$ rooted triplets involving x takes $O(n^2)$ time by Lemma 6. Hence, the total running time is $O(n^3)$. ∎

3. An exact algorithm for inferring a galled phylogenetic network from a dense set of rooted triplets with optimal running time. Here, we present our algorithm *FastGalledNetwork* for constructing a galled network consistent with a given dense set $\mathcal{T}$ of rooted triplets if such a network exists. Its running time is $O(n^3)$, where $n = |L|$ and L denotes the leaf set of $\mathcal{T}$, which is optimal since the size of the input is $\Theta(n^3)$ when $\mathcal{T}$ is dense.

In section 3.1, we give an algorithm named *ComputeSNTree* which computes the so-called SN-tree for $\mathcal{T}$ in $O(n^3)$ time. Then, in section 3.2, we describe *FastGalledNetwork*. It uses *ComputeSNTree* as well as *SimpleNetworks* from section 2 to construct a galled network consistent with $\mathcal{T}$ (if one exists) from the SN-tree for $\mathcal{T}$. In sections 3.1 and 3.2 below, we assume $\mathcal{T}$ is dense.

3.1. Computing the SN-tree. For any $X \subseteq L$, the set $SN(X)$ is defined recursively as $SN(X \cup \{c\})$ if there exist some $x, x' \in X$ and $c \in L \setminus X$ such that $(\{x, c\}, x') \in \mathcal{T}$, and as X otherwise. SN-sets were introduced in [15]. Intuitively, each SN-set is a subset of L which will form the leaf set of one subnetwork in the final solution. The SN-sets satisfy the following important property.

LEMMA 7 (see [15]). *If $\mathcal{T}$ is dense, then for any $A, B \subseteq L$, $SN(A) \cap SN(B)$ equals $\emptyset$, $SN(A)$, or $SN(B)$.*

Reference [15] showed how to compute $SN(\{a, b\})$ for any $a, b \in L$ in $O(n^3)$ time; that approach therefore takes $O(n^5)$ time to compute $SN(\{a, b\})$ for *all* $a, b \in L$. This section presents a faster method for implicitly computing all SN-sets of this form when $\mathcal{T}$ is dense, which requires only $O(n^3)$ time. The algorithm (*ComputeSNTree*) is listed in Figure 5. Given a dense $\mathcal{T}$, it builds a rooted tree called *the SN-tree for $\mathcal{T}$* which encodes all SN-sets so that $SN(X)$ for any $X \subseteq L$ can be retrieved efficiently.

In the first step, *ComputeSNTree* constructs a directed graph $G_{\mathcal{T}}$ with vertex set $V(G_{\mathcal{T}})$ and edge set $E(G_{\mathcal{T}})$. $V(G_{\mathcal{T}})$ is defined as $\{v_{\{a,b\}} \mid a, b \in L\}$, where $v_{\{a,a\}}$ for any $a \in L$ is denoted by $v_{\{a\}}$ for short, and $E(G_{\mathcal{T}})$ is $\{(v_{\{a,c\}}, v_{\{a,b\}}), (v_{\{a,c\}}, v_{\{b,c\}}), (v_{\{b,c\}}, v_{\{a,b\}}), (v_{\{b,c\}}, v_{\{a,c\}}) \mid (\{a, b\}, c) \in \mathcal{T}\} \bigcup \{(v_{\{a,b\}}, v_{\{a\}}), (v_{\{a,b\}}, v_{\{b\}}) \mid a, b \in$

Algorithm *ComputeSNTree*
Input: A dense set $\mathcal{T}$ of rooted triplets with a leaf set L.
Output: The SN-tree $R_{\mathcal{T}}$ for $\mathcal{T}$.
1 Construct the directed graph $G_{\mathcal{T}}$.
2 Compute the set $\mathcal{C}$ of strongly connected components of $G_{\mathcal{T}}$ and then construct $G'_{\mathcal{T}}$ for $\mathcal{T}$.
3 Construct the SN-tree $R_{\mathcal{T}}$ and **return** $R_{\mathcal{T}}$.
End *ComputeSNTree*

FIG. 5. *Computing the SN-tree for a dense set* $\mathcal{T}$.

$L\}$. Note that $|V(G_{\mathcal{T}})| = O(n^2)$ and $|E(G_{\mathcal{T}})| = O(n^3)$. Before describing the remaining steps of *ComputeSNTree*, we first investigate the structure of $G_{\mathcal{T}}$ and the relationship between $G_{\mathcal{T}}$ and the SN-sets of the form $SN(\{a,b\})$.

LEMMA 8. *For every* $a,b,y,z \in L$, *if* $G_{\mathcal{T}}$ *contains a path from* $v_{\{a,b\}}$ *to* $v_{\{a,y\}}$ *and a path from* $v_{\{a,b\}}$ *to* $v_{\{a,z\}}$, *then* $G_{\mathcal{T}}$ *has a path from* $v_{\{a,b\}}$ *to* $v_{\{y,z\}}$.

Proof. If $|\{a,y,z\}| < 3$, then the lemma follows from the construction of $G_{\mathcal{T}}$. Otherwise, since $\mathcal{T}$ is dense, we have one of the following cases for $\{a,y,z\}$:

- Case (1): $(\{y,z\},a) \in \mathcal{T}$ or $(\{a,z\},y) \in \mathcal{T}$. Then $(v_{\{a,y\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}})$, and thus there is a path from $v_{\{a,b\}}$ to $v_{\{a,y\}}$ and then to $v_{\{y,z\}}$.
- Case (2): $(\{a,y\},z) \in \mathcal{T}$. Then $(v_{\{a,z\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}})$, and thus there is a path from $v_{\{a,b\}}$ to $v_{\{a,z\}}$ and then to $v_{\{y,z\}}$. ∎

LEMMA 9. *For every* $a,b,c \in L$, *if* $c \in SN(\{a,b\})$, *then there exists a directed path from* $v_{\{a,b\}}$ *to* $v_{\{a,c\}}$ *in* $G_{\mathcal{T}}$.

Proof. Define $SN_0(\{a,b\}) = \{a,b\}$ and for $\ell = 1,2,\ldots,n$, define $SN_\ell = \big\{x \in L \mid (\{y,x\},z) \in \mathcal{T}$ for some $y,z \in SN_{\ell-1}(\{a,b\})\big\}$. Note that $SN(\{a,b\}) = \bigcup_{i=0}^{n} SN_\ell(\{a,b\})$. We prove by induction that the following statement $P(\ell)$ is true for $\ell \in \{0,1,2,\ldots,n\}$.

$P(\ell)$: For every $x \in SN_\ell(\{a,b\})$, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,x\}}$ in $G_{\mathcal{T}}$.

When $\ell = 0$ (the base case), the statement follows trivially by the construction of $G_{\mathcal{T}}$.

Next, when $\ell > 0$, suppose the statement $P(\ell-1)$ is true; i.e., for every $w \in SN_{\ell-1}(\{a,b\})$, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,w\}}$ in $G_{\mathcal{T}}$. Consider any $x \in SN_\ell(\{a,b\})$. By the definition of SN_ℓ, there exist $y,z \in SN_{\ell-1}(\{a,b\})$ such that $(\{y,x\},z) \in \mathcal{T}$. Then $P(\ell-1)$ implies that there is a path from $v_{\{a,b\}}$ to $v_{\{a,y\}}$ and a path from $v_{\{a,b\}}$ to $v_{\{a,z\}}$, which means there exists a path from $v_{\{a,b\}}$ to $v_{\{y,z\}}$ according to Lemma 8. Moreover, $(v_{\{y,z\}}, v_{\{x,y\}}) \in E(G_{\mathcal{T}})$ because $(\{y,x\},z) \in \mathcal{T}$. Now, if $x = a$ or $y = a$, then $P(\ell)$ follows directly; therefore assume $x \neq a$ and $y \neq a$. Since $\mathcal{T}$ is dense, for the set $\{a,x,y\}$, there are two cases:

- Case (1): $(\{a,y\},x) \in \mathcal{T}$ or $(\{a,x\},y) \in \mathcal{T}$. Then $(v_{\{x,y\}}, v_{\{a,x\}}) \in E(G_{\mathcal{T}})$.
- Case (2): $(\{x,y\},a) \in \mathcal{T}$. Then $(v_{\{a,y\}}, v_{\{a,x\}}) \in E(G_{\mathcal{T}})$.

In both cases, there exists a path from $v_{\{a,b\}}$ to $v_{\{a,x\}}$, and thus $P(\ell)$ holds.

By induction, $P(\ell)$ is true for every $\ell \in \{0,1,2,\ldots,n\}$. Since c belongs to at least one set SN_ℓ, the lemma follows. ∎

LEMMA 10. *For every* $a,b,c,d \in L$, *if there is a directed path from* $v_{\{a,b\}}$ *to* $v_{\{c,d\}}$ *in* $G_{\mathcal{T}}$, *then* $SN(\{c,d\}) \subseteq SN(\{a,b\})$.

Proof. For any $e \in SN(\{c,d\})$, there is a directed path from $v_{\{c,d\}}$ to $v_{\{c,e\}}$ by Lemma 9. Since $E(G_{\mathcal{T}})$ contains the directed edge $(v_{\{c,e\}}, v_{\{e\}})$, and since there is a directed path from $v_{\{a,b\}}$ to $v_{\{c,d\}}$, this means there is a directed path from $v_{\{a,b\}}$ to

$v_{\{e\}}$. Without loss of generality, let the path be $v_{\{x_0,x_1\}}, v_{\{x_1,x_2\}}, v_{\{x_2,x_3\}}, \ldots, v_{\{x_{p-1},x_p\}}$, $v_{\{x_p\}}$, where $\{x_0, x_1\} = \{a, b\}$, and $x_p = e$. Then, based on the rooted triplets in $\mathcal{T}$ for the sets $\{x_0, x_1, x_2\}, \{x_1, x_2, x_3\}, \ldots, \{x_{p-2}, x_{p-1}, x_p\}$, we can deduce that $x_p \in SN(\{a, b\})$. Thus, $e \in SN(\{a, b\})$, so we have just shown that $SN(\{c, d\}) \subseteq SN(\{a, b\})$. ∎

COROLLARY 1. *For any two nodes $v_{\{a,b\}}$ and $v_{\{c,d\}}$ on a directed cycle in $G_{\mathcal{T}}$, $SN(\{a, b\}) = SN(\{c, d\})$.*

By the above, computing $SN(\{a, b\})$ for any $a, b \in L$ is equivalent to finding all nodes of the form $v_{\{c\}}$ reachable from $v_{\{a,b\}}$. Let the set of all strongly connected components of $G_{\mathcal{T}}$ be $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$. By Corollary 1, $SN(\{a, b\}) = SN(\{c, d\})$ if $v_{\{a,b\}}$ and $v_{\{c,d\}}$ are in the same C_i. So, we define $SN(C_i)$ as $SN(\{a, b\})$ for any $v_{\{a,b\}} \in C_i$. The set $\mathcal{C}$ has the following properties.

LEMMA 11. *For every $c \in L$, $\{v_{\{c\}}\} \in \mathcal{C}$. Moreover, for every $i \neq j$, $SN(C_i) \neq SN(C_j)$.*

Proof. Since $v_{\{c\}}$ has no outgoing edge, $\{v_{\{c\}}\}$ is a strongly connected component in $G_{\mathcal{T}}$ and therefore belongs to $\mathcal{C}$.

To prove the second statement, suppose for the sake of contradiction that there exist strongly connected components C_i, C_j with $i \neq j$ such that $SN(C_i) = SN(C_j)$. Take any $v_{\{w,x\}} \in C_i$ and $v_{\{y,z\}} \in C_j$. Since $y, z \in SN(C_j) = SN(C_i) = SN(\{w, x\})$, it follows from Lemma 9 that $G_{\mathcal{T}}$ has a path from $v_{\{w,x\}}$ to $v_{\{w,y\}}$ and a path from $v_{\{w,x\}}$ to $v_{\{w,z\}}$, and hence a path from $v_{\{w,x\}}$ to $v_{\{y,z\}}$ by Lemma 8. Symmetrically, $G_{\mathcal{T}}$ contains a path from $v_{\{y,z\}}$ to $v_{\{w,x\}}$. But then $C_i \cup C_j$ must be a strongly connected component of $G_{\mathcal{T}}$, and we have arrived at a contradiction. ∎

In the second step of *ComputeSNTree*, we compute $\mathcal{C}$ and then let $G'_{\mathcal{T}}$ be the directed graph with vertex set $V(G'_{\mathcal{T}}) = \mathcal{C}$ and edge set $E(G'_{\mathcal{T}}) = \{(C_i, C_j) \mid$ there exists some $(v_{\{w,x\}}, v_{\{y,z\}}) \in E(G_{\mathcal{T}})$ where $v_{\{w,x\}} \in C_i$ and $v_{\{y,z\}} \in C_j\}$. Note that $G'_{\mathcal{T}}$ is a directed acyclic graph. From $G'_{\mathcal{T}}$, construct a graph $R_{\mathcal{T}}$ with $V(R_{\mathcal{T}}) = \mathcal{C}$ and $E(R_{\mathcal{T}}) = \{(C_i, C_j) \in E(G'_{\mathcal{T}}) \mid$ there exists no path of length at least 2 from C_i to C_j in $G'_{\mathcal{T}}\}$. Finally, return $R_{\mathcal{T}}$. The next two lemmas show that $R_{\mathcal{T}}$ is indeed a tree.

LEMMA 12. *There is only one node in $G'_{\mathcal{T}}$ with indegree 0.*

Proof. Suppose $G'_{\mathcal{T}}$ has two different nodes r, s with indegree 0. Denote the two strongly connected components in $G_{\mathcal{T}}$ which correspond to r and s in $R_{\mathcal{T}}$ by C_r and C_s, respectively. Let $v_{\{a,b\}}$ be any node in C_r and let $v_{\{c,d\}}$ be any node in C_s. Clearly, $a \neq b$ and $c \neq d$ since any node of the form $v_{\{a\}}$ belongs to a strongly connected component consisting only of $v_{\{a\}}$ and therefore cannot have indegree 0 by the construction of $G_{\mathcal{T}}$. Consider the three possible rooted triplets with leaf set $\{a, b, c\}$ (of which at least one belongs to $\mathcal{T}$ since $\mathcal{T}$ is dense). By the definition of $E(G_{\mathcal{T}})$, there will always be at least one edge ending at $v_{\{a,b\}}$. Since r has indegree 0 in $G'_{\mathcal{T}}$, this implies that (1) at least one of $v_{\{a,c\}}$ and $v_{\{b,c\}}$ must be in C_r. In the same way, we see that (2) at least one of $v_{\{a,d\}}$ and $v_{\{b,d\}}$ is in C_r; (3) at least one of $v_{\{a,c\}}$ and $v_{\{a,d\}}$ is in C_s; and (4) at least one of $v_{\{b,c\}}$ and $v_{\{b,d\}}$ is in C_s.

Assume without loss of generality that $v_{\{a,c\}} \in C_r$. Then (3) yields $v_{\{a,d\}} \in C_s$, and thus we have $v_{\{b,d\}} \in C_r$ by (2), and then $v_{\{b,c\}} \in C_s$ by (4). There are three cases:

- Case (1): $(\{a, b\}, c) \in T$. Then the two edges $(v_{\{a,c\}}, v_{\{b,c\}})$ and $(v_{\{b,c\}}, v_{\{a,c\}})$ in $E(G_{\mathcal{T}})$ imply that C_r is reachable from C_s in $G'_{\mathcal{T}}$ and vice versa.
- Case (2): $(\{a, c\}, b) \in T$. Then $(v_{\{a,b\}}, v_{\{b,c\}}), (v_{\{b,c\}}, v_{\{a,b\}}) \in E(G_{\mathcal{T}})$ imply that C_r is reachable from C_s in $G'_{\mathcal{T}}$ and vice versa.
- Case (3): $(\{b, c\}, a) \in T$. Then $(v_{\{a,b\}}, v_{\{b,c\}}) \in E(G_{\mathcal{T}})$, and thus C_s is reachable from C_r in $G'_{\mathcal{T}}$. Next, by considering all possible rooted triplets on

$\{a, c, d\}$, we see that $(v_{\{c,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$; $(v_{\{a,c\}}, v_{\{c,d\}}), (v_{\{c,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$; or $(v_{\{a,c\}}, v_{\{a,d\}}), (v_{\{a,d\}}, v_{\{a,c\}}) \in E(G_{\mathcal{T}})$, which means that C_r is always reachable from C_s in $G'_{\mathcal{T}}$.

Every case contradicts that C_r and C_s are disjoint strongly connected components. Hence, $G'_{\mathcal{T}}$ can only have one node with indegree 0. ∎

LEMMA 13. *$R_{\mathcal{T}}$ is a tree with no nodes having outdegree 1. Its set of leaves is $\{\{v_{\{c\}}\} \mid c \in L\}$.*

Proof. From Lemma 12 and by the construction of $R_{\mathcal{T}}$, it follows that $R_{\mathcal{T}}$ has only one node with indegree 0, i.e., only one root node.

Now, suppose $R_{\mathcal{T}}$ is not a tree. Then there exists a node C_j in $R_{\mathcal{T}}$ with at least two parents, say C_i and $C_{i'}$. By definition, $(C_i, C_j), (C_{i'}, C_j) \in E(G'_{\mathcal{T}})$ and by Lemma 10, $SN(C_j) \subseteq SN(C_i)$ as well as $SN(C_j) \subseteq SN(C_{i'})$. Next, by Lemmas 7 and 11, we have either $SN(C_i) \subsetneq SN(C_{i'})$ or $SN(C_{i'}) \subsetneq SN(C_i)$. Without loss of generality, assume $SN(C_i) \subsetneq SN(C_{i'})$. Then, by Lemmas 8 and 9, we have a path in $G'_{\mathcal{T}}$ from $C_{i'}$ to C_i and then to C_j. But this implies that $(C_{i'}, C_j) \notin E(R_{\mathcal{T}})$, which is a contradiction. Thus, $R_{\mathcal{T}}$ must be a tree.

Next, suppose some node C_i in $R_{\mathcal{T}}$ has a single child C_j. By Lemma 10, $SN(C_j) \subseteq SN(C_i)$. Observe that for any $c \in SN(C_i)$, there is a path from C_i to $v_{\{c\}}$ in $G'_{\mathcal{T}}$ according to Lemma 9, and since this path passes through C_j, we also have $c \in SN(C_j)$ by Lemma 10. This means that $SN(C_i) \subseteq SN(C_j)$, giving us $SN(C_i) = SN(C_j)$. But this contradicts Lemma 11. Thus, $R_{\mathcal{T}}$ has no nodes with outdegree 1.

Finally, for every $c \in L$, $\{v_{\{c\}}\}$ is of outdegree 0 in $G'_{\mathcal{T}}$ and is therefore a leaf in $R_{\mathcal{T}}$. The lemma follows. ∎

COROLLARY 2. $|\mathcal{C}| = O(n)$.

Proof. By the definition of $R_{\mathcal{T}}$, we have $V(R_{\mathcal{T}}) = \mathcal{C}$. Lemma 13 states that $R_{\mathcal{T}}$ is a tree with n leaves and no nodes with outdegree 1. Hence, $|\mathcal{C}| = |V(R_{\mathcal{T}})| = O(n)$. ∎

In the rest of the paper, $R_{\mathcal{T}}$ is called the *SN-tree for $\mathcal{T}$*. This is because $SN(\{a, b\})$ for any $a, b \in L$ can be obtained from $R_{\mathcal{T}}$ using the following theorem.

THEOREM 4. *Given any $a, b \in L$, let u be the lowest common ancestor of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$. Then, $SN(\{a, b\}) = \{c \in L \mid v_{\{c\}}$ is a descendant of u in $R_{\mathcal{T}}\}$.*

Proof. We first prove that for any $c \in L$, if $v_{\{c\}}$ is a descendant of the lowest common ancestor u of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$, then $c \in SN(\{a, b\})$. Let $v_{\{y,z\}}$ be any node in the strongly connected component in $G_{\mathcal{T}}$ which corresponds to u in $R_{\mathcal{T}}$. Then $a \in SN(\{y, z\})$ by Lemma 10. Since also $a \in SN(\{a, b\})$, Lemma 7 implies that either (1) $SN(\{a, b\}) \subsetneq SN(\{y, z\})$ or (2) $SN(\{y, z\}) \subseteq SN(\{a, b\})$. If (1) holds, then there is a path in $G_{\mathcal{T}}$ from $v_{\{y,z\}}$ to $v_{\{a,b\}}$ by Lemma 9, but then u cannot be the lowest common ancestor of $v_{\{a\}}$ and $v_{\{b\}}$ in $R_{\mathcal{T}}$, which is a contradiction. Thus, (2) must hold, which means that $y, z \in SN(\{a, b\})$, so there is a path in $G_{\mathcal{T}}$ from $v_{\{a,b\}}$ to $v_{\{y,z\}}$ as can be seen by applying Lemma 9 two times and then Lemma 8. Now, since $v_{\{c\}}$ is a descendant of u in $R_{\mathcal{T}}$, there is a path in $G_{\mathcal{T}}$ from $v_{\{y,z\}}$ to $v_{\{c\}}$. This shows that $v_{\{c\}}$ is reachable from $v_{\{a,b\}}$ in $G_{\mathcal{T}}$, i.e., that $c \in SN(\{a, b\})$ by Lemma 10.

Next, we prove that if $c \in SN(\{a, b\})$, then $v_{\{c\}}$ is a descendant of u in $R_{\mathcal{T}}$. Take any $c \in SN(\{a, b\})$, and suppose that $v_{\{c\}}$ is not a descendant of u in $R_{\mathcal{T}}$. Then $v_{\{b\}}$ is a descendant of the lowest common ancestor u' of $v_{\{a\}}$ and $v_{\{c\}}$ in $R_{\mathcal{T}}$, so $b \in SN(\{a, c\})$ by the preceding paragraph and, similarly, $a \in SN(\{b, c\})$. But then, $SN(\{a, b\}) = SN(\{a, c\}) = SN(\{b, c\})$ by Lemma 9 and Corollary 1, which is impossible since this would imply that u and u' coincide. Therefore, $v_{\{c\}}$ must be a descendant of u in $R_{\mathcal{T}}$. ∎

Thus, the SN-tree has the properties we want. The next theorem shows that the

SN-tree for $\mathcal{T}$ can be constructed efficiently.

THEOREM 5. *The time complexity of Algorithm ComputeSNTree is* $O(n^3)$.

Proof. By scanning all rooted triplets in $\mathcal{T}$, we can construct $G_{\mathcal{T}}$ in step 1 in $O(|\mathcal{T}|) = O(n^3)$ time. For step 2, the time complexity is $O(|V(G_{\mathcal{T}})| + |E(G_{\mathcal{T}})|) = O(n^3)$. To build the SN-tree $R_{\mathcal{T}}$ in step 3, we need two substeps. First, for each node $C_i \in V(G'_{\mathcal{T}})$, we compute the set of all nodes that are reachable from C_i in $G'_{\mathcal{T}}$. By Corollary 2, $|V(G'_{\mathcal{T}})| = O(n)$ and thus $|E(G'_{\mathcal{T}})| = O(n^2)$, so this takes $O(n^2)$ time for each C_i using depth-first search, so $O(n^3)$ time in total. Next, we check, for each $(C_i, C_j) \in E(G'_{\mathcal{T}})$, if there is a path of length at least 2 from C_i to C_j. If the answer is no, then (C_i, C_j) is an edge in $R_{\mathcal{T}}$. Each such check can be performed in $O(n)$ time by asking if C_j is reachable from any of the children (except C_j itself) of C_i. Since there are $O(n^2)$ edges, we can check all edges in $O(n^3)$ time. Thus, the algorithm's total running time is $O(n^3)$. □

3.2. Algorithm *FastGalledNetwork*. The main algorithm of this section, Algorithm *FastGalledNetwork*, is listed in Figure 6. The key observation is that a galled network consistent with $\mathcal{T}$ (if one exists) can be obtained from the SN-tree for $\mathcal{T}$ by replacing each internal node of degree 3 or higher with a subnetwork whose structure is inferred by Algorithm *SimpleNetworks*.

Recall that for any node u in a rooted, leaf-labeled tree R, $R[u]$ is the subtree of R rooted at u, and $\Lambda(R[u])$ denotes the set of leaves in $R[u]$. Below, $\mathcal{T} \mid u$ is shorthand for the set $\mathcal{T} \mid \Lambda(R[u])$.

In step 1, *FastGalledNetwork* computes the SN-tree R for $\mathcal{T}$ using Algorithm *ComputeSNTree* from section 3.1. Then, in steps 2 and 3, it tries to construct a galled network N_u consistent with all rooted triplets in $\mathcal{T} \mid u$ for each node u in R in bottom-up order. If successful, it returns N_r, where r is the root of R (note that $\mathcal{T} = \mathcal{T} \mid r$); otherwise, it returns *null*. To obtain N_u for any node u in R, *FastGalledNetwork* proceeds as follows. Let q be the degree of u and denote the children of u by $\{u_1, u_2, \ldots, u_q\}$. If $q = 0$, then let N_u be a network consisting of one leaf, labeled by u. If $q = 2$, then form N_u by joining the roots of N_{u_1} and N_{u_2} to a new root node. Otherwise, $q \geq 3$ by Lemma 13. In this case, let $\alpha_1, \alpha_2, \ldots, \alpha_q$ be q new symbols not in L, and define a function f as follows. For every $x \in \Lambda(R[u])$, let $f(x) = \alpha_i$, where $x \in \Lambda(R[u_i])$. Next, define $\mathcal{T}'$ as the set $\big\{(\{f(x), f(y)\}, f(z)) : (\{x, y\}, z) \in (\mathcal{T} \mid u) \text{ and } f(x), f(y), f(z) \text{ all differ}\big\}$, and apply Algorithm *SimpleNetworks* from section 2 to $\mathcal{T}'$. If there is a simple phylogenetic network N' consistent with $\mathcal{T}'$, then replace each α_i in N' with N_{u_i} and let N_u be the resulting network; otherwise, terminate and output *null*.

The correctness of this method follows from the next two lemmas.

LEMMA 14. *For any node u in R, if $\mathcal{T} \mid u$ is consistent with a galled network with leaf set $\Lambda(R[u])$ and if $q \geq 3$, then there exists a simple network consistent with $\mathcal{T}'$.*

Proof. Let M be a galled network with leaf set $\Lambda(R[u])$ consistent with $\mathcal{T} \mid u$. First we show that if $q \geq 3$, then the root r of M must be a split node. Suppose r is not a split node and let A and B be the disjoint sets of leaves in the two subnetworks rooted at the children of r. For every child u_i of u, we have either $\Lambda(R[u_i]) \subseteq A$ or $\Lambda(R[u_i]) \subseteq B$ (otherwise, let a, b be two leaves in $\Lambda(R[u_i])$ such that $a \in A$ and $b \in B$; for each $x \in \Lambda(R[u]) \setminus \{a, b\}$, at least one of $(\{a, x\}, b)$ and $(\{b, x\}, a)$ belongs to $\mathcal{T} \mid u$ since $\mathcal{T}$ is dense, so $x \in \Lambda(R[u_i])$, i.e., $\Lambda(R[u_i]) = \Lambda(R[u])$, which is not possible). Since $q \geq 3$, there exist i, j, k where i, j, k differ such that both $\Lambda(R[u_i])$ and $\Lambda(R[u_j])$ are subsets of one of A and B, and $\Lambda(R[u_k])$ is a subset of the other. Assume without loss of generality that $\Lambda(R[u_i]), \Lambda(R[u_j]) \subseteq A$ and $\Lambda(R[u_k]) \subseteq B$.

Algorithm *FastGalledNetwork*
Input: A dense set $\mathcal{T}$ of rooted triplets with a leaf set L.
Output: A galled network consistent with $\mathcal{T}$, if one exists; otherwise, *null*.

1 Let $R = ComputeSNTree(\mathcal{T})$.
2 Define N_u for every leaf u in R to be a single node labeled by u.
3 **for** each nonleaf node u in R, in bottom-up order **do**
/* Construct a galled network N_u for the set of leaves in $\Lambda(R[u])$. */
3.1 Denote the set of children of u in R by $\{u_1, u_2, \ldots, u_q\}$.
3.2 If $q = 2$, let N_u be a network with a root node joined to N_{u_1} and N_{u_2}.
3.3 Otherwise ($q \geq 3$), build $\mathcal{T}'$ from $\mathcal{T} \mid u$, compute $\mathcal{N} = SimpleNetworks(\mathcal{T}')$, and check if $\mathcal{N}$ is empty; if yes then **return** *null*, else select any $N' \in \mathcal{N}$ and form a network N_u by replacing each α_i in N' with N_{u_i}.
endfor
4 **return** N_r, where r is the root of R.
End *FastGalledNetwork*

FIG. 6. *Constructing a galled phylogenetic network consistent with a dense set $\mathcal{T}$ of rooted triplets.*

For any $x, y \in A$ and $b_k \in \Lambda(R[u_k])$, $\mathcal{T}$ cannot contain $(\{x, b_k\}, y)$ or $(\{y, b_k\}, x)$, so $SN(\{a_i, a_j\})$ is a proper subset of $SN(\{a_i, b_k\})$ for every $a_i \in \Lambda(R[u_i])$, $a_j \in \Lambda(R[u_j])$. However, u is the lowest common ancestor in R of $\{a_i, a_j\}$ as well as of $\{a_i, b_k\}$, so $SN(\{a_i, a_j\}) = SN(\{a_i, b_k\})$ by Theorem 4, which is a contradiction. Hence, r is a split node.

Next, observe that each side network of M can contain leaves from only one $R[u_i]$. To see this, let $M[v]$ be a side network of M. For any i, j with $i \neq j$, if $M[v]$ contains a leaf $a \in \Lambda(R[u_i])$ and a leaf $b \in \Lambda(R[u_j])$, then since $\mathcal{T}$ is dense, $\Lambda(R[u_i]) \subsetneq SN(\{a, b\})$ and $\Lambda(R[u_j]) \subsetneq SN(\{a, b\})$ by Lemma 7 while $SN(\{a, b\}) \neq \Lambda(R[u])$ (otherwise, $M[v]$ cannot be a side network of M), contradicting the maximality of $\Lambda(R[u_i])$ and $\Lambda(R[u_j])$ in $\Lambda(R[u])$.

By the preceding two paragraphs, the root of M is a split node of some hybrid node h, and each side network attached to a merge path of h contains leaves from only one $\Lambda(R[u_i])$. We now show that there exists a galled network M^* consistent with $\mathcal{T} \mid u$ such that for every i, all leaves in $\Lambda(R[u_i])$ belong to only one side network of M^* attached to a merge path of h. Suppose M has two side networks $M[v]$ and $M[w]$ attached to merge paths of h such that both $M[v]$ and $M[w]$ contain leaves from the same $\Lambda(R[u_i])$. $M[v]$ and $M[w]$ must be attached to the same merge path p of h (otherwise, $\Lambda(R[u_i]) = \Lambda(R[u])$, which is impossible), and, furthermore, all side networks attached to p between $M[v]$ and $M[w]$ contain leaves from $\Lambda(R[u_i])$ only. Thus, all side networks containing leaves from the same $\Lambda(R[u_i])$ are consecutively ordered along one merge path of h and can therefore be concatenated into one side network in such a way that all rooted triplets involving $\Lambda(R[u_i])$ are still consistent with it (note that $\mathcal{T}$ does not contain any rooted triplet of the form $(\{a, x\}, b)$ where $a, b \in \Lambda(R[u_i])$ and x is located in a side tree of M below the side trees leaf-labeled by $\Lambda(R[u_i])$). Let M^* be the resulting galled network consistent with $\mathcal{T} \mid u$ such that each side network M_i^* attached to a merge path of h is bijectively leaf-labeled by one $\Lambda(R[u_i])$.

Finally, construct a simple phylogenetic network M' from M^* by replacing each M_i^* by a leaf labeled by α_i. M' is consistent with $\mathcal{T}'$, which can be seen as follows. Let t' be any rooted triplet in $\mathcal{T}'$ and write $t' = (\{\alpha_i, \alpha_j\}, \alpha_k)$. Then there exists some rooted triplet $t = (\{x, y\}, z)$ in $\mathcal{T}$ such that $x \in \Lambda(R[u_i])$, $y \in \Lambda(R[u_j])$, and

$z \in \Lambda(R[u_k])$, where i, j, k all differ. t is consistent with M^*, so t' is consistent with M' by the construction of M'. Hence, $\mathcal{T}'$ and M' are consistent. ∎

LEMMA 15. *Let u be any node in R and suppose each $\mathcal{T} \mid u_i$ is consistent with a galled network N_{u_i}. If $q = 2$, then the galled network obtained by joining the roots of N_{u_1} and N_{u_2} to a new root node is consistent with $\mathcal{T} \mid u$. If $q \geq 3$ and $\mathcal{T}'$ is consistent with a simple network N' with leaf set $\{\alpha_1, \alpha_2, \ldots, \alpha_q\}$, then the galled network N_u obtained from N' by replacing each α_i by N_{u_i} is consistent with $\mathcal{T} \mid u$.*

Proof (analogous to the proof of Lemma 8 in [15]). Consider any rooted triplet t in $\mathcal{T} \mid u$ and write $t = (\{x, y\}, z)$. If $x \in \Lambda(R[u_i])$, $y \in \Lambda(R[u_j])$, and $z \in \Lambda(R[u_k])$, where i, j, k all differ, then t is consistent with N_u (otherwise, $t' = (\{f(x), f(y)\}, f(z)) = (\{\alpha_i, \alpha_j\}, \alpha_k)$ cannot be consistent with N' which is a contradiction since $t' \in \mathcal{T}'$). If $x, y \in \Lambda(R[u_i])$ and $z \in \Lambda(R[u_j])$ with $i \neq j$, then t is consistent with N_u by the construction of N_u. The case $x, z \in \Lambda(R[u_i])$ and $y \in \Lambda(R[u_j])$ (or symmetrically, $y, z \in \Lambda(R[u_i])$ and $x \in \Lambda(R[u_j])$) with $i \neq j$ is not possible because then y would not belong to $SN(\{x, z\})$ by Theorem 4, contradicting that $y \in SN(\{x, z\})$ according to the definition of SN-sets. Finally, if x, y, z belong to the same $\Lambda(R[u_i])$, then t is consistent with N_{u_i} and therefore with N_u. In all possible cases, t is consistent with N_u. ∎

We now analyze the running time of *FastGalledNetwork*.

THEOREM 6. *The time complexity of Algorithm FastGalledNetwork is $O(n^3)$.*

Proof. Step 1 takes $O(n^3)$ by Theorem 5. For every node u in R with $\deg(u) < 3$, N_u can be constructed in $O(1)$ time. The total time for constructing all networks N_u with $\deg(u) \geq 3$ is given by the total time needed to build all the $\mathcal{T}'$-sets plus the total time taken by all calls to *SimpleNetworks*; both of these are shown below to be $O(n^3)$. Thus, the theorem follows.

First note that to construct $\mathcal{T}'$ for a node u, we need to consider only rooted triplets in $\mathcal{T}$ whose three leaves belong to subtrees rooted at three different children of u. For this purpose, we may create a list $T(u)$ for each node u in R containing all rooted triplets in $\mathcal{T}$ of the form $(\{x, y\}, z)$ such that u is the lowest common ancestor in R of x, y, and z. All the $T(u)$-lists can be constructed using an additional $O(n^3)$ time after computing the SN-tree R in step 1 by doing a bottom-up traversal of R. Then, when constructing $\mathcal{T}'$ in step 3.3, check each rooted triplet in $T(u)$ to see if its leaves belong to three different subtrees, and if so, update $\mathcal{T}'$ accordingly. This way, each rooted triplet in $\mathcal{T}$ is considered for one $\mathcal{T}'$-set only, so the total time required to build all $\mathcal{T}'$-sets is bounded by $O(n^3)$.

Next, note that each constructed $\mathcal{T}'$ has $\deg(u)$ leaves. Running *SimpleNetworks* on $\mathcal{T}'$ therefore takes $O((\deg(u))^3)$ time by Theorem 1. Summing over all nodes, the calls to *SimpleNetworks* take a total of $\sum_{u \in R} O((\deg(u))^3) = O((\sum_{u \in R} \deg(u))^3) = O(n^3)$ time. ∎

Finally, we remark that *FastGalledNetwork* can be modified to return *all* galled networks consistent with $\mathcal{T}$ by utilizing all simple networks computed in step 3.3. However, this may take exponential time.

4. NP-hardness of the nondense case. We now prove that the problem of inferring a galled phylogenetic network which is consistent with a given set of $\mathcal{T}$ rooted triplets, if one exists, is NP-hard when $\mathcal{T}$ is not required to be dense. Our proof consists of a polynomial-time reduction from the NP-complete problem Set Splitting (see, e.g., [5]) to the decision version of our problem. We use the same reduction to prove that the closely related problem of inferring a *simple* phylogenetic network which is consistent with a given (nondense) set of rooted triplets is also NP-hard.

Set Splitting. Given a set $S = \{s_1, s_2, \ldots, s_n\}$ and a collection $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of subsets of S, where $|C_j| = 3$ for every $C_j \in \mathcal{C}$, does $(S, \mathcal{C})$ have a set splitting; i.e., can S be partitioned into two disjoint subsets S_1, S_2 such that for every $C_j \in \mathcal{C}$ it holds that C_j is not a subset of S_1 and C_j is not a subset of S_2?

First, we describe the reduction from Set Splitting. Given an instance $(S, \mathcal{C})$, where we assume without loss of generality that $\bigcup_{C_j \in \mathcal{C}} C_j = S$, construct a nondense set $\mathcal{T}$ of rooted triplets having a leaf set L with $L = \{h, x, y\} \cup \{s_i^j \mid s_i \in S,\ 1 \le j \le m\}$, where h, x, y, and all elements of the form s_i^j are new elements not belonging to S. Initially, let $\mathcal{T}$ consist of the two rooted triplets $(\{x, h\}, y)$ and $(\{y, h\}, x)$. Next, for each $C_j \in \mathcal{C}$, write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$ and include three rooted triplets $(\{s_a^j, h\}, s_b^j)$, $(\{s_b^j, h\}, s_c^j)$, and $(\{s_c^j, h\}, s_a^j)$ in $\mathcal{T}$. Finally, for each $s_i \in S$, add m rooted triplets $(\{s_i^1, s_i^2\}, h)$, $(\{s_i^2, s_i^3\}, h), \ldots, (\{s_i^{m-1}, s_i^m\}, h)$, $(\{s_i^m, s_i^1\}, h)$ and $2m$ rooted triplets $(\{s_i^1, h\}, x)$, $(\{y, h\}, s_i^1)$, $(\{s_i^2, h\}, x)$, $(\{y, h\}, s_i^2), \ldots, (\{s_i^m, h\}, x)$, $(\{y, h\}, s_i^m)$ to $\mathcal{T}$. (The main idea in the reduction is to encode $\mathcal{C}$ by rooted triplets of the form $(\{s_a^j, h\}, s_b^j)$ and use other rooted triplets to force any galled network N consistent with $\mathcal{T}$ to have a special structure; see Lemma 17. Then, for each $C_j = \{s_a, s_b, s_c\} \in \mathcal{C}$, at most two of s_a^j, s_b^j, s_c^j can descend from the same clipped merge path from the root in N, inducing a set splitting of S.)

LEMMA 16. *If $(S, \mathcal{C})$ has a set splitting, then there exists a simple phylogenetic network which is consistent with $\mathcal{T}$.*

Proof. Let (S_1, S_2) be a set splitting of $(S, \mathcal{C})$. Define $S_1^* = \{s_i^j \mid s_i \in S_1,\ 1 \le j \le m\}$ and $S_2^* = \{s_i^j \mid s_i \in S_2,\ 1 \le j \le m\}$. Note that $S_1^* \cup S_2^* \cup \{h, x, y\} = L$. Let O_1 be any ordering of $S_1^* \cup \{x\}$ in which x is the first element, and for every pair of elements of the form s_a^j and s_b^j in S_1^*, if there exists a C_j in $\mathcal{C}$ with $C_j = \{s_a, s_b, s_c\}$ and either $b < a < c$, $a < c < b$, or $c < b < a$, then s_a^j precedes s_b^j. (Except for this requirement, the elements may be ordered arbitrarily in O_1.) Define an ordering O_2 of $S_2^* \cup \{y\}$ analogously, letting y be the last element in O_2, respectively. Next, build a simple phylogenetic network N having a root node r and a hybrid node whose child is a leaf labeled by h, where (1) $|S_1^*| + 1$ leaves distinctly labeled by $S_1^* \cup \{x\}$ are attached to the left clipped merge path in order according to O_1, and $|S_2^*| + 1$ leaves distinctly labeled by $S_2^* \cup \{y\}$ are attached to the right clipped merge path in order according to O_2. See Figure 7 for an example. It is easy to verify that N and $\mathcal{T}$ are consistent. ∎

To prove the other direction (i.e., that a galled network consistent with $\mathcal{T}$ yields a set splitting of $(S, \mathcal{C})$), we need the following lemma.

LEMMA 17. *Suppose N is a galled network with leaf set L which is consistent with $\mathcal{T}$. Then* (1) *the root r of N is a split node,* (2) *one side network attached to a merge path of r contains h but no other leaves, and* (3) *h is a descendant of the hybrid node for r.*

Proof. (1) Suppose r is not a split node. Then L can be partitioned into two disjoint, nonempty subsets U and V such that every path between a leaf in U and a leaf in V passes through r. It follows that for any rooted triplet $(\{a, b\}, c)$ which is consistent with N, if $a \in U$, then $b \in U$, and if $a \in V$, then $b \in V$. Now consider any element of the form s_i^j in L. If $s_i^j \in U$, then $h, x, y \in U$ because N is consistent with $(\{s_i^j, h\}, x)$, $(\{x, h\}, y)$, and $(\{y, h\}, s_i^j)$. But then also $s_z^k \in U$ for every s_z^k in L by $(\{s_z^k, h\}, x) \in \mathcal{T}$, contradicting that V is nonempty. The case $s_i^j \in V$ is analogous. Hence, r must be a split node.

(2) Let N' be the side network attached to a merge path of r such that $h \in \Lambda(N')$.

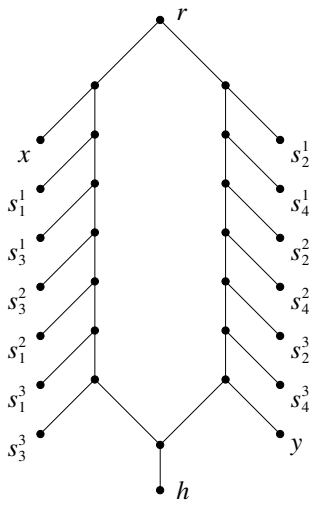

FIG. 7. *Let* $C_1 = \{s_1, s_2, s_3\}$, $C_2 = \{s_1, s_3, s_4\}$, *and* $C_3 = \{s_2, s_3, s_4\}$, *and suppose* $S_1 = \{s_1, s_3\}$ *and* $S_2 = \{s_2, s_4\}$. *The construction described in Lemma* 16 *yields a simple phylogenetic network* N *as shown above.*

If $x \in \Lambda(N')$, then $\Lambda(N') = L$ because $y \in \Lambda(N')$ by $(\{y, h\}, x) \in \mathcal{T}$ and for every element of the form s_i^j in L, it holds that $s_i^j \in \Lambda(N')$ by $(\{s_i^j, h\}, x) \in \mathcal{T}$. If $y \in \Lambda(N')$, then $\Lambda(N') = L$ because of $x \in \Lambda(N')$ by $(\{x, h\}, y) \in \mathcal{T}$ and the above. If $\Lambda(N')$ contains an element of the form s_i^j, then $\Lambda(N') = L$ because of $y \in \Lambda(N')$ by $(\{y, h\}, s_i^j) \in \mathcal{T}$ and the above. Thus, if $\Lambda(N')$ contains any element in addition to h, then $\Lambda(N') = L$, which is not possible. Therefore, $\Lambda(N') = \{h\}$.

(3) By (1), r is a split node of N. Let $hn(r)$ be the hybrid node for r and let N' be the subnetwork of N rooted at $hn(r)$. Suppose, on the contrary, that h is not contained in N'. Take any $C_j \in \mathcal{C}$ and write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$. Since $(\{s_a^j, h\}, s_b^j)$, $(\{s_b^j, h\}, s_c^j)$, and $(\{s_c^j, h\}, s_a^j)$ are in $\mathcal{T}$ and $h \notin \Lambda(N')$, exactly one of s_a^j, s_b^j, and s_c^j belongs to $\Lambda(N')$. Assume without loss of generality that $s_a^j \in \Lambda(N')$. Next, neither x nor y can belong to the same side network as h by (2), and since $(\{x, h\}, y)$ and $(\{y, h\}, x)$ are consistent with N, we have either $x \in \Lambda(N')$ and $y \notin \Lambda(N')$ or $x \notin \Lambda(N')$ and $y \in \Lambda(N')$. If $x \in \Lambda(N')$, then $(\{s_a^j, h\}, x)$ is not consistent with N, and if $y \in \Lambda(N')$, then $(\{y, h\}, s_a^j)$ is not consistent with N, which is a contradiction in both cases. Therefore, $h \in \Lambda(N')$. □

LEMMA 18. *If there exists a galled network which is consistent with* $\mathcal{T}$, *then* $(S, \mathcal{C})$ *has a set splitting.*

Proof. Let N be a galled network with leaf set L that is consistent with $\mathcal{T}$. By Lemma 17, the root r of N is a split node. Also by Lemma 17, the subnetwork of N rooted at the child of $hn(r)$, where $hn(r)$ denotes the hybrid node for r, consists of a single leaf which is labeled by h. Let P_1 and P_2 be the two clipped merge paths of $hn(r)$, and define L_1 and L_2 as the set of all leaves except x, y, and h which are descendants of nodes on P_1 and P_2, respectively. We now show that (L_1, L_2) induces a set splitting (S_1, S_2) of $(S, \mathcal{C})$. For every $s_i \in S$, if s_i^1 belongs to L_1, then all elements in $\{s_i^k \mid 1 \le k \le m\}$ belong to L_1 because $(\{s_i^1, s_i^2\}, h)$, $(\{s_i^2, s_i^3\}, h)$, …, $(\{s_i^{m-1}, s_i^m\}, h)$, $(\{s_i^m, s_i^1\}, h) \in \mathcal{T}$. Similarly, if $s_i^1 \in L_2$, then $\{s_i^k \mid 1 \le k \le m\} \subseteq L_2$. Define $S_1 = \{s_i \mid s_i^1 \in L_1\}$ and $S_2 = \{s_i \mid s_i^1 \in L_2\}$. Clearly, $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$. Moreover, for every $C_j \in \mathcal{C}$, it holds that $C_j \not\subseteq S_1$ and $C_j \not\subseteq S_2$ (to

see this, write $C_j = \{s_a, s_b, s_c\}$ with $a < b < c$ and note that if all of s_a, s_b, and s_c belonged to just one of S_1 and S_2, then s_a^j, s_b^j, and s_c^j would be descendants of nodes on the same clipped merge path of $hn(r)$, and then all three rooted triplets $(\{s_a^j, h\}, s_b^j)$, $(\{s_b^j, h\}, s_c^j)$, and $(\{s_c^j, h\}, s_a^j)$ could never be consistent with N, which is a contradiction). Thus, (S_1, S_2) is a set splitting of $(S, \mathcal{C})$. ∎

THEOREM 7. *Given any nondense set $\mathcal{T}$ of rooted triplets, it is NP-hard to determine if there exists a galled network which is consistent with $\mathcal{T}$. It is also NP-hard to determine if there exists a simple network which is consistent with $\mathcal{T}$.*

Proof. If $(S, \mathcal{C})$ has a set splitting, then there exists a simple network which is consistent with $\mathcal{T}$ according to Lemma 16. Next, Lemma 18 shows that if there exists a galled network consistent with $\mathcal{T}$, then $(S, \mathcal{C})$ has a set splitting. Since a simple phylogenetic network is always a galled network, and the reduction can be carried out in polynomial time, the theorem follows. ∎

5. Approximating the maximum number of consistent rooted triplets. This section studies the problem of constructing a galled network consistent with the maximum number of rooted triplets in $\mathcal{T}$ for any (not necessarily dense) given $\mathcal{T}$. Section 5.2 presents a polynomial-time approximation algorithm for this problem which always outputs a galled network consistent with at least a factor of $\frac{5}{12}$ (> 0.4166) of the rooted triplets in $\mathcal{T}$. On the negative side, section 5.1 shows that there exist inputs for which any galled network can be consistent with at most a factor of 0.4883 of the rooted triplets in $\mathcal{T}$.

5.1. Inapproximability result. Given any positive integer n, fix $\mathcal{T}$ to contain all possible rooted triplets for a leaf set L of size n, that is, $\mathcal{T} = \big\{(\{a, b\}, c), (\{a, c\}, b), (\{b, c\}, a) \mid a, b, c \in L\big\}$. For any phylogenetic network N, let $\#N$ denote the number of rooted triplets from $\mathcal{T}$ that are consistent with N.

LEMMA 19. *Let N be a galled network with $\Lambda(N) = L$. If N contains a nonsplit node u with two children u_1, u_2 such that u is the root node or a child of a hybrid node, then making u into a split node by removing the edges (u, u_1) and (u, u_2), adding two new nodes v and w, and inserting the edges (u, v), (u, w), (v, u_1), (w, u_2), and (v, w) yields a galled network N' with $\#N' = \#N$.*

LEMMA 20. *Let N be a galled network with $\Lambda(N) = L$. Suppose N contains a merge path P of a hybrid node h, c is the child of h, $N[u]$ is a side network attached to P, $u \neq c$, and u has two children u_1, u_2. Then one of the following holds:*

- *If $|\Lambda(N[u])| > |\Lambda(N[c])|$, then N can be transformed into a galled network N' with $\#N' > \#N$ by letting $N[u]$ and $N[c]$ trade places.*
- *Else, if $|\Lambda(N[u])| \leq |\Lambda(N[c])|$, then N can be transformed into a galled network N' with $\#N' > \#N$ as follows: First, in case u is a split node in N, delete a hybrid edge descending from u (and contract all edges from vertices with outdegree 1) so that $N[u_1]$ and $N[u_2]$ become disjoint. Second, remove $N[u]$ and instead attach the resulting disjoint $N[u_1]$ and $N[u_2]$ to P.*

Proof. Let s be the split node corresponding to h, and define $L_u = \Lambda(N[u])$, $L_c = \Lambda(N[c])$, $L_m = \Lambda(N[s]) \setminus (L_u \cup L_c)$, and $L_{rest} = L \setminus (L_u \cup L_c \cup L_m)$. First consider the case $|\Lambda(N[u])| > |\Lambda(N[c])|$. For any subset $\{x, y, z\}$ of L, we have the following possibilities:

- When at least one of $\{x, y, z\}$ belongs to L_{rest}, it is easy to see that N and N' are consistent with exactly the same rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains at least two leaves from L_u or at least two leaves from L_c, or when $\{x, y, z\}$ contains three leaves from L_m, then N and N' are

again consistent with exactly the same rooted triplets labeled by $\{x, y, z\}$.

- When $\{x, y, z\}$ contains one leaf from L_u, one leaf from L_c, and one leaf from L_m, then each of N and N' is consistent with exactly two rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_u and two leaves from L_m, then N is consistent with one rooted triplet labeled by $\{x, y, z\}$, whereas N' is consistent with two.
- When $\{x, y, z\}$ contains one leaf from L_c and two leaves from L_m, then N is consistent with two rooted triplets labeled by $\{x, y, z\}$, whereas N' is consistent with one.

Since $|L_u| > |L_c|$, the difference in number of consistent rooted triplets is given by $\#N' - \#N = |L_u| \cdot \binom{|L_m|}{2} - |L_c| \cdot \binom{|L_m|}{2} > 0$.

Next consider the case $|\Lambda(N[u])| \leq |\Lambda(N[c])|$. If u is a split node in N, then delete a hybrid edge e descending from u so that the resulting $\Lambda(N[u_1])$ and $\Lambda(N[u_2])$ are disjoint; assume without loss of generality that e is a descendant of u_2. In addition to the above, define $L_{u_1} = \Lambda(N[u_1])$ and $L_{u_2} = \Lambda(N[u]) \setminus \Lambda(N[u_1])$. For any subset $\{x, y, z\}$ of L, we have the following possibilities:

- When at least one of $\{x, y, z\}$ belongs to L_{rest}, then N and N' are consistent with exactly the same rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ does not contain at least one leaf from L_{u_1} and at least one leaf from L_{u_2}, then N and N' are consistent with the same number of rooted triplets labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_{u_1}, one leaf from L_{u_2}, and one leaf from L_m, then each of N and N' is consistent with one rooted triplet labeled by $\{x, y, z\}$.
- When $\{x, y, z\}$ contains one leaf from L_{u_1}, one leaf from L_{u_2}, and one leaf from L_c, then N is consistent with one rooted triplet labeled by $\{x, y, z\}$, whereas N' is consistent with two.
- When $\{x, y, z\}$ contains two leaves from L_{u_1} and one leaf from L_{u_2}, or one leaf from L_{u_1} and two leaves from L_{u_2}, then N is consistent with one or two rooted triplets labeled by $\{x, y, z\}$, whereas N' is consistent with one.

Since $|L_c| \geq |L_u|$, $|L_u| \geq |L_{u_1}|$, and $|L_u| \geq |L_{u_2}|$, the difference in number of consistent rooted triplets satisfies $\#N' - \#N \geq |L_{u_1}| \cdot |L_{u_2}| \cdot |L_c| - \binom{|L_{u_1}|}{2} \cdot |L_{u_2}| - |L_{u_1}| \cdot \binom{|L_{u_2}|}{2} \geq \frac{1}{2} \cdot |L_{u_1}| \cdot |L_{u_2}| \cdot (2 \cdot |L_u| - |L_{u_1}| - |L_{u_2}| + 2) > 0$. □

By repeatedly applying Lemmas 19 and 20, the next lemma concludes that for any fixed n, at least one of the galled networks N for a set of n leaves that maximizes $\#N$ must be a *caterpillar network*. A galled network N is called a caterpillar network if (1) the root of N and every nonleaf child of a hybrid node are split nodes and (2) for every merge path P in N, all side networks attached to P except for the one at the hybrid node are leaves.

LEMMA 21. *For any galled network N, there is a caterpillar network N' with $\Lambda(N') = \Lambda(N)$ and $\#N' \geq \#N$.*

Now, we are ready to show the bound on the approximation ratio. Let $S(n)$ be the maximum value of $\#N$ taken over all galled networks N with n leaves.

LEMMA 22. $S(n) = \max_{1 \leq a \leq n} \left\{ \binom{a}{3} + 2 \cdot \binom{a}{2} \cdot (n - a) + a \cdot \binom{n-a}{2} + S(n - a) \right\}$.

Proof. By Lemma 21, there is a caterpillar network N that maximizes $\#N$ among all galled networks with n leaves. The recurrence for $S(n)$ counts the maximum number of rooted triplets in $\mathcal{T}$ consistent with a caterpillar network with n leaves because if such a network contains a set A of a leaves attached to the two merge

paths starting at the root, then it must be consistent with

- $\binom{a}{3}$ rooted triplets labeled by three elements in A;
- $2 \cdot \binom{a}{2} \cdot (n-a)$ rooted triplets labeled by two elements in A and one element in $L \setminus A$;
- $a \cdot \binom{n-a}{2}$ rooted triplets labeled by one element in A and two elements in $L \setminus A$;
- $S(n-a)$ rooted triplets labeled by three elements in $L \setminus A$. ∎

THEOREM 8. *There is no approximation algorithm with approximation ratio larger than* 0.4883.

Proof. Define $T(n) = |\mathcal{T}|$ for any given positive integer n, i.e., $T(n) = 3 \cdot \binom{n}{3}$. Note that the approximation ratio can be at most $\min_{n \in \mathbb{Z}^+} \frac{S(n)}{T(n)}$. By inserting $n = 1000$ into the recurrence in Lemma 22, we obtain $S(1000) = 243383298$. Hence, the approximation ratio must be less than or equal to $\frac{S(1000)}{T(1000)} < 0.4883$. ∎

5.2. A polynomial-time $\frac{5}{12}$-approximation algorithm. Given any set $\mathcal{T}$ of rooted triplets, our approximation algorithm called *Approximate* (shown in Figure 8) infers a galled network which is consistent with at least $\frac{5}{12}$ of the rooted triplets in $\mathcal{T}$. We first describe the algorithm and then present the analysis.

Initially, *Approximate* partitions the set of leaves L into three subsets A, B, C so that none of them equals L using an algorithm named *LeafPartition* (also listed in Figure 8 and described in detail below). Then, for each $X \in \{A, B, C\}$, it recursively infers a galled network K_X by calling *Approximate*$(\mathcal{T}|X)$. Next, for each $X \in \{A, B, C\}$, it generates a galled network $Network_X$ such that the root node is a split node whose hybrid node is the parent of K_X, and the other two networks in $\{K_A, K_B, K_C\} \setminus \{K_X\}$ are side networks. Finally, it returns the best network among $Network_A$, $Network_B$, and $Network_C$.

We now explain the algorithm *LeafPartition*. It divides L into the three subsets A, B, C in such a way that a special condition $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$ holds, where for $i \in \{0, 1, 2, 3\}$, we define $N_i = |Z_i(A, B, C)|$ and where $Z_i(A, B, C)$ is the set defined as follows:

- $Z_0(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x$ and z are in one of the subsets A, B, C and y is in another$\}$;
- $Z_1(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x, y$, and z are in one of the subsets $A, B, C\}$;
- $Z_2(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x, y$, and z are in three different subsets among $A, B, C\}$;
- $Z_3(A, B, C) = \{(\{x, y\}, z) \in \mathcal{T} \mid x$ and y are in one of the subsets A, B, C and z is in another$\}$.

Note that $Z_0(A, B, C) \cup Z_1(A, B, C) \cup Z_2(A, B, C) \cup Z_3(A, B, C) = \mathcal{T}$. As shown below, any A, B, C which imply $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$ guarantee a good approximation ratio for *Approximate*. Algorithm *LeafPartition* is a greedy algorithm which first divides L into the three subsets arbitrarily and then moves leaves (one at a time) from one subset to another until $score(A, B, C)$ cannot be further improved, where we define $score(A, B, C) = 4N_1 + 7N_2 + 12N_3$. If one of the subsets, say A, equals L after finishing moving the leaves, then it selects a leaf u that maximizes $\frac{p(u)}{c(u)}$, where $p(u) = |\{(\{x, y\}, u) \in T\}|$ and $c(u) = |\{(\{u, x\}, y) \in \mathcal{T}\}|$, and moves u from A to either B or C. (This step is to ensure that none of the three subsets equals L.) The next lemma shows that this extra move does not reduce the value of $score(A, B, C)$. Since *score* keeps increasing by at least 1 as long as the while-loop iterates and $score(A, B, C) \leq 12 \cdot |\mathcal{T}|$, step 2.1 is performed at most $12 \cdot |\mathcal{T}|$ times in total; i.e., the algorithm is guaranteed to terminate.

Algorithm *LeafPartition*
Input: A set $\mathcal{T}$ of rooted triplets with a leaf set L.
Output: A partition of L into three subsets A, B, C such that none of them equals L and such that $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$.

1 Arbitrarily partition L into three subsets A, B, C.
2 **while** moving a leaf m from one subset to another increases $score(A, B, C) = 4N_1 + 7N_2 + 12N_3$ **do**
2.1 Move m accordingly.
endwhile
3 **if** one of the subsets A, B, C equals L **then**
3.1 Choose a leaf u that maximizes $\frac{p(u)}{c(u)}$ and move u to another subset. Go to Step 2.
endif
4 **return** A, B, C.
End *LeafPartition*

Algorithm *Approximate*
Input: A set $\mathcal{T}$ of rooted triplets with a leaf set L.
Output: A galled network that is consistent with at least $\frac{5}{12} \cdot |\mathcal{T}|$ of the rooted triplets in $\mathcal{T}$.

1 Partition L into A, B, C using *LeafPartition*.
2 For $X \in \{A, B, C\}$, let $K_X = Approximate(\mathcal{T}|X)$.
3 For $X \in \{A, B, C\}$, generate a galled network $Network_X$ in which the root node is a split node whose hybrid node h is the parent of K_X, and the other two networks in $\{K_A, K_B, K_C\} \setminus \{K_X\}$ are side networks attached to the merge paths of h.
4 **return** the $Network_X$ among $X \in \{A, B, C\}$ that is consistent with the most rooted triplets in $\mathcal{T}$.
End *Approximate*

FIG. 8. *An approximation algorithm for computing a galled network consistent with as many rooted triplets in $\mathcal{T}$ as possible.*

LEMMA 23. *Algorithm LeafPartition partitions L into three subsets A, B, C so that $score(A, B, C)$ cannot be further improved by moving a single element from one subset to another.*

Proof. If none of A, B, C equals L after step 2 in Algorithm *LeafPartition* is done, the lemma follows. Hence, assume that one of the subsets, say A, equals L after step 2. We only need to show that step 3.1 does not decrease $score(A, B, C)$. When u is moved from A, all rooted triplets in $\mathcal{T}$ of the form $(\{x, y\}, u)$ are moved from $Z_1(A, B, C)$ to $Z_3(A, B, C)$ and all rooted triplets in $\mathcal{T}$ of the form $(\{u, x\}, y)$ are moved from $Z_1(A, B, C)$ to $Z_0(A, B, C)$. The difference in *score* is equal to $score(A \setminus \{u\}, \{u\}, \emptyset) - score(A, \emptyset, \emptyset) = p(u) \cdot (12 - 4) - c(u) \cdot 4 \geq 0$, where the last inequality follows since $p(u) \geq \frac{1}{2} \cdot c(u)$ by the choice of u. Thus, step 3.1 will not decrease $score(A, B, C)$. □

The next two lemmas are needed to analyze the approximation ratio of *Approximate*.

LEMMA 24. *When Algorithm LeafPartition terminates, we have $5N_1 + 8N_2 + 12N_3 \geq 5 \cdot |\mathcal{T}|$.*

Proof. Write $score(A, B, C) = x \cdot N_1 + y \cdot N_2 + z \cdot N_3$. For any $U, V, W \in \{A, B, C\}$, denote by $(\{U, V\}, W)$ the set of all rooted triplets in $\mathcal{T}$ of the form $(\{u, v\}, w)$ where $u \in U$, $v \in V$, and $w \in W$. Similarly, for any $m \in L$ and $U, V \in \{A, B, C\}$, let

$(\{m,U\},V)$ and $(\{U,V\},m)$ denote the set of all rooted triplets in $\mathcal{T}$ of the form $(\{m,u\},v)$ and $(\{u,v\},m)$, respectively, where $u \in U$ and $v \in V$. For each of the six possible ways of moving a leaf m from one of the subsets A, B, C to another, we can derive a formula to express how *score* is affected, as described next.

First, suppose m is moved from A to B. Then, for every element t in $(\{m,A\},A)$, since t will be moved from $Z_1(A,B,C)$ to $Z_0(A,B,C)$, the corresponding change in *score* is $-x$. In the same way, we can calculate the change in *score* for each element in $(\{m,U\},V)$ and $(\{U,V\},m)$ for every $U,V \in \{A,B,C\}$ when m is moved from A to B. After *LeafPartition* is done, moving m will not increase the value of *score*, so $score(A\setminus\{m\},\, B\cup\{m\},\, C) - score(A,B,C) \le 0$. Thus, we have $-x|(\{m,A\},A)| - z|(\{m,A\},B)| + (y-z)|(\{m,A\},C)| + z|(\{m,B\},A)| + x|(\{m,B\},B)| + (z-y)|(\{m,B\},C)| + y|(\{m,C\},A)| - y|(\{m,C\},B)| + 0\cdot|(\{m,C\},C)| + (z-x)|(\{A,A\},m)| + 0\cdot|(\{A,B\},m)| + y|(\{A,C\},m)| + (x-z)|(\{B,B\},m)| - y|(\{B,C\},m)| + 0\cdot|(\{C,C\},m)| \le 0$.

Next, by summing over all $m \in A$, we obtain the following inequality I_{AB}:

$$I_{AB}:\ -2x|(\{A,A\},A)| - 2z|(\{A,A\},B)| + 2(y-z)|(\{A,A\},C)| + z|(\{A,B\},A)| + x|(\{A,B\},B)| + (z-y)|(\{A,B\},C)| + y|(\{A,C\},A)| - y|(\{A,C\},B)| + (z-x)|(\{A,A\},A)| + y|(\{A,C\},A)| + (x-z)|(\{B,B\},A)| - y|(\{B,C\},A)| \le 0.$$

In the summation, each element of the form $(\{a_1,a_2\},x)$ where $a_1,a_2 \in A$ is counted twice; therefore, the coefficient of each $|(\{A,A\},x)|$ is multiplied by 2.

We derive five inequalities I_{AC}, I_{BA}, I_{BC}, I_{CA}, and I_{CB} analogously. Finally, we add I_{AB}, I_{AC}, I_{BA}, I_{BC}, I_{CA}, and I_{CB} together, and use $N_0 = |(\{A,B\},A)| + |(\{A,B\},B)| + |(\{A,C\},A)| + |(\{A,C\},C)| + |(\{B,C\},B)| + |(\{B,C\},C)|$, $N_1 = |(\{A,A\},A)| + |(\{B,B\},B)| + |(\{C,C\},C)|$, $N_2 = |(\{A,B\},C)| + |(\{A,C\},B)| + |(\{B,C\},A)|$, and $N_3 = |(\{A,A\},B)| + |(\{A,A\},C)| + |(\{B,B\},A)| + |(\{B,B\},C)| + |(\{C,C\},A)| + |(\{C,C\},B)|$ to obtain $(z+2y+x)\cdot N_0 + (2z-6x)\cdot N_1 + (2z-6y)\cdot N_2 + (2y+x-5z)\cdot N_3 \le 0$. By substituting $N_0 = |\mathcal{T}| - N_1 - N_2 - N_3$ and replacing $x=4$, $y=7$, $z=12$, we get $5N_1 + 8N_2 + 12N_3 \ge 5\cdot|\mathcal{T}|$. □

Let $m(\mathcal{T})$ be the number of rooted triplets in $\mathcal{T}$ consistent with the network returned by *Approximate*$(\mathcal{T})$.

LEMMA 25. *If $m(\mathcal{T}|Z) \ge q\cdot|\mathcal{T}|Z|$ for every $Z \in \{A,B,C\}$, then $m(\mathcal{T}) \ge q\cdot N_1 + \frac{2}{3}\cdot N_2 + N_3$.*

Proof. Every rooted triplet in $Z_2(A,B,C)$ is consistent with two of $Network_A$, $Network_B$, and $Network_C$, and every rooted triplet in $Z_3(A,B,C)$ is consistent with all of these three networks. Thus, $Network_X$ returned by *Approximate* must be consistent with at least $\frac{2}{3}\cdot N_2 + N_3$ of the rooted triplets in $Z_2(A,B,C)\cup Z_3(A,B,C)$. Also, each of $Network_A$, $Network_B$, and $Network_C$ is consistent with $m(\mathcal{T}|A) + m(\mathcal{T}|B) + m(\mathcal{T}|C) \ge q\cdot(|\mathcal{T}|A| + |\mathcal{T}|B| + |\mathcal{T}|C|) = q\cdot N_1$ of the rooted triplets in $Z_1(A,B,C)$. Thus, in total, $Network_X$ is consistent with at least $q\cdot N_1 + \frac{2}{3}\cdot N_2 + N_3$ rooted triplets in $\mathcal{T}$. □

THEOREM 9. $m(\mathcal{T}) \ge \frac{5}{12}\cdot|\mathcal{T}|$.

Proof. By induction on $|L|$. Base case ($|L| = 3$): Steps 3 and 4 of Algorithm *Approximate* construct a network consistent with at least 2/3 of the rooted triplets in $\mathcal{T}$; i.e., $m(\mathcal{T}) \ge \frac{5}{12}\cdot|\mathcal{T}|$. Inductive case ($|L| > 3$) Step 2 of *Approximate* recursively constructs three networks K_A, K_B, K_C for $\mathcal{T}|A$, $\mathcal{T}|B$, and $\mathcal{T}|C$, respectively. By the induction assumption, $m(\mathcal{T}|X) \ge \frac{5}{12}\cdot|\mathcal{T}|X|$ for each $X \in \{A,B,C\}$. By Lemmas 24 and 25, $m(\mathcal{T}) \ge \frac{5}{12}\cdot N_1 + \frac{2}{3}\cdot N_2 + N_3 \ge \frac{5}{12}\cdot|\mathcal{T}|$. □

Finally, the algorithm's running time is given by the following theorem.

THEOREM 10. *The time complexity of Algorithm Approximate is $O(n\cdot|\mathcal{T}|^3)$.*

Proof. Denote by $t(\mathcal{T})$ and $f(\mathcal{T})$ the running times of *Approximate*$(\mathcal{T})$ and *LeafPartition*$(\mathcal{T})$, respectively. We have $t(\mathcal{T}) = f(\mathcal{T}) + t(\mathcal{T}|A) + t(\mathcal{T}|B) + t(\mathcal{T}|C)$.

In *LeafPartition*, step 2 is performed at most $12 \cdot |\mathcal{T}|$ times in total. Every time, the algorithm needs to compute $O(n)$ values of *score*, and each *score* can be computed in $O(|\mathcal{T}|)$ time. Steps 1 and 3 can easily be implemented in $O(|\mathcal{T}|)$ time. Therefore, $f(\mathcal{T}) = O(n \cdot |\mathcal{T}|^2)$.

Furthermore, $|\mathcal{T}|A| + |\mathcal{T}|B| + |\mathcal{T}|C| < |\mathcal{T}|$. Solving the recurrence for $t(\mathcal{T})$ gives us $t(\mathcal{T}) = O(n \cdot |\mathcal{T}|^3)$. □

Acknowledgment. We thank Ho-Leung Chan for his comments on this paper.

REFERENCES

[1] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM J. Comput., 10 (1981), pp. 405–421.

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

[3] B. Chor, M. Hendy, and D. Penny, *Analytic solutions for three-taxon ML_{MC} trees with variable rates across sites*, in Proceedings of the 1st Annual Workshop on Algorithms in Bioinformatics (WABI 2001), Lecture Notes in Comput. Sci. 2149, Springer-Verlag, Berlin, 2001, pp. 204–213.

[4] C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung, *Computing the maximum agreement of phylogenetic networks*, Theoret. Comput. Sci., 335 (2005), pp. 93–107.

[5] M. Garey and D. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[6] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin, *Inferring ordered trees from local constraints*, in Proceedings the 4th Annual Australasian Theory Symposium of Computing (CATS'98), Aust. Comput. Sci. Commun. 20, Springer-Verlag, Singapore, 1998, pp. 67–76.

[7] L. Gąsieniec, J. Jansson, A. Lingas, and A. Östlin, *On the complexity of constructing evolutionary trees*, J. Comb. Optim., 3 (1999), pp. 183–197.

[8] D. Gusfield, S. Eddhu, and C. Langley, *Optimal, efficient reconstruction of phylogenetic networks with constrained recombination*, J. Bioinform. Comput. Biol., 2 (2004), pp. 173–213.

[9] J. Hein, *Reconstructing evolution of sequences subject to recombination using parsimony*, Math. Biosci., 98 (1990), pp. 185–200.

[10] M. R. Henzinger, V. King, and T. Warnow, *Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology*, Algorithmica, 24 (1999), pp. 1–13.

[11] J. Holm, K. de Lichtenberg, and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree,* 2*-edge, and biconnectivity*, J. ACM, 48 (2001), pp. 723–760.

[12] D. H. Huson, T. Dezulian, T. Klöpper, and M. Steel, *Phylogenetic super-networks from partial trees*, in Proceedings of the 4th Annual Workshop on Algorithms in Bioinformatics (WABI 2004), Lecture Notes in Comput. Sci. 3240, Springer-Verlag, Berlin, 2004, pp. 388–399.

[13] J. Jansson, *On the complexity of inferring rooted evolutionary trees*, in Proceedings of the Brazilian Symposium on Graphs, Algorithms, and Combinatorics (GRACO 2001), Electron. Notes Discrete Math. 7, Elsevier, 2001, pp. 121–125.

[14] J. Jansson, J. H.-K. Ng, K. Sadakane, and W.-K. Sung, *Rooted maximum agreement supertrees*, Algorithmica, 43 (2005), pp. 293–307.

[15] J. Jansson and W.-K. Sung, *Inferring a level-*1 *phylogenetic network from a dense set of rooted triplets*, Theoret. Comput. Sci., to appear.

[16] T. Jiang, P. Kearney, and M. Li, *A polynomial time approximation scheme for inferring evolutionary trees from quartet topologies and its application*, SIAM J. Comput., 30 (2001), pp. 1942–1961.

[17] S. Kannan, E. Lawler, and T. Warnow, *Determining the evolutionary tree using experiments*, J. Algorithms, 21 (1996), pp. 26–50.

[18] P. Kearney, *Phylogenetics and the quartet method*, in Current Topics in Computational Molecular Biology, T. Jiang, Y. Xu, and M. Q. Zhang, eds., The MIT Press, Cambridge, MA, 2002, pp. 111–133.

[19] L. Nakhleh, T. Warnow, C. R. Linder, and K. St. John, *Reconstructing reticulate evolution in species—Theory and practice*, J. Comput. Biol., 12 (2005), pp. 796–811.

[20] M. P. Ng, M. Steel, and N. C. Wormald, *The difficulty of constructing a leaf-labelled tree including or avoiding given subtrees*, Discrete Appl. Math., 98 (2000), pp. 227–235.

[21] M. P. Ng and N. C. Wormald, *Reconstruction of rooted trees from subtrees*, Discrete Appl. Math., 69 (1996), pp. 19–31.

[22] D. Posada and K. A. Crandall, *Intraspecific gene genealogies: Trees grafting into networks*, Trends Ecol. Evol., 16 (2001), pp. 37–45.

[23] M. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.

[24] L. Wang, K. Zhang, and L. Zhang, *Perfect phylogenetic networks with recombination*, J. Comput. Biol., 8 (2001), pp. 69–78.

[25] B. Y. Wu, *Constructing the maximum consensus tree from rooted triples*, J. Comb. Optim., 8 (2004), pp. 29–39.