JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 26, Number 9, 2019
© Mary Ann Liebert, Inc.
Pp. 893–907
DOI: 10.1089/cmb.2019.0033

An Efficient Algorithm for the Rooted Triplet Distance Between Galled Trees

JESPER JANSSON,[1] RAMESH RAJABY,[2,3] and WING-KIN SUNG[2,4]

ABSTRACT

The previous fastest algorithm for computing the rooted triplet distance between two input galled trees (i.e., phylogenetic networks whose cycles are vertex-disjoint) runs in $O(n^{2.687})$ time, where n is the cardinality of the leaf label set. In this article, we present an $O(n \log n)$-time solution. Our strategy is to transform the input so that the answer can be obtained by applying an existing $O(n \log n)$-time algorithm for the simpler case of two phylogenetic trees a constant number of times. The new algorithm has been implemented, and applying it to pairs of randomly generated galled trees with up to 500,000 leaves confirms that it is fast in practice.

Keywords: algorithm, computational complexity, galled tree, implementation, phylogenetic network comparison, rooted triplet.

1. INTRODUCTION

Measuring the similarity between phylogenetic trees is essential for evaluating the accuracy of methods for phylogenetic reconstruction (Kuhner and Felsenstein, 1994). The *rooted triplet distance* (Dobson, 1975) between two rooted phylogenetic trees having the same leaf label sets is given by the number of phylogenetic trees of size three that are embedded subtrees in either one of the input trees, but not the other. Since two phylogenetic trees with a lot of branching structure in common will typically share many such subtrees, the rooted triplet distance provides a natural measure of how dissimilar the two trees are.

A naive algorithm can compute the rooted triplet distance between two input rooted phylogenetic trees in $O(n^3)$ time, where n is the cardinality of the leaf label set, by directly checking each of the $\binom{n}{3}$ different cardinality-3 subsets of the leaf label set in $O(1)$ time after some linear-time preprocessing (see, e.g., Jansson and Lingas, 2014). More efficient algorithms have been developed (Bansal et al., 2011; Brodal et al., 2013; Critchlow et al., 1996; Jansson and Rajaby, 2017), and the asymptotically fastest one (Brodal et al., 2013) solves the problem in $O(n \log n)$ time.

[1]Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong.
[2]School of Computing, National University of Singapore, Singapore, Singapore.
[3]NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore, Singapore, Singapore.
[4]Genome Institute of Singapore, Singapore, Singapore.

A preliminary version of this article appeared in *Proceedings of the 4th International Conference on Algorithms for Computational Biology* (AlCoB 2017), volume 10252 of *Lecture Notes in Computer Science*, pp. 115–126, Springer International Publishing AG, 2017.

Gambette and Huber (2012) extended the rooted triplet distance from the phylogenetic tree setting to the *phylogenetic network* setting. In a *phylogenetic network* (Huson et al., 2010; Morrison, 2011), internal nodes are allowed to have more than one parent. Phylogenetic networks enable scientists to represent more complex evolutionary relationships than phylogenetic trees, for example, involving horizontal gene transfer events, or to visualize conflicting branching structure among a collection of two or more phylogenetic trees. The special case of a phylogenetic network in which all underlying cycles are vertex-disjoint is called a *galled tree* (Gusfield et al., 2004; Huson et al., 2010; Wang et al., 2000). Galled trees may be sufficient in cases where a phylogenetic tree is not good enough but it is known that only a few reticulation events have happened; see figure 9.22 in Huson et al. (2010) for a biological example. For a summary of other distances for comparing two galled trees such as the Robinson–Foulds distance, the tripartitions distance, the μ-distance, and the split nodal distance, see Cardona et al. (2011).

The fastest known algorithm for computing the rooted triplet distance between two galled trees relies on triangle counting and runs in $O(n^{2.687})$ time (Jansson and Lingas, 2014). More precisely, its time complexity is $O(n^{(3+\omega)/2})$, where ω is the exponent in the running time of the fastest existing method for matrix multiplication. Since $\omega < 2$ is impossible, the running time for computing the rooted triplet distance between two galled trees using the algorithm from Jansson and Lingas (2014) will never be better than $O(n^{2.5})$. In this article, we present an algorithm for the case of galled trees that does not use triangle counting but instead transforms the input to an appropriately defined set of *phylogenetic trees* to which the $O(n \log n)$-time algorithm of Brodal et al. (2013) is applied a constant number of times. Basically, in any galled tree, removing one of the two edges leading to an indegree-2 vertex in every cycle yields a tree that still contains most of the branching information, and we show how to compensate for what is lost by doing so while avoiding double counting. The resulting time complexity of our new algorithm is $O(n \log n)$.

The article is organized as follows. Section 2 defines the problem formally. Next, Section 3 presents the new algorithm and its analysis. An implementation of the algorithm along with experimental results is presented in Section 4. Finally, Section 5 contains some concluding remarks.

2. PROBLEM DEFINITIONS

We recall the following definitions from Jansson and Lingas (2014).

A *rooted phylogenetic tree* (from here on simply referred to as a *phylogenetic tree*) is an unordered rooted tree in which every internal node has at least two children and all leaves are distinctly labeled. A *phylogenetic network* is a directed acyclic graph with a single root vertex and a set of distinctly labeled leaves, and no vertices having both indegree 1 and outdegree 1. A *reticulation vertex* in a phylogenetic network is any vertex of indegree >1. For any phylogenetic network N, define its *underlying undirected graph* as the undirected graph obtained by replacing every directed edge in N by an undirected edge. A *cycle* C in a phylogenetic network is any subgraph with at least three edges whose corresponding subgraph in the underlying undirected graph is a cycle, and a vertex of C that is an ancestor of all vertices on C is called a *root* of C. A phylogenetic network is called a *galled tree* if all of its cycles are vertex-disjoint (Gusfield et al., 2004; Huson et al., 2010; Wang et al., 2000). For example, in Figure 1, N_1 is a galled tree and N_2 is a phylogenetic tree. Note that every reticulation vertex in a galled tree must have indegree 2. Every cycle C in a galled tree (also called a *gall*) has exactly one root (also referred to as its *split vertex*) and one reticulation vertex, and C consists of two directed internally disjoint paths from its split vertex to its reticulation vertex.

A phylogenetic tree with exactly three leaves is called a *rooted triplet*. A rooted triplet leaf-labeled by $\{a, b, c\}$ with one internal node is called a *fan triplet* and is denoted by $a|b|c$, whereas a rooted triplet leaf-labeled by $\{a, b, c\}$ with two internal nodes is called a *resolved triplet*; in the latter case, there are three possibilities, denoted by $ab|c$, $ac|b$, and $bc|a$, corresponding to when the lowest common ancestor of the two leaves labeled by a and b, or a and c, or b and c, respectively, is a proper descendant of the root. Let a, b, c be three leaf labels in a phylogenetic network N. The fan triplet $a|b|c$ is *consistent with* N if and only if N contains a vertex v and three directed paths from v to a, from v to b, and from v to c that are vertex-disjoint except for in the common start vertex v. Similarly, the resolved triplet $ab|c$ is *consistent with* N if and only if N contains two vertices v and w ($v \neq w$) such that there are four directed paths of nonzero length from v to a, from v to b, from w to v, and from w to c that are vertex-disjoint except for in the vertices v and w, and furthermore, the path from w to c does not pass through v. For any phylogenetic network N, $t(N)$ denotes the set of all rooted triplets (i.e., fan triplets as well as resolved triplets) that are consistent with N.

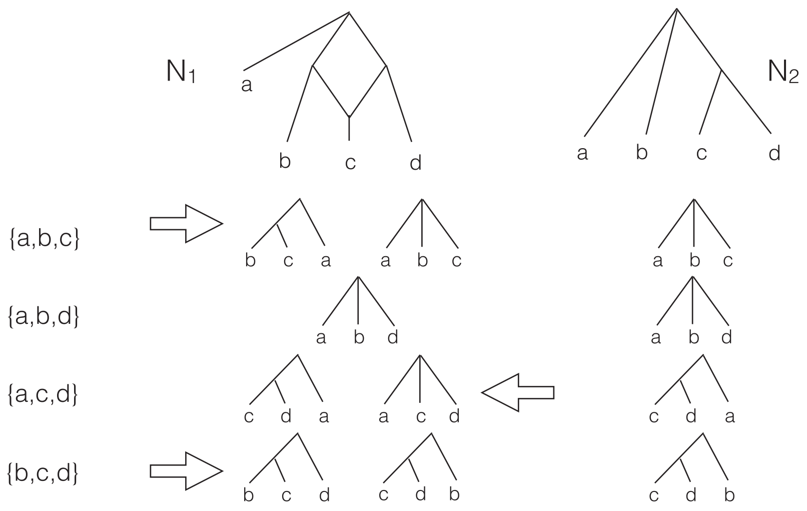

FIG. 1. An example. The galled tree N_1 on the left and the phylogenetic tree N_2 on the right have the same leaf label set $\{a, b, c, d\}$. (In this figure and in the figures that will follow, all directed edges are assumed to be oriented in the downward direction.) N_1 is consistent with the fan triplets $a|b|c$, $a|b|d$, and $a|c|d$, and the resolved triplets $bc|a$, $cd|a$, $bc|d$, and $cd|b$. N_2 is consistent with the fan triplets $a|b|c$ and $a|b|d$ and the resolved triplets $cd|a$ and $cd|b$. The arrows indicate rooted triplets that appear in only one of N_1 and N_2. According to the definitions, $fcount(N_1, N_2) = 2$, $rcount(N_1, N_2) = 2$, $count(N_1, N_2) = 4$, and $d_{rt}(N_1, N_2) = 3$.

Definition 1. *(Adapted from Gambette and Huber, 2012) Let N_1, N_2 be two phylogenetic networks on the same leaf label set L. The* rooted triplet distance between N_1 and N_2, *denoted by $d_{rt}(N_1, N_2)$, is the number of fan triplets and resolved triplets with leaf labels from L that are consistent with exactly one of N_1 and N_2.*

See also Section 3.2 in Jansson and Lingas (2014) for a discussion of the aforementioned definition. Refer to Figure 1 for an example.

Define $fcount(N_1, N_2)$ as the number of fan triplets consistent with both N_1 and N_2, $rcount(N_1, N_2)$ as the number of resolved triplets consistent with both N_1 and N_2, and $count(N_1, N_2) = fcount(N_1, N_2) + rcount(N_1, N_2)$. Note that for $i \in \{1, 2\}$, we have $|t(N_i)| = count(N_i, N_i)$. Then one can compute $d_{rt}(N_1, N_2)$ by the formula

$$d_{rt}(N_1, N_2) = count(N_1, N_1) + count(N_2, N_2) - 2 \cdot count(N_1, N_2)$$

The following result was shown in Brodal et al. (2013):

Theorem 1. *(Brodal* et al., *2013) If T_1, T_2 are two phylogenetic trees on the same leaf label set L then $fcount(T_1, T_2)$ and $rcount(T_1, T_2)$ (and hence, $d_{rt}(T_1, T_2)$) can be computed in $O(n \log n)$ time, where $n = |L|$.*

From now on, we assume that the input consists of two galled trees N_1 and N_2 over a leaf label set L and that the objective is to compute $d_{rt}(N_1, N_2)$. We define $n = |L|$. It is known that $d_{rt}(N_1, N_2)$ can be computed in $O(n^{2.687})$ time (Jansson and Lingas, 2014). We will show below how to do it faster by using Theorem 1, which yields our main result:

Theorem 2. *If N_1, N_2 are two galled trees on the same leaf label set L then $fcount(N_1, N_2)$ and $rcount(N_1, N_2)$ (and hence, $d_{rt}(N_1, N_2)$) can be computed in $O(n \log n)$ time, where $n = |L|$.*

3. THE NEW ALGORITHM

Section 3.1 describes how to compute $rcount(N_1, N_2)$ efficiently, whereas Section 3.2 is focused on $fcount(N_1, N_2)$. (Both subsections rely on Theorem 1.) In addition to the definitions provided in Section 2, the following notation and terminology will be needed.

Suppose that N is a galled tree. For each internal vertex in N, fix an arbitrary left-to-right ordering of its children. Also, arbitrarily designate one of the two parents of each reticulation vertex as its left parent and the other one as its right parent. Then $N^{\searrow}$ is the phylogenetic tree obtained by removing the right parent edge of every reticulation vertex in N and contracting every edge (if any) leading to a vertex with exactly one child. Similarly, $N^{\swarrow}$ is the phylogenetic tree formed by removing the left parent edge of every reticulation vertex in N and contracting all edges leading to degree-1 vertices. Let $N^{\downarrow}$ be the phylogenetic tree formed by removing both the left and right parent edges of the reticulation vertex h in each gall, inserting a new edge between the gall's split vertex and h, and contracting all edges leading to degree-1 vertices. See Figure 2 for an illustration.

Let $r(N)$ denote the root of N and let $gall(N)$ be the set of all galls in N. For each $Q \in gall(N)$, let $r(Q)$ be the root of Q and h_Q the reticulation vertex of Q. Let Q_L and Q_R be the *left and right paths of* Q, obtained by removing $r(Q)$, h_Q, and all edges incident to $r(Q)$ and h_Q.

Suppose that $\{x, y, z\}$ is a cardinality-3 subset of the leaf label set of N. Then $\{x, y, z\}$ is called an *ambiguous triplet* if N contains a gall Q such that (i) x, y, z are in three different subtrees attached to Q or $r(Q)$; (ii) exactly one of x, y, z is in the subtree attached to h_Q; and (iii) at least one of x, y, z is in a subtree attached to Q_L or Q_R. The ambiguous triplets are partitioned into type-A, type-B, and type-C triplets, defined as follows (see Fig. 3 for an illustration).

- $\{x, y, z\}$ is a *type-A triplet of* N if there exists a gall Q in N such that two leaves in $\{x, y, z\}$ appear in two different subtrees attached to the same Q_δ ($\delta = L$ or R), whereas the remaining leaf appears in the subtree rooted at h_Q. Furthermore, if $\{x, y, z\}$ is a type-A triplet of N and $\{x, y, z\}$ are attached to a gall Q in N with z appearing in the subtree rooted at h_Q then $\{x, y, z\}$ is called a type-A triplet of N with *reticulation leaf* z.

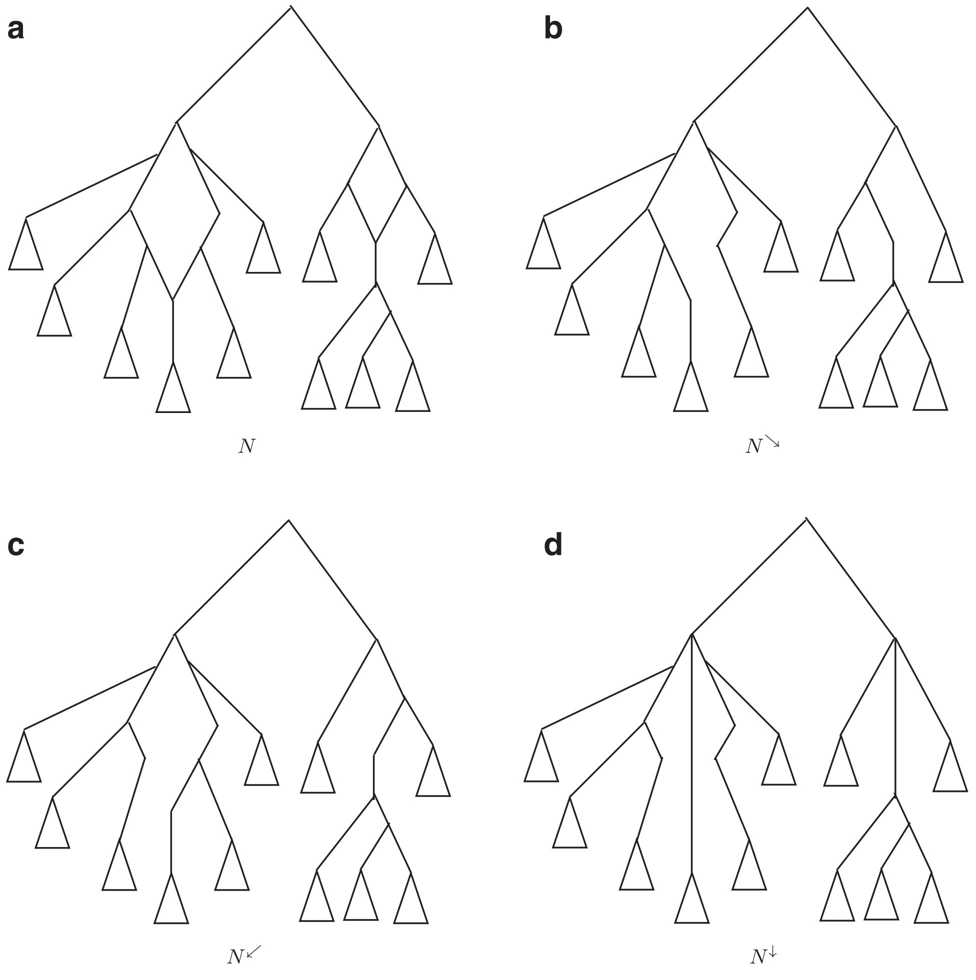

FIG. 2. The galled tree N in **(a)** is transformed into three phylogenetic trees: **(b)** $N^{\searrow}$, **(c)** $N^{\swarrow}$, and **(d)** $N^{\downarrow}$.

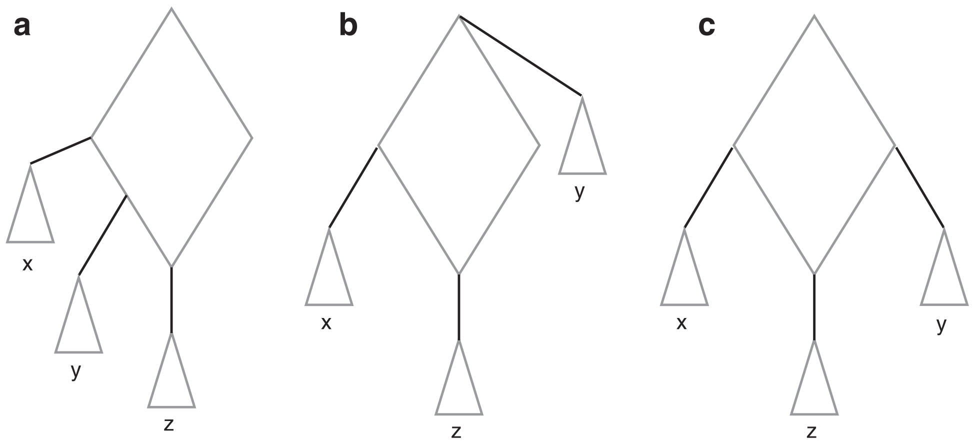

FIG. 3. **(a–c)** Illustrates the definitions of type-A, type-B, and type-C triplets, respectively.

- $\{x, y, z\}$ is a *type-B triplet of N* if there exists a gall Q in N such that, among the three leaves in $\{x, y, z\}$, one leaf is attached to $r(Q)$ but is not in Q, another leaf appears in a subtree attached to Q_L (or Q_R), and the last leaf appears in the subtree rooted at h_Q.
- $\{x, y, z\}$ is a *type-C triplet of N* if there exists a gall Q in N such that, among the three leaves in $\{x, y, z\}$, one leaf appears in a subtree attached to Q_L, another leaf appears in a subtree attached to Q_R, and the last leaf appears in the subtree rooted at h_Q.

Lemma 1. *An ambiguous triplet must belong to exactly one among the type-A, type-B, and type-C categories.*

Proof. By definition, each ambiguous triplet has one leaf in the subtree rooted at h_Q and one leaf in a subtree attached to Q_δ, where $\delta \in \{L, R\}$. Depending on the third leaf:

- If it is attached to a subtree in Q_δ, then the triplet is of type-A.
- If it is attached to a subtree in Q_γ, $\gamma \neq \delta$, then the triplet is of type-C.
- If it is not attached to a subtree in Q but attached to a subtree of $r(Q)$, then the triplet is of type-B.

Since the three options are mutually exclusive and exhaustive, the lemma follows. □

3.1. Counting common resolved triplets in N_1 and N_2

The main idea of our algorithm for computing $rcount(N_1, N_2)$ is to count the common resolved triplets between N_1 and each of the three trees $N_2^{\swarrow}$, $N_2^{\searrow}$, and $N_2^{\downarrow}$ and then combine the results appropriately. However, there is one type of triplet, which we miss by doing so, depending on the position of its leaves within the gall containing its lowest common ancestor. This case corresponds to the type-A ambiguous triplets and we count these triplets separately with an extra function $rcount_A$ (see Lemma 3). The problem of counting the common resolved triplets between N_1 and a tree is similarly reduced to three instances of counting the common resolved triplets between two phylogenetic trees (covered by Theorem 1) and adjusting the result by using $rcount_A$.

We now present the details. Define $rcount_A(N_1, N_2)$ as the number of resolved triplets $xy|z$ in both N_1 and N_2 such that $\{x, y, z\}$ is a type-A triplet of N_2 with reticulation leaf z. Similarly, define $rcount_A^*(N_1, N_2)$ as the number of resolved triplets $xy|z$ in both N_1 and N_2 such that $\{x, y, z\}$ is a type-A triplet of both N_1 and N_2 with reticulation leaf z. Observe that in general, $rcount_A(N_1, N_2) \neq rcount_A(N_2, N_1)$, but $rcount_A^*(N_1, N_2) = rcount_A^*(N_2, N_1)$ always holds.

The following lemmas express the relationships between $rcount(N_1, N_2)$, $rcount_A(N_1, N_2)$, and $rcount_A^*(N_1, N_2)$.

Lemma 2. *Suppose that $xy|z$ is a resolved triplet such that x, y, z are in the leaf label set. Then $rcount(xy|z, N_2) = rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2)$.*

Proof. Any resolved triplet $xy|z$ either appears or does not appear in $t(N_2)$. Also, any ambiguous triplet is of type-A, type-B, or type-C. Hence, the following cases are possible:

– (1) $xy|z \notin t(N_2)$

– (2) $xy|z \in t(N_2)$:

- (2.1) $xy|z \in t(N_2)$ and $\{x, y, z\}$ is not ambiguous.
- (2.2) $xy|z \in t(N_2)$ and $\{x, y, z\}$ is a type-A triplet with reticulation leaf z in N_2.
- (2.3) $xy|z \in t(N_2)$ and $\{x, y, z\}$ is a type-A triplet with reticulation leaf x or y in N_2.
- (2.4) $xy|z \in t(N_2)$ and $\{x, y, z\}$ is a type-B or type-C triplet.

In case (1), $xy|z \notin t(N_2^{\swarrow}), t(N_2^{\searrow}), t(N_2^{\downarrow})$. Also, $rcount_A(xy|z, N_2) = 0$ since $\{x, y, z\}$ cannot be a type-A triplet. Hence, $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2) = 0$.

In case (2.1), since $xy|z \in t(N_2)$ and $\{x, y, z\}$ is not ambiguous, it follows that $xy|z \in t(N_2^{\swarrow}), t(N_2^{\searrow}), t(N_2^{\downarrow})$. Also, $rcount_A(xy|z, N_2) = 0$. Hence, we have $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2) = 1$.

In case (2.2), $xy|z$ appears in either $N_2^{\swarrow}$ or $N_2^{\searrow}$, but not both, and in $N_2^{\downarrow}$. Because $rcount_A(xy|z, N_2) = 1$ by definition, we see that $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2) = 1$.

Finally, in cases (2.3) and (2.4), $xy|z$ appears in either $N_2^{\swarrow}$ or $N_2^{\searrow}$, but not both, and $xy|z$ does not appear in $N_2^{\downarrow}$. Also, $rcount_A(xy|z, N_2) = 0$ by definition. Thus, $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) = 1$. □

Lemma 3. $rcount(N_1, N_2) = rcount(N_1, N_2^{\swarrow}) + rcount(N_1, N_2^{\searrow}) - rcount(N_1, N_2^{\downarrow}) + rcount_A(N_1, N_2)$.

Proof. Write $rcount(N_1, N_2) = \sum_{xy|z \in N_1} rcount(xy|z, N_2)$. For $xy|z \in t(N_1)$, by Lemma 2, we have $rcount(xy|z, N_2) = rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2)$. □

The next lemma describes the formula for $rcount_A(N_1, N_2)$.

Lemma 4. $rcount_A(N_1, N_2) = rcount_A(N_1^{\swarrow}, N_2) + rcount_A(N_1^{\searrow}, N_2) - rcount_A(N_1^{\downarrow}, N_2) + rcount_A^*(N_1, N_2)$.

Proof. For $xy|z \in t(N_1)$, by an argument identical to the one in the proof of Lemma 2, we have $rcount(N_1, xy|z) = rcount(N_1^{\swarrow}, xy|z) + rcount(N_1^{\searrow}, xy|z) - rcount(N_1^{\downarrow}, xy|z) + rcount'_A(N_1, xy|z)$, where $rcount'_A(N_1, xy|z) = 1$ if $\{x, y, z\}$ is a type-A triplet of N_1 with reticulation leaf z, and $rcount'_A(N_1, xy|z) = 0$ otherwise.

Define $W = \{xy|z : \{x, y, z\}$ is a type-A triplet of N_2 with reticulation leaf $z\}$. Then $rcount_A(N_1, N_2) = \sum_{xy|z \in W} rcount(N_1, xy|z) = \sum_{xy|z \in W} (rcount(N_1^{\swarrow}, xy|z) + rcount(N_1^{\searrow}, xy|z) - rcount(N_1^{\downarrow}, xy|z) + rcount'_A(N_1, xy|z)) = rcount_A(N_1^{\swarrow}, N_2) + rcount_A(N_1^{\searrow}, N_2) - rcount_A(N_1^{\downarrow}, N_2) + rcount_A^*(N_1, N_2)$. □

Next, we discuss the computation of $rcount_A(T_1, N_2)$ and $rcount_A^*(N_1, N_2)$, where N_1 and N_2 are galled trees and T_1 is a phylogenetic tree. (The case of $rcount_A(N_1, T_2)$ where N_1 is a galled tree and T_2 is a tree, needed in Lemma 3, is symmetric.) For $\delta \in \{L, R\}$, denote by τ_δ the tree formed by attaching all subtrees attached to Q_δ to a common root. For any galled tree N, let N^L be a tree formed from $N^{\searrow}$ by contracting all edges on Q_L for every gall Q (observe that the edges $(r(Q), r(\tau_L))$ and $(r(\tau_L), h_Q)$ are in N^L). Define N^{LL} to be a tree formed from $N^{\downarrow}$ by replacing all the trees attached to Q_L with τ_L, inserting a new vertex m_Q between $r(Q)$ and h_Q, and replacing the edge $(r(Q), r(\tau_L))$ with $(m_Q, r(\tau_L))$. See Figure 4 for an example. N^R and N^{RR} are defined analogously.

Lemma 5. *For $\delta \in \{L, R\}$, the following two properties hold.*

- *All resolved triplets in N^δ are in $N^{\delta\delta}$.*
- *All additional resolved triplets $xy|z$ in $N^{\delta\delta}$, that is, those not in N^δ, are type-A triplets of N with reticulation leaf z.*

Proof. Consider any three leaves $\{x, y, z\}$. If zero, one, or all three of them belong to τ_δ then $N^\delta|\{x, y, z\} = N^{\delta\delta}|\{x, y, z\}$, where $N|W$ is the galled subtree of N formed by retaining only leaves in W. Otherwise, when two of $\{x, y, z\}$ belong to τ_δ, let γ be the lowest common ancestor of x, y, z in $N^{\delta\delta}$. There are two cases:

1. If γ is a proper ancestor of m_Q, then $N^\delta|\{x, y, z\} = N^{\delta\delta}|\{x, y, z\} = xy|z$.
2. Otherwise, $\gamma = m_Q$ and then $N^\delta|\{x, y, z\} = x|y|z$ while $N^{\delta\delta}|\{x, y, z\} = xy|z$.

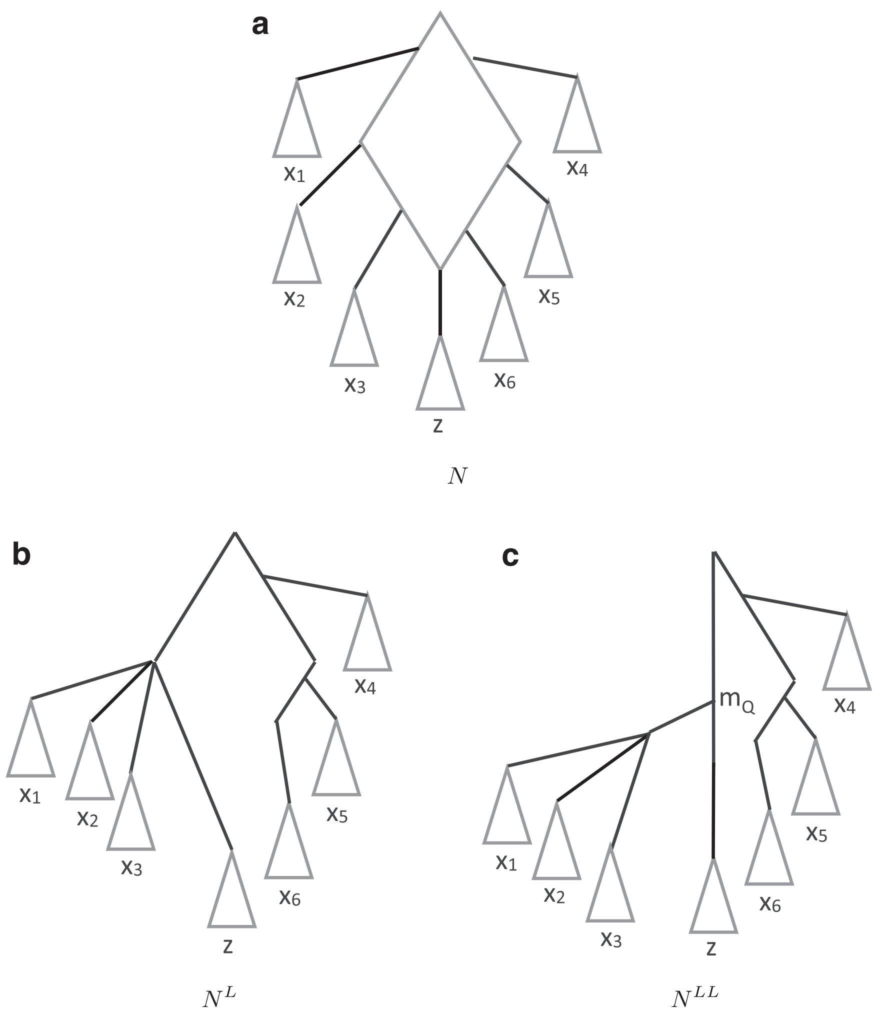

FIG. 4. (**a**) shows a galled tree N, (**b**) shows N^L, and (**c**) shows N^{LL}.

Hence, all resolved triplets in N^{δ} are in $N^{\delta\delta}$.

Finally, $xy|z$ is a type-A triplet of N with reticulation leaf z if and only if $\gamma = m_Q$. This shows that the second property also holds. □

Lemma 6. $rcount_A(T_1, N_2) = \sum_{\delta \in \{L,R\}} \left(rcount(T_1, N_2^{\delta\delta}) - rcount(T_1, N_2^{\delta}) \right)$.

Proof. By Lemma 5, all type-A triplets in N_2 appear in $N_2^{\delta\delta}$ but not in N_2^{δ} for some $\delta \in \{L, R\}$. The lemma follows. □

Lemma 7. $rcount_A^*(N_1, N_2) = \sum_{\delta \in \{L,R\}} \left(rcount_A(N_1^{\delta\delta}, N_2) - rcount_A(N_1^{\delta}, N_2) \right)$.

Proof. (Similar to the proof of Lemma 6.) By Lemma 5, all type-A triplets in N_1 appear in $N_1^{\delta\delta}$ but not in N_1^{δ} for some $\delta \in \{L, R\}$. □

The algorithm in Figure 5 computes $rcount(N_1, N_2)$ using Lemmas 3, 4, 6, and 7.

Lemma 8. *The algorithm* $rcount(N_1, N_2)$ *in* Figure 5 *makes a total of* 49 *calls to* $rcount(T_1, T_2)$*, where* T_1 *and* T_2 *are phylogenetic trees.*

Proof. First observe that $rcount_A(T_1, N_2)$ is obtained by making four calls to $rcount(T_1, T_2)$, and $rcount_A^*(N_1, N_2)$ by four calls to $rcount_A(T_1, N_2)$. Next, $rcount_A(N_1, N_2)$ is obtained by three calls to $rcount_A(T_1, N_2)$ and one call to $rcount_A^*(N_1, N_2)$. In total, $rcount_A(N_1, N_2)$ uses $3 \cdot 4 + 1 \cdot 4 \cdot 4 = 28$ calls to $rcount(T_1, T_2)$.

```
Algorithm rcount$(N_1, N_2)$
    Return $rcount(N_1, N_2^{\swarrow}) + rcount(N_1, N_2^{\searrow}) - rcount(N_1, N_2^{\downarrow}) + rcount_A(N_1, N_2)$;

Algorithm rcount$(N_1, T_2)$
    Return $rcount(N_1^{\swarrow}, T_2) + rcount(N_1^{\searrow}, T_2) - rcount(N_1^{\downarrow}, T_2) + rcount_A(T_2, N_1)$;

Algorithm rcount$_A(N_1, N_2)$
    Return $rcount_A(N_1^{\swarrow}, N_2) + rcount_A(N_1^{\searrow}, N_2) - rcount_A(N_1^{\downarrow}, N_2) + rcount_A^*(N_1, N_2)$;

Algorithm rcount$_A(T_1, N_2)$
    Return $\sum_{\delta \in \{L,R\}} \big(rcount(T_1, N_2^{\delta\delta}) - rcount(T_1, N_2^{\delta})\big)$;

Algorithm rcount$_A^*(N_1, N_2)$
    Return $\sum_{\delta \in \{L,R\}} \big(rcount_A(N_1^{\delta\delta}, N_2) - rcount_A(N_1^{\delta}, N_2)\big)$;
```

FIG. 5. The algorithm for computing *rcount*(N_1, N_2).

Similarly, *rcount*(N_1, T_2) is obtained by making three calls to *rcount*(T_1, T_2) and one call to *rcount*$_A(T_2, N_1)$. In total, *rcount*(N_1, T_2) is computed by $3+1 \cdot 4 = 7$ calls to *rcount*(T_1, T_2).

Finally, *rcount*(N_1, N_2) is obtained by making three calls to *rcount*(N_1, T_2) and one call to *rcount*$_A(N_1, N_2)$. In total, *rcount*(N_1, N_2) makes $3 \cdot 7 + 1 \cdot 28 = 49$ calls to *rcount*(T_1, T_2). □

By Theorem 1, *rcount*(T_1, T_2) can be computed in $O(n \log n)$ time for any two trees T_1, T_2. Lemma 8 shows that the algorithm in Figure 5 makes a constant number of calls to *rcount*(T_1, T_2). Lastly, constructing each of the constant number of trees used as arguments to *rcount*(T_1, T_2) ($N_1^{\swarrow}$, $N_1^{\downarrow}$, etc.) takes $O(n)$ time. Thus, the total running time to obtain *rcount*(N_1, N_2) is $O(n \log n)$.

3.2. Counting common fan triplets in N_1 and N_2

To compute *fcount*(N_1, N_2), we modify the technique from the previous subsection. The main difference is that we count type-B and type-C triplets separately.

Define *fcount*$_{BC}(N_1, N_2)$ as the number of triplets $\{x, y, z\}$ such that $x|y|z$ is a fan triplet in N_1 and $\{x, y, z\}$ is a type-B or type-C triplet in N_2. Also, define *fcount*$^*_{BC}(N_1, N_2)$ as the number of type-B and type-C triplets $\{x, y, z\}$ that appear in both N_1 and N_2. Similar to what was done in Section 3.1 where *rcount*(N_1, N_2) was expressed using *rcount*$_A(N_1, N_2)$ and *rcount*$^*_A(N_1, N_2)$, we express *fcount*(N_1, N_2) using *fcount*$_{BC}(N_1, N_2)$ and *fcount*$^*_{BC}(N_1, N_2)$.

Lemma 9. *Let $x|y|z$ be a fan triplet. Then $fcount(x|y|z, N_2) = fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2)$.*

Proof. There are five cases:

1. $x|y|z \notin t(N_2)$ and $\{x, y, z\}$ is not a type-C triplet in N_2,
2. $x|y|z \notin t(N_2)$ and $\{x, y, z\}$ is a type-C triplet in N_2,
3. $x|y|z \in t(N_2)$ and $\{x, y, z\}$ is not ambiguous,
4. $x|y|z \in t(N_2)$ and $\{x, y, z\}$ is a type-B triplet in N_2,
5. $x|y|z \in t(N_2)$ and $\{x, y, z\}$ is ambiguous, but it is not a type-B triplet in N_2.

In case 1, $x|y|z \notin t(N_2^{\swarrow}), t(N_2^{\searrow}), t(N_2^{\downarrow})$. Also, $fcount_{BC}(x|y|z, N_2) = 0$ as it cannot be a type-B (since $x|y|z \notin t(N_2)$) or type-C (by hypothesis) triplet. Hence, $fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 0$.

In case 2, $x|y|z \notin t(N_2^{\swarrow}), t(N_2^{\searrow})$ and $x|y|z \in t(N_2^{\downarrow})$. Also, $fcount_{BC}(x|y|z, N_2) = 1$ since it is a type-C triplet. It follows that $fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 0$.

In case 3, since $x|y|z \in t(N_2)$ and $\{x, y, z\}$ is not ambiguous, we have $x|y|z \in t(N_2^{\swarrow}),\ t(N_2^{\searrow}), t(N_2^{\downarrow})$. Also, $fcount_{BC}(x|y|z, N_2) = 0$. Thus, $fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 1$.

In case 4, $x|y|z$ appears in either $N_2^{\swarrow}$ or $N_2^{\searrow}$, but not both. $x|y|z$ also appears in $N_2^{\downarrow}$. Furthermore, $fcount_{BC}(x|y|z, N_2) = 1$. Hence, $fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 1$.

In case 5, since $x|y|z \in t(N_2)$ and $\{x, y, z\}$ is not a type-B triplet, $fcount_{BC}(x|y|z, N_2) = 0$ (note that $\{x, y, z\}$ cannot be a type-C triplet since $x|y|z \in t(N_2)$ by hypothesis). In this case, two of $\{x, y, z\}$ are attached to the same vertex on Q_δ and the remaining leaf is a descendant of h_Q. Then, $x|y|z$ appears in either $N_2^{\swarrow}$ or $N_2^{\searrow}$, but not both. $x|y|z$ does not appear in $N_2^{\downarrow}$. Hence, $fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 1$. □

Lemma 10. $fcount(N_1, N_2) = fcount(N_1, N_2^{\swarrow}) + fcount(N_1, N_2^{\searrow}) - fcount(N_1, N_2^{\downarrow}) + fcount_{BC}(N_1, N_2)$.

Proof. Note that $fcount(N_1, N_2) = \sum_{x|y|z \in N_1} fcount(x|y|z, N_2)$. For $x|y|z \in t(N_1)$, Lemma 9 implies $fcount(x|y|z, N_2) = fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2)$. □

Lemma 11. $fcount_{BC}(N_1, N_2) = fcount_{BC}(N_1^{\swarrow}, N_2) + fcount_{BC}(N_1^{\searrow}, N_2) - fcount_{BC}(N_1^{\downarrow}, N_2) + fcount^*_{BC}(N_1, N_2)$.

Proof. For $x|y|z \in t(N_1)$, by arguing as in the proof of Lemma 9, we obtain $fcount(N_1, x|y|z) = fcount(N_1^{\swarrow}, x|y|z) + fcount(N_1^{\searrow}, x|y|z) - fcount(N_1^{\downarrow}, x|y|z) + fcount'_{BC}(N_1, x|y|z)$, where $fcount'_{BC}(N_1, x|y|z)$ equals the number of $\{x, y, z\}$ that are type-B or type-C triplets in N_1.

Next, let $W = \{x|y|z\ :\ \{x, y, z\} \text{ is a type} - \text{B or type} - \text{C triplet of } N_2\}$. Then it follows that $fcount_{BC}(N_1, N_2) = \sum_{x|y|z \in W} fcount(N_1, x|y|z) = \sum_{x|y|z \in W} (fcount(N_1^{\swarrow}, x|y|z) + fcount(N_1^{\searrow}, x|y|z) - fcount(N_1^{\downarrow}, x|y|z) + fcount'_{BC}(N_1, x|y|z)) = fcount_{BC}(N_1^{\swarrow}, N_2) + fcount_{BC}(N_1^{\searrow}, N_2) - fcount_{BC}(N_1^{\downarrow}, N_2) + fcount^*_{BC}(N_1, N_2)$. □

The rest of this subsection explains how to compute $fcount_{BC}(T_1, N_2)$ and $fcount^*_{BC}(N_1, N_2)$ efficiently.

A *caterpillar tree* is a binary phylogenetic tree in which every internal node has at least one leaf child. We define the *length* of a caterpillar tree as the number of edges on any longest path starting at the root and ending at a leaf. Hence a length-k caterpillar tree has k internal nodes and $k+1$ leaves. Given a galled tree N, we define N^B as the tree formed by performing the following steps on a copy of N (see Fig. 6 for an example of the construction):

- First replace every degree-k vertex v with $k > 2$, which is not a split vertex in N by a length-$(k-1)$ caterpillar tree P, and then replace the k leaves of P by the k children of v (in any arbitrary order).
- For every gall Q, if the split vertex $r(Q)$ is of degree $k > 2$, first create a length-$(k-3)$ caterpillar tree P, then replace the $k-2$ leaves of P by the $k-2$ children of $r(Q)$ that are not on the gall (in any arbitrary order), and finally replace the edges between $r(Q)$ and these $k-2$ children by a single edge from $r(Q)$ to $r(P)$. Next, create a new child u of $r(Q)$ and let $r(Q_L)$ and $r(Q_R)$ become children of u. Finally, remove the reticulation vertex h_Q's two parent edges, attach the subnetwork of N rooted at h_Q as a child of $r(Q)$, and contract every edge (if any) leading to a vertex with exactly one child.

Next, we define N^C as the tree obtained in exactly the same way as N^B except that for each gall Q, the subnetwork of N rooted at h_Q is attached as a sibling of $r(Q_L)$ and $r(Q_R)$ instead of as a child of $r(Q)$ (again, see Fig. 6 for an example of the construction).

The next lemma states how N, N^B, and N^C are related.

Lemma 12. *(1) $\{x, y, z\}$ is a type-B triplet of N if and only if $x|y|z$ is a fan triplet in N^B; (2) $\{x, y, z\}$ is a type-C triplet of N if and only if $x|y|z$ is a fan triplet in N^C.*

Proof. By construction, every vertex in N^B and N^C has two or three children. Any vertex in N^B and N^C with three children corresponds to a split vertex of a gall in N.

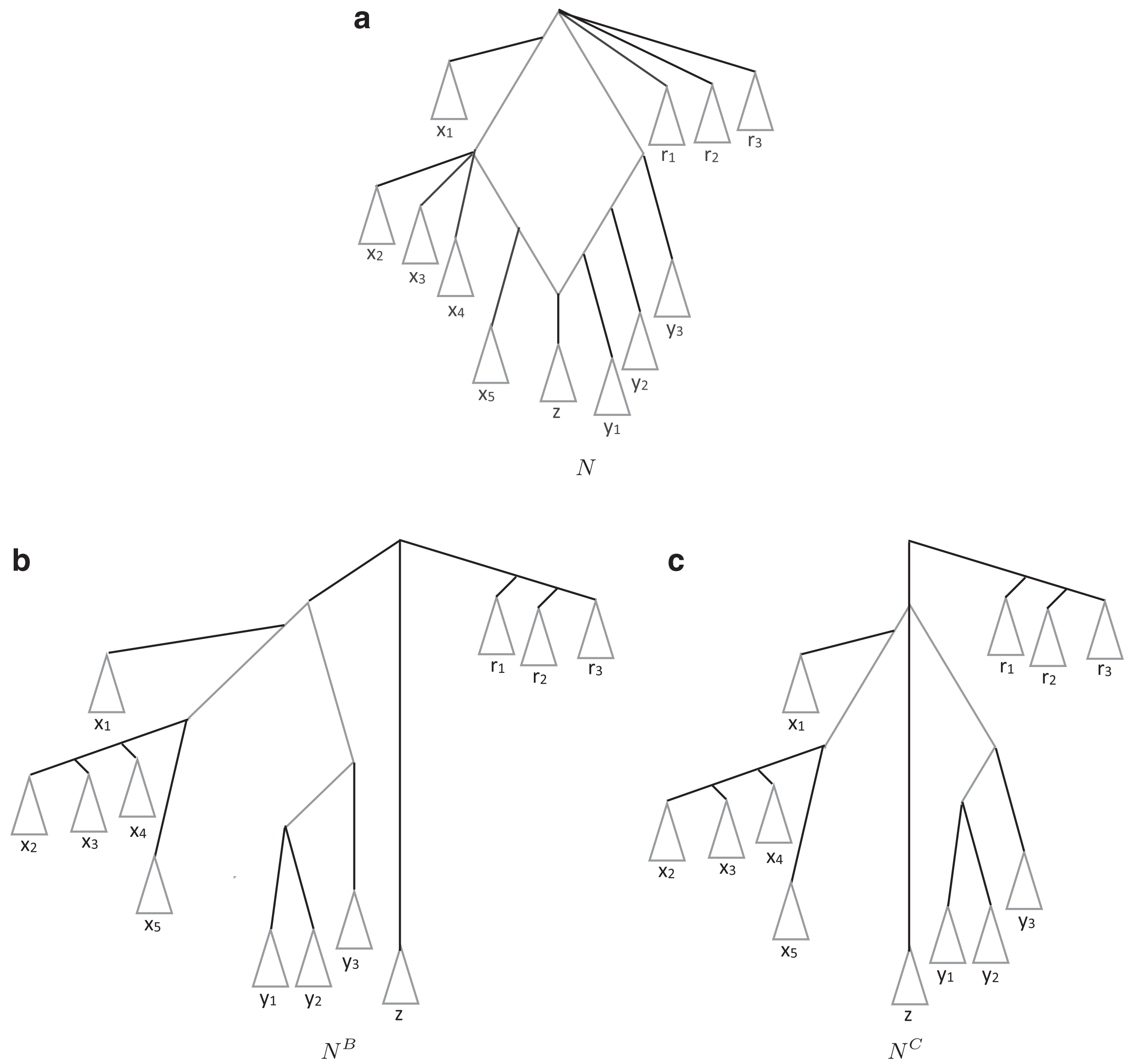

FIG. 6. **(a)** shows a galled tree N, **(b)** shows N^B, and **(c)** shows N^C.

For (1): ($\rightarrow$) If $\{x, y, z\}$ is a type-B triplet of N, there exists a gall Q in N such that x, y, z are in three different subtrees attached to Q where one leaf (say x) is in a subtree attached to Q_L or Q_R, another leaf (say y) is in a subtree attached to $r(Q)$, and the remaining leaf (say z) is in a subtree attached to h_Q. Then, by construction, $x|y|z$ is a fan triplet in N^B.

($\leftarrow$) If $x|y|z$ is a fan triplet in N^B, let u be the lowest common ancestor of x, y, z in N^B. u is of degree-3 and corresponds to a gall Q. This implies that x, y, z are in a subtree attached to $r(Q)$, a subtree attached to h_Q, and a subtree attached to Q_L or Q_R. Hence, $\{x, y, z\}$ is a type-B triplet of N.

For (2): ($\rightarrow$) If $\{x, y, z\}$ is a type-C triplet of N, there exists a gall Q in N such that x, y, z are in three different subtrees attached to Q, where one leaf (say x) is in a subtree attached to Q_L, another leaf (say y) is in a subtree attached to Q_R, and the remaining leaf (say z) is in a subtree attached to h_Q. Then, by construction, $x|y|z$ is a fan triplet in N^C.

($\leftarrow$) If $x|y|z$ is a fan triplet in N^C, let u be the lowest common ancestor of x, y, z in N^C. u is of degree-3 and corresponds to a gall Q. This implies that x, y, z are in a subtree attached to Q_L, a subtree attached to Q_R, and a subtree attached to h_Q. Hence, $\{x, y, z\}$ is a type-C triplet of N. $\square$

As a consequence, we have the following two lemmas.

```
Algorithm fcount($N_1, N_2$)
    Return $fcount(N_1, N_2^{\swarrow}) + fcount(N_1, N_2^{\searrow}) - fcount(N_1, N_2^{\downarrow}) + fcount_{BC}(N_1, N_2)$;

Algorithm fcount($N_1, T_2$)
    Return $fcount(N_1^{\swarrow}, T_2) + fcount(N_1^{\searrow}, T_2) - fcount(N_1^{\downarrow}, T_2) + fcount_{BC}(T_2, N_1)$;

Algorithm fcount$_{BC}$($N_1, N_2$)
    Return $fcount_{BC}(N_1^{\swarrow}, N_2) + fcount_{BC}(N_1^{\searrow}, N_2) - fcount_{BC}(N_1^{\downarrow}, N_2) + fcount^*_{BC}(N_1, N_2)$;

Algorithm fcount$_{BC}$($T_1, N_2$)
    Return $fcount(T_1, N_2^B) + fcount(T_1, N_2^C)$;

Algorithm fcount$^*_{BC}$($N_1, N_2$)
    Return $fcount(N_1^B, N_2^B) + fcount(N_1^B, N_2^C) + fcount(N_1^C, N_2^B) + fcount(N_1^C, N_2^C)$;
```

FIG. 7. The algorithm for computing *fcount*(N_1, N_2).

Lemma 13. *fcount*$_{BC}(T_1, N_2)$=*fcount*(T_1, N_2^B)+*fcount*(T_1, N_2^C).

Proof. By Lemma 12 (1), $\{x, y, z\}$ is a type-B triplet of N_2 if and only if $x|y|z$ is a fan triplet in N_2^B. Hence, *fcount*(T_1, N_2^B) is the number of fan triplets $x|y|z$ in T_1 that also are type-B triplets of N_2. Similarly, by Lemma 12 (2), *fcount*(T_1, N_2^C) is the number of fan triplets $x|y|z$ in T_1 that are also type-C triplets of N_2. □

Lemma 14. *fcount*$^*_{BC}(N_1, N_2)$=*fcount*(N_1^B, N_2^B)+*fcount*(N_1^B, N_2^C)+*fcount*(N_1^C, N_2^B)+*fcount*(N_1^C, N_2^C).

Proof. Following Lemma 12:

- *fcount*(N_1^B, N_2^B) counts the number of triplets that are of type-B in both N_1 and N_2.
- *fcount*(N_1^B, N_2^C) counts the number of triplets that are of type-B in N_1 and type-C in N_2.
- *fcount*(N_1^C, N_2^B) counts the number of triplets that are of type-C in N_1 and type-B in N_2.
- *fcount*(N_1^C, N_2^C) counts the number of triplets that are of type-C in both N_1 and N_2.

Since *fcount*$^*_{BC}(N_1, N_2)$ equals the number of triplets that are type B or C in N_1 and type B or C in N_2, the lemma follows. □

The algorithm in Figure 7 computes *fcount*(N_1, N_2) by combining Lemmas 10, 11, 13, and 14.

Lemma 15. The algorithm *fcount*(N_1, N_2) *in* Figure 7 *makes a total of* 25 *calls to fcount*(T_1, T_2)*, where* T_1 *and* T_2 *are phylogenetic trees.*

Proof. Note that *fcount*$_{BC}(T_1, N_2)$ makes two calls to *fcount*(T_1, T_2), and *fcount*$^*_{BC}(N_1, N_2)$ makes four calls to *fcount*(T_1, T_2). Also, *fcount*$_{BC}(N_1, N_2)$ makes three calls to *fcount*$_{BC}(T_1, N_2)$ and one call to *fcount*$^*_{BC}(N_1, N_2)$, so in total, *fcount*$_{BC}(N_1, N_2)$ uses $3 \cdot 2 + 1 \cdot 4 = 10$ calls to *fcount*(T_1, T_2).

Moreover, *fcount*(N_1, T_2) makes three calls to *fcount*(T_1, T_2) and one call to *fcount*$_{BC}(T_2, N_1)$. In total, *fcount*(N_1, T_2) is obtained from $3 + 1 \cdot 2 = 5$ calls to *fcount*(T_1, T_2).

Finally, *fcount*(N_1, N_2) makes three calls to *fcount*(N_1, T_2) and one call to *fcount*$_{BC}(N_1, N_2)$. In total, *fcount*(N_1, N_2) makes $3 \cdot 5 + 1 \cdot 10 = 25$ calls to *fcount*(T_1, T_2). □

Since *fcount*(T_1, T_2) can be computed in $O(n \log n)$ time for any two trees T_1, T_2 by Theorem 1, *fcount*(T_1, T_2) is called a constant number of times according to Lemma 15, and constructing each of the constant number of trees used as arguments to *fcount*(T_1, T_2) takes $O(n)$ time, the algorithm in Figure 7 runs in $O(n \log n)$ time.

4. IMPLEMENTATION AND EXPERIMENTS

We have implemented our new algorithm and conducted a series of experiments to evaluate its running time. The details of the implementation and the experimental results are presented below.

4.1. Implementation

The implementation is available at (https://github.com/Mesh89/Galled-CPDT-dist), along with instructions for how to run it. It uses CPDT-dist (Jansson and Rajaby, 2017) as the subroutine for computing the rooted triplet distance between two input phylogenetic trees. The implementation consists of three parts:

(i) A C++ executable that transforms a galled network N into the set of phylogenetic trees defined in Section 3, namely $N^{\searrow}$, $N^{\swarrow}$, $N^{\downarrow}$, N^{L}, N^{LL}, N^{R}, N^{RR}, N^{B}, and N^{C};

(ii) CPDT-dist from Jansson and Rajaby (2017) modified so that, given a pair T_1, T_2 of phylogenetic trees with identical leaf label sets, it outputs $fcount(T_1, T_2)$ and $rcount(T_1, T_2)$;

(iii) A Bash script that first uses (i) to construct two sets of trees from the given N_1 and N_2, then calls (ii) for pairs of trees according to the description of the algorithm in Section 3 and combines the results, and finally outputs the rooted triplet distance between N_1 and N_2.

The implementation is straightforward, but a few details are worth mentioning. First, the accepted input format for N_1 and N_2 is the Extended Newick Format, described in Cardona et al. (2008). Second, as described in Section 2, the rooted triplet distance between N_1 and N_2 is given by the formula $count(N_1, N_1) + count(N_2, N_2) - 2 \cdot count(N_1, N_2)$, where $count(N_i, N_i)$ is the total number of rooted triplets (i.e., fan triplets and resolved triplets) in N_i for $i \in \{1, 2\}$. Instead of computing $count(N_i, N_i)$ by running the algorithm, it is faster in practice to compute it by taking $\binom{n}{3}$ plus the number of ambiguous triplets in N_i (which can be counted directly in linear time). Third, the theoretical time complexity of the implemented algorithm is $O(n \log^3 n)$ rather than $O(n \log n)$. The reason is that the time complexity of the subroutine CPDT-dist is $O(n \log^3 n)$ and not $O(n \log n)$; however, it was shown experimentally in Jansson and Rajaby (2017) that CPDT-dist is faster for $n \leq 4,000,000$ than the software package tqDist (Sand et al., 2014) that implements the algorithm from Brodal et al. (2013).

4.2. Experimental setup

The experiments were performed on a server running Ubuntu 16.04 with four Intel(R) Xeon(R) CPU X5680 processors clocked at 3.33 GHz and 135 GB of RAM. As a C++ compiler, g++ version 5.4 was used. Running times were measured using the command "time."

4.3. Generating the input

To generate random galled trees for the input, we used a two-step procedure that first generates a random nonbinary phylogenetic tree and then modifies it into a galled tree.

The first step is the same as the one described in Jansson and Rajaby (2017), that is, first generate a binary phylogenetic tree in the uniform model (McKenzie and Steel, 2000) and then for each nonroot, internal node, contract it (thus making its children become children of its parent instead) with probability 0.2, to obtain a nonbinary phylogenetic tree.

The second step takes a phylogenetic tree with n leaves and tries to create $\lfloor \log n \rfloor$ galls in it. To generate a gall, the procedure does the following:

1. Choose an internal node h uniformly at random as the reticulation vertex of the new gall.
2. Choose an ancestor of h, s, to be the split vertex.
3. Ensure that the path $s \rightarrow h$ does not touch any existing gall.
4. Insert a new edge from s to h.
5. Move $q\%$ of the side trees from the original path to the new one, where q is chosen at random between 0 and 100 and is different for each gall.

To choose s, the procedure first draws a number p between 0.0 and 0.4; then, starting from h, it moves up the tree, stopping with probability p at each step and lets s be the final node. The procedure fails if the path

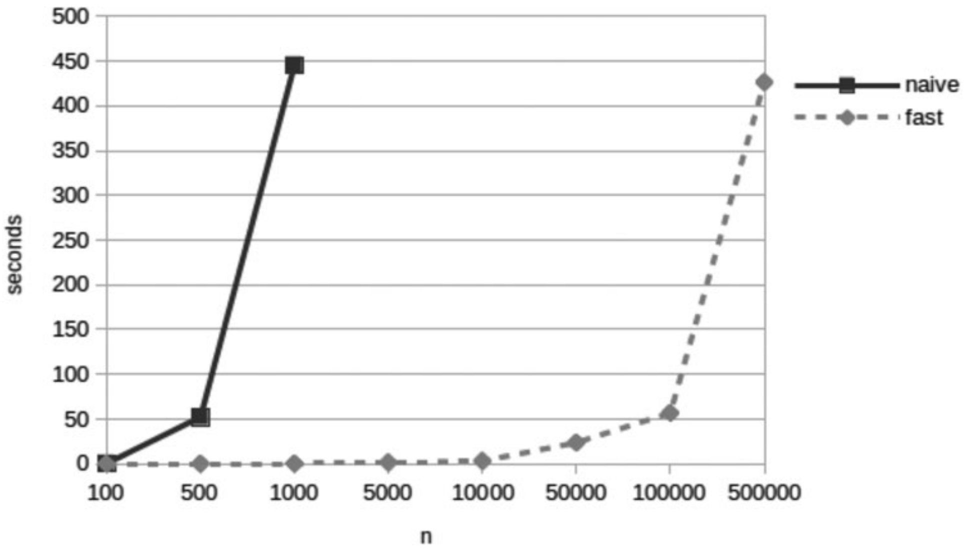

FIG. 8. Running times of the naive algorithm and our new algorithm for increasing values of n.

between and s and h touches a gall. In this case, it tries again by picking another pair of s and h; however, if it fails too many times, the procedure stops and outputs a galled tree with fewer than $\lfloor \log n \rfloor$ galls.

4.4. *Naive algorithm*

We also implemented a naive algorithm for the rooted triplet distance between two galled trees in order to verify the correctness of our implementation and to assess its achieved speed-up. The naive algorithm explicitly computes the two sets of $O(n^3)$ rooted triplets consistent with each of the networks, sorts them, and computes their intersection, using a total of $O(n^3 \log n)$ time. (By using radix sort in the sorting step, the time complexity would become $O(n^3)$, but the algorithm would become slower in practice.)

The $O(n^{2.687})$-time algorithm from Jansson and Lingas (2014) was not included in the experiments. We expect that for small values of n, any implementation of it would be much slower than the naive algorithm due to the huge constants involved in the state-of-the-art algorithms for matrix multiplication that it depends on. On the other hand, for large values of n, our new method will be a lot faster since its time complexity is asymptotically much better than $O(n^{2.687})$.

4.5. *Experimental results*

Figure 8 shows the running times of the naive algorithm and the new algorithm. The benefits of using the latter are evident: the new algorithm is able to compare two galled trees with 500,000 leaves in less time than the naive algorithm can compare two galled trees with 1000 leaves. More importantly, the implementation of the fast algorithm is currently the only program available for measuring the similarity between two large galled trees.

As for the memory usage, only one instance of CPDT-dist is run at a time. The memory needed to compare two phylogenetic trees with 500,000 leaves is around 1 GB; an extensive analysis of the memory used by CPDT-dist for various input sizes can be found in Jansson and Rajaby (2017).

5. CONCLUDING REMARKS

The presented algorithm requires a subroutine for computing the rooted triplet distance between two phylogenetic trees. If a faster algorithm for the case of trees than the one referred to in Theorem 1 is discovered (e.g., running in $O(n \log \log n)$ time), this would immediately imply a faster algorithm for the case of galled trees as well.

Note that even though the number of calls to the subroutine is constant, the constant is quite large ($49+25=74$). Is it possible to reduce this number? Doing so might improve the practical running time. The running time could also potentially be reduced by parallelizing the algorithm to handle many calls to the subroutine at the same time. In this case, memory will become an issue for large inputs, and the interested reader is referred to the experimental results reported in Jansson and Rajaby (2017) showing how much memory is consumed by the CPDT-dist subroutine for inputs with up to four million leaves.

An open problem is to determine whether the techniques used here can be extended to compute the rooted triplet distance between more general phylogenetic networks than galled trees. For example, even for the slight generalization of a galled tree in which cycles are allowed to share vertices and only required to be *edge-disjoint* (as opposed to *vertex-disjoint*), the current algorithm fails. The problem lies in the construction of N^B and N^C (Section 3.2); intuitively, they are constructed so that three leaves induce a fan in N^B (resp. N^C) if and only if they induce a type-B (resp. type-C) triplet in the original galled tree N. However, in the current construction, when different galls share the same root, leaves in different galls in N may induce fans in N^B and N^C. Finally, can the techniques be adapted to compute the *quartet distance* between two *unrooted phylogenetic networks* efficiently, for example, by making use of the known $O(dn \log n)$-time algorithm from Brodal et al. (2013) for the quartet distance between two unrooted phylogenetic trees with n leaves each and maximum degree d?

ACKNOWLEDGMENTS

This research work is partially supported by MOE Tier2 research grant R-252-000-664-112.

AUTHOR DISCLOSURE STATEMENT

The authors declare there are no competing financial interests.

REFERENCES

Bansal, M.S., Dong, J., and Fernández-Baca, D. 2011. Comparing and aggregating partially resolved trees. *Theor. Comput. Sci.* 412, 6634–6652.

Brodal, G.S., Fagerberg, R., Mailund, T., et al. 2013. Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree, 1814–1832. Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013). SIAM. New Orleans, LA, USA.

Cardona, G., Llabrés, M., Rosselló, F., et al. 2011. Comparison of galled trees. *IEEE ACM Trans. Comput. Biol. Bioinform.* 8, 410–427.

Cardona, G., Rosselló, F., and Valiente, G. 2008. Extended Newick: It is time for a standard representation of phylogenetic networks. *BMC Bioinform.* 9, 532.

Critchlow, D.E., Pearl, D.K., and Qian, C. 1996. The triples distance for rooted bifurcating phylogenetic trees. *Syst. Biol.* 45, 323–334.

Dobson, A.J. 1975. Comparing the shapes of trees, 95–100. Proceedings of Combinatorial Mathematics III, Lecture Notes in Mathematics, volume 452. Springer-Verlag. The University of Queensland, Australia.

Gambette, P., and Huber, K.T. 2012. On encodings of phylogenetic networks of bounded level. *J. Math. Biol.* 65, 157–180.

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

Huson, D.H., Rupp, R., and Scornavacca, C. 2010. *Phylogenetic Networks: Concepts, Algorithms and Applications*. Cambridge University Press, Cambridge, UK.

Jansson, J., and Lingas, A. 2014. Computing the rooted triplet distance between galled trees by counting triangles. *J. Discrete Algorithms* 25, 66–78.

Jansson, J., and Rajaby, R. 2017. A more practical algorithm for the rooted triplet distance. *J. Comput. Biol.* 24, 106–126.

Kuhner, M.K., and Felsenstein, J. 1994. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Mol. Biol. Evol.* 11, 459–468.

McKenzie, A., and Steel, M. 2000. Distributions of cherries for two models of trees. *Math. Biosci.* 164, 81–92.

Morrison, D. 2011. *Introduction to Phylogenetic Networks*. RJR Productions.

Sand, A., Holt, M.K., Johansen, J., et al. 2014. tqDist: A library for computing the quartet and triplet distances between binary or general trees. *Bioinformatics* 30, 2079–2080.

Wang, L., Ma, B., and Li, M. 2000. Fixed topology alignment with recombination. *Discrete Appl. Math.* 104, 281–300.

Address correspondence to:
Prof. Wing-Kin Sung
School of Computing
National University of Singapore
13 Computing Drive
Singapore 117417

E-mail: ksung@comp.nus.edu.sg