An Efficient Algorithm for the Rooted Triplet Distance Between Galled Trees

Jesper Jansson[1,2], Ramesh Rajaby[3,4], and Wing-Kin Sung[3,5(✉)]

[1] Laboratory of Mathematical Bioinformatics, ICR,
Kyoto University, Gokasho, Uji, kyoto 611-0011, Japan
jj@kuicr.kyoto-u.ac.jp
[2] Department of Computing, The Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong, China
[3] School of Computing, National University of Singapore,
13 Computing Drive, Singapore 117417, Singapore
e0011356@u.nus.edu, ksung@comp.nus.edu.sg
[4] NUS Graduate School for Integrative Sciences and Engineering,
National University of Singapore, 28 Medical Drive, Singapore 117456, Singapore
[5] Genome Institute of Singapore, 60 Biopolis Street, Genome,
Singapore 138672, Singapore

Abstract. The previously 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. Here, 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.

Keywords: Phylogenetic network comparison · Galled tree · Rooted triplet · Algorithm · Computational complexity

1 Introduction

Measuring the similarity between phylogenetic trees is essential for evaluating the accuracy of methods for phylogenetic reconstruction [11]. The *rooted triplet distance* [5] 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. More involved algorithms have been developed [1,2,4,10], and the asymptotically fastest one [2] solves the problem in $O(n \log n)$ time.

D. Figueiredo et al. (Eds.): AlCoB 2017, LNBI 10252, pp. 115–126, 2017.
DOI: 10.1007/978-3-319-58163-7_8

Gambette and Huber [6] extended the rooted triplet distance from the phylogenetic tree setting to the *phylogenetic network* setting. In a *phylogenetic network* [8,12], internal nodes are allowed to have more than one parent. Phylogenetic networks enable scientists to represent more complex evolutionary relationships than phylogenetic trees, e.g., 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* [7,8,13]. 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 Fig. 9.22 in [8] for a biological example. For a summary of other distances for comparing two galled trees (the Robinson-Foulds distance, the tripartitions distance, the μ-distance, the split nodal distance, etc.), see [3].

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 [9]. 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 [9] will never be better than $O(n^{2.5})$. In this paper, 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 [2] 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 which 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 paper is organized as follows. Section 2 defines the problem formally, Sect. 3 presents the algorithm and its analysis, and Sect. 4 contains some concluding remarks.

2 Problem Definitions

We recall the following definitions from [9].

A (rooted) *phylogenetic tree* is an unordered, rooted tree in which every internal node has at least two children and all leaves are distinctly labeled. A (rooted) *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 greater than 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 isomorphic to a cycle, and the vertex of C that is an ancestor of all vertices on C is called the *root* of C. A phylogenetic network is called a *galled*

tree if all of its cycles are vertex-disjoint [7,8,13]. 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 $a|b|c$, while 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 possible topologies, 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 non-zero 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. 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.

Definition 1 *(Adapted from [6]). Let N_1, N_2 be two phylogenetic networks on the same leaf label set L. The* rooted triplet distance between N_1 a*nd 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 Sect. 3.2 in [9] for a discussion of the above definition.) 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 by Brodal *et al.* in [2]:

Theorem 2. [2] *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 [9]. Below, we show how to do it faster by using Theorem 2, which yields our main result:

Theorem 3. *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, while Sect. 3.2 is focused on $fcount(N_1, N_2)$. (Both subsections rely on Theorem 2.) In addition to the definitions provided in Sect. 2, the following notation and terminology will be needed.

Suppose that N is a galled tree. For each internal vertex in N, fix some arbitrarily left-to-right ordering of its children. Then $N^{\searrow}$ is the 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 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 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.

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. A rooted triplet in $t(N)$ with leaf label set $\{x, y, z\}$ is called *ambiguous* if N contains a gall Q such that:

1. x, y, z are in three different subtrees attached to Q or $r(Q)$;
2. exactly one leaf is in the subtree attached to h_Q; and
3. at least one leaf 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. 1 for an illustration):

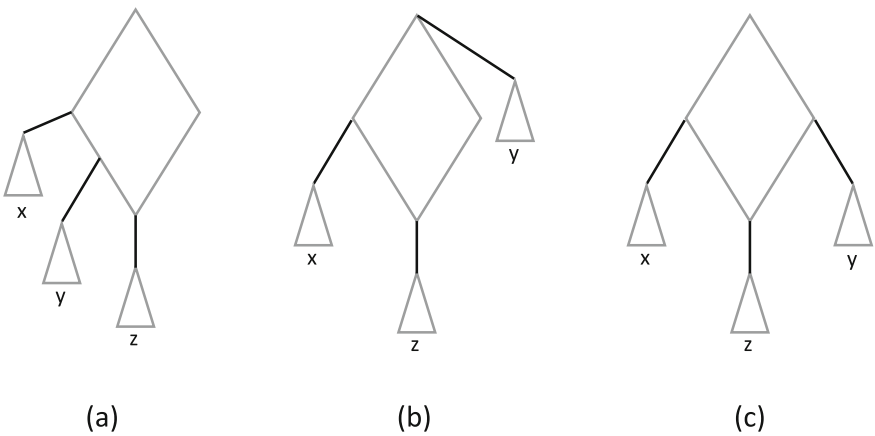

Fig. 1. (a), (b) and (c) illustrate the definitions of type-A, type-B, and type-C triplets, respectively.

- $\{x, y, z\}$ 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) while 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.
- $\{x, y, z\}$ 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.

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 5 below). 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 2) 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 4. *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 either of type-A, type-B, or type-C. Hence, we have the following cases.

- (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 it 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, $xy|z \in t(N_2^{\swarrow}), t(N_2^{\searrow}), t(N_2^{\downarrow})$. Also, $rcount_A(xy|z, N_2) = 0$. Hence, $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2) = 1$.

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 we have $rcount_A(xy|z, N_2) = 1$ by definition, $rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2) = 1$.

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

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 4, we have $rcount(xy|z, N_2) = rcount(xy|z, N_2^{\swarrow}) + rcount(xy|z, N_2^{\searrow}) - rcount(xy|z, N_2^{\downarrow}) + rcount_A(xy|z, N_2)$. □

Lemma 6. $rcount_A(N_1, N_2) = rcount_A(N_1^{\swarrow}, N_2) + rcount_A(N_1^{\searrow}, N_2) - rcount_A(N_1^{\downarrow}, N_2) + rcount^*_A(N_1, N_2)$.

Proof. For $xy|z \in t(N_1)$, by an argument identical to the one in the proof of Lemma 4, 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.

Let $W = \{xy|z : \{x, y, z\}$ 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 5,

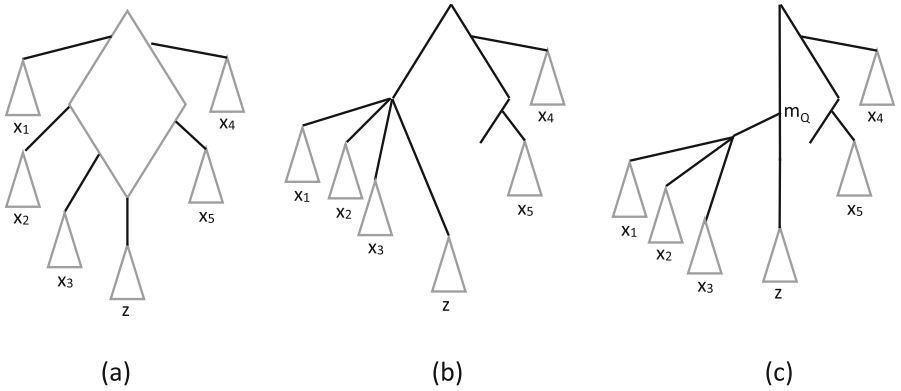

Fig. 2. (a) shows a galled tree N, (b) shows N^L, and (c) shows N^{LL}.

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 Fig. 2 for an example. N^R and N^{RR} are defined analogously.

Lemma 7. *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 $N^{\delta\delta}$, i.e., 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 $N^{\delta\delta}|\{x, y, z\} = xy|z$.

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 8. $rcount_A(T_1, N_2) = \sum_{\delta \in \{L,R\}} \big(rcount(T_1, N_2^{\delta\delta}) - rcount(T_1, N_2^\delta)\big)$.

Proof. By Lemma 7, 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 9. $rcount^*_A(N_1, N_2) = \sum_{\delta \in \{L,R\}} \big(rcount_A(N_1^{\delta\delta}, N_2) - rcount_A(N_1^{\delta}, N_2)\big)$.

Proof. (Similar to the proof of Lemma 8.) By Lemma 7, 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 Fig. 3 computes $rcount(N_1, N_2)$ using Lemmas 5, 6, 8, and 9.

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. 3. The algorithm for computing $rcount(N_1, N_2)$.

Lemma 10. *The algorithm* $rcount(N_1, N_2)$ *in Fig. 3 makes a total of* 37 *calls to* $rcount(T_1, T_2)$*, where* T_1 *and* T_2 *are phylogenetic trees.*

Proof. First, $rcount_A(T_1, N_2)$ is obtained by making 4 calls to $rcount(T_1, T_2)$, and $rcount^*_A(N_1, N_2)$ by 4 calls to $rcount_A(T_1, T_2)$. Next, $rcount_A(N_1, N_2)$ is obtained by 3 calls to $rcount_A(T_1, N_2)$ and 1 call to $rcount^*_A(N_1, N_2)$. In total, $rcount_A(N_1, N_2)$ uses $3 \cdot 4 + 4 = 16$ calls to $rcount(T_1, T_2)$.

Similarly, $rcount(N_1, T_2)$ is obtained by 3 calls to $rcount(T_1, T_2)$ and 1 call to $rcount_A(T_2, N_1)$. In total, $rcount(N_1, T_2)$ can be computed by $3 \cdot 1 + 4 = 7$ calls to $rcount(T_1, T_2)$.

Finally, $rcount(N_1, N_2)$ is obtained by 3 calls to $rcount(N_1, T_2)$ and 1 call to $rcount_A(N_1, N_2)$. In total, $rcount(N_1, N_2)$ uses $3 \cdot 7 + 16 = 37$ calls to $rcount(T_1, T_2)$. □

By Theorem 2, $rcount(T_1, T_2)$ can be computed in $O(n \log n)$ time for any two trees T_1, T_2. Lemma 10 shows that the algorithm in Fig. 3 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. (Some proofs have been omitted from the conference version of the paper.)

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 Sect. 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 11. *Let $x|y|z$ be a fan triplet. Then* $fcount(x|y|z, N_2) = fcount(x|y|z, N_2^{\swarrow}) + fcount(x|y|z, N_2^{\searrow}) - fcount(x|y|z, N_2^{\downarrow}) + fcount_{BC}(x|y|z, N_2) = 0$.

Lemma 12. $fcount(N_1, N_2) = fcount(N_1, N_2^{\swarrow}) + fcount(N_1, N_2^{\searrow}) - fcount(N_1, N_2^{\downarrow}) + fcount_{BC}(N_1, N_2)$.

Lemma 13. $fcount_{BC}(N_1, N_2) = fcount_{BC}(N_1^{\swarrow}, N_2) + fcount_{BC}(N_1^{\searrow}, N_2) - fcount_{BC}(N_1, N_2^{\downarrow}) + fcount^*_{BC}(N_1, N_2)$.

The rest of this subsection considers how to compute $fcount_{BC}(T_1, N_2)$ and $fcount^*_{BC}(N_1, N_2)$ efficiently. A *caterpillar* is a binary phylogenetic tree in which every internal node has at least one leaf child. Given a galled tree N, we define N^B as the tree formed by performing the following three steps on a copy of N:

- Replace every degree-k vertex which is not a split vertex in N by a length-k caterpillar.
- For every gall Q, if the split vertex $r(Q)$ is of degree $k > 2$, for all $k-2$ children of $r(Q)$ which are not on the gall, replace them by a length-$(k-2)$ caterpillar and attach it $r(Q)$. Furthermore, creating a new vertex u and attach the roots of Q_L and Q_R to u and attach u to $r(Q)$.
- Remove the reticulation vertex h_Q's two parent edges and attach h_Q and its subtrees to $r(Q)$.

Also, we define N^C as a tree obtained by performing the following three steps on a copy of N:

- Replace every degree-k vertex which is not a split vertex in N by a length-k caterpillar.
- For every gall Q, if the split vertex $r(Q)$ is of degree $k > 2$, for all $k-2$ children of $r(Q)$ which are not on the gall, replace them by a length-$(k-2)$ caterpillar and attach it to a new vertex between $r(Q)$ and its parent.
- Remove the reticulation vertex h_Q's two parent edges and attach h_Q and its subtrees to $r(Q)$.

Figure 4 gives an example illustrating how to construct N^B and N^C from N. The next lemma states how N, N^B, and N^C are related.

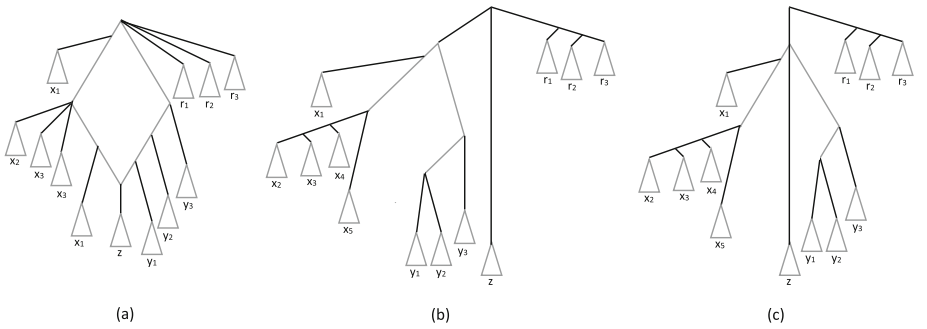

Fig. 4. (a) is an example of a galled tree N, (b) is N^B, and (c) is N^C.

Lemma 14. *(1) $\{x, y, z\}$ is a type-B triplet in N if and only if $x|y|z$ is a fan triplet in N^B; (2) $\{x, y, z\}$ is a type-C triplet in 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 2 or 3 children. Any vertex in N^B and N^C with 3 children corresponds to a split vertex of a gall in N.

For (1): ($\rightarrow$) If $\{x, y, z\}$ is a type-B triplet in 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, in N^B, 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 it corresponds to a gall Q. This implies, 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 in N.

For (2): ($\rightarrow$) If $\{x, y, z\}$ is a type-C triplet in 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, in N^C, 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 it corresponds to a gall Q. This implies, 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 in N. □

We have the following two lemmas.

Lemma 15. $fcount_{BC}(T_1, N_2) = fcount(T_1, N_2^B) + fcount(T_1, N_2^C)$.

Lemma 16. $fcount^*_{BC}(N_1, N_2) = fcount(N_1^B, N_2^B) + fcount(N_1^C, N_2^C)$.

The algorithm in Fig. 5 computes $fcount(N_1, N_2)$ by combining Lemmas 12, 13, 15, and 16.

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

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

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 $fcount(N_1^B, N_2^B) + fcount(N_1^C, N_2^C)$;

Fig. 5. The algorithm for computing $fcount(N_1, N_2)$.

Lemma 17. *The algorithm* $fcount(N_1, N_2)$ *in Fig. 5 makes a total of* 23 *calls to* $fcount(T_1, T_2)$*, where* T_1 *and* T_2 *are phylogenetic trees.*

Proof. Note that $fcount_{BC}(T_1, N_2)$ is computed by 2 calls to $fcount(T_1, T_2)$, and $fcount^*_{BC}(N_1, N_2)$ by 2 calls to $fcount_A(T_1, T_2)$. Moreover, $fcount_{BC}(N_1, N_2)$ makes 3 calls to $fcount_{BC}(T_1, N_2)$ and 1 call to $fcount^*_{BC}(N_1, N_2)$. In total, $fcount_{BC}(N_1, N_2)$ uses $3 \cdot 2 + 2 = 8$ calls to $fcount(T_1, T_2)$.

In the same way, $fcount(N_1, T_2)$ makes 3 calls to $fcount(T_1, T_2)$ and 1 call to $fcount_{BC}(T_2, N_1)$. In total, $fcount(N_1, T_2)$ is obtained from $3 \cdot 1 + 2 = 5$ calls to $fcount(T_1, T_2)$.

Finally, $fcount(N_1, N_2)$ is obtained from 3 calls to $fcount(N_1, T_2)$ and 1 call to $rcount_{BC}(N_1, N_2)$. In total, $fcount(N_1, N_2)$ makes $3 \cdot 5 + 8 = 23$ 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 2, $fcount(T_1, T_2)$ is called a constant number of times according to Lemma 17, 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 Fig. 5 runs in $O(n \log n)$ time.

4 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 2 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. As an alternative, the algorithm in [10] was shown experimentally to be faster for reasonably sized inputs; hence its usage may be preferred in a practical setting.

Possible future work is to implement the new algorithm and evaluate its performance in practice. Although the number of calls to the subroutine for computing the rooted triplet distance between trees is constant, the constant is quite large ($37 + 23 = 60$). Is it possible to reduce this number? An implementation of the new algorithm would benefit considerably by doing so.

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.

Acknowledgments. J.J. was partially funded by The Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. Theor. Comput. Sci. **412**(48), 6634–6652 (2011)
2. Brodal, G.S., Fagerberg, R., Mailund, T., Pedersen, C.N.S., Sand, A.: Efficient algorithms for computing the triplet and quartet distance between trees of arbitrary degree. In: Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013), pp. 1814–1832. SIAM (2013)
3. Cardona, G., Llabrés, M., Rosselló, R., Valiente, G.: Comparison of galled trees. IEEE/ACM Trans. Comput. Biol. Bioinf. **8**(2), 410–427 (2011)
4. Critchlow, D.E., Pearl, D.K., Qian, C.: The triples distance for rooted bifurcating phylogenetic trees. Syst. Biol. **45**(3), 323–334 (1996)
5. Dobson, A.J.: Comparing the shapes of trees. In: Street, A.P., Wallis, W.D. (eds.) Combinatorial Mathematics III. LNM, vol. 452, pp. 95–100. Springer, Heidelberg (1975). doi:10.1007/BFb0069548
6. Gambette, P., Huber, K.T.: On encodings of phylogenetic networks of bounded level. J. Math. Biol. **65**(1), 157–180 (2012)
7. Gusfield, D., Eddhu, S., Langley, C.: Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. J. Bioinf. Comput. Biol. **2**(1), 173–213 (2004)
8. Huson, D.H., Rupp, R., Scornavacca, C.: Phylogenetic Networks: Concepts Algorithms and Applications. Cambridge University Press, Cambridge (2010)
9. Jansson, J., Lingas, A.: Computing the rooted triplet distance between galled trees by counting triangles. J. Discrete Algorithms **25**, 66–78 (2014)
10. Jansson, J., Rajaby, R.: A more practical algorithm for the rooted triplet distance. J. Comput. Biol. **24**(2), 106–126 (2017)
11. Kuhner, M.K., Felsenstein, J.: A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. Mol. Biol. Evol. **11**(3), 459–468 (1994)
12. Morrison, D.: Introduction to Phylogenetic Networks. RJR Productions (2011)
13. Wang, L., Ma, B., Li, M.: Fixed topology alignment with recombination. Discrete Appl. Math. **104**(1–3), 281–300 (2000)