Fast Algorithms for the Rooted Triplet Distance Between Caterpillars

Jesper Jansson[1,2] and Wing Lik Lee[1(✉)]

[1] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong
jesper.jansson@polyu.edu.hk, wing-lik.lee@connect.polyu.hk
[2] Graduate School of Informatics, Kyoto University, Kyoto, Japan

Abstract. The *rooted triplet distance* measures the structural dissimilarity between two rooted *phylogenetic trees* (unordered trees with distinct leaf labels and no outdegree-1 nodes) having the same leaf label set. It is defined as the number of 3-subsets of the leaf label set that induce two different subtrees in the two trees. The fastest currently known algorithm for computing the rooted triplet distance was designed by Brodal *et al.* (SODA 2013). It runs in $O(n \log n)$ time, where n is the number of leaf labels in the input trees, and a long-standing open question is whether this is optimal or not. In this paper, we present two new $o(n \log n)$-time algorithms for the special case of *caterpillars* (rooted phylogenetic trees in which every node has at most one non-leaf child), thus breaking the $O(n \log n)$-time bound for a fundamental class of trees. Our first algorithm makes use of a dynamic rank-select data structure by Raman *et al.* (WADS 2001) and runs in $O(n \log n / \log \log n)$ time. Our second algorithm relies on an efficient orthogonal range counting algorithm invented by Chan and Pătraşcu (SODA 2010) and runs in $O(n\sqrt{\log n})$ time.

Keywords: Phylogenetic tree · Caterpillar · Rooted triplet distance · Dynamic rank-select data structure · Orthogonal range counting

1 Introduction

Phylogenetics is the study of evolutionary relationships between different species or groups. To describe a set of inferred evolutionary relationships, scientists commonly use a *phylogenetic tree*, which is a leaf-labeled tree where each leaf represents one entity such as a biological species. It is a diagrammatic representation of evolutionary history such that the more closely related two species are, the closer they are to each other in the tree.

Similarity measures between phylogenetic trees are frequently used in phylogenetics. For instance, analysis methods such as Bayesian inference [9] produce a set of phylogenetic trees that are the most likely to represent true evolutionary history. A consensus method is then applied to condense them into a single tree which is close to the original set based on some well-defined measure. Another

E. Bampis and A. Pagourtzis (Eds.): FCT 2021, LNCS 12867, pp. 327–340, 2021.
https://doi.org/10.1007/978-3-030-86593-1_23

use case of similarity measures is for evaluating new phylogenetic tree or network reconstruction methods. As explained in [15], in one evaluation method, some biomolecular sequences are evolved according to a model of evolution represented by a base tree and then the new reconstruction method generates a tree from these evolved sequences. The similarity between the base tree and the generated tree then gives an indicator of the quality of the reconstruction method.

One of the earliest similarity measures proposed that is still popular today is the Robinson–Foulds metric [19], which counts the number of *clusters* (leaf label sets of subtrees rooted at the nodes in the trees) that can only be found in one tree and not the other. It can be computed in linear time [7], but has the disadvantage that a small change in the input trees can lead to a huge change in the value. As an example, suppose the input consists of two identical caterpillars. If we were to move a single leaf from the bottom of one tree to the top, the Robinson-Foulds distance would go from zero to near its maximum possible value even though the two trees still share a lot of branching structure.

Another previously well-studied measure is the is the NNI (nearest neighbor interchange) distance [18], which counts the number of branch-swapping transformations needed to turn one input tree into the other. However, it has remained of mostly theoretical interest as it has been proved that computing the NNI distance is NP-hard [6].

The Kendall-Colijn metric [12] is computed by finding the lowest common ancestors of each leaf pair in the two input trees, and comparing the difference in distance to the root node. This measure can be computed in $O(n^2)$ time, but has the disadvantage that leaves close to the root will influence the score more than leaves further away from the root.

Finally, the rooted triplet distance [8] (defined formally in Sect. 1.1 below) counts the number of 3-subsets of the leaf label set that induce two different subtrees in the two trees. Its main advantages are that it is robust to small changes in the input [1] and that it can be computed in polynomial time (see Sect. 1.2). The rooted triplet distance has since its inception found a wide range of practical applications in phylogenetics, as well as in other fields of biology. See [13,14,16] for examples.

In this paper, we focus on fast algorithms for the rooted triplet distance.

1.1 Problem Definitions

A *rooted phylogenetic tree* (from here on called a *tree* for short) is a rooted, unordered tree where each internal node has at least two children, and the leaves are distinctly labeled by a set of leaf labels. A *caterpillar* is a tree where every node has at most one non-leaf child. To simplify the presentation below, we will identify every leaf in a tree with its unique label.

A *rooted triplet* is a tree with exactly three leaves. A *resolved triplet* is a rooted triplet where two of its leaves, say x, y, have a common parent that is not a parent of the remaining leaf z. We write $xy|z$ to denote such a resolved triplet. In contrast, an *unresolved triplet* is a rooted triplet where all three leaves have the same parent. An unresolved triplet with leaves x, y, z will be denoted by $x|y|z$. Unresolved triplets are also referred to as *fan triplets* in the literature.

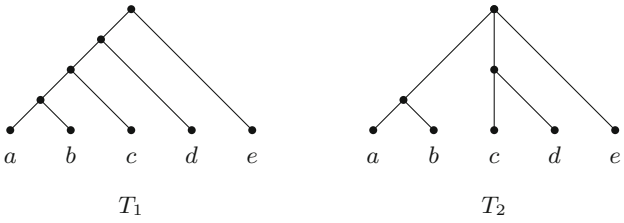

Fig. 1. An example: T_1 and T_2 are two rooted phylogenetic trees with leaf label set $\{a, b, c, d, e\}$, and T_1 is a caterpillar. Observe that, e.g., the resolved triplet $ac|d$ is consistent with T_1 but not T_2, while the unresolved triplet $a|c|e$ is consistent with T_2 but not T_1. Since $rt(T_1) = \{ab|c, ab|d, ab|e, ac|d, ac|e, ad|e, bc|d, bc|e, bd|e, cd|e\}$ and $rt(T_2) = \{ab|c, ab|d, ab|e, cd|a, a|c|e, a|d|e, cd|b, b|c|e, b|d|e, cd|e\}$, we have $d_{rt}(T_1, T_2) = 6$.

A resolved triplet $xy|z$ is said to be *consistent* with a tree T if the lowest common ancestor (LCA) of x, y in T is a proper descendant of the LCA of all three leaves in T. An unresolved triplet $x|y|z$ is said to be *consistent* with T if the LCA of all three leaves are the same in T as the LCA of any two leaves. Define $rt(T)$ to be the set of all rooted triplets that are consistent with T. If T_1, T_2 share the same set of leaf labels, define the *rooted triplet distance* between T_1 and T_2, written as $d_{rt}(T_1, T_2)$, as $\frac{1}{2}\,|rt(T_1) \triangle rt(T_2)|$, where $\triangle$ denotes the symmetric difference. See Fig. 1 for an example.

The rooted triplet distance problem can be stated as follows:

The Rooted Triplet Distance Problem
Input: Two rooted phylogenetic trees T_1, T_2 with the same leaf label set Λ.
Output: The rooted triplet distance $d_{rt}(T_1, T_2)$.

In the rest of the paper, we let n be the number of leaves in each input tree.

1.2 Previous Results

The rooted triplet distance was introduced in 1975 by Dobson [8]. A straightforward algorithm for computing it runs in $O(n^3)$ time. In 1996, Critchlow *et al.* [5] presented an $O(n^2)$-time algorithm for the special case of two binary trees which categorizes triplets based on their potential ancestor pairs. In 2011, Bansal *et al.* [1] gave an $O(n^2)$-time algorithm that computes the distance between two unrestricted trees by using postorder tree traversals. In 2013, Sand *et al.* [20] described an $O(n \log^2 n)$-time algorithm for binary trees using a data structure called the hierarchical decomposition tree (HDT). Brodal *et al.* [2] developed an HDT-based $O(n \log n)$-time algorithm that works for trees of arbitrary degrees. Jansson and Rajaby [11] later proposed an $O(n \log^3 n)$-time algorithm, modified from [2] by using a simpler data structure called the centroid path decomposition tree, that although slower in theory, runs faster in practice for values of n up to $4,000,000$. Subsequently, Brodal and Mampentzidis [3] designed an even more practical $O(n \log n)$-time algorithm that scales to external memory, using a

Table 1. Previous results on the rooted triplet distance problem

Year	Reference	Degree	Time complexity
1975	Dobson [8]	Arbitrary	$O(n^3)$
1996	Critchlow *et al.* [5]	Binary	$O(n^2)$
2011	Bansal *et al.* [1]	Arbitrary	$O(n^2)$
2013	Sand *et al.* [20]	Binary	$O(n \log^2 n)$
2013	Brodal *et al.* [2]	Arbitrary	$O(n \log n)$
2017	Jansson and Rajaby [11]	Arbitrary	$O(n \log^3 n)$
2017	Brodal and Mampentzidis [3]	Arbitrary	$O(n \log n)$
2019	Jansson *et al.* [10]	Arbitrary	$O(qn)$

further modified centroid decomposition technique. Recently, an algorithm with time complexity $O(qn)$, where at least one of the input trees has at most q internal nodes, was given by Jansson *et al.* [10]. See Table 1 for a summary.

1.3 New Results and Organization of Paper

We present two new algorithms for the rooted triplet distance problem restricted to caterpillars. The first algorithm is simple to implement and performs well in practice, while the second algorithm is even faster in theory, having a lower time complexity. These two algorithms are the first to achieve sub-$O(n \log n)$ time complexity for any non-trivial special cases of the rooted triplet distance problem when the number of internal nodes is unrestricted.

In Sect. 2 we give definitions, preliminary results, and summaries of data structures that will be used in the following sections. The first algorithm, presented in Sect. 3, computes the distance in $O(n \log n / \log \log n)$ time by defining a series of steps to transform one input tree to the other, and counts the number of rooted triplets that change in each step using the rank-select data structure by Raman *et al.* [17]. The second algorithm, presented in Sect. 4, uses the orthogonal range counting algorithm by Chan and Pătraşcu [4], and computes the distance in $O(n\sqrt{\log n})$ time by mapping each leaf label onto a 2-D grid and then making $O(n)$ orthogonal range counting queries on the grid. Finally, Sect. 5 summarizes our new results and lists some open problems.

2 Preliminaries

First, we describe the tree transformation steps to be used in the first algorithm. They allow us to break down the transformation of one input tree into the other into a series of steps, and compute the rooted triplet distance by adding up the changes in triplets that occur in each step.

Given two input trees on label set Λ, call one of them the *start tree* T_{start}, and the other one the *goal tree* T_{goal}. Fixing T_{goal}, define good(T) to be the set of all 3-subsets of Λ that induce the same rooted triplet in T as in T_{goal}. Define bad(T) to be the set of all 3-subsets of Λ that induce different triplets in T and T_{goal}. Then, given trees T, T', define $\Phi(T, T') = |\text{bad}(T) \cap \text{good}(T')| - |\text{good}(T) \cap \text{bad}(T')|$.

We may view $\Phi(T, T')$ as the change in d_{rt} with respect to T_{goal} as we transform T into T'. Φ counts the number of triplets that are turned from bad to good, and subtracts the number of triplets turning from good to bad. Rewriting $d_{rt}(T, T_{goal}) = |\text{bad}(T)|$ and $d_{rt}(T', T_{goal}) = |\text{bad}(T')|$, we see that $d_{rt}(T, T_{goal}) = \Phi(T, T') + d_{rt}(T', T_{goal})$.

If we can find a sequence of trees and the Φ-values between any two adjacent trees, then we can compute the triplet distance between any two trees in the sequence by summing up these Φ-values. This gives us a method for computing $d_{rt}(T_{start}, T_{goal})$:

Lemma 1. *Let $T_1 = T_{start}, T_2, \ldots, T_{k-1}, T_k = T_{goal}$ be a sequence of trees. Then $d_{rt}(T_{start}, T_{goal}) = \sum_{i=1}^{k-1} \Phi(T_i, T_{i+1})$.*

Next, we demonstrate that we can ignore most leaves of a tree if the changes in a transformation step are contained in some small subsets:

Lemma 2. *Given trees T_a, T_b, $u \in T_a$, and $v \in T_b$, let T'_a, T'_b be subtrees of T_a, T_b rooted at u, v respectively. Obtain T''_a, T''_b by replacing T'_a, T'_b each by a single node. Then $\Phi(T_a, T_b) = \Phi(T'_a, T'_b)$ if T''_a and T''_b are isomorphic.*

The lemma holds because the condition implies that only those rooted triplets whose three leaves all lie inside T'_a and T'_b will affect $\Phi(T_a, T_b)$.

Our first algorithm makes use of a data structure based on the dynamic rank-select data structure by Raman *et al.* [17]. A query tree Q stores a multiset of integers in $[1..n]$, and supports the following operations:

- *insert*(a): insert a value a to Q.
- *query*(a): output the number of inserted values less than a.

The *insert*, *query* operations are respectively direct analogs to the *update*, *sum* operations in [17]. The next lemma summarizes its time complexity.

Lemma 3. *There is a data structure that supports the insert and query operations in $O(\log n / \log \log n)$ time.*

Finally, in our second algorithm, we will use an orthogonal range counting result given by Chan and Pătraşcu [4]. We restate Corollary 2.3 from [4] here:

Lemma 4. *Given n points and n axis-aligned rectangles on the grid, we can obtain the number of points inside each rectangle in $O(n\sqrt{\log n})$ total time.*

3 The First Algorithm

The main idea behind this algorithm is to treat the input caterpillars as lists of leaves, and transform one list into the other by performing insertion sort on the list while moving one group of leaves at a time. See Algorithm 1 for the pseudocode. We first describe the sequence of intermediate caterpillars used in the transformation from T_{start} to T_{goal}. We then show how $d_{rt}(T_{start}, T_{goal})$ can be computed by performing $O(n)$ insertions and queries to the rank-select query tree, which by Lemma 3 yields a total time complexity of $O(n \log n / \log \log n)$.

Algorithm 1: Rank-Select Method

Input: Caterpillars T_{start}, T_{goal} on label set Λ.
Output: $d_{rt}(T_{start}, T_{goal})$

```
1 Relabel T_start, T_goal;
2 Build query tree Q according to input size n;
3 Parse T_start to obtain leaf groups G_1, ..., G_m;
4 foreach G_i do
5 |   Perform insertions and queries to Q to get values |A_i|, |B_i|, ...;
6 |   Compute Φ(T_{i-1}, T_i) using values |A_i|, ...;
7 end
8 Compute and output ∑ Φ(T_{i-1}, T_i);
```

Define a mapping $\Lambda \to \mathbb{Z}$ such that the labels of each leaf in T_{goal} is mapped to the distance from the leave to the internal node that is the farthest from the root. Then, apply this mapping to both T_{goal} and T_{start}. See Fig. 2 for an example. Note that the leaves will no longer be distinctly labeled and that all leaves in T_{goal} having the same parent will receive the same label.

Define a *leaf group* of a tree T as a maximal multiset of identical leaf labels in which each element corresponds to a distinct leaf in T and the leaves corresponding to its elements all have the same parent in T. In Fig. 2, the multisets $\{5\}$, $\{2, 2\}$, $\{1, 1\}$ are leaf groups in T_{start}, while $\{2\}$ and $\{4, 4\}$ are not. Two leaf groups G_1, G_2 are said to be *connected* if their leaves are siblings in T.

Let U, V be multisets of Λ. Define $U \prec V$ if for all $u \in U, v \in V$, $u < v$ by their value. Write $U \simeq V$ if $u = v$, and $U \preceq V$ if $u \leq v$, for all $u \in U, v \in V$.

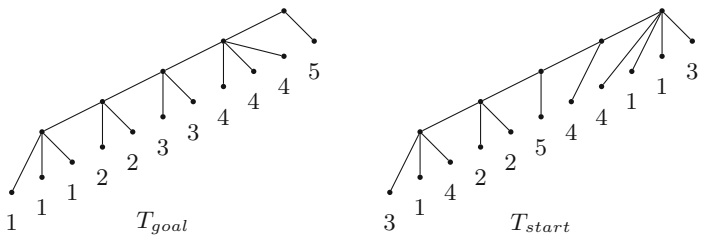

Fig. 2. In this example, after relabeling the leaves according to T_{goal}, the leaf groups in T_{start} are {3}, {1}, {4}, {2, 2}, {5}, {4}, {4}, {1, 1}, {3}.

3.1 Algorithm Description

To apply the insertion sort strategy, represent trees as lists of leaf groups. Given any T, define the *ordering* of T to be a list of leaf groups in T, subject to the following additional rules:

- The ordering of the leaf groups follows a post order traversal.
- A set of connected leaf groups appear in the list in ascending order, so that if $G_1, G_2, \ldots, G_k$ are connected groups listed in this order, then $G_1 \preceq G_2 \preceq \cdots \preceq G_k$.

In Fig. 2, the ordering of T_{goal} is $(\{1,1,1\},\{2,2\},\{3,3\},\{4,4,4\},\{5\})$, and the ordering of T_{start} is $(\{1\},\{3\},\{4\},\{2,2\},\{5\},\{4\},\{1,1\},\{3\},\{4\})$.

Suppose T_{start} has m leaf groups, so that its ordering is $(G_1, G_2, \ldots, G_m)$. We define a sequence of trees $T_1 = T_{start}, T_2, \ldots, T_m = T_{goal}$, where each T_i is the tree with ordering $(G_{\sigma_i(1)}, \ldots, G_{\sigma_i(i)}, G_{i+1}, \ldots, G_m)$, σ_i being a permutation of $\{1, \ldots, i\}$, and:

- $G_{\sigma_i(1)} \preceq G_{\sigma_i(2)} \preceq \cdots \preceq G_{\sigma_i(i)}$.
- Leaf groups $G_{\sigma_i(i)}$ and G_{i+1} are not connected.
- Leaf groups in $\{G_{i+1}, \ldots, G_m\}$ are connected if and only if they are also connected in T_{start}.
- Adjacent leaf groups $G_{\sigma_i(j)}, G_{\sigma_i(j+1)}$ are connected if and only if $G_{\sigma_i(j)} \simeq G_{\sigma_i(j+1)}$.

By Lemma 1, $d_{rt}(T_{start}, T_{goal}) = \sum_{i=1}^{m-1} \Phi(T_i, T_{i+1})$.

Consider leaf group G_i in T_i. G_i may be connected with some $G_{i+1}, \ldots, G_{i+j}$. By Lemma 2, to compute $\Phi(T_i, T_{i+1})$ it suffices to consider the subtree of T_i containing $G_1, \ldots, G_{i+j}$. Separate the leaves in this subtree minus G_i into four (possibly empty) subsets, A_i, B_i, C_i, D_i, so that $G_{i+1}, \ldots, G_{i+j} \subset D_i, B_i \simeq G_i$, and $A_i \prec G_i \prec C_i$.

To transform T_i into T_{i+1}, we may think of detaching G_i and attaching it to the parent of B_i, or, if $B_i = \varnothing$, attaching it to a new internal node at the appropriate position. Figure 3 illustrates this process.

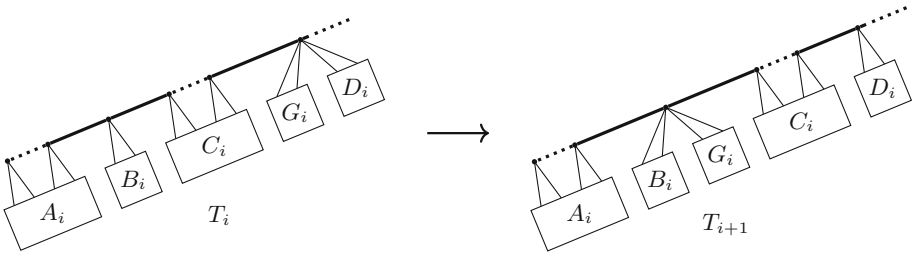

Fig. 3. Moving G_i in step i

Table 2. Listing for each type, the induced triplets in T_i, T_{i+1}, its effects on Φ, and its counts. Lowercase letters represent leaves in the corresponding subsets, so that $a_i \in A_i, b_i \in B_i, \ldots$, and repeated labels represent distinct leaves. Here, E_i, F_i are defined as sets of leaves $\{c_i, d_i\}$ satisfying the given constraints.

Type	T_i	T_{i+1}	Effect on Φ	Count
$\{g_i, a_i, a_i\}$	$a_i a_i \mid g_i$	$a_i a_i \mid g_i$	None	–
$\{g_i, a_i, b_i\}$	$a_i b_i \mid g_i$	$a_i \mid b_i \mid g_i$	Increased	$\lvert G_i\rvert \lvert A_i\rvert \lvert B_i\rvert$
$\{g_i, a_i, c_i\}$	$a_i c_i \mid g_i$	$a_i g_i \mid c_i$	Increased	$\lvert G_i\rvert \lvert A_i\rvert \lvert C_i\rvert$
$\{g_i, a_i, d_i\}$	$a_i \mid g_i \mid d_i$	$a_i g_i \mid d_i$	Increased	$\lvert G_i\rvert \lvert A_i\rvert \lvert D_i\rvert$
$\{g_i, b_i, b_i\}$	$b_i b_i \mid g_i$	$b_i \mid b_i \mid g_i$	Increased	$\lvert G_i\rvert \binom{\lvert B_i\rvert}{2}$
$\{g_i, b_i, c_i\}$	$b_i c_i \mid g_i$	$b_i g_i \mid c_i$	Increased	$\lvert G_i\rvert \lvert B_i\rvert \lvert C_i\rvert$
$\{g_i, b_i, d_i\}$	$b_i \mid g_i \mid d_i$	$b_i g_i \mid d_i$	Increased	$\lvert G_i\rvert \lvert B_i\rvert \lvert D_i\rvert$
$\{g_i, c_i, c_i\}$	$c_i c_i \mid g_i$	$g_i c_i \mid c_i$	Increased	$\lvert G_i\rvert \binom{\lvert C_i\rvert}{2}$
		$g_i \mid c_i \mid c_i$		
$\{g_i, c_i, d_i\}$	$c_i \mid g_i \mid d_i$	$g_i c_i \mid d_i$	Decreased ($c_i = d_i$)	$\lvert G_i\rvert \lvert E_i\rvert$
			None ($c_i > d_i$)	–
			Increased ($c_i < d_i$)	$\lvert G_i\rvert \lvert F_i\rvert$
$\{g_i, d_i, d_i\}$	$g_i \mid d_i \mid d_i$	$g_i \mid d_i \mid d_i$	None	–
$\{g_i, g_i, a_i\}$	$a_i \mid g_i \mid g_i$	$a_i \mid g_i \mid g_i$	None	–
$\{g_i, g_i, b_i\}$	$b_i \mid g_i \mid g_i$	$b_i \mid g_i \mid g_i$	None	–
$\{g_i, g_i, c_i\}$	$c_i \mid g_i \mid g_i$	$g_i g_i \mid c_i$	Increased	$\binom{\lvert G_i\rvert}{2} \lvert C_i\rvert$
$\{g_i, g_i, d_i\}$	$g_i \mid g_i \mid d_i$	$g_i g_i \mid d_i$	Increased	$\binom{\lvert G_i\rvert}{2} \lvert D_i\rvert$

3.2 Computing Φ

We now show how each Φ can be computed by making $O(1)$ queries to the query tree Q. Fill the query tree following the ordering of T_{start}, where for each leaf group G of size k and value a, we perform $insert(a)$ k times. By making $O(1)$ queries at different states of Q, we can get the number of leaves within a range of values in any continuous range of leaf groups.

By Lemma 2, we only need to consider triplets where each of its leaves are in one of the subsets A_i, B_i, C_i, G_i, D_i. Categorize these triplets based on where each of its leaves are located. We can immediately disregard most triplet types: any triplet types not containing at least one leaf from G_i are unchanged, so are any triplet types where all three of its leaves are in the same subset. For the remaining types, we list their effect on Φ and counts in Table 2.

The values $|A_i|, |B_i|, |C_i|$ can be found by making $O(1)$ queries to Q for the number of leaves in $G_1, \ldots, G_{i-1}$ that are less than, equal to, or greater than g_i, respectively. $|G_i|$ can be directly read from each leaf group.

The values $|D|, |E|, |F|$ can be found using a similar approach. Let $\{G_i, \ldots, G_{i+j}\}$ be a maximal set of connected leaf groups. $|D|$-values can be computed using formulas $|D_i| = \sum_{k=1}^{j} |G_{i+k}|$ and $|D_{i+k}| = |D_{i+k-1}| - |G_{i+k}|$. For the $|E|$-values, first make $O(j)$ queries to Q to obtain the number of all possible $\{c, d\}$ leaf pairs, where $c = d, c \in \{G_1, \ldots, G_{i-1}\}$, and $d \in \{G_i, \ldots, G_{i+j}\}$. Subtract $|B_i||G_i|$ from this total to get $|E_i|$, and then subtract $|B_{i+j+1}||G_{i+j+1}|$ from each $|E_{i+j}|$ to obtain $|E_{i+j+1}|$. Modifying the query ranges, the same method can be used to find the $|F|$-values. The precomputing steps require a number of queries proportional to the number of leaf groups in the maximal set, therefore adds $O(1)$ amortized number of queries per leaf group.

For the time complexity of the algorithm, preprocessing involves building a label map, applying it to T_{start}, and retrieving the list of leaf groups. Each one of these tasks can be done in $O(n)$ time. Then, $O(n)$ insertion and query operations are needed to compute all Φ-values. By Lemma 3, this proves:

Theorem 1. *Given two caterpillars on the same leaf label set of size n, Algorithm 1 computes the rooted triplet distance between them in $O(n \log n / \log \log n)$ time.*

4 The Second Algorithm

Our second method maps each leaf label onto a 2-D integer grid of size $n \times n$, according to their positions in the two input caterpillars. This is done so that by making $O(n)$ queries, we retrieve the total number of *good triplets*, triplets that are consistent with both input trees, and thus the rooted triplet distance. See Algorithm 2 for the pseudocode.

Algorithm 2: Orthogonal Range Counting Method

Input: Caterpillars T_1, T_2 on leaf label set Λ.
Output: $d_{rt}(T_1, T_2)$.

```
foreach ℓ ∈ Λ do
    Compute and store the point f(ℓ);
end
foreach point p in Im f do
    Perform range counting queries to get the values A, B, C, D;
    Compute the number of good triplets rooted at p;
end
Compute and output d_rt(T1, T2) using Lemma 5;
```

4.1 Mapping Leaves to the Grid

First, we define the mapping of leaves into the grid. Index the internal nodes of each input caterpillar T_1, T_2 in ascending order from the bottom to top, so that the lowest internal node is labeled 1, its parent is labeled 2, etc.

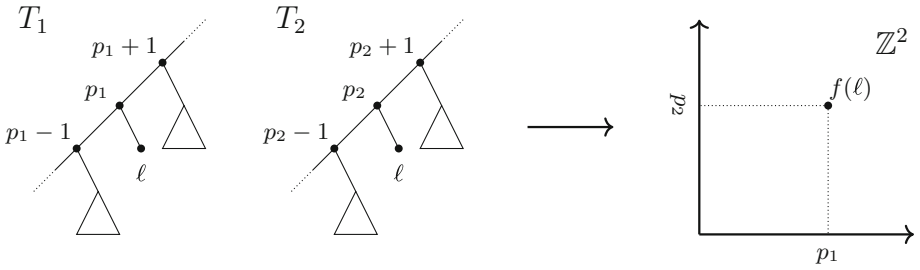

Fig. 4. f maps each leaf to a point in $\mathbb{Z}^2$ according to its positions in T_1 and T_2.

Suppose T_1 has h_1 internal nodes, and T_2 has h_2 internal nodes. Let $S = [1, h_1] \times [1, h_2]$ be a subset of $\mathbb{Z}^2$, and define a mapping $f : \Lambda \to S$, which maps each leaf label to a point in the grid according to the indices of its parent nodes in the two input trees. If for leaf ℓ, its parents in T_1, T_2 are indexed p_1, p_2 respectively, then ℓ is mapped to the point (p_1, p_2). See Fig. 4 for an illustration.

Next, define mapping f' from the set of good triplets to S as follows. For each good triplet τ, if the root node of the induced subtree in T_1, T_2 is indexed p_1, p_2 respectively, then $f'(\tau) = (p_1, p_2)$. Summing up the number of good triplets mapped to each point in $\operatorname{Im} f'$, we get the total number of good triplets.

4.2 Counting Good Triplets

Consider any point $\mathbf{p} = (p_1, p_2)$. Write $A_\mathbf{p}, B_\mathbf{p}, C_\mathbf{p}, D_\mathbf{p}$ for the number of leaves mapped to the regions $\{p_1\} \times \{p_2\}, \{p_1\} \times [1, p_2 - 1], [1, p_1 - 1] \times \{p_2\}, [1, p_1 - 1] \times [1, p_2 - 1]$ respectively. See Fig. 5. These values will be used to count the number of good triplets mapped to $\mathbf{p}$. For clarity of presentation, the $\mathbf{p}$-subscripts will be omitted below.

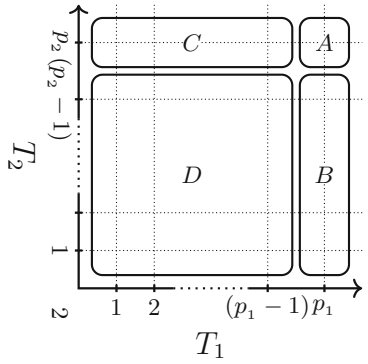

Fig. 5. Divide $[1, p_1] \times [1, p_2]$ into four regions, and define A, B, C, D as the number of leaves mapped to each region.

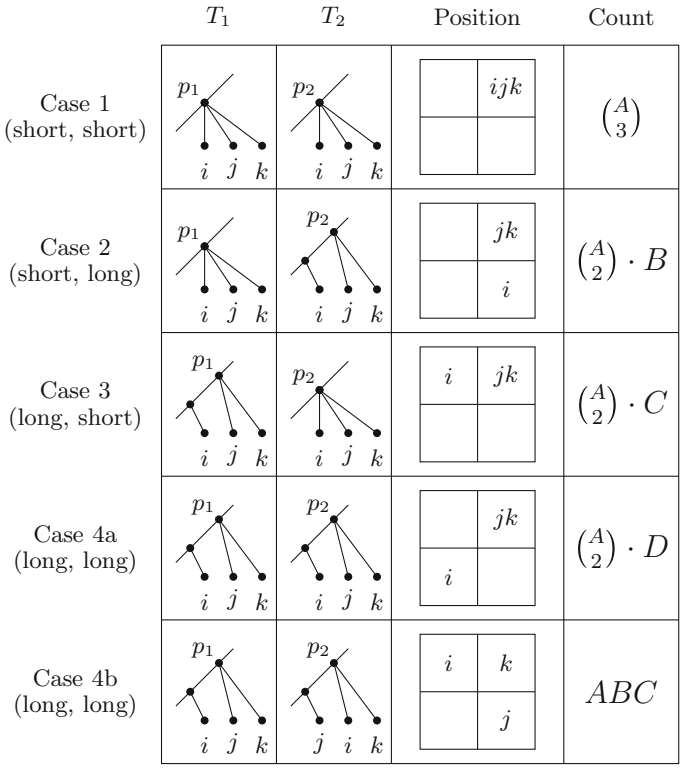

Fig. 6. The cases for counting unresolved triplets rooted at point (p_1, p_2). The Position column shows the partition of $[1, p_1] \times [1, p_2]$ mirroring Fig. 5, so that the top right quadrant corresponds to A, etc.

For every resolved triplet, one of its leaves must be a child of node p_1 in T_1, and p_2 in T_2, therefore it is mapped to the area A. The other two leaves are more related to each other than to the first leaf, which happens only if both leaves are descendants of an internal node in T_1 with index less than p_1, and similarly in T_2. This means these leaves are mapped to the area D. To count the number of good resolved triplets mapped to $\mathbf{p}$, we choose one leaf mapped to A, then choose two leaves mapped to D. Therefore the number of such triplets is $A \cdot \binom{D}{2}$.

For unresolved triplets, we proceed as follows. An unresolved triplet may either be *short*, where all three leaves share the same parent node, or it may be *long*, where only two leaves share a parent node, and the third leaf is located lower in the tree. Counting these triplets splits into four main cases. See Fig. 6.

– Case 1: The triplet is short in both T_1, T_2. This means that each leaf in such a triplet must share the same parents, namely, nodes p_1, p_2 in T_1, T_2 respectively. Therefore, these leaves are all mapped to the point $\mathbf{p}$. The number of such triplets is thus $\binom{A}{3}$.

- Case 2: The triplet is short in T_1, and long in T_2. The number of such triplets is $\binom{A}{2} \cdot B$.
- Case 3: The triplet is long in T_1, and short in T_2. The number of such triplets is $\binom{A}{2} \cdot C$.
- Case 4: The triplet is long in both T_1, T_2. In this triplet, the leaf that is in the lower position of T_1, T_2 may or may not be the same leaf. Thus, two subcases arise:
  - Case 4a: They are the same leaf. The number of such triplets is $\binom{A}{2} \cdot D$.
  - Case 4b: They are different leaves. The number of such triplets is ABC.

Repeating the above for each point to calculate the total number of good triplets subsequently gives us the rooted triplet distance:

Lemma 5.

$$d_{rt}(T_1, T_2) = \binom{n}{3} - \sum_{\mathbf{p} \in \operatorname{Im} f'} \left(A \cdot \binom{D}{2} + \binom{A}{3} + \binom{A}{2} (B + C + D) + ABC \right).$$

The mapping f can be built in $O(n)$ time. Then, for each point $\mathbf{p} \in \operatorname{Im} f'$, we make four range counting queries to find the values A, B, C, D. Afterwards, apply Lemma 5 to obtain $d_{rt}(T_1, T_2)$.

If $f'(\tau) = \mathbf{p}$, then there is at least one leaf $\ell \in \tau$ where $f(\ell) = \mathbf{p}$ also. Therefore, we have $\operatorname{Im} f' \subseteq \operatorname{Im} f$. Since $|\operatorname{Im} f| \leq n$, applying Lemma 4 yields:

Theorem 2. *Given two caterpillars on the same leaf label set of size n, Algorithm 2 computes the rooted triplet distance between them in $O(n\sqrt{\log n})$ time.*

5 Conclusion

The only known lower bound on the time complexity of computing the rooted triplet distance is the trivial one of $\Omega(n)$, which holds because any algorithm has to look at all of its input at least once. Thus, there is a gap between the known upper and lower bounds, and to close this gap is a major open problem. In this paper, we have presented two algorithms that go below the $O(n \log n)$-time upper bound for a certain special class of inputs, namely caterpillars. Although this doesn't solve the open problem, it makes some partial progress. Whether or not the techniques developed here can be extended to more general (non-caterpillar) inputs remains to be seen, but we believe our findings open up an interesting new research direction and that they show there is hope for an $o(n \log n)$-time algorithm for the general case.

Initial experiments on the first algorithm show promising practical performance. A C++ implementation of the algorithm, running on a computer with AMD Ryzen 7 2700X, 16 GB RAM, Arch Linux with kernel version 5.10.16, and g++ compiler version 10.2.0, was able to process inputs of size $n = 1,000,000$ in 3.5 s, using 69 MB of memory. The outcome of the experimental results will be reported in the full version of this paper.

We conclude with some open questions:

- Can the two algorithms be extended to work on a larger class of input trees?
- Our first algorithm uses a dynamic rank-select data structure that provides additional operations such as *delete* and *select* that are not needed by our algorithm. Is it possible to design a simpler and faster data structure that still fits our purposes?
- Is a practical implemention of the second algorithm possible? We note that certain steps such as the mapping and query steps can be parallelized easily.

Acknowledgment. JJ was partially funded by RGC/GRF project 15217019.

References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. Theoret. 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 Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013), pp. 1814–1832. SIAM (2013)
3. Brodal, G.S., Mampentzidis, K.: Cache oblivious algorithms for computing the triplet distance between trees. In: 25th Annual European Symposium on Algorithms (ESA 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
4. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 161–173. SIAM (2010)
5. Critchlow, D.E., Pearl, D.K., Qian, C.: The triples distance for rooted bifurcating phylogenetic trees. Syst. Biol. **45**(3), 323–334 (1996)
6. Dasgupta, B., He, X., Jiang, T., Li, M., Tromp, J.: On computing the nearest neighbor interchange distance. In: Proceedings of the DIMACS Workshop on Discrete Problems with Medical Applications, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 55, pp. 125–143. American Mathematical Soc. (2000)
7. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. J. Classif. **2**(1), 7–28 (1985)
8. 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). https://doi.org/10.1007/BFb0069548
9. Huelsenbeck, J.P., Nielsen, R., Ronquist, F., Bollback, J.P.: Bayesian inference of phylogeny and its impact on evolutionary biology. Science **294**(5550), 2310–2314 (2001)
10. Jansson, J., Mampentzidis, K., T.P, S.: Building a small and informative phylogenetic supertree. In: 19th International Workshop on Algorithms in Bioinformatics (WABI 2019). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019)
11. Jansson, J., Rajaby, R.: A more practical algorithm for the rooted triplet distance. J. Comput. Biol. **24**(2), 106–126 (2017)
12. Kendall, M., Colijn, C.: Mapping phylogenetic trees to reveal distinct patterns of evolution. Mol. Biol. Evol. **33**(10), 2735–2743 (2016)

13. Liao, W., et al.: Alignment-free transcriptomic and metatranscriptomic comparison using sequencing signatures with variable length Markov chains. Sci. Rep. **6**(1), 1–15 (2016)
14. Moreno-Dominguez, D., Anwander, A., Knösche, T.R.: A hierarchical method for whole-brain connectivity-based parcellation. Hum. Brain Mapp. **35**(10), 5000–5025 (2014)
15. Nakhleh, L., Sun, J., Warnow, T., Linder, C.R., Moret, B.M.E., Tholse, A.: Towards the development of computational tools for evaluating phylogenetic network reconstruction methods. In: Biocomputing 2003, pp. 315–326. World Scientific (2002)
16. Page, R.D.M., Cruickshank, R., Johnson, K.P.: Louse (Insecta: Phthiraptera) mitochondrial 12S rRNA secondary structure is highly variable. Insect Mol. Biol. **11**(4), 361–369 (2002)
17. Raman, R., Raman, V., Rao, S.S.: Succinct dynamic data structures. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, pp. 426–437. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44634-6_39
18. Robinson, D.F.: Comparison of labeled trees with valency three. J. Combin. Theory Ser. B **11**(2), 105–119 (1971)
19. Robinson, D.F., Foulds, L.R.: Comparison of phylogenetic trees. Math. Biosci. **53**(1–2), 131–147 (1981)
20. Sand, A., Brodal, G.S., Fagerberg, R., Pedersen, C.N.S., Mailund, T.: A practical $O(n \log^2 n)$ time algorithm for computing the triplet distance on binary trees. BMC Bioinform. **14**, S18 (2013). https://doi.org/10.1186/1471-2105-14-S2-S18