Algorithmica (2013) 66:329–345
DOI 10.1007/s00453-012-9639-1

Constructing the R* Consensus Tree of Two Trees in Subcubic Time

Jesper Jansson · Wing-Kin Sung

Received: 8 March 2011 / Accepted: 29 February 2012 / Published online: 13 April 2012

Abstract The previously fastest algorithms for computing the R* consensus tree of two given (rooted) phylogenetic trees with a leaf label set of cardinality n run in $\Theta(n^3)$ time (Bryant and Berry in Adv. Appl. Math. 27(4):705–732, 2001; Kannan et al. in SIAM J. Comput. 27(6):1695–1724, 1998). In this manuscript, we describe a new $O(n^2\sqrt{\log n})$-time algorithm to solve the problem. This is a significant improvement because the R* consensus tree is defined in terms of a set $\mathcal{R}_{maj}$ which may contain $\Omega(n^3)$ elements, so any direct approach that explicitly constructs $\mathcal{R}_{maj}$ requires $\Omega(n^3)$ time.

Keywords Phylogenetic tree · R* consensus tree · Triplet · Strong cluster · Apresjan cluster · Lowest common ancestor · Offline orthogonal range counting

1 Introduction

Phylogenetic trees are leaf-labeled trees commonly used to describe the evolutionary history of a set of objects such as biological species or languages [1, 9, 10, 16, 18]. Typically, in a phylogenetic tree, each leaf represents one of the objects being studied and the branching structure of the tree shows the assumed evolutionary relationships among the objects. Depending on the available data, it can be difficult to

J. Jansson (✉)
Ochanomizu University, 2-1-1 Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan
e-mail: Jesper.Jansson@ocha.ac.jp

W.-K. Sung
School of Computing, National University of Singapore, COM 1, 13 Computing Drive, Singapore 117417, Singapore
e-mail: ksung@comp.nus.edu.sg

W.-K. Sung
Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672, Singapore

infer an accurate phylogenetic tree. For example, small changes in the input data may result in trees with very different structures. A *consensus tree* is a single phylogenetic tree which summarizes the branching information contained in an input collection of phylogenetic trees with identical leaf label sets. Consensus trees are useful when different data sets or different tree inference methods have produced a set of trees with the same leaf labels and slightly conflicting structures, yet a single tree is required to represent all of them [1, 3, 7, 9, 14, 18] (indeed, phylogenetic analyses often output alternative trees for the same set of species [1]). Also, by exclusion, a consensus tree indicates areas of conflict in the input trees [3]. Furthermore, consensus trees are sometimes used as a basis for new phylogenetic inferences [3].

Over the years, many types of consensus trees have been defined and studied in detail. For a survey, see, e.g., [3], Chap. 30 in [9], Chap. 3.2 in [18], or Chap. 8.4 in [19]. Different types of consensus trees use different criteria to resolve conflicts among the input trees, so their mathematical properties vary. Thus, the most suitable type of consensus tree to use in practice depends on the particular application. In this paper, we focus on the so-called *R* consensus tree*. One advantage of the R* consensus tree is that it provides a statistically consistent estimator of the species tree topology when combining a set of gene trees, as recently demonstrated by Degnan *et al.* [8]; moreover, R* consensus outperformed other methods such as majority-rule consensus in the study conducted by [8]. For the case of two input trees, it is known [3] that the R* consensus tree is equivalent to the *RV-III tree* introduced in [14].

The R* consensus tree is defined in Sect. 2 below.[1] The idea is: For any set of input phylogenetic trees on a fixed leaf label set L, there are certain small binary trees called *rooted triplets* (each containing exactly three leaf labels from L) that occur as embedded subtrees more frequently than others, and the R* consensus tree includes as many of these rooted triplets as possible and none of the others. In short, if the rooted triplet $xy|z$ for any $\{x, y, z\} \subseteq L$ is consistent with more input trees than each of the two rooted triplets $xz|y$ and $yz|x$ is, then $xy|z$ belongs to a set named $\mathcal{R}_{maj}$; the R* consensus tree is the phylogenetic tree τ having the largest possible number of internal nodes such that every rooted triplet consistent with τ belongs to $\mathcal{R}_{maj}$.

The goal of this paper is to develop a fast algorithm for constructing the R* consensus tree for two input trees T_1 and T_2 with a leaf label set L of cardinality n. The previous algorithms for this problem require $\Theta(n^3)$ time [4, 14], which was believed to be optimal because when T_1 and T_2 have similar branching structures, $|\mathcal{R}_{maj}| = \Omega(n^3)$. Our main result is that the time complexity of the problem is in fact *subcubic*. In order to obtain a subcubic-time algorithm, we have to avoid explicitly constructing the set $\mathcal{R}_{maj}$. For this purpose, we compute the values of a function named $s_{\mathcal{R}_{maj}}$ associated to $\mathcal{R}_{maj}$ by using a novel formulation based on distances from leaves to lowest common ancestors of pairs of leaves, and then apply an algorithm by Bryant and Berry [4] to compute the Apresjan clusters of $s_{\mathcal{R}_{maj}}$. Next, we check each of the obtained Apresjan clusters to find all strong clusters of $\mathcal{R}_{maj}$, which are subsequently used to build the R* consensus tree. In total, the running time of our new algorithm `R*_consensus_tree` is $O(n^2\sqrt{\log n})$. (The preliminary conference version of this paper [13] described a slightly slower $O(n^2 \log n)$-time method.)

[1] See Sect. 2 for formal definitions of *rooted triplet*, *consistent with*, *strong cluster*, etc.

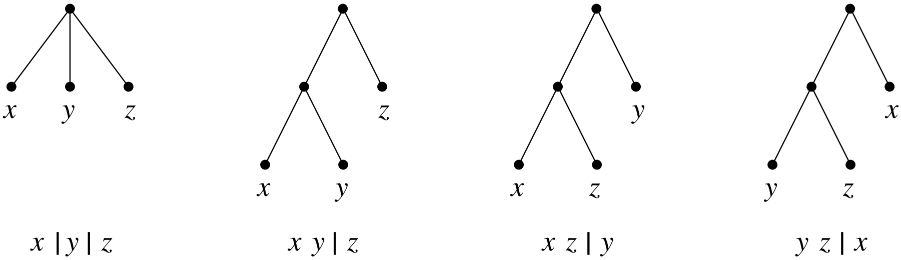

Fig. 1 The four different triplets $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ leaf-labeled by $\{x, y, z\}$

The paper is organized as follows. Section 2 introduces the basic definitions and terminology used throughout the paper and states some simple properties of clusters and the R* consensus tree. Next, our algorithm `R*_consensus_tree` is presented in Sect. 3. Two crucial steps of the algorithm (how to compute the values of the function $s_{\mathcal{R}_{maj}}$ efficiently and how to determine if a cluster is a strong cluster of $\mathcal{R}_{maj}$) are described in detail in Sects. 4 and 5. Finally, Sect. 6 discusses open problems for further research.

2 Preliminaries

2.1 Basic Definitions

A *phylogenetic tree* is a rooted, unordered, distinctly leaf-labeled tree in which every internal node has at least two children. From here on, "tree" means "phylogenetic tree", and every leaf in a tree is identified with its label.

We shall use the following terminology and notation. For any tree T and any node u of T, the subtree of T rooted at u is denoted by $T[u]$. The set of all leaves in a tree T is written as $\Lambda(T)$. To simplify the presentation, every node in a tree is considered to be an ancestor as well as a descendant of itself. For any nodes u, v in a tree, in case u is a descendant of v and $u \neq v$ then we call u a *proper* descendant of v. For any set A of nodes in a tree T, the *lowest common ancestor of A in T* is the node w such that: (1) every node in A is a descendant of w; and (2) w is a proper descendant of x for every other node x which is an ancestor of all nodes in A. The lowest common ancestor of any two nodes u and v in a tree T is denoted by $lca^T(u, v)$.

A *triplet* is a tree with three leaves. Any *non-binary* tree containing exactly three leaves $\{x, y, z\}$ is called a *fan triplet* and is written as $x|y|z$. On the other hand, any *binary* tree with exactly three leaves $\{x, y, z\}$ is called a *rooted triplet*, and is denoted by $xy|z$ if the lowest common ancestor of x and y is a proper descendant of the lowest common ancestor of x and z. Note that there are precisely four different triplets for any set of three leaf labels $\{x, y, z\}$, namely $x|y|z$, $xy|z$, $xz|y$, and $yz|x$; see Fig. 1.

For any tree T and $\{x, y, z\} \subseteq \Lambda(T)$, the fan triplet $x|y|z$ is said to be *consistent with T* if $lca^T(x, y) = lca^T(x, z) = lca^T(y, z)$. Similarly, the rooted triplet $xy|z$ is *consistent with T* if $lca^T(x, y)$ is a proper descendant of $lca^T(x, z)$

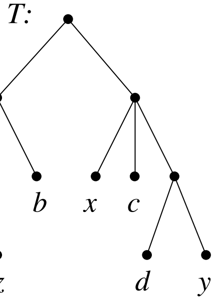

Fig. 2 The rooted triplet $xy|z$ and the fan triplet $x|c|d$ are consistent with the tree T. Hence, $xy|z \in r(T)$, $xy|z \in t(T)$, and $x|c|d \in t(T)$

$= lca^T(y, z)$. Let $T||_{\{x,y,z\}}$ denote the unique triplet with leaf label set $\{x, y, z\}$ which is consistent with T. Finally, for any tree T, let $r(T)$ be the set of all rooted triplets which are consistent with T, i.e., define $r(T) = \{T||_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T)$ and $T||_{\{x,y,z\}}$ is a rooted triplet$\}$, and define $t(T)$ as the set of *all* triplets (rooted triplets and fan triplets) consistent with T, i.e., $t(T) = \{T||_{\{x,y,z\}} : \{x, y, z\} \subseteq \Lambda(T)\}$. (See Fig. 2 for some examples.) It follows that $|t(T)| = \Theta(|\Lambda(T)|^3)$ for any tree T, and $|r(T)| = \Theta(|\Lambda(T)|^3)$ when T is a binary tree because $|r(T)| = |t(T)|$ in this case.

2.2 Strong Clusters and Apresjan Clusters

Below, let $\mathcal{R}$ be a given set of triplets over a leaf label set $L = \bigcup_{r\in\mathcal{R}} \Lambda(r)$ such that for each $\{x, y, z\} \subseteq L$, at most one of $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{R}$. A *cluster of* L is any non-empty subset of L. We define two special types of clusters:

Definition 1 A cluster A of L is called a *strong cluster of* $\mathcal{R}$ if $aa'|x \in \mathcal{R}$ for all $a, a' \in A$ and $x \in L \setminus A$. Furthermore, L as well as every singleton set of L is also defined to be a strong cluster of $\mathcal{R}$.

Definition 2 For each $a, b \in L$ with $a \neq b$, define $s_{\mathcal{R}}(a, b) = |\{y \in L \setminus \{a, b\} : ab|y \in \mathcal{R}\}|$, and for each $a \in L$, define $s_{\mathcal{R}}(a, a) = |L| - 1$. A cluster A of L is called an *Apresjan cluster of* $s_{\mathcal{R}}$ if $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$ for all $a, a' \in A$ and $x \in L \setminus A$.

Write $n = |L|$. By Theorem 2.3 and Corollary 2.1 of [4], the following holds:

Lemma 1 (Bryant and Berry [4])

1. *There are $O(n)$ Apresjan clusters of $s_{\mathcal{R}}$.*
2. *Given the values of $s_{\mathcal{R}}(a, b)$ for all $a, b \in L$, the Apresjan clusters of $s_{\mathcal{R}}$ can be computed in $O(n^2)$ time.*

There is a connection between the strong clusters of $\mathcal{R}$ and the Apresjan clusters of $s_{\mathcal{R}}$:

Lemma 2 *Every strong cluster of $\mathcal{R}$ is an Apresjan cluster of $s_{\mathcal{R}}$.*

Proof Let C be a strong cluster of $\mathcal{R}$. Consider any fixed $a, a' \in C$ and $x \in L \setminus C$. We need to show that $s_{\mathcal{R}}(a, a') > s_{\mathcal{R}}(a, x)$.

First of all, for every $y \in L \setminus C$, we have $aa'|y \in \mathcal{R}$ by the definition of a strong cluster, which gives $s_{\mathcal{R}}(a, a') = |\{aa'|y : aa'|y \in \mathcal{R}\}| \geq |L \setminus C|$. In the same way, $ab|x \in \mathcal{R}$ holds for every $b \in C$, which (along with the requirement that for each $\{x, y, z\} \subseteq L$, at most one of $x|y|z$, $xy|z$, $xz|y$, and $yz|x$ belongs to $\mathcal{R}$) implies that $ax|b \notin \mathcal{R}$. Thus, $s_{\mathcal{R}}(a, x) = |\{ax|y : ax|y \in \mathcal{R}\}| \leq |(L \setminus C) \setminus \{x\}| < |L \setminus C|$. Since $s_{\mathcal{R}}(a, x) < s_{\mathcal{R}}(a, a')$, C is an Apresjan cluster of $s_{\mathcal{R}}$ by definition. □

2.3 R* Consensus Trees

Let T_1 and T_2 be two given trees with $\Lambda(T_1) = \Lambda(T_2) = L$. For any $\{a, b, c\} \subseteq L$, define $\#ab|c$ as the number of trees T_i for which $ab|c \in r(T_i)$. The set of "majority rooted triplets" $\mathcal{R}_{maj}$ is defined as $\{ab|c : a, b, c \in L \text{ and } \#ab|c > \#ac|b, \#bc|a\}$. An *R* consensus tree of* T_1 *and* T_2 is a tree τ with $\Lambda(\tau) = L$ which satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and which maximizes the number of internal nodes.[2]

The next two lemmas describe some useful properties of the strong clusters of $\mathcal{R}_{maj}$.

Lemma 3 *Let T be a tree with $\Lambda(T) = L$ and $r(T) \subseteq \mathcal{R}_{maj}$. For any node u of T, $\Lambda(T[u])$ is a strong cluster of $\mathcal{R}_{maj}$.*

Proof If u is a leaf or the root of T then $\Lambda(T[u])$ is trivially a strong cluster of $\mathcal{R}_{maj}$. If u is an internal non-root node then for any two $a, a' \in \Lambda(T[u])$ and any $x \notin \Lambda(T[u])$, the triplet $aa'|x$ belongs to $\mathcal{R}_{maj}$, so $\Lambda(T[u])$ is a strong cluster of $\mathcal{R}_{maj}$. □

Lemma 4 *There exists a tree τ such that $\{\Lambda(\tau[u]) : u \text{ is a node in } \tau\}$ equals the set of strong clusters of $\mathcal{R}_{maj}$.*

Proof First observe that for any two strong clusters A and B of $\mathcal{R}_{maj}$, either $A \subsetneq B$, $B \subsetneq A$, or $A \cap B = \emptyset$. To prove this claim, suppose for the sake of contradiction that there are $x, y, z \in L$ such that $x \in A \setminus B$, $y \in B \setminus A$, and $z \in A \cap B$. Since A is a strong cluster, $xz|y \in \mathcal{R}_{maj}$. Since B is a strong cluster, $yz|x \in \mathcal{R}_{maj}$. But by the definition of $\mathcal{R}_{maj}$, we cannot have both of $xz|y$ and $yz|x$ in $\mathcal{R}_{maj}$. The claim follows.

As a direct consequence, the strong clusters of $\mathcal{R}_{maj}$ form a nested hierarchy (laminar family) on L. By Theorem 13.21 of [17], the collection of strong clusters of $\mathcal{R}_{maj}$ can be represented by a rooted tree τ together with a function $\pi : L \rightarrow \{\text{nodes of } \tau\}$ such that the strong clusters of $\mathcal{R}_{maj}$ are in one-to-one-correspondence with the nodes of τ in the following sense: For any strong cluster A and any $x \in L$, it holds that $x \in A$

[2] Observe that every rooted triplet consistent with τ must also belong to $\mathcal{R}_{maj}$, i.e., the R* consensus tree is not allowed to introduce any new rooted triplets. In a related problem called *the maximum rooted triplets consistency problem* (MaxRTC), the input is a set $\mathcal{R}$ of (possibly conflicting) rooted triplets and the objective is to infer a tree T consistent with as many rooted triplets as possible from $\mathcal{R}$; in particular, the output T may also be consistent with rooted triplets not present in $\mathcal{R}$. MaxRTC is NP-hard; see [5] for more details and references.

if and only if $\pi(x)$ is a descendant of the node $\tau(A)$ corresponding to A (as before, every node is also defined to be a descendant of itself).[3]

Next, we show that:

- For every $x \in L$, the node $\pi(x)$ must be a leaf: Suppose that for some $x \in L$, the node $\pi(x)$ is an internal node. Consider the strong cluster $\{x\}$. By the above, the node $\pi(x)$ is equal to or a proper descendant of the node $\tau(\{x\})$. Since $\pi(x)$ has at least one child, there exists some node v in τ which is a proper descendant of $\tau(\{x\})$. The (non-empty) strong cluster corresponding to node v is therefore a proper subset of $\{x\}$, which is impossible. Hence, π cannot map any element of L to an internal node of τ.
- Every internal node of τ has at least two children: If A and B are strong clusters of $\mathcal{R}_{maj}$ with $A \subsetneq B$ then there exists some strong cluster C of $\mathcal{R}_{maj}$ such that $C \subsetneq B$ and $C \cap A = \emptyset$. For example, take any $c \in B \setminus A$ and use the fact that $\{c\}$ is a strong cluster of $\mathcal{R}_{maj}$.

By these observations, there exists a tree τ leaf-labeled by L where every internal node has at least two children such that each strong cluster of $\mathcal{R}_{maj}$ equals $\Lambda(\tau[u])$ for some rooted subtree $\tau[u]$ of τ. □

Say that a tree T *includes* a cluster A of L if T contains a node u such that $\Lambda(T[u]) = A$.

Theorem 1 *An R* consensus tree of T_1 and T_2 always exists. In particular, it includes every strong cluster of $\mathcal{R}_{maj}$ and no other clusters of L.*

Proof By Lemma 4, there exists a tree τ that includes all the strong clusters of $\mathcal{R}_{maj}$ and does not include any other clusters. For any $aa'|x \in r(\tau)$, there exists a node u in τ such that $a, a' \in \Lambda(\tau[u])$ and $x \notin \Lambda(\tau[u])$, i.e., $a, a' \in A$ and $x \notin A$ for some strong cluster A of $\mathcal{R}_{maj}$, which implies that $aa'|x \in \mathcal{R}_{maj}$. Thus, $r(\tau) \subseteq \mathcal{R}_{maj}$.

To prove the optimality of τ, suppose that there exists a tree τ' which satisfies $r(\tau') \subseteq \mathcal{R}_{maj}$ and has more internal nodes than τ. By Lemma 3, the set of leaves in each rooted subtree of τ' forms a strong cluster of $\mathcal{R}_{maj}$. Since τ' has more internal nodes than τ, it follows that τ' includes some strong cluster of $\mathcal{R}_{maj}$ which τ does not include. This contradicts Lemma 4. Hence, τ is an R* consensus tree of T_1 and T_2. □

2.4 Constructing a Tree from a Set of Clusters

Here, we address the problem of constructing a tree from a given set of clusters. We first recall a problem known as the directed perfect phylogeny problem with binary characters. Suppose M is a binary matrix. Let I be the set of rows and J the set of columns in M. A *directed perfect phylogeny for M* is a tree T such that: (1) the leaves of T are bijectively labeled by I; and (2) each $c \in J$ is associated with exactly one

[3] In other words, each strong cluster A corresponds to a node $\tau(A)$ in τ and the set of elements from L that are mapped by π to the subtree rooted at $\tau(A)$ is precisely A. Also, $A \subsetneq B$, where A and B are strong clusters of $\mathcal{R}_{maj}$, if and only if the node $\tau(A)$ is a proper descendant of the node $\tau(B)$.

```
Algorithm R*_consensus_tree
Input: Two trees $T_1, T_2$ with $\Lambda(T_1) = \Lambda(T_2)$
Output: The R* consensus tree of $T_1$ and $T_2$

Define $L := \Lambda(T_1) = \Lambda(T_2)$ and compute $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ as described in Sect. 4
Compute the Apresjan clusters of $s_{\mathcal{R}_{maj}}$ according to Lemma 1
for each Apresjan cluster $A$ of $s_{\mathcal{R}_{maj}}$ do
    Determine if $A$ is a strong cluster of $\mathcal{R}_{maj}$ as described in Sect. 5
end for
Let $C$ be the set of strong clusters of $\mathcal{R}_{maj}$, and build a tree $T$ which includes all clusters in $C$ and no
  other clusters of $L$ in accordance with the method in Lemma 5
Output $T$
```

Fig. 3 Algorithm `R*_consensus_tree`

node $T(c)$ of T in such a way that for any $x \in I$, it holds that $M[x, c] = 1$ if and only if leaf x belongs to the subtree of T rooted at node $T(c)$. *The directed perfect phylogeny problem with binary characters* (DPPB) is to, given a binary matrix M, construct a directed perfect phylogeny for M (if one exists). For examples and more details, refer to, e.g., Sect. 17.3 in [10] or Sect. 7.2.2 in [19]. DPPB can be solved in $O(|I| \cdot |J|)$ time by Gusfield's algorithm; see Sect. 17.3.4 in [10] or Sect. 7.2.2.1 in [19].

Now, suppose we are given a set L of leaf labels and a set C of clusters of L. To build a tree that includes all clusters in C and no other clusters of L (if such a tree exists), we first construct a binary matrix M of size $|L| \times |C|$ whose rows represent the leaf labels in L and whose columns represent the clusters in C by letting each entry $M[i, j] = 1$ if and only if the ith leaf label belongs to the jth cluster. Then, we try to build a directed perfect phylogeny T for M by applying Gusfield's algorithm. If such a T exists, it must be unique since all columns of M are distinct (see the comments in Sect. 2 in [11]). However, T might include some clusters not present in C; to ensure that this is not the case, count the number of nodes in T and check if it is equal to $|C|$. This gives:

Lemma 5 *Given a set L of leaf labels and a set C of clusters of L, we can build a tree that includes all clusters in C and no other clusters of L* (*or determine that no such tree exists*) *in $O(|L| \cdot |C|)$ time.*

3 Constructing the R* Consensus Tree

Based on the properties mentioned in Sects. 2.2–2.4, we can construct the R* consensus tree of two given trees T_1 and T_2 as outlined in our main algorithm `R*_consensus_tree` (displayed in Fig. 3).

The strategy of Algorithm `R*_consensus_tree` is as follows: First compute $s_{\mathcal{R}_{maj}}$ and all the Apresjan clusters of $s_{\mathcal{R}_{maj}}$.[4] Next, check each Apresjan cluster to

[4] Recall from Sect. 2.2 that for a set $\mathcal{R}$ of triplets over a leaf label set L, the function $s_{\mathcal{R}}$ is defined as $s_{\mathcal{R}}(a, b) = \left|\{y : ab|y \in \mathcal{R}\}\right|$ for each $a, b \in L$ with $a \neq b$, and $s_{\mathcal{R}}(a, a) = |L| - 1$ for each $a \in L$.

see if it is a strong cluster of $\mathcal{R}_{maj}$ (by Lemma 2, the set of strong clusters of $\mathcal{R}_{maj}$ is a subset of the set of Apresjan clusters of $s_{\mathcal{R}_{maj}}$). Finally, construct a tree T that includes all the strong clusters of $\mathcal{R}_{maj}$ and no other clusters of L by the method in Lemma 5. According to Theorem 1, the resulting T is the R* consensus tree of T_1 and T_2.

We now analyze the time complexity of Algorithm `R*_consensus_tree`. We shall explain how to implement step 1 in $O(n^2\sqrt{\log n})$ time in Sect. 4 (step 1 is in fact the only step that does not take $O(n^2)$ time). In step 2, the Apresjan clusters of $s_{\mathcal{R}_{maj}}$ are computed by running the algorithm of Bryant and Berry from [4], which takes $O(n^2)$ time according to Lemma 1 above. Next, there are $O(n)$ Apresjan clusters to consider in the loop of step 3 by Lemma 1, and each one is checked in $O(n)$ time in step 4, as detailed in Sect. 5 below. Finally, in step 5, the set C satisfies $|C| = O(n)$ because C is a subset of the Apresjan clusters, so applying the method of Lemma 5 takes $O(n^2)$ time.

In summary, we have the following theorem.

Theorem 2 *Algorithm* `R*_consensus_tree` *constructs the R* consensus tree of T_1 and T_2 in $O(n^2\sqrt{\log n})$ time.*

The following two sections are devoted to implementing steps 1 and 4 of `R*_consensus_tree` efficiently.

4 Computing $s_{\mathcal{R}_{maj}}(a, b)$ for All $a, b \in L$ with $a \neq b$

From the definition of $\mathcal{R}_{maj}$, we have:

Lemma 6 *For any $a, b, c \in L$, $ab|c \in \mathcal{R}_{maj}$ if and only if either*

1. *$ab|c \in t(T_1) \cap t(T_2)$; or*
2. *$ab|c \in t(T_1)$ and $a|b|c \in t(T_2)$; or*
3. *$a|b|c \in t(T_1)$ and $ab|c \in t(T_2)$.*

We introduce the following three auxiliary functions:

$$\begin{cases} count_{r,r}(a,b) = \left|\left\{w \in L \setminus \{a,b\} : ab|w \in t(T_1) \cap t(T_2)\right\}\right|, \\ count_{r,f}(a,b) = \left|\left\{w \in L \setminus \{a,b\} : ab|w \in t(T_1), a|b|w \in t(T_2)\right\}\right|, \\ count_{f,r}(a,b) = \left|\left\{w \in L \setminus \{a,b\} : a|b|w \in t(T_1), ab|w \in t(T_2)\right\}\right|. \end{cases}$$

For any $a, b \in L$, the function $count_{r,r}(a,b)$ tells us how many times a triplet involving a and b and some other leaf w occurs as a rooted triplet of the form $ab|w$ in both T_1 and T_2. Similarly, $count_{r,f}(a,b)$ counts how many times a triplet involving a and b and some other leaf w occurs as a rooted triplet of the form $ab|w$ in T_1 and as a fan triplet of the form $a|b|w$ in T_2 (and analogously for $count_{f,r}(a,b)$). Then, Definition 2 and Lemma 6 immediately imply:

Corollary 1 *For every $a, b \in L$ with $a \neq b$,*

$$s_{\mathcal{R}_{maj}}(a,b) = count_{r,r}(a,b) + count_{r,f}(a,b) + count_{f,r}(a,b).$$

(Observe that three auxiliary *count*-functions suffice to express the value of $s_{\mathcal{R}_{maj}}(a,b)$. For our purposes, it is unnecessary to define a fourth auxiliary function $count_{f,f}(a,b)$ that would count how many fan triplets of the form $a|b|w$ that are consistent with both T_1 and T_2.)

To compute $count_{r,r}$, $count_{r,f}$, and $count_{f,r}$, we could preprocess T_1 and T_2 in $O(n)$ time so that lowest common ancestor queries in T_1 and T_2 could be answered in $O(1)$ time [2, 12]. Then, for any given $a, b \in L$, a brute-force solution would easily obtain all of $count_{r,r}(a,b)$, $count_{r,f}(a,b)$, and $count_{f,r}(a,b)$ in $O(n)$ time by checking $T_1||_{\{a,b,w\}}$ and $T_2||_{\{a,b,w\}}$ for every $w \in L \setminus \{a,b\}$. This approach would therefore take $O(n^3)$ time to compute $count_{r,r}(a,b)$, $count_{r,f}(a,b)$, and $count_{f,r}(a,b)$ for all $a, b \in L$. In the rest of this section, we show how to obtain these values more efficiently.

First, in Sect. 4.1, we reformulate the *count*-functions in terms of distances from leaves to lowest common ancestors of leaves. All of these distances may be computed in $O(n^2)$ time. Then, based on this alternative formulation, for any fixed leaf label $a \in L$, Sect. 4.2 describes an $O(n\sqrt{\log n})$-time method for computing $count_{r,r}(a,b)$ for all $b \in L \setminus \{a\}$, and Sect. 4.3 describes an $O(n)$-time method for computing $count_{r,f}(a,b)$ for all $b \in L \setminus \{a\}$. (By symmetry, we can obtain $count_{f,r}(a,b)$ for all $b \in L \setminus \{a\}$ in $O(n)$ time with the same technique as in Sect. 4.3.) To summarize, after $O(n^2)$ time preprocessing, $O(n\sqrt{\log n})$ time is enough to compute $count_{r,r}(a,b)$, $count_{r,f}(a,b)$, and $count_{f,r}(a,b)$ for all $b \in L \setminus \{a\}$ for each fixed $a \in L$. Therefore, we only need $O(n^2\sqrt{\log n})$ time in total to obtain $s_{\mathcal{R}_{maj}}(a,b)$ for all $a, b \in L$ according to Corollary 1.

4.1 Distances from Leaves to Lowest Common Ancestors

For any tree T and any $a, b \in \Lambda(T)$, let $d_{a,T}(b)$ be the distance (i.e., the number of edges) between a and $lca^T(a,b)$ in T, and let $e_{a,T}(b)$ be the child of $lca^T(a,b)$ which is an ancestor of b. Without loss of generality, define $e_{a,T}(a) = \emptyset$. See Fig. 4(a) for an illustration of these definitions.

Note that the values of $d_{a,T}(b)$ and $e_{a,T}(b)$ for all $a, b \in \Lambda(T)$ can be obtained in $O(n^2)$ time in total by performing n simple traversals of T, where $n = |\Lambda(T)|$. More precisely, for each leaf $a \in \Lambda(T)$, spend $O(n)$ time to compute $d_{a,T}(b)$ and $e_{a,T}(b)$ for all $b \in \Lambda(T)$ as follows: Let $d_{a,T}(a) := 0$ and $e_{a,T}(a) := \emptyset$. Start at a and follow the upwards path P from a to the root of T while using a variable d to remember the distance from a to the current location on P. Whenever a new node u on P is visited, let C be the set of children of u not belonging to P, and for each $c \in C$, traverse the subtree of T rooted at c and assign the values $d_{a,T}(b) := d$ and $e_{a,T}(b) := c$ for all leaves b that are encountered.

The next lemma states the relationship between d and e and the triplets consistent with T. For an example, refer to Fig. 4(b). The first part of Lemma 7 is a special case of Theorem 1 in [15], which was derived by Lee *et al.* [15] to solve a different problem known as *the maximum agreement subtree problem* (MAST).

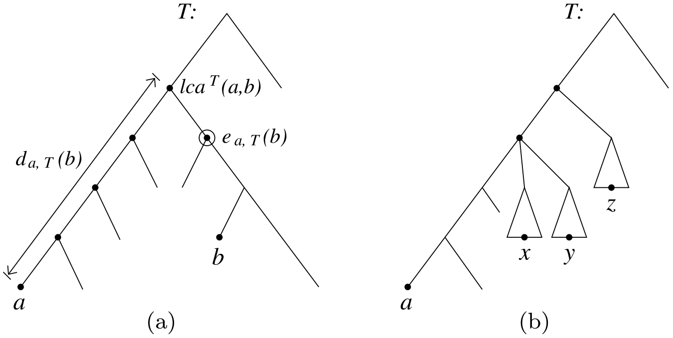

Fig. 4 (**a**) Illustrating the definitions of $d_{a,T}(b)$ and $e_{a,T}(b)$. (**b**) The tree T satisfies $d_{a,T}(x) < d_{a,T}(z)$ so by Lemma 7, $ax|z$ must be consistent with T; similarly, $d_{a,T}(x) = d_{a,T}(y)$ while $e_{a,T}(x) \neq e_{a,T}(y)$, so $a|x|y$ is consistent with T

Lemma 7 *For any tree T and any three distinct $a, x, w \in \Lambda(T)$, it holds that*:

- $d_{a,T}(x) < d_{a,T}(w)$ *if and only if* $ax|w \in t(T)$;
- $d_{a,T}(x) = d_{a,T}(w)$ *and* $e_{a,T}(x) \neq e_{a,T}(w)$ *if and only if* $a|x|w \in t(T)$.

Proof For the first part, $d_{a,T}(x) < d_{a,T}(w)$ if and only if $lca^T(a,x)$ is a proper descendant of $lca^T(a,w)$. This is equivalent to $ax|w \in t(T)$ by the definition of "consistent with T" for a rooted triplet and the definition of $t(T)$.

For the second part, $a|x|w \in t(T)$ if and only if $lca^T(a,x) = lca^T(a,w) = lca^T(x,w)$ by the definition of "consistent with T" for a fan triplet and the definition of $t(T)$. The condition $lca^T(a,x) = lca^T(a,w) = lca^T(x,w)$ directly implies that $d_{a,T}(x) = d_{a,T}(w)$ and $e_{a,T}(x) \neq e_{a,T}(w)$. On the other hand, if $d_{a,T}(x) = d_{a,T}(w)$ and $e_{a,T}(x) \neq e_{a,T}(w)$ then $lca^T(a,x) = lca^T(a,w)$; denote this node by v. Since $e_{a,T}(x)$ and $e_{a,T}(w)$ are two different children of v, we have $lca^T(x,w) = v$ and thus $lca^T(a,x) = lca^T(a,w) = lca^T(x,w)$. □

Next, consider two trees T_1 and T_2 with $\Lambda(T_1) = \Lambda(T_2) = L$. We have:

Lemma 8 *For any $a, b \in L$ with $a \neq b$*:

- $count_{r,r}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$ *and* $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.
- $count_{r,f}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w)$, *and* $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$.
- $count_{f,r}(a,b) = |\{w : d_{a,T_1}(b) = d_{a,T_1}(w), e_{a,T_1}(b) \neq e_{a,T_1}(w)$, *and* $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.

Proof By Lemma 7, $d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)$ mean that $ab|w \in t(T_1) \cap t(T_2)$. Hence, $count_{r,r}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$.

Again, by Lemma 7, $d_{a,T_1}(b) < d_{a,T_1}(w)$, $d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)$ mean that $ab|w \in t(T_1)$ and $a|b|w \in t(T_2)$. Therefore, $count_{r,f}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$.

```
Algorithm Compute_count_rr
Input: $a \in L$
Output: The values of $count_{r,r}(a, b)$ for all $b \in L \setminus \{a\}$

Initialize the data structure for offline orthogonal range counting on an $(n \times n)$-grid $G$ containing the following $n - 1$ points: for $b \in L \setminus \{a\}$, include the point $(d_{a,T_1}(b), d_{a,T_2}(b))$
for $b \in L \setminus \{a\}$ do
    Let $Q_b$ be the rectangle $[(d_{a,T_1}(b) + 1) : n] \times [(d_{a,T_2}(b) + 1) : n]$
end for
Make a query to the data structure for $G$ to ask how many points lie inside each of the $n - 1$ rectangles $Q_b$, where $b \in L \setminus \{a\}$
for $b \in L \setminus \{a\}$ do
    Set $count_{r,r}(a, b)$ to the reported number of points for the rectangle $Q_b$
end for
```

Fig. 5 Algorithm `Compute_count_rr`

The formula for $count_{f,r}(a, b)$ follows by symmetry. □

4.2 Computing $count_{r,r}(a, b)$ for All $b \in L \setminus \{a\}$

Suppose $a \in L$ is fixed. This section describes how to compute $count_{r,r}(a, b)$ for all $b \in L \setminus \{a\}$ in $O(n\sqrt{\log n})$ time, assuming all values of $d_{a,T_1}(b)$, $d_{a,T_2}(b)$ have been precomputed.

Our algorithm is named `Compute_count_rr` and is listed in Fig. 5. It works as follows. Suppose that G is an integer grid of size $(n \times n)$ and each $b \in L \setminus \{a\}$ is represented by the point $(d_{a,T_1}(b), d_{a,T_2}(b))$ on G. Then:

Lemma 9 *For each $b \in L \setminus \{a\}$, the value of $count_{r,r}(a, b)$ equals the number of points inside the rectangle $[(d_{a,T_1}(b) + 1) : n] \times [(d_{a,T_2}(b) + 1) : n]$ in G.*

Proof Lemma 8 states that $count_{r,r}(a, b)$ equals $|\{w : d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)\}|$. Every element w that contributes to this expression satisfies $d_{a,T_1}(b) < d_{a,T_1}(w)$ and $d_{a,T_2}(b) < d_{a,T_2}(w)$, and thus, $d_{a,T_1}(w) \in [(d_{a,T_1}(b)+1) : n]$ and $d_{a,T_2}(w) \in [(d_{a,T_2}(b) + 1) : n]$. □

According to Lemma 9, we can obtain $count_{r,r}(a, b)$ by calculating how many points lie inside the corresponding rectangle on G. Actually, we have to do this for *all* of the $n - 1$ rectangles defined by the elements of $L \setminus \{a\}$. So to implement Lemma 9 efficiently, we borrow a data structure by Chan and Pătraşcu [6] for offline orthogonal range counting on G. More precisely, we use the following result (Corollary 2.3 in [6]):

Lemma 10 (Chan and Pătraşcu [6]) *Given n points and n axis-aligned rectangles in the plane, we can count the number of points inside each rectangle in $O(n\sqrt{\log n})$ total time.*

Note that one query to the data structure returns the number of points in each of the rectangles. By Lemma 10, the running time of Algorithm `Compute_count_rr` becomes $O(n\sqrt{\log n})$.

4.3 Computing $count_{r,f}(a,b)$ for All $b \in L \setminus \{a\}$

Again, suppose $a \in L$ is fixed. Here, we show how to compute $count_{r,f}(a,b)$ (and by symmetry, $count_{f,r}(a,b)$) for all $b \in L \setminus \{a\}$ in $O(n)$ time. We assume that the values of $d_{a,T_1}(b)$, $d_{a,T_2}(b)$, $e_{a,T_1}(b)$, $e_{a,T_2}(b)$ for all $b \in L \setminus \{a\}$ have been precomputed.

For any given a, b, we need to count how many leaves w appear in a rooted triplet of the form $ab|w$ in $t(T_1)$ and in a fan triplet of the form $a|b|w$ in $t(T_2)$ at the same time. Our idea is to first use T_2 to partition $L \setminus \{a\}$ into disjoint subsets $C_1, \dots, C_p$ in such a way that for every pair of elements x, y in the same C_i, it holds that $d_{a,T_2}(x) = d_{a,T_2}(y)$. Also, further partition each C_i into $C_{i,1}, \dots, C_{i,q}$ so that for every pair of elements x, y in the same $C_{i,j}$, it holds that $e_{a,T_2}(x) = e_{a,T_2}(y)$. The second part of Lemma 7 immediately yields:

Lemma 11 *For any $b, w \in L \setminus \{a\}$, the fan triplet $a|b|w$ belongs to $t(T_2)$ if and only if b and w belong to the same C_i-set but different $C_{i,q}$-sets.*

Then, for any $b \in C_{i,j}$, the number of fan triplets of the form $a|b|w$ that are consistent with T_2 equals $|C_i \setminus C_{i,j}|$. However, what we need to know is when a fan triplet $a|b|w$ from T_2 also satisfies $ab|w \in t(T_1)$. To also include this crucial condition on T_1, we refine Lemma 11 in the following way:

Lemma 12 *Let $b \in L \setminus \{a\}$ and suppose $b \in C_{i,j}$. Then, $count_{r,f}(a,b) = |\{w \in C_i : d_{a,T_1}(b) < d_{a,T_1}(w)\}| - |\{w \in C_{i,j} : d_{a,T_1}(b) < d_{a,T_1}(w)\}|$.*

Proof By Lemma 8, $count_{r,f}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), d_{a,T_2}(b) = d_{a,T_2}(w)$, and $e_{a,T_2}(b) \neq e_{a,T_2}(w)\}|$. Since $d_{a,T_2}(b) = d_{a,T_2}(w)$ if and only if $w \in C_i$, and moreover, $e_{a,T_2}(b) = e_{a,T_2}(w)$ if and only if $w \in C_{i,j}$, we have $count_{r,f}(a,b) = |\{w : d_{a,T_1}(b) < d_{a,T_1}(w), w \in C_i$, and $w \notin C_{i,j}\}|$. Next, to count the total number of leaves w from $C_i \setminus C_{i,j}$ that satisfy $d_{a,T_1}(b) < d_{a,T_1}(w)$, just count how many leaves in C_i that satisfy the condition and then subtract the number of leaves in $C_{i,j}$ that also satisfy it. This gives the desired formula for $count_{r,f}(a,b)$. □

Our algorithm Compute_count_rf is specified in Fig. 6. It uses Lemma 12 to compute $count_{r,f}(a,b)$ for all $b \in L \setminus \{a\}$ for any fixed $a \in L$. For each b, define $s(b) = |\{w \in C_i : d_{a,T_1}(b) < d_{a,T_1}(w)\}|$ and $ss(b) = |\{w \in C_{i,j} : d_{a,T_1}(b) < d_{a,T_1}(w)\}|$, where $b \in C_{i,j}$. Then, by Lemma 12, $count_{r,f}(a,b) = s(b) - ss(b)$. The algorithm computes the values of $s(b)$ for all leaves b in each set C_i in steps 4–13 by simple counting and by using a variable named c to keep track of how many consecutive elements from C_i that have the same d_{a,T_1}-value. Next, $ss(b)$ for all leaves b in each set $C_{i,j}$ are obtained analogously in steps 17–26. At last, step 30 assigns the value $s(b) - ss(b)$ to $count_{r,f}(a,b)$. The correctness follows from the correctness of Lemma 12.

Algorithm Compute_count_rf can be implemented to run in $O(n)$ time by using radix sort to sort the elements inside each C_i- and $C_{i,j}$-set in steps 3 and 16. Each $b \in L \setminus \{a\}$ belongs to exactly one C_i-set and exactly one $C_{i,j}$-set, and is therefore treated once in steps 4–13 and once in steps 17–26.

```
Algorithm Compute_count_rf
Input: $a \in L$
Output: The values of $count_{r,f}(a,b)$ for all $b \in L \setminus \{a\}$

Partition $L \setminus \{a\}$ into $C_1, \dots, C_p$ by letting $x \in C_i$ if and only if $d_{a,T_2}(x) = i$
for every $C_i$ do
    Sort the elements of $C_i$ according to their $d_{a,T_1}$-values and denote the resulting ordering
    $b_1, b_2, \dots, b_{|C_i|}$ so that $d_{a,T_1}(b_1) \le d_{a,T_1}(b_2) \le \dots \le d_{a,T_1}(b_{|C_i|})$
    Let $d_{a,T_1}(b_{|C_i|+1}) := \infty$, $s(b_{|C_i|+1}) := 0$, and $c := 0$
    for $j = |C_i|, \dots, 1$ do
        if $d_{a,T_1}(b_j) < d_{a,T_1}(b_{j+1})$ then
            $s(b_j) := s(b_{j+1}) + c$
            $c := 1$
        else
            $s(b_j) := s(b_{j+1})$
            $c := c + 1$
        end if
    end for
    Partition $C_i$ into $C_{i,1}, \dots, C_{i,q}$ so that for every pair of elements $x, y$ in the same $C_{i,j}$, it holds
    that $e_{a,T_2}(x) = e_{a,T_2}(y)$
    for every $C_{i,j}$ do
        Sort the elements of $C_{i,j}$ according to their $d_{a,T_1}$-values and denote the resulting ordering
        $b_1, b_2, \dots, b_{|C_{i,j}|}$ so that $d_{a,T_1}(b_1) \le d_{a,T_1}(b_2) \le \dots \le d_{a,T_1}(b_{|C_{i,j}|})$
        Let $d_{a,T_1}(b_{|C_{i,j}|+1}) := \infty$, $ss(b_{|C_{i,j}|+1}) := 0$, and $c := 0$
        for $k = |C_{i,j}|, \dots, 1$ do
            if $d_{a,T_1}(b_k) < d_{a,T_1}(b_{k+1})$ then
                $ss(b_k) := ss(b_{k+1}) + c$
                $c := 1$
            else
                $ss(b_k) := ss(b_{k+1})$
                $c := c + 1$
            end if
        end for
    end for
end for
for every $b \in L \setminus \{a\}$ do
    $count_{r,f}(a,b) := s(b) - ss(b)$
end for
```

Fig. 6 Algorithm Compute_count_rf

5 Determining if a Cluster Is a Strong Cluster

This section shows how to check whether any given cluster A of L with $|A| \ge 2$ is a strong cluster of $\mathcal{R}_{maj}$ in $O(n)$ time (recall from the definitions in Sect. 2.2 that A is trivially a strong cluster if $|A| = 1$). Our solution relies on the next lemma, illustrated in Fig. 7. For any node u of a tree T, each subtree of T rooted at a child of u is said to be *attached to u*.

Lemma 13 *Let $A \subseteq L$ with $|A| \ge 2$ and define u_1 and u_2 to be the lowest common ancestor of A in T_1 and T_2, respectively. A is a strong cluster of $\mathcal{R}_{maj}$ if and only if:*

(1) *For $i = 1, 2$, every subtree U attached to u_i in T_i satisfies either $\Lambda(U) \subseteq A$ or $\Lambda(U) \subseteq L \setminus A$; and*

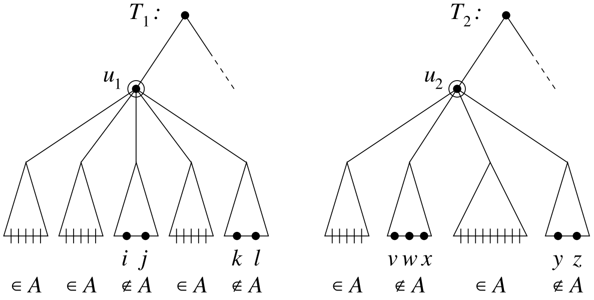

Fig. 7 In this example, $A \subsetneq L$ is a cluster that does not include any of the leaves $i, j, k, l, v, w, x, y, z$. Nodes u_1 and u_2 are the lowest common ancestors of A in T_1 and T_2, respectively. The leaves v, w, x, y, z are not descendants of u_1 in T_1, and i, j, k, l are not descendants of u_2 in T_2. According to the condition in Lemma 13, A is a strong cluster of $\mathcal{R}_{maj}$ because: (1) for each subtree attached to u_1 and u_2, either all its leaves belong to A or none of them do; and (2) the two sets $X_1 = \{i, j, k, l\}$ and $X_2 = \{v, w, x, y, z\}$ (each consisting of leaves that are descendants of u_1 and u_2 but do not belong to A) are disjoint

(2) $X_1 \cap X_2 = \emptyset$, *where* $X_i = \Lambda(T_i[u_i]) \setminus A$.

Proof ($\Rightarrow$) Let A be a strong cluster of $\mathcal{R}_{maj}$. We shall prove that both conditions (1) and (2) hold.

We first prove (1). For the sake of obtaining a contradiction, suppose that for some $i \in \{1, 2\}$, there exist $a \in A$, $x \in L \setminus A$ where both a and x are in the same subtree attached to u_i. Then, $lca^{T_i}(a, x)$ is a proper descendant of u_i. The node u_i is defined as the lowest common ancestor of A, so $lca^{T_i}(a, a') = u_i$ for some $a' \in A$. However, then $lca^{T_i}(a, x)$ is a proper descendant of $lca^{T_i}(a, a')$, giving $ax|a' \in t(T_i)$. This implies that $aa'|x$ cannot belong to $\mathcal{R}_{maj}$, which contradicts the fact that A is a strong cluster of $\mathcal{R}_{maj}$. Thus, condition (1) must hold.

To prove condition (2), we first show that there exist two leaves $a, a' \in A$ such that $lca^{T_1}(a, a') = u_1$ and $lca^{T_2}(a, a') = u_2$. Observe that for both $i = 1, 2$, the leaves from A are located in at least two different subtrees attached to u_i. Thus, there exists at least one pair $a, a' \in A$ with $lca^{T_1}(a, a') = u_1$. Now, assume on the contrary that for every pair $a, a' \in A$ such that $lca^{T_1}(a, a') = u_1$, $lca^{T_2}(a, a')$ is a proper descendant of u_2. Consider any pair of different subtrees attached to u_1 that contain elements from A, and denote their two sets of leaves by A^1 and A^2. By the assumption, $lca^{T_2}(x, y)$ is a proper descendant of u_2 for all $x \in A^1$ and $y \in A^2$, and thus all leaves in $A^1 \cup A^2$ must appear in a single subtree attached to u_2. By repeating this argument, all elements of A appear in one subtree attached to u_2, which is impossible. Therefore, there exists a pair $a, a' \in A$ with $lca^{T_1}(a, a') = u_1$ such that $lca^{T_2}(a, a')$ is not a proper descendant of u_2, i.e., $lca^{T_2}(a, a') = u_2$.

Finally, we are ready to prove (2). Take any $a, a' \in A$ with $lca^{T_1}(a, a') = u_1$ and $lca^{T_2}(a, a') = u_2$ as in the preceding paragraph. To obtain a contradiction, suppose that $X_1 \cap X_2 \neq \emptyset$. Then there exists an $x \in X_1 \cap X_2$, and moreover, $x|a|a' \in t(T_1)$ and $x|a|a' \in t(T_2)$ because the subtree attached to u_i for $i = 1, 2$ that contains x cannot contain a or a' by condition (1) above. But then $aa'|x \not\in \mathcal{R}_{maj}$, contradicting the fact that A is a strong cluster of $\mathcal{R}_{maj}$. Hence, (2) holds.

```
Algorithm Check_if_strong_cluster
Input: A cluster $A$ of $L$ with $|A| \geq 2$
Output: "yes", if $A$ is a strong cluster of $\mathcal{R}_{maj}$; "no", otherwise

Let $u_1$ and $u_2$ be the lowest common ancestor of $A$ in $T_1$ and $T_2$, respectively
for $i = 1, 2$ do
    if any subtree of $T_i$ attached to $u_i$ contains leaves from $A$ as well as from $L \setminus A$ then
        return "no"
    end if
end for
Let $X_i = \Lambda(T_i[u_i]) \setminus A$ for $i = 1, 2$
if $X_1 \cap X_2 = \emptyset$ then
    return "yes"
else
    return "no"
end if
```

Fig. 8 Algorithm Check_if_strong_cluster, based on Lemma 13

($\Leftarrow$) Suppose A satisfies the two conditions stated in the lemma. We will prove that for every $a, a' \in A$ and $c \in L \setminus A$, it holds that $aa'|c \in \mathcal{R}_{maj}$. The leaf c belongs to exactly one of the three disjoint sets X_1, X_2, and $L \setminus (A \cup X_1 \cup X_2)$, which gives three cases:

- If $c \in L \setminus (A \cup X_1 \cup X_2)$: Then $lca^{T_i}(a, a')$ is a proper descendant of $lca^{T_i}(a, c)$ for both $i = 1, 2$. Thus, $aa'|c \in \mathcal{R}_{maj}$.
- If $c \in X_1$: Then c is not a descendant of u_2 in T_2 because $c \notin X_2$, so $aa'|c \in t(T_2)$ always holds. There are two subcases depending on whether or not $lca^{T_1}(a, a')$ is a proper descendant of u_1. If yes, then $aa'|c \in t(T_1)$, and thus $aa'|c \in \mathcal{R}_{maj}$. If no, then $lca^{T_1}(a, a') = u_1$, and thus $a|a'|c \in t(T_1)$, which also yields $aa'|c \in \mathcal{R}_{maj}$.
- If $c \in X_2$: Then it follows that $aa'|c \in \mathcal{R}_{maj}$ in the same way as for the case $c \in X_1$ above.

By definition, A is a strong cluster of $\mathcal{R}_{maj}$. □

Armed with Lemma 13, it is straightforward to check if any given cluster A of L is a strong cluster of $\mathcal{R}_{maj}$ in $O(n)$ time as shown in Algorithm Check_if_strong_cluster in Fig. 8.

6 Concluding Remarks

In this paper, we have shown how to construct the R* consensus tree of two input trees in $O(n^2\sqrt{\log n}) = o(n^3)$ time. It is an open problem to determine if the time complexity can be further improved to $O(n^2)$ or better. The bottleneck of our algorithm R*_consensus_tree is the computation of $count_{r,r}$ in Sect. 4.2, which uses $O(n\sqrt{\log n})$ time for each fixed $a \in L$ to obtain the values of $count_{r,r}(a, b)$ for all $b \in L \setminus \{a\}$.

We remark that for the restricted case where both input trees are *binary*, the problem becomes much easier because then the R* consensus tree is equivalent to the so-called *RV-I tree* in [14]. The RV-I tree differs slightly from the R* consensus tree in general in that if $ab|c$ belongs to one of $t(T_1)$ and $t(T_2)$, and $a|b|c$ belongs to the other, then $ab|c$ may never be consistent with the output tree; however, for the special case of binary trees, neither $t(T_1)$ nor $t(T_2)$ contain any fan triplets so $\mathcal{R}_{maj}$ equals the set of all rooted triplets that are consistent with both T_1 and T_2, and this is precisely the "set of triples which are resolved identically in T_1 and T_2" in Definition 5.1 in [14]. Furthermore, according to Theorem 5.1 in [14], the RV-I tree of T_1 and T_2 is always equivalent to the *strict consensus tree* (see also [3, 7, 9, 18, 19]) of T_1 and T_2, which can be constructed in $O(n)$ time by a classical algorithm of Day [7]. Thus, there is a large gap in the time complexity of constructing the R* consensus tree of two trees in the binary case ($O(n)$ time) and in the general, non-binary case ($O(n^2\sqrt{\log n})$ time).

The definition of the set of majority rooted triplets $\mathcal{R}_{maj}$ as well as the definition of an R* consensus tree can be naturally extended to the case of $k > 2$ input trees; see [3]. The currently fastest algorithm for the case $k > 2$, outlined in [3], runs in $O(kn^3)$ time. It can be implemented by explicitly constructing the sets $r(T_i)$ for every input tree T_i using $O(kn^3)$ total time, then obtaining $\mathcal{R}_{maj}$ in $O(kn^3)$ time by finding the most frequently occurring rooted triplet (if it exists) for each $\{x, y, z\}$ in the leaf label set in $O(k)$ time, and finally applying the $O(n^3)$-time strong cluster algorithm from Corollary 2.2 in [4] to $\mathcal{R}_{maj}$. An important open problem is to reduce the time complexity to $o(kn^3)$. It seems difficult to extend the approach used in this paper because it would yield an exponential number (in k) of cases in Lemma 6 and thus express $s_{\mathcal{R}_{maj}}(a, b)$ as the sum of an exponential number of auxiliary *count*-functions, causing the total running time to be exponential in k.

Acknowledgements The authors would like to thank David Bryant for some clarifications and the anonymous referees for their helpful comments. JJ was supported by the Special Coordination Funds for Promoting Science and Technology and KAKENHI grant number 23700011.

References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. In: Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN 2008). LNCS, vol. 4957, pp. 72–83. Springer, Berlin (2008)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN 2000). LNCS, vol. 1776, pp. 88–94. Springer, Berlin (2000)
3. Bryant, D.: A classification of consensus methods for phylogenetics. In: Janowitz, M.F., Lapointe, F.-J., McMorris, F.R., Mirkin, B., Roberts, F.S. (eds.) Bioconsensus. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 61, pp. 163–184. Am. Math. Soc., Providence (2003)
4. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. Adv. Appl. Math. **27**(4), 705–732 (2001)
5. Byrka, J., Guillemot, S., Jansson, J.: New results on optimizing rooted triplets consistency. Discrete Appl. Math. **158**(11), 1136–1147 (2010)

6. Chan, T.M., Pătraşcu, M.: Counting inversions, offline orthogonal range counting, and related problems. In: Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010), pp. 161–173. SIAM, Philadelphia (2010)
7. Day, W.H.E.: Optimal algorithms for comparing trees with labeled leaves. J. Classif. **2**(1), 7–28 (1985)
8. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. Syst. Biol. **58**(1), 35–54 (2009)
9. Felsenstein, J.: Inferring Phylogenies. Sinauer, Sunderland (2004)
10. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, New York (1997)
11. Gusfield, D.: Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In: Proceedings of the 6th Annual International Conference on Computational Biology (RECOMB 2002), pp. 166–175. ACM, New York (2002)
12. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
13. Jansson, J., Sung, W.-K.: Constructing the R* consensus tree of two trees in subcubic time. In: Proceedings of the 18th Annual European Symposium on Algorithms (ESA 2010). LNCS, vol. 6346, pp. 573–584. Springer, Berlin (2010)
14. Kannan, S., Warnow, T., Yooseph, S.: Computing the local consensus of trees. SIAM J. Comput. **27**(6), 1695–1724 (1998)
15. Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. Inf. Process. Lett. **94**(5), 211–216 (2005)
16. Nakhleh, L., Warnow, T., Ringe, D., Evans, S.N.: A comparison of phylogenetic reconstruction methods on an Indo-European dataset. Trans. Philol. Soc. **103**(2), 171–192 (2005)
17. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Algorithms and Combinatorics, vol. 24/A. Springer, Berlin (2003)
18. Scornavacca, C.: Supertree methods for phylogenomics. PhD thesis, University of Montpellier II, France (2009)
19. Sung, W.-K.: Algorithms in Bioinformatics: A Practical Introduction. Chapman & Hall/CRC, Boca Raton (2010)