Faster Algorithms for Computing the R* Consensus Tree

Jesper Jansson[1](✉), Wing-Kin Sung[2,3], Hoa Vu[4], and Siu-Ming Yiu[5]

[1] Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan
jj@kuicr.kyoto-u.ac.jp

[2] School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Singapore
ksung@comp.nus.edu.sg

[3] Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672, Singapore

[4] Department of Computer Science and Engineering, University of Minnesota – Twin Cities, Minneapolis, MN, USA
hoavu89@gmail.com

[5] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong, China
smyiu@cs.hku.hk

Abstract. The fastest known algorithms for computing the R* consensus tree of k rooted phylogenetic trees with n leaves each and identical leaf label sets run in $O(n^2\sqrt{\log n})$ time when $k = 2$ (ref. [10]) and $O(kn^3)$ time when $k \geq 3$ (ref. [4]). This paper shows how to compute it in $O(n^2)$ time for $k = 2$, $O(n^2 \log^{4/3} n)$ time for $k = 3$, and $O(n^2 \log^{k+2} n)$ time for unbounded k.

1 Introduction

Distinctly leaf-labeled, unordered trees known as *phylogenetic trees* are used by scientists to describe evolutionary history [8,13]. Given a set $\mathcal{S}$ of phylogenetic trees with the same leaf labels but different branching structures, a single phylogenetic tree that summarizes the trees in $\mathcal{S}$ according to some well-defined rule is called a *consensus tree* [4,8,13]. Consensus trees are used when dealing with unreliable data; e.g., to infer an accurate phylogenetic tree for a fixed set of species, one may first construct a collection of alternative trees by applying resampling techniques such as bootstrapping to the same data set, by running different tree construction algorithms, or by using many independent data sets, and then compute a consensus tree from the obtained trees.

A number of different consensus trees have been defined and studied in the literature; see [4], Chapter 30 in [8], or Chapter 8.4 in [13] for some surveys. This paper deals with one particular consensus tree called the *R* consensus tree* [4],

Jesper Jansson: Funded by The Hakubi Project and KAKENHI grant number 26330014.

DOI: 10.1007/978-3-319-13075-0_33

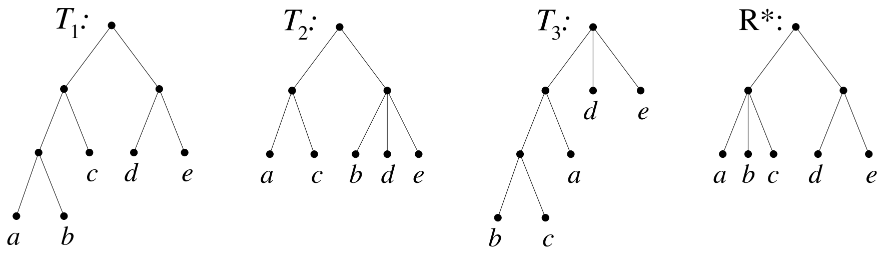

Fig. 1. An example. Let $\mathcal{S} = \{T_1, T_2, T_3\}$ as above. Then $\mathcal{R}_{maj} = \{ab|d,\ ab|e,\ ac|d,\ ac|e,\ de|a,\ bc|d,\ bc|e,\ de|b,\ de|c\}$ and the R* consensus tree of $\mathcal{S}$ is the tree on the right.

defined in Section 1.1 below. The R* consensus tree has several nice mathematical properties [7]. On the negative side, the existing algorithms for building it [4,10,11] are rather slow. To alleviate this issue, we present faster algorithms.

1.1 Definitions and Notation

In this paper, a *phylogenetic tree* is a rooted, unordered, leaf-labeled tree in which every internal node has at least two children and all leaves have different labels. See Fig. 1 for some examples. (*Unrooted* phylogenetic trees are also useful in many contexts [8], but will not be considered here.) Phylogenetic trees are called "trees" from here on, and every leaf in a tree is identified with its label.

Let T be a tree. The set of all nodes in T and the set of all leaves in T are denoted by $V(T)$ and $\Lambda(T)$, respectively. For any $u \in V(T)$, T^u is the subtree of T rooted at u. For any $X \subseteq V(T)$, $lca^T(X)$ is the lowest common ancestor in T of the nodes in X; when $|X| = 2$, we simplify the notation to $lca^T(u,v)$, where $X = \{u,v\}$, and if T is unambiguous, we sometimes just write $lca(u,v)$.

A *triplet* is a tree with exactly three leaves. Suppose t is a triplet with $\Lambda(t) = \{x,y,z\}$. If t is non-binary, it has one internal node; in this case, t is called a *fan triplet* and is denoted by $x|y|z$. Otherwise, t is binary and has two internal nodes; in this case, t is called a *resolved triplet* and is denoted by $xy|z$ where $lca^t(x,y)$ is a proper descendant of $lca^t(x,z) = lca^t(y,z)$. Thus, there are four possible triplets $x|y|z$, $xy|z$, $xz|y$, $yz|x$ for any set of three leaves $\{x,y,z\}$. For any tree T and $\{x,y,z\} \subseteq \Lambda(T)$, $x|y|z$ is said to be *consistent with* T if $lca^T(x,y) = lca^T(x,z) = lca^T(y,z)$, and $xy|z$ is *consistent with* T if $lca^T(x,y)$ is a proper descendant of $lca^T(x,z) = lca^T(y,z)$. Let $T||_{\{x,y,z\}}$ be the unique triplet with leaf set $\{x,y,z\}$ that is consistent with T. For any tree T, let $r(T)$ be the set of resolved triplets consistent with T and let $t(T)$ be the set of *all* triplets (resolved triplets as well as fan triplets) 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 resolved triplet$\}$ and $t(T) = \{T||_{\{x,y,z\}} : \{x,y,z\} \subseteq \Lambda(T)\}$.

Next, let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a given set of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$. Write $n = |L|$. For any $\{a,b,c\} \subseteq L$, define $\#ab|c$ as the number of trees $T_i \in \mathcal{S}$ for which $ab|c \in t(T_i)$. The *set of majority resolved triplets*, denoted by $\mathcal{R}_{maj}$, is defined as $\{ab|c : a, b, c \in L \text{ and } \#ab|c > \max\{\#ac|b, \#bc|a\}\}$. (Note that the fan triplets consistent with the trees in $\mathcal{S}$ have no impact here.) An *R^* consensus tree of $\mathcal{S}$* is a tree τ with $\Lambda(\tau) = L$ that satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ and that maximizes the number of internal nodes. See Fig. 1 for an example.

For any leaf label set L, a *cluster of L* is any nonempty subset of L, and a tree T is said to *include* a cluster A of L if T contains a node u such that $\Lambda(T^u) = A$. Let $\mathcal{R}$ be a 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 A of L is called a *strong cluster of $\mathcal{R}$* if $aa'|x \in \mathcal{R}$ for all $a, a' \in A$ with $a \neq a'$ and all $x \in L \setminus A$. Furthermore, L as well as every singleton set of L is also defined to be a strong cluster of $\mathcal{R}$. Strong clusters provide an alternative characterization of R* consensus trees, stated in the last part of the next lemma:

Lemma 1. *[4, 10] The R^* consensus tree always exists, is unique, and includes every strong cluster of $\mathcal{R}_{maj}$ and no other clusters.*

1.2 Previous Work

The R* consensus tree can be computed in $O(kn^3)$ time, where $k = |\mathcal{S}|$ and $n = |L|$, by an algorithm from [4]: First construct $r(T_i)$ for all $T_i \in \mathcal{S}$ in $O(kn^3)$ time, then construct $\mathcal{R}_{maj}$ by counting the occurrences in the $r(T_i)$-sets of the different resolved triplets for every $\{x, y, z\} \in L$ in $O(kn^3)$ total time, and finally apply the $O(n^3)$-time strong cluster algorithm from Corollary 2.2 in [5] to $\mathcal{R}_{maj}$. For $k = 2$, an older algorithm for computing the so-called RV-III tree of two input trees in $O(n^3)$ time [11] can also be used [4] to achieve the same running time.

Since $\mathcal{R}_{maj}$ may contain $\Omega(n^3)$ elements, any method that explicitly constructs $\mathcal{R}_{maj}$ requires $\Omega(n^3)$ time. For the special case of $k = 2$, it was shown in [10] that the R* consensus tree can in fact be computed in $O(n^2\sqrt{\log n})$ $(= o(n^3))$ time. The algorithm from [10] is reviewed in Section 1.3.

1.3 Overview and Organization of the Paper

To compute the R* consensus tree without constructing $\mathcal{R}_{maj}$, the algorithm in [10] for $k = 2$ and the new algorithms in this paper follow the same basic strategy, summarized as Algorithm `R*_consensus_tree` in Fig. 2. To explain the details, some additional definitions are needed.

Suppose that $\mathcal{R}$ is 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 $\mathcal{R}$. For each $a, b \in L$ with $a \neq b$, define $s_{\mathcal{R}}(a, b) = \left|\{y \in L : ab|y \in \mathcal{R}\}\right|$, and for each $a \in L$, define $s_{\mathcal{R}}(a, a) = \left|L\right| - 1$. 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 $x \in L \setminus A$. Since every strong cluster of $\mathcal{R}$ is an Apresjan cluster of $s_{\mathcal{R}}$ [4, 10], we see that in the case $\mathcal{R} = \mathcal{R}_{maj}$, the set of Apresjan clusters of $s_{\mathcal{R}_{maj}}$ forms a superset of the set of strong clusters of $\mathcal{R}_{maj}$. Moreover, by Theorem 2.3 in [5], there are

```
Algorithm R*_consensus_tree
Input: A set $\mathcal{S} = \{T_1, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \ldots = \Lambda(T_k) = L$
Output: The R* consensus tree of $\mathcal{S}$
1: Compute and store $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$
2: Compute the Apresjan clusters of $s_{\mathcal{R}_{maj}}$
3: for each Apresjan cluster $A$ of $s_{\mathcal{R}_{maj}}$ do
4:     Determine if $A$ is a strong cluster of $\mathcal{R}_{maj}$
5: end for
6: 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 $L$
7: Output $T$
```

Fig. 2. Algorithm `R*_consensus_tree`

$O(n)$ Apresjan clusters of $s_{\mathcal{R}_{maj}}$ and they form a nested hierarchy on L, i.e., a tree, which can be constructed in $O(n^2)$ time with the method of Corollary 2.1 in [5] when the value of $s_{\mathcal{R}_{maj}}(a, b)$ for any $a, b \in L$ is available in $O(1)$ time.

Now, the idea behind Algorithm `R*_consensus_tree` is to first compute a superset of the set of strong clusters of $\mathcal{R}_{maj}$, namely the Apresjan clusters of $s_{\mathcal{R}_{maj}}$ (Steps 1 and 2), then remove any clusters that are not strong clusters of $\mathcal{R}_{maj}$ (Steps 3–5), and return a tree that includes precisely the remaining clusters (Steps 6–7). By Lemma 1, this tree is the R* consensus tree.

The algorithm's time complexity depends on various factors. As shown in [10], if $k = 2$ then computing the values of $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ in Step 1 can be done in $O(n^2\sqrt{\log n})$ time in total, while all other steps take $O(n^2)$ time. Section 2 below improves it to $O(n^2)$, yielding an $O(n^2)$-time solution for $k = 2$.

For $k \geq 3$, we observe that Steps 2, 6, and 7 do not depend on k, so these steps take a total of $O(n^2)$ time as in [10]. However, Steps 1 and 3–5 have to be modified; for example, the condition from Lemma 13 in [10] for checking if a given cluster is a strong cluster of $\mathcal{R}_{maj}$ does not work if $k = 3$. As for Step 1, Sections 3.1–3.3 show how to compute $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ in $O(n^2 \log^{4/3} n)$ time when $k = 3$, and Section 4.1 in $O(n^2 \log^k n)$ time for unbounded k. For Steps 3–5, Section 3.4 gives an $O(n^2\alpha(n))$-time solution when $k = 3$, where $\alpha(n)$ is the inverse Ackermann function of n, while Section 4.2 gives an $O(n^2 \log^{k+2} n)$-time solution for unbounded k. In summary, we obtain:

Theorem 1. *Let $\mathcal{S}$ be an input set of k trees with n leaves each and identical leaf label sets. The R* consensus tree of $\mathcal{S}$ can be computed in:*

- *$O(n^2)$ time when $k = 2$;*
- *$O(n^2 \log^{4/3} n)$ time when $k = 3$; and*
- *$O(n^2 \log^{k+2} n)$ time when k is unbounded.*

Thus, if $k < \frac{\log n}{(\log \log n)^{1+\epsilon}}$ for some $\epsilon > 0$, the time complexity is subcubic in n.

Due to space constraints, most of the proofs have been omitted from the conference proceedings version of this paper.

2 Computing the R* Consensus Tree When $k = 2$

This section proves that $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ with $a \neq b$ can be computed in $O(n^2)$ time in total when $k = 2$, thereby reducing the time complexity of Step 1 of Algorithm `R*_consensus_tree` in Section 1.3 (and hence the algorithm's overall running time) to $O(n^2)$.

Recall that $s_{\mathcal{R}_{maj}}(a, b) = \big|\{w : ab|w \in \mathcal{R}_{maj}\}\big|$ for any $a, b \in L$ with $a \neq b$, and $s_{\mathcal{R}_{maj}}(a, a) = |L| - 1$ for any $a \in L$. By definition, $ab|w \in \mathcal{R}_{maj}$ if and only if it is consistent with both T_1 and T_2, or it is consistent with one of T_1 and T_2 and $a|b|w$ is consistent with the other tree. By Corollary 1 in [10], $s_{\mathcal{R}_{maj}}(a, b) = count_{r,r}(a, b) + count_{r,f}(a, b) + count_{f,r}(a, b)$ for every $a, b \in L$ with $a \neq b$, where $count_{r,r}(a, b) = \big|\{w \in L \setminus \{a, b\} : ab|w \in t(T_1) \cap t(T_2)\}\big|$, $count_{r,f}(a, b) = \big|\{w \in L \setminus \{a, b\} : ab|w \in t(T_1),\ a|b|w \in t(T_2)\}\big|$, and $count_{f,r}(a, b) = \big|\{w \in L \setminus \{a, b\} : a|b|w \in t(T_1),\ ab|w \in t(T_2)\}\big|$. It was shown in [10] that $count_{r,r}(a, b)$, $count_{r,f}(a, b)$, and $count_{f,r}(a, b)$ for all $a, b \in L$ can be calculated in $O(n^2\sqrt{\log n})$, $O(n^2)$, and $O(n^2)$ total time, respectively. We now eliminate the bottleneck.

Lemma 2. *For every $a, b \in L$, it holds that $count_{r,r}(a, b) = |L| - |\Lambda(T_1^{lca(a,b)})| - |\Lambda(T_2^{lca(a,b)})| + |\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)})|$.*

Lemma 3. *$count_{r,r}(a, b)$ for all $a, b \in L$ can be computed in $O(n^2)$ time in total.*

Proof. For $i \in \{1, 2\}$, compute and store all values of $|\Lambda(T_i^u)|$, where $u \in V(T_i)$, in $O(n)$ time by doing a bottom-up traversal of each tree. Also, compute and store all values of $|\Lambda(T_1^u) \cap \Lambda(T_2^v)|$, where $u \in V(T_1)$ and $v \in V(T_2)$, in $O(n^2)$ time by the postorder traversal-based method used in Lemma 7.1 in [1]. Preprocess T_1 and T_2 in $O(n)$ time so that any subsequent *lca*-query can be answered in $O(1)$ time [2,9]. Next, for each $a, b \in L$, obtain $count_{r,r}(a, b)$ in $O(1)$ time by applying the formula in Lemma 2. The total running time is $O(n^2)$. $\square$

3 Computing the R* Consensus Tree When $k = 3$

We now focus on the case $k = 3$. Sections 3.1–3.3 and Section 3.4 describe how to implement Step 1 and Steps 3–5, respectively, of Algorithm `R*_consensus_tree`.

3.1 Computing $s_{\mathcal{R}_{maj}}$ When $k = 3$

Suppose $\mathcal{S} = \{T_1, T_2, T_3\}$. For every $ab|w \in \mathcal{R}_{maj}$, there are three possibilities:

Lemma 4. *For any $a, b, w \in L$, $ab|w \in \mathcal{R}_{maj}$ if and only if either*

1. *$ab|w$ is consistent with T_1, T_2, and T_3; or*
2. *$ab|w$ is consistent with T_i and T_j but not T_k for $\{i, j, k\} = \{1, 2, 3\}$; or*
3. *$ab|w$ is consistent with one of T_1, T_2, T_3, and $a|b|w$ with the other two.*

To count the triplets covered by the different cases in Lemma 4, define:

$$\begin{cases} count_{r,r,r}(a,b) = \big|\{w \in L \setminus \{a,b\} : ab|w \in t(T_1) \cap t(T_2) \cap t(T_3)\}\big| \\ count_{r,r,*}^{T_i,T_j}(a,b) = \big|\{w \in L \setminus \{a,b\} : ab|w \in t(T_i) \cap t(T_j)\}\big|, i,j \in \{1,2,3\}, i<j \\ count_{r,f,f}^{T_i}(a,b) = \big|\{w \in L \setminus \{a,b\} : ab|w \in t(T_i) \text{ and } a|b|w \text{ is consistent with} \\ \qquad\qquad \text{the other two trees}\}\big|, \text{ for } i \in \{1,2,3\} \end{cases}$$

Then, $s_{\mathcal{R}_{maj}}(a,b)$ can be expressed as in the next lemma.

Lemma 5. *Let $a,b \in L$ with $a \neq b$. Then $s_{\mathcal{R}_{maj}}(a,b) = \sum_{i=1}^{3} count_{r,f,f}^{T_i}(a,b) + \sum_{1 \le i < j \le 3} count_{r,r,*}^{T_i,T_j}(a,b) - 2count_{r,r,r}(a,b)$.*

For each pair $i,j \in \{1,2,3\}$ with $i<j$, the values of $count_{r,r,*}^{T_i,T_j}(a,b)$ for all $a,b \in L$ can be obtained in $O(n^2)$ time by the method from Lemma 3 in Section 2 with T_i and T_j as the two input trees. The next subsections show how to calculate the values of $count_{r,r,r}(a,b)$ for all $a,b \in L$ in $O(n^2 \log^{4/3} n)$ time (Lemma 9 in Section 3.2) and $count_{r,f,f}^{T_i}(a,b)$ for all $a,b \in L$ for each $i \in \{1,2,3\}$ in $O(n^2)$ time (Lemma 12 in Section 3.3). Then, we can apply the formula in Lemma 5 to get each value of $s_{\mathcal{R}_{maj}}(a,b)$ in $O(1)$ time. In summary:

Lemma 6. *When $k=3$, the values of $s_{\mathcal{R}_{maj}}(a,b)$ for all $a,b \in L$ can be computed in $O(n^2 \log^{4/3} n)$ time in total.*

3.2 Computing $\boldsymbol{count_{r,r,r}}$

First, rewrite $count_{r,r,r}(a,b)$ in a way analogous to the expression in Lemma 2:

Lemma 7. *For every $a,b \in L$, $count_{r,r,r}(a,b) = |L| - \sum_{i=1}^{3} |\Lambda(T_i^{lca(a,b)})| + \sum_{1 \le i < j \le 3} |\Lambda(T_i^{lca(a,b)}) \cap \Lambda(T_j^{lca(a,b)})| - |\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$.*

Lemma 8. *Let $a \in L$ be fixed. Then the values of $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$ for all $b \in L \setminus \{a\}$ can be computed in $O(n \log^{4/3} n)$ time in total.*

Proof. For $w \in L \setminus \{a\}$ and $i \in \{1,2,3\}$, let $d^{T_i}(w)$ be the distance in T_i from a to $lca(a,w)$. For any $b,w \in L \setminus \{a\}$ and $i \in \{1,2,3\}$, $w \in \Lambda(T_i^{lca(a,b)})$ if and only if $d^{T_i}(w) \le d^{T_i}(b)$. Thus, for $b \in L \setminus \{a\}$, $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})| = |\{w \in L \setminus \{a,b\} : d^{T_1}(w) \le d^{T_1}(b),\ d^{T_2}(w) \le d^{T_2}(b),\ d^{T_3}(w) \le d^{T_3}(b)\}|$.

Represent each $w \in L \setminus \{a\}$ as a 3D point with coordinates $(d^{T_1}(w), d^{T_2}(w), d^{T_3}(w))$. For any $b \in L \setminus \{a\}$, $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$ equals the number of points on or inside the box $[1 : d^{T_1}(b)] \times [1 : d^{T_2}(b)] \times [1 : d^{T_3}(b)]$. Use Corollary 4.1 in [6] for offline orthogonal range counting in 3D to obtain these numbers for all $b \in L \setminus \{a\}$ in $O(n \log^{3-2+1/3} n) = O(n \log^{4/3} n)$ total time. □

Lemma 9. *The values of $count_{r,r,r}(a,b)$ for all $a,b \in L$ can be computed in $O(n^2 \log^{4/3} n)$ total time.*

3.3 Computing $count_{r,f,f}^{T_i}$

This subsection describes how to compute all values of $count_{r,f,f}^{T_1}(a,b) = \big|\{w \in L \setminus \{a,b\} : ab|w \in t(T_1),\ a|b|w \in t(T_2), \text{ and } a|b|w \in t(T_3)\}\big|$, where $a,b \in L$. (The two functions $count_{r,f,f}^{T_2}$ and $count_{r,f,f}^{T_3}$ can be computed in the same way.)

Suppose that $a \in L$ is fixed. Let $v_0 = a, v_1, \ldots, v_p$ be the path in T_3 from leaf a to the root of T_3. For $j \in \{1,\ldots,p\}$, define $W_j = \Lambda(T_3^{v_j}) \setminus \Lambda(T_3^{v_{j-1}})$. Importantly, $\{W_1,\ldots,W_p\}$ forms a partition of $L \setminus \{a\}$. For any $S \subseteq L$ and $b \in S$, define $\sigma^{T_1,\neg T_2}(S,b) = |\{w \in S : ab|w \in t(T_1) \text{ and } a|b|w \in t(T_2)\}|$. Lemma 10 explains how to use $\sigma^{T_1,\neg T_2}(S,b)$ to compute $count_{r,f,f}^{T_1}(a,b)$.

Lemma 10. *For any W_j, $j \in \{1,\ldots,p\}$, and any $b \in W_j$, let c_b be the child of v_j such that $b \in \Lambda(T_3^{c_b})$. Then $count_{r,f,f}^{T_1}(a,b) = \sigma^{T_1,\neg T_2}(W_j,b) - \sigma^{T_1,\neg T_2}(\Lambda(T_3^{c_b}),b)$.*

Lemma 11. *After $O(n)$ time preprocessing, given any $S \subseteq L$, $\sigma^{T_1,\neg T_2}(S,b)$ for all $b \in S$ can be computed in $O(|S|)$ time.*

This suggests the following algorithm, which we call `Compute_count_rff_T1`, for computing $count_{r,f,f}^{T_1}(a,b)$ for all $b \in L \setminus \{a\}$ for any fixed $a \in L$. First, it builds the partition $\{W_1,\ldots,W_p\}$ of $L \setminus \{a\}$. This takes $O(n)$ time. Then, T_1 and T_2 are preprocessed in $O(n)$ time so Lemma 11 can be applied. For each $j \in \{1,\ldots,p\}$, the algorithm then computes $\sigma^{T_1,\neg T_2}(W_j,b)$ and $\sigma^{T_1,\neg T_2}(\Lambda(T_3^{c_b}),b)$ for all $b \in W_j$. By Lemma 11, this step takes $O(\sum_{j=1}^{p}|W_j|) = O(n)$ time. (For every $b \in W_j$, to identify the child c_b of v_j such that $b \in \Lambda(T_3^{c_b})$ in $O(1)$ time, one can store the depths of all nodes in T_3 and use the level-ancestor data structure after $O(n)$ time extra preprocessing [3].) Finally, Lemma 10 is used to obtain $count_{r,f,f}^{T_1}(a,b)$ for every $b \in W_j$ and $j \in \{1,\ldots,p\}$ in $O(n)$ time. In total, the time complexity of `Compute_count_rff_T1` is $O(n)$. By running `Compute_count_rff_T1` once for each $a \in L$, we get $count_{r,f,f}^{T_1}(a,b)$ for all $a,b \in L$ in $O(n^2)$ total time.

Lemma 12. *For each $i \in \{1,2,3\}$, the values of $count_{r,f,f}^{T_i}(a,b)$ for all $a,b \in L$ can be computed in $O(n^2)$ total time.*

3.4 Determining if a Given Cluster Is a Strong Cluster When $k = 3$

Steps 3–5 of `R*_consensus_tree` in Section 1.3 need to determine which Apresjan clusters of $s_{\mathcal{R}_{maj}}$ are strong clusters of $\mathcal{R}_{maj}$. This subsection presents a method for doing so efficiently. Let $A \subseteq L$. For any $j \in \{1,2,3\}$, a leaf $x \in L \setminus A$ is called an *outsider in T_j* if x is not a descendant of u_A^j in T_j, where $u_A^j = lca^{T_j}(A)$. Define the following two disjoint subsets of $L \setminus A$: (i) $P_A =$ the set of all $x \in L \setminus A$ such that $lca^{T_j}(a,x)$ is a proper descendant of u_A^j for some $a \in A$ and some $j \in \{1,2,3\}$; and (ii) $Q_A =$ the set of all $x \in L \setminus A$ such that $lca^{T_j}(a,x) = u_A^j$ for all $a \in A$ and all $j \in \{1,2,3\}$. (If $|A| = 1$ then $P_A = Q_A = \emptyset$.) Also define an undirected graph $G_A = (A, E_A)$, whose edge set is $E_A = \big\{\{a,a'\} : lca^{T_j}(a,a') \text{ is a proper descendant of } u_A^j \text{ for at least one } j \in \{1,2,3\}\big\}$. Then we have:

Lemma 13. *For any $A \subseteq L$, A is a strong cluster of $\mathcal{R}_{maj}$ if and only if: (1) each $x \in P_A$ is an outsider in exactly two trees from $\{T_1,T_2,T_3\}$; and (2) if Q_A is nonempty, the graph G_A is a complete graph.*

```
Procedure Check_all_Apresjan_clusters
Input: A tree $\mathcal{A}$ of all Apresjan clusters of $s_{\mathcal{R}_{maj}}$
Output: A list of all the strong clusters of $\mathcal{R}_{maj}$
 1: for all nodes $v$ in $\mathcal{A}$ in bottom-up order do
 2:    Let $A$ be the Apresjan cluster of $s_{\mathcal{R}_{maj}}$ corresponding to $v$;
 3:    if $v$ is a leaf then
 4:       /* Without loss of generality, assume $A = \{a\}$ */
 5:       Set $u_A^1 = u_A^2 = u_A^3$ to be the leaf with label $a$ and let $G_A$ be a graph with
          a single vertex $a$. Let $\mathcal{B}_A^1 = \mathcal{B}_A^2 = \mathcal{B}_A^3 = \{A\}$;
 6:    else
 7:       Let $A_1, \ldots, A_m$ be the Apresjan clusters corresponding to the children of $v$
          and form $G_A$ by merging $G_{A_1}, \ldots, G_{A_m}$;
 8:       for $j = 1, 2, 3$ do
 9:          Update $u_A^j = lca^{T_j}(u_{A_1}^j, \ldots, u_{A_m}^j)$. Partition $A$ into a set of blocks $\mathcal{B}_A^j$
             such that each block $B \in \mathcal{B}_A^j$ contains all the elements of $A$ that appear
             in the same subtree attached to $u_A^j$;
10:          Compute $Z_B = \bigcup_{i=1}^{m}(\mathcal{B}_{A_i}^j | B)$ for every block $B \in \mathcal{B}_A^j$;
11:          for every block $B \in \mathcal{B}_A^j$ do
12:             Insert all edges $\{x, y\}$ into $G_A$ where $x \in X$, $y \in Y$ and where $X$ and
                $Y$ are two different sets in $Z_B$;
13:          end for
14:       end for
15:    end if
16:    If $A$ satisfies the condition in Lemma 13 then output $A$;
17: end for
```

Fig. 3. Procedure for finding all strong clusters of $\mathcal{R}_{maj}$

Procedure `Check_all_Apresjan_clusters` in Fig. 3 applies the condition in Lemma 13 to find all strong clusters of $\mathcal{R}_{maj}$. To avoid building each G_A-graph from scratch, it assumes that the Apresjan clusters are specified in the form of a tree $\mathcal{A}$, so that the information in the G_A-graphs can be reused as it goes upwards in $\mathcal{A}$. (As mentioned in Section 1.3, $\mathcal{A}$ can be obtained in $O(n^2)$ time [5].) The procedure builds the G_A-graphs for all Apresjan clusters A bottom-up, according to the given tree $\mathcal{A}$. Each G_A is represented as a set of edges. To simplify the construction, for $j = \{1, 2, 3\}$, the procedure maintains $u_A^j = lca^{T_j}(A)$. It also maintains $\mathcal{B}_A^j$, which is the partition of A such that each block $B \in \mathcal{B}_A^j$ contains all elements in A that appear in one subtree attached to the node u_A^j.

For any set $\mathcal{X}$ of subsets of L and any $L' \subseteq L$, let $\mathcal{X}|L' = \{X \in \mathcal{X} : X \subseteq L'\}$.

Lemma 14. *Procedure* `Check_all_Apresjan_clusters` *outputs all strong clusters of* $\mathcal{R}_{maj}$ *in* $O(n^2\alpha(n))$ *time, where* $\alpha(n)$ *is the inverse Ackermann function.*

4 Computing the R* Consensus Tree for Unbounded *k*

Section 4.1 computes $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ in $O(n^2 \log^k n)$ time. Section 4.2 checks which Apresjan clusters are strong clusters in $O(n^2 \log^{k+2} n)$ time.

4.1 Computing $s_{\mathcal{R}_{maj}}$ for Unbounded k

Here, we give a procedure that, for any fixed $a \in L$, computes $s_{\mathcal{R}_{maj}}(a, b)$ for all $b \in L \setminus \{a\}$ in $O(n \log^k n)$ time.

Let $occ(ab|w, T_{[i..j]})$ be the number of occurrences of $ab|w$ in $t(T_i), \ldots, t(T_j)$. Denote $s^{W,x,y,z}_{T_{[1..i]}}(a, b) = \big|\{w \in W : occ(ab|w, T_{[1..i]}) + x > \max\{occ(aw|b, T_{[1..i]}) + y,\ occ(bw|a, T_{[1..i]}) + z\}\}\big|$. For a fixed $a \in L$, our goal is to compute $s_{\mathcal{R}_{maj}}(a, b) = s^{L,0,0,0}_{T_{[1..k]}}(a, b)$ for all $b \in L \setminus \{a\}$. Note that in the formula for $s^{W,x,y,z}_{T_{[1..i]}}(a, b)$, W is not any arbitrary subset of L; we require, for all $w \in W$, that x, y and z are the number of occurrences of $ab|w$, $aw|b$ and $bw|a$, respectively, in $T_{i+1}, \ldots, T_k$. These three integers will be used to pass information during recursive calls.

In each tree $T_i \in \{T_1, \ldots, T_k\}$, any $w \in L\setminus\{a\}$ is represented by a pair $(d^{T_i}(w),\ \pi_i(w))$, where $d^{T_i}(w)$ is the distance in T_i from a to $lca^{T_i}(a, w)$, and $\pi_i(w) = j$, where w is a descendant of the jth child of $lca^{T_i}(a, w)$. The occurrence of a triplet in $t(T_i)$ is then given by (cf. Theorem 1 in [12] and Lemma 7 in [10]):

Lemma 15. *Let $b \in L \setminus \{a\}$. For any $w \in L \setminus \{a, b\}$ and $i \in \{1, \ldots, k\}$:*

1. *$ab|w \in t(T_i)$ if and only if $d^{T_i}(b) < d^{T_i}(w)$;*
2. *$aw|b \in t(T_i)$ if and only if $d^{T_i}(b) > d^{T_i}(w)$; and*
3. *$bw|a \in t(T_i)$ if and only if $d^{T_i}(b) = d^{T_i}(w)$ and $\pi_i(b) = \pi_i(w)$.*

We build a data structure $B_{W,k}$ in $O(|W| \log^k |W|)$ time that yields the value of $s^{W,x,y,z}_{T_{[1..k]}}(a, b)$ for any $b \in W \setminus \{a\}$ and any x, y, z in $O(\log^k |W|)$ time as follows.

For the base case $k = 1$, the data structure $B_{W,1}$ consists of a balanced binary search tree $BT(W, T_1)$ for all distinct $d^{T_1}(w)$-values, where $w \in W$. There may be multiple elements of W with the same $d^{T_1}(w)$-value. For each such node, we replace it by a balanced binary search tree for these multiple elements and index them using the keys $\pi_1(w)$. The additional nodes are called *yellow nodes*. The data structure $B_{W,1}$ can be constructed in $O(|W|)$ time.

Now we show how to compute $s^{W,x,y,z}_{T_{[1..1]}}(a, b)$ from $B_{W,1}$. For any $b \in W$, let P be the path from the root of $BT(W, T_1)$ to b. Since $BT(W, T_1)$ is balanced, P is of length $O(\log |W|)$. We partition the subtrees attached to P into four sets:

- W_{fan} is the set of subtrees attached to the yellow nodes of P where $\pi_1(b) \neq \pi_1(w)$ for all leaves w in the subtrees of W_{fan}.
- W_{mid} is the set of subtrees attached to the yellow nodes of P where $\pi_1(b) = \pi_1(w)$ for all leaves w in the subtrees of W_{mid}.
- W_{left} is the set of left subtrees attached to the non-yellow nodes of P.
- W_{right} is the set of right subtrees attached to the non-yellow nodes of P.

Note that $a|b|w \in t(T_1)$ for all $w \in \Lambda(S)$ and $S \in W_{fan}$. Similarly, $bw|a \in t(T_1)$ for all $w \in \Lambda(S)$ and $S \in W_{mid}$. Also, $aw|b \in t(T_1)$ for all $w \in \Lambda(S)$ and $S \in W_{left}$, and $ab|w \in t(T_1)$ for all $w \in \Lambda(S)$ and $S \in W_{right}$.

By the definitions and Lemma 15, $s^{W,x,y,z}_{T_{[1..1]}}(a, b) = A + B + C + D$ where:

- $A = \sum_{S \in W_{fan}} |\Lambda(S)|$ if $x > y$, $x > z$; and 0 otherwise.
- $B = \sum_{S \in W_{mid}} |\Lambda(S)|$ if $x > y$, $x > 1 + z$; and 0 otherwise.
- $C = \sum_{S \in W_{left}} |\Lambda(S)|$ if $x > 1 + y$, $x > z$; and 0 otherwise.
- $D = \sum_{S \in W_{right}} |\Lambda(S)|$ if $x + 1 > y$, $x + 1 > z$; and 0 otherwise.

```
Procedure counting_query
Input: Integer $i \in \{0, 1, \ldots, k\}$, $W \subseteq L$, integers $x$, $y$, $z$, leaf $b \in L \setminus \{a\}$.
Output: $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$
1: if $i = 0$ then
2:     if $x > y$ and $x > z$ then
3:         return $|W|$;
4:     else
5:         return 0;
6:     end if
7: else
8:     Let $P$ be the path from the root of $BT(W, T_i)$ to $b$;
9:     Compute the sets $W_{fan}, W_{mid}, W_{right}, W_{left}$ of subtrees attached to $P$;
10:    $A = \sum_{S \in W_{fan}}$ counting_query$(i-1, \Lambda(S), x, y, z, b)$;
11:    $B = \sum_{S \in W_{mid}}$ counting_query$(i-1, \Lambda(S), x, y, z+1, b)$;
12:    $C = \sum_{S \in W_{left}}$ counting_query$(i-1, \Lambda(S), x, y+1, z, b)$;
13:    $D = \sum_{S \in W_{right}}$ counting_query$(i-1, \Lambda(S), x+1, y, z, b)$;
14:    return $A + B + C + D$;
15: end if
```

Fig. 4. Procedure for computing $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$, assuming $B_{W,i}$ is available

There are $O(\log |W|)$ subtrees, so we can find $s_{T_{[1..1]}}^{W,x,y,z}(a, b)$ in $O(\log |W|)$ time.

Next, assume we can create a data structure $B_{W,k-1}$ from which $s_{T_{[1..k-1]}}^{W,x,y,z}(a, b)$ can be computed in $O(\log^{k-1} |W|)$ time. Then we build the data structure $B_{W,k}$, consisting of two parts, as follows. Firstly, similar to the case $k = 1$, we build a binary search tree $BT(W, T_k)$. Secondly, for every subtree S in $BT(W, T_k)$, we build the data structure $B_{\Lambda(S),k-1}$. The time required to build $B_{W,k}$ depends on the time needed for the two parts. For the first part, as shown above, $BT(W, T_k)$ can be constructed in $O(|W| \log |W|)$ time. For the second part, $\sum\{|\Lambda(S)| : S$ is a subtree of $BT(W, T_k)\} = O(|W| \log |W|)$. Since $B_{\Lambda(S),k-1}$ can be constructed in $O(|\Lambda(S)| \log^{k-1} |\Lambda(S)|)$ time, the second part takes $O(|W| \log^k |W|)$ time.

We now discuss how to use $B_{W,k}$ to compute $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$. For any $b \in W$, similar to the case $k = 1$, first find the path P from the root of $BT(W, T_k)$ to b. There are $O(\log |W|)$ subtrees attached to P. Partition them into the sets W_{fan}, W_{mid}, W_{left}, and W_{right} according to the same criteria as for $k = 1$ above. Then:

Lemma 16. *For any $b \in W$, it holds that $s_{T_{[1..k]}}^{W,x,y,z}(a, b) = A + B + C + D$, where $A = \sum_{S \in W_{fan}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y,z}(a, b)$, $B = \sum_{S \in W_{mid}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y,z+1}(a, b)$, $C = \sum_{S \in W_{left}} s_{T_{[1..k-1]}}^{\Lambda(S),x,y+1,z}(a, b)$, and $D = \sum_{S \in W_{right}} s_{T_{[1..k-1]}}^{\Lambda(S),x+1,y,z}(a, b)$.*

Fig. 4 lists the pseudocode of the procedure `counting_query` for computing $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$, given $B_{W,k}$. The next lemma bounds its running time.

Lemma 17. *Given the data structure $B_{W,k}$ for a fixed $a \in L$, for any $b \in L \setminus \{a\}$, `counting_query`(k, W, x, y, z, b) computes $s_{T_{[1..k]}}^{W,x,y,z}(a, b)$ in $O(\log^k n)$ time.*

4.2 Determining if a Given Cluster Is a Strong Cluster for Unbounded k

Let $\mathcal{A}$ be the tree of all Apresjan clusters. For any $A \subseteq L$ and $a, b \in A$ with $a \neq b$, define $s^{A}_{\mathcal{R}_{maj}}(a,b) = |\{w \in A : ab|w \in \mathcal{R}_{maj}\}|$. The following lemma allows us to verify if A is a strong cluster.

Lemma 18. *Let $A \subseteq L$. A is a strong cluster of R_{maj} if and only if $s_{\mathcal{R}_{maj}}(a,b) = |L \setminus A| + s^{A}_{\mathcal{R}_{maj}}(a,b)$ for all $a, b \in A$ with $a \neq b$.*

Observe that $s^{A}_{\mathcal{R}_{maj}}(a,b) = s^{A,0,0,0}_{T_{[1..k]}}(a,b)$, using the notation from Section 4.1. For any fixed $a \in L$, the next lemma gives a data structure for computing $s^{A}_{\mathcal{R}_{maj}}(a,b)$ in $O(\log^{k+1} n)$ time for any cluster $A \in \mathcal{A}$ and $b \in A \setminus \{a\}$.

Lemma 19. *For any $a \in L$, we can construct a data structure in $O(n \log^{k+1} n)$ time which enables us to compute $s^{A}_{\mathcal{R}_{maj}}(a,b) = s^{A,0,0,0}_{T_{[1..k]}}(a,b)$ in $O(\log^{k+1} n)$ time for any cluster $A \in \mathcal{A}$ that contains the element a and any $b \in A \setminus \{a\}$.*

Lemma 20. *If a node u in $\mathcal{A}$ satisfies $s_{\mathcal{R}_{maj}}(a,b) = |L \setminus \Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$, then, for every ancestor u' of u, $s_{\mathcal{R}_{maj}}(a,b) = |L \setminus \Lambda(\mathcal{A}^{u'})| + s^{\Lambda(\mathcal{A}^{u'})}_{\mathcal{R}_{maj}}(a,b)$ holds.*

Thus, $\mathcal{A}$ contains a node $u^{a,b}_{min}$ such that $s_{\mathcal{R}_{maj}}(a,b) = |L \setminus \Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$ for any ancestor u of $u^{a,b}_{min}$. In fact, $u^{a,b}_{min}$ can be found in $O(\log^{k+2} n)$ time:

Lemma 21. *Given the data structure in Lemma 19, $u^{a,b}_{min}$ for any $b \in L$ can be found in $O(\log^{k+2} n)$ time.*

Finally, we describe the procedure `Verify_strong_clusters` for checking which clusters in $\mathcal{A}$ are strong clusters. See Fig. 5 for the pseudocode. First, initialize $count(u) = 0$ for every node u in $\mathcal{A}$. Then, compute $u^{a,b}_{min}$ for all $a, b \in L$ using Lemma 21, and increase

```
Procedure Verify_strong_clusters
Input: A tree $\mathcal{A}$ of all Apresjan clusters of $s_{\mathcal{R}_{maj}}$
Output: A tree including all strong clusters of $\mathcal{R}_{maj}$
 1: Set $count(u) = 0$ for all nodes $u$ in $\mathcal{A}$;
 2: for $a, b \in L$ do
 3:     Find $u^{a,b}_{min}$ by Lemma 21 and set $count(u^{a,b}_{min}) = count(u^{a,b}_{min}) + 1$;
 4: end for
 5: Set $sum(u) = 0$ for all leaves $u$ in $\mathcal{A}$;
 6: for every internal node $u \in \mathcal{A}$ in bottom-up order do
 7:     Set $sum(u) = count(u) + \sum\{sum(c) : c$ is a child of $u$ in $\mathcal{A}\}$;
 8:     if $sum(u) < \binom{|\Lambda(\mathcal{A}^u)|}{2}$ then
 9:         Contract node $u$;      /* $\Lambda(\mathcal{A}^u)$ is not a strong cluster */
10:     end if
11: end for
12: return $\mathcal{A}$;
```

Fig. 5. Procedure for checking which Apresjan clusters are strong clusters

each $count(u^{a,b}_{min})$ by 1. Next, set $sum(u)$ to be the total sum of $count(v)$ for all descendants v of u in $\mathcal{A}$. By Lemma 22 below, if $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$ then $\Lambda(\mathcal{A}^u)$ is a strong cluster; otherwise, it is not. In case $\Lambda(\mathcal{A}^u)$ is not a strong cluster, contract u in $\mathcal{A}$ (that is, attach all children of u to the parent of u in $\mathcal{A}$ and remove the node u). By Lemmas 19 and 21, the running time of `Verify_strong_clusters` is $O(n^2 \log^{k+2} n)$.

Lemma 22. *For any node u in $\mathcal{A}$, $\Lambda(\mathcal{A}^u)$ is a strong cluster if and only if $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$.*

References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. Theoretical Computer Science **412**(48), 6634–6652 (2011)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
3. Bender, M.A., Farach-Colton, M.: The Level Ancestor Problem simplified. Theoretical Computer Science **321**(1), 5–12 (2004)
4. 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. American Mathematical Society (2003)
5. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. Advances in Applied Mathematics **27**(4), 705–732 (2001)
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 (2010)
7. Degnan, J.H., DeGiorgio, M., Bryant, D., Rosenberg, N.A.: Properties of consensus methods for inferring species trees from gene trees. Systematic Biology **58**(1), 35–54 (2009)
8. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates Inc., Sunderland (2004)
9. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM Journal on Computing **13**(2), 338–355 (1984)
10. Jansson, J., Sung, W.-K.: Constructing the R* consensus tree of two trees in subcubic time. Algorithmica **66**(2), 329–345 (2013)
11. Kannan, S., Warnow, T., Yooseph, S.: Computing the local consensus of trees. SIAM Journal on Computing **27**(6), 1695–1724 (1998)
12. Lee, C.-M., Hung, L.-J., Chang, M.-S., Shen, C.-B., Tang, C.-Y.: An improved algorithm for the maximum agreement subtree problem. Information Processing Letters **94**(5), 211–216 (2005)
13. Sung, W.-K.: Algorithms in Bioinformatics: A Practical Introduction. Chapman & Hall/CRC (2010)