Algorithmica (2016) 76:1224–1244
DOI 10.1007/s00453-016-0122-2

Faster Algorithms for Computing the R* Consensus Tree

Jesper Jansson[1] · **Wing-Kin Sung**[2,3] · **Hoa Vu**[4] · **Siu-Ming Yiu**[5]

Received: 30 January 2015 / Accepted: 13 January 2016 / Published online: 27 January 2016

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 $k = 2$ (Jansson and Sung in Algorithmica 66(2):329–345, 2013) and $O(kn^3)$ time when $k \geq 3$ (Bryant in Bioconsensus, volume 61 of DIMACS series in Discrete Mathematics and Theoretical Computer Science. American Mathe-

A preliminary version of this article appeared in Proceedings of the 25th International Symposium on Algorithms and Computation (ISAAC 2014), volume 8889 of Lecture Notes in Computer Science, pp. 414–425, Springer International Publishing Switzerland, 2014.

✉ Jesper Jansson
jj@kuicr.kyoto-u.ac.jp

Wing-Kin Sung
ksung@comp.nus.edu.sg

Hoa Vu
hoavu89@gmail.com

Siu-Ming Yiu
smyiu@cs.hku.hk

[1] Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan

[2] School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Singapore

[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

[5] Department of Computer Science, The University of Hong Kong, Pokfulam Road, Pokfulam, Hong Kong, China

matical Society, pp 163–184, 2003). 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.

Keywords Phylogenetic tree · R* consensus tree · Triplet · Strong cluster · Apresjan cluster

1 Introduction

Distinctly leaf-labeled, unordered trees known as *phylogenetic trees* are used by scientists to describe evolutionary history [11,18–20]. Given a set $\mathcal{S}$ of phylogenetic trees having the same leaf labels but different branching structures, a single phylogenetic tree that summarizes all the trees in $\mathcal{S}$ according to some well-defined rule is called a *consensus tree* [4,11,20]. Consensus trees are useful when dealing with unreliable data. For example, to infer an accurate phylogenetic tree for a 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. In general, maximum likelihood-based methods may construct a set of almost-equally-optimal trees which can then be summarized in a consensus tree.

A number of different consensus trees have been defined and studied in the literature; see [4], Chapter 30 in [11], or Chapter 8.4 in [20] for some surveys. This paper deals with one particular consensus tree called the *R* consensus tree* [4], formally defined in Sect. 1.1 below. Simply put, it is a tree with the same leaf label set as the input trees and as many internal nodes as possible and whose *resolved triplets* (embedded binary subtrees on three leaves) all occur frequently in the input trees.

One advantage of the R* consensus tree is that it provides a statistically consistent estimator of the species tree topology when combining gene trees [9]. Furthermore, under the multispecies coalescent model, the R* consensus tree is asymptotically guaranteed to be fully resolved [9]. Also, as shown in [4], it is always a refinement of the popular *majority rule consensus tree* [17], which means that besides all the clusters of leaf labels present in the majority rule consensus tree, the R* consensus tree may contain additional informative branching structure. From a theoretical point of view, the R* consensus tree is also interesting because it generalizes the *RV-III tree* of [15] to more than two input trees [4]. On the negative side, the existing algorithms for building the R* consensus tree [4,14,15] 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 used in many contexts [11], 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

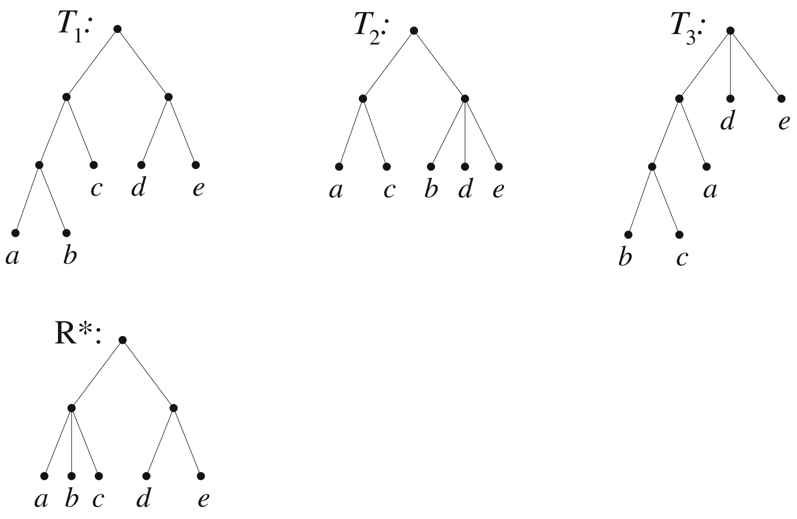

Fig. 1 An example. Let $\mathcal{S} = \{T_1, T_2, T_3\}$ with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e\}$ as above. Then $\mathcal{R}_{maj} = \{ab|d,\ ab|e,\ ac|d,\ ac|e,\ de|a,\ bc|d,\ bc|e,\ de|b,\ de|c\}$ and the R* consensus tree τ of $\mathcal{S}$ is the tree on the bottom. In this example, $r(\tau) = \mathcal{R}_{maj}$

u. For any $X \subseteq V(T)$, $lca^T(X)$ is the lowest common ancestor in T of the nodes in X, and 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 that 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 given set of three leaves $\{x, y, z\}$.

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 $lca^T(x, y) = lca^T(x, z) = lca^T(y, z)$. The resolved triplet $xy|z$ 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 $r(T) = \{T||_{\{x,y,z\}} \colon \{x, y, z\} \subseteq \Lambda(T)$ and $T||_{\{x,y,z\}}$ is a resolved triplet$\}$ and $t(T) = \{T||_{\{x,y,z\}} \colon \{x, y, z\} \subseteq \Lambda(T)\}$.

Next, let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be a given set of trees with $\Lambda(T_1) = \cdots = \Lambda(T_k) = L$. 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 $\{ab|c \colon a, b, c \in L$ 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 $L = \bigcup_{t \in \mathcal{R}} \Lambda(t)$ 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 $x \in L \backslash A$. Furthermore, L as well as every singleton set of L is also defined to be a strong cluster of $\mathcal{R}$. Strong clusters provide a useful alternative characterization of R* consensus trees, stated in the last part of the next lemma:

Lemma 1.1 [4,14] *The R* consensus tree always exists, is unique, and includes every strong cluster of* $\mathcal{R}_{maj}$ *and no other clusters.*

Lastly, the following definitions will be used in our algorithms for constructing the R* consensus tree. Suppose that $\mathcal{R}$ is a set of triplets as in the paragraph before Lemma 1.1. For each $a, b \in L$ with $a \neq b$, define $s_{\mathcal{R}}(a, b) = \left|\{w : ab|w \in \mathcal{R}\}\right|$ 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 all $x \in L \backslash A$.

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 the sets $r(T_i)$ for all $T_i \in \mathcal{S}$ in $O(kn^3)$ time, then construct the set $\mathcal{R}_{maj}$ 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 RV-III tree of two input trees in $O(n^3)$ time [15] 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 [14] that the R* consensus tree can in fact be computed in $O(n^2\sqrt{\log n})$ $(= o(n^3))$ time. The algorithm from [14] is reviewed in Sect. 1.3.

Note that the R* consensus tree τ satisfies $r(\tau) \subseteq \mathcal{R}_{maj}$ by definition; in other words, it is not allowed to introduce new resolved triplets that were not already in $\mathcal{R}_{maj}$. Relaxing this requirement leads to other types of consensus trees such as the *local consensus trees* studied in [4,13,15] and the *triplec consensus tree* [10].

1.3 Overview and Summary of New Results

To compute the R* consensus tree without constructing $\mathcal{R}_{maj}$, the algorithm in [14] for $k = 2$ and the new algorithms in this paper follow the same basic strategy, summarized as Algorithm `R*_consensus_tree` in Fig. 2. Before explaining the details, some simple observations are needed. Since every strong cluster of $\mathcal{R}$ is an Apresjan cluster of $s_{\mathcal{R}}$ [4,14], one can 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 $O(n)$ 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.

```
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`

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 1.1, this tree is the R* consensus tree.

The time complexity of Algorithm `R*_consensus_tree` depends on various factors. As shown in [14], 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 the time complexity of Step 1 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 [14]. However, Steps 1 and 3–5 have to be modified; for example, the conditions in Lemma 13 in [14] for checking if a given cluster is a strong cluster of $\mathcal{R}_{maj}$ only work if $k = 2$.[1] As for Step 1, Sects. 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 Sect. 4.1 in $O(n^2 \log^k n)$ time for unbounded k. For Steps 3–5, Sect. 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 Sect. 4.2 gives an $O(n^2 \log^{k+2} n)$-time solution for unbounded k.

In summary, we obtain the following new results:

Theorem 1.2 *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.*

[1] An example with $k = 3$ for which Lemma 13 in [14] fails is: $T_1 = (((a, b), c, d), (e, f));$, $T_2 = (((b, f), a, c), (d, e));$, $T_3 = (((a, c), b, e), (d, f));$ (here, trees are expressed using Newick notation; see http://evolution.genetics.washington.edu/phylip/newicktree.html). Then $\mathcal{R}_{maj} = \{ab|d, ab|e, ab|f, ac|d, ac|e, ac|f, bc|d, bc|e, bc|f\}$, and $A = \{a, b, c\}$ is a strong cluster of $\mathcal{R}_{maj}$ by definition. However, condition (1) in Lemma 13 of [14] does not hold for $i = 2$ as the subtree $U = (b, f);$ of T_2 rooted at a child of $lca^{T_2}(A)$ does not satisfy $\Lambda(U) \subseteq A$ or $\Lambda(U) \subseteq L \backslash A$.

Thus, if $k < \frac{\log n}{(\log \log n)^{1+\epsilon}}$ for some $\epsilon > 0$, the time complexity of computing the R* consensus tree is subcubic in n. We remark that in case k is large compared to n (for example, if resampling techniques are used to generate the input trees), the simple $O(kn^3)$-time method referred to in Sect. 1.2 is faster than the above. On the other hand, in applications where k may be much smaller than n (e.g., if the input trees are derived from alternative data sets or collected manually from the literature), the new algorithms will be useful.

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 Sect. 1.3 (and hence the algorithm's overall running time) to $O(n^2)$.

Recall that the function $s_{\mathcal{R}_{maj}}$ is defined by $s_{\mathcal{R}_{maj}}(a, b) = \left|\{w : ab|w \in \mathcal{R}_{maj}\}\right|$ 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, a resolved triplet $ab|w$ belongs to $\mathcal{R}_{maj}$ 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. Corollary 1 in [14] states that $s_{\mathcal{R}_{maj}}(a, b) = count_{r,r}(a, b) + count_{r,f}(a, b) + count_{f,r}(a, b)$ for every $a, b \in L$ with $a \neq b$, where:

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

Section 4 in [14] showed 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 by computing $count_{r,r}(a, b)$ for all $a, b \in L$ in $O(n^2)$ total time.

Lemma 2.1 *For every $a, b \in L$, it holds that:*

$$count_{r,r}(a, b) = |L| - |\Lambda\left(T_1^{lca(a,b)}\right)| - |\Lambda\left(T_2^{lca(a,b)}\right)| + |\Lambda\left(T_1^{lca(a,b)}\right) \cap \Lambda\left(T_2^{lca(a,b)}\right)|.$$

Proof It is easy to see that $count_{r,r}(a, b) = \left|\{w \in L \backslash \{a, b\} : ab|w \in t(T_1) \text{ and } ab|w \in t(T_2)\}\right| = |(L \backslash \Lambda(T_1^{lca(a,b)})) \cap (L \backslash \Lambda(T_2^{lca(a,b)}))| = |L| - |\Lambda(T_1^{lca(a,b)}) \cup \Lambda(T_2^{lca(a,b)})|$. By the inclusion-exclusion principle, the latter expression is equal to $|L| - |\Lambda(T_1^{lca(a,b)})| - |\Lambda(T_2^{lca(a,b)})| + |\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)})|$. □

Lemma 2.2 *The values of $count_{r,r}(a, b)$ 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 [2,12]. Next, for each $a, b \in L$, obtain $count_{r,r}(a, b)$ in $O(1)$ time by applying the formula in Lemma 2.1. The total running time is $O(n^2)$. □

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

We now focus on the case of three input trees. Sects. 3.1–3.3 and 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 that $\mathcal{S} = \{T_1, T_2, T_3\}$. For every $ab|w \in \mathcal{R}_{maj}$, there are three possibilities:

Lemma 3.1 *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 help us count the triplets covered by the different cases in Lemma 3.1, we define:

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

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

Lemma 3.2 *For every $a, b \in L$ with $a \neq b$,*

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

Proof For any $\{i, j, k\} = \{1, 2, 3\}$ with $i < j$, define $W^{T_i,T_j}(a,b) = \{w \in L\backslash\{a,b\} : ab|w \in t(T_i) \cap t(T_j)$ and $ab|w \notin t(T_k)\}$. Then $count_{r,r,*}^{T_i,T_j}(a,b) = |W^{T_i,T_j}(a,b)| + count_{r,r,r}(a,b)$. By summing over all pairs of trees, we get $|W^{T_1,T_2}(a,b)| + |W^{T_1,T_3}(a,b)| + |W^{T_2,T_3}(a,b)| = \sum_{1 \le i<j \le 3} count_{r,r,*}^{T_i,T_j}(a,b) - 3count_{r,r,r}(a,b)$.

Next, by Lemma 3.1,

$$\begin{aligned} s_{\mathcal{R}_{maj}}(a,b) &= count_{r,r,r}(a,b) + |W^{T_1,T_2}(a,b)| + |W^{T_1,T_3}(a,b)| + |W^{T_2,T_3}(a,b)| \\ &\quad + count^{T_1}_{r,f,f}(a,b) + count^{T_2}_{r,f,f}(a,b) + count^{T_3}_{r,f,f}(a,b) \\ &= count_{r,r,r}(a,b) + \sum_{1\le i<j\le 3} count^{T_i,T_j}_{r,r,*}(a,b) - 3count_{r,r,r}(a,b) \\ &\quad + \sum_{i=1}^{3} count^{T_i}_{r,f,f}(a,b) \\ &= \sum_{i=1}^{3} count^{T_i}_{r,f,f}(a,b) + \sum_{1\le i<j\le 3} count^{T_i,T_j}_{r,r,*}(a,b) - 2count_{r,r,r}(a,b). \end{aligned}$$

□

For each pair $i, j \in \{1,2,3\}$ with $i < j$, the values of $count^{T_i,T_j}_{r,r,*}(a,b)$ for all $a,b \in L$ can be obtained in $O(n^2)$ time by the method from Lemma 2.2 in Sect. 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 3.6 in Sect. 3.2) and $count^{T_i}_{r,f,f}(a,b)$ for all $a,b \in L$ for each $i \in \{1,2,3\}$ in $O(n^2)$ time (Lemma 3.9 in Sect. 3.3). Then, we can apply the formula in Lemma 3.2 to get each value of $s_{\mathcal{R}_{maj}}(a,b)$ in $O(1)$ time. In summary:

Lemma 3.3 *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 $count_{r,r,r}$

First, rewrite $count_{r,r,r}(a,b)$ in a way analogous to the expression in Lemma 2.1:

Lemma 3.4 *For every $a,b \in L$, it holds that:*

$$\begin{aligned} count_{r,r,r}(a,b) = |L| - \sum_{i=1}^{3}\left|\Lambda\left(T_i^{lca(a,b)}\right)\right| + \sum_{1\le i<j\le 3}\left|\Lambda\left(T_i^{lca(a,b)}\right) \cap \Lambda\left(T_j^{lca(a,b)}\right)\right| \\ - \left|\Lambda\left(T_1^{lca(a,b)}\right) \cap \Lambda\left(T_2^{lca(a,b)}\right) \cap \Lambda\left(T_3^{lca(a,b)}\right)\right|. \end{aligned}$$

Proof By definition, $count_{r,r,r}(a,b) = \left|\{w \in L\backslash\{a,b\}: ab|w \in t(T_1),\ ab|w \in t(T_2),\ ab|w \in t(T_3)\}\right| = |(L\backslash\Lambda(T_1^{lca(a,b)}))\cap(L\backslash\Lambda(T_2^{lca(a,b)}))\cap(L\backslash\Lambda(T_3^{lca(a,b)}))| = |L| - |\Lambda(T_1^{lca(a,b)}) \cup \Lambda(T_2^{lca(a,b)}) \cup \Lambda(T_3^{lca(a,b)})|$. The inclusion-exclusion principle gives $|\Lambda(T_1^{lca(a,b)}) \cup \Lambda(T_2^{lca(a,b)}) \cup \Lambda(T_3^{lca(a,b)})| = \sum_{i=1}^{3} |\Lambda(T_i^{lca(a,b)})| - \sum_{1\le i<j\le 3} |\Lambda(T_i^{lca(a,b)}) \cap \Lambda(T_j^{lca(a,b)})| + |\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$. The lemma follows. □

Lemma 3.5 *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 every $w \in L\setminus\{a\}$ and $i \in \{1, 2, 3\}$, let $d^{T_i}(w)$ be the distance in T_i from the leaf a to the node $lca^{T_i}(a, w)$. Observe that 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 $d^{T_i}(w) \leq d^{T_i}(b)$. Thus, for any $b \in L\setminus\{a\}$, we have $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})| = |\{w \in L\setminus\{a, b\}: d^{T_1}(w) \leq d^{T_1}(b),\ d^{T_2}(w) \leq d^{T_2}(b),\ \text{and } d^{T_3}(w) \leq d^{T_3}(b)\}|$.

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

Lemma 3.6 *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.*

Proof As in the proof of Lemma 2.2, use $O(n^2)$ time to compute and store all values of $|\Lambda(T_i^u)|$, where $u \in V(T_i)$ and $i \in \{1, 2, 3\}$, and all values of $|\Lambda(T_i^u) \cap \Lambda(T_j^v)|$, where $u \in V(T_i)$, $v \in V(T_j)$ and $1 \leq i < j \leq 3$. Also, preprocess T_i for each $i \in \{1, 2, 3\}$ in $O(n)$ time so that *lca*-queries can be answered in $O(1)$ time [2,12]. Then, for each $a \in L$, apply the method in Lemma 3.5 to compute and store $|\Lambda(T_1^{lca(a,b)}) \cap \Lambda(T_2^{lca(a,b)}) \cap \Lambda(T_3^{lca(a,b)})|$ for all $b \in L\setminus\{a\}$; this takes $O(n \log^{4/3} n) \cdot O(n) = O(n^2 \log^{4/3} n)$ time.

Finally, for each $a, b \in L$, compute $count_{r,r,r}(a, b)$ in $O(1)$ time according to Lemma 3.4. The total time complexity is $O(n^2 \log^{4/3} n)$. □

3.3 Computing $count^{T_i}_{r,f,f}$

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

Suppose that $a \in L$ is fixed. Let $\langle v_0 = a,\ v_1,\ \ldots,\ v_p\rangle$ 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 $L\setminus\{a\}$. Also, for any $b \in L\setminus\{a\}$ and any $w \in L\setminus\{a, b\}$, $a|b|w$ is a fan triplet in $t(T_3)$ if and only if w belongs to the same W_j-set as b while b and w belong to different subtrees rooted at the children of v_j; see Fig. 3. 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 3.7 explains how to use $\sigma^{T_1,\neg T_2}(S, b)$ to compute $count^{T_1}_{r,f,f}(a, b)$.

Lemma 3.7 *For any W_j, where $j \in \{1, \ldots, p\}$, 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 $count^{T_1}_{r,f,f}(a, b) = \sigma^{T_1,\neg T_2}(W_j, b) - \sigma^{T_1,\neg T_2}(\Lambda(T_3^{c_b}), b)$.*

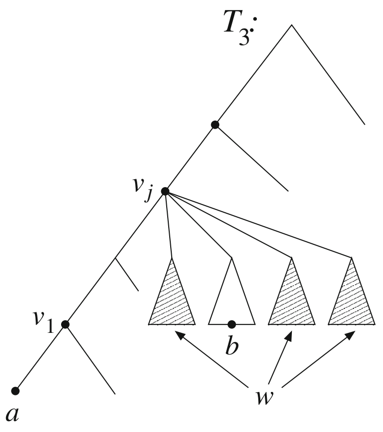

Fig. 3 Suppose $b \in W_j$. Then $a|b|w \in t(T_3)$ for every leaf w in the shaded parts of T_3.

Proof

$$\begin{aligned} count^{T_1}_{r,f,f}(a,b) &= |\{w \in L\backslash\{a,b\}: ab|w \in t(T_1),\ a|b|w \in t(T_2),\ a|b|w \in t(T_3)\}| \\ &= \left|\left\{w \in \left(W_j \backslash \Lambda(T_3^{c_b})\right): ab|w \in t(T_1) \text{ and } a|b|w \in t(T_2)\right\}\right| \\ &= \left|\left\{w \in W_j: ab|w \in t(T_1) \text{ and } a|b|w \in t(T_2)\right\}\right| \\ &\quad - \left|\left\{w \in \Lambda(T_3^{c_b}): ab|w \in t(T_1) \text{ and } a|b|w \in t(T_2)\right\}\right| \\ &= \sigma^{T_1,\neg T_2}(W_j, b) - \sigma^{T_1,\neg T_2}(\Lambda(T_3^{c_b}), b) \qquad \text{(by definition)} \end{aligned}$$ □

To compute $\sigma^{T_1,\neg T_2}(S,b)$ efficiently, we rely on the next lemma.

Lemma 3.8 *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.*

Proof Use the method in Section 8 of [7] to preprocess each T_j, $j \in \{1,2\}$, in $O(n)$ time so that the subtree of T_j induced by any $L' \subseteq L$ can be retrieved in $O(|L'|)$ time. Then, given any $S \subseteq L$, apply Algorithm `Compute_count_rf` in Section 4.3 of [14] (which computes $count_{r,f}(a,b) = \big|\{w \in S: ab|w$ is consistent with the first tree and $a|b|w$ is consistent with the other tree$\}\big|$ for all b), to the subtrees of T_1 and T_2 induced by $S \cup \{a\}$ to get $\sigma^{T_1,\neg T_2}(S,b)$ for all $b \in S$ in $O(|S|)$ time. □

This suggests the algorithm named `Compute_count_rff_T1` in Fig. 4 for computing $count^{T_1}_{r,f,f}(a,b)$ for all $b \in L\backslash\{a\}$ for any fixed $a \in L$. First, it builds the partition $\{W_1,\ldots,W_p\}$ of $L\backslash\{a\}$ as defined above. This takes $O(n)$ time. Then, T_1 and T_2 are preprocessed in $O(n)$ time so that Lemma 3.8 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 3.8, this step can be done in $O(\sum_{j=1}^{p}|W_j|) = O(n)$ 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, which requires an additional $O(n)$ time preprocessing [3]). Finally, Lemma 3.7 is used to obtain $count^{T_1}_{r,f,f}(a,b)$ for every $b \in W_j$ and $j \in \{1,\ldots,p\}$ in $O(n)$ time.

```
Algorithm Compute_count_rff_T1
Input: $a \in L$
Output: $count^{T_1}_{r,f,f}(a,b)$ for all $b \in L \setminus \{a\}$
1: Partition $L \setminus \{a\}$ into $\{W_1, \ldots, W_p\}$ by traversing $T_3$ from $a$ to the root of $T_3$ and
   setting, for every $j \in \{1, \ldots, p\}$, $W_j = \Lambda(T_3^{v_j}) \setminus \Lambda(T_3^{v_{j-1}})$, where $v_j$ is the $j$th
   internal node along the path and $v_0 = a$;
2: Preprocess $T_1$ and $T_2$ so that Lemma 3.8 can be applied;
3: for $j = 1, \ldots, p$ do
4:     Compute $\sigma^{T_1, \neg T_2}(W_j, b)$ and $\sigma^{T_1, \neg T_2}(\Lambda(T_3^{c_b}), b)$ for all $b \in W_j$;
5:     Compute $count^{T_1}_{r,f,f}(a,b)$ for all $b \in W_j$ using Lemma 3.7;
6: end for
```

Fig. 4 Algorithm `Compute_count_rff_T1`

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 $count^{T_1}_{r,f,f}(a,b)$ for all $a, b \in L$ in $O(n^2)$ total time. The functions $count^{T_2}_{r,f,f}$ and $count^{T_3}_{r,f,f}$ are handled similarly.

Lemma 3.9 *For each $i \in \{1, 2, 3\}$, the values of $count^{T_i}_{r,f,f}(a,b)$ for all $a, b \in L$ can be computed in $O(n^2)$ total time.*

3.4 Determining Which Clusters are Strong Clusters for $k = 3$

Steps 3–5 of Algorithm `R*_consensus_tree` in Sect. 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.

We first give some definitions. Let $A \subseteq L$. For each $j \in \{1, 2, 3\}$, write $u^j_A = lca^{T_j}(A)$. Define the following two disjoint subsets of $L \backslash A$:

(i) P_A = the set of all $x \in L \backslash A$ such that $lca^{T_j}(a, x)$ is a proper descendant of u^j_A for some $a \in A$ and some $j \in \{1, 2, 3\}$; and
(ii) Q_A = the set of all $x \in L \backslash A$ such that $lca^{T_j}(a, x) = u^j_A$ for all $a \in A$ and all $j \in \{1, 2, 3\}$.

If $|A| = 1$ then $P_A = Q_A = \emptyset$. A leaf $x \in L \backslash A$ is called an *outsider in T_j*, where $j \in \{1, 2, 3\}$, if x is not a descendant of u^j_A in T_j. See Fig. 5 for some examples.

Also, for any $A \subseteq L$, define an undirected graph $G_A = (A, E_A)$ whose edge set is given by $E_A = \big\{\{a, a'\}: lca^{T_j}(a, a')$ is a proper descendant of u^j_A for at least one $j \in \{1, 2, 3\}\big\}$. Then we have:

Lemma 3.10 *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.*

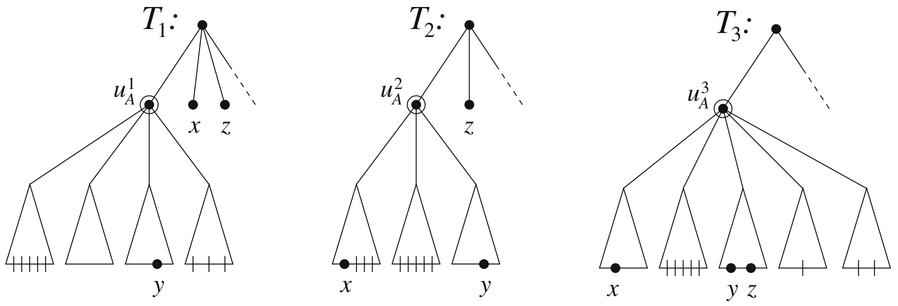

Fig. 5 Illustrating the definitions of P_A and Q_A. Suppose that A consists of the leaves marked by short vertical lines in T_1, T_2, and T_3. Then $x \in P_A$, $y \in Q_A$, and $z \notin P_A \cup Q_A$. Also, x is an outsider in T_1 while z is an outsider in T_1 and T_2

Proof ($\rightarrow$) If A is a strong cluster of $\mathcal{R}_{maj}$ then $aa'|x \in \mathcal{R}_{maj}$ for all $a, a' \in A$, $a \neq a'$, and $x \in L \setminus A$. For every $x \in P_A$, by definition, there exists a tree T_i with $i \in \{1, 2, 3\}$ and some $b \in A$ such that $lca^{T_i}(x, b)$ is a proper descendant of u^i_A. Let B be the subset of A such that $lca^{T_i}(x, b)$ is a proper descendant of u^i_A for all $b \in B$. Then, for every $a \in A \setminus B$ and $b \in B$, it immediately follows that $bx|a \in t(T_i)$. For any $a \in A \setminus B$, $b \in B$, in order for $ab|x \in \mathcal{R}_{maj}$ to hold, we therefore need $ab|x \in t(T_j)$ for all $j \in \{1, 2, 3\} \setminus \{i\}$, $a \in A \setminus B$, $b \in B$. We claim that this requires x to be an outsider in T_j for both $j \in \{1, 2, 3\} \setminus \{i\}$. For the sake of obtaining a contradiction, assume x is not an outsider in T_j. To ensure $ab|x \in t(T_j)$, $lca^{T_j}(a, b)$ must be a proper descendant of u^j_A for all $a \in A \setminus B$ and $b \in B$. This means that $lca^{T_j}((A \setminus B) \cup B) = lca^{T_j}(A)$ is a proper descendant of u^j_A, which is impossible because $u^j_A = lca^{T_j}(A)$. Hence, $x \in P_A$ is an outsider in two trees.

When Q_A is nonempty, there exists some $x \in Q_A$. For all $a, a' \in A$ with $a \neq a'$, since $aa'|x \in \mathcal{R}_{maj}$, the node $lca^{T_j}(a, a')$ is a proper descendant of u^j_A for at least one $j \in \{1, 2, 3\}$, so $\{a, a'\} \in E_A$. Hence, G_A is a complete graph.

($\leftarrow$) First consider any $x \in P_A$. Since x is an outsider in two trees, for every $a, a' \in A$ with $a \neq a'$, the resolved triplet $aa'|x$ occurs twice in the sets $t(T_1)$, $t(T_2)$, and $t(T_3)$. Hence, $aa'|x \in \mathcal{R}_{maj}$.

Next, suppose that $Q_A \neq \emptyset$. For any $x \in Q_A$, x is not an outsider in any of the trees and thus either $a|a'|x \in t(T_j)$ or $aa'|x \in t(T_j)$ for each $j \in \{1, 2, 3\}$ and $a, a' \in A$, $a \neq a'$. Since G_A is a complete graph, it holds for all $a, a' \in A$ with $a \neq a'$ that $lca^{T_i}(a, a')$ is a proper descendant of u^i_A for at least one $i \in \{1, 2, 3\}$, which implies that $aa'|x \in t(T_i)$. Thus, for all $a, a' \in A$, $aa'|x \in \mathcal{R}_{maj}$.

Finally, consider any $x \in (((L \setminus A) \setminus P_A) \setminus Q_A)$. Since $x \notin P_A$, $lca^{T_j}(x, a)$ is not a proper descendant of u^j_A for any $a \in A$ and $j \in \{1, 2, 3\}$. Thus, for any $a, a' \in A$ with $a \neq a'$, either $aa'|x \in t(T_j)$ or $a|a'|x \in t(T_j)$ for each $j \in \{1, 2, 3\}$. In addition, since $x \notin Q_A$, we have $a|a'|x \notin t(T_j)$ for every $j \in \{1, 2, 3\}$, which shows that $aa'|x \in \mathcal{R}_{maj}$ for all $a, a' \in A$.

In conclusion, $aa'|x \in \mathcal{R}_{maj}$ for all $a, a' \in A$, $a \neq a'$, and $x \in L \setminus A$, so A is a strong cluster. □

```
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 $a$;
6:       Let $\mathcal{B}_A^1 = \mathcal{B}_A^2 = \mathcal{B}_A^3 = \{A\}$;
7:    else
8:       /* $v$ is an internal node */
9:       Let $A_1, \ldots, A_m$ be the Apresjan clusters corresponding to the children of $v$
         and form $G_A$ by merging $G_{A_1}, \ldots, G_{A_m}$;
10:      for $j = 1, 2, 3$ do
11:         Update $u_A^j = lca^{T_j}(u_{A_1}^j, \ldots, u_{A_m}^j)$;
12:         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 $u_A^j$;
13:         Compute $Z_B = \bigcup_{i=1}^{m}(\mathcal{B}_{A_i}^j | B)$ for every block $B \in \mathcal{B}_A^j$;
14:         for every block $B \in \mathcal{B}_A^j$ do
15:            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 $Z_B$;
16:         end for
17:      end for
18:   end if
19:   If $A$ satisfies the condition in Lemma 3.10 then output $A$;
20: end for
```

Fig. 6 Procedure for finding all strong clusters of $\mathcal{R}_{maj}$

Procedure `Check_all_Apresjan_clusters` in Fig. 6 applies the condition in Lemma 3.10 to find all strong clusters of $\mathcal{R}_{maj}$. To avoid building each G_A-graph from scratch, the procedure 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 Sect. 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 in bottom-up order, 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 $L' \subseteq L$, define $\mathcal{X}|L' = \{X \in \mathcal{X} : X \subseteq L'\}$.

Lemma 3.11 *The 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 of n.*

Proof For each leaf in $\mathcal{A}$ (corresponding to an Apresjan cluster of the form $A = \{a\}$), G_A has no edges. For $j \in \{1, 2, 3\}$, u_A^j is the leaf with label a and $\mathcal{B}_A^j$ consists of exactly one set A. This shows that every leaf in $\mathcal{A}$ can be processed in $O(1)$ time.

For each internal node v in $\mathcal{A}$ (corresponding to a non-singleton Apresjan cluster A), the procedure needs to compute u_A^j and $\mathcal{B}_A^j$ for $j \in \{1, 2, 3\}$ and the graph G_A. Let $A_1, \ldots, A_m$ be the Apresjan clusters corresponding to the children of v in $\mathcal{A}$. For $j \in \{1, 2, 3\}$, $u_A^j = lca^{T_j}(A)$ can be computed in $O(|A|)$ time [2,12] after linear-time preprocessing. Similarly, by enumerating all $a \in A$ and checking which subtree of u_A^j that a belongs to by the level-ancestor data structure [3], $\mathcal{B}_A^j$ for $j \in \{1, 2, 3\}$ can be computed in $O(|A|)$ time. The graph G_A should contain all edges $\{x, y\}$ where $x, y \in B$ and $B \in \mathcal{B}_A^j$. To construct G_A, include: (1) all edges in $G_{A_1}, \ldots, G_{A_m}$; and (2) for $j \in \{1, 2, 3\}$, for all $B \in \mathcal{B}_A^j$, all edges $\{x, y\}$ where $x \in X$, $y \in Y$, $X \neq Y$, and $X, Y \in \bigcup_{i=1}^{m}(\mathcal{B}_{A_i}^j | B)$.

Finally, we analyze the time complexity of `Check_all_Apresjan_clusters`. For each Apresjan cluster A, the partition $\mathcal{B}_A^j$ and the node u_A^j can be obtained in $O(|A|) = O(n)$ time. To build the G_A-graphs, observe that we will insert $O(n^2)$ new edges in total and do a merge $O(n)$ times. Using the union-and-find data structure (see, e.g., [8]), building all G_A-graphs takes $O(n^2\alpha(n))$ time. The total running time becomes $O(n^2\alpha(n))$. □

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

Section 4.1 shows how to compute $s_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in L$ in $O(n^2 \log^k n)$ time. Section 4.2 shows how to check 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 $b \in L \backslash \{a\}$ 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 $s_{T_{[1..i]}}^{W,x,y,z}(a, b) = \big|\{w \in W : occ(ab|w, T_{[1..i]}) + x > \max\{occ(aw|b, T_{[1..i]}) + y,\ occ(bw|a, T_{[1..i]}) + z\}\}\big|$. For a fixed $a \in L$, our goal is to compute $s_{\mathcal{R}_{maj}}(a, b) = s_{T_{[1..k]}}^{L,0,0,0}(a, b)$ for all $b \in L \backslash \{a\}$. In the formula for $s_{T_{[1..i]}}^{W,x,y,z}(a, b)$, 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 $w \in L \backslash \{a\}$ 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 [16] and Lemma 7 in [14]):

Lemma 4.1 *Let $b \in L \backslash \{a\}$. For any $w \in L \backslash \{a, b\}$ 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 $\pi_i(b) = \pi_i(w)$.*

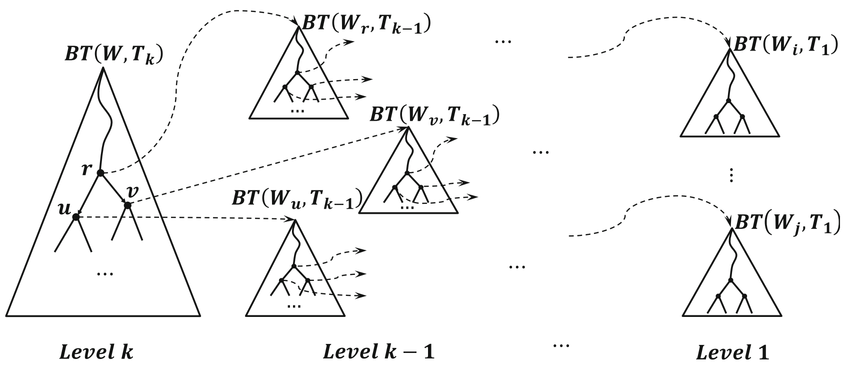

Fig. 7 Sketch of the data structure $B_{W,k}$. Each node u in $BT(W_i, T_i)$ at level i points to $BT(W_{i-1}, T_{i-1})$ at level $i-1$, where $W_{i-1} = \Lambda(BT(W_i, T_i)^u)$

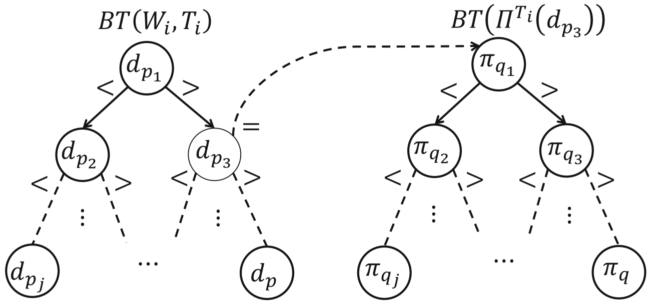

Fig. 8 At each level i, $BT(W_i, T_i)$ is a balanced binary search tree for $\{d_1, \ldots, d_p\}$, where $W = \{w_1, \ldots, w_{|W|}\} \subseteq L$ and $\{d_1, \ldots, d_p\}$ are the distinct values in the set $\{d^{T_i}(w_1), \ldots, d^{T_i}(w_{|W|})\}$. Every node v in $BT(W_i, T_i)$ points to a balanced binary search tree for $\{\pi_1, \ldots, \pi_q\}$, denoted by $BT(\pi^{T_i}(d^{T_i}(v)))$ above, where $\pi^{T_i}(d) = \{w \in W : d^{T_i}(w) = d\}$ and where $\{\pi_1, \ldots, \pi_q\}$ are the distinct values in the set $\{\pi_i(w) : w \in \pi^{T_i}(d)\}$. Every node u in $BT(\pi^{T_i}(d^{T_i}(v)))$ contains a set $\{w \in W : d^{T_i}(w) = d,\ \pi_i(w) = \pi\}$

We build a recursive 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 $b \in W \backslash \{a\}$ and any x, y, z in $O(\log^k |W|)$ time. Figures 7 and 8 outline the structure of $B_{W,k}$ graphically. The details of $B_{W,k}$ are described next.

4.1.1 The Case $k = 1$

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 leaf $b \in W$, let P be the path from the root of $BT(W, T_1)$ to b. Since $BT(W, T_1)$ is a balanced tree, 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 $S \in W_{left}$.

By the definitions and Lemma 4.1, $s^{W,x,y,z}_{T_{[1..1]}}(a,b) = A + B + C + D$ 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.

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

4.1.2 The Case $k > 1$

Next, for any $k > 1$, assume we can create a data structure $B_{W,k-1}$ from which $s^{W,x,y,z}_{T_{[1..k-1]}}(a,b)$ 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 $s^{W,x,y,z}_{T_{[1..k]}}(a,b)$. 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 these subtrees into the four sets W_{fan}, W_{mid}, W_{left}, and W_{right} according to the same criteria as for $k = 1$ above. Then:

Lemma 4.2 *For any $b \in W$, let W_{fan}, W_{mid}, W_{right}, W_{left} be the four sets of subtrees attached to the path from the root of $BT(W, T_k)$ to b. It holds that $s^{W,x,y,z}_{T_{[1..k]}}(a,b) = A + B + C + D$, where*

- $A = \sum_{S \in W_{fan}} s^{\Lambda(S),x,y,z}_{T_{[1..k-1]}}(a,b)$.
- $B = \sum_{S \in W_{mid}} s^{\Lambda(S),x,y,z+1}_{T_{[1..k-1]}}(a,b)$.

```
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: $s^{W,x,y,z}_{T_{[1..i]}}(a, b)$
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. 9 Procedure for computing $s^{W,x,y,z}_{T_{[1..i]}}(a, b)$, assuming $B_{W,i}$ is available

- $C = \sum_{S \in W_{left}} s^{\Lambda(S),x,y+1,z}_{T_{[1..k-1]}}(a, b)$.
- $D = \sum_{S \in W_{right}} s^{\Lambda(S),x+1,y,z}_{T_{[1..k-1]}}(a, b)$.

Figure 9 lists the pseudocode of the procedure `counting_query` for computing $s^{W,x,y,z}_{T_{[1..k]}}(a, b)$, given $B_{W,k}$. See also Fig. 10 for an illustration. The next lemma bounds its running time.

Lemma 4.3 *Given the data structure $B_{W,k}$ for a fixed $a \in L$, for any $b \in L \setminus \{a\}$, Procedure* `counting_query`(k, W, x, y, z, b) *computes $s^{W,x,y,z}_{T_{[1..k]}}(a, b)$ in $O(\log^k n)$ time.*

Proof The procedure first identifies the path P from the root to b in $BT(W, T_k)$. Then, it computes four sets of subtrees W_{fan}, W_{mid}, W_{right}, and W_{left}. Since $BT(W, T_k)$ is a balanced tree, it takes $O(\log n)$ time to obtain them. Then, for each subtree S, it makes a recursive call using the data structure $B_{\Lambda(S),k-1}$, which requires $O(\log^{k-1} n)$ time. In total, the running time is $O(\log^k n)$. □

4.2 Determining Which Clusters are Strong Clusters for Unbounded *k*

Let $\mathcal{A}$ be the tree of all Apresjan clusters. An $O(n^2 \log^{k+2} n)$-time method for checking all the clusters included in $\mathcal{A}$ to see which of them are strong clusters is developed in this subsection.

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.

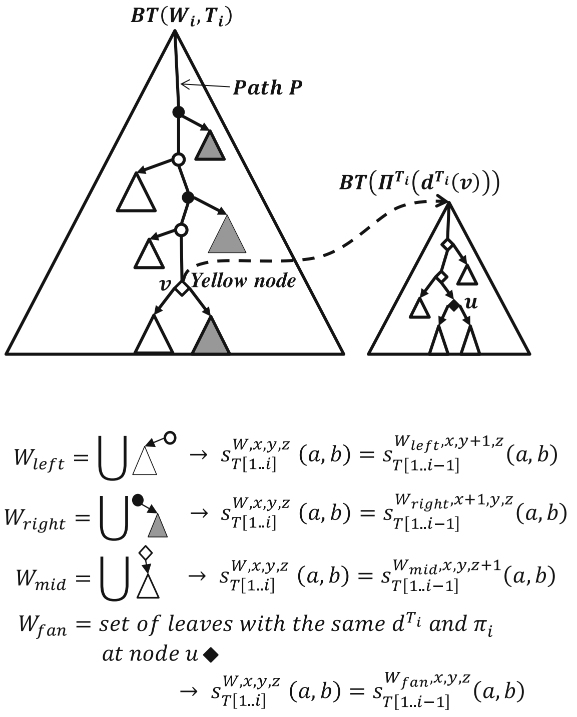
$BT(W_i, T_i)$
Path P
$BT(\Pi^{T_i}(d^{T_i}(v)))$
v
Yellow node
u
(1) $W_{left} = \bigcup$ → $s^{W,x,y,z}_{T[1..i]}(a,b) = s^{W_{left},x,y+1,z}_{T[1..i-1]}(a,b)$
(2) $W_{right} = \bigcup$ → $s^{W,x,y,z}_{T[1..i]}(a,b) = s^{W_{right},x+1,y,z}_{T[1..i-1]}(a,b)$
(3) $W_{mid} = \bigcup$ → $s^{W,x,y,z}_{T[1..i]}(a,b) = s^{W_{mid},x,y,z+1}_{T[1..i-1]}(a,b)$
(4) W_{fan} = set of leaves with the same d^{T_i} and π_i at node u ◆
→ $s^{W,x,y,z}_{T[1..i]}(a,b) = s^{W_{fan},x,y,z}_{T[1..i-1]}(a,b)$

Fig. 10 Illustrating the process to compute $s^{W,x,y,z}_{T[1..k]}(a, b)$ from $B_{W,k}$ (Procedure `counting_query`). Suppose the current recursion level is i. Let P be the path from the root of $BT(W_i, T_i)$ to the node v, where $d^{T_i}(v) = d^{T_i}(b)$ or v is a leaf of $BT(W_i, T_i)$. Along P, nodes with values $< d^{T_i}(b)$ and $> d^{T_i}(b)$ are marked by *white* and *black filled circles*, respectively, and a *diamond symbol* indicates that $d_{T_i}(v) = d^{T_i}(b)$. Any leaf w_L stored at some node in $BT(W_{i-1}, T_{i-1})$ pointed to from a *white circle* satisfies $d_{T_i}(w_L) < d_{T_i}(b)$, which yields formula (1). Similarly, any leaf w_R stored at some node pointed to from a *black circle* satisfies $d_{T_i}(w_R) > d_{T_i}(b)$, giving formula (2). For the yellow node v pointing to $BT(\pi^{T_i}(d^{T_i}(v)))$, traverse from the root to node u, where $\pi_i(u) = \pi_i(b)$ or u is a leaf of $BT(\pi^{T_i}(d^{T_i}(v)))$. If $\pi_i(u) = \pi_i(b)$ then node u contains every leaf w_F with $d^{T_i}(w_F) = d^{T_i}(b)$ and $\pi_i(w_F) = \pi_i(b)$; any other leaf w_M in $BT(\pi^{T_i}(d^{T_i}(v)))$ satisfies $d^{T_i}(w_M) = d^{T_i}(b)$ and $\pi_i(w_M) \neq \pi_i(b)$. This gives formulas (3) and (4). Procedure `counting_query` computes all contributions to $s^{W,x,y,z}_{T[1..i]}(a, b)$ from (1), (2), (3), and (4) recursively, and sums them up

Lemma 4.4 *For any $A \subseteq L$, A is a strong cluster of $\mathcal{R}_{maj}$ if and only if $s_{\mathcal{R}_{maj}}(a, b) = |L \backslash A| + s^{A}_{\mathcal{R}_{maj}}(a, b)$ for all $a, b \in A$ with $a \neq b$.*

Proof (→) For any $a, b \in L$ with $a \neq b$, note that $s_{\mathcal{R}_{maj}}(a, b) = s^{L\backslash A}_{\mathcal{R}_{maj}}(a, b) + s^{A}_{\mathcal{R}_{maj}}(a, b)$. Since A is a strong cluster, we have $ab|c \in \mathcal{R}_{maj}$ for all $c \in L\backslash A$. Hence, $s^{L\backslash A}_{\mathcal{R}_{maj}}(a, b) = |L\backslash A|$, which yields $s_{\mathcal{R}_{maj}}(a, b) = |L\backslash A| + s^{A}_{\mathcal{R}_{maj}}(a, b)$.

(←) For any $a, b \in L$ with $a \neq b$, it is given that $s_{\mathcal{R}_{maj}}(a, b) = |L\backslash A| + s^{A}_{\mathcal{R}_{maj}}(a, b)$. Together with the fact that $s_{\mathcal{R}_{maj}}(a, b) = s^{L\backslash A}_{\mathcal{R}_{maj}}(a, b) + s^{A}_{\mathcal{R}_{maj}}(a, b)$, we get

$s^{L\backslash A}_{\mathcal{R}_{maj}}(a,b) = |L\backslash A|$. This implies that $ab|q \in \mathcal{R}_{maj}$ for all $q \in L\backslash A$. By definition, A is a strong cluster. □

Observe that $s^{A}_{\mathcal{R}_{maj}}(a,b) = s^{A,0,0,0}_{T_{[1..k]}}(a,b)$, using the notation from Sect. 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 $b \in A\backslash\{a\}$.

Lemma 4.5 *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 $b \in A\backslash\{a\}$.*

Proof As in Sect. 4.1, we create a data structure with two parts. The first part is a balanced binary tree $BT(L,\mathcal{A})$ (see Sect. 4.1). For the second part, we build $B_{\Lambda(S),k}$ (again, see Sect. 4.1) for all subtrees S in $BT(L,\mathcal{A})$. This takes $O(n \log^{k+1} n)$ time in total.

For any Apresjan cluster A that contains a and any $b \in A\backslash\{a\}$, use $BT(L,\mathcal{A})$ to partition A into $p = O(\log n)$ subsets $\{A_1, \ldots, A_p\}$ where each subset corresponds to a subtree in $BT(L,\mathcal{A})$. For each subset A_j, compute $s^{A_j}_{\mathcal{R}_{maj}}(a,b)$ in $O(\log^k n)$ time with the data structure $B_{A_j,k}$. Since $s^{A}_{\mathcal{R}_{maj}}(a,b) = \sum_{j=1}^{p} s^{A_j}_{\mathcal{R}_{maj}}(a,b)$, $s^{A}_{\mathcal{R}_{maj}}(a,b)$ can be computed in $O(p \log^k n) = O(\log^{k+1} n)$ time. □

Also, observe the following:

Lemma 4.6 *If a node u in $\mathcal{A}$ satisfies $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$, then, for every ancestor u' of u, $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^{u'})| + s^{\Lambda(\mathcal{A}^{u'})}_{\mathcal{R}_{maj}}(a,b)$ holds.*

Proof Since $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$, we have $s^{L\backslash\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b) = s_{\mathcal{R}_{maj}}(a,b) - s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^u)|$. This means that $ab|w \in \mathcal{R}_{maj}$ for all $w \in L\backslash\Lambda(\mathcal{A}^u)$. Now let v be the parent of u in $\mathcal{A}$. Because $L\backslash\Lambda(\mathcal{A}^v) \subseteq L\backslash\Lambda(\mathcal{A}^u)$, we have $s^{L\backslash\Lambda(\mathcal{A}^v)}_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^v)|$. Moreover, $s_{\mathcal{R}_{maj}}(a,b) = s^{L\backslash\Lambda(\mathcal{A}^v)}_{\mathcal{R}_{maj}}(a,b) + s^{\Lambda(\mathcal{A}^v)}_{\mathcal{R}_{maj}}(a,b)$, so $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^v)| + s^{\Lambda(\mathcal{A}^v)}_{\mathcal{R}_{maj}}(a,b)$. □

By Lemma 4.6, $\mathcal{A}$ contains a node $u^{a,b}_{min}$ such that $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$ 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 4.7 *Given the data structure in Lemma 4.5, $u^{a,b}_{min}$ for any $b \in L$ can be found in $O(\log^{k+2} n)$ time.*

Proof Let P be the path in $\mathcal{A}$ between $lca^{\mathcal{A}}(a,b)$ and the root. Among the clusters $\Lambda(\mathcal{A}^u)$ for all $u \in P$, we aim to find the lowest node u such that $s_{\mathcal{R}_{maj}}(a,b) = |L\backslash\Lambda(\mathcal{A}^u)| + s^{\Lambda(\mathcal{A}^u)}_{\mathcal{R}_{maj}}(a,b)$.

```
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 $u_{min}^{a,b}$ by Lemma 4.7 and set $count(u_{min}^{a,b}) = count(u_{min}^{a,b}) + 1$;
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. 11 Procedure for checking which Apresjan clusters are strong clusters

For each node $u \in P$, it takes $O(\log^{k+1} n)$ time to check if $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus \Lambda(\mathcal{A}^u)| + s_{\mathcal{R}_{maj}}^{\Lambda(\mathcal{A}^u)}(a, b)$. Checking all nodes in P would thus take $O(|P| \log^{k+1} n)$ time. We can speed up the process to $O(\log |P| \log^{k+1} n) = O(\log^{k+2} n)$ time by performing a binary search instead. □

Finally, we describe the procedure `Verify_strong_clusters` for checking which clusters in $\mathcal{A}$ are strong clusters. See Fig. 11 for the pseudocode. First, initialize $count(u) = 0$ for every node u in $\mathcal{A}$. Then, compute $u_{min}^{a,b}$ for all $a, b \in L$ using Lemma 4.7, and increase each $count(u_{min}^{a,b})$ 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 4.8 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 4.5 and 4.7, the running time of the procedure `Verify_strong_clusters` is $O(n^2 \log^{k+2} n)$.

Lemma 4.8 *For any node u in $\mathcal{A}$, $\Lambda(\mathcal{A}^u)$ is a strong cluster if and only if $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$.*

Proof According to Lemma 4.4, $\Lambda(\mathcal{A}^u)$ is a strong cluster of $\mathcal{R}_{maj}$ if and only if $s_{\mathcal{R}_{maj}}(a, b) = |L \setminus \Lambda(\mathcal{A}^u)| + s_{\mathcal{R}_{maj}}^{\Lambda(\mathcal{A}^u)}(a, b)$ for all $a, b \in \Lambda(\mathcal{A}^u)$. This is equivalent to saying that $u_{min}^{a,b}$ is a descendant of u for all $a, b \in \Lambda(\mathcal{A}^u)$, which in turn is equivalent to the condition $sum(u) = \binom{|\Lambda(\mathcal{A}^u)|}{2}$. □

Acknowledgments J.J. was funded by The Hakubi Project at Kyoto University and KAKENHI Grant No. 26330014.

References

1. Bansal, M.S., Dong, J., Fernández-Baca, D.: Comparing and aggregating partially resolved trees. Theor. Comput. Sci. **412**(48), 6634–6652 (2011)
2. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN 2000). Volume 1776 of LNCS, pp. 88–94. Springer, Berlin (2000)
3. Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. Theor. Comput. Sci. **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, Volume 61 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pp. 163–184. American Mathematical Society (2003)
5. Bryant, D., Berry, V.: A structured family of clustering and tree construction methods. Adv. Appl. Math. **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. Cole, R., Farach-Colton, M., Hariharan, R., Przytycka, T., Thorup, M.: An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. SIAM J. Comput. **30**(5), 1385–1404 (2000)
8. Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. The MIT Press, Cambridge, MA (1990)
9. 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)
10. Ewing, G.B., Ebersberger, I., Schmidt, H.A., von Haeseler, A.: Rooted triple consensus and anomalous gene trees. BMC Evol. Biol. **8**, 118 (2008)
11. Felsenstein, J.: Inferring Phylogenies. Sinauer Associates Inc, Sunderland, MA (2004)
12. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. **13**(2), 338–355 (1984)
13. Henzinger, M.R., King, V., Warnow, T.: Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. Algorithmica **24**(1), 1–13 (1999)
14. Jansson, J., Sung, W.-K.: Constructing the R* consensus tree of two trees in subcubic time. Algorithmica **66**(2), 329–345 (2013)
15. Kannan, S., Warnow, T., Yooseph, S.: Computing the local consensus of trees. SIAM J. Comput. **27**(6), 1695–1724 (1998)
16. 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)
17. Margush, T., McMorris, F.R.: Consensus n-trees. Bull. Math. Biol. **43**(2), 239–244 (1981)
18. 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)
19. Semple, C., Steel, M.: Phylogenetics, Volume 24 of Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford (2003)
20. Sung, W.-K.: Algorithms in Bioinformatics: A Practical Introduction. Chapman & Hall/CRC, London (2010)