On Finding the Adams Consensus Tree

Jesper Jansson[1], Zhaoxian Li[2], and Wing-Kin Sung[2,3]

1 **Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan**
E-mail: jj@kuicr.kyoto-u.ac.jp
Funded by The Hakubi Project and KAKENHI grant number 26330014.
2 **School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417**
E-mail: lizhaoxianfgg@gmail.com, ksung@comp.nus.edu.sg
3 **Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672**

Abstract

This paper presents a fast algorithm for finding the Adams consensus tree of a set of conflicting phylogenetic trees with identical leaf labels, for the first time improving the time complexity of a widely used algorithm invented by Adams in 1972 [1]. Our algorithm applies the centroid path decomposition technique [9] in a new way to traverse the input trees' centroid paths in unison, and runs in $O(kn \log n)$ time, where k is the number of input trees and n is the size of the leaf label set. (In comparison, the old algorithm from 1972 has a worst-case running time of $O(kn^2)$.) For the special case of $k = 2$, an even faster algorithm running in $O(n \cdot \frac{\log n}{\log \log n})$ time is provided, which relies on an extension of the wavelet tree-based technique by Bose *et al.* [6] for orthogonal range counting on a grid. Our extended wavelet tree data structure also supports truncated range maximum queries efficiently and may be of independent interest to algorithm designers.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, J.3 Life and Medical Sciences

Keywords and phrases phylogenetic tree, Adams consensus, centroid path, wavelet tree

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.487

1 Introduction

Scientists use *phylogenetic trees* to describe treelike evolutionary history [10, 17, 20, 22]. A *consensus tree* is a phylogenetic tree that reconciles two or more given phylogenetic trees with identical leaf labels but different branching patterns, e.g., obtained from alternative data sets or obtained by resampling during phylogenetic reconstruction or phylogenetic analysis.

The concept of a consensus tree was introduced by Adams in 1972 [1], and the tree constructed by the algorithm in [1] is nowadays referred to as the *Adams consensus tree*. Since conflicting branching information can be resolved in various ways, a number of alternative definitions of consensus trees have been proposed and analyzed in the literature since then; see, e.g., the surveys in [8], Chapter 30 in [10], or Chapter 8.4 in [22]. However, the Adams consensus tree was the only existing consensus tree of any kind for several years and thus gained popularity among the research community early on. It has been implemented in classic phylogenetics software packages such as PAUP* [23] and COMPONENT [18]. Over the decades, many articles in biology have utilized the Adams consensus tree to reach their conclusions; some examples of highly cited ones include [15, 19, 24].

Apart from its historical significance, two useful features of the Adams consensus tree are that it preserves the nesting information common to all the input trees [2] and that it

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 487–499

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


does not introduce any new rooted triplet information [8]. Another feature of the Adams consensus tree is its robustness; adding extra copies of any of the input trees will not affect the output [10], whereas the structure of the so-called *majority rule consensus tree* [16] or the *frequency difference consensus tree* [12] may change completely. In addition, the Adams consensus tree is insensitive to the order in which the input trees are provided [1], as opposed to the *greedy consensus tree* [8, 11]. Finally, it may be much more informative than the *strict consensus tree* [21] and the *loose consensus tree* [7] in cases where a few leaves are in the wrong positions in some of the input trees due to noisy data (for an example, refer to Figure 1 in reference [2]).

The original algorithm of [1] for building the Adams consensus tree has a worst-case running time of $O(kn^2)$, where k is the number of input trees and n is the size of the leaf label set [20]. Despite its practical usefulness, its running time has not been improved in the last forty years. The purpose of this paper is to achieve a better time complexity. The algorithm of [1] is reviewed in Section 1.2, and Section 2 shows that its *expected* running time is in fact $o(kn^2)$ for trees generated by some realistic models of evolution. Next, Section 3 gives an improved algorithm whose worst-case running time is $O(kn \log n)$, based on a new way of applying the centroid path decomposition technique [9]. Finally, Section 4 presents an even faster method for the case $k = 2$ with a worst-case running time of $O(n \cdot \frac{\log n}{\log \log n})$, using an extension of the wavelet tree of Bose *et al.* [6] (described in Section 4.2).

1.1 Definitions and notation

We will use the following definitions. A *phylogenetic tree* is a rooted, unordered, leaf-labeled tree such that all leaves have different labels and every internal node has at least two children. Below, phylogenetic trees are called "trees" for short, and every leaf in a tree is identified with its label. All edges in a tree are assumed to be directed from the root to the leaves.

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 $u, v \in V(T)$, u is called a *descendant of* v and v is called an *ancestor of* u if there exists a (possibly empty) directed path in T from v to u; if this path is nonempty then we write $u \prec v$ and call u a *proper descendant of* v and v a *proper ancestor of* u. For any $u \in V(T)$, T^u is the subtree of T rooted at u, i.e., the subgraph of T induced by the node u and all of its proper descendants in T. For any $u \in V(T)$, let $Child^T(u)$ be the set of all children of u in T. The *depth* of any $u \in V(T)$, denoted by $depth^T(u)$, is the number of edges on the unique path from the root of T to u. For any nonempty $X \subseteq V(T)$, $lca^T(X)$ is the lowest common ancestor in T of the nodes in X.

For any nonempty $B \subseteq \Lambda(T)$, define the *restriction of* T *to* B, denoted by $T|B$, as the tree T' with leaf label set B and node set $\{lca^T(\{u, v\}) : u, v \in B\}$ that preserves the ancestor relations from T, i.e., that satisfies $lca^T(B') = lca^{T'}(B')$ for all nonempty $B' \subseteq B$.

Next, let $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ be any set of trees satisfying $\Lambda(T_1) = \Lambda(T_2) = \cdots = \Lambda(T_k) = L$ for some leaf label set L. The *Adams consensus tree of* $\mathcal{S}$ [1, 2] is the unique tree T with $\Lambda(T) = L$ for which the following two properties hold:

- For any $A, B \subseteq L$, if $lca^{T_j}(A) \prec lca^{T_j}(B)$ in every $T_j \in \mathcal{S}$ then $lca^T(A) \prec lca^T(B)$.
- For any $u, v \in V(T)$, if $u \prec v$ in T then $lca^{T_j}(\Lambda(T^u)) \prec lca^{T_j}(\Lambda(T^v))$ in every $T_j \in \mathcal{S}$.

See Figure 1 for an example. Importantly, it was proved in [2] that these two properties are satisfied by the output of the algorithm in [1] (reviewed in Section 1.2 below). This means that to prove the correctness of a new algorithm for building the Adams consensus tree, one just needs to show that its output is equal to the output of the algorithm in [1].

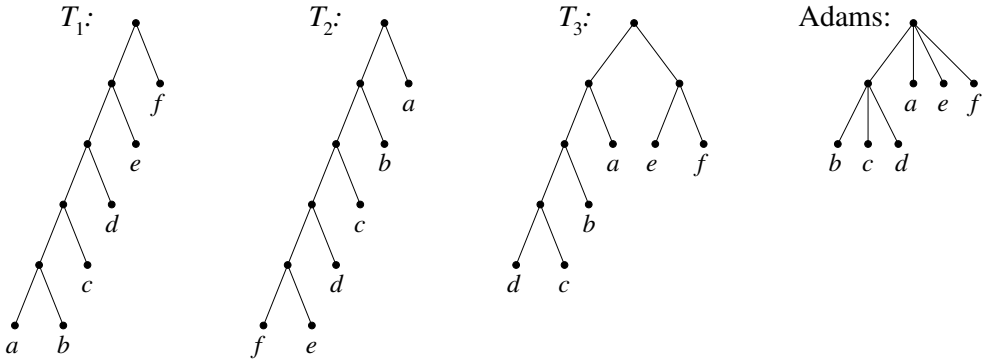

Figure 1 An example. Let $\mathcal{S} = \{T_1, T_2, T_3\}$ as above with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$. The Adams consensus tree of $\mathcal{S}$ is shown on the right. Also note that in this particular example, the Adams consensus tree of $\mathcal{S}$ does not equal the Adams consensus tree of $\{A, T_3\}$, where A is the Adams consensus tree of $\{T_1, T_2\}$.

For any input set $\mathcal{S}$ of trees with identical leaf label sets, we write $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ and define $L = \Lambda(T_1) = \Lambda(T_2) = \cdots = \Lambda(T_k)$. To express the time complexity of any algorithm computing the Adams consensus tree of $\mathcal{S}$, we define $k = |\mathcal{S}|$ and $n = |L|$.

1.2 Previous work

The Adams consensus tree can be computed by the algorithm from [1], which we will now describe. From here on, it will be referred to as `Old_Adams_consensus`. The pseudocode is given in Algorithm 1.

For any tree T, define $\pi(T) = \{\Lambda(T^c) : c \in Child^T(r)$, where r is the root of $T\}$. Observe that $\pi(T)$ is a partition of $\Lambda(T)$. Next, for any set of trees $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ with $\Lambda(T_1) = \Lambda(T_2) = \cdots = \Lambda(T_k) = L$ for a leaf label set L, define $\pi(\mathcal{S})$ to be the partition of L in which, for every part $B \in \pi(\mathcal{S})$, it holds that $B = \cap_{j=1}^{k} \Lambda(T_j^{c_j})$ for some child c_j of the root of T_j for each $j \in \{1, 2, \ldots, k\}$. Thus, $\pi(\mathcal{S})$ is the product of the partitions $\pi(T_1), \pi(T_2), \ldots, \pi(T_k)$. As an example, in Figure 1, we have $\pi(T_1) = \big\{\{a, b, c, d, e\}, \{f\}\big\}$, $\pi(T_2) = \big\{\{a\}, \{b, c, d, e, f\}\big\}$, $\pi(T_3) = \big\{\{a, b, c, d\}, \{e, f\}\big\}$, and $\pi(\mathcal{S}) = \big\{\{a\}, \{b, c, d\}, \{e\}, \{f\}\big\}$.

To compute $\pi(\mathcal{S})$, one can apply Procedure `Compute_partition` in Algorithm 2. It encodes each $\ell \in L$ by a vector of length k whose jth entry $m_j(\ell)$ (for $j \in \{1, 2, \ldots, k\}$) indicates which child of the root of T_j is an ancestor of ℓ. In this way, any two leaf labels

Algorithm 1 Algorithm `Old_Adams_consensus`, adapted from [1].

Algorithm `Old_Adams_consensus`

Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.
Output: The Adams consensus tree of $\mathcal{S}$.

1: **if** T_1 has only one leaf **then** let $T := T_1$; /* Base case of the recursion */
2: **else** /* General case of the recursion */
3: $\pi :=$ `Compute_partition`$(\mathcal{S})$;
4: **for** every $B \in \pi$ **do** $T_B :=$ `Old_Adams_consensus`$(\{T_1|B, T_2|B, \ldots, T_k|B\})$;
5: Create a tree T whose root is the parent of the root of T_B for every $B \in \pi$;
6: **end if**
7: **return** T;

Algorithm 2 Procedure `Compute_partition`.

Procedure `Compute_partition`

Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L'$.
Output: A list of all parts in the partition $\pi(\mathcal{S})$ of L'.

1: Fix an arbitrary left-to-right ordering of the children of the root of every $T_j \in \mathcal{S}$ and denote the ith child (according to this ordering) of the root of T_j by c_j^i;
2: **for** every $\ell \in L'$ **do** compute the vector $(m_1(\ell), m_2(\ell), \ldots, m_k(\ell))$, where for $j \in \{1, 2, \ldots, k\}$, $m_j(\ell) = i$ if and only if ℓ is a descendant of c_j^i in T_j;
3: Put the vectors $(m_1(\ell), m_2(\ell), \ldots, m_k(\ell))$ for all $\ell \in L'$ in a list $\mathcal{W}$ and sort $\mathcal{W}$;
4: Do a single scan of $\mathcal{W}$ to identify the parts in $\pi(\mathcal{S})$ and **return** them;

in L belong to the same part in $\pi(\mathcal{S})$ if and only if their vectors are identical. By sorting the list $\mathcal{W}$ of all vectors and scanning $\mathcal{W}$ to find vectors that are identical, the parts in $\pi(\mathcal{S})$ are obtained.

`Old_Adams_consensus` first computes $\pi(\mathcal{S})$. It then recursively constructs the Adams consensus tree of $\{T_1|B, T_2|B, \ldots, T_k|B\}$ for each B in $\pi(\mathcal{S})$ and attaches all of them to a newly created common root node. By Theorem 3 in [2], this yields the Adams consensus tree of $\mathcal{S}$. According to [20], the time complexity of `Old_Adams_consensus` is $O(kn^2)$.

2 Preliminaries

This section reanalyzes the time complexity of `Old_Adams_consensus`. For any $B \subseteq L$, say that B is a *relevant block* if at any point of the algorithm's execution, Step 4 makes a recursive call with $\{T_1|B, T_2|B, \ldots, T_k|B\}$ as the argument. Define $\mathcal{B} = \{B : B \text{ is a relevant block}\}$. For every $\ell \in L$, define $\mathcal{B}(\ell) = \{B \in \mathcal{B} : \ell \in B\}$.

▶ **Lemma 1.** *For every $\ell \in L$, it holds that $|\mathcal{B}(\ell)| \leq \min_{j=1}^{k} depth^{T_j}(\ell)$.*

Proof. Step 3 of `Old_Adams_consensus` initially generates a partition π_1 of L, and there exists exactly one relevant block B_1 in π_1 such that $\ell \in B_1$. Then, during the recursive call `Old_Adams_consensus` $(\{T_1|B_1, T_2|B_1, \ldots, T_k|B_1\})$, a partition π_2 of B_1 is generated in the same way, and there exists exactly one relevant block B_2 in π_2 such that $\ell \in B_2$. This process is repeated until a relevant block of the form $B_m = \{\ell\}$ is reached and the recursion stops. At any recursion level i, when `Old_Adams_consensus`$(\{T_1|B_i, T_2|B_i, \ldots, T_k|B_i\})$ makes a call to `Old_Adams_consensus` $(\{T_1|B_{i+1}, T_2|B_{i+1}, \ldots, T_k|B_{i+1}\})$, it always holds that $depth^{T_j|B_{i+1}}(\ell) \leq depth^{T_j|B_i}(\ell) - 1$ for all trees $T_j \in \mathcal{S}$. Hence, the number of recursive calls that involve ℓ is upper-bounded by $\min_{j=1}^{k} depth^{T_j}(\ell)$. ◀

▶ **Theorem 2.** `Old_Adams_consensus` *runs in $O(k \cdot \sum_{\ell \in L} \min_{j=1}^{k} depth^{T_j}(\ell))$ time.*

Proof. We first explain how to implement the procedure `Compute_partition` to run in $O(k|L'|)$ time, where L' is the leaf label set of its input $\mathcal{S}$. In Step 2, use the *level ancestor* data structure from [4] as follows: Spend $O(|L'|)$ time to preprocess each $T_j \in \mathcal{S}$ so that the ancestor of any $\ell \in L'$ at depth 1 in T_j can be retrieved in $O(1)$ time. This preprocessing takes $O(k|L'|)$ time, and finding the vectors $(m_1(\ell), m_2(\ell), \ldots, m_k(\ell))$ for all $\ell \in L'$ subsequently takes a total of $O(k|L'|)$ time. In Step 3, sort the list $\mathcal{W}$ in $O(k|L'|)$ time by radix sort.

Next, we consider `Old_Adams_consensus`. Before running the algorithm, use the method in Section 8 of [9] to preprocess each $T_j \in \mathcal{S}$ in $O(n)$ time so that $T_j|B$ for any $B \subseteq L$ can be constructed in $O(|B|)$ time. This takes $O(kn)$ time in total. It follows from the

definition of $T_j|B$ in Section 1.1 that for any $A \subsetneq B$, $(T_j|B)|A = T_j|A$ holds, so the same preprocessing works for all recursion levels and does not need to be repeated during recursive calls. Excluding the time required by its recursive calls, the running time of `Old_Adams_consensus`$(\{T_1|B, T_2|B, \ldots, T_k|B\})$ then becomes $O(k|B|)$ for each $B \in \mathcal{B}$. In total, the running time of `Old_Adams_consensus`$(\mathcal{S})$ is $O(kn + \sum_{B\in\mathcal{B}} k|B|) = O(k \cdot \sum_{B\in\mathcal{B}} |B|) = O(k \cdot \sum_{\ell\in L} |\mathcal{B}(\ell)|)$. By Lemma 1, $\sum_{\ell\in L} |\mathcal{B}(\ell)| \leq \sum_{\ell\in L} \min_{j=1}^{k} depth^{T_j}(\ell)$. The theorem follows. ◀

Since $|L| = n$ and $depth^{T_j}(\ell) < n$ for all $\ell \in L$ and $T_j \in \mathcal{S}$, Theorem 2 implies that the worst-case running time of `Old_Adams_consensus` is $O(kn^2)$, as already mentioned in [20]. However, if the average leaf depth is small then the running time will be better. According to Theorem 2, we obtain:

▶ **Corollary 3.** *If $\mathcal{S}$ is a set of trees with expected average leaf depth α then the expected running time of* `Old_Adams_consensus` *is $O(kn\alpha)$.*

For example, the expected average leaf depth in a random binary phylogenetic tree with n leaves generated in the Yule-Harding model [5, 14, 20], the uniform model [5, 20], and the activity model [14] (with the activity parameter p set to $\frac{1}{2}$) is $O(\log n)$ [5, 14], $O(n^{1/2})$ [5], and $O(n^{1/2})$ [14], respectively. In these cases, the expected running time of `Old_Adams_consensus` will be $O(kn \log n)$, $O(kn^{1.5})$, and $O(kn^{1.5})$.

3 New algorithm for k input trees

This section gives a more efficient solution for computing the Adams consensus tree of k input trees. The algorithm is called `New_Adams_consensus_k` and its worst-case running time is $O(kn \log n)$.

The main idea is to use the centroid path decomposition technique [9] in a new manner to avoid making recursive calls to "large" subproblems, and treat them iteratively instead. Essentially, by utilizing Lemma 4 below, the algorithm implicitly computes $\pi(\mathcal{S})$ in such a way that the Adams consensus tree can be constructed recursively for all parts in $\pi(\mathcal{S})$, *except for one*. To handle the remaining part, its corresponding Adams consensus tree is constructed iteratively by going down the centroid paths in all the trees in unison and applying Lemma 4 at each level. (As a side note, this kind of "synchronized centroid path traversal" appears to be a novel way of applying the centroid path decomposition technique.) Finally, the Adams consensus tree of $\mathcal{S}$ is assembled by attaching the root of each tree constructed for the parts in $\pi(\mathcal{S})$ to a new root node.

The details of the algorithm are described below, and the pseudocode is listed in Algorithm 3.

Some additional definitions are needed. Recall from [9] that a *centroid path* in a tree T is a path in T of the form $P = \langle p_\alpha, p_{\alpha-1}, \ldots, p_1\rangle$, where the node p_{w-1} for every $w \in \{2, \ldots, \alpha\}$ is any child of p_w with the maximum number of leaf descendants, and p_1 is a leaf. Let P be a centroid path in a tree T. For any $u \in V(T)$ such that u does not belong to P but the parent of u does, the subtree T^u is called a *side tree* of P. For any side tree τ of a centroid path starting at the root of a tree T, the property $|\Lambda(\tau)| \leq |\Lambda(T)|/2$ holds.

A *delete* operation on any non-root, internal node u in a tree is the operation of letting all of u's children become children of the parent of u, and then removing u and the edge between u and its parent. A *fan tree* is a tree in which either all the leaves are children of the root, or there is just a single leaf.

Algorithm 3 Algorithm `New_Adams_consensus_k`.

Algorithm `New_Adams_consensus_k`

Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$.
Output: The Adams consensus tree of $\mathcal{S}$.

1: **if** T_1 has only one leaf **then**
2: $T := T_1$;
3: **else**
4: **for** $j := 1$ **to** k **do**
5: Let P_j be a centroid path in T_j starting at the root, construct the tree T'_j based on P_j, and preprocess T_j;
6: **end for**
7: $h := 0$;
8: **repeat**
9: $h := h + 1$;
10: $X_h := \{x \in L :$ for some $j \in \{1, 2, \ldots, k\}$, x belongs to a fan tree attached to the root of $T'_j\}$;
11: $\pi_{X_h} :=$ `Compute_restricted_partition`$(\{T'_1, T'_2, \ldots, T'_k\}; X_h)$;
12: **for** $j := 1$ **to** k **do** $T'_j := T'_j|(\Lambda(T'_j) \setminus X_h)$;
13: **until** $\Lambda(T'_1) = \emptyset$;
14: **for** $j := 1$ **to** k **do**
15: **for** $w := 1$ **to** h **do** construct $T_i|B$ for all $B \in \pi_{X_w}$;
16: **end for**
17: **for** $w := h$ **downto** 1 **do**
18: **for** $B \in \pi_{X_w}$ **do** $T_B :=$ `New_Adams_consensus_k`$(\{T_1|B, T_2|B, \ldots, T_k|B\})$;
19: Create a tree Q_w whose root is the parent of the root of every T_B, $B \in \pi_{X_w}$;
20: **if** $w < h$ **then** attach the root of Q_{w+1} as a child of the root of Q_w;
21: **end for**
22: $T := Q_1$;
23: **end if**
24: **return** T;

For each $j \in \{1, 2, \ldots, k\}$, let P_j be a centroid path in T_j that starts at the root of T_j. Let T'_j be the tree obtained by taking a copy of T_j and doing a delete operation on every non-root, internal node whose parent does not belong to P_j; note that by performing all delete operations in top-down order, T'_j can be constructed in $O(n)$ time. Thus, T'_j consists of the centroid path P_j with a collection of fan trees attached to it, and each such fan tree's leaf label set is equal to the leaf label set of one of the side trees of P_j. The T'_j-tree is a useful summary of T_j that enables us to quickly retrieve the leaf label set of any side tree in T_j or to check which side tree in T_j that a specified leaf belongs to in $O(1)$ time.

As in `Old_Adams_consensus` above, `New_Adams_consensus_k` needs to compute the partition $\pi(\mathcal{S})$ of L to determine the branching structure at the top level of the Adams consensus tree. However, for efficiency reasons, it does not compute $\pi(\mathcal{S})$ directly. Instead, it computes a *restricted partition*, defined as follows: For any $X \subseteq L$, let $\pi(\mathcal{S}; X) = \{B \cap X : B \in \pi(\mathcal{S})$ and $|B \cap X| \geq 1\}$. In other words, $\pi(\mathcal{S}; X)$ is the partition $\pi(\mathcal{S})$ restricted to elements in X. Note that $\pi(\mathcal{S}; X)$ may not be a true partition of X as it can be a singleton. To continue the example from Figure 1 in Section 1.2 where we had $\pi(\mathcal{S}) = \big\{\{a\}, \{b, c, d\}, \{e\}, \{f\}\big\}$, if $X = \{a, b, c\}$ then $\pi(\mathcal{S}; X) = \big\{\{a\}, \{b, c\}\big\}$ and if $X = \{b, c\}$ then $\pi(\mathcal{S}; X) = \big\{\{b, c\}\big\}$.

▶ **Lemma 4.** *Let* $X = \{x \in L : \text{for some } j \in \{1,2,\ldots,k\},\ x \text{ belongs to a fan tree attached to the root of } T'_j\}$. *If* $X \neq L$ *then* $\pi(\{T_1,T_2,\ldots,T_k\}) = \pi(\{T'_1,T'_2,\ldots,T'_k\};X) \cup \{L \setminus X\}$, *and if* $X = L$ *then* $\pi(\{T_1,T_2,\ldots,T_k\}) = \pi(\{T'_1,T'_2,\ldots,T'_k\};X)$.

Proof. X is also equal to $\{x \in L :$ for some $j \in \{1,2,\ldots,k\}$, x belongs to a side tree of P_j attached to the root of $T_j\}$. Consider any $B \in \pi(\{T_1,T_2,\ldots,T_k\})$. If B contains at least one element from X then $B \subseteq X$, and consequently $B \cap X = B$ and $B \in \pi(\{T_1,T_2,\ldots,T_k\};X)$. On the other hand, if B contains no elements from X then B must be equal to $L \setminus X$. Therefore, $\pi(\{T_1,T_2,\ldots,T_k\}) \subseteq \pi(\{T_1,T_2,\ldots,T_k\};X) \cup \{L \setminus X\}$ when $X \neq L$, and $\pi(\{T_1,T_2,\ldots,T_k\}) \subseteq \pi(\{T_1,T_2,\ldots,T_k\};X)$ when $X = L$.

Next, consider any $B \in \pi(\{T_1,T_2,\ldots,T_k\};X)$. By definition, $B \in \pi(\{T_1,T_2,\ldots,T_k\})$. Also, if $X \neq L$ then $L \setminus X$ is nonempty and consists of all leaves that are descendants of the child of the root of T_j that lies on P_j for every $j \in \{1,2,\ldots,k\}$; since all these leaves belong to the same part in $\pi(T_j)$ for each $j \in \{1,2,\ldots,k\}$, we have $L \setminus X \in \pi(\{T_1,T_2,\ldots,T_k\})$. Thus, $\pi(\{T_1,T_2,\ldots,T_k\};X) \cup \{L \setminus X\} \subseteq \pi(\{T_1,T_2,\ldots,T_k\})$ when $X \neq L$, and $\pi(\{T_1,T_2,\ldots,T_k\};X) \subseteq \pi(\{T_1,T_2,\ldots,T_k\})$ when $X = L$.

Finally, $\pi(\{T'_1,T'_2,\ldots,T'_k\};X) = \pi(\{T_1,T_2,\ldots,T_k\};X)$ by the construction of the T'_j-trees. The lemma follows. ◀

We now describe `New_Adams_consensus_k`.

First, for each $j \in \{1,2,\ldots,k\}$, Steps 4–6 build P_j and T'_j and preprocess T_j in $O(n)$ time as in Section 8 of [9] so that for any specified partition π of L, the set of all trees of the form $T_j|B_i$ with $B_i \in \pi$ can be constructed in $O(n)$ total time later on. The algorithm then enters a **repeat**-loop (Steps 8–13) that computes and stores the restricted partition $\pi(\{T'_1,T'_2,\ldots,T'_k\};X_1)$, where X_1 is the subset X of $\Lambda(T'_1)$ $(= \Lambda(T'_2) = \ldots = \Lambda(T'_k))$ defined in Lemma 4. By Lemma 4, the parts in $\pi(\{T'_1,T'_2,\ldots,T'_k\};X_1)$ along with $\Lambda(T'_1) \setminus X_1$ yield the partition at the top level of the Adams consensus tree. After that, the leaves belonging to X_1 are removed from all the T'_j-trees. The process is repeated until the T'_j-trees are empty, and each subsequent iteration of the **repeat**-loop mimics the computations at one recursion level in `Old_Adams_consensus` that determine how to further partition the leaves in the set $\Lambda(T'_1) \setminus X_1$. Next, the algorithm constructs $T_j|B$ for every part B previously computed by the **repeat**-loop for all $j \in \{1,2,\ldots,k\}$ (Steps 14–16). Then, the Adams consensus tree Q_w at each level w is built by recursively computing the Adams consensus tree T_B for every part B in $\pi(\{T'_1,T'_2,\ldots,T'_k\};X_w)$ at this level (Step 18), combining the obtained solutions (Step 19), and attaching the Adams consensus tree Q_{w+1} for the part corresponding to $L \setminus X_w$ in Lemma 4 (Step 20). Lastly, the tree Q_1 obtained at the topmost level is returned (Step 24). The correctness follows from Lemma 4 and the correctness of `Old_Adams_consensus`.

The time complexity is given by the next theorem:

▶ **Theorem 5.** `New_Adams_consensus_k` *runs in* $O(kn \log n)$ *time.*

Proof. Denote the time complexity of `New_Adams_consensus_k`$(\{T_1|L', T_2|L', \ldots, T_k|L'\})$ for any $L' \subseteq L$ by $t(L')$.

We derive a recurrence for $t(L')$ in the following way. Steps 4–6 build P_j and T'_j and preprocess T_j in $O(|L'|)$ time for each $j \in \{1,2,\ldots,k\}$, i.e., in $O(k|L'|)$ time in total. Iteration h of the **repeat**-loop computes a set X_h in Step 10, which takes $O(k|X_h|)$ time by using the T'_j-trees, and the restricted partition $\pi_{X_h} = \pi(\{T'_1,T'_2,\ldots,T'_k\};X_h)$ of X_h in Step 11, which also takes $O(k|X_h|)$ time by using the technique from Procedure `Compute_partition` in Algorithm 2 and the first part of the proof of Theorem 2. To implement Step 12 in $O(k|X_h|)$ time, update each T'_j-tree directly by removing all leaves that belong to X_h as well as any previously internal node that turns into a leaf as a result and contracting any outgoing edge from

a node of degree 1. Constructing all the trees $T_j|B$ in Steps 14–16 takes a total of $O(k|L'|)$ time with the technique from Section 8 of [9]. Finally, for each $w \in \{1, 2, \ldots, h\}$, the recursive calls in Step 18 take $\sum_{B \in \pi_{X_w}} t(B)$ time and building Q_w in Steps 19 and 20 takes $O(|X_w|)$ time. In total, the time complexity is $t(L') = O(k|L'|) + \sum_{w=1}^{h} \big(O(k|X_w|) + \sum_{B \in \pi_{X_w}} t(B)\big)$.

To solve the recurrence, we use the fact that $\bigcup_{w=1}^{h} \pi_{X_w}$ is a partition of L'. Write $\pi_{L'} = \bigcup_{w=1}^{h} \pi_{X_w}$. Then $t(L') = O(k|L'|) + \sum_{B \in \pi_{L'}} t(B)$. Since every part $B \in \pi_{L'}$ is of size at most $|L'|/2$ according to the definition of a side tree of a centroid path, the problem size is reduced by (at least) half for each successive recursive call. Thus, there are $O(\log |L'|)$ recursion levels. The total size of all subproblems in each recursive level is $O(|L'|)$, so each recursion level takes $O(k|L'|)$ time. This gives $t(L') = O(k|L'| \log |L'|)$. ◀

4 New algorithm for two input trees

Here, we present an even faster algorithm for the case $k = 2$. The algorithm is named `New_Adams_consensus_2` and has a worst-case running time of $O(n \cdot \frac{\log n}{\log \log n})$.

4.1 Outline of the algorithm

Consider any recursive call of the form `Old_Adams_consensus`$(\{T_1|B, T_2|B\})$ for some $B \subseteq L$ in the algorithm in Section 1.2. To obtain the partition of the leaves in B, the algorithm will spend $\Omega(|B|)$ time using the procedure `Compute_partition`. A faster method for doing the partitioning is needed to improve the overall running time. First, we observe that by the definition of the algorithm, B always satisfies $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$ for some pair of nodes $u \in V(T_1)$, $v \in V(T_2)$. This means that successive recursive calls to the algorithm can be specified by pairs of vertices from T_1 and T_2. Secondly, we observe that the algorithm needs to proceed recursively from (u, v) only to those (u', v'), where $u' \in Child^{T_1}(u)$ and $v' \in Child^{T_2}(v)$, for which $|\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0$. Based on these observations, define $Z_{u,v} = \{(u', v') : u' \in Child^{T_1}(u), v' \in Child^{T_2}(v), |\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0\}$. We have:

▶ **Lemma 6.** *Suppose $u \in V(T_1)$ and $v \in V(T_2)$ are given. Let $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$, $\gamma = lca^{T_1}(B)$, and $\delta = lca^{T_2}(B)$. If $|B| > 1$ then $\pi(\{T_1|B, T_2|B\}) = \pi(\{T_1^u|B, T_2^v|B\}) = \pi(\{T_1^\gamma|B, T_2^\delta|B\}) = \{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : (u', v') \in Z_{\gamma,\delta}\}$.*

Proof. By definition, $\pi(\{T_1^\gamma|B, T_2^\delta|B\})$ is equal to $\{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : u' \in Child^{T_1}(\gamma), v' \in Child^{T_2}(\delta), \Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) \neq \emptyset\} = \{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : (u', v') \in Z_{\gamma,\delta}\}$. ◀

Algorithm `New_Adams_consensus_2` (summarized in Algorithm 4) uses this lemma to compute the Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$ for any two specified nodes $u \in V(T_1)$, $v \in V(T_2)$, where $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$. (Selecting u and v to be the roots of T_1 and T_2 thus yields the Adams consensus tree of T_1 and T_2.)

The algorithm works as follows. If $|B| = 1$ then the answer is just the common leaf in $\Lambda(T_1^u) \cap \Lambda(T_2^v)$. Otherwise, the algorithm computes $\gamma = lca^{T_1}(B)$ and $\delta = lca^{T_2}(B)$, calls a procedure `Compute_Z` (to be described in Section 4.3) to construct $Z_{\gamma,\delta}$, and then, for every $(u', v') \in Z_{\gamma,\delta}$, computes its corresponding Adams consensus tree $T_{u',v'}$ recursively. The Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$ is obtained by attaching the computed $T_{u',v'}$-trees to a newly created common root node. Lemma 6 implies that this gives the same output as `Old_Adams_consensus`, so the correctness is guaranteed by the correctness of the latter.

Algorithm 4 Algorithm `New_Adams_consensus_2`.

Algorithm `New_Adams_consensus_2`

Input: $u \in V(T_1)$, $v \in V(T_2)$, where T_1, T_2 are two given trees with $\Lambda(T_1) = \Lambda(T_2)$.
Output: The Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$, where $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$.

1: Compute $c := |\Lambda(T_1^u) \cap \Lambda(T_2^v)|$;
2: **if** $c = 1$ **then**
3: Let T be a tree consisting of the only common leaf in $\Lambda(T_1^u) \cap \Lambda(T_2^v)$;
4: **else**
5: Find the leftmost leaf a and the rightmost leaf a' in $T_1|(\Lambda(T_1^u) \cap \Lambda(T_2^v))$, and the leftmost leaf b and the rightmost leaf b' in $T_2|(\Lambda(T_1^u) \cap \Lambda(T_2^v))$;
6: $\gamma := lca^{T_1}(a, a')$; $\delta = lca^{T_2}(b, b')$;
7: $Z :=$ `Compute_Z`(γ, δ);
8: Let T be a tree consisting of a single root node r;
9: **for** every $(u', v') \in Z$ **do**
10: $T_{u',v'} :=$ `New_Adams_consensus_2`(u', v');
11: Attach $T_{u',v'}$ as a child of r;
12: **end for**
13: **end if**
14: **return** T;

4.2 Auxiliary data structure for orthogonal range counting on a grid

The time complexity of `New_Adams_consensus_2` is analyzed in Section 4.3 below. It relies on an efficient data structure for orthogonal range counting on a grid, developed in this subsection and summarized in Lemma 8. This data structure is an extension of the wavelet tree-based data structure used by Bose *et al.* [6] for supporting orthogonal range counting queries on a grid. Our extension consists of also supporting *truncated range maximum* (or *minimum*) *queries* efficiently, where the objective is to report the point with the maximum (or minimum) x-coordinate inside any query rectangle $[1..\ell] \times [s..s']$, if any. Furthermore, we bound the time needed to *construct* the data structure since this is crucial in our application.

Firstly, for smaller grids, we have:

▶ **Lemma 7.** *Let $N = \{(1, N[1]), \ldots, (n, N[n])\}$ be a set of points on an $n \times t$ grid, where $t = O(\log^\epsilon n)$ for any constant ϵ with $0 < \epsilon < 1/2$, such that every column contains exactly one point. We can build a data structure in $O(n)$ time after which: (i) counting the number of points inside any query rectangle $[1..\ell] \times [s..s']$ takes $O(1)$ time; and (ii) reporting the point with the maximum (or minimum) x-coordinate inside any query rectangle $[1..\ell] \times [s..s']$ takes $O(1)$ time.*

Proof. Omitted from the conference version of the paper due to space constraints. ◀

For larger grids, we apply Lemma 7 to obtain:

▶ **Lemma 8.** *Let $N = \{(1, N[1]), \ldots, (n, N[n])\}$ be a set of points on an $n \times n$ grid such that every column contains exactly one point and every row contains exactly one point. We can build a data structure $D(N)$ in $O(n \cdot \frac{\log n}{\log \log n})$ time after which: (i) counting the number of points inside any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time; and (ii) reporting the point with the maximum (or minimum) x-coordinate inside any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time.*

Proof. The basic data structure is the same as in the proof of Lemma 6 in [6], namely a t-ary wavelet tree. Here, we select $t = \log^{\epsilon} n$ for any $0 < \epsilon < 0.5$.

On the top level, we project the n points into the 2d-space $[1..n] \times [1..t]$ by converting each point $(i, N[i])$ to $(i, N_1[i])$, where $N_1[i] = \lfloor N[i]/(n/t) \rfloor$. We use Lemma 7 to maintain a range query data structure for $\{(i, N_1[i]) : i = 1, \ldots, n\}$, and also build a rank data structure that lets us compute $rank_j(i)$ in $N_1[1..n]$ (here, $rank_j(i)$ is the number of occurrences of j in $N_1[1..i]$). This data structure can be built in $O(n)$ time. To be precise, we store $rank_j(i)$ for every i which is a multiple of $\log^2 n$, requiring $O(\frac{nt \log n}{\log^2 n}) = O(n)$ bits space. We also store $rank_j(i) - rank_j(\log^2 n \lfloor \frac{i}{\log^n} \rfloor)$ for every i which is a multiple of $\frac{\log n}{\log \log n}$, requiring $O(t \frac{n \log \log n}{\log n} \log \log nt) = O(n)$ bits. We precompute a table occtable$(x_1, \ldots, x_\ell, j)$ that stores the number of occurrences of j in $x_1, \ldots, x_\ell$, where $1 \le x_i \le t$, $1 \le j \le t$, $\ell \le \frac{\log n}{\log \log n}$. This table has $o(n)$ entries and can be computed in $o(n)$ time. By taking $x = \lfloor \frac{i}{\log^2 n} \rfloor \log^2 n$ and $y = \lfloor \frac{i \log \log n}{\log n} \rfloor \frac{\log n}{\log \log n}$, we have $rank_j(i) = rank_j(x) + (rank_j(y) - rank_j(x)) +$ occtable$(N[i - y + 1], \ldots, N[i], j)$, which can be computed in constant time.

On the second level, based on $N_1[]$, we partition the n points into t point sets $N_{2,1}, \ldots, N_{2,t}$. The set $N_{2,j}$ contains all the points where $N_1[i] = j$. Let $n_{2,j} = |N_{2,j}|$. Every point $(i, N[i])$ in $N_{2,j}$ is projected into the 2d-space $[1..n_{2,j}] \times [1..t]$. Suppose the rank of i is r among all the x-coordinates of the points in $N_{2,j}$. Then, $(i, N[i])$ is converted to $(r, N_{2,j}[r])$ where $N_{2,j}[r] = \lfloor (N[i] - (n/t)j)/(n/t^2) \rfloor$. We use Lemma 7 again to maintain a range query data structure for these $n_{2,j}$ points. We also build a rank data structure for $N_{2,j}[1..n_{2,j}]$ in $O(n_{2,j})$ time. We continue the process recursively and build the above data structures on each level. Since there are $\log_t n$ levels, the entire data structure can be constructed in $O(n \log_t n)$ time.

Next, given any query rectangle $[\ell_1..\ell_2] \times [s_1..s_2]$, we proceed in a similar manner as in [6]. Let $z_1 = \lceil s_1/(n/t) \rceil$ and $z_2 = \lfloor s_2/(n/t) \rfloor$. The query is partitioned into: (1) $[\ell_1..\ell_2] \times [s_1..(n/t)z_1]$; (2) $[\ell_1..\ell_2] \times [(n/t)z_1 + 1..(n/t)z_2]$; and (3) $[\ell_1..\ell_2] \times [(n/t)z_2 + 1..s_2]$.

Query (2) is equivalent to the query $[\ell_1..\ell_2] \times [z_1 + 1..z_2]$ among the points in $\{(i, N_1[i]) : i = 1, \ldots, n\}$, and can be solved in $O(1)$ time according to Lemma 7. Let $x_1 = rank_{z_1 - 1}(\ell_1)$ and $x_2 = rank_{z_1 - 1}(\ell_2)$, and denote $y_1 = s_1 - (n/t)(z_1 - 1)$ and $y_2 = s_2 - (n/t)(z_2 - 1)$. Query (1) is equivalent to the query $[x_1..x_2] \times [y_1..y_2]$ among the points in $N_{2,z_1 - 1}$. We handle this query recursively. Query (3) is handled in the same way. As there are $\log_t n$ levels and each level takes $O(1)$ time, the query is answered in $O(\log_t n) = O(\frac{\log n}{\log \log n})$ time. ◀

4.3 Time complexity

This subsection explains how to implement `New_Adams_consensus_2`. Do the following preprocessing:

- Fix an arbitrary left-to-right ordering of the children at every node in T_1. For $i \in \{1, 2, \ldots, n\}$, let $L_1(i)$ be the ith leaf in T_1 in the resulting left-to-right ordering. (Thus, $(L_1(1), L_1(2), \ldots, L_1(n))$ is a permutation of L.) Define $L_2(i)$ for $i \in \{1, 2, \ldots, n\}$ analogously using T_2. Let $N = \{(L_1^{-1}(\ell), L_2^{-1}(\ell)) : \ell \in L\}$ and build the data structure $D(N)$ from Lemma 8.
- For $j \in \{1, 2\}$, preprocess T_j in $O(n)$ time so that any $lca^{T_j}(B)$-query can be answered in $O(|B|)$ time [3, 13].
- As in the proof of Theorem 2 in Section 2 above, preprocess T_j for $j \in \{1, 2\}$ with the level ancestor data structure of [4] in $O(n)$ time so that the ancestor of any $\ell \in L$ at depth 1 in T_j can be returned in $O(1)$ time. Also preprocess T_j for $j \in \{1, 2\}$ in $O(n)$ time as in Section 8 of [9] so that $T_j|B$ for any $B \subseteq L$ can be constructed in $O(|B|)$ time.

The preprocessing takes $O(n \cdot \frac{\log n}{\log\log n})$ time in total.

Next, for any pair of siblings $u, u' \in V(T_j)$, $j \in \{1,2\}$, let $T_j^{u..u'}$ denote the set of all rooted subtrees of the form T_j^x, where x belongs to the interval of siblings $[u, \ldots, u']$ in T_j, and define $\Lambda(T_j^{u..u'}) = \bigcup_{x\in[u,\ldots,u']} \Lambda(T_j^x)$.

▶ **Lemma 9.** *Given the data structure $D(N)$ in Lemma 8, for any siblings u and u' in T_1 and any siblings v and v' in T_2, the value of $|\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})|$ can be found in $O(\frac{\log n}{\log\log n})$ time. Furthermore, the leftmost and rightmost leaves in T_1 (or T_2) among all leaves in $\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})$ can be reported in $O(\frac{\log n}{\log\log n})$ time.*

Proof. Let l_u be the leftmost leaf in T_1^u and $r_{u'}$ the rightmost leaf in $T_1^{u'}$. Then each $\ell \in \Lambda(T_1^{u..u'})$ satisfies $L_1^{-1}(l_u) \le L_1^{-1}(\ell) \le L_1^{-1}(r_{u'})$. Similarly, each $\ell \in \Lambda(T_2^{v..v'})$ satisfies $L_2^{-1}(l_v) \le L_2^{-1}(\ell) \le L_2^{-1}(r_{v'})$, where l_v is the leftmost leaf in T_2^v and $r_{v'}$ the rightmost leaf in $T_2^{v'}$. Hence, any $\ell \in L$ belongs to $\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})$ if and only if the point $(L_1^{-1}(\ell), L_2^{-1}(\ell))$ lies in the rectangle defined by $[L_1^{-1}(l_u)..L_1^{-1}(r_{u'})] \times [L_2^{-1}(l_v)..L_2^{-1}(r_{v'})]$ on the grid represented by $D(N)$. By Lemma 8, the lemma follows. ◀

Lemma 9 allows the $Z_{u,v}$-sets to be computed quickly by the procedure `Compute_Z` shown in Algorithm 5. More precisely:

▶ **Lemma 10.** *Given the data structure $D(N)$ in Lemma 8, the procedure* `Compute_Z` *can compute the set $Z_{u,v}$ for any $u \in V(T_1)$ and $v \in V(T_2)$ in $O(|Z_{u,v}| \cdot \frac{\log n}{\log\log n})$ time.*

Proof. Let $u_1, \ldots, u_\alpha$ be the ordered list of children of u and $v_1, \ldots, v_\beta$ the ordered list of children of v. The procedure identifies the pairs $(u_p, v_q) \in Z_{u,v}$ in increasing order of u_p and then in increasing order of v_q. In the outer loop, each child u_p of u satisfying $\Lambda(T_1^{u_p}) \cap \Lambda(T_2^v) \neq \emptyset$ is identified from left to right by using Lemma 9 in Step 4 and the level

Algorithm 5 Procedure `Compute_Z`.

Procedure `Compute_Z`

Input: $u \in V(T_1)$, $v \in V(T_2)$, where T_1, T_2 are two given trees with $\Lambda(T_1) = \Lambda(T_2)$.
Output: $Z_{u,v} = \{(u', v') : u' \in Child^{T_1}(u), v' \in Child^{T_2}(v), |\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0\}$.

1: Let $u_1..u_\alpha$ and $v_1..v_\beta$ be the ordered lists of children of u and v, respectively;
2: $Z := \emptyset$; $i := 1$;
3: **while** $i \le \alpha$ **do**
4: Find the leftmost leaf a in T_1 such that $a \in \Lambda(T_1^{u_i..u_\alpha}) \cap \Lambda(T_2^v)$;
5: If no such a exists, **break**;
6: Identify the $u_p \in Child^{T_1}(u)$ such that $a \in \Lambda(T_1^{u_p})$;
7: $j := 1$;
8: **while** $j \le \beta$ **do**
9: Find the leftmost leaf b in T_2 such that $b \in \Lambda(T_1^{u_p}) \cap \Lambda(T_2^{v_j..v_\beta})$;
10: If no such b exists, **break**;
11: Identify the $v_q \in Child^{T_2}(v)$ such that $b \in \Lambda(T_2^{v_q})$;
12: Let $Z := Z \cup \{(u_p, v_q)\}$ and $j := q + 1$;
13: **end while**
14: Let $i := p + 1$;
15: **end while**
16: **return** Z;

ancestor data structure in Step 6. Each u_p is thus identified in $O(\frac{\log n}{\log\log n})$ time. Then, for each such u_p, the inner loop similarly finds every child v_q of v with $\Lambda(T_1^{u_p}) \cap \Lambda(T_2^{v_q}) \neq \emptyset$ from left to right, using $O(\frac{\log n}{\log\log n})$ time per v_q. In total, the procedure spends $O(|Z_{u,v}| \cdot \frac{\log n}{\log\log n})$ time to compute $Z_{u,v}$. ◀

Finally, we are ready to analyze the running time of `New_Adams_consensus_2`. For any $u' \in V(T_1)$, $v' \in V(T_2)$, denote $S_{u',v'} = \Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})$.

▶ **Theorem 11.** *Given the data structure $D(N)$ in Lemma 8,* `New_Adams_consensus_2`(u, v) *for any $u \in V(T_1)$ and $v \in V(T_2)$ runs in $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$ time.*

Proof. Let $T(u, v)$ be the running time of `New_Adams_consensus_2`(u, v), including the time required to compute γ, δ, and $Z_{\gamma,\delta}$ and the recursive calls `New_Adams_consensus_2`(u', v') for all $(u', v') \in Z_{\gamma,\delta}$. By using Lemma 9, Steps 1 and 5 can be carried out in $O(\frac{\log n}{\log\log n})$ time. Step 6 takes $O(1)$ time because of the preprocessing and Step 7 takes $O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log\log n})$ time according to Lemma 10. We therefore have $T(u, v) = \sum_{(u',v') \in Z_{\gamma,\delta}} T(u', v') + O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log\log n})$. Observe that in the base case, i.e., where $|S_{u,v}| = 1$, it holds that $T(u, v) = O(\frac{\log n}{\log\log n})$.

We apply the recursion-tree method to solve the recurrence for $T(u, v)$. The root of the recursion tree for $T(u, v)$ represents the top level of recursion, and its cost is $O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log\log n})$. There are $|Z_{\gamma,\delta}|$ subtrees attached to the root, each of which corresponds to a recursion tree for one $T(u', v')$ where $(u', v') \in Z_{\gamma,\delta}$. The leaves of the recursion tree represent the base cases of the recursion, i.e., those $T(x, y)$ satisfying $|S_{x,y}| = 1$, and they each have cost $O(\frac{\log n}{\log\log n})$. It follows that the recursion tree for $T(u, v)$ has exactly $|S_{u,v}|$ leaves and no nodes with degree 1. Now, the value of $T(u, v)$ is equal to the sum of the costs taken over all nodes in the recursion tree. Clearly, the total contribution of the leaves is $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. We rewrite the cost of each internal node in the recursion tree as $O(d \cdot \frac{\log n}{\log\log n})$, where d is the degree of that node. Since the sum of the degrees of all internal nodes in a tree without any nodes of degree 1 is less than twice the number of leaves, the contribution of the internal nodes is also $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. The total running time is $T(u, v) = O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. ◀

Recall that $D(N)$ is constructed during the preprocessing phase using $O(n \cdot \frac{\log n}{\log\log n})$ time. Theorem 11 implies that `New_Adams_consensus_2`(r_1, r_2), where r_i is the root of T_i for $i \in \{1, 2\}$, computes the Adams consensus tree of $\{T_1, T_2\}$ in $O(n \cdot \frac{\log n}{\log\log n})$ time.

Acknowledgments: The authors would like to thank the anonymous reviewers for their suggestions.

References

1 E. N. Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.

2 E. N. Adams III. N-trees as nestings: Complexity, similarity, and consensus. *Journal of Classification*, 3(2):299–317, 1986.

3 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4^{th} Latin American Symposium on Theoretical Informatics* (LATIN 2000), volume 1776 of *LNCS*, pages 88–94. Springer-Verlag, 2000.

4 M. A. Bender and M. Farach-Colton. The Level Ancestor Problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.

5 M. G. B. Blum, O. François, and S. Janson. The mean, variance and limiting distribution of two statistics sensitive to phylogenetic tree balance. *The Annals of Applied Probability*, 16(4):2195–2214, 2006.

6 P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *Proceedings of the* 11^{th} *International Symposium on Algorithms and Data Structures* (WADS 2009), volume 5664 of *LNCS*, pages 98–109. Springer-Verlag, 2009.

7 K. Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.

8 D. Bryant. A classification of consensus methods for phylogenetics. In M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, B. Mirkin, and F. S. Roberts, editors, *Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society, 2003.

9 R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.

10 J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

11 J. Felsenstein. PHYLIP, version 3.6. Software package, Department of Genome Sciences, University of Washington, Seattle, U.S.A., 2005.

12 P. A. Goloboff, J. S. Farris, M. Källersjö, B. Oxelman, M. J. Ramírez, and C. A. Szumik. Improvements to resampling measures of group support. *Cladistics*, 19(4):324–332, 2003.

13 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

14 E. Hernández-García, M. Tuğrul, E. Alejandro Herrada, V. M. Eguíluz, and K. Klemm. Simple models for scaling in phylogenetic trees. *International Journal of Bifurcation and Chaos*, 20(3):805–811, 2010.

15 Z.-X. Luo, Q. Ji, J. R. Wible, and C.-X. Yuan. An early Cretaceous tribosphenic mammal and metatherian evolution. *Science*, 302(5652):1934–1940, 2003.

16 T. Margush and F. R. McMorris. Consensus n-Trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.

17 L. Nakhleh, T. Warnow, D. Ringe, and S. N. Evans. A comparison of phylogenetic reconstruction methods on an Indo-European dataset. *Transactions of the Philological Society*, 103(2):171–192, 2005.

18 R. Page. COMPONENT, version 2.0. Software package, University of Glasgow, U.K., 1993.

19 E. R. Seiffert. Revised age estimates for the later Paleogene mammal faunas of Egypt and Oman. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 103(13):5000–5005, 2006.

20 C. Semple and M. Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.

21 R. R. Sokal and F. J. Rohlf. Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology*, 30(3):309–325, 1981.

22 W.-K. Sung. *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC, 2010.

23 D. L. Swofford. PAUP*, version 4.0. Software package, Sinauer Associates, Inc., Sunderland, Massachusetts, 2003.

24 X. Xu, J. M. Clark, C. A. Forster, M. A. Norell, G. M. Erickson, D. A. Eberth, C. Jia, and Q. Zhao. A basal tyrannosauroid dinosaur from the Late Jurassic of China. *Nature*, 439(7077):715–718, 2006.