Information and Computation 256 (2017) 334–347

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

On finding the Adams consensus tree ☆

Jesper Jansson [a,*], Zhaoxian Li [b], Wing-Kin Sung [b,c]

[a] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong, China
[b] School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417, Singapore
[c] Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672, Singapore

ARTICLE INFO

Article history:
Received 17 September 2015
Received in revised form 3 July 2017
Available online 12 August 2017

Keywords:
Phylogenetic tree
Adams consensus
Centroid path
Wavelet tree

ABSTRACT

This article presents a fast algorithm for finding the Adams consensus tree of a set of conflicting phylogenetic trees with identical leaf labels. Its worst-case running time is $O(kn \log n)$, where k is the number of input trees and n is the size of the leaf label set; in comparison, the original algorithm of Adams has a worst-case running time of $O(kn^2)$. To achieve subquadratic running time, the centroid path decomposition technique is applied in a novel way that traverses the input trees by following a centroid path in each of them in unison. For $k = 2$, 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 of Bose et al. for orthogonal range counting on a grid. Our extended wavelet tree data structure also supports truncated range maximum/minimum queries efficiently.

1. Introduction

Scientists use *phylogenetic trees* to describe treelike evolutionary history [11,21,25,27]. 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 multiple data sets or obtained by resampling. 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 the surveys in [8], Chapter 30 in [11], or Chapter 8.4 in [27]. 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 COMPONENT [22], EPoS [14], and PAUP* [28]. Over the decades, many articles in biology have utilized the Adams consensus tree to reach their conclusions; some examples of highly cited ones include [19], [24], and [29], and more recently, [18] and [23].

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 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

☆ A preliminary version of this article appeared in J. Jansson, Z. Li, W.-K. Sung, On finding the Adams consensus tree, in: Proceedings of the 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, in: LIPIcs, vol. 30, Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, 2015, pp. 487–499.

* Corresponding author.
E-mail addresses: jesper.jansson@polyu.edu.hk (J. Jansson), lizhaoxianfgg@gmail.com (Z. Li), ksung@comp.nus.edu.sg (W.-K. Sung).

http://dx.doi.org/10.1016/j.ic.2017.08.002
0890-5401/

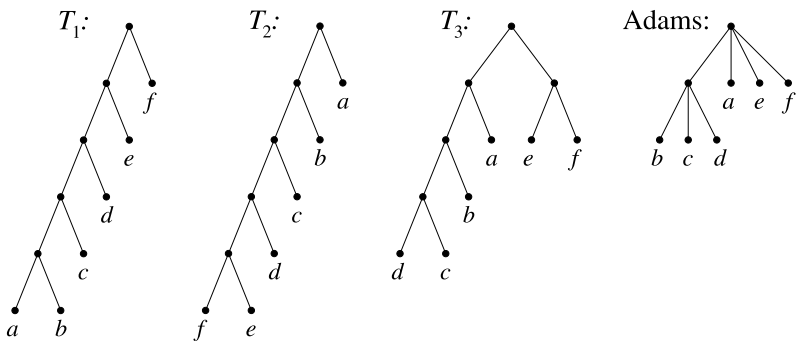

Fig. 1. An example. Let $\mathcal{S} = \{T_1, T_2, T_3\}$ as above with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$. The Adams consensus tree of $\mathcal{S}$ is shown on the right. Also note that in this particular example, the Adams consensus tree of $\mathcal{S}$ does not equal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

output [11], whereas the structure of the so-called *majority rule consensus tree* [20] or the *frequency difference consensus tree* [13] 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* [8,12]. Finally, it may be much more informative than the *strict consensus tree* [26] 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 [25]. Despite its practical usefulness, its running time has not been improved in the last forty years. The purpose of this article 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]. Section 4 describes an even faster method for the special case of $k = 2$ 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]. Finally, Section 5 presents a prototype implementation of our algorithm from Section 3 and discusses some experimental results.

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 with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$ 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 Fig. 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].

Given 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 $L = \Lambda(T_1)(= \Lambda(T_2) = \ldots = \Lambda(T_k))$. To express the time complexity of any algorithm computing the Adams consensus tree of $\mathcal{S}$, we define $k = |\mathcal{S}|$ and $n = |L|$.

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 recursion */
2: **else** /* General case of recursion */
3: $\pi :=$ `Compute_partition`$(\mathcal{S})$;
4: **for** $B \in \pi$ **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 all $B \in \pi$;
6: **end if**
7: **return** T;

Fig. 2. Algorithm `Old_Adams_consensus`, adapted from [1].

Procedure `Compute_partition`
Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L'$ such that $|L'| \geq 2$.
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;

Fig. 3. Procedure `Compute_partition`.

1.2. Previous work

The Adams consensus tree can be computed by the algorithm from [1], which we will now describe. From here on, this algorithm will be referred to as `Old_Adams_consensus`. The pseudocode is given in Fig. 2.

For any tree T with at least two leaves, define $\pi(T) = \{\Lambda(T^c) : c \in Child^T(r)$, where r is the root of $T\}$. Observe that $\pi(T)$ is a non-trivial partition of $\Lambda(T)$; i.e., $|\pi(T)| \geq 2$. Next, for any set of trees $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k) = L$ for a leaf label set L with $|L| \geq 2$, define $\pi(\mathcal{S})$ to be the product of the partitions $\pi(T_1), \pi(T_2), \ldots, \pi(T_k)$; i.e., $\pi(\mathcal{S})$ is the partition of L in which, for each part $B \in \pi(\mathcal{S})$, it holds that $B \neq \emptyset$ and $B = \bigcap_{j=1}^{k} \Lambda(T_j^{c_j})$ for some child c_j of the root of T_j for every $j \in \{1, 2, \ldots, k\}$. As an example, in Fig. 1, $\pi(T_1) = \big\{\{a, b, c, d, e\}, \{f\}\big\}$, $\pi(T_2) = \big\{\{a\}, \{b, c, d, e, f\}\big\}$, $\pi(T_3) = \big\{\{a, b, c, d\}, \{e, f\}\big\}$, and $\pi(\mathcal{S}) = \big\{\{a\}, \{b, c, d\}, \{e\}, \{f\}\big\}$.

To compute $\pi(\mathcal{S})$, one can apply Procedure `Compute_partition` in Fig. 3. 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 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 such vectors and scanning $\mathcal{W}$ to find all 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 [25], 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 $B \subseteq L$, B is called 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 $\mathcal{B} = \{B : B$ is a relevant block$\}$. For every $\ell \in L$, define $\mathcal{B}(\ell) = \{B \in \mathcal{B} : \ell \in B\}$. The next lemma gives a bound on $|B(\ell)|$.

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

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 1. `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 from 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. The same preprocessing works for all recursion levels and does not need to be repeated during recursive calls because for any $A \subsetneq B$, $(T_j|B)|A = T_j|A$ holds (to see this, observe that $lca^{T_j|B}(\{\ell_1, \ell_2\}) = lca^{T_j}(\{\ell_1, \ell_2\})$ for any $\ell_1, \ell_2 \in A$, so $V((T_j|B)|A) = V(T_j|A)$ and furthermore, $lca^{(T_j|B)|A}(A') = lca^{T_j|A}(A')$ for all nonempty $A' \subseteq A$, giving $(T_j|B)|A = T_j|A$ in accordance with the definitions in Section 1.1). 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 1 implies that the worst-case running time of `Old_Adams_consensus` is $O(kn^2)$, as already mentioned in [25]. However, if the average leaf depth is very small then the running time can be much less than that. According to Theorem 1, we obtain:

Corollary 1. *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 [5,16,25], the uniform model [5,25], and the activity model [16] (with the activity parameter p set to $\frac{1}{2}$) is $O(\log n)$ [5,16], $O(n^{1/2})$ [5], and $O(n^{1/2})$ [16], 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 provides 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 avoid making recursive calls to large subproblems, and treat them iteratively instead. For this purpose, we apply the centroid path decomposition technique [9] in a new manner. Essentially, by utilizing Lemma 2 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 2 at each level. (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.

Some definitions needed to describe the details of `New_Adams_consensus_k` are given in Section 3.1. Then, the algorithm is presented in Section 3.2 and its time complexity is analyzed in Section 3.3.

3.1. Additional definitions

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 p_α can be any node in T, p_{w-1} for every $w \in \{2, \ldots, \alpha\}$ is a child of p_w with the maximum number of leaf descendants (with ties broken arbitrarily), and p_1 is a leaf. For example, in Fig. 4, the path in the tree T_j from the root to the leaf i indicated by dashed lines is a centroid path in T_j because the child of the root lying on this path has eight leaf descendants while none of the root's other children have more than that, and similarly at each non-leaf node along the path. Next, 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. In Fig. 4, the indicated centroid path in T_j has seven side trees. 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. This property will be used in the proof of Theorem 2 to bound the time complexity of the new algorithm.

Suppose $|L| \geq 2$. As `Old_Adams_consensus` above, `New_Adams_consensus_k` will 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 $\pi(\mathcal{S}; X) = \{B \cap X : B \in \pi(\mathcal{S}) \text{ and } |B \cap X| \geq 1\}$. In other words, $\pi(\mathcal{S}; X)$ is the partition $\pi(\mathcal{S})$ restricted to elements in X. Note that

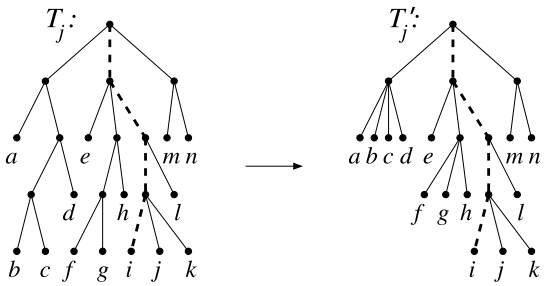

Fig. 4. A tree T_j and a centroid path in T_j (indicated by dashed lines) are shown on the left. Contracting the side trees of the given centroid path in T_j to fan trees yields the tree T'_j on the right.

$\pi(\mathcal{S}; X)$ may be the trivial partition of X as it can be a singleton. To continue the example from Fig. 1 in Section 1.2 where we had $\pi(\mathcal{S}) = \{\{a\}, \{b, c, d\}, \{e\}, \{f\}\}$, if $X = \{a, b, c\}$ then $\pi(\mathcal{S}; X) = \{\{a\}, \{b, c\}\}$ and if $X = \{b, c\}$ then $\pi(\mathcal{S}; X) = \{\{b, c\}\}$.

For each $j \in \{1, 2, \ldots, k\}$, let P_j be any centroid path in T_j that starts at the root of T_j. By taking the set of all leaves that belong to a topmost side tree of at least one P_j, the partition $\pi(\mathcal{S})$ can be expressed as a restricted partition as follows:

Lemma 2. *Let* $X = \{x \in L :$ *for some* $j \in \{1, 2, \ldots, k\}$, *x belongs to a side tree of* P_j *attached to the root of* $T_j\}$. *It holds that:*

- *If* $X \neq L$ *then* $\pi(\mathcal{S}) = \pi(\mathcal{S}; X) \cup \{L \setminus X\}$.
- *If* $X = L$ *then* $\pi(\mathcal{S}) = \pi(\mathcal{S}; X)$.

Proof. Consider any $B \in \pi(\mathcal{S})$. If B contains at least one element from X then $B \subseteq X$, and consequently $B \cap X = B$ and $B \in \pi(\mathcal{S}; X)$. On the other hand, if B contains no elements from X then B is equal to $L \setminus X$. Therefore, $\pi(\mathcal{S}) \subseteq \pi(\mathcal{S}; X) \cup \{L \setminus X\}$ when $X \neq L$, and $\pi(\mathcal{S}) \subseteq \pi(\mathcal{S}; X)$ when $X = L$.

To prove the other direction, consider any $B \in \pi(\mathcal{S}; X)$. By the definition of X, $B \in \pi(\mathcal{S})$. 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 $L \setminus X \in \pi(\mathcal{S})$. Thus, $\pi(\mathcal{S}; X) \cup \{L \setminus X\} \subseteq \pi(\mathcal{S})$ when $X \neq L$, and $\pi(\mathcal{S}; X) \subseteq \pi(\mathcal{S})$ when $X = L$. The lemma follows. □

Next, we define a tree T'_j for each $T_j \in \mathcal{S}$. We remark here that using the T'_j-trees does not increase or decrease the asymptotic time complexity, but greatly simplifies the description of the algorithm and its implementation. First, 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. For each $j \in \{1, 2, \ldots, k\}$, 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 the centroid path P_j; 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. See Fig. 4 for an illustration. The T'_j-tree is a useful summary of T_j that helps us to directly retrieve the leaf label set of any side tree of P_j or to check which side tree of P_j that a specified leaf belongs to in $O(1)$ time. In particular, Lemma 2 can be rephrased in terms of the T'_j-trees as in Corollary 2 below, which will be used in the new algorithm.

Corollary 2. *Let* $X = \{x \in L :$ *for some* $j \in \{1, 2, \ldots, k\}$, *x belongs to a fan tree attached to the root of* $T'_j\}$. *It holds that:*

- *If* $X \neq L$ *then* $\pi(\mathcal{S}) = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X) \cup \{L \setminus X\}$.
- *If* $X = L$ *then* $\pi(\mathcal{S}) = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X)$.

Proof. The set X defined in Lemma 2 is equal to $\{x \in L :$ for some $j \in \{1, 2, \ldots, k\}$, x belongs to a fan tree attached to the root of $T'_j\}$. Next, note that $\pi(\{T'_1, T'_2, \ldots, T'_k\}; X) = \pi(\{T_1, T_2, \ldots, T_k\}; X)$ by the construction of the T'_j-trees. □

3.2. Algorithm description

We now present `New_Adams_consensus_k`. Refer to Fig. 5 for the pseudocode.

The base case in which there is only one leaf is handled in Steps 1–2. In the general case, Steps 4–6 first build P_j and T'_j for every $j \in \{1, 2, \ldots, k\}$. The algorithm then enters a **repeat**-loop (Steps 8–13) that initially 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 Corollary 2.

Algorithm `New_Adams_consensus_k`
Input: A set $\mathcal{S} = \{T_1, T_2, \ldots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \ldots = \Lambda(T_k)$.
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, and construct the tree T'_j based on P_j;
6: **end for**
7: $h := 0$;
8: **repeat**
9: $h := h + 1$;
10: $X_h := \{x \in L : \text{ for some } j \in \{1, 2, \ldots, k\}, \ x \text{ belongs to a fan tree attached to the root of } T'_j\}$;
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: Construct $T_j|B$ for all $B \in \bigcup_{w=1}^{h} \pi_{X_w}$;
16: **end for**
17: **for** $w := h$ **downto** 1 **do**
18: **for** $B \in \pi_{X_w}$ **do**
19: $T_B :=$ `New_Adams_consensus_k`$(\{T_1|B, T_2|B, \ldots, T_k|B\})$;
20: **end for**
21: Create a tree Q_w whose root is the parent of the root of every T_B, where $B \in \pi_{X_w}$;
22: **if** $w < h$ **then** attach the root of Q_{w+1} as a child of the root of Q_w;
23: **end for**
24: $T := Q_1$;
25: **end if**
26: **return** T;

Fig. 5. Algorithm `New_Adams_consensus_k`.

(By Corollary 2, 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.) To obtain $\pi(\{T'_1, T'_2, \ldots, T'_k\}; X_1)$, the algorithm uses a procedure called `Compute_restricted_partition` that is identical to Procedure `Compute_partition` in Fig. 3 except that it only computes and sorts the vectors $(m_1(\ell), m_2(\ell), \ldots, m_k(\ell))$ for those $\ell \in L'$ that belong to X_1 rather than of all $\ell \in L'$ in the second and third steps. 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 solution 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 level w (Steps 18–20), combining the obtained solutions (Step 21), and attaching the Adams consensus tree Q_{w+1} for the part corresponding to $L \setminus X_w$ in Corollary 2 (Step 22); Corollary 2 ensures that Q_w will indeed get the same structure as the tree output by `Old_Adams_consensus` on this level. Lastly, the tree Q_1 obtained at the topmost level is returned (Step 26). The correctness follows from the correctness of `Old_Adams_consensus`.

3.3. Time complexity

The time complexity of the algorithm is given by the next theorem:

Theorem 2. `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 for each $j \in \{1, 2, \ldots, k\}$ in $O(k|L'|)$ total time. Next, iteration h of the **repeat**-loop computes a set X_h in Step 10 and the restricted partition $\pi_{X_h} = \pi(\{T'_1, T'_2, \ldots, T'_k\}; X_h)$ of X_h in Step 11, both in $O(k|X_h|)$ time. The former is accomplished by examining the T'_j-trees, and the latter by computing and radix sorting the vectors $(m_1(\ell), m_2(\ell), \ldots, m_k(\ell))$ for $\ell \in X_h$ as in Procedure `Compute_partition` in Fig. 3 except that only $\ell \in X_h$ are considered and each $m_j(\ell)$ is recovered directly from T'_j in $O(1)$ time. To implement Step 12 in $O(k|X_h|)$ time, update every T'_j-tree 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 Lemma 5.2 from [10] (alternatively, its generalization in Section 8 of [9] can also be used). Finally, for each $w \in \{1, 2, \ldots, h\}$, the recursive

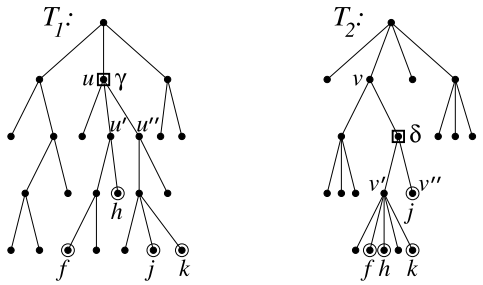

Fig. 6. Illustrating Lemma 3. Let $u \in V(T_1)$ and $v \in V(T_2)$ be two nodes as shown above and suppose $B = \Lambda(T_1^u) \cap \Lambda(T_2^v) = \{f, h, j, k\}$. Then $Z_{\gamma,\delta} = \{(u', v'), (u'', v'), (u'', v'')\}$, corresponding to the three parts $\{f, h\}$, $\{k\}$, and $\{j\}$, respectively, in $\pi(\{T_1|B, T_2|B\})$.

calls in Steps 18–20 take $\sum_{B\in\pi_{X_w}} t(B)$ time and building Q_w in Steps 21 and 22 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 on each recursion level is $O(|L'|)$, so each level takes $O(k|L'|)$ time (excluding its recursive calls). 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 $O(n \cdot \frac{\log n}{\log\log n})$. It is described in Section 4.1 and its time complexity is analyzed in Section 4.3. The algorithm relies on an efficient data structure for orthogonal range counting on a grid, developed in Section 4.2.

4.1. Outline of the new algorithm

Consider any recursive call to the original Adams consensus tree algorithm reviewed in Section 1.2. It has the form `Old_Adams_consensus`$(\{T_1|B, T_2|B\})$ for some $B \subseteq L$. The algorithm will spend $\Omega(|B|)$ time to obtain $\{T_1|B, T_2|B\}$ and to compute the partition $\pi(\{T_1|B, T_2|B\})$ of the leaves in B with the procedure `Compute_partition`. The new algorithm in this section avoids constructing $\{T_1|B, T_2|B\}$ and employs a faster method for doing the partitioning, thereby improving 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 nodes from T_1 and T_2. Secondly, we observe that the algorithm does not need to proceed recursively from (u, v) 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 two 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 3. *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 $\{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : u' \in Child^{T_1}(\gamma),\ v' \in Child^{T_2}(\delta),\ \Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) \neq \emptyset\} = \{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : (u', v') \in Z_{\gamma,\delta}\}$. □

See Fig. 6 for an example.

Algorithm `New_Adams_consensus_2`, summarized in Fig. 7, uses Lemma 3 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 all of the computed $T_{u',v'}$-trees to a newly created common root node. Lemma 3 implies that this gives the same output as `Old_Adams_consensus`, so the correctness of the new algorithm is guaranteed by the correctness of `Old_Adams_consensus`.

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: $\gamma := lca^{T_1}(\Lambda(T_1^u) \cap \Lambda(T_2^v))$; $\delta := lca^{T_2}(\Lambda(T_1^u) \cap \Lambda(T_2^v))$;
6: $Z :=$ `Compute_Z`(γ, δ);
7: Let T be a tree consisting of a root node r;
8: **for** every $(u', v') \in Z$ **do**
9: $T_{u',v'} :=$ `New_Adams_consensus_2`(u', v');
10: Attach $T_{u',v'}$ as a child of r;
11: **end for**
12: **end if**
13: **return** T;

Fig. 7. Algorithm `New_Adams_consensus_2`.

4.2. Auxiliary data structure for orthogonal range counting on a grid

This subsection presents an extension of the wavelet tree-based data structure by Bose et al. [6] for storing a set of 2d-points lying on a grid while supporting *orthogonal range counting queries* (outputting the number of points in any query rectangle) and *orthogonal range reporting queries* (outputting the set of all points in any query rectangle). 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 $[\ell_1..\ell_2] \times [s_1..s_2]$, if any. Furthermore, we bound the time needed to *construct* the data structure since this is crucial in our application. The main result of this subsection is summarized in Theorem 3.

Firstly, for smaller grids, we have:

Lemma 4. *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) reporting the number of points inside any query rectangle $[1..\ell] \times [s_1..s_2]$ takes $O(1)$ time; and (ii) reporting the point with the maximum x-coordinate inside any query rectangle $[1..\ell] \times [s_1..s_2]$, if any, takes $O(1)$ time.*

Proof. We first describe the data structure for (i), i.e., orthogonal range counting queries, analogous to Lemma 3 in [6]. For every $i = 1, \ldots, \frac{n}{\log^2 n}$, we store $C_i[s, s'] =$ the number of points in $[1..i\log^2 n] \times [s..s']$ for all $1 \le s \le s' \le t$. Since there are $\frac{n}{\log^2 n}$ possible indices for i, t possible indices for s, and t possible indices for s', there are at most $\frac{nt^2}{\log^2 n}$ possible entries in the table $C_i[s, s']$. Since the number of points is upper-bounded by n, each entry $C_i[s, s']$ can be represented in $O(\log n)$ bits. In total, all entries $C_i[s, s']$ require $O(\frac{n \log^{1+2\epsilon} n}{\log^2 n}) = O(n)$ bits. To compute C_i, define $NP_i[s] =$ the number of occurrences of s in $\{N[1], \ldots, N[i\log^2 n]\}$ for $1 \le s \le t$. For convenience, also define $NP_0[s] = 0$ for $1 \le s \le t$. We can obtain NP_i iteratively for $i = 1, \ldots, \frac{n}{\log^2 n}$ by first computing the number of occurrences of s in $\{N[(i-1)\log^2 n + 1], \ldots, N[i\log^2 n]\}$ for $1 \le s \le t$ in $O(t + \log^2 n)$ time. Then, $NP_i[s] = NP_{i-1}[s] +$ the number of occurrences of s in $\{N[(i-1)\log^2 n + 1], \ldots, N[i\log^2 n]\}$. Hence, $NP_i[s]$ can be computed using additional $O(t)$ time. In total, NP_i for all $i \in \{1, \ldots, \frac{n}{\log^2 n}\}$ can be computed in $O((t + \log^2 n)\frac{n}{\log^2 n}) = O(n)$ time. Since $C_i[s, s'] = \sum_{s \le s'' \le s'} NP_i[s'']$, each C_i-table can be obtained using $O(t^2)$ time. In total, all C_i-tables can be computed in $O((t^2)\frac{n}{\log^2 n} + n) = O(n)$ time.

For every $i = 1, \ldots, \frac{n}{\log^2 n}$ and every $j = 1, \ldots, \log n \log\log n$, we also store $C_{ij}[s, s'] =$ the number of points in $[(i-1)\log^2 n + 1..(i-1)\log^2 n + j\frac{\log n}{\log\log n}] \times [s..s']$ for all $1 \le s \le s' \le t$. There are $\frac{n}{\log^2 n}$ possible indices for i, $\log n \log\log n$ possible indices for j, t possible indices for s and t possible indices for s', so there are at most $\frac{nt^2 \log n \log\log n}{\log^2 n}$ entries in the table $C_i[s, s']$. Since the number of points is upper-bounded by $\log^2 n$, each entry $C_{ij}[s, s']$ can be represented in $O(\log\log n)$ bits. In total, all entries $C_{ij}[s, s']$ require $O(\frac{n(\log^{2\epsilon} n)(\log\log n)^2}{\log n}) = O(n)$ bits. Similar to the above, to compute C_{ij}, we first compute NP_{ij}, defined by $NP_{ij}[s] =$ the number of occurrences of s in $\{N[(i-1)\log^2 n + 1], \ldots, N[(i-1)\log^2 n + j\frac{\log n}{\log\log n}]\}$ for $1 \le s \le t$. Also define $NP_{i0}[s] = 0$ for $1 \le s \le t$. For each i, we build NP_{ij} iteratively for $j = 1, \ldots, \log n \log\log n$. To compute NP_{ij}, we first compute the number of occurrences of s in $\{N[(i-1)\log^2 n + (j-1)\frac{\log n}{\log\log n} + 1], \ldots, N[(i-1)\log^2 n + j\frac{\log n}{\log\log n}]\}$ for $1 \le s \le t$. This can be done in $O(t + \frac{\log n}{\log\log n})$ time. Then, $NP_{ij}[s] = NP_{i(j-1)}[s] +$ the number of occurrences of s in $\{N[(i-1)\log^2 n + (j-1)\frac{\log n}{\log\log n} + 1], \ldots, N[(i-1)\log^2 n + j\frac{\log n}{\log\log n}]\}$. Hence, $NP_{ij}[s]$ for all s can be computed in $O(t)$ time. In

total, NP_{ij}, for $i = 1, \ldots, \frac{n}{\log^2 n}$ and $j = 1, \ldots, \log n \log\log n$, can be computed in $O((t + \frac{\log n}{\log\log n})(\log n \log\log n)\frac{n}{\log^2 n}) = O(n)$ time. Since $C_{ij}[s, s'] = \sum_{s \le s'' \le s'} NP_{ij}[s'']$, the table C_{ij} can be computed using $O(t^2)$ time for each i, j. In total, all entries $C_{ij}[s, s']$ can be computed in $O((t^2)(\log n \log\log n)\frac{n}{\log^2 n} + n) = O(n)$ time.

Furthermore, we precompute a table range$(x_1, \ldots, x_\ell, s, s')$ which stores $|\{x_i : s \le x_i \le s'\}|$, where $1 \le x_i \le t$, $1 \le s \le s' \le t$, $\ell \le \frac{\log n}{\log\log n}$. The table has $\sum_{\ell=1}^{\frac{\log n}{\log\log n}} t^{\ell+2} = O(n^\epsilon)$ entries, where each entry is represented in $\log\log n$ bits, so it can be constructed in $O(n)$ time and stored in $O(n)$ space.

Now, for any query rectangle $[1..\ell] \times [s_1..s_2]$, where $1 \le \ell \le n$ and $1 \le s_1 \le s_2 \le t$, we first find $1 \le i \le \frac{n}{\log^2 n}$, $1 \le j \le \log n \log\log n$, and $0 \le k \le \frac{\log n}{\log\log n} - 1$ such that $\ell = (i-1)\log^2 n + (j-1)\frac{\log n}{\log\log n} + k$. Then, the range count for the query rectangle $[1..\ell] \times [s_1..s_2]$ equals $C_{i-1}[s_1, s_2] + C_{i(j-1)}[s_1, s_2] + \text{range}(N[\ell - k], \ldots, N[\ell], s_1, s_2)$, which can be obtained in $O(1)$ time.

To support (ii), i.e., truncated range maximum queries, we augment the data structure as follows. For every $i = 1, \ldots, \frac{n}{\log^2 n}$ and $1 \le s \le s' \le t$, define $S_i[s, s']$ to be x if x is the biggest index smaller than $i\log^2 n$ such that $s \le N[x] \le s'$; otherwise, set $S_i[s, s'] = -\infty$. For every $j = 1, \ldots, \log n \log\log n$, define $S_{ij}[s, s']$ to be $x - (i-1)\log^2 n$ if x is the biggest index in the range $[(i-1)\log^2 n + 1..(i-1)\log^2 n + j\frac{\log n}{\log\log n}]$ such that $s \le N[x] \le s'$; otherwise, set $S_{ij}[s, s'] = -\infty$. S_i and S_{ij} can be constructed in $O(n)$ time. Also precompute a table maxi$(x_1, \ldots, x_\ell, s, s')$ which stores the biggest index j such that $s \le x_j \le s'$ where $1 \le x_i \le t$, $1 \le s \le s' \le t$, $\ell \le \frac{\log n}{\log\log n}$. This table has $\sum_{\ell=1}^{\frac{\log n}{\log\log n}} t^{\ell+2} = O(n^\epsilon \log^2 n)$ entries and each entry is represented in $\log\log n$ bits. Since $\epsilon < 1/2$, the table can be constructed in $O(n)$ time and stored in $O(n)$ space.

Then, for any query rectangle $[1..\ell] \times [s_1..s_2]$, where $1 \le \ell \le n$ and $1 \le s_1 \le s_2 \le t$, we first find $1 \le i \le \frac{n}{\log^2 n}$, $1 \le j \le \log n$ and $0 \le k \le \log n - 1$ such that $\ell = (i-1)\log^2 n + (j-1)\frac{\log n}{\log\log n} + k$. Using the precomputed table, $\max\{x : s_1 \le N[x] \le s_2,\ \ell - k < x \le \ell\}$ can be retrieved in $O(1)$ time. Then, the point with the maximum x-coordinate inside the rectangle $[1..\ell] \times [s_1..s_2]$ equals $\max\{S_{i-1}[s_1, s_2], S_{i(j-1)}[s_1, s_2] + (i-1)\log^2 n, \ell - k + \text{maxi}(N[\ell - k + 1], \ldots, N[\ell], s_1, s_2)\}$, which can be obtained in constant time. □

The next lemma uses the data structure from Lemma 4 to answer slightly more general queries.

Lemma 5. *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) reporting the number of points inside any query rectangle $[\ell_1..\ell_2] \times [s_1..s_2]$ takes $O(1)$ time; and (ii) reporting the point with the maximum or minimum x-coordinate inside any query rectangle $[\ell_1..\ell_2] \times [s_1..s_2]$, if any, takes $O(1)$ time.*

Proof. Use Lemma 4 on $N = \{(1, N[1]), \ldots, (n, N[n])\}$ directly, and also on $N^R = \{(1, N[n]), \ldots, (n, N[1])\}$, the sequence of points in reverse order. Assume without loss of generality that $1 \le \ell_1 \le \ell_2 \le n$ and $1 \le s_1 \le s_2 \le t$ for any given query rectangle $[\ell_1..\ell_2] \times [s_1..s_2]$. To support (i) in $O(1)$ time, apply Lemma 4 to find the number of points inside $[1..\ell_2] \times [s_1..s_2]$ in N and the number of points inside $[1..\ell_1 - 1] \times [s_1..s_2]$ in N, and return the difference. To support (ii) in $O(1)$ time, do the following. If the query is asking for the maximum then apply Lemma 4 to retrieve the point x with the maximum x-coordinate in $[1..\ell_2] \times [s_1..s_2]$ in N; if $x \ge \ell_1$ then x is the answer, and otherwise no such point exists. If the query is asking for the minimum then retrieve the point x^R with the maximum x-coordinate in $[1..(n + 1 - \ell_1)] \times [s_1..s_2]$ in N^R; if $n + 1 - x^R \le \ell_2$ then $n + 1 - x^R$ is the answer, and otherwise no such point exists. □

For larger grids, we apply Lemma 5 to obtain:

Theorem 3. *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) reporting the number of points inside any query rectangle $[\ell_1..\ell_2] \times [s_1..s_2]$ 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 $[\ell_1..\ell_2] \times [s_1..s_2]$, if any, 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 5 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 $rank_j(i) - rank_j(\log^2 n \lfloor \frac{i}{\log^2 n} \rfloor)$ for every i which is a multiple of $\frac{\log n}{\log\log n}$, requiring $O(t\frac{n}{\log n}\frac{\log\log n}{\log n}\log\log nt) = O(n)$ 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 5 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:

(A) $[\ell_1..\ell_2] \times [s_1..(n/t)z_1]$;
(B) $[\ell_1..\ell_2] \times [(n/t)z_1 + 1..(n/t)z_2]$; and
(C) $[\ell_1..\ell_2] \times [(n/t)z_2 + 1..s_2]$.

Query (B) 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 5. 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 (A) 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 (C) 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

Finally, we analyze the time complexity of `New_Adams_consensus_2` in Fig. 7. We use 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 an arbitrary left-to-right ordering of the children at every node in T_2. Let $N = \{(L_1^{-1}(\ell), L_2^{-1}(\ell)) : \ell \in L\}$ and build the data structure $D(N)$ from Theorem 3.
- 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 [3,15].
- As in the proof of Theorem 1 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.

The preprocessing takes $O(n \cdot \frac{\log n}{\log\log n})$ time in total. From here on, we will assume that the preprocessing has already been taken care of.

Next, any $u, u' \in V(T_j)$ where $j \in \{1, 2\}$ such that u and u' have the same parent (possibly with $u = u'$) are called *siblings*. 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 6. *Given the data structure $D(N)$ in Theorem 3, 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 Theorem 3, the lemma follows. □

Lemma 6 allows the $Z_{u,v}$-sets to be computed in an efficient manner by the procedure `Compute_Z` shown in Fig. 8. More precisely:

Lemma 7. *Given the data structure $D(N)$ in Theorem 3, `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.*

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** $i \leq \alpha$ **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** $j \leq \beta$ **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;

Fig. 8. Procedure `Compute_Z`.

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 6 in Step 4 and the level 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 analyze the time complexity of `New_Adams_consensus_2`. For any $u \in V(T_1)$, $v \in V(T_2)$, denote $S_{u,v} = \Lambda(T_1^u) \cap \Lambda(T_2^v)$.

Lemma 8. *Given the data structure $D(N)$ in Theorem 3, the running time of* `New_Adams_consensus_2`(u, v) *for any $u \in V(T_1)$ and $v \in V(T_2)$ is $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$.*

Proof. Let $T(u, v)$ be the total running time of `New_Adams_consensus_2`(u, v), including the time required to compute γ, δ, and $Z_{\gamma,\delta}$ as well as the recursive calls `New_Adams_consensus_2`(u', v') for all $(u', v') \in Z_{\gamma,\delta}$. By using the first part of Lemma 6, Step 1 can be carried out in $O(\frac{\log n}{\log\log n})$ time. To compute γ and δ in $O(\frac{\log n}{\log\log n})$ time in Step 5, first apply the second part of Lemma 6 to find the leftmost leaf a and the rightmost leaf a' in T_1 among all leaves in $\Lambda(T_1^u) \cap \Lambda(T_2^v)$ and the leftmost leaf b and the rightmost leaf b' in T_2 among all leaves in $\Lambda(T_1^u) \cap \Lambda(T_2^v)$. Then, compute and assign $\gamma := lca^{T_1}(a, a')$ and $\delta := lca^{T_2}(b, b')$, which takes $O(1)$ time because of the *lca*-preprocessing. Step 6 takes $O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log\log n})$ time according to Lemma 7. 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 $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. Rewrite the cost of each internal node x in the recursion tree as $O(\deg(x) \cdot \frac{\log n}{\log\log n})$, where $\deg(x)$ is the degree of x. Then, 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, we see that the contribution of the internal nodes is also $O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. In total, the running time is $T(u, v) = O(|S_{u,v}| \cdot \frac{\log n}{\log\log n})$. □

Theorem 4. *Algorithm* `New_Adams_consensus_2` *computes the Adams consensus tree of T_1 and T_2 in $O(n \cdot \frac{\log n}{\log\log n})$ time.*

Proof. Recall that $D(N)$ is constructed during the preprocessing phase using $O(n \cdot \frac{\log n}{\log \log n})$ time. Lemma 8 shows 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 $O(|S_{r_1,r_2}| \cdot \frac{\log n}{\log \log n}) = O(n \cdot \frac{\log n}{\log \log n})$ time. □

5. Experiments

We have implemented two algorithms for constructing the Adams consensus tree: Adam's [1] algorithm `Old_Adams_consensus` from 1972 described in Section 1.2 and our new algorithm `New_Adams_consensus_k` from Section 3. (We have not implemented Algorithm `New_Adams_consensus_2` from Section 4 for the special case of $k = 2$.) A series of experiments were performed to compare the running times of these two algorithms, as described below.

The implementations were written in the C++ programming language. Our prototype implementations are publicly available in the FACT (Fast Algorithms for Consensus Trees) package [17] at: http://compbio.ddns.comp.nus.edu.sg/~consensus.tree/ FACT also contains implementations of the algorithms presented in [17] for three other types of consensus trees: the majority rule consensus tree [20], the loose consensus tree [7], and the greedy consensus tree [8,12].

5.1. Setup

All experiments were done on a MacBook Pro computer with a 2.3 GHz Intel i7 processor and 8 GB memory. The compiler g++ 4.8.4 was used to compile the source code. For various specified values of (k, n), we generated 10 sets of k random trees over the leaf label set $\{1, 2, \ldots, n\}$, ran our C++-implementations of `Old_Adams_consensus` and `New_Adams_consensus_k`, and measured the running times in seconds. Two types of inputs were considered:

- Random binary trees generated in the *Yule-Harding model* [5,16,25] of evolution (starting from an unlabeled tree consisting of a singleton leaf, repeatedly select one leaf in the current tree uniformly at random and replace it by an internal node whose two children are leaves until there are n unlabeled leaves; after that, assign the n leaf labels uniformly at random to the n leaves).
- Random binary caterpillars, where a tree is a *caterpillar* if every node has at most one child that is an internal node (for example, trees T_1 and T_2 in Fig. 1 are caterpillars).

5.2. Results

The worst-case (taken over 10 independent trials for each considered value of (k, n)) running times of our implementations of `Old_Adams_consensus` and `New_Adams_consensus_k` are reported in Fig. 9. Plots for a fixed value of k ($k = 1000$) and varying values of n, as well as for a fixed value of n ($n = 2000$) and varying values of k, are shown for the two types of inputs.

For random binary trees generated in the Yule-Harding model, the experiments show that `New_Adams_consensus_k` is faster than `Old_Adams_consensus` for all large enough n. In the top left plot in Fig. 9, corresponding to $k = 1000$, this happens for $n \geq 500$. We remark that by Corollary 1 and the comments that follow it, the expected running time of `Old_Adams_consensus` for inputs generated in the Yule-Harding model is $O(kn \log n)$, so the $O(kn \log n)$-time bound for `New_Adams_consensus_k` given in Theorem 2 may be overly pessimistic in practice. On the other hand, for small inputs (e.g., inputs with $k = 1000$ and $n = 200$), `Old_Adams_consensus` is faster than `New_Adams_consensus_k` due to the overhead involved in the centroid path computations.

For random binary caterpillars, `New_Adams_consensus_k` shows a substantial improvement in the running time over `Old_Adams_consensus`. This is because all internal nodes in any caterpillar will belong to a single centroid path, so `New_Adams_consensus_k` always needs just one level of recursion. In contrast, a binary caterpillar with n leaves has height $n - 1$, forcing `Old_Adams_consensus` to do $\Omega(n)$ levels of recursion in the worst case since each level satisfying $\pi(T_1) = \pi(T_2) = \cdots = \pi(T_k)$ reduces the number of leaves in the remaining leaf label set by one (or two, at the bottommost level). Thus, binary caterpillars are a kind of worst-case scenario for `Old_Adams_consensus` and a best-case scenario for `New_Adams_consensus_k`. Although caterpillars are unlikely to arise in real-world evolutionary data, the results above suggest that the centroid path-based technique may be especially powerful for "skewed" inputs containing long paths.

5.3. Other software

Methods for constructing the Adams consensus tree also exist in the software packages COMPONENT [22], PAUP* [28], and EPoS [14], However, these are based on Algorithm `Old_Adams_consensus` and hence not very efficient for large inputs. For example, in EPoS, inputs with $(k, n) = (100, 100)$ typically take a minute or more to run, whereas our prototype implementation of `New_Adams_consensus_k` takes less than 0.1 second. Also, inputs with large n such as $(k, n) = (10, 1000)$ often lead to stack overflow error messages in EPoS, while our implementation works without problems.

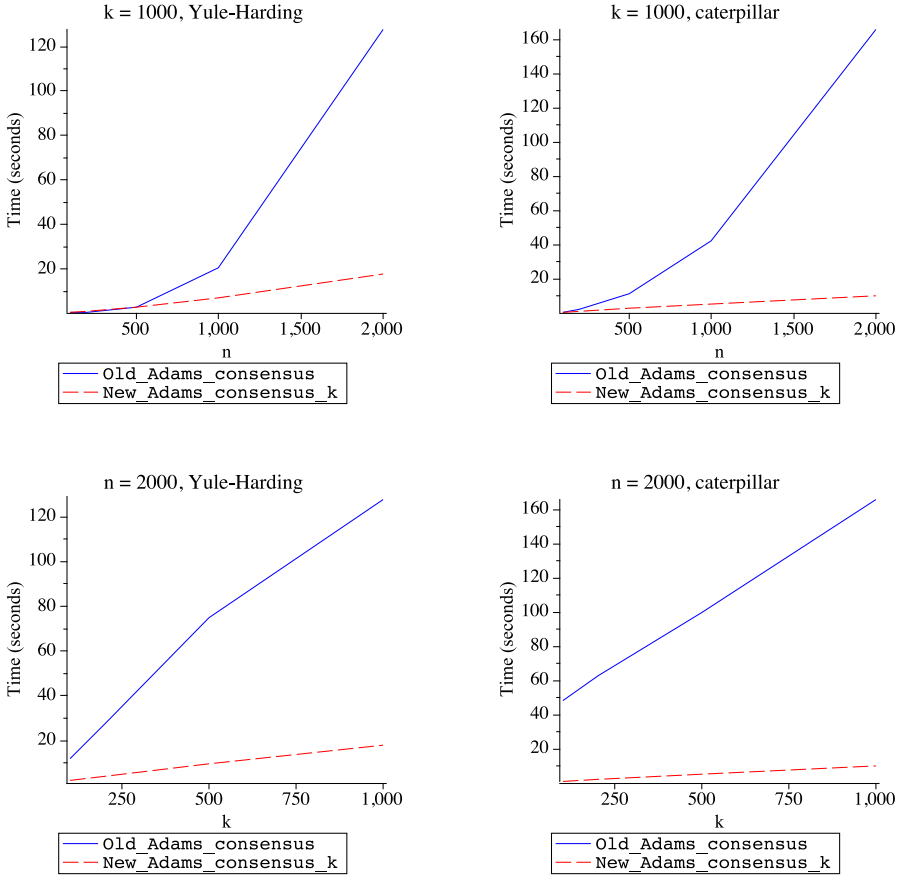

Fig. 9. The worst-case running times of our C++-implementations of `Old_Adams_consensus` and `New_Adams_consensus_k` for $k = 1000$ and varying values of n (top row), for $n = 2000$ and varying values of k (bottom row), for random binary trees generated in the Yule-Harding model (left column), and for random binary caterpillars (right column).

6. Final remark

The only known lower bound on the computational complexity of constructing the Adams consensus tree is the trivial one of $\Omega(kn)$, corresponding to the size of the input. An interesting open problem is to determine whether this is tight or not.

Acknowledgments

The authors would like to thank the anonymous reviewers for their suggestions. J.J. was partially funded by The Hakubi Project at Kyoto University and KAKENHI grant number 26330014.

References

[1] E.N. Adams III, Consensus techniques and the comparison of taxonomic trees, Syst. Zool. 21 (4) (1972) 390–397.
[2] E.N. Adams III, N-trees as nestings: complexity, similarity, and consensus, J. Classif. 3 (2) (1986) 299–317.
[3] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: Proceedings of the 4th Latin American Symposium on Theoretical Informatics, LATIN 2000, in: Lect. Notes Comput. Sci., vol. 1776, Springer-Verlag, 2000, pp. 88–94.
[4] M.A. Bender, M. Farach-Colton, The Level Ancestor Problem simplified, Theor. Comput. Sci. 321 (1) (2004) 5–12.
[5] M.G.B. Blum, O. François, S. Janson, The mean, variance and limiting distribution of two statistics sensitive to phylogenetic tree balance, Ann. Appl. Probab. 16 (4) (2006) 2195–2214.
[6] P. Bose, M. He, A. Maheshwari, P. Morin, Succinct orthogonal range search structures on a grid with applications to text indexing, in: Proceedings of the 11th International Symposium on Algorithms and Data Structures, WADS 2009, in: Lect. Notes Comput. Sci., vol. 5664, Springer-Verlag, 2009, pp. 98–109.
[7] K. Bremer, Combinable component consensus, Cladistics 6 (4) (1990) 369–372.
[8] D. Bryant, A classification of consensus methods for phylogenetics, in: M.F. Janowitz, F.-J. Lapointe, F.R. McMorris, B. Mirkin, F.S. Roberts (Eds.), Bioconsensus, in: DIMACS Ser. Discret. Math. Theor. Comput. Sci., vol. 61, American Mathematical Society, 2003, pp. 163–184.
[9] R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, M. Thorup, An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees, SIAM J. Comput. 30 (5) (2000) 1385–1404.

[10] M. Farach, M. Thorup, Fast comparison of evolutionary trees, Inf. Comput. 123 (1) (1995) 29–37.
[11] J. Felsenstein, Inferring Phylogenies, Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.
[12] J. Felsenstein, PHYLIP, Version 3.6, Software package, Department of Genome Sciences, University of Washington, Seattle, USA, 2005.
[13] P.A. Goloboff, J.S. Farris, M. Källersjö, B. Oxelman, M.J. Ramírez, C.A. Szumik, Improvements to resampling measures of group support, Cladistics 19 (4) (2003) 324–332.
[14] T. Griebel, M. Brinkmeyer, S. Böcker, EPoS: a modular software framework for phylogenetic analysis, Bioinformatics 24 (20) (2008) 2399–2400.
[15] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.
[16] E. Hernández-García, M. Tuğrul, E. Alejandro Herrada, V.M. Eguíluz, K. Klemm, Simple models for scaling in phylogenetic trees, International Journal of Bifurcation and Chaos 20 (3) (2010) 805–811.
[17] J. Jansson, C. Shen, W.-K. Sung, Improved algorithms for constructing consensus trees, J. ACM 63 (3) (2016) 28.
[18] M.N.-Y. Leung, C.A. Paszkowski, A.P. Russell, Genetic structure of the endangered Greater Short-horned Lizard (Phrynosoma hernandesi) in Canada: evidence from mitochondrial and nuclear genes, Can. J. Zool. 92 (10) (2014) 875–883.
[19] Z.-X. Luo, Q. Ji, J.R. Wible, C.-X. Yuan, An early Cretaceous tribosphenic mammal and metatherian evolution, Science 302 (5652) (2003) 1934–1940.
[20] T. Margush, F.R. McMorris, Consensus n-trees, Bull. Math. Biol. 43 (2) (1981) 239–244.
[21] L. Nakhleh, T. Warnow, D. Ringe, S.N. Evans, A comparison of phylogenetic reconstruction methods on an Indo-European dataset, Trans. Philol. Soc. 103 (2) (2005) 171–192.
[22] R. Page, COMPONENT, Version 2.0, Software package, University of Glasgow, UK, 1993.
[23] J.C. Regier, C. Mitter, A. Zwick, A.L. Bazinet, M.P. Cummings, A.Y. Kawahara, J.-C. Sohn, D.J. Zwickl, S. Cho, D.R. Davis, J. Baixeras, J. Brown, C. Parr, S. Weller, D.C. Lees, K.T. Mitter, A large-scale, higher-level, molecular phylogenetic study of the insect order Lepidoptera (moths and butterflies), PLoS ONE 8 (3) (2013) e58568.
[24] E.R. Seiffert, Revised age estimates for the later Paleogene mammal faunas of Egypt and Oman, Proc. Natl. Acad. Sci. USA 103 (13) (2006) 5000–5005.
[25] C. Semple, M. Steel, Phylogenetics, Oxf. Lect. Ser. Math. Appl., vol. 24, Oxford University Press, 2003.
[26] R.R. Sokal, F.J. Rohlf, Taxonomic congruence in the Leptopodomorpha re-examined, Syst. Zool. 30 (3) (1981) 309–325.
[27] W.-K. Sung, Algorithms in Bioinformatics: A Practical Introduction, Chapman & Hall/CRC, 2010.
[28] D.L. Swofford, PAUP* Version 4.0, Software package, Sinauer Associates, Inc., Sunderland, Massachusetts, 2003.
[29] X. Xu, J.M. Clark, C.A. Forster, M.A. Norell, G.M. Erickson, D.A. Eberth, C. Jia, Q. Zhao, A basal tyrannosauroid dinosaur from the Late Jurassic of China, Nature 439 (7077) (2006) 715–718.