A Faster Algorithm for Computing the Kernel of Maximum Agreement Subtrees

Biing-Feng Wang, Krister M. Swenson

A Faster Algorithm for Computing the Kernel of Maximum Agreement Subtrees

Biing-Feng Wang and Krister M. Swenson

Abstract—The maximum agreement subtree method determines the consensus of a collection of phylogenetic trees by identifying maximum cardinality subsets of leaves for which all input trees agree. The trees induced by these maximum cardinality subsets are maximum agreement subtrees (MASTs). A single MAST may be misleading, since there can exist two MASTs which share almost no leaves; nevertheless, it may be impossible to inspect all MASTs, since the number of MASTs can be exponential in the number of leaves. To overcome this drawback, Swenson *et al.* suggested to further summarize the information common to all MASTs by their intersection, which is called the kernel agreement subtree (KAST). The construction of the KAST is the focus of this paper. Swenson *et al.* had an $O(kn^3 + n^4 + n^{d+1})$ time algorithm for computing the KAST of k trees on n leaves, in which at least one tree has maximum degree d. In this paper, an $O(kn^3 + n^d)$-time algorithm is presented. We demonstrate the efficiency of our algorithm on simulated trees as well as on ribosomal RNA alignments, where trees with 13,000 taxa took only hours to process, whereas the previous algorithm did not terminate after a week of computation.

Index Terms— Algorithms, phylogenetic trees, consensus trees, agreement subtrees

1. Introduction

The reconstruction of evolutionary relationships among sets of genes or sets of species is fundamental. These relationships often take the form of a phylogenetic tree. When combined with geographical data, phylogenies are essential to understanding the movements and interactions of populations [12]. When combined with geographical and temporal data, they are essential to understanding the spread of epidemics [22]. While the utility of phylogenies is evident in areas like phylogeography and epidemiology, they are also at the heart of seemingly less related fields like functional genomics. Indeed, large-scale studies linking a biological trait to a function rely on phylogenetic relationships to differentiate between "Selected Effect" and "Causal Role" [26].

It is, however, often difficult to find the true phylogenetic tree for a set of taxa. Many methods for constructing phylogenetic trees have been proposed, and many different evolutionary characters can be considered [11], [17], [23], [24], [40]. The use of different methods or different evolutionary characters, however, may result in different trees on the same set of taxa. These trees need to be compared or summarized.

One approach is to calculate a numerical index of agreement or a distance between rival trees. Many tree comparison metrics have been proposed for this purpose, such as the cophenetic correlation coefficient [41], the path-difference distance [43], [47], the nearest-neighbor interchange (NNI) distance [14], [48], the Robinson-Foulds (RF) distance [15], [39], the quartet distance [8], and the matching distance [33].

Another approach is to compute a new tree that represents the information shared by the rival trees. Such an approach is called a *consensus method* and the computed tree is called a *consensus tree*. Since Adams [2] introduced the first consensus method in 1972, a great variety of different consensus methods have been developed and studied. Interested readers may refer to [6], [9] for excellent surveys.

Most consensus methods assume that all the input trees have equal taxon sets, and output a tree having the same taxon set as the input trees. Two widely-used examples are the *strict consensus tree* and the *majority-rule consensus tree* [36]. Removing an edge from a phylogenetic tree yields a *bipartition* of the entire taxon set. The strict consensus tree contains exactly those bipartitions common to all input trees while the majority-rule consensus tree contains exactly those bipartitions that appear in more than half of the input trees.

A *rogue* leaf in a collection of phylogenetic trees is one whose position is obviously different from tree to tree. These two consensus methods are susceptible to the presence of rogue leaves. Even a small number of rogue leaves may substantially increase tree distances, and deteriorate the resolutions (*i.e.* the number of internal edges) of the strict and majority-rule consensus trees [1], [38], [45], [49].

One way to overcome the problem caused by rogue leaves is to use a *consensus subtree* method, which allows some taxa to be removed from the input trees. The most popular of such methods is the *agreement subtree* method, introduced by Finden and Gordon [25]. This method determines the consensus of a collection of trees by identifying maximum cardinality subsets of leaves for which all input trees agree. The trees induced by these maximum cardinality subsets are *maximum agreement subtrees* (MASTs). The problem of constructing MASTs has been extensively studied in the literature [3], [7], [13], [18]–[21], [25], [27], [29], [30], [32], [44].

The agreement subtree method avoids the problem caused by rogue leaves. However, unlike the strict and majority-rule consensus trees, MASTs of a collection of trees are not necessarily unique. The number of MASTs can be exponential in the number of leaves [30] and there can exist two MASTs which share almost no leaves [45]. Therefore, the agreement subtree method has the following disadvantage: using a single

• *B.-F. Wang is with the Department of Computer Science, National Tsing Hua University Hsinchu, Taiwan 30013, Republic of China. E-mail: bfwang@cs.nthu.edu.tw.*
• *K. M. Swenson is with LIRMM, Université Montpellier, CNRS, Montpellier, France; and Institut de Biologie Computationnelle (IBC), Montpellier, France. E-mail: swenson@lirmm.fr.*

MAST to represent the information shared by rival trees can be misleading, while it may be impossible to inspect all MASTs.

Aiming at providing a baseline method to report a subtree of high confidence that is not susceptible to rogue leaves, Swenson *et al.* [45] suggested using the summary of the information common to all MASTs, as represented by their intersection. This is called the *kernel agreement subtree* (*KAST*). Like the agreement subtree, this new consensus subtree method avoids the problem caused by rogue leaves. In addition, as with the strict consensus tree, it reports a single subtree of highest confidence. Experimental results showed that the KAST can be used as a baseline method to find subtrees of confidence, to report subsets of input trees for which we are confident, and to be an indicator of randomness in the input [45]. In summary, the KAST is complementary to the strict consensus tree, since it reports a leaf set of high confidence while the strict consensus tree reports an edge set of high confidence.

A fast algorithm for computing the KAST is of interest. In the age of modern phylogenetic inference researchers are now addressing datasets with thousands [5], [31], [35], or even tens-of-thousands of taxa [34], and equally as many trees [50] derived from diverse evolutionary characters [46]. Version 4.0a123 of the popular phylogenetic program PAUP* [46] includes an implementation of the KAST that does not scale to datasets of these sizes.

The focus of this paper is speeding up the construction of the KAST. Let $\mathcal{T}$ be a set of k phylogenetic trees on a set of n taxa. Assume that at least one of the trees in $\mathcal{T}$ has degree bounded by a constant d. The current best known algorithms for finding a MAST of $\mathcal{T}$ are due to Farach *et al.* [19] and Bryant [7], which require $O(kn^3 + n^d)$ time. Although the number of all MASTs can be exponential, Swenson *et al.* [45] showed that the KAST of $\mathcal{T}$ can be computed in polynomial time through a modification of Bryant's MAST algorithm. They did not analyze the complexity of their algorithm, yet claimed a running time similar to that of Bryant's by ignoring the time necessary to compute set operations. In Section 4, a detailed analysis is given, which shows that their algorithm takes $O(kn^3 + n^4 + n^{d+1})$ time. In this paper, a faster algorithm is presented. The presented algorithm requires $O(kn^3 + n^d)$ time, which matches the current best upper bound for the MAST problem.

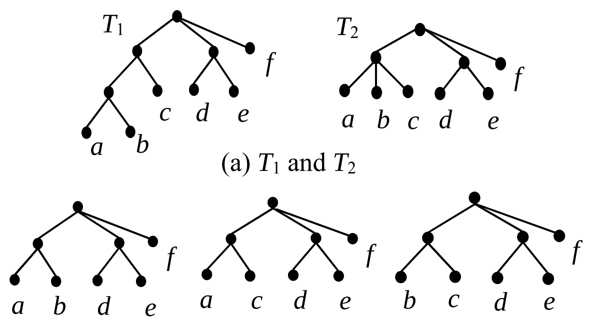
The experimental section applies our algorithm to simulated data, where rogue taxa are added to a fixed topology. On a pair of trees with 2,000 taxa our new algorithm is on average over 40 times faster than the previous, and uses more than an order of magnitude less RAM. The importance of our innovation is glaring when applying our algorithm to trees produced from large ribosomal RNA alignments. Four of these tests did not terminate using the old algorithm, after a week of computation using more than 100 gigabytes of RAM on a server. Our new algorithm took at most 8 hours and 9 gigabytes of RAM for all datasets. A positive side effect of our new algorithm is significant savings in memory consumption.

The remainder of this paper is organized as follows. Section 2 introduces notation and definitions. Section 3 reviews Bryant's MAST algorithm. Section 4 describes Swenson *et al.*'s KAST algorithm. Section 5 gives a faster algorithm for the KAST problem, which requires $O(kn^3 + n^4 + n^d)$ time. For $d \geq 4$, the algorithm in Section 5 is as fast as the current best MAST algorithms. However, for the cases of $d = 2$ and 3, it is slower by a factor of n. We note that these are the most important special cases, because in practice, phylogenetic trees usually have very small degrees, typically no larger than three [3]. Section 6 shows how to solve the KAST problem in $O(kn^3 + n^d)$ time. Section 7 presents our experimental results, while Section 8 concludes the paper.

2. Notation and definitions

A *phylogenetic tree* is a tree in which the leaves are uniquely labeled by a set of taxa. For convenience, a leaf of a phylogenetic tree is simply identified with its label. A phylogenetic tree can be rooted or unrooted. In this paper, only rooted trees are considered. The *degree* of a node is its number of children. We assume that the degree of every non-leaf node is at least two, so that the number of non-leaf nodes is bounded by the number of leaves. As in all MAST literature, a node with degree d greater than two is considered a "hard polytomy", the alternative being a "soft polytomy" which ambiguously represents all possible tree topologies on the d children.

Consider a phylogenetic tree T. If the path from a node a to the root passes through a node b, we call b an *ancestor* of a and call a a *descendant* of b. A *proper descendant* of a node v is a descendant of v which is not v itself. A *proper ancestor* is defined similarly. For any two nodes a, b of T, the *lowest common ancestor* (*LCA*) of a and b is the ancestor of a and b that is a descendant of all ancestors of a and b. For any subset S of the leaves of T, we follow convention by denoting $T|_S$ as the *subtree of* T *induced by* S. $T|_S$ is the tree with leaf set S and interior node set $\{x : x$ is the LCA of some pair of leaves in $S\}$ inheriting the ancestor relation from T (*i.e.* for all $a, b \in S$, the LCAs of a, b in T and $T|_S$ are the same). Two trees T_1 and T_2 on the same label set are *isomorphic* if there is a 1-1 mapping between their internal nodes such that the LCA of any two leaves a, b in T_1 is mapped to the LCA of a, b in T_2.



(a) T_1 and T_2

(b) MASTs of $\mathcal{T}$

Figure 1. The three MASTs of $\mathcal{T} = \{T_1, T_2\}$.

Let $\mathcal{T} = \{T_1, T_2, ..., T_k\}$ be a set of phylogenetic trees on the same set $\mathcal{L}$ of n labels. Throughout this paper, we assume that at least one of the trees in $\mathcal{T}$ has maximum degree d, where $d \geq 2$ is a constant. An *agreement subtree* of $\mathcal{T}$ is a tree T such that T, $T_1|_S$, $T_2|_S$, ..., and $T_k|_S$ are mutually isomorphic, where S is the leaf set of T. The leaf set of an agreement subtree is called an *agreement set*. The *size* of an agreement subtree is the cardinality of its leaf set. A *maximum agreement* *subtree* (*MAST*) is an agreement subtree of maximum size. For example, for the set $\mathcal{T} = \{T_1, T_2\}$ in Figure 1(a), the three MASTs of $\mathcal{T}$ are depicted in Figure 1(b). The *MAST problem* is to find a MAST of $\mathcal{T}$. The *kernel agreement subtree* (*KAST*) of $\mathcal{T}$, denoted by $KAST(\mathcal{T})$, is the intersection of the leaf sets of all MASTs of $\mathcal{T}$. For example, in Figure 1, we have $KAST(\mathcal{T}) = \{a, b, d, e, f\} \cap \{a, c, d, e, f\} \cap \{b, c, d, e, f\} = \{d, e, f\}$. The *KAST problem* is to compute the KAST of $\mathcal{T}$.

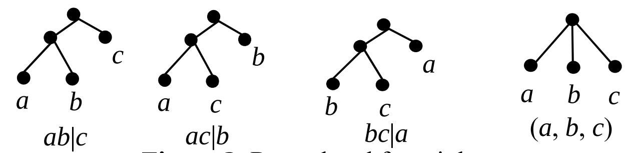


Figure 2. Rooted and fan triples.

In the following, we introduce some notation and definitions that are used throughout this paper. Let T be a phylogenetic tree. The leaf set of T is denoted by $L(T)$. For each node v, the subtree rooted at v is denoted by $T(v)$ and the subtrees rooted at the children of v are called the *subtrees of* v. The subtrees of the root are the *maximal subtrees of* T. Let a, b, c be three leaves of T. Figure 2 depicts the four possible topologies of the subtree induced by $\{a, b, c\}$. For the first three cases, we say that a, b, c form a *rooted triple* in T. We use $ab|c$ to denote the rooted triple in which the LCA of a, b is a descendant of the LCA of a, c. For the last case, we say that a, b, c form a *fan triple* in T and use (a, b, c) to denote the fan triple. A set $S \subseteq L(T)$ is a *fan set* of T if in the induced tree $T|_S$ all the leaves are children of the root. For example, in T_2 of Figure 1, both $\{a, b, c\}$ and $\{a, d, f\}$ are fan sets.

For a pair $(a, b) \in \mathcal{L}^2$, we use $lca_i(a, b)$ to denote the LCA of a, b in a tree $T_i \in \mathcal{T}$ and use $lca^*(a, b)$ to denote the sequence $(lca_1(a, b), lca_2(a, b), ..., lca_k(a, b))$. The set of all rooted triples common to all trees in $\mathcal{T}$ is denoted by $\mathcal{R}$ and the set of all fan triples common to all trees in $\mathcal{T}$ is denoted by $\mathcal{F}$. A set is called a fan set of $\mathcal{T}$ if it is a fan set of every tree in $\mathcal{T}$. Since at least one tree has maximum degree d, it is easy to see that any fan set of $\mathcal{T}$ has size at most d.

3. Bryant's MAST algorithm

This section reviews Bryant's MAST algorithm. For brevity, only the computation for the size of a MAST of $\mathcal{T}$, denoted by $mast(\mathcal{T})$, is described. The following lemma provides the basis of Bryant's algorithm.

Lemma 3.1. [7] A tree T is an agreement subtree of $\mathcal{T}$ if and only if $r(T) \subseteq \mathcal{R}$ and $f(T) \subseteq \mathcal{F}$, where $r(T)$ and $f(T)$ are, respectively, the set of rooted triples and the set of fan triples in T.

For each pair $(a, b) \in \mathcal{L}^2$, define the following:

$\mathcal{A}(a, b)$:	the set of agreement subtrees of $\mathcal{T}$ in each of which a, b are leaves and the LCA of a, b is the root;
$m(a, b)$:	the size of the largest trees in $\mathcal{A}(a, b)$; and
$\mathcal{M}(a, b)$:	the set of largest trees in $\mathcal{A}(a, b)$.

Clearly, $mast(\mathcal{T})$ is the maximum value of $m(a, b)$ over all $(a, b) \in \mathcal{L}^2$. Thus, to compute $mast(\mathcal{T})$, it suffices to compute $m(a, b)$ for every $(a, b) \in \mathcal{L}^2$.

Bryant's algorithm computes all $m(a, b)$ by dynamic programming. It determines $m(a, b)$ before $m(a', b')$ if in T_1 the LCA of a, b is a proper descendant of the LCA of a', b'. We proceed by discussing the computation for a fixed pair $(a, b) \in \mathcal{L}^2$. If $a = b$, we simply have $m(a, b) = 1$. Assume that $a \neq b$. Consider a tree $Q \in \mathcal{M}(a, b)$. Since the LCA of a, b is the root of Q, we know that a and b are in different maximal subtrees of Q. Let Q_a be the maximal subtree containing a, Q_b be the maximal subtree containing b, and $Q_1, Q_2, ..., Q_r$ be the remaining maximal subtrees, if there are any. Note that $r \leq d - 2$, since at least one tree in $\mathcal{T}$ has maximum degree d. For $1 \leq j \leq r$, let c_j be any leaf of Q_j.

For $1 \leq i \leq k$, let $\alpha_i = lca_i(a, b)$. By the definition of $\mathcal{M}(a, b)$, Q is a largest agreement subtree of $\{T_i(\alpha_i) : 1 \leq i \leq k\}$ under the condition that a and b should be contained in an agreement subtree. Since $\{a, b, c_1, c_2, ..., c_r\}$ is a fan set of Q, according to Lemma 3.1, we know that in each T_i, $1 \leq i \leq k$, the leaves $a, b, c_1, c_2, ...,$ and c_r are in different subtrees of α_i. As a result, the sizes of $Q_a, Q_b, Q_1, Q_2, ...,$ and Q_r can be discussed individually.

Consider the subtree Q_a first. For $1 \leq i \leq k$, let A_i be the subtree containing a that is rooted at a child of α_i. Then, Q_a is a largest agreement subtree of $\{A_i : 1 \leq i \leq k\}$ under the condition that a should be contained in an agreement subtree. Let

$$X(a|b) = \{x : ax|b \in \mathcal{R}\} \cup \{a\}.$$

Note that $X(a|b)$ is the set of common labels of $A_1, A_2, ..., A_k$ and thus only the labels in $X(a|b)$ can be leaves of an agreement subtree of $\{A_i : 1 \leq i \leq k\}$. Clearly, under the condition that a should be contained in an agreement subtree, a tree T is an agreement subtree of $\{A_i : 1 \leq i \leq k\}$ if and only if $T \in \mathcal{A}(a, x)$ for some $x \in X(a|b)$. Therefore, the size of Q_a can be computed as

$$m(a|b) = \max\{m(a, x) : x \in X(a|b)\}.$$

Similarly, the size of Q_b can be computed as $m(b|a)$ and the size of each Q_j, $1 \leq j \leq r$, can be computed as $m(c_j|a)$ (or $m(c_j|b)$). Consequently, given $c_1, c_2, ..., c_r$, the size of Q can be computed as

$$m(a|b) + m(b|a) + \textstyle\sum_{1 \leq j \leq r} m(c_j|a).$$

We proceed by deriving a recurrence for $m(a, b)$. Let $C(a, b) = \{c : (a, b, c) \in \mathcal{F}\}$. If $C(a, b)$ is empty, all trees in $\mathcal{A}(a, b)$ are binary and thus we have

$$m(a, b) = m(a|b) + m(b|a).$$

Assume that $C(a, b)$ is not empty. Define $G(a, b)$ to be a graph in which for each $c \in C(a, b)$ there is a vertex c with *weight* $m(c|a)$ and for every pair (u, v) of vertices, there is an edge between u and v if and only if $(a, u, v) \in \mathcal{F}$. A *clique* in a graph is a subset of the vertices such that every two distinct vertices in the clique are connected by an edge. By the definition of $G(a, b)$, a set U is a clique in $G(a, b)$ if and only if $\{a, b\} \cup U$ is a fan set of $\{T_i(\alpha_i) : 1 \leq i \leq k\}$. As a result, it can be concluded that the value of $m(a, b)$ is $m(a|b) + m(b|a) + w$ if and only if w is the maximum total weight of any clique in $G(a, b)$. Therefore,

$$m(a, b) = m(a|b) + m(b|a) + \textstyle\sum_{c \in S} m(c|a), \qquad (1)$$

where S is a maximum weight clique in $G(a, b)$. Bryant's algorithm for computing each $m(a, b)$ is formally described as follows.

```
Procedure FINDMAST($(a, b)$)
input: $(a, b) \in \mathcal{L}^2$
output: $m(a, b)$
begin
```

```
1. if $a = b$ then return 1
2. $m(a|b) \leftarrow \max\{m(a, x) : x \in X(a|b)\}$
3. $m(b|a) \leftarrow \max\{m(b, x) : x \in X(b|a)\}$
4. for each $c \in C(a, b)$ do
5.     $m(c|a) \leftarrow \max\{m(c, x) : x \in X(c|a)\}$
6. $S \leftarrow$ a maximum weight clique in $G(a, b)$
7. $m(a, b) \leftarrow m(a|b) + m(b|a) + \sum_{c\in S} m(c|a)$
8. return $m(a, b)$
end
```

The time complexity of FINDMAST is analyzed as follows. Lines 1-3 require $O(n)$ time. Lines 4-5 require $O(|C(a, b)| \times n) = O(n^2)$ time. The finding of a maximum weight clique in Line 6 is done by considering all subsets $\{c_1, c_2, ..., c_l\}$ of $C(a, b)$, where $0 \le l \le d - 2$. Since $|C(a, b)| \le n$ and d is a constant, the number of such subsets is at most $\sum_{0\le l\le d-2}\binom{n}{l} = O(n^{d-2})$. Each subset can be checked in $O(d^2) = O(1)$ time. Thus, Line 6 requires $O(n^{d-2})$ time. Line 7 requires $O(d) = O(1)$ time. Therefore, the time complexity of FINDMAST is $O(n^2 + n^{d-2})$.

To solve the MAST problem, FINDMAST is called for every $(a, b) \in \mathcal{L}^2$. As a result, a total of $O(n^2 \times (n^2 + n^{d-2})) = O(n^4 + n^d)$ time is required. The n^4 term is contributed by Lines 4-5, which computes $m(c|a)$ for each $c \in C(c, b)$. This term can be reduced to n^3 by simply avoiding the work of recomputing the value of $m(c|a)$ for the same pair of (c, a). Therefore, the total time spent on computing the values of all $m(a, b)$ is $O(n^3 + n^d)$. The construction of $\mathcal{R}$ and $\mathcal{F}$ takes $O(kn^3)$ time [7]. Consequently, the following is obtained.

Theorem 3.1. [7] The MAST problem on a set of k trees can be solved in $O(kn^3 + n^d)$ time, where n is the size of the trees and at least one tree has maximum degree d.

4. Swenson *et al.*'s KAST algorithm

Swenson *et al.* solved the KAST problem through a modification of Bryant's MAST algorithm. Let $\mathcal{M}(\cdot, \cdot)$, $m(\cdot, \cdot)$, $m(\cdot|\cdot)$, $C(\cdot, \cdot)$ be defined the same as in Section 3. For each pair $(a, b) \in \mathcal{L}^2$, define

$$\mathcal{K}(a, b) = \bigcap_{Q\in\mathcal{M}(a, b)} L(Q).$$

Then, the KAST of $\mathcal{T}$ is the intersection of all $\mathcal{K}(a, b)$ such that $m(a, b) = mast(\mathcal{T})$. Thus, the KAST problem can be solved by computing $\mathcal{K}(a, b)$ for every $(a, b) \in \mathcal{L}^2$.

Consider the computation of $\mathcal{K}(a, b)$ for a fixed pair $(a, b) \in \mathcal{L}^2$. For convenience, define

$$X^*(a|b) = \{x : x \in X(a|b), m(a, x) = m(a|b)\} \text{ and}$$
$$\mathcal{K}(a|b) = \bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x). \qquad (2)$$

For example, consider the set $\mathcal{T} = \{T_1, T_2\}$, where $T_1 = (((((((a, w), x), p), y), z), q), (b, c))$ and $T_2 = (((((((a, w), z), y), q), x), p), c), b)$. In this example, $X(a|b) = \{p, q, x, y, z, w\}$, $m(a, w) = 2$, $m(a, x) = m(a, y) = m(a, z) = 3$, and $m(a, p) = m(a, q) = 4$. Hence, $m(a|b) = 4$ and $X^*(a|b) = \{p, q\}$. Since $M(a, p) = \{(((a, w), x), p)\}$ and $M(a, q) = \{(((a, w), y), q), (((a, w), z), q)\}$, we have $K(a, p) = \{a, w, x, p\}$ and $K(a, q) = \{a, w, y, q\} \cap \{a, w, z, q\} = \{a, w, q\}$. Thus, $K(a|b) = K(a, p) \cap K(a, q) = \{a, w\}$.

If $a = b$, we have $\mathcal{K}(a, b) = \{a\}$. Assume that $a \ne b$. By definition, $\mathcal{K}(a, b)$ is the set of leaves common to all trees in $\mathcal{M}(a, b)$. Consider the case that $C(a, b)$ is empty. In this case, all trees in $\mathcal{M}(a, b)$ are binary trees. Under the condition that a should be contained in an agreement subtree, the set of all maximum agreement subtrees of $\mathcal{T}$ on the leaf set $X(a|b)$ is

$$\mathcal{M}(a|b) = \{Q : Q \in \mathcal{M}(a, x), x \in X^*(a|b)\}. \qquad (3)$$

Similarly, under the condition that b should be contained in an agreement subtree, the set of all maximum agreement subtrees of $\mathcal{T}$ on the leaf set $X(b|a)$ is $\mathcal{M}(b|a)$. It can be shown that a tree $Q \in \mathcal{A}(a, b)$ is in $\mathcal{M}(a, b)$ if and only if one of its maximal subtrees is in $\mathcal{M}(a|b)$ and the other is in $\mathcal{M}(b|a)$. Since $X(a|b)$ and $X(b|a)$ are disjoint, we have

$$\begin{aligned}
&\mathcal{K}(a, b)\\
=\ & \textstyle\bigcap_{Q\in\mathcal{M}(a, b)} L(Q)\\
=\ & (\textstyle\bigcap_{Q\in\mathcal{M}(a|b)} L(Q)) \cup (\bigcap_{Q\in\mathcal{M}(b|a)} L(Q))\\
=\ & (\textstyle\bigcap_{x\in X^*(a|b)}(\bigcap_{Q\in\mathcal{M}(a,x)} L(Q))) \cup (\bigcap_{x\in X^*(b|a)}(\bigcap_{Q\in\mathcal{M}(b,x)} L(Q)))\\
=\ & (\textstyle\bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x)) \cup (\bigcap_{x\in X^*(b|a)} \mathcal{K}(b, x))\\
=\ & \mathcal{K}(a|b) \cup \mathcal{K}(b|a).
\end{aligned}$$

We proceed by discussing the case where $C(a, b)$ is not empty. Let $G(a, b)$ be defined the same as in Section 3. Let $S = \{c_1, c_2, ..., c_r\}$ be a maximum weight clique in $G(a, b)$. Define $Q^*(S)$ to be the set of trees that can be constructed as follows: choose an arbitrary tree from each of $\mathcal{M}(a|b)$, $\mathcal{M}(b|a)$, $\mathcal{M}(c_1|a)$, $\mathcal{M}(c_2|a)$, ..., and $\mathcal{M}(c_r|a)$; and then create a new vertex and make the roots of the chosen trees as its children. According to (1), any member of $Q^*(S)$ is a tree in $\mathcal{M}(a, b)$.

Consider the computation of $\bigcap_{Q\in Q^*(S)} L(Q)$. For a set $\mathcal{H}$ of trees, let $L^*(\mathcal{H})$ denote the union of the leaf sets of the trees in $\mathcal{H}$. By the definitions of $\mathcal{M}(\cdot|\cdot)$ and $G(a, b)$, we know that $L^*(\mathcal{M}(a|b))$, $L^*(\mathcal{M}(b|a))$, $L^*(\mathcal{M}(c_1|a))$, $L^*(\mathcal{M}(c_2|a))$, ..., and $L^*(\mathcal{M}(c_r|a))$ are pairwise disjoint. Furthermore, by the definition of $Q^*(S)$, any tree in the sets $\mathcal{M}(a|b)$, $\mathcal{M}(b|a)$, $\mathcal{M}(c_1|a)$, $\mathcal{M}(c_2|a)$, ..., and $\mathcal{M}(c_r|a)$ is necessarily contained in some tree in $Q^*(S)$. Therefore,

$$\begin{aligned}
&\textstyle\bigcap_{Q\in Q^*(S)} L(Q)\\
=\ & (\textstyle\bigcap_{Q\in\mathcal{M}(a|b)} L(Q)) \cup (\bigcap_{Q\in\mathcal{M}(b|a)} L(Q)) \cup\\
& \qquad (\textstyle\bigcup_{c\in S} (\bigcap_{Q\in\mathcal{M}(c|a)} L(Q)))\\
=\ & (\textstyle\bigcap_{x\in X^*(a|b)}(\bigcap_{Q\in\mathcal{M}(a,x)} L(Q))) \cup (\bigcap_{x\in X^*(b|a)}(\bigcap_{Q\in\mathcal{M}(b,x)} L(Q))) \cup\\
& \qquad (\textstyle\bigcup_{c\in S}(\bigcap_{x\in X^*(c|a)} (\bigcap_{Q\in\mathcal{M}(c,x)} L(Q))))\\
=\ & (\textstyle\bigcap_{x\in X^*(a|b)}\mathcal{K}(a, x)) \cup (\bigcap_{x\in X^*(b|a)} \mathcal{K}(b, x)) \cup\\
& \qquad (\textstyle\bigcup_{c\in S}(\bigcap_{x\in X^*(c|a)} \mathcal{K}(c, x)))\\
=\ & \mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup (\textstyle\bigcup_{c\in S} \mathcal{K}(c|a)).
\end{aligned}$$

Let $S^*(a, b)$ be the set of maximum weight cliques in $G(a, b)$. It can be shown that for any agreement subtree Q in $\mathcal{M}(a, b)$, there exists at least a clique S in S^* such that $Q \in Q^*(S)$. Consequently, we have the following recurrence:

$$\begin{aligned}
&\mathcal{K}(a, b)\\
=\ & \textstyle\bigcap_{Q\in\mathcal{M}(a, b)} L(Q)\\
=\ & \textstyle\bigcap_{S\in S^*(a, b)} (\bigcap_{Q\in Q^*(S)} L(Q))\\
=\ & \textstyle\bigcap_{S\in S^*(a, b)} (\mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup (\bigcup_{c\in S} \mathcal{K}(c|a)))\\
=\ & \mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup \Phi(a, b), \qquad (4)
\end{aligned}$$

where

$$\Phi(a, b) = \textstyle\bigcap_{S\in S^*(a, b)} (\bigcup_{c\in S} \mathcal{K}(c|a)). \qquad (5)$$

Swenson *et al.*'s algorithm for computing each $\mathcal{K}(a, b)$ is formally described as the following procedure.

```
Procedure FINDKAST$(a, b)$
input: $(a, b) \in \mathcal{L}^2$
output: $\mathcal{K}(a, b)$
begin
1. if $a = b$ then return $\{a\}$
2. $\mathcal{K}(a|b) \leftarrow \bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x)$
3. $\mathcal{K}(b|a) \leftarrow \bigcap_{x\in X^*(b|a)} \mathcal{K}(b, x)$
4. for each $c \in C(a, b)$ do
5.     $\mathcal{K}(c|a) \leftarrow \bigcap_{x\in X^*(c|a)} \mathcal{K}(c, x)$
```

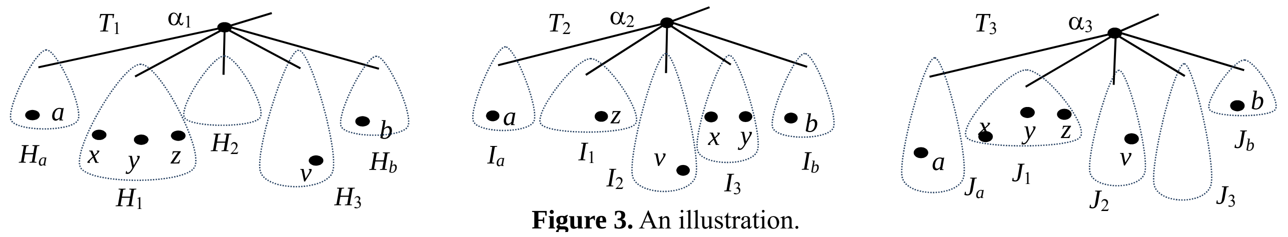


Figure 3. An illustration.

```
6. $\Phi(a, b) \leftarrow \bigcap_{S \in S^*(a, b)} (\bigcup_{c \in S} \mathcal{K}(c|a))$
7. $\mathcal{K}(a, b) \leftarrow \mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup \Phi(a, b)$
8. return $\mathcal{K}(a, b)$
end
```

Assume that $\mathcal{R}$, $\mathcal{F}$, and all $m(a|b)$ and $m(a, b)$, where $(a, b) \in \mathcal{L}^2$, have been computed by performing Bryant's algorithm. The time complexity of FINDKAST is analyzed as follows. Line 1 takes $O(1)$ time. Lines 2-5 compute the following sets: $\mathcal{K}(a|b)$, $\mathcal{K}(b|a)$, and $\mathcal{K}(c|a)$ of each $c \in C(a, b)$. According to (2), each of these sets needs a sequence of $O(n)$ set intersection operations to compute. A set operation requires $O(n)$ time. Therefore, the running time of Lines 2-5 is $O((|C(a, b)| + 2) \times n \times n) = O(n^3)$. Lines 6 and 7 compute $\mathcal{K}(a, b)$ according to (4) and (5). The computation of $\Phi(a, b)$ in Line 6 needs a sequence of $O(d \times |S^*(a, b)|) = O(n^{d-2})$ set union and set intersection operations. Therefore, Line 6 requires $O(n^{d-1})$ time. Line 7 takes $O(n)$ time. As a result, the overall running time of Lines 6-7 is $O(n^{d-1})$.

To solve the KAST problem, FINDKAST is called for every $(a, b) \in \mathcal{L}^2$. For convenience, we say that the total running time of Lines 2-5 of FINDKAST(a, b) over all $(a, b) \in \mathcal{L}^2$ is spent on computing all $\mathcal{K}(a|b)$ and the total running time of Lines 6-7 of FINDKAST over all $(a, b) \in \mathcal{L}^2$ is spent on computing all $\mathcal{K}(a, b)$. For a fixed pair (a, b), Lines 2-5 require $O(n^3)$ time. However, the time spent on computing all $\mathcal{K}(a|b)$ is $O(n^4)$, instead of $O(n^5)$, since we only need to compute $\mathcal{K}(a|b)$ once for each distinct (a, b). For a fixed pair (a, b), Lines 6-7 requires $O(n^{d-1})$ time. Therefore, the time spent on computing all $\mathcal{K}(a, b)$ is $O(n^{d+1})$. Bryant's algorithm for computing $\mathcal{R}$, $\mathcal{F}$, and all $m(a|b)$ and $m(a, b)$ requires $O(kn^3 + n^d)$ time. Consequently, we obtain the following.

Theorem 4.1. [45] The KAST problem on a set of k trees can be solved in $O(kn^3 + n^4 + n^{d+1})$ time, where n is the size of the trees and at least one tree has maximum degree d.

An important case of the KAST problem is the case when the given trees are binary trees. For this case, the following is obtained.

Corollary 4.1. [45] The KAST problem on a set of k binary trees can be solved in $O(kn^3 + n^4)$ time, where n is the size of the trees.

5. An $O(kn^3 + n^4 + n^d)$-time KAST algorithm

A bottleneck of the algorithm in Section 4 is the computation of $\Phi(a, b)$ in Line 6 of FINDKAST. According to (5), computing each $\Phi(a, b)$ needs $O(n^{d-2})$ set operations. In this section, we improve the upper bound of the KAST problem to $O(kn^3 + n^4 + n^d)$ by showing that each $\Phi(a, b)$ can be computed by using amortized $O(1)$ set operations.

Consider the computation of $\Phi(a, b)$ for a fixed pair $(a, b) \in \mathcal{L}^2$. As in Section 3, for $1 \le i \le k$, we define α_i to be $lca_i(a, b)$. For a tree $T_i(\alpha_i)$, $1 \le i \le k$, and a leaf l of $T_i(\alpha_i)$, we use $\beta(i, l)$ to denote the subtree of α_i that contains l. For example, in Figure 3, $\beta(1, x) = H_1$, $\beta(2, x) = I_3$, and $\beta(3, x) = J_1$. Let $G(a, b)$, $S^*(a, b)$ and $Q^*(S)$ be defined the same as in Section 4. Our first intent is to avoid redundant set operations for computing $\Phi(a, b)$ by utilizing the topological structure of $T_1(\alpha_1)$. We start with the following two simple observations.

Lemma 5.1. Any clique in $G(a, b)$ contains at most one leaf of each subtree of α_1.

Proof. Let $S \subseteq C(a, b)$ be a set containing two leaves, say u and v, of a subtree of α_1. In $G(a, b)$, there is an edge between u and v if and only if $(a, u, v) \in \mathcal{F}$. Since $uv|a$ is a rooted triple of T_1, we have $(a, u, v) \notin \mathcal{F}$. Thus, S is not a clique in $G(a, b)$ and the lemma holds. □

Lemma 5.2. Let S be a clique in $S^*(a, b)$. A leaf $l \in \mathcal{L}$ can be contained in a tree $Q \in Q^*(S)$ only if there is leaf $s \in \{a, b\} \cup S$ such that l and s are in the same subtree of α_i for every $i = 1, 2, ..., k$.

Proof. According to the construction of the trees in $Q^*(S)$, Q is made up by taking a tree from $M(a|b)$, a tree from $M(b|a)$, and a tree from $M(s|a)$ for each $s \in S$. The trees in $M(a|b)$ are agreement subtrees of $\{\beta(i, a) : 1 \le i \le k\}$ and thus contain only leaves common to all $\beta(i, a)$, $1 \le i \le k$. Similarly, the trees in $M(b|a)$ contain only leaves common to all $\beta(i, b)$, $1 \le i \le k$, and for each $s \in S$ the trees in $M(s|a)$ contain only leaves common to all $\beta(i, s)$, $1 \le i \le k$. Therefore, a leaf $l \in \mathcal{L}$ can be contained in a tree $Q \in Q^*(S)$ only if there exists a leaf $s \in \{a, b\} \cup S$ such that l is in $\beta(i, s)$ for every $i = 1, 2, ..., k$. Thus, the lemma holds. □

For convenience, if a clique S in $G(a, b)$ contains a leaf of a subtree H of α_1, we say that S *uses* H. For example, in Figure 3, $S = \{z, v\}$ is a clique in $G(a, b)$ and it uses H_1 and H_3. We have the following.

Lemma 5.3. Let H be a subtree of α_1 that contains neither a nor b. If H is not used by all cliques in $S^*(a, b)$, then $\mathcal{K}(a, b)$ does not contain any leaf of H.

Proof. Assume that H is not used by a clique $S \in S^*(a, b)$. By definition, $\mathcal{K}(a, b)$ is the set of leaves common to all trees in $\mathcal{M}(a, b)$. This lemma can be proved by showing that there exists a tree $Q \in \mathcal{M}(a, b)$ that does not contain any leaf of H. Let $Q \in Q^*(S)$ be a tree. Note that any member of $Q^*(S)$ is a tree in $\mathcal{M}(a, b)$. Consider any leaf l of H. Since a and b are not in H and H is not used by S, leaf l is not in $\beta(1, s)$ for all $s \in \{a, b\} \cup S$. As a result, by Lemma 5.2, Q does not contain any leaf of H. Thus, the lemma holds. □

For each subtree H of α_1, let $C(a, b, H)$ be the set of leaves in H that are contained in some clique in $S^*(a, b)$. That is,

$$C(a, b, H) = \bigcup_{S \in S^*(a, b)} (L(H) \cap S).$$

For convenience, we say that some leaves of $\mathcal{L}$ are *coherent* with respect to (a, b) if they are in the same subtree of α_i for every $i = 1, 2, ..., k$. For example, in Figure 3, x and y are coherent; but x, y, and z are not, since in T_2, they are not contained in the same subtree of α_2. According to Lemma 5.3, in the computation of $\mathcal{K}(a, b)$, we can ignore any subtree of α_1 that contains neither a nor b and is not used by all cliques in $S^*(a, b)$. The following lemma allows us to identify more subtrees that can be ignored.

Lemma 5.4. Let H be a subtree of α_1 that is used by all cliques in $S^*(a, b)$. If $C(a, b, H)$ contains two leaves that are not coherent with respect to (a, b), then the set $\mathcal{K}(a, b)$ does not contain any leaf of H.

Proof. Assume that there are two leaves, say c and c', in $C(a, b, H)$ that are not coherent with respect to (a, b). In the following, we prove this lemma by showing that there are two trees $Q, Q' \in \mathcal{M}(a, b)$ such that $L(Q) \cap L(Q')$ does not contain any leaf in H.

By the definition of $C(a, b, H)$, there are two cliques, say S and S', in $S^*(a, b)$ such that S contains c and S' contains c'. Note that $S \neq S'$, since by Lemma 5.1 each of S and S' contains at most one leaf of H. Let Q be a tree in $Q^*(S)$ and Q' be a tree in $Q^*(S')$. Consider a leaf l in H. By Lemma 5.1, c is the unique leaf of S that is in H. Thus, c is the only leaf of S such that c and l are in the same subtree of α_1. Consequently, according to Lemma 5.2, l can be contained in Q only if l and c are coherent with respect to (a, b). Similarly, l can be contained in Q' only if l and c' are coherent with respect to (a, b). Since c and c' are not coherent with respect to (a, b), we know that l cannot be a leaf of both Q and Q'. The lemma holds.□

Let $Valid(a, b) = \{H : H$ is a subtree of α_1 used by all cliques in $S^*(a, b)$ and all leaves in $C(a, b, H)$ are coherent$\}$. According to (4), $\mathcal{K}(a, b)$ is the union of three disjoint sets $\mathcal{K}(a|b)$, $\mathcal{K}(b|a)$, and $\Phi(a, b)$. The leaves of $\mathcal{K}(a|b)$ and $\mathcal{K}(b|a)$ are contributed, respectively, by the subtree of α_1 that contains a and the subtree of α_1 that contains b; and the leaves of $\Phi(a, b)$ are contributed by the subtrees of α_1 that contain neither a nor b. By Lemmas 5.3 and 5.4, if a subtree H of α_1 contains neither a nor b, $\mathcal{K}(a, b)$ contains a leaf of H only if it is in $Valid(a, b)$. Thus, it can be concluded that only the subtrees in $Valid(a, b)$ can give leaves to $\Phi(a, b)$. Therefore, the right side of equation (5) can be simplified as follows:

$$\begin{aligned}
& \textstyle\bigcap_{S\in S^*(a,b)} \left(\bigcup_{c\in S} \mathcal{K}(c|a)\right) \\
= & \textstyle\bigcap_{S\in S^*(a,b)} \left(\bigcup_{c\in S \text{ and } \beta(1,c)\in Valid(a,b)} \mathcal{K}(c|a)\right) \\
= & \textstyle\bigcap_{S\in S^*(a,b)} \left(\bigcup_{H\in Valid(a,b),\, c\in S\cap L(H)} \mathcal{K}(c|a)\right) \\
= & \textstyle\bigcup_{H\in Valid(a,b)} \left(\bigcap_{S\in S^*(a,b),\, c\in S\cap L(H)} \mathcal{K}(c|a)\right) \\
& \text{(since the subtrees of } \alpha_1 \text{ are mutually disjoint)} \\
= & \textstyle\bigcup_{H\in Valid(a,b)} \left(\bigcap_{c\in C(a,b,H)} \mathcal{K}(c|a)\right) \\
& \text{(since } C(a,b,H) = \textstyle\bigcup_{S\in S^*(a,b)} (L(H)\cap S)\text{)}.
\end{aligned}$$

Consequently, (5) can be written as

$$\Phi(a, b) = \textstyle\bigcup_{H\in Valid(a,b)} \Phi_a(C(a, b, H)), \qquad (6)$$

where for any $U \subseteq \mathcal{L}$,

$$\Phi_a(U) = \textstyle\bigcap_{c\in U} \mathcal{K}(c|a). \qquad (7)$$

Clearly, according to (6) and (7), the number of set operations required for computing each $\Phi(a, b)$ is reduced to $O(n)$. Consequently, the total number of set operations required to compute $\Phi(a, b)$ for all $(a, b) \in \mathcal{L}^2$ is $O(n^3)$. In the following, by further utilizing the topological structure of each input tree, we show that the total number of required set operations can be reduced to $O(n^2)$. More specifically, we show that for each $a \in \mathcal{L}$, $O(n)$ set operations are sufficient for computing $\Phi(a, b)$ for all $b \in \mathcal{L}$.

Consider a fixed $a \in \mathcal{L}$. According to (6), to compute $\Phi(a, b)$ for all $b \in \mathcal{L}$, it suffices to compute $\Phi_a(C(a, b, H))$ for all $b \in \mathcal{L}$ and all $H \in Valid(a, b)$. If we compute $\Phi(a, b)$ for all $b \in \mathcal{L}$ individually according to (4) and (5), it may happen that $\Phi_a(U)$ is computed many times for the same set U. For instance, consider the artificial example in Figure 4, in which $k = 3$ and $T_1(\alpha_1)$, $T_2(\alpha_2)$, and $T_3(\alpha_3)$ are isomorphic. In this example, it is easy to see that for each $b \in L(B)$, we have $H \in Valid(a, b)$ and $C(a, b, H) = L(H)$; and if we compute all $\Phi(a, b)$ individually according to (4) and (5), $\Phi_a(L(H))$ will be computed $|L(B)|$ times. Define Γ_a to be the following collection of sets:

$$\{C(a, b, H) : b \in \mathcal{L}, H \in Valid(a, b)\}.$$

Then, to compute $\Phi(a, b)$ for all $b \in \mathcal{L}$, it suffices to compute $\Phi_a(U)$ for all $U \in \Gamma_a$. Our intent is to avoid re-computing the content of $\Phi_a(U)$ for the same set U and to show that $O(n)$ set operations are sufficient for computing $\Phi_a(U)$ for all $U \in \Gamma_a$.

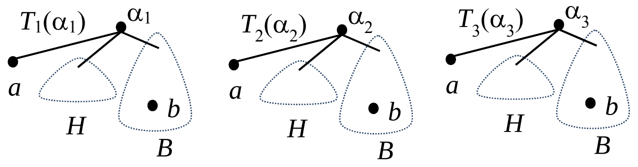


Figure 4. An example.

For convenience, we say that the subtrees in $Valid(a, b)$ are *valid* for $\Phi(a, b)$. We need the following lemma.

Lemma 5.5. Let H be a subtree of α_1 that is valid for $\Phi(a, b)$. The value of $m(c|a)$ is the same for every $c \in C(a, b, H)$.

Proof. We prove this lemma by contradiction. Suppose that there are two leaves $c, c' \in C(a, b, H)$ such that $m(c|a) < m(c'|a)$. By the definition of $C(a, b, H)$, there is a clique $S \in S^*(a, b)$ which contains c. Since H is valid for $\Phi(a, b)$, we know that c and c' are coherent with respect to (a, b). Thus, for each $x \in C(a, b)$, we have $(a, c, x) \in \mathcal{F}$ if and only if $(a, c', x) \in \mathcal{F}$. As a result, c and c' have the same neighborhood in the graph $G(a, b)$. Therefore, $S - \{c\} \cup \{c'\}$ is clique. Since $m(c|a) < m(c'|a)$, this clique, $S - \{c\} \cup \{c'\}$, has larger total weight than S, contradicting to that S is a maximum weight clique in $G(a, b)$. Therefore, the lemma holds □

Lemma 5.6. Let a, b, b' be three leaves. If $lca^*(a, b) \neq lca^*(a, b')$, we have $C(a, b) \cap C(a, b') = \emptyset$.

Proof. Assume that $lca^*(a, b) \neq lca^*(a, b')$. Since $lca^*(a, b) \neq lca^*(a, b')$, there is a tree T_g such that $lca_g(a, b) \neq lca_g(a, b')$, where $1 \leq g \leq k$. Since $lca_g(a, b) \neq lca_g(a, b')$ and there is an ancestor-descendant relationship between them, there is no leaf l such that both (a, b, l) and (a, b', l) are fan triples in T_g. By definition, if $C(a, b) \cap C(a, b') \neq \emptyset$, there exists a leaf l such that both (a, b, l) and (a, b', l) are fan triples in T_i for every $i = 1, 2, .., k$. Therefore, it can be concluded that $C(a, b) \cap C(a, b') = \emptyset$. Thus, the lemma holds. □

Lemma 5.7 Let a, b, b' be three leaves such that $lca^*(a, b) = lca^*(a, b')$. Let H be a subtree of $\alpha_1 = lca_1(a, b)$. If H is valid for both $\Phi(a, b)$ and $\Phi(a, b')$, then either $C(a, b, H) \cap C(a, b', H) = \emptyset$ or $C(a, b, H) = C(a, b', H)$.

Proof. Note that since $lca^*(a, b) = lca^*(a, b')$, we know that b' and a are in different subtrees of $\alpha_i = lca_i(a, b)$ for every $i = 1, 2, ..., k$; but b' and b may be in the same subtree or in different subtrees of α_i. Since H is valid for $\Phi(a, b)$, all leaves in $C(a, b, H)$ are in the same subtree of α_i for every $i = 1, 2, ..., k$. Similarly, all leaves in $C(a, b', H)$ are in the same subtree of α_i for every $i = 1, 2, ..., k$. Two cases are considered.

Case 1. There is a tree T_g, $1 \le g \le k$, such that the leaves in $C(a, b, H)$ and $C(a, b', H)$ are in different subtrees of α_g. (See Figure 5(a) for an illustration, in which $g = 2$.)
Since the leaves in $C(a, b, H)$ and the leaves in $C(a, b', H)$ are contained, respectively, in two disjoint subtrees of α_g, we have $C(a, b, H) \cap C(a, b', H) = \varnothing$.

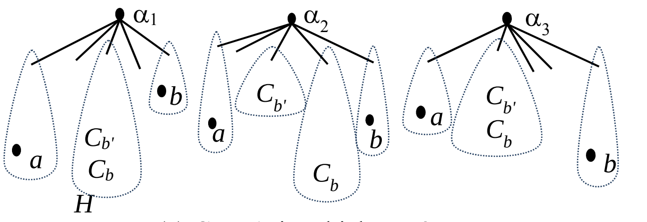
Case 2. The leaves in $C(a, b, H)$ and $C(a, b', H)$ are in the same subtree of α_i for every $i = 1, 2, ..., k$. (See Figure 5(b).)
In the following, we prove that in this case $C(a, b, H) = C(a, b', H)$. First, we show that $C(a, b, H) \subseteq C(a, b', H)$. Consider a leaf c in $C(a, b, H)$. Since H is valid for $\Phi(a, b)$, there is a clique $S \in S^*(a, b)$ that contains c. Let S' be any clique in $S^*(a, b')$. Since H is valid for $\Phi(a, b')$, set S' contains a leaf $c' \in C(a, b', H)$. Note that c and c' are in the same subtree of α_i for every $i = 1, 2, ..., k$. Recall that a set U is a clique in $G(a, b)$ if and only if $\{a, b\} \cup U$ is a fan set of $\mathcal{T}$. Thus, $\{a, b\} \cup S$ is a fan set of $\mathcal{T}$. Since S contains c and the two leaves c, c' are in the same subtree of α_i for every $i = 1, 2, ..., k$, we know that $\{a, b\} \cup (S - \{c\} \cup \{c'\})$ is also a fan set of $\mathcal{T}$. That is, $S - \{c\} \cup \{c'\}$ is a clique in $G(a, b)$. Consequently, we have $m(c'|a) \le m(c|a)$; otherwise this clique, $S - \{c\} \cup \{c'\}$, has larger total weight than S, contradicting to that S is a maximum weight clique in $G(a, b)$. Similarly, from the fact that S' is a maximum weight clique in $G(a, b')$, it can also be derived that $S' - \{c'\} \cup \{c\}$ is a clique in $G(a, b')$ and $m(c|a) \le m(c'|a)$. As a result, it can be concluded that $m(c|a) = m(c'|a)$ and $S' - \{c'\} \cup \{c\}$ is a maximum weight clique in $G(a, b')$. Therefore, c is contained in $C(a, b', H)$. From the above discussion, we know that $C(a, b, H) \subseteq C(a, b', H)$. Similarly, it can be shown that $C(a, b', H) \subseteq C(a, b, H)$. By combining these two statements, we have $C(a, b, H) = C(a, b', H)$, which completes the proof of this lemma. □

By combining Lemmas 5.6 and 5.7, the following is obtained.

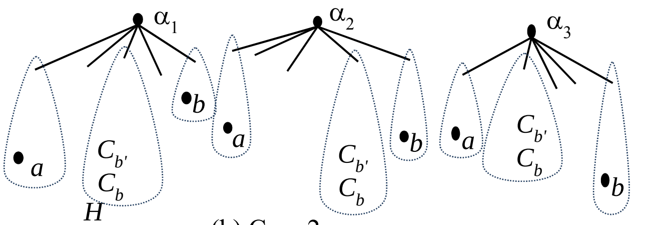
Lemma 5.8. Let a, b, b' be three leaves. Let H be a subtree of $lca_1(a, b)$ that is valid for $\Phi(a, b)$ and H' be a subtree of $lca_1(a, b')$ that is valid for $\Phi(a, b')$. Then, either $C(a, b, H) \cap C(a, b', H') = \varnothing$ or $C(a, b, H) = C(a, b', H')$.

Proof. If $lca^*(a, b) \ne lca^*(a, b')$, the lemma holds, since by Lemma 5.6, we have $C(a, b, H) \cap C(a, b', H') \subseteq C(a, b) \cap C(a, b') = \varnothing$. Assume that $lca^*(a, b) = lca^*(a, b')$. Two cases are considered: $H \ne H'$ and $H = H'$. If $H \ne H'$, we know that H and H' are two different subtrees of $lca_1(a, b)$, since $lca_1(a, b) = lca_1(a, b')$. Thus, we have $C(a, b, H) \cap C(a, b', H') = \varnothing$. If $H = H'$, by Lemma 5.7, either $C(a, b, H) \cap C(a, b', H') = \varnothing$ or $C(a, b, H) = C(a, b', H')$, which completes the proof. □

We proceed to show that for each $a \in \mathcal{L}$, $O(n)$ set operations are sufficient for computing $\Phi_a(U)$ for all $U \in \Gamma_a$. Recall that $\Gamma_a = \{C(a, b, H) : b \in \mathcal{L}, H \in Valid(a, b)\}$. Let $size(\Gamma_a)$ be the total size of the sets in Γ_a. According to (7), for each $U \in \Gamma_a$, the number of set operations required for computing $\Phi_a(U)$ is $|U| - 1$. Therefore, the number of set operations required for computing $\Phi_a(U)$ for all $U \in \Gamma_a$ is less than $size(\Gamma_a)$. The following lemma gives an upper bound on $size(\Gamma_a)$.



(a) Case 1, in which $g = 2$



(b) Case 2

Figure 5. An illustration for the proof of Lemma 5.7, in which $k = 3$ and C_b and $C_{b'}$ denote, respectively, $C(a, b, H)$ and $C(a, b', H)$.

Lemma 5.9. For any $a \in \mathcal{L}$, $size(\Gamma_a) \le n - 1$.

Proof. Clearly, $a \notin C(a, b, H)$ for any $b \in \mathcal{L}$ and $H \in Valid(a, b)$. By Lemma 5.8, any two sets in Γ_a are disjoint. Therefore, $size(\Gamma_a) \le |\mathcal{L} - \{a\}| \le n - 1$ and the lemma holds. □

According to (4), (6), (7), and Lemma 5.9, a more efficient algorithm for computing $\mathcal{K}(a, b)$ is given as follows.

```
Procedure NEWKAST-1(a, b)
input: $(a, b) \in \mathcal{L}^2$
output: $\mathcal{K}(a, b)$
begin
1.  if $a = b$ then return $\{a\}$
2.  $\mathcal{K}(a|b) \leftarrow \bigcap_{x \in X^*(a|b)} \mathcal{K}(a, x)$
3.  $\mathcal{K}(b|a) \leftarrow \bigcap_{x \in X^*(b|a)} \mathcal{K}(b, x)$
4.  for each $c \in C(a, b)$ do
5.      $\mathcal{K}(c|a) \leftarrow \bigcap_{x \in X^*(c|a)} \mathcal{K}(c, x)$
6.  find Valid(a, b) and compute C(a, b, H) for each subtree
    $H \in Valid(a, b)$
7.  for each $H \in Valid(a, b)$ do
8.      $\Phi_a(U) \leftarrow \bigcap_{c \in U} \mathcal{K}(c|a)$, where $U = C(a, b, H)$
9.  $\Phi(a, b) \leftarrow \bigcup_{H \in Valid(a, b)} \Phi_a(C(a, b, H))$
10. $\mathcal{K}(a, b) \leftarrow \mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup \Phi(a, b)$
11. return $\mathcal{K}(a, b)$
end
```

The detailed implementation of NEWKAST-1 is described as follows. Without loss of generality, assume that the maximum degree of T_1 is d. The following lemma gives the time complexity of Line 6.

Lemma 5.10. Given a pair $(a, b) \in \mathcal{L}^2$, we can find $Valid(a, b)$ and compute $C(a, b, H)$ for each $H \in Valid(a, b)$ in $O(n^{d-2})$ time.

Proof. For $d = 2$, since $Valid(a, b)$ is empty, the lemma holds trivially. Assume that $d \ge 3$. We prove this lemma by presenting an algorithm. Initially, all subtrees of α_1 and all leaves in $T_1(\alpha_1)$ are *unmarked*. This initialization takes $O(n)$ time. First, for each clique $S \in S^*(a, b)$, we mark all the leaves in S and mark all the subtrees of α_1 that are not used by S. Since the degree of α_1 and the size of a clique are both bounded by d, this step requires $O(d \times |S^*(a, b)|) = O(n^{d-2})$ time. Next, for each subtree H of α_1, we compute $C(a, b, H)$ as the set of marked leaves in H. This step requires $O(|L(T_1(\alpha_1))|) = O(n)$ time.

To find the set $Valid(a, b)$, we need to determine whether each subtree of α_1 is valid for $\Phi(a, b)$. All marked subtrees are not valid, since they are not used by all the cliques in $S^*(a, b)$. Consider an unmarked subtree H. By definition, H is valid if all leaves in $C(a, b, H)$ are coherent with respect to (a, b). Clearly, two leaves c, c' in H are coherent with respect to (a, b) if and only if $cc'|a \in \mathcal{R}$. Therefore, we determine whether H is valid as follows: pick an arbitrary leaf $c \in C(a, b, H)$ and then check whether $cc'|a \in \mathcal{R}$ for all $c' \in C(a, b, H) - \{c\}$. The above checking requires $O(|C(a, b, H)|)$ time. Consequently, the set $Valid(a, b)$ can be found in $O(\sum_H |C(a, b, H)|) = O(|L(T_1(\alpha_1))|) = O(n)$ time.

The above algorithm requires $O(n^{d-2})$ time. Therefore, the lemma holds. □

We proceed to describe the implementation of Lines 7 and 8. By Lemma 5.9, $O(n)$ set operations are sufficient for computing $\Phi_a(U)$ for all $U \in \Gamma_a$. To avoid re-computation, for each leaf $a \in \mathcal{L}$, we maintain an array, TABLE$_a$ of n entries to save the results of computed $\Phi_a(U)$ and implement the finding of $\Phi_a(U)$ in Line 8 as follows: first check to see whether $\Phi_a(U)$ has been already computed for $U = C(a, b, H)$; if so, use the result stored in TABLE$_a$; if not, compute $\Phi_a(C(a, b, H))$ and then store the result in TABLE$_a$. We need to establish a relationship between the positions of TABLE$_a$ and the sets in Γ_a. By Lemma 5.8, any two sets in Γ_a are disjoint. Thus, each set in Γ_a can be uniquely identified by any of its elements. We let each set in Γ_a be represented by its smallest element. More specifically, when $\Phi_a(U)$ has been computed for a set U, the result is stored in TABLE$_a[l_{\min}]$, where $l_{\min}$ is the smallest element in U. Accordingly, the corresponding position of a given set U in TABLE$_a$ can be found in $O(|U|) = O(n)$ time.

For a fixed $a \in \mathcal{L}$, Lines 7-8 of NEWKAST-1 over all $b \in \mathcal{L}$ require $O(n)$ set operations. Therefore, the required set operations of Lines 7-8 over all $(a, b) \in \mathcal{L}^2$ is $O(n^2)$. Consequently, the total time of Lines 7-8 over all $(a, b) \in \mathcal{L}^2$ is $O(n^3)$.

By Lemma 5.10, the total time of Line 6 over all $(a, b) \in \mathcal{L}^2$ is $O(n^d)$. As discussed in Section 4, the total time of Lines 1-5 and Line 10 over all $(a, b) \in \mathcal{L}^2$ is $O(n^4)$. Therefore, we obtain the following.

Theorem 5.1. The KAST problem on a set of k trees can be solved in $O(kn^3 + n^4 + n^d)$ time, where n is the size of the trees and at least one tree has maximum degree d.

6. An $O(kn^3 + n^d)$-time KAST algorithm

In this section, we show how to further improve the upper bound of the KAST problem to $O(kn^3 + n^d)$.

To solve the KAST problem, the algorithm in Section 5 computes $\mathcal{K}(a, b)$ for all $(a, b) \in \mathcal{L}^2$. At the time of writing, we are not aware of a more efficient way to compute all $\mathcal{K}(a, b)$ explicitly. In the time complexity of Theorem 5.1, the n^4 term comes from Lines 2-5 of NEWKAST-1, which computes $\mathcal{K}(a|b)$ for all $(a, b) \in \mathcal{L}^2$. Recall that this straightforward method for computing $\mathcal{K}(a|b)$ potentially requires a linear number of set intersections on subsolutions $\mathcal{K}(a, x)$ such that $m(a, x) = m(a|b)$ and $x \in X(a|b)$. Thus, for each pair in $\mathcal{L}^2$ we must take potentially $\Omega(n^2)$ time to perform all set intersections.

This section reduces the total number of set intersections necessary by working with what we call refinements of $\mathcal{K}(a, b)$ and $\mathcal{K}(a|b)$, denoted by $\mathcal{K}_1(a, b)$ and $\mathcal{K}_1(a|b)$, rather than $\mathcal{K}(a, b)$ and $\mathcal{K}(a|b)$ themselves. A *refinement* of a set is a subset that preserves all KAST leaves. We compute $\mathcal{K}_1(a, b)$ and $\mathcal{K}_1(a|b)$ only for the pairs (a, b) that are relevant to the computation of the KAST of $\mathcal{T}$. The idea is to group computations for a fixed leaf a; all relevant pairs (a, b) have equivalent $\mathcal{K}_1(a|b)$ if they have the same value for $m(a|b)$ and their subsolutions are all computed from the same intersection of sets $\mathcal{K}_1(a, x)$ such that $m(a, x) = m(a|b)$ and (a, x) are relevant. The implication is that $\mathcal{K}_1(a, x)$ appears only in a single set intersection for a fixed a. This results in an algorithm that uses a linear number of total set operations to compute refinements of $\mathcal{K}_1(a|b)$ for a fixed a, rather than $\Omega(n^2)$. Consequently, the n^4 term in the time complexity of Theorem 5.1 is removed. Since $\mathcal{K}_1(a, b)$ preserves all KAST leaves of $\mathcal{K}(a, b)$, the KAST of $\mathcal{T}$ is the intersection of all $\mathcal{K}_1(a, b)$ such that $m(a, b) = mast(\mathcal{T})$.

Section 6.1 shows how to identify all relevant pairs (a, b). Section 6.2 describes the computation of $\mathcal{K}_1(a, b)$ and $\mathcal{K}_1(a|b)$ for each relevant pair.

6.1 Finding relevant pairs

Let (a, b) be a pair in $\mathcal{L}^2$. We say that (a, b) is *relevant* if there exists a MAST of $\mathcal{T}$ that contains a tree in $\mathcal{M}(a, b)$ as a subtree, and is *irrelevant* otherwise. That is, (a, b) is relevant if $\mathcal{M}(a, b)$ contains a sub-solution to the problem of finding the MASTs of $\mathcal{T}$. Suppose that there is a MAST of $\mathcal{T}$ that contains a tree X in $\mathcal{M}(a, b)$ as a subtree. If we modify the MAST by replacing X with any other tree in $\mathcal{M}(a, b)$, it is easy to concluded from Lemma 3.1 that the resulting tree is still a MAST. Therefore, if (a, b) is relevant, every tree in $\mathcal{M}(a, b)$ is a subtree of a MAST of $\mathcal{T}$.

As discussed in Section 3, given a clique $S \in S^*(a, b)$, a tree in $\mathcal{M}(a, b)$ can be constructed by taking a tree in $\mathcal{M}(a|b)$, a tree in $\mathcal{M}(b|a)$, and a tree in $\mathcal{M}(c|a)$ for each $c \in S$. From this, and the definition of $\mathcal{M}(a|b)$ in (3), it can be concluded that if (a, b) is a relevant pair, all pairs in the following set are relevant:

$$\Pi(a, b) = \{(a, x) : x \in X^*(a|b)\} \cup \{(b, x) : x \in X^*(b|a)\} \cup \left(\bigcup_{S \in S^*(a, b)} \left(\bigcup_{c \in S} \{(c, x) : x \in X^*(c|a)\}\right)\right)$$

As a result, after $m(a|b)$ and $m(a, b)$ have been computed for every $(a, b) \in \mathcal{L}^2$ by applying Bryant's dynamic programming algorithm, all relevant pairs can be recognized by tracing back the steps that led to each maximum table entry. More specifically, all relevant pairs can be recognized, in a top-down manner, according to the following rules:

(i) if $m(a, b) = mast(\mathcal{T})$, (a, b) is relevant (since all trees in $\mathcal{M}(a, b)$ are MASTs); and

(ii) if (a, b) is relevant, all pairs in $\Pi(a, b)$ are relevant.

Assume that $\mathcal{R}$, $\mathcal{F}$, and all $m(a|b)$ and $m(a, b)$, where $(a, b) \in \mathcal{L}^2$, have been computed. A procedure that finds all relevant pairs is given as follows.

```
Procedure FINDREVELANT
/* find the set of relevant pairs
begin
1. REVELANT ← ∅                      /* initialization
2. for each $(a, b) \in \mathcal{L}^2$ do   /* initialization
3.    compute $X^*(a|b)$ and mark $X^*(a|b)$ unexplored
4. for each $(a, b) \in \mathcal{L}^2$ do   /* find relevant pairs recursively
5.    if $m(a, b) = mast(\mathcal{T})$ then ADDREVELANT$(a, b)$
6. return REVELANT
end
Procedure ADDREVELANT$(a, b)$
// add $(a, b)$ and the pairs in $\Pi(a, b)$ into REVELANT
begin
1. REVELANT ← REVELANT ∪ $\{(a, b)\}$
2. if $X^*(a|b)$ is unexplored then EXPLORE$(X^*(a|b))$
3. if $X^*(b|a)$ is unexplored then EXPLORE$(X^*(b|a))$
4. for each $c \in \bigcup_{S\in S^*(a,b)} S$ do
5.    if $X^*(c|a)$ is unexplored then EXPLORE$(X^*(c|a))$
end
Procedure EXPLORE$(X^*(p|q))$
//add the pairs in $X^*(p|q)$ into REVELANT
begin
1. mark $X^*(p|q)\}$ explored      /* avoid re-exploring
2. for each $x \in X^*(p|q)$ do
3.    if $(p, x) \notin$ REVELANT then ADDREVELANT$(p, x)$
end
```

Lemma 6.1. FINDREVELANT requires $O(n^3 + n^d)$ time.

Proof. Each call to ADDREVELANT takes $O(d|S^*(a, b)|) = O(n^{d-2})$ time, excluding the time spent on the calls to EXPLORE. Each call to EXPLORE takes $O(|X^*(p|q)|) = O(n)$ time, excluding the time spent on the calls to ADDREVELANT. The number of calls to ADDREVELANT is bounded by the number of relevant pairs, which is at most n^2. The number of calls to EXPLORE is bounded by the number of explored sets $X^*(p|q)$, which is also bounded n^2. Therefore, the total time of FINDREVELANT is $O(n^3 + n^d)$ and the lemma holds. □

For ease of discussion, we call $\mathcal{K}(a, b)$ and $\mathcal{K}(a|b)$, respectively, the *$\mathcal{K}$-set* and the *conditional $\mathcal{K}$-set* of (a, b). According to (2), (4), and (5), $\mathcal{K}(a, b)$ is determined from the $\mathcal{K}$-sets of the pairs in $\Pi(a, b)$. Therefore, if (a, b) is relevant, all the $\mathcal{K}$-sets required for computing $\mathcal{K}(a, b)$ are also of relevant pairs. As a result, it can be concluded that a set $\mathcal{K}(a, b)$ needs to be computed only if (a, b) is relevant. In the remainder of this section, we show that a set $\mathcal{K}(a|b)$ also needs to be computed only if (a, b) is relevant.

Lemma 6.2. Let a, b, c be three leaves such that $(a, b, c) \in \mathcal{F}$ and c is a leaf of a tree in $\mathcal{M}(a, b)$. If (a, b) is relevant, then (a, c) is relevant.

Proof. Assume that (a, b) is relevant. Let X be a tree in $\mathcal{M}(a, b)$ that contains the leaf c. Since (a, b) is relevant, there is a MAST, say Z, that contains X as a subtree. Since $(a, b, c) \in \mathcal{F}$, tree X is an agreement subtree in which a, c are leaves and the root is the LCA of a, c. Thus, $X \in \mathcal{A}(a, c)$. All trees in $\mathcal{A}(a, c)$ are agreement subtrees of $\{T_i(\alpha_i) : \alpha_i = lca_i(a, c), 1 \le i \le k\}$. Therefore, after modifying Z by replacing X with any tree in $\mathcal{A}(a, c)$, the resulting tree is still an agreement subtree. Consequently, X is a largest tree in $\mathcal{A}(a, c)$; otherwise, Z is not a MAST. That is, $X \in \mathcal{M}(a, c)$. Since X is a subtree of a MAST, (a, c) is relevant and the lemma holds. □

According to (4) and (5), the following conditional $\mathcal{K}$-sets are required for the computation of the $\mathcal{K}$-set of a relevant pair (a, b):

$\mathcal{K}(a|b)$, $\mathcal{K}(b|a)$, and $\mathcal{K}(c|a)$ of each $c \in \bigcup_{S\in S^*(a,b)} S$.

The sets $\mathcal{K}(a|b)$ and $\mathcal{K}(b|a)$ are of relevant pairs (a, b) and (b, a). Note that (b, a) is a relevant pair since $\mathcal{M}(b, a) = \mathcal{M}(a, b)$. Recall that given a set $S \in S^*(a, b)$, a tree in $\mathcal{M}(a, b)$ can be constructed by taking a tree in $\mathcal{M}(a|b)$, a tree in $\mathcal{M}(b|a)$, and a tree in $\mathcal{M}(c|a)$ for each $c \in S$. Therefore, each $c \in \bigcup_{S\in S^*(a,b)} S$ is a leaf of a tree in $\mathcal{M}(a, b)$. In addition, each $c \in \bigcup_{S\in S^*(a,b)} S$ is a leaf in $C(a, b)$. As a result, it can be concluded from Lemma 6.2 that (c, a) is relevant for each $c \in \bigcup_{S\in S^*(a,b)} S$. Based upon the above discussion, we know that $\mathcal{K}(a|b)$ also needs to be computed only if (a, b) is relevant.

6.2 Computing refinements

In this section, we show that $\mathcal{K}_1(a, b)$, which is a refinement of $\mathcal{K}(a, b)$, can be computed for every relevant pair (a, b) in $O(kn^3 + n^d)$ time. Recall that the computation of $\mathcal{K}(a, b)$ is based on computing $\mathcal{K}(a|b) = \bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x)$, where $X^*(a|b) = \{x : x \in X(a|b), m(a, x) = m(a|b)\}$. Similarly, for each relevant pair (a, b), the computation of $\mathcal{K}_1(a, b)$ is based on an analogue of $\mathcal{K}(a|b)$ called $\mathcal{K}_1(a|b)$. The key observation will be described by Lemma 6.7, which says that, for any two relevant pairs (a, b) and (a, b'), $\mathcal{K}_1(a|b)$ and $\mathcal{K}_1(a|b')$ are equal if $m(a|b) = m(a|b')$. Thus, for a fixed a we group computations of $\mathcal{K}_1(a|b)$ according to the value of $m(a|b)$ so that we only need to compute $\mathcal{K}_1(a|b)$ once for all b with the same $m(a|b)$. As with $\mathcal{K}(a|b)$, $\mathcal{K}_1(a|b)$ only depends on the relevant pairs (a, x) with the same $m(a, x)$. Therefore, each subsolution $\mathcal{K}_1(a, x)$ for relevant pair (a, x) is implicated in a set operation for exactly one group. Thus, all $\mathcal{K}_1(a|b)$ can be computed in quadratic time for a fixed a, yielding an algorithm that computes $\mathcal{K}_1(a, b)$ for all relevant pairs in $O(kn^3 + n^d)$ time.

A *refinement* of a set $U \subseteq L$ is any set obtained by deleting from U none or some leaves that are not in $KAST(\mathcal{T})$. For example, if $KAST(\mathcal{T}) = \{a, c, d\}$ and $U = \{a, b, d, f\}$, then $\{a, d\}$, $\{a, b, d\}$, $\{a, d, f\}$, and $\{a, b, d, f\}$ are all refinements of U; but $\{a, b, f\}$ is not, since it does not contain the KAST leaf d. Recall that $KAST(\mathcal{T})$ is the intersection of all $\mathcal{K}(a, b)$ with $m(a, b) = mast(\mathcal{T})$. Since a refinement of a $\mathcal{K}$-set preserves all the KAST leaves in the $\mathcal{K}$-set, in the above computation of the intersection, each $\mathcal{K}$-set can be replaced by any of its refinement. Therefore, to solve the KAST problem, it suffices to find a refinement of $\mathcal{K}(a, b)$ for each relevant pair (a, b).

By the definition of a refinement, it is easy to obtain the following.

Observation 6.3. Let $U_1, U'_1, U_2, U'_2, .., U_p, U'_p$ be sets such that for each i, $1 \le i \le p$, U'_i is a refinement of U_i. Then, $U'_1 \cup U'_2 \cup ... \cup U'_p$ is a refinement of $U_1 \cup U_2 \cup ... \cup U_p$; and $U'_1 \cap U'_2 \cap ... \cap U'_p$ is a refinement of $U_1 \cap U_2 \cap ... \cap U_p$.

Lemma 6.4. Let (a, b) be a relevant pair and Q be any tree in $\mathcal{M}(a, b)$. Then, for each $T_i \in \mathcal{T}$, a leaf l of $T_i(\alpha_i)$ is in $KAST(\mathcal{T})$ only if Q contains l, where $\alpha_i = lca_i(a, b)$.

Proof. Let T_i be a tree in $\mathcal{T}$ and l be a leaf of $T_i(\alpha_i)$. Since (a, b) is relevant, there is a MAST, say Z, that contains Q as a subtree. By definition, the leaf l is in $KAST(\mathcal{T})$ only if it is contained in every MAST of $\mathcal{T}$. Assume that $l \notin L(Q)$. In the following, we prove this lemma by showing that Z does not contain l. Suppose by contradiction that Z contains l. From

Lemma 3.1, it is easy to conclude that any subset of an agreement set is also an agreement set. Thus, $L(Q) \cup \{l\}$ is an agreement set and $T_i|_{L(Q)\cup\{l\}}$ is an agreement subtree. Since l is a leaf of $T_i(\alpha_i)$ and $Q \in \mathcal{A}(a, b)$, we know that in $T_i|_{L(Q)\cup\{l\}}$, a, b are leaves and the LCA of a, b is the root. Therefore, $T_i|_{L(Q)\cup\{l\}}$ is in $\mathcal{A}(a, b)$, contradicting to that Q is a largest tree in $\mathcal{A}(a, b)$. Thus, the lemma holds. □

For ease of description, we define a *valid filter* of a set $U \subseteq L$ to be a set containing all the KAST leaves in U. For example, if $KAST(\mathcal{T}) = \{a, c, d\}$ and $U = \{a, b, d, f\}$, then $\{a, d\}$, $\{a, b, c, d\}$, are $\{a, c, d, g\}$ are all valid filters of U; but $\{a, c, f\}$ is not, since $\{a, c, f\}$ does not contain the KAST leaf d. Note that a refinement of U is also a valid filter of U, but the reverse may not be true. For example, $\{a, b, c, d\}$ is a valid filter of U, but it is not a refinement of U, since it is not a subset of U. A valid filter of a set U can be used to remove from U some leave that are not in $KAST(\mathcal{T})$. More specifically, if U' is a valid filter of U, then $U' \cap U$ is a refinement of U

Lemma 6.5. Let (a, b) be a relevant pair. Then, for each $T_i \in \mathcal{T}$, $\mathcal{K}(a, b)$ is a valid filter of any subset of $L(T_i(\alpha_i))$, where $\alpha_i = lca_i(a, b)$.

Proof. We first show that $\mathcal{K}(a, b)$ is a valid filter of $L(T_i(\alpha_i))$. Let l be any leaf of $T_i(\alpha_i)$. By Lemma 6.4, l is in $KAST(\mathcal{T})$ only if it is in every tree in $M(a, b)$. Thus, l is in $KAST(\mathcal{T})$ only if l is in $\mathcal{K}(a, b)$. Therefore, $\mathcal{K}(a, b)$ is a valid filter of $L(T_i(\alpha_i))$. Clearly, a valid filter of a set U is a valid filter of any subset of U. Thus, the lemma holds. □

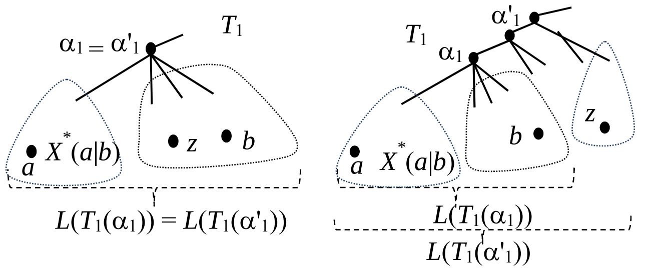
For each relevant pair (a, b), define a variant of $X^*(a|b)$:

$$X_1^*(a|b) = \{x : (a, x) \text{ is relevant}, m(a, x) = m(a|b)\}.$$

We have the following.

Lemma 6.6. For any relevant pair (a, b), $X^*(a|b) \subseteq X_1^*(a|b)$.

Proof. Let x be any leaf in $X^*(a|b)$. Recall that $X^*(a|b) = \{x : x \in X(a|b), m(a, x) = m(a|b)\}$. Since $x \in X^*(a|b)$, we have $m(a, x) = m(a|b)$. In addition, since (a, b) is relevant and $\{(a, x) : x \in X^*(a|b)\} \subseteq \Pi(a, b)$, we know that (a, x) is relevant. Therefore, it can be concluded that each $x \in X^*(a|b)$ is contained in $X_1^*(a|b)$. Thus, the lemma holds. □



Case 1. $\alpha_1 = \alpha'_1$ Case 2. $\alpha_1 \neq \alpha'_1$

Figure 6. A leaf $z \in X_1^*(a|b) - X^*(a|b)$ such that $az|b$ is not a rooted triple in T_1.

Before presenting an algorithm that computes a refinement for the $\mathcal{K}$-set of each relevant pair, we describe the idea behind our approach. Consider the computation of $\mathcal{K}(a|b)$ for a relevant pair (a, b). By definition, $\mathcal{K}(a|b) = \bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x)$. Essentially, our idea is to compute $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$ to take the place of $\mathcal{K}(a|b)$. Let $Z = X_1^*(a|b) - X^*(a|b)$. Since $X^*(a|b) \subseteq X_1^*(a|b)$, this set, $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$, can be considered as the set obtained from $\mathcal{K}(a|b)$ by removing leaves not in $\mathcal{K}(a, z)$ for every $z \in Z$. Consider a leaf $z \in Z$. Since $m(a, z) = m(a|b)$, we know that $az|b \notin \mathcal{R}$; otherwise $z \in X^*(a|b)$. That is, $az|b$ is not a rooted triple in at least one of the trees in $\mathcal{T}$. Without loss of generality, assume that $az|b$ is not a rooted triple in T_1. Since $az|b$ is not a rooted triple in T_1, we know that $\alpha_1 = lca_1(a, b)$ is a descendant of $\alpha'_1 = lca_1(a, z)$. There are two cases: $\alpha_1 = \alpha'_1$, and α_1 is a proper descendant of α'_1. (See Figure 6.) In either case, we have $L(T_1(\alpha'_1)) \supseteq L(T_1(\alpha_1))$. Since $\mathcal{K}(a|b)$ contains a leaf l only if $al|b \in \mathcal{R}$, we know that $\mathcal{K}(a|b)$ is a subset of $L(T_1(\alpha_1))$ and thus is a subset of $L(T_1(\alpha'_1))$. Hence, by Lemma 6.5, $\mathcal{K}(a, z)$ is a valid filter of $\mathcal{K}(a|b)$. Therefore, $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$ preserves all KAST leaves in $\mathcal{K}(a|b)$ and thus is a refinement of $\mathcal{K}(a|b)$.

Based upon the above idea, by replacing $X^*(a|b)$ with $X_1^*(a|b)$ in the definition of $\mathcal{K}(a|b)$, we define $\mathcal{K}_1(a, b)$ as follows: if $a = b$, $\mathcal{K}_1(a, b) = \{a\}$; otherwise

$$\mathcal{K}_1(a, b) = \mathcal{K}_1(a|b) \cup \mathcal{K}_1(b|a) \cup \left(\bigcap_{S\in S^*(a,b)} \left(\bigcup_{c\in S} \mathcal{K}_1(c|a)\right)\right), \tag{8}$$

where

$$\mathcal{K}_1(a|b) = \bigcap_{x\in X_1^*(a|b)} \mathcal{K}_1(a, x). \tag{9}$$

Our algorithm uses $\mathcal{K}_1(a|b)$ to replace $\mathcal{K}(a|b)$. As mentioned, essentially, our idea is to compute $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$ to take the place of $\mathcal{K}(a|b)$. We remark that according to the definition in (9), $\mathcal{K}_1(a|b)$ may not be the same as $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$.

Before showing that $\mathcal{K}_1(a, b)$ is a refinement of $\mathcal{K}(a, b)$, we describe the advantage of replacing $X^*(a|b)$ with $X_1^*(a|b)$ in the definition of $\mathcal{K}(a|b)$.

Lemma 6.7. Let a, b, and b' be leaves such that (a, b) and (a, b') are both relevant, and $m(a|b) = m(a|b')$. Then, $\mathcal{K}_1(a|b) = \mathcal{K}_1(a|b')$.

Proof. Since $m(a|b) = m(a|b')$, according to the definition of $X_1^*(\cdot, \cdot)$, we have $X_1^*(a|b) = X_1^*(a|b')$. Thus, $\mathcal{K}_1(a|b) = \mathcal{K}_1(a|b')$ and the lemma holds. □

In the time complexity of Theorem 5.1, the n^4 term comes from Lines 2-5 of NEWKAST-1, which computes $\mathcal{K}(a|b)$ for all $(a, b) \in \mathcal{L}^2$. Lemma 6.7 indicates that for a fixed $a \in \mathcal{L}$ and size $t < mast(\mathcal{T})$, $\mathcal{K}_1(a|b)$ only needs to be computed once for all b with $m(a|b) = t$. Later, we will show that this property allows $\mathcal{K}_1(a|b)$ to be computed for all relevant pairs (a, b) in $O(n^3)$ time.

We proceed by showing that for each relevant pair (a, b), $\mathcal{K}_1(a, b)$ is a refinement of $\mathcal{K}(a, b)$. The proof is done by induction on on the size $m(a, b)$. The following lemma is needed.

Lemma 6.8. Let (a, b) be a relevant pair. Suppose that $\mathcal{K}_1(a, x)$ is a refinement of $\mathcal{K}(a, x)$ for each $x \in X_1{*}(a|b)$. Then, $\mathcal{K}_1(a|b)$ is a refinement of $\mathcal{K}(a|b)$.

Proof. Recall that, previous to Lemma 6.7, we explained that $\bigcap_{x\in X_1^*(a|b)} \mathcal{K}(a, x)$ is a refinement of $\mathcal{K}(a|b)$. The proof of this lemma is similar. By Lemma 6.6, $X^*(a|b) \subseteq X_1^*(a|b)$. Thus, we can write $\mathcal{K}_1(a|b) = U_1 \cap U_2$, where

$$U_1 = \bigcap_{x\in X^*(a|b)} \mathcal{K}_1(a, x) \text{ and } U_2 = \bigcap_{z\in X_1^*(a|b)-X^*(a|b)} \mathcal{K}_1(a, z).$$

Since $\mathcal{K}_1(a, x)$ is a refinement of $\mathcal{K}(a, x)$ for each $x \in X{*}(a|b)$, by Observation 6.3, $\bigcap_{x\in X^*(a|b)} \mathcal{K}_1(a, x)$ is a refinement of $\bigcap_{x\in X^*(a|b)} \mathcal{K}(a, x)$. That is, U_1 is a refinement of $\mathcal{K}(a|b)$. In the following, we show that U_2 is a valid filter of $\mathcal{K}(a|b)$.

Consider a fixed $z \in X_1{*}(a|b) - X{*}(a|b)$. Since $m(a, z) = m(a|b)$ and $z \notin X^*(a|b)$, we know that $az|b$ is not a rooted triple in at least one of the trees in $\mathcal{T}$. Without loss of generality, assume that $az|b$ is not a rooted triple in T_1. Since $az|b$ is not a rooted triple in T_1, we know that $\alpha_1 = lca_1(a, b)$ is a descendant of $\alpha'_1 = lca_1(a, z)$. (See Figure 6.) Since $\mathcal{K}(a|b) \subseteq L(T_1(\alpha_1)) \subseteq L(T_1(\alpha'_1))$, by Lemma 6.5, $\mathcal{K}(a, z)$ is a valid filter of $\mathcal{K}(a|b)$. In addition, since $\mathcal{K}_1(a, z)$ is a refinement of $\mathcal{K}(a, z)$, it is easy to see that $\mathcal{K}_1(a, z)$ is also a valid filter of $\mathcal{K}(a|b)$. Clearly, the intersection of valid filters is also a valid filter. Therefore, U_2 is a valid filter of $\mathcal{K}(a|b)$.

In summary, U_1 is a refinement of $\mathcal{K}(a|b)$ and U_2 is a valid filter of $\mathcal{K}(a|b)$. From this, it is easy to conclude that $U_1 \cap U_2$ is a refinement of $\mathcal{K}(a|b)$, which completes the proof of this lemma. □

Our new algorithm is designed based upon the following.

Theorem 6.1. For any relevant pair (a, b), $\mathcal{K}_1(a, b)$ is a refinement of $\mathcal{K}(a, b)$.

Proof. We prove the theorem by induction on the size $m(a, b)$. First, consider the case $m(a, b) = 1$. In this case, we have $a = b$; otherwise, $|\mathcal{M}(a, b)| \geq |\{a, b\}| \geq 2$. Since $a = b$, by definition, we have $\mathcal{K}_1(a, b) = \mathcal{K}(a, b) = \{a\}$ and thus the theorem holds.

Consider the case $m(a, b) > 1$. Suppose, by induction, that this theorem is true for all $\mathcal{K}(a', b')$ such that (a', b') is relevant and $m(a', b') < m(a, b)$; we will show that this theorem holds for (a, b) as well. Recall that

$$\mathcal{K}(a, b) = \mathcal{K}(a|b) \cup \mathcal{K}(b|a) \cup (\textstyle\bigcap_{S\in S^*(a,b)} (\bigcup_{c\in S} \mathcal{K}(c|a))), \text{ and}$$

$$\mathcal{K}_1(a, b) = \mathcal{K}_1(a|b) \cup \mathcal{K}_1(b|a) \cup (\textstyle\bigcap_{S\in S^*(a,b)} (\bigcup_{c\in S} \mathcal{K}_1(c|a))).$$

According to Observation 6.3, to prove that $\mathcal{K}_1(a, b)$ is a refinement of $\mathcal{K}(a, b)$, it suffices to show the following:

(i) $\mathcal{K}_1(a|b)$ is a refinement of $\mathcal{K}(a|b)$;

(ii) $\mathcal{K}_1(b|a)$ is a refinement of $\mathcal{K}(b|a)$; and

(iii) $\mathcal{K}_1(c|a)$ is a refinement of $\mathcal{K}_1(c|a)$ for any $c \in S$ and $S \in S^*(a, b)$.

In the following, only the proof of (i) is given. The proofs of (ii) and (iii) are similar. Consider an $x \in X_1^*(a|b)$. Note that according to the definition of $X_1^*(a|b)$, (a, x) is relevant and $m(a, x) = m(a|b)$. According to (1), we have $m(a|b) < m(a, b)$ and thus $m(a, x) = m(a|b) < m(a, b)$. Consequently, by the induction hypothesis, $\mathcal{K}_1(a, x)$ is a refinement of $\mathcal{K}(a, x)$ for any $x \in X_1^*(a|b)$. As a result, by Lemma 6.8, $\mathcal{K}_1(a|b)$ is a refinement of $\mathcal{K}(a|b)$, which completes the proof of this theorem. □

Due to the similarity between the recurrences of $\mathcal{K}(a, b)$ and $\mathcal{K}_1(a, b)$, $\mathcal{K}_1(a, b)$ can be computed by slightly modified NEWKAST-1 as follows.

```
Procedure NEWKAST-2($(a, b)$)
input: $(a, b) \in \mathcal{L}^2$ is a relevant pair
output: $\mathcal{K}_1(a, b)$
begin
1.  if $a = b$ then return $\{a\}$
2.  $\mathcal{K}_1(a|b) \leftarrow \bigcap_{x\in X_1*(a|b)} \mathcal{K}_1(a, x)$        /* $(a, b)$ is relevant
3.  $\mathcal{K}_1(b|a) \leftarrow \bigcap_{x\in X_1*(b|a)} \mathcal{K}_1(b, x)$        /* $(b, a)$ is relevant
4.  for each $c \in C(a, b)$ do
5.      if $(c, a)$ is relevant then $\mathcal{K}_1(c|a) \leftarrow \bigcap_{x\in X_1*(c|a)} \mathcal{K}_1(c, x)$
6.  find $Valid(a, b)$ and compute $C(a, b, H)$ for each subtree
    $H \in Valid(a, b)$
7.  for each $H \in Valid(a, b)$ do
8.      $\Phi_a(U) \leftarrow \bigcap_{c\in U} \mathcal{K}_1(c|a)$, where $U = C(a, b, H)$
9.  $\Phi(a, b) \leftarrow \bigcup_{H\in Valid(a, b)} \Phi_a(C(a, b, H))$
10. $\mathcal{K}_1(a, b) \leftarrow \mathcal{K}_1(a|b) \cup \mathcal{K}_1(b|a) \cup \Phi(a, b)$
11. return $\mathcal{K}_1(a, b)$
end
```

To solve the KAST problem, procedure NEWKAST-2 is called for every relevant pair (a, b), in non-decreasing order of $m(a, b)$. The overall time complexity is analyzed as follows. As shown in Section 5, Lines 1 and 6-10 of NEWKAST-2 can be implemented such that their execution requires $O(n^d)$ time over all relevant pairs (a, b). In the following, we show that Lines 2-5 of NEWKAST-2 can be implemented such that their execution takes $O(n^3)$ time over all relevant pairs (a, b).

The purpose of Lines 2-5 is to compute $\mathcal{K}_1(a|b)$ for all relevant pairs (a, b). According to Lemma 6.7, for any two relevant pairs (a, b) and (a, b'), $\mathcal{K}_1(a|b)$ and $\mathcal{K}_1(a|b')$ are the same if $m(a|b) = m(a|b')$. For each $a \in \mathcal{L}$ and positive integer $t < mast(\mathcal{T})$, define

$$\mathcal{K}_1^{(t)}(a) = \textstyle\bigcap_{x\in \{x\,:\,(a,x) \text{ is relevant},\, m(a,x)=t\}} \mathcal{K}_1(a, x).$$

Then, $\mathcal{K}_1(a|b) = \mathcal{K}_1^{(m(a|b))}(a)$ for each relevant pair (a, b), Thus, our problem is to compute $\mathcal{K}_1^{(t)}(a)$ for all $a \in \mathcal{L}$ and all $t \in [1, mast(\mathcal{T}))$.

Consider the computation of $\mathcal{K}_1^{(t)}(a)$ for a fixed $a \in \mathcal{L}$ and all $t \in [1, mast(\mathcal{T}))$. Let $\lambda(t)$ be the number of relevant pairs (a, x) with $m(a, x) = t$. For each $t \in [1, mast(\mathcal{T}))$, the computation of $\mathcal{K}_1^{(t)}(a)$ requires $\lambda(t) - 1$ set operations. Since $\sum_t \lambda(t) = n$, we know that $O(n)$ set operations are sufficient for computing $\mathcal{K}_1^{(t)}(a)$ for a fixed $a \in \mathcal{L}$ and all $t \in [1, mast(\mathcal{T}))$. As a result, computing $\mathcal{K}_1^{(t)}(a)$ for all $a \in \mathcal{L}$ and all $t \in [1, mast(\mathcal{T}))$ requires $O(n^2)$ set operations. Consequently, the execution time of Lines 2-5 of NEWKAST-2 over all relevant pairs (a, b) is $O(n^3)$. In summary, we obtain the following.

Theorem 6.2. The KAST problem on a set of k trees can be solved in $O(kn^3 + n^d)$ time, where n is the size of the trees and at least one tree has maximum degree d.

For binary trees, the following is obtained.

Corollary 6.1. The KAST problem on a set of k binary trees can be solved in $O(kn^3)$ time, where n is the size of the trees.

7. Experiments

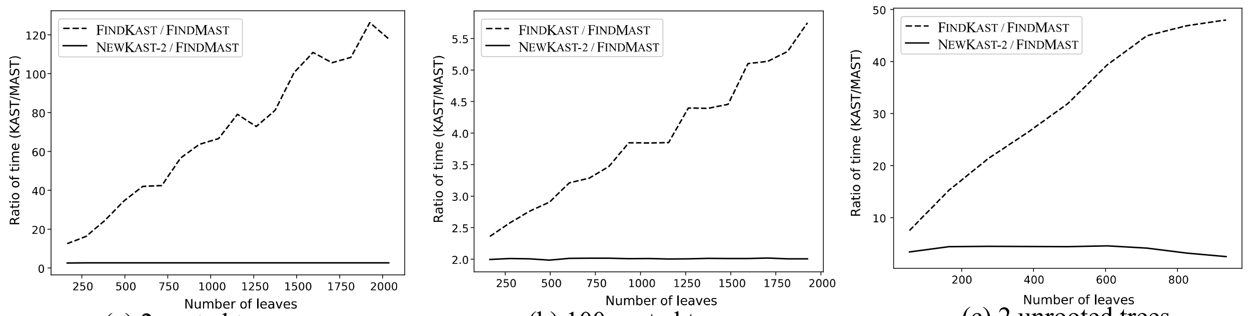
We tested our C implementations of Bryant's MAST algorithm (FINDMAST), the original KAST algorithm (FINDKAST), and our new KAST algorithm (NEWKAST-2). The Swenson *et al.* [45] paper already conducted a thorough study on the utility of the KAST, therefore, the focus of our new experiments was on running time. We timed our implementations on both simulated data and on real data where we compared independently inferred binary trees. Our results on rRNA alignments show that comparison of even the largest trees can be performed on a modern laptop in one work day. Our simulated datasets show an average 43 fold speedup on trees with 2000 taxa. The low KAST values between different Maximum Likelihood trees highlight the importance of the KAST for reporting high-confidence subtrees.

7.1 Running Time on RNA Alignments

Hand-curated RNA alignments at the Comparative RNA Web (CRW) site [10], a Living Tree Project (LTP) alignment

Dataset	k	n	KAST	NEWKAST-2		FINDKAST	
				Time	RAM	Time	RAM
Bacteria/Archea ssuRNA (RAxML D1604)	2	1,604	1,033	51s	143 M	12.2m	3.5 G
	5		698	1.3m	140 M	8.3m	2.3 G
Bacteria/Archea lsuRNA (LTP)	2	1,614	1,186	56s	154 M	8.9m	3.3 G
	5		984	1.4m	157 M	12.5m	3.4 G
Bacteria/Archea ssuRNA (Greengenes)	2	5,088	1321	15.5m	1 G	2h 43m	30 G
	5		479	20m	1.2 G	1h 22m	10 G
Bacteria/Archea/Eukaryot e ssuRNA	2	6,116	5,838	1h 10m	4.2 G	> 1 week	-
	5		2,614	1h	1.9 G	> 1 week	-
Bacteria ssuRNA	2	13,073	9,667	8h 9m	9.4 G	> 1 week	-
	5		1,274	6h 35m	6.7 G	> 1 week	-

Table 1. Running Time on rRNA Alignments.



(a) 2 rooted trees (b) 100 rooted trees (c) 2 unrooted trees

Figure 7. Average timing ratios for FINDKAST/FINDMAST and NEWKAST-2/FINDMAST

[37], a GreenGenes alignment [16], as well as an RNA alignment from [4], were used to test the advantages and the limits of our algorithm. Alignments with up to 13,000 taxa were downloaded and maximum likelihood (ML) phylogenies were reconstructed using RAxML [42]. Sets of most likely RAxML trees were reconstructed and then compared for similarity using our algorithms.

Results are summarized in Table 1. For trees with less than 2,000 leaves NEWKAST-2 takes less than 2 minutes and very little memory, while FINDKAST takes 8 to 12 minutes and a few gigabytes of RAM. For larger datasets the advantage of NEWKAST-2 is clear: for FINDKAST, the dataset with 5,088 leaves takes more than an hour and many gigabytes of RAM while NEWKAST-2 uses only 15-20 minutes and about 1 gigabyte of RAM. All other datasets failed to terminate after a week of computation (on a server with hundreds of gigabytes of RAM) for FINDKAST, while taking hours and under 10 gigabytes of RAM for NEWKAST-2.

7.2 Running Time on Simulated Data

Our simulations started with the construction of a rooted binary birth-death tree T_1 (with parameter 1 for birth and ½ for death, but with a fixed number of leaves). We constructed k copies of T_1 and then added $0.10 \times L(T_1)$ leaves uniformly at random to each of the trees. The expected size of the MAST, therefore was $|L(T_1)|$ for each set of trees.

Both FINDKAST and NEWKAST-2 have FINDMAST at their core, therefore we report times as a proportion of the FINDMAST compute time. The advantage of our new algorithm is clear when there are few rooted trees (*e.g.* $k = 2$). NEWKAST-2 is always 2.7 times slower than FINDMAST while FINDKAST is already 10 times slower at $n = 100$ and more than 100 times slower at $n = 1600$. (Figure 7(a).) At $n = 2000$, NEWKAST-2 takes under seven minutes while FINDKAST takes about five hours.

When the number of trees is larger (*e.g.* $k = 100$), NEWKAST-2 is three times faster (taking 2.1 hours) than FINDKAST (taking 6.1 hours) on trees with 2000 leaves (Figure 7(b)).

Although the computation of the unrooted KAST has yet to be directly studied in the literature, we can compute it by running the rooted version n times, each time rooting the input at a leaf that is assumed to be part of the solution. For the unrooted case, our implementations of NEWKAST-2 and FINDKAST scale similarly to the rooted case, where a tree on 900 leaves takes over 71 hours for FINDKAST but under four hours for NEWKAST-2. (Figure 7(c).)

7.3 Availability of code and data

Our implementation of FINDMAST, FINDKAST, and NEWKAST-2 is available at https://bitbucket.org/thekswenson/kast, along with the RAxML trees inferred from the rRNA alignments and the scripts to reproduce our plots.

8. Concluding remarks

The MAST algorithms in [7], [19] work for any k and d. For some special cases, more efficient solutions exist. For example, there is an $O(n \lg n)$-time algorithm for $k = d = 2$ [13]; there is an $O(n \lg n)$-time algorithm for $k = 2$ and constant d [28]; and there is an $O(n^{1.5})$-time algorithm for $k = 2$ and arbitrary d [28]. Clearly, a leaf is a KAST leaf of $\mathcal{T}$ if and only if its removal from each tree of $\mathcal{T}$ reduces the size of MASTs by one. That is, whether a leaf $l \in \mathcal{L}$ is a KAST leaf can be determined by performing a MAST algorithm to check whether $mast(\mathcal{T}') < mast(\mathcal{T})$, where $\mathcal{T}'$ is the set obtained by removing l from each tree in $\mathcal{T}$. As a result, the set of KAST leaves can be identified by simply running a MAST algorithm n times. By applying this simple approach to the algorithms in [13], [28], the KAST problem is solved in $O(n^2 \lg n)$ time for $k = d = 2$, in $O(n^2 \lg n)$ time for $k = 2$ and constant d, and in $O(n^{2.5})$ time for $k = 2$ and arbitrary d. These simple results are better than the time complexity in Theorem 6.2. One direction for further study is to design more efficient KAST algorithms for these special cases. To beat our KAST algorithm for arbitrary k and d, however, further innovation may have to be made for the general MAST problem.

ACKNOWLEDGMENTS

This research is supported by the National Science Council of the Republic of China under grant MOST-106-2221-E-007-022-MY3.

REFERENCES

[1] J. Aberer and A. Stamatakis, "A simple and accurate method for rogue taxon identification," in *Proceedings of the 2011 IEEE International Conference on Bioinformatics and Biomedicine*, Atlanta, pp. 118–122, 2011.

[2] E. N. Adams, "Consensus techniques and the comparison of taxonomic trees," *Systematic Zoology*, vol. 21, pp. 390–397, 1972.

[3] Amir and D. Keselman, "Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms," *SIAM Journal on Computing*, vol. 26, pp. 1656–1669, 1997.

[4] S. A. Berger, D. Krompass, and A. Stamatakis, "Performance, accuracy, and web server for evolutionary placement of short sequence reads under maximum likelihood," *Systematic Biology*, vol. 60, no. 3, pp. 291–302, 2011.

[5] O. R. P Bininda-Emonds, *et al.*, "The delayed rise of present-day mammals," *Nature*, vol. 446, no. 7135, pp. 507–511, 2007.

[6] P. Bonizzoni, G. D. Vedova, R. Dondi, and G. Mauri, "The comparison of phylogenetic networks: algorithms and complexity," *Bioinformatics Algorithms: Techniques and Applications*, Wiley InterScience, pp. 143–173, 2008.

[7] D. Bryant, *Building trees, hunting for trees, and comparing trees: theory and methods in phylogenetic analysis*, Ph.D. thesis, Department of Mathematics, University of Canterbury, pp. 174–182, 1997.

[8] D. Bryant, *et al.*, "Computing the quartet distance between evolutionary trees," in *Proceedings of the eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, vol. 9, no. 11, 2000.

[9] D. Bryant, "A Classification of Consensus Methods for Phylogenetics," *Theoretical Computer Science*, vol. 61, pp. 163–184, 2002.

[10] J. J. Cannone, S. Subramanian, M. N. Schnare, J. R. Collett, L. M. D'Souza, Y. D, B. Feng, N. Lin, L. V. Madabusi, K. M. Müller, N. Pande, Z. Shang, N. Yu, and R. R. Gutell, "The comparative RNA web (CRW) site: an online database of comparative sequence and structure information for ribosomal, intron, and other RNAs," *BMC Bioinformatics*, vol. 3, no. 1, pp. 2, 2002.

[11] L. L. Cavallip-Sforza and A. W. F. Edwards, "Phylogenetic analysis: models and estimation procedures," *American Journal of Human Genetics*, vol. 19, pp. 233–257, 1967.

[12] J. Chiaroni, P. A. Underhill, and L. L. Cavalli-Sforza, "Y chromosome diversity, human expansion, drift, and cultural evolution," in *Proceedings of the National Academy of Sciences*, vol. 106, no. 48, pp. 20174–20179, 2009.

[13] R. Cole, M. Farach, 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*, vol. 30, pp. 1385–1404, 2000.

[14] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, L. Zhang L, "On computing the nearest neighbor interchange distance," in *Proceedings of the DIMACS Workshop on Discrete Problems with Medical Applications*, 1997, pp. 125–43.

[15] W.H.E. Day, "Optimal algorithm for comparing trees with labeled leaves," *Journal of Classification*, vol. 2, pp. 7–28, 1985.

[16] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, B. B. Andersen, "Greengenes, a chimera-checked 16S rRNA gene database and workbench compatible with ARB," *Applied and environmental microbiology,* vol. 72, no. 7, pp. 5069–5072, 2006.

[17] R. V. Eck and M. O. Dayoff, *Atlas of Protein Sequence and Structure*, National Biomedical Research Foundation, 1966.

[18] M. Farach, T. Przytycka, and M. Thorup, "The maximum agreement subtree problem for binary trees," in *Proceedings of the 2nd European Symposium on Algorithms*, Corfu, Greece, 1995, pp. 381–393.

[19] M. Farach, T. Przytycka, and M. Thorup, "On the agreement of many trees," *Information Processing Letters*, vol. 55, pp. 297–301, 1995.

[20] M. Farach and M. Thorup, "Fast comparison of evolutionary trees," *Information and Computatio*n, vol. 123, pp. 29–37, 1995.

[21] M. Farach and M. Thorup, "Sparse dynamic programming for evolutionary-tree comparison," *SIAM Journal on Computing*, vol. 26, pp. 210–230, 1997.

[22] N. R. Faria, *et al.*, "Zika virus in the Americas: early epidemiological and genetic findings," *Science*, vol. 352, no. 6283, pp. 345–349, 2016.

[23] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach," *Journal of Molecular Evolution*, vol. 17, pp. 368–376, 1981.

[24] J. Felsenstein, *Inferring Phylogenies*, Sinauer Associates, Inc., 2004.

[25] R. Finden and A. D. Gordon, "Obtaining common pruned trees," *Journal of Classification*, vol. 2, pp. 255–276, 1985.

[26] D. Graur, *et al.*, "On the immortality of television sets: "function" in the human genome according to the evolution-free gospel of ENCODE," *Genome Biology and Evolution*, vol. 5, no. 3, pp. 578–590, 2013.

[27] M.-Y. Kao, "Tree contractions and evolutionary trees," *SIAM Journal on Computing*, vol. 27, pp. 1592–1616, 1998.

[28] M.-Y. Kao, T.-W. Lam, W.-K. Sung, and H.-F. Ting, "An even faster and more unifying algorithm comparing trees via unbalanced bipartite matchings," *Journal*

of Algorithms, vol. 20, no. 2, pp. 212–233, 2001.

[29] Kubicka, G. Kubicki, and F. R. McMorris, "An algorithm to find agreement subtrees," *Journal of Classification*, vol. 12, pp. 91–100, 1995.

[30] Kubicka, G. Kubicki, and F. McMorris, "On Agreement Subtrees of Two Binary Trees," *Congressus numerantium*, vol. 88, pp. 217–224, 1992.

[31] J. M. Lang, A. E. Darling, and J. A. Eisen, "Phylogeny of bacterial and archaeal genomes using conserved genes: supertrees and supermatrices," *PloS One*, vol. 8, no. 4, e62510, 2013.

[32] C.-M. Lee, L.-J. Hung, M.-S. Chang, C.-B. Shen, and C.-Y. Tang, "An improved algorithm for the maximum agreement subtree problem," *Information Processing Letters*, vol. 94, pp. 211–216, 2005.

[33] Y. Lin, V. Rajan, and B. M. E. Moret, "A metric for phylogenetic trees based on matching," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 4, pp. 1014–1022, 2012.

[34] K. Liu, C. R. Linder, and T. Warnow, "Multiple sequence alignment: a major challenge to large-scale phylogenetics," *PLoS Currents*, vol. 2, RRN1198, 2010.

[35] K. Liu, C. R. Linder, and T. Warnow, "RAxML and FastTree: comparing two methods for large-scale maximum likelihood phylogeny estimation," *PloS One*, vol. 6, no. 11, e27731, 2011.

[36] R. McMorris, D. B. Meronk, and D. A. Neumann, "A view of some consensus methods for trees," *Numerical Taxonomy* (J. Felsenstein, ed.), Springer-Verlag, 1983, pp. 122–125.

[37] R. Munoz, P. Yarza, W. Ludwig, J. Euzéby, R. Amann, K.H. Schleifer, F. O. Glöckner, and R. Rosselló-Móra, “Release LTPs104 of the All-Species Living Tree,” *Systematic and Applied Microbioly*, vol. 34, pp.169–170, 2011.

[38] Ni. D. Pattengale, A. J. Aberer, K. M. Swenson, A. Stamatakis, and B. M. E. Moret, "Uncovering hidden phylogenetic consensus in large data sets," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 4, pp. 902–911, 2011.

[39] D. F. Robinson and L. R. Foulds, "Comparison of phylogenetic trees," *Mathematical Biosciences*, vol. 53, no. 1–2, pp. 131–147, 1981.

[40] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees," *Molecular Biology and Evolution*, vol. 4, pp. 406–425, 1987.

[41] R. R. Sokal and F. J. Rohlf, "The comparison of dendrograms by objective methods," *Taxon*, vol. 11, pp. 33–40, 1962.

[42] 42 A. Stamatakis, "RAxML Version 8: A tool for Phylogenetic Analysis and Post-Analysis of Large Phylogenies," *Bioinformatics*, vol. 30, no. 9, pp. 1312–1313, 2014.

[43] M. Steel and D. Penny, "Distribution of tree comparison metrics–some new results," *Systematic Biology*, vol. 42, no. 2, pp. 126–141, 1993.

[44] M. Steel and T. Warnow, "Kaikoura tree theorems: Computing the maximum agreement subtree," *Information Processing Letters*, vol. 48, pp. 77–82, 1993.

[45] K. M. Swenson, E. Chen, N. D. Pattengale, and D. Sankoff, "The kernel of maximum agreement subtrees," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, no. 4, pp. 1023–1031, 2012.

[46] D. L. Swofford, *PAUP*: Phylogenetic Analysis Using Parsimony* (** and Other Methods*), Sinauer Associates, Inc., 2002.

[47] B.-F. Wang and C.-Y. Li, "Fast algorithms for computing path-difference distances," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, accepted.

[48] M.S. Waterman and T. F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, vol. 73, no. 4, pp. 789–800, 1978.

[49] M. Wilkinson, "Common cladistic information and its consensus representation: reduced Adams and reduced cladistic consensus trees and profiles," *Systematic Biology*, vol. 43, no. 3, pp. 343–368, 1994.

[50] X. Zhou, X.-X. Shen, C. T. Hittinger, A. Rokas, "Evaluating fast maximum likelihood-based phylogenetic programs using empirical phylogenomic data sets," *Molecular Biology and Evolution*, vol. 35, no. 2, pp. 486–503, 2018.

Biing-Feng Wang received the BS degree in computer science from National Chiao Tung University, Taiwan, in June 1988 and the PhD degree in computer science from National Taiwan University in June 1991. In August 1993, he joined the Faculty of National Tsing Hua University, where he is a professor in the Department of Computer Science. His current research interests include design and analysis of algorithms and parallel computation. He received the Academia Sinica's Young Scholar Paper Award and the K.T. Li Young Researcher Award in 1999, the ISI Citation Classic Award in 2001, and the National Science Council's Outstanding Research Award in 2002 and 2018. He served as the Chairman of the Department of Computer Science from 2003 to 2006. He was awarded as a Tsing Hua Distinguished Professor in 2006 and as a Tsing Hua Chair Professor in 2018.

Krister M. Swenson received his BS degree in computer science from University of New Hampshire, and his PhD degree in computer science from Ecole Polytechnic Federal de Lausanne, Switzerland, in 2009. He did postdoctoral work in Canada at University of Ottawa, University of Quebec at Montreal (UQAM), University of Montreal, and McGill University. In 2013 he moved to France as a postdoctoral fellow at the University of Montpellier. He is currently an Associate Scientist at the Centre National de la Recherche Scientifique (CNRS), assigned to the Laboratoire d’Informatique, de Robotique, et de Microelectronique de Montpellier (LIRMM), University of Montpellier.