**DIMACS Technical Report 93-46**
**August 1993**

# Fast Comparison of Evolutionary Trees

by

Martin Farach[1,2]
Dept. of Computer Science
Rutgers University
New Brunswick, New Jersey 08903

Mikkel Thorup[3,4]
Dept. of Computing
Oxford University
Oxford, England

[1]Permanent Member
[2]Supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center under NSF contract STC-8809648.
[3]Dimacs Visitor
[4]Supported by the Danish Technical Research Council; Supported in part by the Danish Research Academy and by DIMACS under NSF contract STC-8809648.

DIMACS is a cooperative project of Rutgers University, Princeton University, AT&T Bell Laboratories and Bellcore.

DIMACS is an NSF Science and Technology Center, funded under contract STC–91–19999; and also receives support from the New Jersey Commission on Science and Technology.

## ABSTRACT

Constructing evolutionary trees for species sets is a fundamental problem in biology. Unfortunately, there is no single agreed upon method for this task, and many methods are in use. Current practice dictates that trees be constructed using different methods and that the resulting trees then be compared for consensus. It has become necessary to automate this process as the number of species under consideration has grown. We study the *Unrooted Maximum Agreement Subtree Problem* (`UMAST`) and its rooted variant (`RMAST`).

The `UMAST` problem is as follows: given a set $A$ and two trees $T_0$ and $T_1$ leaf-labeled by the elements of $A$, find a maximum cardinality subset $B$ of $A$ such that the restrictions of $T_0$ and $T_1$ to $B$ are topologically isomorphic. Our main result is an $O(n^{2+o(1)})$ time algorithm for the UMAST problem. As a side-effect we will derive an $O(n^2)$ time algorithm for the RMAST problem. The previous best algorithm for both these problems has running time $O(n^{4.5+o(1)})$.

# 1 Introduction

An *evolutionary tree* for a species set $A$ is a tree in which the leaves are uniquely labeled by the species in $A$, and the internal nodes represent ancestors. The standard models of computation for constructing these trees use either *characters* or *distance matrices*. Both approaches have their problems, not the least of which is that most tree construction criteria are *NP*-hard to optimize [2, 3].

As is typically the case when there is no really good solution to a problem, the number of solutions actually in use is quite large. Within the biology literature, various heuristics have been proposed (see e.g. [5, 6, 8, 15, 16]). More recently, a variety of solutions have been examined from an algorithmic point of view ([1, 4, 11, 12]).

Not surprisingly, these various methods don't always give the same answer on the same inputs. Given that there is no "gold standard" for constructing evolutionary trees, current practice dictates that several different methods be applied to the data. The resulting trees are then compared in order to arrive at some consensus. Mostly, this consensus computation is done by hand in some *ad hoc* manner. However, as the number of species under consideration in these types of studies grows, this labor-intensive method is becoming prohibitively time-consuming. Finden and Gordon [7] formalized the consensus problem as follows.

**Problem:** The Unrooted Maximum Agreement Subtree Problem (`UMAST`)

**Input:** A pair $(T_0, T_1)$ of unrooted evolutionary trees with labels from a common set $A$.

**Output:** A maximum cardinality subset $B$ of $A$ such that $T_0|B$ and $T_1|B$ are isomorphic.

Define $\mathtt{meet}(u, v, w)$ to be the vertex shared by the simple paths from $u$ to $v$, $u$ to $w$, and $v$ to $w$. Then by $T|B$, where $T$ is an evolutionary tree and $B$ is a subset of the leaf labels, we denote the tree whose vertices are vertices from $T$ that are either leaves of $T$ with labels from $B$, or the `meet` of triples of these leaves. The edges in $T|B$ are obtained as replacements of the paths in $T$ between the vertices selected for $T|B$. We refer to $T|B$ as the *restriction of $T$ to $B$*. The isomorphism from $T_0|B$ to $T_1|B$ is an adjacency preserving bijection such that if $l_0, l_1, l_2$ are leaves of $T_0|B$ and $l'_0, l'_1, l'_2$ are leaves of $T_1|B$ with the same labels, then $\mathtt{meet}(l_0, l_1, l_2)$ is mapped to $\mathtt{meet}(l'_0, l'_1, l'_2)$.

A rooted tree variant on this problem is defined similarly:

**Problem:** The Rooted Maximum Agreement Subtree Problem (`RMAST`)

**Input:** A pair $(R_0, R_1)$ of rooted evolutionary trees with labels from a common set $A$.

**Output:** A maximum cardinality subset $B$ of $A$ such that $R_0|^r B$ and $R_1|^r B$ are isomorphic.

Here $R|^r B$ denotes the rooted evolutionary trees whose vertices are the closure of the leaves with labels in $B$ under the `lca` operation, where the $\mathtt{lca}(u, v)$ is the least common ancestor of nodes $u$ and $v$. The arcs in $R|^r B$ are obtained as replacements of the dipaths in $T$ between the vertices selected for $R|^r B$.

Notice that UMAST is at least as hard as RMAST, for suppose we have an oracle for UMAST and that we want to solve RMAST for two rooted evolutionary trees $R_0$ and $R_1$ with label set $A$. First, build an evolutionary star with $|A|$ leaves and new labels. Next, attach the center of this star both to the root of $R_0$ and to the root of $R_1$. Third, apply the UMAST oracle to the augmented trees. Finally, remove the star from the returned maximal agreement subtree to get a rooted maximum agreement subtree of $R_0$ and $R_1$. Clearly, the above describes a linear time reduction.

We will restrict ourselves in the discussion below to simply finding the cardinality of the set $B$. However, it is a trivial modification to augment our algorithms to output, within the same time bounds, a particular such $B$.

Finden and Gordon gave a heuristic method for computing the maximum agreement subtree of two rooted binary trees. Their algorithm, which has a $O(n^5)$ running time, does not, however, guarantee an optimal solution. In [14], Kubicka et al. presented a $O(n^{(\frac{1}{2}+\epsilon)\log_2 n})$ time algorithm for the binary UMAST problem. Steel and Warnow [17] also considered the UMAST problem, giving the first polynomial time algorithm for the general UMAST problem, i.e. the case with no degree bound. Their algorithm, which we will refer to as SW, is a dynamic programming approach which runs in $O(n^2)$ time on bounded degree trees and in $O(n^{4.5}\log n)$ time on unbounded degree trees. They report the same bounds for rooted trees.

The main result of this paper is an $O(n^2 c^{\sqrt{\log n}}) = O(n^{2+o(1)})$ time algorithm for the general UMAST problem. In addition we will derive an $O(n^2)$ algorithm for the general RMAST problem. Thus our results dramatically narrow the gap, closing it in the rooted case, between the bounded degree and general version of these problems. Notice that high degree trees are not just an algorithmic curiosity. Within hierarchical clusterings, large degree nodes are used to capture uncertainty about the relationship amongst species. Therefore, making assumptions that the degree of the input trees is low is equivalent to saying that there is little ambiguity in the data. If this is the case, all reasonable tree construction algorithms will produce very similar trees. It is only in the high-degree case that tree comparison methods are needed.

Our algorithms take starting point in the SW algorithm. The SW algorithm is based on a dynamic program which, for each of the $O(n^2)$ pairs of edges from the opposing trees, deletes those edges and computes RMAST of the rooted sub-trees produced. Each step of the dynamic program boils down to a weighted bipartite matching problem. For bounded degree, each matching problem is of constant size, giving the complexity of $O(n^2)$. For unbounded degree, each matching takes $O(n^{2.5}\log n)$ time, so the bound of $O(n^{4.5}\log n)$ is achieved by summing over all the constituent matchings.

Our results are achieved by identifying structural components of the trees which either participate in an agreement tree, or force the agreement trees to be quite small. As in the SW algorithm, we must compute at least one weighted matching for each pair of nodes. Our structural analysis allows us to dramatically reduce the total work of computing the matchings in two ways. First, the SW algorithm ends up computing many matchings for each pair of nodes. The number of matchings computed per node pair is proportional to the product of the degrees of the nodes. We compute a very small number of matchings per node pair. Furthermore, we use the same structural analysis to show that the weighted matchings can be made sparse enough to no longer be the bottleneck of the computation.

Finally, we point out that one of the main strengths of our algorithm, besides the dramatic speed-up it provides, is its simplicity. At the end of this paper, we give a complete description of the algorithm, summarizing the various lemmas that make up the text of our paper.

In §2, we give an overview of our algorithm. In §3, we show how to reduce the bipartite matching work for unrooted trees. In §4, we describe how to compute $\mathtt{RMAST}$ in $O(n^2)$. We conclude in §5 with a complete (and concise) description of the rooted and unrooted MAST algorithms.

# 2 Unrooted Trees

Fix the two unrooted evolutionary trees $\mathcal{T}_0$ and $\mathcal{T}_1$ of size $n$ for which we want to compute $\mathtt{UMAST}$. The size of an evolutionary tree refers to the number of species or leaves. Clearly $\mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = n$ if $n < 3$, so we may assume that $n \geq 3$. In particular, this implies that $\mathcal{T}_0$ and $\mathcal{T}_1$ have interior vertices.

We introduce a bit of notation before introducing the main structural lemma that relates rooted and unrooted MAST computations. For tree $T$, we take $\mathtt{V}(T)$ to be its vertex set, $\mathtt{V}^I(T)$ to be the set of its interior vertices, $\mathtt{L}(T)$ to be its leaves, and $\mathtt{E}(T)$ to be its edge set. For evolutionary tree $T$, we let $\mathtt{A}(T)$ be the set of leaf labels of $T$. For $v \in \mathtt{V}(T)$, we use $\mathtt{NE}(v)$ to mean the set of directed edges $(v, u)$ such that $\{u, v\} \in \mathtt{E}(T)$. If $T$ is an unrooted tree and $v \in \mathtt{V}(T)$, by $T^v$ we denote rooted tree obtained by rooting $T$ in $v$. If $R$ is a rooted tree and $v \in \mathtt{V}(R)$, by $R^v$ we denote the sub-tree of $R$ descending from $v$. Notice that if $u, v, w$ are vertices occurring in this order on a path in a tree $T$, then $T^{uw} = T^{vw}$. Let $G = (V, E, W)$ be a weighted bipartite graph. Then $\mathtt{MWM}(G)$ is the value of a maximum weighted matching on $G$.

The following lemma gives a reduction from $\mathtt{UMAST}$ to $\mathtt{RMAST}$. Steel and Warnow [17] based their algorithm on a similar but different reduction.

**Lemma 2.1**

$$\mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \max_{v_j \in \mathtt{V}^I(\mathcal{T}_j)} \{\mathtt{umast}(v_0, v_1)\},$$

*where*

$$\mathtt{umast}(v_0, v_1) = \mathtt{MWM}((\mathtt{NE}(v_0) \cup \mathtt{NE}(v_1), \mathtt{NE}(v_0) \times \mathtt{NE}(v_1), \mathtt{rmast}(\cdot)))$$

*and*

$$\mathtt{rmast}((v_0, w_0), (v_1, w_1)) = \mathtt{RMAST}(\mathcal{T}_0^{v_0 w_0}, \mathcal{T}_1^{v_1 w_1}).$$

**Proof:** Clearly, for all $(v_0, v_1) \in \mathtt{V}^I(\mathcal{T}_0) \times \mathtt{V}^I(\mathcal{T}_1)$, the value of $\mathtt{umast}$ corresponds to some agreement subtree, so $\mathtt{umast}(v_0, v_1) \leq \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$. Hence the lemma follows if we can find a specific pair achieving equality.

Let $B$ be a maximum subset of the labels such that $\mathcal{T}_0|B \cong \mathcal{T}_1|B$, i.e. $|B| = \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$. For any vertex $v$ in $\mathcal{T}_0|B$ the corresponding vertex in $\mathcal{T}_1|B$ is denoted $v'$. Fix any interior vertex $v$ in $\mathcal{T}_0|B$. We will show that $\mathtt{umast}(v, v') \geq |B| = \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$.

Let $w_0, \ldots, w_n$ be the neighbors of $v$ in $\mathcal{T}_0|B$. Set $B_i = \mathtt{A}((\mathcal{T}_0|B)^{vw_i}) = \mathtt{A}((\mathcal{T}_1|B)^{v'w'_i})$. Then the $B_i$s are disjoint and $B = \bigcup_i B_i$. For $i = 0, \ldots, n$, let $v_i$ be the neighbor of $v$ on the way to $w_i$ in $\mathcal{T}_0$, and let $v'_i$ be the neighbor of $v'$ on the way to $w'_i$ in $\mathcal{T}_1$ (so we extend the isomorphism $(\cdot)'$ with some degree 2 vertices).

Consider some specific $i$. We want to show that $B_i$ is a rooted agreement subset for $\mathcal{T}_0^{vv_i}$ and $\mathcal{T}_1^{v'v'_i}$, for then, by definition, $\mathtt{rmast}((v, v_i), (v', v'_i)) \geq |B_i|$, and hence $\mathtt{umast}(v, v') \geq |B|$. But this follows since

$$(\mathcal{T}_0^{vv_i}|^r B_i) = (\mathcal{T}_0|B_i)^{vw_i} \cong (\mathcal{T}_1|B_i)^{v'w'_i} = (\mathcal{T}_0^{v'v'_i}|^r B_i).$$

▮

If, for $(v_0, v_1) \in \mathtt{V}^I(\mathcal{T}_0) \times \mathtt{V}^I(\mathcal{T}_1)$, $\mathtt{umast}(v_0, v_1) = \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$, we say that $v_0$ *maps to* $v_1$. Note that the computation of $\mathtt{umast}(v_0, v_1)$ relies on the $\mathtt{rmast}$ values for various neighboring nodes. We will informally refer to these values needed for such a local computation as the *base* of a node pair.

Lemma 2.1 immediately suggests that there are two problems to be solve in computing the $\mathtt{UMAST}$:

- *How do we compute the weighted matchings?* This can be done naïvely in $O(n^2 n^{2.5} \log n)$.
- *How do we compute the bases of the node pairs?* Let
  $\mathtt{C\text{-}RMAST}(R_0, R_1), = \{\mathtt{RMAST}(R_0^{v_0}, R_1^{v_1}) | v_j \in \mathtt{V}(R_j), j \in \{0, 1\}\}$. In §4, we will show an $O(n^2)$ algorithm for computing $\mathtt{C\text{-}RMAST}$. The naïve way of computing the bases for all pairs of interior nodes $(i_0, i_1)$ in $\mathtt{V}^I(\mathcal{T}_0) \times \mathtt{V}^I(\mathcal{T}_1)$, is to compute $\mathtt{C\text{-}RMAST}(\mathcal{T}_0^{i_0}, \mathcal{T}_1^{i_1})$ for each such pair. Given our $O(n^2)$ algorithm for $\mathtt{C\text{-}RMAST}$, this gives a complexity of $O(n^4)$ for computing the bases for all node pairs.

We speed up the computation for both subtasks by introducing the concept of a *$\kappa$-core tree.* Let $T$ be an $n$ leaf tree and $\kappa$ some parameter to be fixed later. We say that $e \in \mathtt{E}(T)$ is a *core edge* if each component of $T - e$ has at least $n/\kappa$ leaves. We say that a node is a *core node* if it is adjacent to a core edge. A core node is *critical* if the number of its core neighbors is different from two. All core edges and nodes make up the *core tree.* The components created by removing the core tree are the *side trees.*

We denote by $\mathcal{C}_i$ the core tree of tree $\mathcal{T}_i$. Note that the core tree is indeed a tree, since it is connected. Further, note that each core tree has at most $\kappa$ leaves, for if we remove all core edges, each core leaf will be in a component with at least $n/\kappa$ tree leaves. Therefore we have $O(\kappa)$ critical nodes.

We can now divide the computation into two cases: either there is some pair $(c_0, c_1) \in \mathtt{V}(\mathcal{C}_0) \times \mathtt{V}(\mathcal{C}_1)$ such that $c_0$ maps to $c_1$, or no such pair exists. Accordingly, we define *core-core* to be the maximum $\mathtt{umast}$ value for core node pairs, and let *non-core-core* denote the maximum over all other pairs. Clearly, the $\mathtt{UMAST}$ of two trees is the maximum over the core-core and non-core-core values.

## 2.1 Computing core-core

By Lemma 2.1, the computation of core-core depends on the bases for core node pairs. The following observation suggests an algorithm for such a computation.

**Observation 2.1.1**

$$\{\texttt{rmast}(e_0, e_1) | e_j \in \texttt{NE}(c_j), c_j \in \texttt{V}(\mathcal{C}_j), j \in \{0,1\}\} \subseteq \bigcup \{\texttt{C-RMAST}(T_0^{l_0}, T_1^{l_1}) | l_j \in L(\mathcal{C}_j)\},$$

*where $L(\mathcal{C}_j)$ is the set of core leaves of $\mathcal{C}_j$.*

In other words, we can compute the C-RMAST of core-leaf rooted trees to get the bases for all core node pairs. Since each core tree has at most $\kappa$ leaves, we can complete this computation in time $O(\kappa^2 n^2)$, using the $O(n^2)$ time C-RMAST algorithm of §4.

Finally, given the bases of the core node pairs, we will show in §3 that we can find the matchings for all of them in time $O((\kappa n)^{1.5} \log n + n^2)$ (by an appropriate thinning argument). Thus core-core can be computed in $O(\kappa^2 n^2)$ total time.

## 2.2 Computing non-core-core

After having computed core-core, we still need to know the maximum umast over all other pairs of nodes. We will take a slightly indirect route to compute non-core-core. Suppose we know that that no core node maps to another. Then core-core < non-core-core = UMAST. But in such a case, it seems intuitive that the final agreement subset must be quite small, since eliminating core-core mappings forces all the core nodes of one tree to map into a single side tree of the other tree. The following lemma confirms this intuition and allows us to compute non-core-core efficiently.

**Lemma 2.2** *Let $\mathcal{T}_0$ and $\mathcal{T}_1$ be two unrooted evolutionary trees with core trees $\mathcal{C}_0$ and $\mathcal{C}_1$. Suppose that for all $(c_0, c_1) \in \texttt{V}(\mathcal{C}_0) \times \texttt{V}(\mathcal{C}_1)$, $c_0$ does not map to $c_1$. Then there are side trees $t_0$ of $\mathcal{T}_0$ and $t_1$ of $\mathcal{T}_1$ such that $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \texttt{UMAST}(\mathcal{T}_0|B, \mathcal{T}_1|B)$, where $B = \texttt{A}(t_0) \cup \texttt{A}(t_1)$.*

**Proof:** Let $B$ be a maximal agreement subset of maximal size, that is, let $|B| = \texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$.

For all $v \in \texttt{V}(\mathcal{T}_0|B)$, denote the corresponding vertex in $v \in \texttt{V}(\mathcal{T}_1|B)$ by $v'$. Fix $v, w \in \texttt{V}(\mathcal{T}_0|B)$ such that $v \in \texttt{V}(\mathcal{C}_0)$ and $w' \in \texttt{V}(\mathcal{C}_1)$. If such a $v$ cannot be found, $T_0|B$ is contained in a side tree of $T_0$ so the result is trivial, and similarly if $w$ cannot be found. By the hypothesis of the lemma, $v' \notin \texttt{V}(\mathcal{C}_1)$ and $w \notin \texttt{V}(\mathcal{C}_0)$. Denote by $P$ the path in $\mathcal{T}_0|B$ from $v$ to $w$. Let $v_0$ be the last core vertex in $P$, and let $v_1$ be its successor in $P$. Then $v_0$ separates $v_1$ from the core vertices in $\mathcal{T}_0$, and $v_1'$ separates $v_0'$ from the core vertices in $\mathcal{T}_1$. Hence $\mathcal{T}_0^{v_0 v_1}$ and $\mathcal{T}_1^{v_1' v_0'}$ are contained in a side tree of $\mathcal{T}_0$ and $\mathcal{T}_1$, respectively. This implies the lemma, for since $(v_0, v_1)$ is an edge in $\mathcal{T}_0|B$, we have $B = B_0 \cup B_1$ where $B_0 = \texttt{A}((\mathcal{T}_0|B)^{v_0 v_1})$ $(= \texttt{A}((\mathcal{T}_1|B)^{v_0' v_1'}))$ and $B_1 = \texttt{A}(\mathcal{T}_1^{v_1' v_0'})$ $(= \texttt{A}(\mathcal{T}_0^{v_1 v_0}))$. ∎

Now define side-side to be the recursively computed maximum over all the side tree pairs, as described in Lemma 2.2. Then Lemma 2.2 tells us that if core-core is less than side-side, then non-core-core equal side-side; otherwise we just know that side-side is bounded by `UMAST`. Thus `UMAST` is simply the maximum of core-core and side-side. We therefore focus on computing side-side instead of non-core-core.

We have noted that we can recursively compute side-side. However, we require an efficient algorithm for the restriction operation. The following lemma provides such an algorithm.

**Lemma 2.3** *Given a partition $B_0, \ldots, B_x$ of the labels of an evolutionary tree $T$ of size $n$, we can compute all of $T|B_0, \ldots, T|B_x$ in total $O(n)$ time.*

**Proof:** Suppose $T$ is rooted. We can impose an arbitrary ordering on the children of each node and produce an in-order traversal of $T$. To each leaf $l$ we assign the pair of $\langle p, i \rangle$ where $p$ is the partition number of $l$ and $i$ is its in-order number. We can radix sort the leaves lexicographically in $O(n)$ time.

For each internal node we can, in $O(n)$ time, assign to each node its depth and preprocess the tree so least `lca` can be answered in constant time [10]. Now, for each partition $B = B_i$, we can compute an in-order traversal of $T|B$ as follows. Let $l_1, \ldots, l_k$ be the in-order list of the leaves of $T|B$. Let $l'_{2i-1} = l_i$ and let $l'_{2i} = \texttt{lca}(l_i, l_{i+1})$. Now $l'_1, \ldots, l'_{2k-1}$ is the in-order traversal of $T|B$. From the $l'_i$ ordering, together with the level information for internal nodes, we can construct $T|B$ in $O(k)$ time [13], thus giving $O(n)$ time to build all trees.

Now assume that $T$ is unrooted. Pick an arbitrary internal node $v$ of $T$ and compute the partition $B_1, \ldots, B_x$ on $T^v$. But each $T^v|B_i$ is the same as $T|B_i$, with the exception of the root – which can be a degree two node. Finally, we simply unroot the computed trees, and suppress the root node, if it has degree two. Thus, the partition of an unrooted tree can also be computed in $O(n)$ time. ▮

**Lemma 2.4** *Let $U(n)$ be the time needed to compute the* `UMAST` *of two $n$ leaf trees. Then side-side can be computed in time $O(n + \kappa^2 U(2n/\kappa))$.*

**Proof:** Clearly the lemma would follow if there were only $O(\kappa)$ side trees, since each side tree contains at most $n/\kappa$ leaves. Unfortunately, there can be $O(n)$ small side trees.

Notice, however, that if $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \texttt{UMAST}(\mathcal{T}_0|B, \mathcal{T}_1|B)$, for some $B \subseteq \texttt{A}(\mathcal{T}_0)$, then $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \texttt{UMAST}(\mathcal{T}_0|B', \mathcal{T}_1|B')$, for any $B'$, $B \subseteq B' \subseteq \texttt{A}(\mathcal{T}_0)$. Hence, we may combine small side trees into at most $2\kappa$ side forests each of size at most $n/\kappa$ and then recurse on the union of label sets of opposing pairs of these forests. ▮

## 2.3 Overall complexity

Combining Lemma 2.4 with the announced complexity for computing core-core, we have that there is a constant $c$ such that

$$U(n) \leq c\kappa^2 n^2 + c\kappa^2 U(2n/\kappa).$$

Solving this recurrence and setting $\kappa = \alpha^{\sqrt{\log n}}$, for some constant $\alpha$ that depends on $c$, we conclude with:

**Theorem 2.5** *The* UMAST *problem can be solved in $O(n^2 \beta^{\sqrt{\log n}})$, for some constant $\beta$.*

# 3 Core Matchings

Recall that we must compute the maximum weighted matchings on graphs defined by $G_{c_0 c_1} = (\mathtt{NE}(c_0) \cup \mathtt{NE}(c_1), \mathtt{NE}(c_0) \times \mathtt{NE}(c_1), \mathtt{rmast}(\cdot))$ for all $(c_0, c_1) \in \mathtt{V}(\mathcal{C}_0) \times \mathtt{V}(\mathcal{C}_1)$. We note that an edge in $G_{c_0 c_1}$ is defined over a pair of directed edges in $\mathtt{E}(\mathcal{T}_0) \times \mathtt{E}(\mathcal{T}_1)$. To avoid confusion, we will always explicitly state if a node or an edge is thought of as coming from the trees or the matching graph. Thus a matching edge is a pair of directed tree edges each of whose tails are core tree nodes.

The aim of this section is to thin out the matching edges so that most of the $O(n^2)$ matchings becomes of some fixed constant size, and such that the remaining matchings contains a total of $O(\kappa n)$ matching edges. In general, our scheme will be first to show that many matching edges must have weight zero, and then that some of the non-zero weighted edges can be set to zero, thus making the matching graphs sparse. Since the total number of edges in all matching graphs is $O(n^2)$, even assuming the graphs are complete (recall that each edge in a matching graph represents a distinct pair of tree edges), we can trivially delete all zero weight edges in $O(n^2)$ time. Assume in the following that this has been done.

We partition the matching nodes into three types: *side matching nodes* are those whose tree edge is incident on a side tree; *critical matching nodes* are those whose tree edges have critical nodes as their heads; and *core matching nodes* are all others. This gives rise to the following partition of matching edges: *side-side matching edges* are derived from two side matching nodes; *critical matching edges* are incident on at least one critical matching node; and *core matching edges* are all others.

Matching graphs come in two varieties: critical graphs are those $G_{c_0 c_1}$ such that either $c_0$ is critical in $\mathcal{C}_0$ or $c_1$ is critical in $\mathcal{C}_1$; core graphs are all others. We bound the work separately.

**Lemma 3.1** *All matchings on critical graphs can be completed in $O((\kappa n)^{1.5} \log n)$.*

**Proof:** First, note that there are only $O(\kappa)$ directed tree edges from critical tree nodes to core tree nodes. Therefore, there are only $O(\kappa n)$ critical matching edges. If there are $O(\kappa n)$ edges, then there are $O(\kappa n)$ nodes, and since the weight of an edge is an `rmast` value, no edge can have weight more than $n$. Using the weighted bipartite matching algorithm of [9] gives the desired bounds. ∎

We bound the work for each core matching graph separately as follows.

**Lemma 3.2** *Let $G_{c_0 c_1}$ be a core matching graph with $k$ side-side matching edges. Then there is a graph $SG_{c_0 c_1}$ with no more than $5k + 8$ edges such that $\mathtt{MWM}(SG_{c_0 c_1}) = \mathtt{MWM}(G_{c_0 c_1})$ and such that $SG_{c_0 c_1}$ can be computed from $G_{c_0 c_1}$ in linear time.*

**Proof:** Let $V$ be the set of the nodes incident on the $k$ side-side matching edges. There are 4 core matching nodes in $G_{c_0c_1}$, so we can leave in the at most $4k$ matching edges between core edges and members of $V$. This gives $5k$ edges.

Consider $c$, one of the 2 core nodes in one of the independent sets. In a maximum matching, if $c$ is not adjacent to a node in $V$, it must be adjacent to one of its 2 other heaviest neighbors. Therefore, we can set all other edges incident on $c$ to zero (ties can be broken arbitrarily). This gives 2 more edges for each of the 4 core nodes, giving a total of 8 edges, and proving the lemma. ▮

**Lemma 3.3** *The core matchings can be computed in* $O((\kappa n)^{1.5}\log n + n^2)$ *time.*

**Proof:** By lemma 3.1, the critical matchings take $O((\kappa n)^{1.5}\log n)$ total time. For the $O(n^2)$ core matchings, first we apply the thinning from Lemma 3.2. There can be at most $n$ side-side matching edges since each side-side edge requires a common label in a pair of side trees, and these labels are partitioned amongst the side trees. Hence, in the matchings with more than 8 matching edges, there can be a total of at most $13n$ matching edges (the factor decreases towards 5 as the number of side-side matching edges per matching increases). Clearly, the $O(n^2)$ constant ($\leq 8$) size matchings can be computed in time $O(n^2)$, and from the linear bound on the number of edges in the remaining matchings, it follows that they can be computed in time $O(n^{1.5}\log n)$. We get a total matching time of $O((\kappa n)^{1.5}\log n + n^2)$ for all matchings. ▮

# 4 Rooted Trees

Fix $\mathcal{R}_0$ and $\mathcal{R}_1$ as the trees on which we want to compute `C-RMAST`. We start with some notation.

Let $\mathtt{C}(v)$ be the set of children of $v$. Let $v_0$ and $v_1$ be the roots of two trees. Then we set $\mathtt{Diag}(v_0, v_1) = \{\mathtt{rmast}(v_0, w_1) | w_1 \in \mathtt{C}(v_1)\} \cup \{\mathtt{rmast}(w_0, v_1) | w_0 \in \mathtt{C}(v_0)\}$ and set $\mathtt{match}(v_0, v_1) = \mathtt{MWM}((\mathtt{C}(v_0) \cup \mathtt{C}(v_1), \mathtt{C}(v_0) \times \mathtt{C}(v_1), \mathtt{rmast}(\cdot)))$. Note that we are now applying the `rmast` function to pairs of nodes, whereas we have been using `rmast` over pairs of edges. For the sake of brevity and notational simplicity, we allow ourselves this slight abuse of notation noting that if $R = T^v$, then $R^u = T^{vu}$, i.e. we can define rooted subtrees of unrooted trees by directed edges, while we need only a node to define rooted subtrees of rooted trees.

The following lemma appears in Steel and Warnow [17] and is the basis for their dynamic programming approach to this problem.

**Lemma 4.1 ([17])** $\forall v_0 \in \mathtt{V}(\mathcal{R}_0), v_1 \in \mathtt{V}(\mathcal{R}_1)$,

$$\mathtt{rmast}(\mathcal{R}_0^{v_0}, \mathcal{R}_1^{v_1}) = \begin{cases} |\mathtt{A}(\mathcal{R}_0^{v_0}) \cap \mathtt{A}(\mathcal{R}_1^{v_1})| & \textit{if } v_0 \textit{ or } v_1 \textit{ is a leaf} \\ \max\{\mathtt{Diag}(v_0, v_1), \mathtt{match}(v_0, v_1)\} & \textit{otherwise.} \end{cases}$$

Intuitively, this expression says that when comparing the roots of two trees, we can either match the two roots together in the final agreement tree, in which case we find the best way of matching their children together, or we can match the root of one tree to be one of the children of the other tree.

Trivially, this lemma implies a dynamic program with running time bounded by $O(n^2)$ plus the time spent on computing `match`. As before, we reduce the matching time by some preprocessing.

**Lemma 4.2** *In the dynamic program to compute* `C-RMAST`*, all matchings can be made to take $O(n^2)$ time.*

**Proof:** For each internal node $v$ of each tree $\mathcal{R}_j$, we select a *heavy child* to one of $\mathtt{C}(v)$ with a maximal number of descendant leaves. All other children of $v$ are said to be *light*.

By an argument similar to that in the proof of Lemma 3.2, we can reduce the edges in a matching graph $G$ to $3k+2$, where $k$ is the number of light-light edges in $G$ (instead of two core matching nodes in each independent set, we have one heavy node). If we set $K$ to be the total number of light-light edges, we get a total of $O(n^2 + K^{1.5} \log n)$.

To bound $K$, notice that each leaf has $O(\log n)$ light ancestors. So, given any label, it can give rise to $O(\log^2 n)$ light-light edges, for a total of $O(n \log^2 n)$ light-light edges. ∎

We conclude with the following:

**Theorem 4.3** `C-RMAST`*, and therefore* `RMAST`*, can be computed in $O(n^2)$.*

# 5 Summing up

We present the complete pseudo-code for computing `UMAST` in $O(n^{2+o(1)})$ time and the `RMAST` in $O(n^2)$.

**Procedure 1** $\mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$*: Computes the* `UMAST` *of $\mathcal{T}_0$ and $\mathcal{T}_1$.*

- $UMAST \leftarrow 0$, $\kappa \leftarrow \alpha^{\sqrt{\log n}}$
- Identify the $\kappa$-core trees of $\mathcal{T}_0$ and $\mathcal{T}_1$.
- Partition sides trees into balanced side forests of sizes between $n/(2k)$ and $n/k$.
- For all pairs $(f_0, f_1)$ of opposing side forests:
  - $B \leftarrow \mathtt{A}(f_0) \cup \mathtt{A}(f_1)$
  - $UMAST \leftarrow \max(UMAST, \mathtt{UMAST}(T_0|B, T_1|B))$
- For all pairs $(l_0, l_1)$ of opposing core leaves:
  - Compute $\mathtt{C\text{-}RMAST}(T_0^{l_0}, T_1^{l_1})$
- For all pairs $(v_0, v_1)$ of opposing core vertices
  - $UMAST \leftarrow \max(UMAST, umast(v_0, v_1))$
- Return $UMAST$.

**Procedure 2** $\mathtt{umast}(v_0, v_1)$ : *Performs local* `umast` *computation on nodes $v_0$, $v_1$.*

- Build the matching graph G. Remove all zero edges.
- If neither $v_0$ nor $v_1$ is critical:
  - For each core child, remove all but the two heaviest of the matching edges not incident with side nodes adjacent to side nodes.
- Return $\mathtt{MWM}(G)$.

**Procedure 3** $\mathtt{C\text{-}RMAST}(\mathcal{R}_0, \mathcal{R}_1)$ : *Compute the complete* RMAST *for rooted tree* $\mathcal{R}_0$ *and* $\mathcal{R}_1$*.*

- For each node in each tree, pick a heavy child.
- For each tree, take the partial ordering of subtree inclusion and produce a total ordering. From these two orderings, produce a total ordering over all pairs of opposing subtrees.
- Loop on the total order of subtree pairs and let $(v_0, v_1)$ be the current pair:
  - If either $v_0$ or $v_1$ is a leaf, return $|\mathtt{A}(\mathcal{R}_0^{v_0}) \cap \mathtt{A}(R_1^{v_1})|$
  - else Set $\mathtt{rmast}(v_0, v_1) = \max(\mathtt{Diag}(v_0, v_1), \mathtt{match}(v_0, v_1))$.

**Procedure 4** $\mathtt{match}(G)$ : *computes the maximum weighted matching on bipartite graphs with two heavy nodes.*

- Remove all zero edges.
- For each heavy node, remove all but the heaviest of the matching edges not incident with light nodes adjacent to other light nodes.
- Return $\mathtt{MWM}(G)$.

## References


[1] R. Agarwala and D. Fernandez-Baca. A polynomial-time algorithm for the phylogeny problem when the number of character states is fixed. Technical Report 93-04, DIMACS, 1993.

[2] H. Bodlaender, M. Fellows, and T. Warnow. Two strikes against perfect phylogeny. *Proc. of 19th International Colloquium on Automata Languages and Programming*, 1992.

[3] W.H.E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.

[4] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 1993. In press. See also STOC '93.

[5] J.S. Farris. Estimating phylogenetic trees from distance matrices. *Am. Nat.*, 106:645–668, 1972.

[6] J. Felsenstein. Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology*, 57(4), 1982.

[7] C. R. Finden and A. D. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.

[8] W.M. Fitch and E. Margoliash. The construction of phylogenetic trees. *Science*, 155:29–94, 1976.

[9] H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[10] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestor. *Computer and System Science*, 13:338–355, 1984.

[11] S. Kannan, E. Lawler, and T. Warnow. Determining the evolutionary tree. *Proc. of the First Ann. ACM-SIAM Symp. on Discrete Algorithms*, Jan. 1990.

[12] S. Kannan and T. Warnow. Using experiments to infer evolutionary trees. Manuscript, 1992.

[13] Donald E. Knuth. *The Art of Computer Programming*, volume 3: Sorting and Searching. Addison-Wesley, Reading, 1973.

[14] E. Kubicka, G. Kubicki, and F.R. McMorris. An algorithm to find agreement subtrees. To apprear in Journal of Classification, 1992.

[15] N. Saitou and M. Nei. The neighbor-joinging method: a new method for reconstructing phylogentic trees. *Mol. Biol. Evol.*, 4:406–424, 1987.

[16] R. Sokal and P. Sneath. *Numberical Toxonomy*. Freeman, 1963.

[17] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. Submitted for publication.