[KLW90] S. Kannan, E. Lawler, and T. Warnow. Determining the evolutionary tree. *Proc. of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 475–484, 1990.

[Knu73] D. E. Knuth. *The Art of Computer Programming, V. 3: Sorting and Searching.* Addison-Wesley, Reading, 1973.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogentic trees. *Mol. Biol. Evol.*, 4:406–424, 1987.

[SS63] R. Sokal and P. Sneath. *Numerical Taxonomy.* Freeman, 1963.

[SW93] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48:77–82, 1993.

References

[AFB94] R. Agarwala and D. Fernandez-Baca. A polynomial-time algorithm for the phylogeny problem when the number of character states is fixed. *Proc. of the 34th IEEE Annual Symp. on Foundation of Computer Science*, pages 140–147, 1994.

[AK94] A. Amir and D. Keselman. Maximum agreement subtrees in multiple evolutionary trees. *Proc. of the 35th IEEE Annual Symp. on Foundation of Computer Science*, pages 758–769, 1994.

[BFW92] H. Bodlaender, M. Fellows, and T. Warnow. Two strikes against perfect phylogeny. *Proc. of 19th International Colloquium on Automata Languages and Programming*, pages 273–283, 1992.

[Day87] W.H.E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.

[Far72] J.S. Farris. Estimating phylogenetic trees from distance matrices. *Am. Nat.*, 106:645–668, 1972.

[Fel82] J. Felsenstein. Numerical methods for inferring evolutionary trees. *The Quarterly Review of Biology*, 57(4), 1982.

[FG85] C. R. Finden and A. D. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.

[FKW93] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 1993. In press. See also STOC '93.

[FM76] W.M. Fitch and E. Margoliash. The construction of phylogenetic trees. *Science*, 155:29–94, 1976.

[FT94a] M. Farach and M. Thorup. Fast comparison of evolutionary trees (extended abstract). *Proc. of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 481–488, 1994.

[FT94b] M. Farach and M. Thorup. Sparse dynamic programming for evolutionary tree comparison. *Proc. of the 35th IEEE Annual Symp. on Foundation of Computer Science*, pages 770–779, 1994.

[GT89] H. Gabow and R. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[HT84] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13:338–355, 1984.

[KKM94] E. Kubicka, G. Kubicki, and F.R. McMorris. An algorithm to find agreement subtrees. To appear in *Journal of Classification*, 1994.

- $UMAST \leftarrow 0$, $\kappa \leftarrow 4^{\sqrt{\log_2 n}}$
- Identify the κ-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:
  - $B \leftarrow \mathtt{A}(f_0) \cup \mathtt{A}(f_1)$
  - $UMAST \leftarrow \max(UMAST, \mathtt{UMAST}(\mathcal{T}_0|B, \mathcal{T}_1|B))$
- For all pairs (l_0, l_1) of opposing core leaves:
  - Compute $\mathtt{C\text{-}RMAST}(\mathtt{R}(\mathcal{T}_0, l_0), \mathtt{R}(\mathcal{T}_1, l_1))$
- For all pairs (v_0, v_1) of opposing core vertices
  - $UMAST \leftarrow \max(UMAST, \mathtt{umast}(v_0, v_1))$
- Return $UMAST$.

Procedure 2 $\mathtt{umast}(v_0, v_1)$

- Build the matching graph $G_{v_0v_1}$: the independent sets are the neighbors of v_0 in $\mathcal{T}_0$ and the neighbors of v_1 in $\mathcal{T}_1$. For any such neighbors w_0 of v_0 and w_1 of v_1, the weight of (w_0, w_1) is $\mathtt{rmast}(w_0, w_1)$ for $\mathcal{T}_0$ and $\mathcal{T}_1$ rooted in v_0 and v_1.
- 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.
- Using the algorithm from [GT89], compute the maximum weight of a matching in the reduced matching graph, and return this value.

Procedure 3 $\mathtt{C\text{-}RMAST}(\mathcal{R}_0, \mathcal{R}_1)$

- For each node in each tree, select a heavy child with a maximum number of descendant leaves.
- Let $\mathcal{O}$ be the lexicographic ordering of $\mathtt{V}(\mathcal{R}_0) \times \mathtt{V}(\mathcal{R}_1)$ where the vertices in each tree are postordered.
- For each (v_0, v_1) in increasing order in $\mathcal{O}$ do
  - If either v_0 or v_1 is a leaf, return $|\mathtt{A}(\mathtt{T}(\mathcal{R}_0, v_0)) \cap \mathtt{A}(\mathtt{T}(R_1, v_1))|$
  - else set $\mathtt{rmast}(v_0, v_1) = \max(\mathtt{Diag}(v_0, v_1), \mathtt{rmatch}(v_0, v_1))$.
- Return $\{\mathtt{rmast}(v_0, v_1) \,|\, (v_0, v_1) \in \mathtt{V}(\mathcal{R}_0) \times \mathtt{V}(\mathcal{R}_1)\}$.

Procedure 4 $\mathtt{rmatch}(G)$*: computes the matching used locally in* $\mathtt{C\text{-}RMAST}$.

- Build the matching graph $G_{v_0v_1} = (\mathtt{C}(v_0) \cup \mathtt{C}(v_1), \mathtt{C}(v_0) \times \mathtt{C}(v_1), \mathtt{rmast}(\cdot))$.
- Remove all zero edges.
- For each heavy node, remove all but one of the heaviest of the matching edges not incident with light nodes adjacent to other light nodes.
- Using the algorithm from [GT89], compute the maximum weight of a matching in the reduced matching graph, and return this value.

Proof: Clearly the lemma would follow if there were only $O(\kappa)$ side trees, since each side tree contains at most n/κ 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 2κ side forests each of size at most n/κ and then recurse on the union of label sets of opposing pairs of these forests. ∎

6 Summing up

Theorem 6.1 *The* UMAST *problem can be solved in* $O(n^2\beta^{\sqrt{\log n}})$, *for some constant* β.

Proof: As in Lemma 5.3, let $U(n)$ be the time needed to compute the UMAST of two at most n leaf trees. Combining Lemmas 4.4 and 5.3, we have that there is a constant c such that

$$U(n) \leq c\kappa^2 n^2 + 4\kappa^2 U(\lfloor 2n/\kappa \rfloor) \quad \forall n \in \mathbb{N}_{>1}, \kappa \in \mathbb{R}_{>1}$$

In particular, the inequality holds fixing $\kappa = 4^{\sqrt{\log_2 n}}$ for all $n > 1$. Without loss of generality, we may assume that $U(1) \leq 16c$.

We claim that for all positive natural numbers, the function U is bounded from above by the function $f : \mathbb{R}_+ \to \mathbb{R}_+$ defined by $f(x) = 16cx^2\beta^{\sqrt{\log_2 x}} = O(x^2\beta^{\sqrt{\log_2 x}})$ where $\beta = 17$. The result follows be induction. For the base case, we have $f(1) = 16c17^{\sqrt{\log_2 1}} = 16c$. For the inductive step, where $n > 1$, we have

$$\begin{aligned} U(n) &\leq c\kappa^2 n^2 + 4\kappa^2 U(\lfloor 2n/\kappa \rfloor) \\ &\leq c\kappa^2 n^2 + 4\kappa^2 f(2n/\kappa) \\ &\leq c\kappa^2 n^2 + 4\kappa^2 16c(2n/\kappa)^2 \beta^{\sqrt{\log_2 2n/\kappa}}. \end{aligned}$$

Now,

$$\sqrt{\log_2(2n/\kappa)} = \sqrt{\log_2(2n/4^{\sqrt{\log_2 n}})} = \sqrt{\log_2 n + 1 - 2\sqrt{\log_2 n}} \leq \sqrt{\log_2 n} - 1,$$

so $\beta^{\sqrt{\log_2 2n/\kappa}} \leq \beta^{\sqrt{\log_2 n}}/\beta$. Thus, we have

$$\begin{aligned} U(n) &\leq c\kappa^2 n^2 + cn^2 16^2 \beta^{\sqrt{\log_2 n}}/\beta \\ &\leq cn^2(16^{\sqrt{\log_2 n}} + 16^2 17^{\sqrt{\log_2 n}}/17) \\ &\leq 16cn^2 17^{\sqrt{\log_2 n}} = f(n) \end{aligned}$$

For the last inequality we used that $n \geq 2$. ∎

6.1 Pseudo-Code

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 $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$

Lemma 5.1 *Let $\mathcal{T}_0$ and $\mathcal{T}_1$ be two unrooted evolutionary trees with core trees $\mathcal{C}_0$ and $\mathcal{C}_1$. Suppose for all $(c_0, c_1) \in \mathtt{V}(\mathcal{C}_0) \times \mathtt{V}(\mathcal{C}_1)$, that 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 $\mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \mathtt{UMAST}(\mathcal{T}_0|B, \mathcal{T}_1|B)$, where $B = \mathtt{A}(t_0) \cup \mathtt{A}(t_1)$.*

Proof: Let B be an agreement subset of maximum size, i.e. $|B| = \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$, and let Φ be the isomorphism from $\mathcal{T}_0|B$ to $\mathcal{T}_1|B$.

Fix $v, w \in \mathtt{V}(\mathcal{T}_0|B)$ such that $v \in \mathtt{V}(\mathcal{C}_0)$ and $\Phi(w) \in \mathtt{V}(\mathcal{C}_1)$. 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 $\Phi(w)$ cannot be found. By Lemma 3.1 and the hypothesis of this lemma, $\Phi(v) \notin \mathtt{V}(\mathcal{C}_1)$ and $w \notin \mathtt{V}(\mathcal{C}_0)$. 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 $\mathcal{T}_0|B$, and $\Phi(v_1)$ separates $\Phi(v_0)$ from the core vertices in $\mathcal{T}_1|B$. Hence $\mathtt{T}(\mathtt{R}(\mathcal{T}_0, v_0), v_1)$ and $\mathtt{T}(\mathtt{R}(\mathcal{T}_1, \Phi(v_1)), \Phi(v_0))$ are contained in side trees of $\mathcal{T}_0$ and $\mathcal{T}_1$. 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 $B_0 = \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_0|B, v_0), v_1))$ $[= \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_1|B, \Phi(v_0)), \Phi(v_1)))] \subseteq \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_0, v_0), v_1))$ and $B_1 = \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_1|B, \Phi(v_1)), \Phi(v_0)))$ $[= \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_0|B, v_1), v_0))] \subseteq \mathtt{A}(\mathtt{T}(\mathtt{R}(\mathcal{T}_1, \Phi(v_1)), \Phi(v_0)))$ ∎

Now define *side-side* to be the recursively computed maximum over all the side tree pairs, as described in Lemma 5.1. Then Lemma 5.1 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 5.2 *Given a partition $B_0, \ldots, B_m$ of the labels of an evolutionary tree T of size n, we can compute all of $T|B_0, \ldots, T|B_m$ in total $O(n)$ time.*

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

Assign to each internal node its depth in $O(n)$ time. Preprocess the tree in $O(n)$ time so that lca can be answered in constant time [HT84]. Now, for each partition $B = B_i$, 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 $l'_{2j-1} = l_j$ and let $l'_{2j} = \mathtt{lca}(l_j, l_{j+1})$. Now $l'_1, \ldots, l'_{2k-1}$ is the in-order traversal of $T|B$. From the l'_j ordering, together with the level information for internal nodes, construct $T|B$ in $O(k)$ time [Knu73], thus giving $O(n)$ time to build all trees.

Now assume that T is unrooted. Pick an arbitrary internal node v of T and let R be T rooted at v. Compute the partition $B_1, \ldots, B_m$ on R. But each $R|B_i$ is the same as $T|B_i$, with the exception of the root – which can be a degree two node. Finally, 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 5.3 *Let $U(n)$ be the time needed to compute the* UMAST *of two at most n leaf trees. Then side-side can be computed in time $O(n + (2\kappa)^2 U(\lfloor 2n/\kappa \rfloor))$.*

others will be side nodes. We will call an edge of $G_{c_0c_1}$ *side-side* if it is incident on two side nodes. Let $s_{c_0c_1}$ be the number of side-side edges in $G_{c_0c_1}$. As noted in the above proof, $\sum s_{c_0c_1} \leq n$.

Lemma 4.3 *For any core matching graph $G_{c_0c_1}$, there is an edge reduced matching graph $H_{c_0c_1}$ such that* $\texttt{match}(H_{c_0c_1}) = \texttt{match}(G_{c_0c_1})$ *and* $|\texttt{E}(H_{c_0c_1})| \leq 5s_{c_0c_1} + 12$.

Proof: Our argument is similar to that used for Lemma 2.2. First, $H_{c_0c_1}$ contains all side-side edges, and the edges between the four core nodes. This gives at most $s_{c_0c_1} + 4$ edges. The question is, which edges do we need to include between the four core nodes and the side nodes?

Let c be one of the core nodes. From c to the side nodes in the opposite independent set, include all of the at most $s_{c_0c_1}$ edges to side nodes incident on side-side edges. Finally, include two of the maximum weight remaining edges to side nodes. Let these edge be $\{c, c^s\}$ and $\{c, c^t\}$. Repeat the same procedure for the other core nodes. Thus, $H_{c_0c_1}$ contains a total of at most $s_{c_0c_1} + 4 + 4(s_{c_0c_1} + 2) = 5s_{c_0c_1} + 12$ edges, as required.

To show that $\texttt{match}(H_{c_0c_1}) = \texttt{match}(G_{c_0c_1})$, consider an arbitrary maximum matching M in $G_{c_0c_1}$. Assume that M has no zero-weight edges. Suppose that M contains an edge outside $H_{c_0c_1}$. Then this edge must be between a core node c and a side node u which is not incident on a side-side edge, and which is different from c^s and c^t. As a result at most one of c^s and c^t, say c^s, can be matched in M; namely by being adjacent to the other core node in the same independent set as c. Hence we get a new matching M' in $G_{c_0c_1}$ if we replace $\{c, u\}$ by the edge $\{c, c^t\}$ from $H_{c_0c_1}$. From our choice of c^t it follows that the weight of M' is at least that of M. We may therefore conclude that one of the maximum matchings in $G_{c_0c_1}$ is a maximum matching in $H_{c_0c_1}$. ∎

We conclude with the following:

Lemma 4.4 *core-core can be computed in $O(\kappa^2 n^2)$.*

Proof: By Observation 4.1, we can compute the edge weights on all matching graphs in $O(\kappa^2 n^2)$. By Lemma 4.2 and [GT89], matching on critical graphs takes $O((\kappa n)^{1.5} \log n)$ time. For core matching graphs, we have bounded the work of matching on $G_{c_0c_1}$ to $O(s_{c_0c_1}\sqrt{n}\log n + 1)$, by Lemma 4.3 and [GT89]. For those c_0, c_1 such that $s_{c_0c_1} = 0$, we get constant work, summing up to $O(n^2)$ for all such pairs. As noted above $\sum s_{c_0c_1} \leq n$. Thus all core matchings for which $s_{c_0c_1} > 0$ add up to $O(n^{1.5} \log n)$. ∎

5 Non-core Matchings

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. First, let v_0 and v_1 be interior vertices in $\mathcal{T}_0$ and $\mathcal{T}_1$. We say that v_0 *maps to* v_1 if $\texttt{umast}(v_0, v_1) = \texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$. Suppose we know that that no core node maps to another core node. 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 any relevant isomorphism to map all the core nodes of one tree to into a single side tree of the other tree. The following lemma confirms this intuition and allows us to compute non-core-core efficiently.

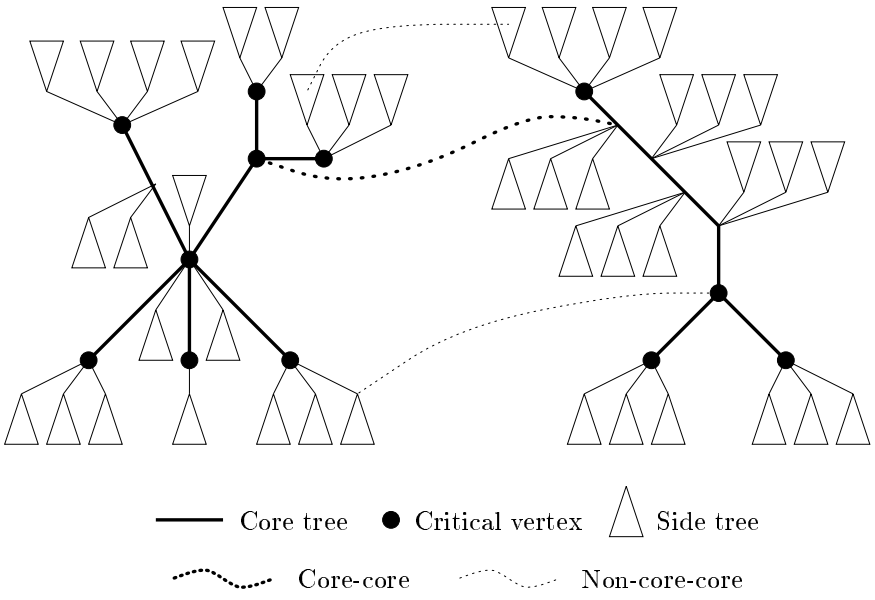

Figure 3.3: The core tree related concepts.

Proof: Simply pick $l_i \in \mathtt{L}(\mathcal{C}_i)$ such that c_i is between v_i and l_i (if $c_i \in \mathtt{L}(\mathcal{C}_i)$, set $l_i = c_i$). Clearly $\mathtt{rmast}(v_0, v_1) \in \mathtt{C\text{-}RMAST}(\mathtt{R}(T_0, l_0), \mathtt{R}(T_1, l_1))$. ∎

In other words, we can compute the `C-RMAST` of core-leaf rooted trees to get the edge weights on all core node pair matching graphs. Since each core tree has at most κ leaves, we can complete this computation in time $O(\kappa^2 n^2)$, using the $O(n^2)$ time `C-RMAST` algorithm of §2.

We now show how to compute all $\mathtt{match}(G_{c_0c_1})$ in time $O((\kappa n)^{1.5} \log n + n^2)$, thus concluding that core-core can be computed in $O(\kappa^2 n^2)$ total time. As in §2, the idea is to reduce the number of relevant matching edges, and again, a first trivial step is, in time $O(n^2)$, to remove all matching edges of weight zero.

Recall that core nodes are critical if and only if they have degree different from two. Call $G_{c_0c_1}$ a *critical matching graph* if c_0 or c_1 is critical. Otherwise call it a *core matching graph.*

Lemma 4.2 *The critical matching graphs have a total of $O(\kappa n)$ non-zero matching edges.*

Proof: First, note that since each core tree has at most κ leaves, in total the critical nodes have only $O(\kappa)$ core neighbors. This is counting with multiplicity if the same core node is neighbor of more than one critical node. On the other hand, no node in a matching graph can be incident to more than n matching edges, so in total, in the critical matching graphs, there are at most $O(\kappa n)$ matching edges incident on a core node. All other edges are between pairs of side nodes. However, each label only appears in a single side tree per tree, so in total there are only $O(n)$ non-zero matching edges between pairs of side nodes. ∎

Now consider core matching graphs. A non-critical core node has exactly two core neighbors. So, for core matching graph $G_{c_0c_1}$, we will have two nodes distinguished as core on each size. All

Let $w_0, \ldots, w_k$ be the neighbors of v in $\mathcal{T}_0|B$. Set $B_i = \mathtt{A}(\mathtt{T}(\mathtt{R}((\mathcal{T}_0|B), v), w_i))$ [$= \mathtt{A}(\mathtt{T}(\mathtt{R}((\mathcal{T}_1|B), \Phi(v)), \Phi(w_i)))$]. Then the B_is are disjoint and $B = \bigcup_i B_i$. For $i = 0, \ldots, k$, let v_i be the neighbor of v on the way to w_i in $\mathcal{T}_0$, and let $\Phi'(v_i)$ denote the neighbor of $\Phi(v)$ on the way to $\Phi(w_i)$ in $\mathcal{T}_1$. Consider some specific i. Now B_i is a rooted agreement subset for $\mathtt{T}(\mathtt{R}(\mathcal{T}_0, v), v_i)$ and $\mathtt{T}(\mathtt{R}(\mathcal{T}_1, \Phi(v)), \Phi'(v_i))$ since

$$\mathtt{T}(\mathtt{R}(\mathcal{T}_0, v), v_i)|B_i = \mathtt{T}(\mathtt{R}(\mathcal{T}_0|B_i, v), w_i) \cong \mathtt{T}(\mathtt{R}(\mathcal{T}_1|B_i, \Phi(v)), \Phi(w_i)) = \mathtt{T}(\mathtt{R}(\mathcal{T}_0, \Phi(v)), \Phi'(v_i))|B_i.$$

Hence, for $\mathtt{R}(\mathcal{T}_0, v), \mathtt{R}(\mathcal{T}_1, \Phi(v))$, we have $\mathtt{rmast}(v_i, \Phi'(v_i)) \geq |B_i|$, and hence $\mathtt{umast}(v, \Phi(v)) \geq |B|$. ∎

Lemma 3.1 immediately suggests that there are two problems to be solved in computing the UMAST:

- *How do we compute the weighted matchings of the G_{vw}?* This can be done naïvely in $O(n^2 n^{2.5} \log n)$.
- *How do we compute the edge weights in each $G_{v_0 v_1}$?* In §2, we showed an $O(n^2)$ algorithm for computing RMAST. Rooting the trees at each v_0 and v_1 and computing their C-RMAST would give all needed values in $O(n^4)$ time.

We speed up the computation for both subtasks by introducing the concept of a *κ-core tree*. Let T be an n-leaf tree and κ 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/κ 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*. The definitions are illustrated in Figure 3.3. 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 κ leaves, for if we remove all core edges, each core leaf will be in a component with at least n/κ 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 $\mathtt{umast}(c_0, c_1) = \mathtt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$, or no such pair exists. Accordingly, we define *core-core* to be the maximum umast value for core node pairs, and let *non-core-core* denote the maximum over all other pairs. Clearly, the UMAST of two trees is the maximum over the core-core and non-core-core values.

In §4, we show how to compute core-core efficiently. In §5, we show how to compute non-core-core efficiently, and give an overall analysis of the running time for our UMAST algorithm in §6.

4 Core Matchings

By Lemma 3.1, we must compute $G_{c_0 c_1}$ for all $(c_0, c_1) \in \mathtt{V}(\mathcal{C}_0) \times \mathtt{V}(\mathcal{C}_1)$.

Observation 4.1 *Let $(c_0, c_1) \in \mathtt{V}(\mathcal{C}_0) \times \mathtt{V}(\mathcal{C}_1)$. Let v_0 and v_1 be arbitrary neighbors of c_0 and c_1. Then, for $\mathtt{R}(\mathcal{T}_0, c_0), \mathtt{R}(\mathcal{T}_1, c_1)$,*

$$\mathtt{rmast}(v_0, v_1) \in \bigcup \{\mathtt{C\text{-}RMAST}(\mathtt{R}(T_0, l_0), \mathtt{R}(T_1, l_1)) | (l_0, l_1) \in \mathtt{L}(\mathcal{C}_0) \times \mathtt{L}(\mathcal{C}_1)\}.$$

Proof: We have bounded the work of matching on $G_{v_0v_1}$ to $O(l_{v_0v_1}\sqrt{n}\log n + 1)$, by Lemma 2.2 and [GT89]. For those v_0v_1 such that $l_{v_0v_1} = 0$, we get constant work, summing up to $O(n^2)$ for all such pairs. We now must bound $\sum l_{v_0v_1}$. But 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. Thus all matching for which $l_{v_0v_1} > 0$ add up to $O(n^{1.5}\text{polylog}n)$. ∎

Notice that we compute the `RMAST` for all pairs of subtrees. In fact, this will become crucial when we use `RMAST` to compute `UMAST`. To emphasize this fact, we let $\texttt{C-RMAST}(R_0, R_1) = \{\texttt{RMAST}(\texttt{T}(R_0, v_0), \texttt{T}(R_1, v_1)) | v_j \in \texttt{V}(R_j), j \in \{0, 1\}\}$ and note as a corollary to Theorem 2.3 that `C-RMAST` can also be computed in $O(n^2)$ time.

3 Unrooted Trees

Fix $\mathcal{T}_0$ and $\mathcal{T}_1$ as the two n leaf unrooted evolutionary trees for which we want to compute `UMAST`. Clearly $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = n$ 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 will extend some notation used in rooted trees to the unrooted case. For an unrooted evolutionary tree T, we take $\texttt{A}(T)$ to be its label set. Also, for $v \in \texttt{V}(T)$, we take $\texttt{d}(v)$ to be the degree of v in the unrooted sense. In addition, we introduce the following new notation. We take $\texttt{I}(T)$ to be Ts interior vertices and $\texttt{L}(T)$ to be its leaves, so $\texttt{I}(T) \cup \texttt{L}(T) = \texttt{V}(T)$. For any $v \in \texttt{I}(T)$, we let $\texttt{R}(T, v)$ denote the rooted evolutionary tree obtained from v by rooting T at v. Notice that if u, v, w are vertices occurring in this order on a path in T, then $\texttt{T}(\texttt{R}(T, u), w) = \texttt{T}(\texttt{R}(T, v), w)$

Recall from the definition that $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1)$ is the size of the maximum sized set B such that $\mathcal{T}_0|B$ and $\mathcal{T}_1|B$ are isomorphic. If $\mathcal{T}_0|B$ and $\mathcal{T}_1|B$ are isomorphic, then we will write $T_0|B \cong T_1|B$. Further the isomorphism $\Phi : \texttt{V}(\mathcal{T}_0|B) \rightarrow \texttt{V}(\mathcal{T}_1|B)$ is unique and may be defined as follows. If $v \in \texttt{L}(\mathcal{T}_0|B)$, then $\texttt{A}(v) = \texttt{A}(\Phi(v))$. If $v \in \texttt{I}(\mathcal{T}_0|B)$, then there are $l_0, l_1, l_2 \in \texttt{L}(\mathcal{T}_0|B)$ such that $v = \texttt{meet}(l_0, l_1, l_2)$. Then $\Phi(v) = \texttt{meet}(\Phi(l_0), \Phi(l_1), \Phi(l_2))$.

The main structural lemma that relates rooted and unrooted `MAST` computations is similar in flavor to Lemma 2.1 and depends, once again, on matching graphs. Recall that $G_{v_0v_1}$ was defined to be a bipartite graph over the children on v_0 and v_1 weighted by the `rmast` of the subtrees defined by the children. In an unrooted tree, nodes do not have children. For unrooted trees $\mathcal{T}_0$ and $\mathcal{T}_1$, we define $G_{v_0v_1}$ by first rooting $\mathcal{T}_i$ at v_i, and then defining $G_{v_0v_1}$ as before. Finally we define $\texttt{umast}(v_0, v_1) = \texttt{match}(G_{v_0v_1})$. The lemma below reduces `UMAST` to `rmast`, the `rmast` coming in as the weights of the edges in $G_{v_0v_1}$, hence forming the basis for computing $\texttt{umast}(v_0, v_1)$. Steel and Warnow [SW93] based their algorithm on a similar but different reduction.

Lemma 3.1 *Let B be any maximum cardinality agreement set for $\mathcal{T}_0$ and $\mathcal{T}_1$, and let Φ be the isomorphism from $\mathcal{T}_0|B$ to $\mathcal{T}_1|B$. Then for any $v \in \texttt{I}(\mathcal{T}_0|B)$, we have $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \texttt{umast}(v, \Phi(v))$. As a consequence,*

$$\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = \max\{\texttt{umast}(v_0, v_1) | (v_0, v_1) \in \texttt{I}(\mathcal{T}_0) \times \texttt{I}(\mathcal{T}_1)\}.$$

Proof: Fix $v \in \texttt{I}(\mathcal{T}_0|B)$. Trivially $\texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) \geq \texttt{umast}(v, \Phi(v))$, so the result follows if we can show $\texttt{umast}(v, \Phi(v)) \geq \texttt{UMAST}(\mathcal{T}_0, \mathcal{T}_1) = |B|$.

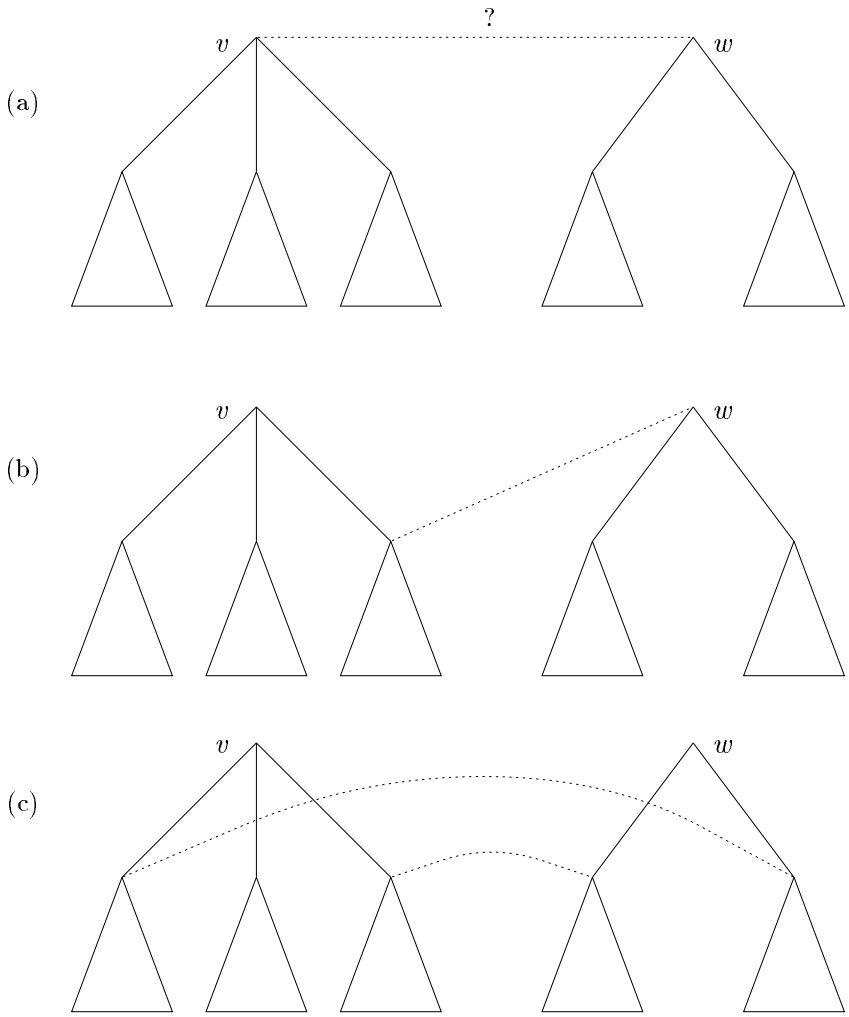

Figure 2.2: In order to find `rmast`(v, w) as in (a), we can either try a "diagonal" as in (b), or we can find a maximum weight matching between the sets of children as in (c).

Lemma 2.1 ([SW93]) $\forall v_0 \in \mathtt{V}(\mathcal{R}_0), v_1 \in \mathtt{V}(\mathcal{R}_1)$,

$$\mathtt{rmast}(v_0, v_1) = \begin{cases} |\mathtt{A}(\mathtt{T}(\mathcal{R}_0, v_0)) \cap \mathtt{A}(\mathtt{T}(\mathcal{R}_1, v_1))| & \textit{if } v_0 \textit{ or } v_1 \textit{ is a leaf} \\ \max \left\{ \begin{array}{l} \{\mathtt{rmast}(v_0, w_1) | w_1 \in \mathtt{C}(v_1)\} \\ \{\mathtt{rmast}(w_0, v_1) | w_0 \in \mathtt{C}(v_0)\} \\ \mathtt{match}(G_{v_0 v_1}) \end{array} \right\} & \textit{otherwise.} \end{cases}$$

The lemma is illustrated at Figure 2.2.

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 one of the descendants of the root 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 the weighted matchings. Using the algorithm in [GT89], we can compute $\mathtt{match}(G_{v_0 v_1})$ in time $O(\mathtt{d}(v_0)\mathtt{d}(v_1)\sqrt{\mathtt{d}(v_0) + \mathtt{d}(v_1)} \log n)$. Summing over all v_0 and v_1 gives a time bound of $O(n^{2.5} \log n)$. In this analysis, we count each of the $\mathtt{d}(v_0) \cdot \mathtt{d}(v_1)$ edges in $G_{v_0 v_1}$ as relevant to the bipartite matching. In the remainder of this section, we show that this is not so, and thereby show that we can reduce the time spent in the matchings. A first trivial step is, in time $O(n^2)$, to remove all matching edges of zero weight. This alone, however, does not give the desired reduction.

The following definitions will be the key to our reduction of the matching graphs. For each internal node v of each tree $\mathcal{R}_j$, we select a *heavy child* to be one of $\mathtt{C}(v)$ with a maximum number of descendant leaves. All other children of v are said to be *light*. In $G_{v_0 v_1}$, we will have one node distinguished as heavy on each side. All others will be light. We will call an edge of $G_{v_0 v_1}$ *light-light* if it is incident on two light nodes. Let $l_{v_0 v_1}$ be the number of light-light edges in $G_{v_0 v_1}$.

Lemma 2.2 $\forall v_0 \in \mathtt{V}(\mathcal{R}_0), v_1 \in \mathtt{V}(\mathcal{R}_1)$, *there is an edge reduced matching graph* $H_{v_0 v_1}$ *such that* $\mathtt{match}(H_{v_0 v_1}) = \mathtt{match}(G_{v_0 v_1})$ *and* $|\mathtt{E}(H_{v_0 v_1})| \leq 3 l_{v_0 v_1} + 3$.

Proof: First, $H_{v_0 v_1}$ contains all the light-light edges, and the edge between the two heavy nodes. This gives at most $l_{v_0 v_1} + 1$ edges. The question is, which edges do we need to include between the two heavy nodes and the light nodes?

Let h be one of the heavy nodes. From h to the light nodes in the opposite independent set, include all of the at most $l_{v_0 v_1}$ edges to light nodes incident on light-light edges. Finally, include one of the maximum weight remaining edges to light nodes. Let this edge be $\{h, h^l\}$. Repeat the same procedure for the other heavy node. Thus, $H_{v_0 v_1}$ contains a total of at most $l_{v_0 v_1} + 1 + 2(l_{v_0 v_1} + 1) = 3 l_{v_0 v_1} + 3$ edges, as required.

To show that $\mathtt{match}(H_{v_0 v_1}) = \mathtt{match}(G_{v_0 v_1})$, consider an arbitrary maximum matching M in $G_{v_0 v_1}$. Assume that M has no zero-weight edges. Suppose that M contains an edge outside $H_{v_0 v_1}$. Then this edge must be between a heavy node h and a light node u which is not incident on a light-light edge, and which is different from h^l. Then h^l cannot be matched in M, so we get a new matching M' in $G_{v_0 v_1}$ if we replace $\{h, u\}$ by the edge $\{h, h^l\}$ from $H_{v_0 v_1}$. From our choice of h^l it follows that the weight of M' is at least that of M. We may therefore conclude that one of the maximum matchings in $G_{v_0 v_1}$ is a maximum matching in $H_{v_0 v_1}$. ∎

We conclude with the following:

Theorem 2.3 `RMAST` *can be computed in* $O(n^2)$.

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 [GT89], 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 $O(n^2)$ algorithm for the RMAST problem. This will be used as a subroutine for the UMAST algorithm, an overview of which is given in §3. In §4 and §5, we give descriptions of the two main subroutines of our UMAST algorithm. We conclude in §6 with a complete (and concise) description of the rooted and unrooted MAST algorithms.

2 Rooted Trees

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

For any graph G, we take $\mathtt{V}(G)$ to be its vertex set and $\mathtt{E}(G)$ to be its edge set. For any evolutionary tree R, we take $\mathtt{A}(R)$ to be its label set. For $v \in \mathtt{V}(R)$, we take $\mathtt{d}(v)$ to be the degree of v in R, and we take $\mathtt{C}(v)$ to be the set of children of v. By $\mathtt{T}(R, v)$ we denote the sub-tree of R rooted at v. For any weighted bipartite graph G, let $\mathtt{match}(G)$ denote the value of a maximum weighted matching on G.

Let $(v_0, v_1) \in \mathtt{V}(\mathcal{R}_0) \times \mathtt{V}(\mathcal{R}_1)$ and for $i = 0, 1$, set $t_i = \mathtt{T}(\mathcal{R}_i, v_i)$. Then we take $\mathtt{rmast}(v_0, v_1)$ as the RMAST of $t_0|B$ and $t_1|B$ where $B = \mathtt{A}(t_0) \cap \mathtt{A}(t_1)$. Thus, in particular $\mathtt{RMAST}(\mathcal{R}_0, \mathcal{R}_1) = \mathtt{rmast}(r_0, r_1)$ where r_i is the root of $\mathcal{R}_i$. Set $G_{v_0 v_1} = (\mathtt{C}(v_0) \cup \mathtt{C}(v_1), \mathtt{C}(v_0) \times \mathtt{C}(v_1), \mathtt{rmast}(\cdot, \cdot))$. The following lemma appears in Steel and Warnow [SW93] and is the basis for their dynamic programming approach to this problem.

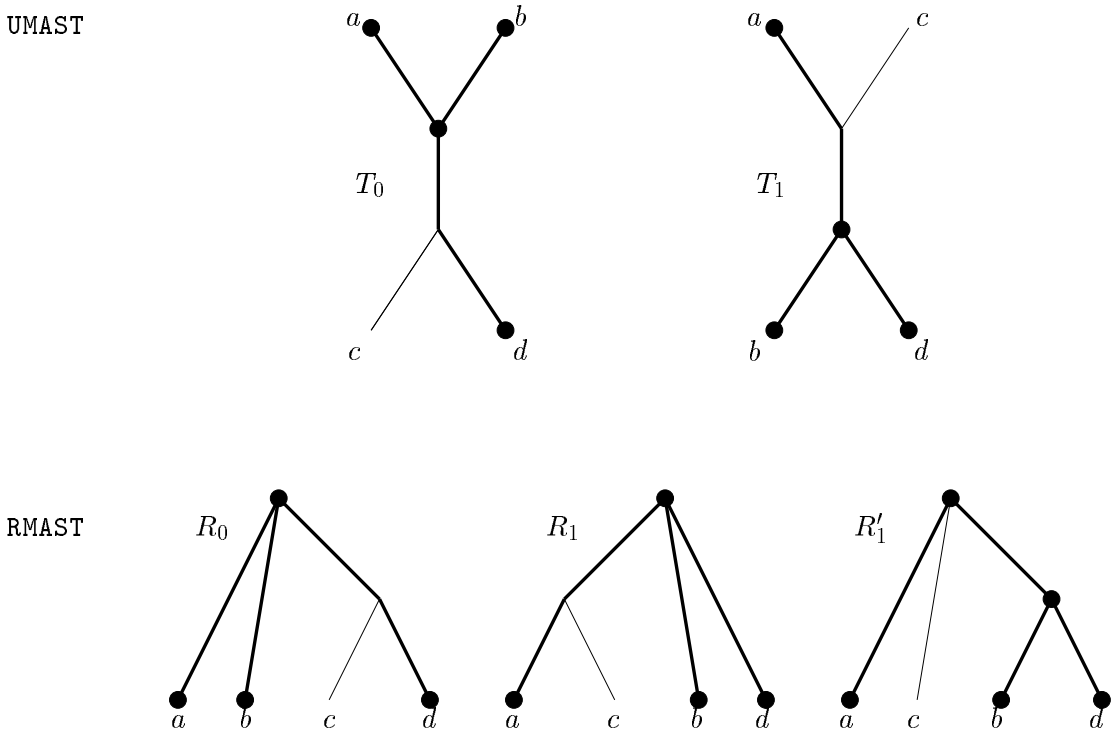

Figure 1.1: The trees T_0 and T_1 are non-isomorphic, but $T_0|\{a,b,d\}$ and $T_1|\{a,b,d\}$ are isomorphic, so $\{a,b,d\}$ is a solution to $\texttt{UMAST}(T_0,T_1)$. Similarly, $\{a,b,d\}$ is a solution to $\texttt{RMAST}(R_0,R_1)$. Note that R_0 and R_1 are rooted variants of T_0 and T_1. Also R'_1 is a rooted variant of T_1, but $\{a,b,d\}$ is not a solution to $\texttt{RMAST}(R_0,R'_1)$.

these leaves. The edges in $T|B$ are obtained as replacements of the paths in T between the vertices selected for $T|B$.

The restriction operator immediately implies a similarity measure on trees. Finden and Gordon [FG85] introduced just such a measure, formalizing the intersection 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.

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 $R_0|B$ and $R_1|B$ are isomorphic.

For both variants, isomorphisms are understood to map leaves with the same labels to each other. The definitions are illustrated in Figure 1.1. Generally, by an *agreement set*, we will refer to any subset of the speices giving rise to an isomophism as in the above definitions. Using this term, both rooted and unrooted agreement subtree problems are questions of finding maximum cardinality agreement sets between pairs of evolutionary trees.

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 $|A| + 1$ 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 maximum cardinality of an agreement 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 [FG85] gave a heuristic method for computing the maximum agreement subtree of two rooted binary trees. Their algorithm, which has an $O(n^5)$ running time, does not, however, guarantee an optimal solution. In [KKM94], Kubicka et al. presented an $O(n^{(\frac{1}{2}+\epsilon)\log_2 n})$ time algorithm for the `UMAST` problem in which each node has degree 3. Steel and Warnow [SW93] also considered the `UMAST` problem, giving the first polynomial time algorithm for the `UMAST` problem. 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. Since this paper was presented several more results on computing agreement trees have followed [AK94, FT94b].

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 upper bounds known for the bounded degree and unbounded degree versions of these problems.

1 Introduction

An evolutionary tree, or phylogeny, is a model of the evolutionary history for a set of species. Constructing such trees from observations on a set of living species is one of the fundamental tasks of computational biology. This is because the evolutionary relation of species provides a great deal of information about their biochemical machinery. For example, RNA's secondary structure is most accurately determined by selecting correlated mutations of a class of related species.

To construct a tree from a set of species, one must have a model of what makes one tree better than another. Many criteria have been proposed, but in general, these turn out to be *NP*-hard to optimize [BFW92, Day87]. There is also no consensus in the biology community as to what makes a good tree. 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. [Far72, Fel82, FM76, SN87, SS63]). More recently, a variety of solutions have been examined rigorously ([AFB94, FKW93, KLW90]). Not surprisingly these various methods do not 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 may agree in some parts and differ in others. In general, one is interested in finding the largest set of species on which the trees agree, and this is exactly the problem addressed in this paper.

More concretely, let A be a set of species. Then we will define an *evolutionary tree*, T, on A to be an unrooted tree, with no degree 2 internal nodes, such that the leaves of T are uniquely labeled with the elements of A. The fact that an evolutionary tree is supposed to model a historical process implies that the edges of the tree can be naturally directed, thus implying that evolutinary trees are naturally rooted. Nevertheless unrooted tree come up because some methods of tree construction are not well-geared towards producing rooted trees. For example, we may want to separate all vertebrates from invertebrates via some edge in the tree, but we may not know for all such separations which time-determined orientation it should take. We will consider both rooted and unrooted trees in this paper. Suppose now that we are given two rooted or unrooted evolutionary trees T_0 and T_1 on the same species set A. If the two trees differ, it is reasonable to ask for the "intersection" of the information contained in the trees. By viewing the input trees as the outcomes of experiments performed to discover the history of some species, we will typically have more confidence in information given in the "intersection" than in information unique to each tree. But what is the intersection of two evolutionary trees?

One of the most intensively studied answers to this question involves the notion of a *restriction* of an evolutionary tree to a subset of the species. Given a rooted tree R on set A, and given $B \subseteq A$, then the *restriction of T to B*, written $T|B$, is the evolutionary tree on B such that B has the same evolutionary relationship in $T|B$ as it does in T. More formally, if T is rooted, $T|B$ denotes the rooted evolutionary trees whose vertices are the closure of the leaves with labels in B under the least common ancestor operation (`lca`). The arcs in $T|B$ are obtained as replacements of the dipaths in T connecting the vertices of $T|B$. Seen in another way, $T|B$ is the evolutionary tree derived from T by deleting leaves labeled from $A \setminus B$, along with unneeded internal nodes. In unrooted trees, the notion of an `lca` is not defined. Instead, we define $\texttt{meet}(u, v, w)$ to be the vertex shared by the simple paths from u to v, u to w, and v to w. Then by $T|B$, we denote the unrooted tree whose vertices are either leaves of T with labels from B, or the `meet` of triples of

Fast Comparison of Evolutionary Trees*

Martin Farach†
Rutgers University

Mikkel Thorup‡
University of Copenhagen

January 13, 1995

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. We also derive an $O(n^2)$ time algorithm for the `RMAST` problem. The previous best algorithm for both these problems has running time $O(n^{4.5+o(1)})$.

*A preliminary version was presented at the 5th Annual ACM-SIAM Symposium on Discrete Algorithms [FT94a].

†`farach@dimacs.rutgers.edu`; Supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center under NSF contract STC-8809648.

‡`mthorup@diku.dk`; Most of the research in this paper was done while the second author was visiting DIMACS. Supported by the Danish Technical Research Council, by the Danish Research Academy and by DIMACS under NSF contract STC-8809648.