we apply the reduction to their two core children, the total sum of their matching weights becomes $O(n)$, and if for each comparison of a spine node and a critical node we apply the reduction to the core child of the spine node, the total sum of their matching weights becomes $O(\kappa n)$. With regards to the $O(\kappa^2)$ comparisons of two critical nodes, their sum cannot exceed $O(\kappa^2 n)$ in total weight. Thus, since we have a total of $O(\kappa n)$ edges involved in the matchings, in time $O(\kappa n)$, we can reduce the total sum of the matching weights to $O(\kappa^2 n)$. ■

Theorem 6.6 *Let $M : \mathbb{R}_{\geq 1} \to \mathbb{R}$ be a monotone function bounding the time complexity* UWBM. *Moreover, let M satisfy that $M(x) = x^{1+\varepsilon} f(x)$, where $\varepsilon \geq 0$ is a constant, $f(x) = O(x^{o(1)})$, f is monotone, and for some constants b_1, b_2, $\forall x, y \geq b_1 : f(xy) \leq b_2 f(x) f(y)$. Then, with $\kappa = \sqrt[\varepsilon]{4}$,* MAST *is computable in time $O(n^{1+o(1)} + M(n))$.*

Proof: We spend $O(n \text{ polylog } n + \texttt{time}(\texttt{UWBM}(k^2 n)))$ on the matchings. So, by Theorem 5.10, we have that Comp-Core-Trees can be computed in time $O(n \text{ polylog} n + \texttt{time}(\texttt{UWBM}(k^2 n)))$. Applying Theorem 4.5 gives the desired complexity. ■

Inserting the best known bounds for weighted bipartite matching [11], with $\kappa = \sqrt[1/2]{4} = 16$, we get

Corollary 6.7 MAST *is computable in time $O(n^{1.5} \log n)$.* ■

References

[1] 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 26th Ann. ACM Symp. on Theory of Computing*, 1994.

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

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

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

[5] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 1994. In press. See also STOC '93.

[6] M. Farach and M. Thorup. Fast comparison of evolutionary trees. *Proc. of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1994.

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

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

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

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

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

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

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

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

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

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

[17] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 1994. To appear.

a non-zero edge between opposing side children. For matching graph G_{vw}, we will let l_{vw} be the number of side-side edges in the graph. By Fact 3.2, there can be no more than n side-side edges, and the sum of their weights is also bounded by n.

Recall that the matchings come in two varieties: E_1 contains all the pairs of core vertices where one is critical; E_2 contains the at most n interesting pairs of spine vertices. We will bound the size of the E_1 and E_2 matchings seperately.

Lemma 6.1 *There are edge reduced matching graphs H_{vw} such that $\texttt{MWM}(H_{vw}) = \texttt{MWM}(G_{vw})$ for all (v,w) in E_1, and such that $\sum_{(v,w)\in E_1} |\texttt{E}(H_{vw})| = O(\kappa n)$.*

Proof: As note above, there are at most n side-side edges. All other edges are incident on a core child of a critical node, of which there are $O(\kappa)$, thus bounding the edges in the H_{vw} by $O(\kappa n)$. ■

Lemma 6.2 *For any $(v,w) \in E_2$ there is an edge reduced matching graph H_{vw} such that $\texttt{MWM}(H_{vw}) = \texttt{MWM}(G_{vw})$ and $|\texttt{E}(H_{vw})| \leq 3l_{vw} + 3$, and $\sum_{(v,w)\in E_2} |\texttt{E}(H_{vw})| = O(n)$.*

Proof: First of all, our reduced matching graph H_{vw} contains all side-side edges, and the edge between the two opposing core nodes. This gives at most $l_{vw} + 1$ edges. The question is which edges we need to include between the two core nodes and their opposing side nodes.

Let c be one of the core nodes. From c to the opposing side nodes, we will include all of the at most l_{vw} edges to side nodes incident with side-side edges. Moreover, we will include one of the maximum weight remaining edges to side nodes. Let this edge be $\{c, c^s\}$. Thus, H_{vw} contains a total of at most $l_{vw} + 1 + 2(l_{vw} + 1) = 3l_{vw} + 3$ edges, as required.

We need to prove that that $\texttt{MWM}(H_{vw}) = \texttt{MWM}(G_{vw})$. Consider an arbitrary maximal matching M in G_{vw}. We assume that M has no zero-weight edges. Suppose that M contains an edge outside H_{vw}. Then this edge must be between a core node c and an opposing side node u which is not incident to a side-side edge, and which is different from c^s. Then c^s cannot be matched in M, so we get a new matching M' in G_{vw} if we replace $\{c, u\}$ by the edge $\{c, c^s\}$ from H_{vw}. Moreover, from our choice of c^s it follows that the weight of M' is at least that of M. We may therefore conclude that one of the maximal matchings in G_{vw} is a maximal matching in H_{vw}.

Finally, we must bound the summation $\sum_{(v,w)\in E_2} 3l_{vw} + 3$. But $|E_2| \leq n$ and $\sum l_{vw} \leq n$, by Fact 3.2. ■

Lemma 6.3 *We can compute all the H_{vw} in $O(n \log^3 n)$ time.*

Proof: First choose an arbitrary ordering of the side children of each vertex. It is now meaningful to talk about intervals of side-children. With the hierarchy technique that was used in the proof of Lemma 5.2, we can make an $O(n \log^2 n)$ preprocessing such that given any vertex v and interval I of side children of some opposing vertex, we can compute $\max\{\texttt{mast}(v,w) | w \in \texttt{V}(I)\}$ in time $O(\log^2 n)$.

Recall our problem: we are given a vertex v together with a vertex w and a subset S of the side children which already participate in the matching since they share labels with the side trees of w. Let w^c be the core children of w. We want to find the side child v^s of v outside S which has the MAST with w^c.

This is done in time $O(|S| \log^2 n)$, for removing the vertices from S leaves us with $\leq |S| + 1$ intervals, each of which we can deal with in time $O(\log^2 n)$, and afterwards we just need to find the maximum which is done in time $O(|S|)$. ■

Having reduced the number of edges in our matching to $O(\kappa n)$, we can trivially conclude the total weight is (κn^2). Our aim is to reduce the weights to a total of $O(\kappa^2 n)$ before we apply the matching algorithm, and then increment the weights afterwards. For this we will use the following technical observation:

Observation 6.4 *Let $G = (V_0 \cup V_1, E, W)$ be a weighted bipartite graph, and let $v \in V_0$. For all edges (v,w), set $\Delta(v,w) = W(v,w) - \max(\{0\} \cup \{W(v',w) | v' \in V_0 \setminus \{v\})$. Moreover, set $\Delta(v) = \max\{\Delta(v,w) | (v,w) \in E\}$. Assume that $\Delta(v) > 1$, and let G' be the weighted bipartite graph obtained by reducing the weights of all edges incident with v by $\Delta(v) - 1$. Then $\Delta'(v) = 1$, and then the maximal weight matchings in G' are the same as those in G and there weights are exactly $\Delta(v) - 1$ smaller.*

Proof: The result follows directly from the fact that v has to be in any maximal matching if $\Delta(v) \geq 1$. ■

Clearly, we can find $\Delta(v)$ in time linear in the number of edges. Thus, for each of our matchings we may choose a constant number of vertices v, that we *reduce*, getting $\Delta'(v) \leq 1$, before we apply a matching procedure.

Proposition 6.5 *The total weight of matching edges can be reduced to $O(k^2 n)$ in time $O(\kappa n)$.*

Proof: The total weight of the side-side edges is at most n, so, if for each comparison of two spine nodes

Proof: Let (I_0, I_1) be any pair in $\mathcal{P}$. By symmetry it is sufficient to show the computation of $\mathtt{mast}(\mathtt{c}(I_0), \mathtt{r}(I_1))$. Suppose there is a vertex $v_0 \in \mathtt{V}(S_0)$ below, or equal to, $\mathtt{c}(I_0)$ which is interesting with respect to some vertex in I_0. Fix v_0' to be the highest such vertex, i.e. the one closest to $\mathtt{c}(I_0)$. Let I_0' be the interval in $\mathcal{I}_0$ which contains v_0 and which is on the same level as I_0, and hence as I_1 in $\mathcal{I}_1$. Thus I_0' is interesting with respect to I_1, and since they are on the same level in their respective interval trees, we can conclude that $(I_0', I_1) \in \mathcal{P}$. Trivially $(I_0', I_1) < (I_0, I_1)$, so we can assume that the ceiling $\mathtt{mast}(\mathtt{r}(I_0'), \mathtt{r}(I_1))$ is computed.

Set $I_0'' =]\mathtt{r}(I_0'), \mathtt{c}(I_0)]$. By choice of v_0', the pair (I_0'', I_1) is boring—but typically not in $\mathcal{P}$. The diagonal $\mathtt{mast}(\mathtt{c}(I_0''), \mathtt{r}(I_1))$ is exactly the ceiling of (I_0', I_1) which we saw was computed, and the diagonal $\mathtt{mast}(\mathtt{r}(I_0'), \mathtt{c}(I_1))$ is the floor of (I_0, I_1) which is computed by Lemma 5.5. Thus by Lemma 5.1 and Lemma 5.2, we can compute the ceiling $\mathtt{mast}(\mathtt{r}(I_0''), \mathtt{r}(I_1))$ in time $O(\log^2 n)$. But this ceiling is exactly the desired diagonal of (I_0, I_1). ■

Corollary 5.7 *Any boring pair in $\mathcal{P}$ can be computed in time $O(\log^2 n)$.*

Proof: By Proposition 5.6 we can compute the diagonals in time $O(\log^2 n)$. But then we can apply Lemma 5.1 and Lemma 5.2 to get the ceiling in time $O(\log^2 n)$. ■

Corollary 5.8 *Any interesting leaf pair (I_0, I_1) in $\mathcal{P}$ can be computed in time $O(\log^2 n + \mathtt{time}(\textsc{Comp}(\mathtt{r}(I_0), \mathtt{r}(I_1))))$.*

Proof: Again, the diagonals are computed by Proposition 5.6, and the floor follows by Lemma 5.5. Hence we have the whole basis of $(\mathtt{V}(I_0), \mathtt{V}(I_1))$ available. Since (I_0, I_1) is an interesting leaf pair, it follows that $\mathtt{V}(I_i) = \{\mathtt{r}(I_i)\}$ for $i = 0, 1$. Thus, in fact, we have the basis of $(\mathtt{r}(I_0), \mathtt{r}(I_1))$ available, so the ceiling can be computed directly by Comp. ■

Summing up, we conclude:

Proposition 5.9 Comp-Spines(S_0, S_1) *(Procedure 3) can be computed in time $O((m_0 + m_1 + l \log n) \log^2 n + (\sum_{(v_0, v_1) \in W} \mathtt{time}(\textsc{Comp}(v_0, v_1))))$, where $m_0 = |\mathtt{V}(S_0)|$, $m_1 = |\mathtt{V}(S_1)|$ and $l = |\mathtt{A}(\mathtt{SideT}(S_0)) \cap \mathtt{A}(\mathtt{SideT}(S_1))|$, and where $W \subseteq \mathtt{V}(S_0) \times \mathtt{V}(S_1)$ contains at most l pairs.* ■

Theorem 5.10 Comp-Core-Trees *can be computed in time $O(\kappa n \log^3 n + \sum_{(v_0, v_1) \in E} \mathtt{time}(\textsc{Comp}(v_0, v_1)))$. Here E divides into two sets E_1 and E_2. The set E_1 contains all the pairs of core vertices where one is critical. The set E_2 contains the at most n interesting pairs of spine vertices.*

Proof: Let S denote the set of pairs of spines from opposing core trees. Since there can be at most κ spines in each core tree, we get $(\sum_{(S_0, S_1) \in S} |\mathtt{V}(S_0)| + |\mathtt{V}(S_1)|) \leq \kappa n$. Moreover, by Fact 3.2, we get that $(\sum_{(S_0, S_1) \in S} |\mathtt{A}(\mathtt{SideT}(S_0)) \cap \mathtt{A}(\mathtt{SideT}(S_1))|) \leq n$. Thus, the result follows directly from Observation 3.3 together with Proposition 5.9. ■

Corollary 5.11 *For bounded degree trees, the* MAST *problem is computable in time $O(nc^{\sqrt{\log n}})$.*

Proof: Since Comp can be computed in $O(\log n)$ time, including retrieval of bases, the result follows from Theorem 4.5 with $\kappa = 4^{\sqrt{\log n}}$ and Theorem 5.10. ■

6 Computing Comp

The aim of this section is to reduce the work of the $O(\kappa n)$ comparisons specified in Theorem 5.10 for unbounded degree trees. Recall that Comp computes $\mathtt{mast}(v, w)$ as the maximum value over $\mathtt{Diag}(v, w)$ and $\mathtt{match}(v, w)$. The maximum diagonal of the $O(\kappa n)$ pairs in E can be easily computed in $O(\kappa n \log n)$ time by a bottom-up dynamic program, so our problem is to bound the work on matchings. Recall that the bipartite matching associated with a pair (v, w) is on the bipartite graph whose vertex sets are $\mathtt{C}(v)$ and $\mathtt{C}(w)$ and whose edges (u, v) are weighted by $\mathtt{mast}(u, v)$. For any opposing pair of core nodes (v, w), we will let G_{vw} such a graph. We will reduce the size of the matchings by reducing the number of nodes and edges, and the total sum of the edge weights involved. Initially, these values are $O(n^2)$, $O(n^2)$, and $O(n^3)$, respectively. Our goal is to reduce them to $O(\kappa n)$, $O(\kappa n)$, and $O(\kappa^2 n)$. We will do this by deleting some edges and reducing the weights of others. In general, we will be building a set of matching graphs H_{vw} from the original matching graphs G_{vw}, such that their maximum weighted matching of the two are closely related. Note that we will not explicitely build the G_{vw} since their total size can be as large as order n^2.

The vertices will be bounded by the number of edges. Recall that nodes come in three types: side, critical and spine. All edges in the matching are between children of core nodes. Let a *side-side* edge be

time $O(|\mathtt{A}(\mathtt{SideT}(I))| \log |\mathtt{A}(\mathtt{SideT}(I))|)$ we can compute $\mathtt{side\text{-}mast}(v, I)$ for all vertices $v \in \mathtt{LCA}(V)$. Now by Lemma 4.2, given any vertex v, we can compute $\mathtt{side\text{-}mast}(v, I)$ in $O(\log n)$.

Next we construct a balanced binary tree for each spine, where the leaves present singleton core nodes of a spine, and where the parent of two children represents the interval derived by merging the (adjacent) intervals of the two children. Consider some fixed level. The sets of vertices of these intervals are disjoint, and hence so are the sets of side labels. Thus, the above preprocessing can be done for all the intervals on our level in time $O(n \log n)$. Doing this for all $O(\log n)$ levels gives us a total preprocessing time of $O(n \log^2 n)$.

Now, let v be an arbitrary core vertex and I an arbitrary interval. Clearly I can be achieved as the concatenation of at most $2 \log n$ intervals from $\mathcal{I}$, and hence $\mathtt{side\text{-}mast}(w, I)$ can be retrieved in time $O(\log^2 n)$, as desired. ∎

Given an opposing pair (I_0, I_1) of intervals, we call $\mathtt{mast}(\mathtt{c}(I_0), \mathtt{c}(I_1))$ the *floor*, $\mathtt{mast}(\mathtt{c}(I_0), \mathtt{r}(I_1))$ and $\mathtt{mast}(\mathtt{r}(I_0), \mathtt{c}(I_1))$ the *diagonals*, and $\mathtt{mast}(\mathtt{r}(I_0), \mathtt{r}(I_1))$ the *ceiling* of the pair. Thus, if I_0 and I_1 are intervals of spines S_0 and S_1, then the diagonals together with the floor are exactly the values from the base of $(\mathtt{V}(I_0), \mathtt{V}(I_1))$ that are not in the base of $(\mathtt{V}(S_0), \mathtt{V}(S_1))$. Proposition 5.1 together with Lemma 5.2 states that given the diagonals of a boring interval pair, we can compute the ceiling in $O(\log^2 n)$ time.

Recall that we are trying to implement Comp-Spines(S_0, S_1) (Procedure 3) where (S_0, S_1) is any pair of opposing spines, which is hereby fixed. To ease the presentation, we will identify S_0 and S_1 with the intervals with the same vertex sets. Moreover, we set $m_0 = |\mathtt{V}(S_0)|$, $m_1 = |\mathtt{V}(S_1)|$, and $l = |\mathtt{A}(\mathtt{SideT}(S_0)) \cap \mathtt{A}(\mathtt{SideT}(S_1))|$. Notice that the number of maximal boring interval pairs can be as bad as $\Omega(l^2)$. However, we are going to identify $O(m_0 + m_1 + l \log n)$ boring interval pairs together with $O(m_0 + m_1 + l)$ vertex pairs whose comparisons will be sufficient to implement Comp-Spines(S_0, S_1).

First, for conceptual reasons, for $i = 0, 1$, we define an *interval tree* $\mathcal{I}_i$ to be a binary tree on the subintervals of S_i as follows. The root interval is S_i itself. Now, let $I =]v, w]$ be any interval in $\mathcal{I}_i$. If $v = w$, then I is a leaf of $\mathcal{I}_i$, otherwise, denote by u the vertex such that $|\mathtt{V}(]u, w])| = \lceil |\mathtt{V}(J)|/2 \rceil$. Then $]v, u]$ and $]u, w]$ are the children of I, and they are denoted by I^r and I^s respectively. For a leaf I, we set $I^r = I^s = I$. Notice that $\mathcal{I}$ had depth $\lceil \log_2 |\mathtt{V}(S_i)| \rceil$. Also, notice that the set of vertex sets of the intervals on any specific level of $\mathcal{I}_i$ partition $\mathtt{V}(S_i)$.

With reference to the interval trees $\mathcal{I}_0$ and $\mathcal{I}_1$, we are going to construct a rooted tree $\mathcal{P}$ whose vertices are interval pairs from $\mathcal{I}_0 \times \mathcal{I}_1$. The root pair is (S_0, S_1). Let (I_0, I_1) be any pair from $\mathcal{P}$. Then (I_0, I_1) is internal if and only if either

- (I_0, I_1) is interesting and one of I_0 and I_1 contains more than one vertex.
- One of I_0 and I_1 contains the spine root, and the other contains more than one vertex.

Such an internal node has four children: $(I_0^l, I_1^l), (I_0^l, I_1^r), (I_0^r, I_1^l), (I_0^r, I_1^r)$. We are going to compute the ceiling and the diagonals of all the pairs in $\mathcal{P}$. Clearly, we will thereby implement Comp-Spines since if r is the root of one of the spines and v is a vertex from the other spine, then the second condition defining $\mathcal{P}$ ensures that $\mathtt{mast}(r, v)$ is the ceiling of some pair in $\mathcal{P}$. The problem is the complexity of the computation.

Proposition 5.3 *The size and construction time for $\mathcal{P}$ is $O(m_0 + m_1 + l \log n)$*

Proof: The essential observation is that there can be at most l interesting pairs on each level, and that these can be found in linear time $O(l)$. Moreover, we need the observation that for any interval $I_i \in \mathcal{I}_i$ with $|\mathtt{V}(I_i)| > 1$ there is exactly one interval $I_{\bar{i}} \in \mathcal{I}_{\bar{i}}$ such that $(I_0, I_1) \in \mathcal{P}$ and $\mathtt{r}(I_{\bar{i}}) = \mathtt{r}(S_{\bar{i}})$, and that this interval is trivially found in time $O(1)$. The two observations together give the desired bounds for the internal pairs in $\mathcal{P}$, but for each internal pair, there are at most 4 leaf pairs. ∎

We are going to compute the ceiling and the diagonals of all the pairs in $\mathcal{P}$ respecting the linear order $<$ such that for each internal pair (I_0, I_1) of $\mathcal{P}$, we have $(I_0^l, I_1^l) < (I_0^l, I_1^r) < (I_0^r, I_1^l) < (I_0^r, I_1^r) < (I_0, I_1)$. Thus, whenever we compute a pair, we can assume that all previous pairs have been computed.

Observation 5.4 *For any internal pair of $\mathcal{P}$, the ceiling and diagonals are ceiling and diagonals of its children.*

Thus we only need to worry about leaf pairs.

Observation 5.5 *The floor of any pair in $\mathcal{P}$ is the ceiling or diagonal of some preceding pair.*

Proposition 5.6 *Any diagonal of a pair in $\mathcal{P}$ can be computed in time $O(\log^2 n)$.*

following theorem which will be used to bound the overall work for the bounded-degree and general versions of the MAST problem throughout the remainder of the paper.

Theorem 4.5 *Assume for some monotone function* $C : \mathbb{R}_{\geq 1} \to \mathbb{R}$ *that* COMP-CORE-TREES *can be computed in time at most* $C(\kappa^a n)$ *where* a *is a constant independent of our parameter* κ*. Moreover, assume that* $C(x) = x^{1+\varepsilon} f(x)$*, where* $\varepsilon \geq 0$ *is a constant,* $f(x) = O(x^{o(1)})$*,* f *is monotone, and for some constants* b_1, b_2*,* $\forall x, y \geq b_1 : f(xy) \leq b_2 f(x) f(y)$*. If* $\varepsilon = 0$*, setting* $\kappa = 4^{\sqrt{\log n}}$*, we can compute* MAST *in time* $O(C(n) c^{\sqrt{\log n}})$ *for some constant* c*. Otherwise, if* $\varepsilon > 0$*, setting* $\kappa = \sqrt[\varepsilon]{4}$*, we can compute* MAST *in time* $O(C(n))$*. Thus* MAST *is generally computable in time* $O(n^{1+o(1)} + C(n))$*.*

5 Computing COMP-SPINES

In order to implement COMP-SPINES (Procedure 3) we introduce the concept of intervals. For spine S with critical child c, let $v, w \in \mathtt{V}(S) \cup \{c\}$. Moreover, assume that v is below or equal to w. Then v and w characterize an *interval* I of S, denoted by $]v, w]$. If, instead, I is the given value, we denote v and w by $\mathtt{c}(I)$ and $\mathtt{r}(I)$. By $\mathtt{V}(I)$ we denotes the set of vertices that are descendants of w and strict ancestors of v. Thus, if $v \neq w$, the interval I corresponds directly to a segment of S. If $v = w$, the interval I corresponds to the empty segment together with a position in S, or just below S if $v = w = c$. Let I and J be intervals from opposing core trees. We say that the pair of opposing intervals (I_0, I_1) is *interesting* if and only if $\mathtt{A}(\mathtt{SideT}(I_0)) \cap \mathtt{A}(\mathtt{SideT}(I_1)) \neq \emptyset$. A pair which is not interesting is said to be *boring.*

Lemma 5.1 *If the interval pair* (I_0, I_1) *is boring, then* $\mathtt{mast}(\mathtt{r}(I_0), \mathtt{r}(I_1))$ *is*

$$\max \begin{cases} \mathtt{mast}(\mathtt{c}(I_0), \mathtt{r}(I_1)), \mathtt{mast}(\mathtt{c}(I_1), \mathtt{r}(I_0)), \\ \mathtt{side\text{-}mast}(\mathtt{c}(I_0), I_1) + \mathtt{side\text{-}mast}(\mathtt{c}(I_1), I_0) \end{cases}$$

Here $\mathtt{side\text{-}mast}(v, I) = \max\{\mathtt{mast}(v, \mathtt{r}(t)) \mid t \in \mathtt{SideT}(I)\}$.

Proof: Let *lhs* and *rhs* denote the left and the right hand side of the equality of the lemma. First we prove $lhs \geq rhs$. Clearly, $\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1)\} \geq \max\{\mathtt{mast}\{\mathtt{c}(I_0), \mathtt{r}(I_1\}, \mathtt{mast}\{\mathtt{c}(I_1), \mathtt{r}(I_0)\}\}$. For $i = 0, 1$, let $t_{\bar{i}}$ be a side tree in $\mathtt{SideT}(I_{\bar{i}})$ such that $\mathtt{mast}\{\mathtt{c}(I_i), \mathtt{r}(t_{\bar{i}})\} = \mathtt{side\text{-}mast}\{\mathtt{c}(I_i), I_{\bar{i}}\}$. Let p_i and c_i denote the parent and core sibling of $\mathtt{r}(t_{\bar{i}})$. By application of Lemma 3.1, we get the following inequalities:

$$\begin{aligned} &\mathtt{mast}\{c_i, \mathtt{r}(t_{\bar{i}})\} \geq \mathtt{mast}\{\mathtt{c}(I_i), \mathtt{r}(t_{\bar{i}})\} \text{ for } i = 0, 1, \\ &\mathtt{mast}\{f_0, f_1\} \geq \mathtt{mast}\{c_0, \mathtt{r}(t_1)\} + \mathtt{mast}\{\mathtt{r}(t_0), c_1\}, \\ &\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1)\} \geq \mathtt{mast}\{f_0, f_1\}. \end{aligned}$$

Hence, it follows that $\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1\} \geq \mathtt{side\text{-}mast}\{\mathtt{c}(I_0), I_1\} + \mathtt{side\text{-}mast}\{\mathtt{c}(I_1), I_0\}$, and we may therefore conclude that $lhs \geq rhs$.

We show that $lhs \leq rhs$ by contradiction. Assume there is a pair $(f_0, f_1) \in \mathtt{V}(I_0) \times \mathtt{V}(I_1)$ such that $\mathtt{mast}\{f_0, f_1\} > rhs$, and fix (f_0, f_1) to be a minimal such pair. First, we observe that the strict inequality implies that $s_0 \neq \mathtt{c}(I_0)$ and $s_1 \neq \mathtt{c}(I_1)$. For $i = 0, 1$, let c_i denote the core child of f_i. By the minimality of (f_0, f_1) we cannot have $\mathtt{mast}\{f_i, f_{\bar{i}}\} = \mathtt{mast}\{c_i, f_{\bar{i}}\}$. Also by minimality, we cannot have $\mathtt{mast}\{f_i, f_{\bar{i}}\} = \mathtt{mast}\{s_i, f_{\bar{i}}\}$, where s_i is a side child of f_i, for since our pair of intervals is borring, $\mathtt{mast}\{s_i, f_{\bar{i}}\} = \mathtt{mast}\{s_i, c_{\bar{i}}\} \leq \mathtt{mast}\{f_i, c_{\bar{i}}\}$. Thus, by Lemma 3.1, we have $\mathtt{mast}\{f_0, f_1\} = \mathtt{match}\{f_0, f_1\}$. Let M be a minimal matching in $\mathtt{C}(f_0) \times \mathtt{C}(f_1)$ such that $\mathtt{match}\{f_0, f_1\} = \sum_{(v_0, v_1) \in M} \mathtt{mast}\{v_0, v_1\}$. Since our interval pair is borring, we can only have an edge in M if either its head or its tail is core. By the minimality of (f_0, f_1), we cannot have $M = \{(c_0, c_1)\}$. Thus, we have $M \subseteq \{(c_0, s_1), (c_1, s_0)\}$ where s_i is a specific side child of f_i. Putting everything together we get:

$$\begin{aligned} rhs &< \mathtt{match}\{f_0, f_1\} \\ &= \textstyle\sum_{(v_0, v_1) \in M} \mathtt{mast}\{v_0, v_1\} \\ &= \mathtt{mast}\{c_0, s_1\} + \mathtt{mast}\{c_1, s_0\} \\ &= \mathtt{mast}\{\mathtt{c}(I_0), s_1\} + \mathtt{mast}\{\mathtt{c}(I_1), s_0\} \\ &\leq \mathtt{side\text{-}mast}\{\mathtt{c}(I_0), I_1\} + \mathtt{side\text{-}mast}\{\mathtt{c}(I_1), s_0\} \\ &\leq rhs, \end{aligned}$$

and hence the desired contradiction. ∎

What makes this useful is the following technical lemma:

Lemma 5.2 *After a general* $O(n \log^2 n)$ *preprocessing based on the base of the core trees, given any pair of a vertex* v *from one core tree and an interval* I *from the other core tree, we can compute* $\mathtt{side\text{-}mast}(v, I)$ *in time* $O(\log^2 n)$*.*

Proof: Let I be an arbitrary fixed interval, and denote by V the set of at most $|\mathtt{A}(\mathtt{SideT}(I))|$ opposing core vertices that are interesting with respect to some vertex in I. Our first observation is that by a simple dynamic bottom-up program in

B.4. For $i \in \{0,1\}$, let C_i' denote the tree obtained by identifying each spine of C_i with a single vertex.

B.5. Let $\mathcal{O}$ be an ordering of $\mathtt{V}(C_0') \times \mathtt{V}(C_1')$ so that if (v_0, v_1) is before (w_0, w_1) in $\mathcal{O}$, then (w_0, w_1) is not a base pair of (v_0, v_1).

B.6. For each (c_0, c_1) in increasing order in $\mathcal{O}$ do

B.6.1. `Case:` c_0 and c_1 are critical. Do

B.6.1.1. Compute COMP$(c_0, c_1,)$.

B.6.2. `Case:` c_0 is a spine node and c_1 is a critical node or vice versa. Do:

B.6.2.1. Let $s_1, \ldots, s_k$ be the spines nodes of c_0 in ascending order.

B.6.2.2. For $i \leftarrow 1$ to k compute COMP(c_0, s_i).

B.6.3. `Case:` c_0 and c_1 are spine nodes. Do:

B.6.3.1. Compute COMP-SPINES(c_0, c_1).

For the sake of exposition, we will simplify the discussion of the above algorithm by referring to steps B.4 to B.6.3.1 as COMP-CORE-TREES. Since we can identify the core trees in linear time, the time to compute MAST is the time for a single call to PREP-CORE-BASE and a single call to COMP-CORE-TREES. The time to compute COMP-CORE-TREES now depends on the time for COMP and COMP-SPINES, as follows.

Observation 3.3 COMP-CORE-TREES *calls* COMP $O(\kappa n)$ *times and* COMP-SPINES $O(\kappa^2)$ *times. All other processing is done in time* $O(\kappa n)$.

4 Computing PREP-CORE-BASE

The goal of PREP-CORE-BASE is to preprocess $\mathcal{T}_0$ and $\mathcal{T}_1$ so that we may quickly retrieve values of the form $\mathtt{mast}(r, c)$, where r is the root of some side tree in $\mathcal{T}_i$, and c is a core node in $\mathcal{T}_{\bar{i}}$. As a subroutine, we will use the following procedure.

Procedure 4 TREES(T_0, T_1) *where each* T_i *is an evolutionary tree. Computes* $\mathtt{mast}(\mathtt{r}(T_i), v_{\bar{i}})$ *for* $i = 0, 1$, *and for all* $v_{\bar{i}} \in \mathtt{V}(T_{\bar{i}})$.

Notice that TREES is a strengthening of MAST in that it computes more values. Algorithm B implements TREES except for the mast values between the roots and their opposing side nodes. In this section, we will refer to these missing values as the *side values*.

In order to make Algorithm B implement TREES completely, we extend our specification of PREP-CORE-BASE to compute the side values as well. That is, PREP-CORE-BASE should not only make the values from the base of the core trees but also the side values quickly retrievable.

We will show that with some $O(n)$ additional processing, in order to compute PREP-CORE-BASE it suffices to compute TREES(s, t_s) where s is a side tree of $\mathcal{T}_i$ for $i \in \{0,1\}$ and $t_s = \mathcal{T}_{\bar{i}}|\mathtt{A}(s)$. Since the label set of different side trees of the same evolutionary tree are disjoint, the following lemma implies that we can compute all the t_s in linear time.

Lemma 4.1 ([6]) *Given a partition* $L_0, \ldots, L_x$ *of the labels of an evolutionary tree* T *of size* n, *we can compute all of* $T|L_0, \ldots, T|L_x$ *in total* $O(n)$ *time.*

Recall that if $\{v, w\}$ is in the base of the core trees then one of v and w is the root of a side tree (and the other is either a core vertex or a root of a side tree). Hence the following simple lemmas shows that computing TREES(s, t_s) essentially suffices for retrieving the needed base of the core trees:

Lemma 4.2 *Let* s *be a side tree of* $\mathcal{T}_i$ *and set* $W = \mathtt{V}(t_s)$. *Then for all* $v \in \mathtt{V}(\mathcal{T}_{\bar{i}})$ *we have* $\mathtt{mast}(\mathtt{r}(s), v) = 0$ *if* v *has no descendant in* W; *otherwise* $\mathtt{mast}(\mathtt{r}(s), v) = \mathtt{mast}(\mathtt{r}(s), w)$ *where* w *is the unique first descendant of* v *in* W. ∎

Lemma 4.3 *For any* $v \in \mathtt{V}(s)$, *where* s *is a side tree of* $\mathcal{T}_i$, $\mathtt{mast}(v, \mathtt{r}(\mathcal{T}_{\bar{i}})) = \mathtt{mast}(v, \mathtt{r}(t_s))$. ∎

Proposition 4.4 PREP-CORE-BASE *can be computed in time*

$$O(n) + 2 \max_{\substack{\sum n_i = n, \\ 1 \le n_i \le \lfloor n/\kappa \rfloor}} \left(\sum_{n_i} \mathtt{time}(\text{TREES}(n_i)) \right).$$

Proof: Lemma 4.1 allows us to compute all (s, t_s) pairs in $O(n)$ time. Clearly, $\sum_s \mathtt{A}(s) = n$, and for each s, $|\mathtt{A}(s)| < n/\kappa$. Hence it follows that we can compute TREES(s, t_s) for all s within the the stated time bound. By Lemma 4.3, we are now done with the side values. Concerning the base of the core trees, using, say, Euler tours, it is a standard exercise to organize in linear time all the $O(n)$ values from TREES(s, t_s) so as to make the descendant look-up from Lemma 4.2 computable in $O(\log n)$ time. ∎

Recall that $\mathtt{time}(\text{MAST}(n)) \le \mathtt{time}(\text{TREES}(n)) \le \mathtt{time}(\text{PREP-CORE-BASE}(n)) + \mathtt{time}(\text{COMP-CORE-TREES}(n))$. Inserting the bound on $\mathtt{time}(\text{PREP-CORE-BASE}(n))$ from Proposition 4.4, we get a recursion formula for the time it takes to compute Trees and hence MAST. This formula implies the

can match the root of one tree to one of the children of the other tree. It is clear from this lemma that we need some values on subtrees in order to compute the mast of two nodes. To simplify discussion, we introduce the following notation. By the *base pairs* of (x_0, x_1), where x_i is a set of vertices in $\mathcal{T}_i$, or just a single vertex, we understand the set of opposing pairs (w_0, w_1) such that either $w_0 \in \mathtt{C}(x_0)$ and $w_1 \in \mathtt{C}(x_1) \cup x_1$, or vice versa. By the *base* of (x_0, x_1) we mean the set of values $\mathtt{mast}(w_0, w_1)$ for all base pairs of (x_0, x_1). Thus the base forms the set of values needed by a dynamic program to compute the mast values for (x_0, x_1).

Lemma 3.1 suggests the following dynamic programming algorithm for the MAST problem.

Algorithm A: First Algorithm for MAST.

A.1. Input T_0 and T_1.

A.2. Let $\mathcal{O}$ be an ordering of $\mathtt{V}(T_0) \times \mathtt{V}(T_1)$ so that if (v_0, v_1) is before (w_0, w_1) in $\mathcal{O}$, then (w_0, w_1) is not a base pair of (v_0, v_1).

A.3. For each (v_0, v_1) in increasing order in $\mathcal{O}$ do

A.3.1. Compute COMP(v_0, v_1).

In this algorithm, the procedure COMP is defined as follows.

Procedure 1 COMP(v_0, v_1), *where* $(v_0, v_1) \in \mathtt{V}(\mathcal{T}_0) \times \mathtt{V}(\mathcal{T}_1)$, *computes* $\mathtt{mast}(v_0, v_1)$.

The above naïve algorithm computes MAST by applying COMP to all the $O(n^2)$ pairs of nodes from $\mathtt{V}(\mathcal{T}_0) \times \mathtt{V}(\mathcal{T}_1)$. The bottom-up ordering of $\mathcal{O}$ guarantees that the base of a node pair is always ready when we compute COMP on that pair. For bounded degree, COMP can be computed in $O(1)$, thus giving $O(n^2)$ total work. For unbounded degree, the bottleneck in the comparisons is the matchings, which sum up to $O(\sum_{v \in \mathtt{V}(T_1), w \in \mathtt{V}(T_2)} \mathtt{d}(v) \cdot \mathtt{d}(w) \sqrt{\mathtt{d}(v) + \mathtt{d}(w)} \log n) = O(n^2 \sqrt{n} \log n)$ using the matching algorithm in [11]. In [6], we showed that most matching graphs can be preprocessed so that they are quite sparse. We where able to show a bound of $O(n^2)$ time for all calls to COMP, thus showing that the MAST of two trees can be computed in $O(n^2)$, even when the degree is unbounded.

We improve this result in two ways, first by reducing the number of comparisons evaluated in the dynamic program and secondly by reducing the work done on the matchings. This second phase is analogous to our previous work [6] in reducing the matching work, but we require new techniques since the new results are much tighter. For the moment, we will focus entirely on reducing the number of comparisons, returning to weighted matchings in §6.

3.2 A Faster Algorithm

Our key to sparsifying the dynamic program is to introduce *core trees* which are defined in terms of some parameter κ. We say that a node is a *core node* if it has more than n/κ descendant leaves. Otherwise it is a *side node*. Then the *core tree* is the component induced by the core nodes, and the *side trees* are the components induced by side nodes. 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 the core tree has at most κ leaves, since the leaves are roots of disjoint sub-trees, each with at least n/κ leaves.

We make one final distinction within the core tree. We partition the nodes of the core tree into *critical nodes* and *spine nodes*. A critical node is either the root, a leaf of the core tree, or a branching node, i.e. a core node with at least two children which are core nodes. If a core node is not a critical node then it is a spine node. Notice the spine nodes can be further partitioned into connected components. In fact, each such connected component forms a chain of nodes where all nodes but the last have exactly one core child - but possibly order n side children. We call each such a component a *spine*. Note that we have $O(\kappa)$ critical nodes and spines, but $O(n)$ spine nodes. By $\mathtt{SideT}(x)$, where x is a set of core nodes, we will mean the set of side trees T, such that $t \in T$ if $\mathtt{r}(t) \in \mathtt{C}(x)$. The following trivial fact gives one of the main uses of side trees.

Fact 3.2 *Let* $S_1, \ldots, S_x$ *and* $S'_1, \ldots, S'_y$ *be a partitionings of the side trees of* $\mathcal{T}_0$ *and* $\mathcal{T}_1$, *respectively. Let* $l_{ij} = |\bigcup_{t \in S_i} \mathtt{A}(t) \cap \bigcup_{t \in S'_j} \mathtt{A}(t)|$. *Then* $\sum_{i=1}^{x} \sum_{j=1}^{y} l_{ij} = n$.

In the following sections we will find efficient implementations of the following two procedures:

Procedure 2 PREP-CORE-BASE, *makes any value in the base of the core trees available in* $O(\log n)$ *time.*

Procedure 3 COMP-SPINES(S_0, S_1), *where* S_0 *and* S_1 *are opposing spines, computes* $\mathtt{mast}(\mathtt{r}(S_i), v_{\bar{i}})$ *for* $i = 0, 1$, *and for all* $v_{\bar{i}} \in \mathtt{V}(S_{\bar{i}})$.

With these procedures we get the following algorithm for MAST:

Algorithm B:

B.1. Input $\mathcal{T}_0$ and $\mathcal{T}_1$.

B.2. Identify their core trees $\mathcal{C}_0$ and $\mathcal{C}_1$.

B.3. Compute PREP-CORE-BASE.

for this problem. Another variant is that of considering three or more trees. Amir and Keselman [2] showed that the MAST problem on three or more trees is *NP*-hard for trees of unbounded degree, while showing that if the tree with the least degree bound has a bound of b, then the MAST of k trees can be computed in $O(kn^b)$ time. As a side effect of our construction, if $k = 2$ and b is a constant, we show in this paper an $O(nc^{\sqrt{\log n}}) = O(n^{1+o(1)})$ time algorithm for the MAST problem.

In §2, we show the reduction from UWBM to MAST. In §3, we give some basic definitions and give an overview of our algorithm. In §4, we prove the correctness of the algorithm and give a general sketch of its time complexity. In §5, we reduce the number of comparisons in the dynamic program and finish the time analysis for the bounded degree case. In §6, we describe how to reduce the work of the matchings and finish the time analysis for the unbounded degree case.

2 Lower Bound

We show a reduction from a size n UWBM problem to an $O(n)$ leaf MAST problem. We define the size of a UWBM instance to be the sum of the edge weights, hence an n size UWBM instance can code a WBM instance with total integer edge weight n and up to $O(n)$ edges and $O(n)$ nodes. Using the the best algorithm known [11] for WBM, such a problem can be solved in $O(n^{1.5}\log n)$.

We now show a reduction from UWBM to MAST. Given a connected weighted bipartite graph $G = (U \cup V, E, W : E \to \mathcal{N})$ such that $\sum_{e\in E} W(e) = n$, construct evolutionary trees T_U and T_V with $O(n)$ labels as follows. For each $X \in \{U, V\}$, set r_X to be the root of T_X, and for each $x \in X$, create a child c_x of r_X. For each $e = \{u, v\} \in E$, add $W(e)$ children to c_u in T_U and label them $< u, v, i >$, $1 \le i \le W(e)$, and add children with the same labels to c_v in T_V. Finally, pick $n+2$ new labels and build a star tree $\mathcal{S}$ on the labels. Attach the root of one copy of $\mathcal{S}$ to the root of T_X and attach the root of another copy of $\mathcal{S}$ to the root of T_Y. The size of the star $\mathcal{S}$ guarantees that any solution to MAST will map the roots to each other. Hence there is a bijection between the maximum weight matchings M and the maximum agreement subsets B such that if $\{v, w\} \in M$ then $\mathtt{A}(c_v) \cap \mathtt{A}(c_w) \subseteq B$.

Theorem 2.1 *There is a linear reduction from* UWBM *to* MAST. ■

Amir and Keselman [2] show a similar reduction from *Three-Dimensional Matching* with which they proved that the MAST problem on 3 or more trees is NP-Hard.

3 Ideas and Outline

3.1 Definitions and Ideas

For the rest of the paper, fix an instance of the MAST problem consisting of two evolutionary trees $\mathcal{T}_0$ and $\mathcal{T}_1$ for some common set A of species. We measure the size n of the problem as the cardinality of A which equals the number of leaves for both $\mathcal{T}_0$ and $\mathcal{T}_1$. To simplify some boundary cases in the discussion below, we allow the situation where $\mathcal{T}_0$ and $\mathcal{T}_1$ are empty. In that case $\mathtt{MAST}(\mathcal{T}_0, \mathcal{T}_1) = |\mathtt{A}| = n = 0$. We describe the SW algorithm after first introducing some notation.

Given an evolutionary tree T, by $\mathtt{V}(T)$ we denote the set of its vertices, by $\mathtt{r}(T)$ its root, by $\mathtt{A}(T)$ the set of its leaf labels. For $v \in \mathtt{V}(T)$, let $\mathtt{C}(v)$ be the set of children of v and let $\mathtt{d}(v)$ be its degree. If $v \in \mathtt{V}(T)$, then $\mathtt{t}(v)$ denotes the subtree descending from v. For a set of nodes $V \subseteq \mathtt{V}(T)$, we define $\mathtt{LCA}(V)$ to be the closure of V with respect to least common ancestors, and we set $\mathtt{C}(V) = \bigcup\{\mathtt{C}(v) | v \in V\} \setminus V$. We call the members of $\mathtt{C}(V)$ the *proper children* of V. We will commonly use $\bar{0}$ to mean 1, and vice versa. By an *opposing pair* we will refer to a pair (x, y) where x relates to $\mathcal{T}_0$ and y relates to $\mathcal{T}_1$, or vice versa. Let $G = (V, E, W)$ be a weighted bipartite graph. Then $\mathtt{MWM}(G)$ is the value of a maximum weighted matching on G.

For any opposing pair (v, w) of vertices, let $\mathtt{mast}(v, w)$ denote MAST of $\mathcal{T}_0|B$ and $\mathcal{T}_1|B$ where $B = \mathtt{A}(\mathtt{t}(v)) \cap \mathtt{A}(\mathtt{t}(w))$. Thus, loosely speaking $\mathtt{mast}(v, w)$ is the MAST of the subtrees rooted respectively at v and w. The following lemma appears, with some minor modifications, in [17] and is the basis for their dynamic program approach to the unrooted version of this problem.

Lemma 3.1 ([17]) $\forall v \in V(\mathcal{T}_0), w \in V(\mathcal{T}_1)$,

$$\mathtt{mast}(v, w) = \begin{cases} |\mathtt{A}(v) \cap \mathtt{A}(w)| & \text{if } v \text{ or } w \text{ is a leaf;} \\ \max\{\mathtt{Diag}(v, w), \mathtt{match}(v, w)\} & \text{otherwise} \end{cases}$$

where $\mathtt{Diag}(v, w) = \{\mathtt{mast}(v, w_1) | w_1 \in \mathtt{C}(w)\} \cup \{\mathtt{mast}(v_0, w) | v_0 \in \mathtt{C}(v)\}$ *and* $\mathtt{match}(v, w) = \mathtt{MWM}((\mathtt{C}(v) \cup \mathtt{C}(w), \mathtt{C}(v) \times \mathtt{C}(w), \mathtt{mast}(\cdot, \cdot)))$.

Intuitively, this expression says that when comparing the root of two trees, we can either match the two roots together in the final agreement tree, in which case we try all ways of matching their children together, or we

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 tree T 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 history in $T|B$ as it does in T. Formally, $T|B$ denotes the 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. The restriction operator immediately implies a similarity measure on trees. Finden and Gordon [9] introduced just such a measure, formalizing the intersection problem as follows.

Problem: The Maximum Agreement Subtree Problem (`MAST`)

Input: A pair (T_0, T_1) of evolutionary trees for some common set A of species.

Output: A maximum cardinality subset B of A such that $T_0|B$ and $T_1|B$ have a (leaf-label preserving) isomorphism.

Finden and Gordon gave a heuristic method for computing the maximum agreement subtree of two 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 an $O(n^{(\frac{1}{2}+\epsilon)\log n})$ time algorithm for the binary `MAST` problem. Steel and Warnow [17] gave the first polynomial algorithm, which we will refer to as SW. The SW algorithm 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. We showed in [6] that the SW algorithm can be modified to yield an $O(n^2)$ time algorithm for the unbounded case.

Both the SW algorithm and our modification of it perform some computation—a weighted bipartite matching—for each pair of nodes from the input trees. Hence this approach cannot give a $o(n^2)$ time algorithm for the `MAST` problem. We therefore run into the dynamic programming bottleneck which is endemic in computational molecular biology. A wide variety of biocomputing problems, e.g. in the string-edit-distance problem, RNA secondary structure, etc., have solutions which involve dynamic programming. In this paper we use sparsity conditions to break the dynamic programming bottleneck and achieve subquadratic running times. In fact, we show surprisingly tight bounds. Our main result is an algorithm which solves `MAST` within the same asymptotic time bound as that for solving *Unary Weighted Bipartite Matching (*`UWBM`*)*, i.e. a weighted bipartite matching where the size of the input is measured as the sum of the weights of all edges—so unweighted bipartite matching is a special case. More precisely, we show that $\texttt{time}(\texttt{MAST}(n)) = O(n^{1+o(1)} + \texttt{time}(\texttt{UWBM}(n)))$[1]. Using the best known algorithm [11] for weighted bipartite matching, this gives us an $O(n^{1.5}\log n)$ time algorithm for the `MAST` problem, thus breaking the $\Omega(n^2)$ bottleneck.

While `UWBM` does not often appear as a natural upper bound, and typically one see reference to either unweighted or fully weighted bipartite matching instead, we show that the unary weighting is inherent in bounding the complexity of `MAST` by observing that, in fact, $\texttt{time}(\texttt{MAST}(n)) = \Omega(\texttt{time}(\texttt{UWBM}(n)))$. Thus for all intents and purposes, our reduction is optimal, since getting the complexity of `UWBM`, or just bipartite matching, down anywhere near $O(n^{1+o(1)})$ is a long-standing open problem.

The fact that our algorithm works by sparsity means that we identify a small set of *significant computations* in the dynamic program. The exact size of this set depends on the running time of the bipartite matching algorithm used, thus we balance the time spent at a single node pair in the dynamic program with the number of such node pairs we evaluate. The key to this balancing is the *parameterized core trees* which we introduce in this paper. The core tree is a generalization of the separator of a tree which we use to guide our computation.

Finally, we note that two variants of the `MAST` problem have been investigated. First, the unrooted version was the primary focus of the Steele and Warnow paper [17]. The above cited complexities for their algorithms apply to the unrooted case, to which the rooted case reduces. In [6], our main result was to improve the complexity of the unbounded case to $O(n^2 c^{\sqrt{\log n}})$. We believe that the unrooted problem is much harder than the rooted case and know of no $O(n^2)$ algorithm

[1] This complexity is understood to be modulo a class of "well-behaved" functions explicitly defined in Theorem 4.5.

Optimal Evolutionary Tree Comparison by Sparse Dynamic Programming (Extended Abstract)

Martin Farach*
Department of Computer Science
Rutgers University
Piscataway, NJ 08855
USA

Mikkel Thorup†
Department of Computer Science
University of Copenhagen
2100 København Ø
Denmark

Abstract

In computational biology one is often interested in finding the concensus between different evolutionary trees for the same set of species. A popular formalizations is the Maximum Agreement Subtree Problem *(*`MAST`*) defined as follows: given a set A and two rooted trees $\mathcal{T}_0$ and $\mathcal{T}_1$ leaf-labeled by the elements of A, find a maximum cardinality subset B of A such that the restrictions of $\mathcal{T}_0$ and $\mathcal{T}_1$ to B are topologically isomorphic. Polynomial time solutions exist, but they rely on a dynamic program with $\Theta(n^2)$ nodes—and $\Theta(n^2)$ running time. We sparsify this dynamic program and show that* `MAST` *is equivalent to Unary Weighted Bipartite Matching (*`UWBM`*) modulo an $O(nc^{\sqrt{\log n}})$ additive overhead. Applying the best bound for* `UWBM`*, we get an $O(n^{1.5}\log n)$ algorithm for* `MAST`*. From our sparsification follows an $O(nc^{\sqrt{\log n}})$ time algorithm for the special case of bounded degrees. Also here the best previous bound was $\Theta(n^2)$.*

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 relationship 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 [3, 4]. 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. [7, 8, 10, 15, 16]). More recently, a variety of solutions have been examined rigorously ([1, 5, 12, 13]). 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.

More concretely, let A be a set of species. Then we will define an *evolutionary tree*, T, on A to be a rooted tree, with no degree 1 internal nodes, such that the leaves of T are uniquely labeled with the elements of A. In such a tree, the leaves represent the species under consideration, and the internal nodes represent posited ancestors. Suppose now that we are given two trees, T_0 and T_1, which are evolutionary trees on the same species set A. If the two trees differ, it is reasonable to ask for the "intersection" of the information

*`farach@cs.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.