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

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

[13] D. M. Hillis. Molecular vs. morphological approaches to systematics. *Annual Review of Systematics and Ecology*, 18:23–42, 1987.

[14] J. Hunt and T. Szymanski. A fast algortihm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

[15] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[16] 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.

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

[18] G. J. Olsen. Earliest phylogenetic branchings: Comparing rRNA-based evolutionary trees inferred with various techniques. *Cold Spring Harbor Symposia on Quantitative Biology*, 52:825–837, 1987.

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

[20] P. H. A. Sneath and R. R. Sokal. *Numerical Taxonomy*. W. H. Freeman, San Francisco, California, 1973.

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

[22] D. L. Swofford and G. J. Olsen. Phylogeny reconstruction. In D. M. Hillis and C. Moritz, editors, *Molecular Systematics*, pages 411–501. Sinauer Associates Inc., Sunderland, MA., 1990.

[23] R. M. Verma. General techniques for analyzing recursive algorithms with applications. Technical report, Univ. of Houston, Computer Science Dept., October 1992.

[24] H. T. Wareham. On the computational complexity of inferring evolutionary trees. Master's thesis, Department of Computer Science, Memorial University of Newfoundland, May 1993. Technical Report no. 9301.

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

As a general remark, we note that we can get somewhat tighter bounds as follows. Let $\texttt{UWBM}(n,b)$ be the Unary Weighted Bipartite Matching problem in which the independent sets can be no larger than b and the sum of the weights in bounded by n. Let the $\texttt{MAST}(n,b)$ problem be the mast problem on two n leaf trees with degree bound b. We can generalize our results to show that $\texttt{UWBM}(n,b)$ reduces linearly to $\texttt{MAST}(n,b)$. We cannot, in general, bound the work on $\texttt{MAST}(n,b)$ by $O(n^{1+o(1)} + \texttt{time}(\texttt{UWBM}(n,b)))$, but, we note that using the Gabow/Tarjan algorithms gives a time of $O(n\sqrt{b}\log n)$ for $\texttt{UWBM}(n,b)$. In this case, we can bound the total work for $\texttt{MAST}(n,b)$ by $O(n^{1+o(1)} + n\sqrt{b}\log n)$, thus unifying the complexities of the bounded and general cases.

Acknowledgment

We thank the referees for their careful reading and helpful comments.

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 34th IEEE Annual Symp. on Foundation of Computer Science*, pages 140–147, 1994.

[2] S.F. Altschul, W. Gish, W. Miller, E.W Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] 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.

[4] M.J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *J. Algorithms*, 8:106–112, 1987.

[5] W. H. E. Day. Foreward: Comparison and consensus of classifications. *Journal of Classification*, 3:183–185, 1986.

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

[7] 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.

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

[9] J. Felsenstein. Phylogenies from molecular sequences: inference and reliability. *Annual Review of Genetics*, 22:521–65, 1988.

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

Having reduced the number of edges in our matching to $O(\kappa n)$, we can trivially conclude that the total weight is (κn^2). We will now show that we can reduce some edge weights before applying the matching algorithm so that the total becomes $O(\kappa^2 n)$. We will further show that the weight reductions are reversible in that we will easily be able to retrieve the maximum weighted matching on the original graph from the matching on the weight reduced graph. With the current best algorithm for (unary) weighted bipartite matching [12], this reduction of the weights has no significance, for the weights only effect the running time by a logarithmic factor. However, the point of this paper is to show an equivalence to unary weighted bipartite matching which holds even if more weight sensitive algorithms for unary weighted bipartite matching are found. For the weight reduction 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 $\Delta(v) = \max\{\Delta\{v, w\} | \{v, w\} \in E\}$.*

If $\Delta(v) > 1$, 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 observation 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 $O(\kappa^2 n)$ in time $O(\kappa n)$.*

Proof: The total weight of the side-side edges is at most n, so, if for each matching based on spine nodes, we apply the reduction to their two core children, the total sum of their matching weights becomes $O(n)$, and if for each matching based on 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 $O(\kappa^2)$ matchings based on 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 $O(n \text{ polylog } n + \texttt{time}(\texttt{UWBM}(\kappa^2 n)))$ on the matchings. So, by Theorem 5.9, we have that Core-Trees can be computed in time $O(n \text{ polylog } n + \texttt{time}(\texttt{UWBM}(\kappa^2 n)))$. Applying Theorem 4.6 gives the desired complexity. ∎

Inserting the best known bounds for unary weighted bipartite matching [12], with $\kappa = \sqrt[1/2]{4} = 16$, we get

Lemma 6.1 *There are graphs H_{vw} for all $\{v,w\} \in E_1$, such that* $\texttt{match}(H_{vw}) = \texttt{match}(G_{vw})$, *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. There are a total of $O(\kappa)$ such core children, and no node can be involved in more than $O(n)$ matching edges. Thus, just by consideration of non-zero edges, we see that in total we only need $O(\kappa n)$ edges for the H_{vw}. ∎

Lemma 6.2 *There are graphs H_{vw} for all $\{v,w\} \in E_2$ such that* $\texttt{match}(H_{vw}) = \texttt{match}(G_{vw})$, $|\texttt{E}(H_{vw})| \leq 3l_{vw} + 3$, *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 $\texttt{match}(H_{vw}) = \texttt{match}(G_{vw})$. 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 on 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: The matching graphs H_{vw} from Lemma 6.1 come automatically from only considering non-zero weight matching edges. In order to sparsify matching graphs of Lemma 6.2, first we choose an arbitrary ordering of the side children of each vertex. It is now meaningful to talk about intervals of side children. With the same technique that was used in the proof of Lemma 5.4, we can make an $O(n \log n)$ 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 no more than $|S| + 1$ 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 $O(|S|)$. ∎

Theorem 5.9 CORE-TREES *(Procedure 2) can be computed in time* $O(\kappa n \log^3 n + \sum_{\{v_0,v_1\}\in E} \texttt{time}(\texttt{mast}\{v_0,v_1\}))$. *Here* E *divides into two sets* E_1 *and* E_2. *The set* E_1 *contains all the* $O(\kappa n)$ *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 $(\sum_{\{S_0,S_1\}\in S} |\texttt{V}(S_0)| + |\texttt{V}(S_1)|) \leq \kappa n$. Moreover, by Fact 3.2, we get that $(\sum_{\{S_0,S_1\}\in S} |\texttt{A}(\texttt{SideT}(S_0)) \cap \texttt{A}(\texttt{SideT}(S_1))|) \leq n$. Thus, the result follows directly from Observation 3.3 together with Proposition 5.8. ■

Corollary 5.10 *For trees* T_0 *and* T_1 *with bounded degree,* TREES$\{T_0, T_1\}$ *can be computed in time* $O(nc^{\sqrt{\log n}})$.

Proof: For bounded degrees $\texttt{time}(\texttt{mast}\{v_0,v_1\})) = O(\log n)$, so the result follows from Theorem 4.6 with $\kappa = 4^{\sqrt{\log n}}$ and Theorem 5.9. ■

6 Computing the matchings

The aim of this section is to reduce the work of computing the $O(\kappa n)$ mast-values of the set E specified in Theorem 5.9, when the trees given have unbounded degree. Recall that $\texttt{mast}\{v,w\}$ is found as the maximum value over $\texttt{Diag}\{v,w\}$ and $\texttt{match}\{v,w\}$. 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 matching associated with a pair $\{v,w\}$ is on the weighted bipartite graph whose vertex sets are $\texttt{C}(v)$ and $\texttt{C}(w)$ and whose edges (u,v) are weighted by $\texttt{mast}\{u,v\}$. We denote this graph by G_{vw}. For any weighted bipartite graph G, let $\texttt{match}(G)$ be the value of the maximal weighted bipartite matching on G. Thus $\texttt{match}\{v,w\} = \texttt{match}(G_{vw})$ for all opposing vertex pairs $\{v,w\}$. 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 the maximum weight of a matching in G_{vw} can be deduced from the maximum weight of a matching in H_{vw}. Note that we will not explicitly build the G_{vw} since their total size can be as large as $\Omega(n^2)$.

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 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 opposing pairs in E from Theorem 5.9 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 separately.

Thus, all preprocessing of $I_1, \ldots, I_k$ is done in time $O(n)$, so the total preprocessing is done in time $O(n \log n)$, as desired. ∎

5.3 Using the Interval Tree

Proposition 5.5 *Any diagonal of a pair in $\mathcal{E}$ can be computed in time $O(\log^2 n)$.*

Proof: Let (I_0, I_1) be any pair in $\mathcal{E}$. 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_1. 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 $(I_0', I_1) \in \mathcal{E}$. 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 $\mathcal{E}$. 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 5.2. Thus by Lemma 5.3 and Lemma 5.4, 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 5.6 *Any boring pair in $\mathcal{E}$ can be computed in time $O(\log^2 n)$.*

Proof: By Proposition 5.5 we can compute the diagonals in time $O(\log^2 n)$. But then we can apply Lemma 5.3 and Lemma 5.4 to get the ceiling in time $O(\log^2 n)$. ∎

For any opposing pair $\{v_0, v_1\}$, by $\mathtt{time}(\mathtt{mast}\{v_0, v_1\})$ we refer to the time it takes to compute $\mathtt{mast}\{v_0, v_1\}$ assuming that every value in the base of $\{v_0, v_1\}$ is available in time $O(\log n)$.

Corollary 5.7 *Any interesting leaf pair (I_0, I_1) in $\mathcal{E}$ can be computed in time $O(\log^2 n + \mathtt{time}(\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1)\})$.*

Proof: Again, the diagonals are computed by Proposition 5.5, and the floor follows by Lemma 5.2. 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 $\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1)\}$ can be computed directly. ∎

Summing up, we conclude:

Proposition 5.8 SPINES$\{S_0, S_1\}$ *(Procedure 3) can be computed in time $O((m_0 + m_1 + l \log n) \log^2 n + (\sum_{\{v_0, v_1\} \in F} \mathtt{time}(\mathtt{mast}\{v_0, v_1\}))$, 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 $F \subseteq \mathtt{V}(S_0) \times \mathtt{V}(S_1)$ contains at most l pairs.*

Proof: Above, we observed that, indeed, computing all the ceilings of pairs in $\mathcal{E}$ is sufficient for implementing SPINES; and now, from Proposition 5.1, Lemma 5.2, Corollary 5.6, and Corollary 5.7, together with the observation that there are at most l interesting leaf pairs, it follows that we can compute all ceilings and diagonals of the pairs in $\mathcal{E}$ within the desired time bound. ∎

itself, and given any node I in $\mathcal{I}_S$, then I is a leaf if it consists of a single vertex; otherwise I has children I^l and I^r. $\mathcal{I}_S$ has depth $\lceil \log_2 |\mathtt{V}(S)| \rceil$. Denote by $\mathcal{I}$ the forest of the $\mathcal{I}_S$s. Now the spines of $\mathcal{T}$ constitute the top level of $\mathcal{I}$. Generally we have that all intervals at any specific level are mutually disjoint.

Our goal is to find an $O(n \log n)$ preprocessing such that given any vertex from $\mathcal{T}_0$ and interval I from $\mathcal{I}$, we can derive $\mathtt{side\text{-}mast}(v, I)$ in time $O(\log n)$. Assume that this is done. Then any interval I from $\mathcal{T}_1$ is the concatenation $I_1 \cdots I_l$ of at most $2 \log n$ intervals from $\mathcal{I}$, and then $\mathtt{side\text{-}mast}(v, I) = \max_i\{\mathtt{side\text{-}mast}(v, I_i)\}$. Thus such a preprocessing allows us to compute $\mathtt{side\text{-}mast}$ in time $O(\log^2 n)$, as desired.

For $i = 0, 1$ and for every label a, let $\mathtt{sr}_i(a)$ denote the root of the side tree of $\mathcal{T}_i$ containing the leaf with label a. By contraction of the side trees, we pre-compute all values of $\mathtt{sr}$ in time $O(n)$.

The remaining preprocessing is divided into $O(\log n)$ separate $O(n)$ preprocessings: one for each level in $\mathcal{I}$. Let $I_1, \ldots, I_k$ be all the intervals at some specific level in $\mathcal{I}$. Then $I_1, \ldots, I_k$ are mutually disjoint, so, in particular $\mathtt{A}(\mathtt{SideT}(I_1)), \ldots, \mathtt{A}(\mathtt{SideT}(I_k))$ are mutually disjoint.

Consider an arbitrary fixed interval I of $\mathcal{T}_0$, and note the following recursion formula for $\mathtt{side\text{-}mast}(\cdot, I)$:

$$\begin{aligned}\mathtt{side\text{-}mast}(v, I) = \max\{&\\ &\max\{\mathtt{mast}(v, \mathtt{r}(t)) | t \in \mathtt{SideT}(I)\}, &(9)\\ &\max\{\mathtt{side\text{-}mast}(w, I) | w \text{ is a descendant } v\}\} &(10)\end{aligned}$$

Set $V_I = \{\mathtt{p}(\mathtt{sr}_0(a)) | a \in \mathtt{A}(\mathtt{SideT}(I))\}$ and $W_I = \mathtt{LCA}(V_I)$. Then V_I contains all core vertices v for which (9) is relevant. Moreover, W_I contains all the values of w that are relevant for (10). Let t_I denote the topological subtree of $\mathcal{T}_0$ with vertex W_I. All the sets V_{I_i} are easily computed in time $O(n)$, and by Lemma 4.3, in time $O(n)$ we can compute all the t_{I_i}s.

Consider any specific t_{I_i}. We want to compute $\mathtt{side\text{-}mast}(v, I_i)$ for all $v \in \mathtt{V}(t_{I_i}) = W_{I_i}$. For this purpose, we introduce an variable $sm(v)$ for each $v \in W_{I_i}$. Initially $sm(v) := 0$ for all $v \in W_{I_i}$. Now, corresponding to (9), for all $a \in \mathtt{A}(\mathtt{SideT}(I_i))$, set

$$sm(\mathtt{p}(\mathtt{sr}_0(a))) := \max\{sm(\mathtt{p}(\mathtt{sr}_0(a))), \mathtt{mast}\{\mathtt{p}(\mathtt{sr}_0(a))), \mathtt{sr}_1(a)\}.$$

Next, corresponding to (10), go through the vertices of t_{I_i} in postorder. When visiting the vertex v, set

$$v := \max\{sm(v), \max\{sm(w) | w \text{ is a child of } v \text{ in } t_{I_i}\}\}$$

When the computation is finished $sm(v) = \mathtt{side\text{-}mast}(v, I_i)$ for all $v \in \mathtt{V}(t_{I_i}) = W_{I_i}$. The time of the computation is $O(|\mathtt{A}(\mathtt{SideT}(I_i))| + |W_{I_i}|) = O(|\mathtt{A}(\mathtt{SideT}(I_i))|)$. Thus we can pre-compute $\mathtt{side\text{-}mast}(v, I_i)$ for all $I_i, w \in W_{I_i}$ in time $O(\sum |\mathtt{A}(\mathtt{SideT}(I_i))|) = O(n)$.

Now, consider any query $\mathtt{side\text{-}mast}(v, I_i)$ where $v \notin W_{I_i}$. Then $\mathtt{side\text{-}mast}(v, I) = \mathtt{side\text{-}mast}(w, I)$ where w is the nearest descendant of v in W_{I_i}, if any, otherwise $\mathtt{side\text{-}mast}(v, I) = 0$. We solve the nearest descendant query in $O(\log n)$ time by first applying the preprocessing from Lemma 4.4 to all the W_{I_i}s. This preprocessing takes time $O(\sum_{I_i} |W_{I_i}|)$ where

$$\sum_{I_i} |W_{I_i}| < 2 \sum_{I_i} |V_{I_i}| \leq 2 \sum_{I_i} |\mathtt{A}(\mathtt{SideT}(I_i))| = 2n.$$

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, $\texttt{mast}\{\texttt{r}(I_0), \texttt{r}(I_1)\} \geq \max\{\texttt{mast}\{\texttt{c}(I_0), \texttt{r}(I_1\}, \texttt{mast}\{\texttt{c}(I_1), \texttt{r}(I_0)\}\}$. For $i = 0, 1$, let $t_{\bar{i}}$ be a side tree in $\texttt{SideT}(I_{\bar{i}})$ such that $\texttt{mast}\{\texttt{c}(I_i), \texttt{r}(t_{\bar{i}})\} = \texttt{side-mast}\{\texttt{c}(I_i), I_{\bar{i}}\}$. Let p_i and c_i denote the parent and core sibling of $\texttt{r}(t_{\bar{i}})$. By application of Lemma 3.1, we get the following inequalities:

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

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

We show that $lhs \leq rhs$ by contradiction. Assume there is a pair $(f_0, f_1) \in \texttt{V}(I_0) \times \texttt{V}(I_1)$ such that $\texttt{mast}\{f_0, f_1\} > rhs$, and fix (f_0, f_1) to be a minimal such pair. First, we observe that the strict inequality implies that $s_0 \neq \texttt{c}(I_0)$ and $s_1 \neq \texttt{c}(I_1)$. 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 $\texttt{mast}\{f_i, f_{\bar{i}}\} = \texttt{mast}\{c_i, f_{\bar{i}}\}$. Also by minimality, we cannot have $\texttt{mast}\{f_i, f_{\bar{i}}\} = \texttt{mast}\{s_i, f_{\bar{i}}\}$, where s_i is a side child of f_i, for since our pair of intervals is boring, $\texttt{mast}\{s_i, f_{\bar{i}}\} = \texttt{mast}\{s_i, c_{\bar{i}}\} \leq \texttt{mast}\{f_i, c_{\bar{i}}\}$. Thus, by Lemma 3.1, we have $\texttt{mast}\{f_0, f_1\} = \texttt{match}\{f_0, f_1\}$. Let M be a minimal matching in $\texttt{C}(f_0) \times \texttt{C}(f_1)$ such that $\texttt{match}\{f_0, f_1\} = \sum_{(v_0, v_1) \in M} \texttt{mast}\{v_0, v_1\}$. Since our interval pair is boring, 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:

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

and hence the desired contradiction. ∎

What makes this useful is the following technical lemma:

Lemma 5.4 *After an $O(n \log n)$ 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 $\texttt{side-mast}\{v, I\}$ in time $O(\log^2 n)$.*

Proof: For simplicity, we assume that v is from $\mathcal{T}_0$ and I is from $\mathcal{T}_1$. A symmetric preprocessing is needed for the opposite case where I is from $\mathcal{T}_0$ and v is from $\mathcal{T}_1$. First for each spine S in $\mathcal{T}_1$ we construct a balanced binary tree $\mathcal{I}_S$ over some non-empty intervals of S. The root of $\mathcal{I}_S$ is S

satisfied, in time $O(|\mathtt{A}(\mathtt{SideT}(I_0)) \cap \mathtt{A}(\mathtt{SideT}(I_1)|)$, we partition $\mathtt{A}(\mathtt{SideT}(I_0)) \cap \mathtt{A}(\mathtt{SideT}(I_1)$ into $\mathtt{A}(\mathtt{SideT}(I_0^l)) \cap \mathtt{A}(\mathtt{SideT}(I_1^l)$, $\mathtt{A}(\mathtt{SideT}(I_0^r) \cap \mathtt{A}(\mathtt{SideT}(I_1^l)$, and $\mathtt{A}(\mathtt{SideT}(I_0^r)) \cap \mathtt{A}(\mathtt{SideT}(I_1^r)$ for the children. For each of the $O(\log n)$ levels, the total size of these sets is no more than l, so the construction time is $O(l \log n)$.

Concerning the internal nodes satisfying (ii), ignoring the symmetric case, we restrict (ii) to:

(ii)$_0$ $\mathtt{r}(I_0) = \mathtt{r}(S_0)$ and I_1 contains more than one vertex.

Consider a node (I_0, I_1) of $\mathcal{E}$ satisfying (ii)$_0$. Then, due to the first part of the condition, the only children of (I_0, I_1) that can satisfy (ii)$_0$ are (I_0^r, I_1^l) and (I_0^r, I_1^r). Since there is no freedom in choosing first coordinate of the children, the number of internal nodes in $\mathcal{E}$ satisfying (ii)$_0$ is

$$\begin{aligned} & |\{S_1^\alpha : \alpha \in \{l,r\}^*, |\mathtt{V}(S_1^\alpha)| > 1\}| \\ < \;& |\{S_1^\alpha : \alpha \in \{l,r\}^*, |\mathtt{V}(S_1^\alpha)| = 1\}| \\ = \;& |\mathtt{V}(S_1)| \;\leq\; |\mathtt{A}(\mathtt{SideT}(S_1))| \;=\; m_1. \end{aligned}$$

The first inequality follows from the fact that the number of internal nodes of a binary tree is smaller than the number of leaves.

Equivalently for the symmetric case (ii)$_1$ where $\mathtt{r}(I_1) = \mathtt{r}(S_1)$ and I_0 contains more than one vertex, we get at most m_0 internal nodes in $\mathcal{E}$. Thus, there are at most $m_0 + m_1$ internal nodes in $\mathcal{E}$ satisfying (ii) [$=$ (ii)$_0 \vee$ (ii)$_1$], so we conclude that in total there are at most $O(m_0 + m_1 + l \log n)$ nodes in $\mathcal{E}$. Moreover, $\mathcal{E}$ can be generated from the root (S_0, S_1), adding each new node in constant time. ■

We are going to compute the ceiling and the diagonals of all the pairs in $\mathcal{E}$ respecting the linear order $<$ such that for each internal pair (I_0, I_1) of $\mathcal{E}$, 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 5.2 *For any internal pair of $\mathcal{E}$, the ceiling and diagonals are ceiling and diagonals of its children. The floor of any pair in $\mathcal{E}$ is the ceiling or diagonal of some preceding pair.* ■

So we need only concern ourselves with computations at the leaves, which are either boring interval pairs, or interesting node pairs.

5.2 Handling Boring Interval Pairs

Lemma 5.3 *If the interval pair $\{I_0, I_1\}$ is boring, then*

$$\mathtt{mast}\{\mathtt{r}(I_0), \mathtt{r}(I_1)\} = \max \left\{ \begin{array}{l} \mathtt{mast}\{\mathtt{c}(I_0), \mathtt{r}(I_1)\}, \mathtt{mast}\{\mathtt{c}(I_1), \mathtt{r}(I_0)\}, \\ \mathtt{side\text{-}mast}\{\mathtt{c}(I_0), I_1\} + \mathtt{side\text{-}mast}\{\mathtt{c}(I_1), I_0\} \end{array} \right\}.$$

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

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 representing all computations needed to implement SPINES$\{S_0, S_1\}$.

Our algorithm for SPINES will proceed as follows. In Subsection 5.1, we will choose a small subset of interval pairs on which to compute certain values. We will organize these intervals into an *interval tree.* In Subsection 5.2, we will prove the main technical lemmas needed for dealing with boring interval pairs. In Subsection 5.3, we will show how to actually implement SPINES in terms of the interval tree.

5.1 The Interval Tree

Consider an interval $I =]v, w]$, and let u be the vertex such that $|\mathtt{V}(]u, w])| = \lceil |\mathtt{V}(I)|/2 \rceil$. Then I^l denotes $]v, u]$ and I^r denotes $]u, w]$. Note that $\mathtt{V}(I^l)$ and $\mathtt{V}(I^r)$ are disjoint. Also note that if $I =]v, v]$ then $I^l = I^r = I$.

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 $\{\mathtt{V}(S_0), \mathtt{V}(S_1)\}$.

We are now going to construct a rooted tree $\mathcal{E}$ whose nodes are opposing interval pairs. The root pair is (S_0, S_1). Suppose (I_0, I_1) is a node in $\mathcal{E}$ and that one of the following conditions is satisfied:

(i) $\{I_0, I_1\}$ is interesting and one of I_0 and I_1 contains more than one vertex.

(ii) One of I_0 and I_1 has the spine root as root, and the other contains more than one vertex.

Then $\{I_0, I_1\}$ has four children: $(I_0^l, I_1^l), (I_0^l, I_1^r), (I_0^r, I_1^l), (I_0^r, I_1^r)$; otherwise $\{I_0, I_1\}$ is a leaf.

We are going to compute the ceiling and the diagonals of all the pairs in $\mathcal{E}$. Clearly, we will thereby implement 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 $\mathcal{E}$ ensures that $\mathtt{mast}\{r, v\}$ is the ceiling of some pair in $\mathcal{E}$.

Proposition 5.1 *The size and construction time for $\mathcal{E}$ is $O(m_0 + m_1 + l \log n)$*

Proof: In the argument, we will focus on the size of $\mathcal{E}$. However, a corresponding construction will be indicated on the side. The depth of $\mathcal{E}$ is no more than $\max\{\lceil \log_2 |\mathtt{V}(S_0)| \rceil, \lceil \log_2 |\mathtt{V}(S_1)| \rceil\}$. Clearly, we only need to bound the number of internal nodes, since the number of leaves is less than a factor four larger. We will separately count the number of nodes made internal by the two conditions. Thereby, we accept a certain overlap where a node $\{I_0, I_1\}$ satisfies both conditions.

Concerning condition (i), by induction starting at the root, for each $a \in \mathtt{A}(\mathtt{SideT}(S_0)) \cap \mathtt{A}(\mathtt{SideT}(S_1))$, there is exactly one node $\{I_0, I_1\}$ at each level in $\mathcal{E}$ such that $a \in \mathtt{A}(\mathtt{SideT}(I_0)) \cap \mathtt{A}(\mathtt{SideT}(I_1))$. Thus, at each level, we have at most $l = |\mathtt{A}(\mathtt{SideT}(S_0)) \cap \mathtt{A}(\mathtt{SideT}(S_1))|$ satisfying condition (i), so in total, we have at most $l \log n$ nodes in $\mathcal{E}$ satisfying (i). In order to find these nodes, starting from the root, with each node $\{I_0, I_1\}$ we store the set $\mathtt{A}(\mathtt{SideT}(I_0)) \cap \mathtt{A}(\mathtt{SideT}(I_1))$—$\{I_0, I_1\}$ is interesting if the set is non-empty. If condition (i) is

and hence

$$\begin{aligned}\kappa T(x/\kappa) &\leq \kappa 2b_2k_1k_2c^{\sqrt{\log(x/\kappa)}}C(x/\kappa) \leq \kappa 2b_2k_1k_2c^{\sqrt{\log x}-1}(x/\kappa)f(x/\kappa)\\ &\leq 2b_2k_1k_2(c^{\sqrt{\log x}}/c)xf(x) \leq 2b_2k_1k_2c^{\sqrt{\log x}}C(x)/c \leq T(x)/4.\end{aligned}$$

Thus

$$k_1C(\kappa^a x) + 2\kappa T(x/\kappa) \leq T(x)/2 + 2T(x)/4 = T(x),$$

so (6) and hence (4) is satisfied. We may therefore conclude that with $\varepsilon = 0$, we can compute TREES on a problem of size n in time $2b_2k_1k_2c^{\sqrt{\log n}}C(n) = O(C(n)c^{\sqrt{\log n}}) = O(n^{1+o(1)})$, as desired.

$\underline{\varepsilon > 0:}$ In this case, we fix $\kappa = \max\{4^{1/\varepsilon}, \sqrt[a]{b_1}\}$ and set $n_0 = \kappa b_1$. For a solution to our problem, we define T such that

$$\forall x \in \mathbb{R}_{\geq 1} : T(x) = k_1k_3C(x), \text{ where } k_3 = \max\{1, 2b_2C(\kappa^a)\}. \tag{8}$$

Clearly (3) and (5) are satisfied. Fix $x > n_0$. Now,

$$k_1C(\kappa^a x) \leq b_2C(\kappa^a)k_1C(x) \leq T(x)/2$$

and

$$\kappa T(n/\kappa) = \kappa b_2k_3(n/\kappa)^{1+\varepsilon}f(x/\kappa) \leq b_2k_3n^{1+\varepsilon}\kappa^{-\varepsilon}f(x) \leq b_2k_3n^{1+\varepsilon}f(x)/4 = T(x)/4,$$

so

$$k_1C(\kappa^a x) + 2\kappa T(x/\kappa) \leq T(x)/2 + 2T(x)/4 = T(x),$$

Thus (6) and hence (4) is satisfied. We may therefore conclude that with $\varepsilon > 0$, we can compute TREES on a problem of size n in time $b_2k_1k_3C(n) = O(C(n))$, completing the proof. ∎

5 Computing SPINES

To implement `MAST` for bounded degree trees, we need only show how to quickly compute SPINES (Procedure 3). To this end, we introduce the concept of intervals. For spine S with c the critical child of the lowest vertex, let $v, w \in \mathtt{V}(S) \cup \{c\}$, where w is an ancestor of v. Then v and w characterize an *interval* I of S, denoted by $]v, w]$. If $I =]v, w]$, we set $\mathtt{c}(I) = v$ and $\mathtt{r}(I) = w$ By $\mathtt{V}(I)$ we denotes the set of vertices that are descendants of w and strict ancestors of v. If $v \neq w$, $\mathtt{V}(I)$ induces a segment of S, and we will often identify I with this segment. If $v = w$, the interval I corresponds to the empty segment, together with a position. For any interval I, we define $\mathtt{SideT}(I)$ to be the forest of side trees whose roots are children of vertices in I. Let I_0 and I_1 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*. We will show how to implement SPINES quickly by giving an efficient method for dealing with boring interval pairs.

For specificity, fix $\{S_0, S_1\}$ to be a pair of opposing spines for which we wish to compute SPINES$\{S_0, S_1\}$. To ease the presentation, we will identify S_0 and S_1 with the intervals with the same

$$\forall n \in \mathbb{N} : n_0 < n \Rightarrow \langle \forall \kappa \in \mathbb{R} : 1 \leq \kappa \leq n \Rightarrow \mathtt{time}(\textsc{Trees}(n)) \leq k_1 C(\kappa^a n) + 2 \max_{\substack{\sum n_i = n, \\ n_i \in \{1,\ldots,\lfloor n/\kappa \rfloor\}}} \sum \mathtt{time}(\textsc{Trees}(n_i)) \rangle. \tag{2}$$

Inductively from (1) and (2) it follows that the time complexity for TREES is bounded any monotone function $T : \mathbb{R}_{\geq 1} \to \mathbb{R}$ satisfying

$$\forall x \in \mathbb{R} : 1 \leq x \leq n_0 \Rightarrow T(x) \geq k_1 C(x) \tag{3}$$

$$\forall x \in \mathbb{R} : n_0 < x \Rightarrow \langle \exists \kappa \in \mathbb{R} : 1 \leq \kappa \leq x \wedge T(x) \geq k_1 C(\kappa^a x) + 2 \max_{\substack{\sum x_i = n, \\ 1 \leq x_i \leq n/\kappa}} \sum T(n_i) \rangle. \tag{4}$$

In our search for an adequate such function T, we will restrict ourselves to superlinear functions satisfying:

$$\exists \text{monotone } g : \mathbb{R}_{\geq 1} \to \mathbb{R}\ \forall x \in \mathbb{R}_{\geq 1} : T(x) = xg(x). \tag{5}$$

As a convenient consequence, for any $x, x_1, \ldots, x_l \in \mathbb{R}_{\geq 1}$ such that $\sum x_i = x$ and $1 \leq x_i \leq x/\kappa$, we have that $\sum T(x_i) = \sum (x_i g(x_i)) \leq \sum (x_i g(x/\kappa)) = xg(x/\kappa) = \kappa T(x/\kappa)$. Thus (4) follows if

$$\forall x \in \mathbb{R} : x > n_0 \Rightarrow \langle \exists \kappa \in \mathbb{R} : 1 \leq \kappa \leq x \wedge T(x) \geq k_1 C(\kappa^a x) + 2\kappa T(x/\kappa) \rangle. \tag{6}$$

The rest of the proof divides into cases depending on ε.

$\underline{\varepsilon = 0:}$ For this case we let n_0 be the least number such that $n_0 \geq b_1$, $4^a\sqrt{\log n_0} \geq b_1$, and $4\sqrt{\log n_0} \leq n_0$. The last inequality is satisfied if and only if $n_0 \geq 16$. Recall that our choice of n_0 affects k_1. Since $\varepsilon = 0$, $C(x) = O(x^{1+o(1)})$, so we may choose a constant $k_2 \geq (2b_2)^{-1}$ such that $\forall x \in \mathbb{R}_{\geq 1}$: $C(x) \leq k_2 x^{1.5}$. For a solution to our problem, we define T such that

$$\forall x \in \mathbb{R}_{\geq 1} : T(x) = 2b_2 k_1 k_2 c^{\sqrt{\log x}} C(x), \text{ where } c = \max\{8^a, 4\}. \tag{7}$$

Clearly (3) is satisfied since $\forall x \in \mathbb{R}_{\geq 1} : T(x) = 2b_2 k_1 k_2 c^{\sqrt{\log x}} C(x) \geq 2b_2 k_1 (2b_2)^{-1} C(x) = k_1 C(x)$. Also (5) is satisfied, for the function that maps x to $T(x)/x = 2b_2 k_1 k_2 c^{\sqrt{\log x}} C(x)/x = 2b_2 k_1 k_2 c^{\sqrt{\log x}} x f(x)/x = 2b_2 k_1 k_2 c^{\sqrt{\log x}} f(x)$ is monotone since f is monotone. Hence (4) follows if we can settle (6). Fix $x > n_0$ and set $\kappa = 4^{\sqrt{\log x}}$. Notice that $1 < \kappa < x$, $\kappa^a > b_1$, and $x > b_1$. Now

$$\begin{aligned} k_1 C(\kappa^a x) &\leq k_1 b_2 C(\kappa^a) C(x) \leq b_2 k_1 k_2 (\kappa^a)^{1.5} C(x) = b_2 k_1 k_2 4^{1.5a\sqrt{\log x}} C(x) \\ &\leq b_2 k_1 k_2 c^{\sqrt{\log x}} C(x) \leq T(x)/2. \end{aligned}$$

Moreover,

$$\sqrt{\log(x/\kappa)} = \sqrt{\log(x/4^{\sqrt{\log x}})} = \sqrt{\log x - 2\sqrt{\log x}} \leq \sqrt{\log x} - 1,$$

Proof: Lemma 4.3 allows us to compute all $\{s, t_s\}$ pairs in $O(n)$ time. Clearly, $\sum_s |\mathtt{A}(s)| = n$, and for each s, $|\mathtt{A}(s)| < n/\kappa$. We can compute TREES$\{s, t_s\}$ for all s in time bounded by

$$2 \max_{\substack{\sum n_i = n, \\ n_i \in \{1, \ldots, \lfloor n/\kappa \rfloor\}}} \sum \mathtt{time}(\textsc{Trees}(n_i)).$$

By Lemma 4.2, we are now done with the side values. Concerning the base of the core trees, now preprocess for any queries of the form $\mathtt{mast}\{\mathtt{r}(s), v\}$ where s is a side tree and v is any opposing vertex. For simplicity, we assume that s is a side tree of $\mathcal{T}_0$ and v is a vertex of $\mathcal{T}_1$. The case where s is a side tree of $\mathcal{T}_1$ and v is a vertex of $\mathcal{T}_0$ is symmetric.

By Lemma 4.1, our problem is to decide if v has a descendant in t_s, and, if so, to return the first such descendant of v in t_s. Such a query can be answered in time $O(\log |\mathtt{V}(t_s)|) = O(\log n)$, if first we apply the preprocessing of Lemma 4.4 to all the sets $\mathtt{V}(t_s)$. The time for this preprocessing is

$$O(|\mathtt{V}(\mathcal{T}_0)| + \sum_s |\mathtt{V}(t_s)|) = O(n + \sum_s (2|\mathtt{A}(s)| - 1)) = O(n).$$

Thus all the preprocessing is completed within the desired bounds. ■

4.1 A Master Theorem

The following theorem 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. Our goal is to make our result apply, even if the complexity of maximum weighted matching is improved. So we must allow our recurrence to hold for a wide possible set of choices for the complexity of maximum weighted matching. While general theorems have been proven for solving recurrences (see e.g. Verma [23]), we know of no results that directly apply. Thus we offer the following "master theorem" for solving the types of recurrence we need.

Theorem 4.6 *Assume for some monotone function $C : \mathbb{R}_{\geq 1} \to \mathbb{R}$ that* 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 $\varepsilon = 0$, there is a constant c such that we can compute* MAST *in time $O(C(n) c^{\sqrt{\log n}})$. Otherwise, if $\varepsilon > 0$, setting $\kappa = 4^{\sqrt{\log n}}$, we can compute* MAST *in time $O(C(n))$. Thus* MAST *is computable in time $O(n^{1+o(1)} + C(n))$*

Proof: By Proposition 4.5 and the definition of C, using Algorithm B, we compute TREES in time

$$O(n) + 2 \max_{\substack{\sum n_i = n, \\ n_i \in \{1, \ldots, \lfloor n/\kappa \rfloor\}}} \sum \mathtt{time}(\textsc{Trees}(n_i)) + C(k^a n).$$

Thus, for any constant $n_0 \geq 1$ (to be fixed later), we may choose a constant k_1 such that

$$\forall n \in \mathbb{N} : 1 \leq n \leq n_0 \Rightarrow \mathtt{time}(\textsc{Trees}(n)) \leq k_1 C(n) \tag{1}$$

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. Call these missing values the *side values*. In order to make Algorithm B implement TREES completely, we extend our specification of CORE-BASE to compute the side values as well. That is, CORE-BASE should make retrievable not only the values from the base of the core trees but also these side values. However, the following trivial lemma shows that this is already done by our recursion over the side trees:

Lemma 4.2 *For any $v \in \mathtt{V}(s)$, where s is a side tree of $\mathcal{T}_i$, $t_s = \mathcal{T}_{\bar{i}}|\mathtt{A}(s)$, $\mathtt{mast}\{v, \mathtt{r}(\mathcal{T}_{\bar{i}})\} = \mathtt{mast}\{v, \mathtt{r}(t_s)\}$.* ∎

We have now shown that we can compute all values needed for CORE-BASE by recursively applying TREES on all side trees s, along with their opposing restrictions t_s. So the remaining questions is how do we implement the various steps described in an efficient manner. First, we need to compute the t_s. The following lemma states that all t_s can be computed in linear time, since the label set of different side trees of the same evolutionary tree are disjoint.

Lemma 4.3 ([7]) *Let T be a rooted tree with vertex set V, and let $\{V_1, \ldots, V_k\}$ be a family of subsets of V. Then, in time $O(\sum |V_i| + |V|)$, we can compute all the topological subtrees T_i with $\mathtt{V}(T_i) = \mathtt{LCA}(V_i)$.*

Thus, given a partition $L_0, \ldots, L_k$ of the labels of an evolutionary tree T of size n, we can compute all of $T|L_0, \ldots, T|L_k$ in total $O(n)$ time.

Now, in order to implement the descendant operation from Lemma 4.1, we need the following technical lemma:

Lemma 4.4 *Let T be a rooted tree with vertex set V, and let $\{W_1, \ldots, W_k\}$ be a family of subsets of V, each of which is closed under least common ancestors. Then, in $O(\sum |W_i| + |V|)$ time, we can build a data structure such that, given any vertex $v \in V$ and index $i \leq k$, we can return the nearest descendant of v in W_i, if any, in time $O(\log |W_i|)$.*

Proof: First we organize the vertices of T in an Euler Tour E, that is, we make a depth first traversal from the root noting each time we visit every vertex. For every vertex v, denote the first occurrence in E by $f(v)$ and the last occurrence in E by $l(v)$. Now w is a descendant of v if and only if $f(v) \leq f(w) < l(v)$. Next, for $i = 1, \ldots, k$, we construct the subsequence E_i^f of E containing $f(w)$ for each vertex w in W_i. Clearly, both the Euler tour and the splitting of it into the E_i^fs can all be done in time $O(\sum |W_i| + |V|)$.

Now, given a vertex $v \in V$ and index $i \leq k$, by an $O(\log |E_i^f|) = O(\log |W_i|)$ time binary search, we find the first element $f(w)$ in E_i^f greater than or equal to $f(v)$. If $f(w) \geq l(v)$ we may conclude that v has no descendant in t_s. Otherwise $f(w) < l(v)$. Since W_i is closed under the least common ancestor operation, we may then conclude that w is the unique first descendant v in W_i. ∎

Summing up, we have

Proposition 4.5 CORE-BASE *can be computed in time*

$$O(n) + 2 \max_{\substack{\sum n_i = n, \\ n_i \in \{1, \ldots, \lfloor n/\kappa \rfloor\}}} \sum \mathtt{time}(\textsc{Trees}(n_i)).$$

C.1. For $i \in \{0,1\}$, let $\mathcal{U}_i'$ denote the tree obtained by identifying each spine of $\mathcal{U}_i$ with a single vertex.

C.2. Let $\mathcal{O}$ be the lexicographic ordering of $\mathtt{V}(\mathcal{U}_0') \times \mathtt{V}(\mathcal{U}_1')$ where the vertices in each $\mathcal{U}_i'$ are postordered.

C.3. For each (c_0, c_1) in increasing order in $\mathcal{O}$ do

C.3.1. Case: c_0 and c_1 are critical. Do

C.3.1.1. Compute $\mathtt{mast}\{c_0, c_1,\}$.

C.3.2. Case: c_i is a spine node and $c_{\bar{i}}$ is a critical node, for $i \in \{0,1\}$. Do:

C.3.2.1. Let $s_1, \ldots, s_k$ be the spines nodes of c_i in ascending order.

C.3.2.2. For $j \leftarrow 1$ to k compute $\mathtt{mast}\{c_{\bar{i}}, s_j\}$.

C.3.3. Case: c_0 and c_1 are spine representing nodes. Do:

C.3.3.1. Compute SPINES$\{c_0, c_1\}$.

Observation 3.3 CORE-TREES *makes* $O(\kappa n)$ *direct* mast*-computations (Steps C.3.1.1 and C.3.2.2) and calls* SPINES $O(\kappa^2)$ *times (Step C.3.3.1). All other processing is done in time* $O(\kappa n)$.

4 Computing CORE-BASE

The goal of CORE-BASE is to preprocess $\mathcal{T}_0$ and $\mathcal{T}_1$ so that we may quickly retrieve values from the base of the core trees, that is, mast-values of opposing vertex pairs where one is the root of a side tree and the other is either the root of a side tree or a core vertex. This will be done recursively using the following extension of MAST, which for both two roots, computes the mast-value against all opposing vertices:

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

Assuming an appropriate implementation of TREES, we will apply it to each side tree s against the opposing tree restricted to the label set of s. Thus all problems considered are of size $O(n/\kappa)$. The following lemma shows that such a computation for all side trees essentially gives us the whole base of the core trees, or even more: for each root of a side tree, it allows us to derive the mast-value against any opposing vertex.

Lemma 4.1 *Let* s *be a side tree of* $\mathcal{T}_i$, $t_s = \mathcal{T}_{\bar{i}}|\mathtt{A}(s)$, *and* $W = \mathtt{V}(t_s)$. *Then for all* $v \in \mathtt{V}(\mathcal{T}_{\bar{i}})$, $\mathtt{mast}\{\mathtt{r}(s), v\} = 0$, *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.

Proof: Set $B = \mathtt{A}(s) \cap A(\mathtt{t}(v))$. Then $\mathtt{V}(\mathcal{T}_{\bar{i}}|B)$ is exactly the set of descendants of v in $W = \mathtt{V}(t_s) = \mathtt{V}(\mathcal{T}_{\bar{i}}|\mathtt{A}(s))$. By definition $\mathtt{mast}\{\mathtt{r}(s), v\} = \mathtt{MAST}\{\mathcal{T}_0|B, \mathcal{T}_1|B\}$. Thus $\mathtt{mast}\{\mathtt{r}(s), v\} = 0$ if $B = \emptyset$, but then v has no descendants in W. We may therefore assume that $B \neq \emptyset$. Then $\mathtt{r}(\mathcal{T}_{\bar{i}}|B)$ is the first descendant of v in W. Moreover, $\mathtt{A}(\mathtt{t}(v)) \supseteq \mathtt{A}(\mathtt{t}(\mathtt{r}(\mathcal{T}_{\bar{i}}|B))) \supseteq B$, so $\mathtt{A}(s) \cap \mathtt{A}(\mathtt{t}(\mathtt{r}(\mathcal{T}_{\bar{i}}|B))) = B$, and hence $\mathtt{mast}\{\mathtt{r}(s), v)\} = \mathtt{mast}\{\mathtt{r}(s), \mathtt{r}(\mathcal{T}_{\bar{i}}|B)\}$. This completes the proof. ∎

root of a side tree and the other is either the root of a side tree or a core vertex. Second we concentrate on computing mast-values for opposing pairs of core nodes, including the root pair. For the base computation, we note that the base of the core trees is of size $\Theta(n^2)$, so we cannot afford to compute the whole base. Instead, we will build a data structure allowing us to retrieve any desired base value quickly. More specifically, we will implement:

Procedure 1 CORE-BASE*: Computes a data structure such that every* mast*-value in the base of the opposing core trees, i.e. of* $\{\mathtt{V}(\mathcal{U}_0), \mathtt{V}(\mathcal{U}_1)\}$*, can be determined in* $O(\log n)$ *time.*

This data structure will be computed by a routine which recurses on all side trees. Thus, within the recursion, all sub-problems considered are of size proportional to that of the side trees ($O(n/\kappa)$). Using this fact, we will show that for unbounded degree trees the recursion will have no asymptotic consequence for the overall computation time. More precisely, we will show that we have a dominating bottle-neck in the maximum weighted matching evaluations in computing match for opposing pairs of core vertices. For bounded degree trees, match takes constant time, and in this case, it will turn out that the recursion contributes to the overall running time by multiplicative factor of $O(c^{\sqrt{\log n}})$, for some constant c.

Second, we will find an efficient implementation of

Procedure 2 CORE-TREES*: Given the base of the opposing core trees (which we can compute as needed by the* CORE-BASE *data structure), this procedure computes* $\mathtt{mast}\{\mathtt{r}(\mathcal{U}_0), \mathtt{r}(\mathcal{U}_1)\} = \mathtt{mast}\{\mathtt{r}(\mathcal{T}_0), \mathtt{r}(\mathcal{T}_1)\} = \mathtt{MAST}(\mathcal{T}_0, \mathcal{T}_1)$.

This procedure will be implemented by a sparse version of the dynamic program described in Algorithm A. In $O(\kappa n \text{ polylog } n)$ time, it will select $O(\kappa n)$ significant mast-values to be computed, including, in particular, $\mathtt{mast}\{\mathtt{r}(\mathcal{U}_0), \mathtt{r}(\mathcal{U}_1)\}$. For bounded degrees each of these mast-values can be computed in constant time. For unbounded degrees, it will be shown that they can be computed in the same total time as that of **one** unary weighted bipartite matching of size $O(n)$. Thus, very generally, MAST is implemented as follows.

Algorithm B:

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

B.2. Find their core trees $\mathcal{U}_0$ and $\mathcal{U}_1$.

B.3. Compute CORE-BASE.

B.4. Compute CORE-TREES.

B.5. Return $\mathtt{mast}\{\mathtt{r}(\mathcal{T}_0), \mathtt{r}(\mathcal{T}_1)\}$.

As noted above, a naïve algorithm would simply apply Lemma 3.1 to each core-core pair. This would yield once again an $\Omega(n^2)$ algorithm, the bottleneck of which is in comparing spines. The following procedure will be used to circumvent this bottleneck. We will use it to complete a sketch of CORE-TREES.

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

With this procedure we get the following algorithm for CORE-TREES:

Algorithm C:

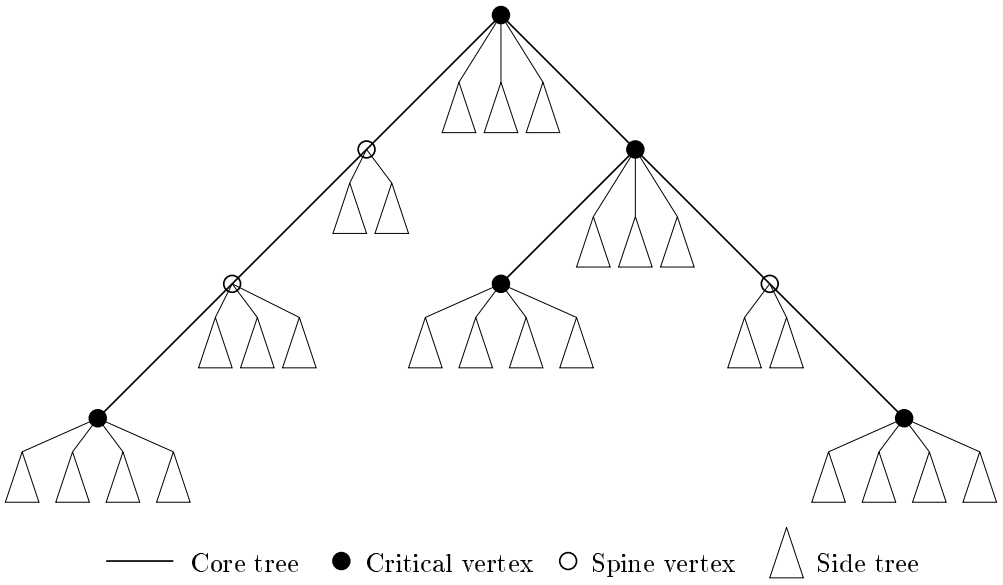

Figure 3.2: The core tree related concepts.

3.2 A Faster Algorithm

Our key to sparsifying the dynamic program is the *core trees*, which is defined in terms of some parameter κ (which will turn out to be 16 for unbounded degrees, and $4^{\sqrt{\log n}}$ for bounded degrees). 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 $\mathcal{U}_i$ 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 that 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 $\Omega(n)$ side children. We call each such component a *spine*. The concepts are illustrated at Figure 3.2. Note that we have $O(\kappa)$ critical nodes and spines, but $O(n)$ spine nodes. 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 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 $\sum_{i=1}^{x} \sum_{j=1}^{y} l_{ij} = n$.*

Given the core trees, we naturally divide the `MAST` computation into two phases. First we compute the base of the opposing core trees, that is, `mast`-values of opposing vertex pairs where one is the

The lemma is illustrated at Figure 3.1. It is clear from this lemma that we need some values on subtrees in order to compute the mast of two nodes. To simplify the discussion, we introduce the following notation. By the *base pairs* of an opposing vertex pair $\{x_0, x_1\}$, we understand the set of opposing pairs $\{w_0, w_1\}$ such that either $w_0 \in \mathtt{C}(v_0)$ and $w_1 \in \mathtt{C}(v_1) \cup \{v_1\}$, or conversely, $w_0 \in \mathtt{C}(v_0) \cup \{v_0\}$ and $w_1 \in \mathtt{C}(v_1)$. 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 $\{x_0, x_1\}$.

Later in the paper, we will need a generalized version of a base, defined over pairs of opposing *sets* of vertices. First, for any set V of vertices in a tree, 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. Now, if V_0 and V_1 are subsets of $\mathtt{V}(\mathcal{T}_0)$ and $\mathtt{V}(\mathcal{T}_1)$, then the *base pairs* of $\{V_0, V_1\}$ are the opposing vertex pairs $\{w_0, w_1\}$ such that either $w_0 \in \mathtt{C}(V_0)$ and $w_1 \in \mathtt{C}(V_1) \cup V_1$, or conversely, $w_0 \in \mathtt{C}(V_0) \cup V_0$ and $w_1 \in \mathtt{C}(V_1)$. By the *base* of $\{V_0, V_1\}$ we mean the values $\mathtt{mast}\{w_0, w_1\}$ for all base pairs of $\{V_0, V_1\}$. Thus the base of $\{V_0, V_1\}$ contains all the values needed for a dynamic bottom-up computation of $\mathtt{mast}\{v_0, v_1\}$, for all $v_0 \in V_0$ and $v_1 \in V_1$.

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

Algorithm A: First Algorithm for MAST.

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

A.2. Let $\mathcal{O}$ be the lexicographic ordering of $\mathtt{V}(\mathcal{T}_0) \times \mathtt{V}(\mathcal{T}_1)$, where the vertices in each $\mathcal{T}_i$ are post-ordered.

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

A.3.1. Compute $\mathtt{mast}\{v_0, v_1\}$.

A.4. Return $\mathtt{mast}\{\mathtt{r}(\mathcal{T}_0), \mathtt{r}(\mathcal{T}_1)\}$.

This first algorithm computes MAST by applying Lemma 3.1 to all the $O(n^2)$ pairs of opposing vertex pairs. The bottom-up ordering of $\mathcal{O}$ guarantees that the base of a node pair is ready whenever mast is evaluated. For bounded degree, mast can be computed in $O(1)$ time, 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 Gabow and Tarjan's matching algorithm [12]. Note that the Gabow/Tarjan algorithm is not only the fastest known for normal weighted bipartite matching where the weights are assumed to be in a binary encoding, it is also the fastest known for the case of unary weighted matching, i.e. the case where the input is measured as the sum of all the edge weights in the graph. It is in this later sense that we are interested in the Gabow/Tarjan algorithm.

In [7], 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 computations of mast, 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 by two types of sparsification. First of all we reduce the number of mast values evaluated in the dynamic program. Secondly, we reduce the work done in the matchings. This second phase is analogous to our previous work [7] in reducing the matching work, but here we require stronger techniques in order to achieve optimality in the sense of equivalence to **one** unary weighted bipartite matching. For the moment, we will focus entirely on reducing the number of comparisons, returning to weighted matchings in §6.

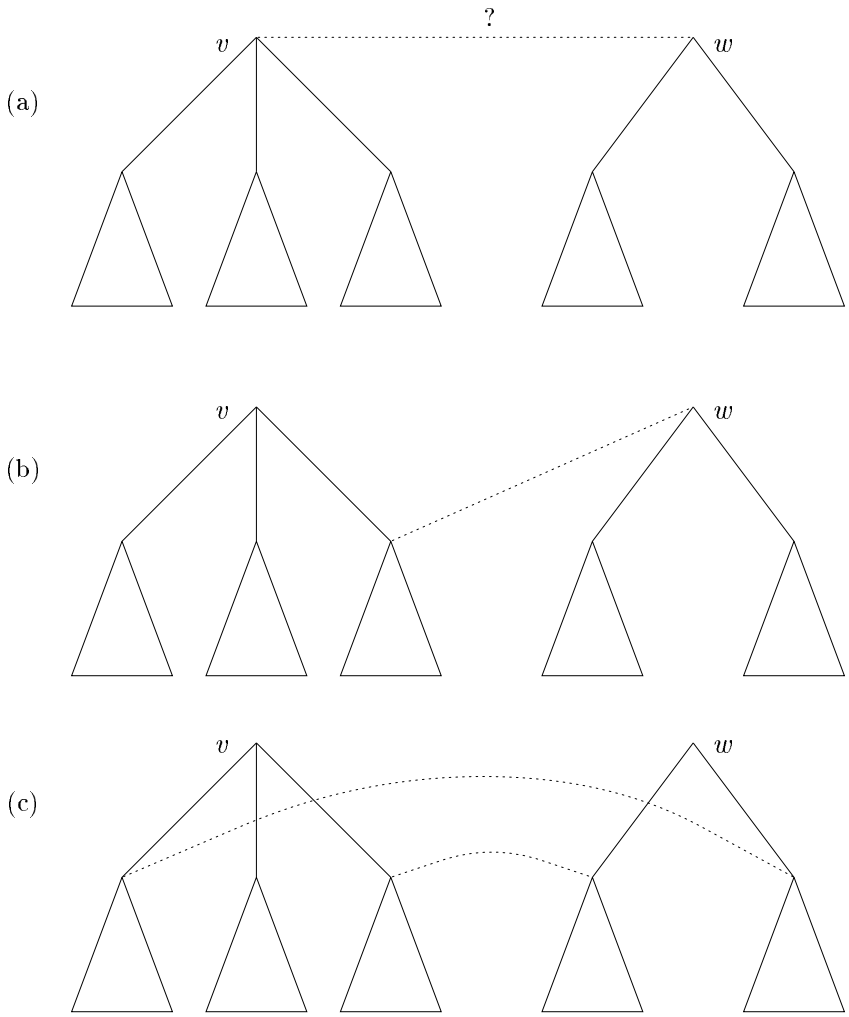

Figure 3.1: In order to find $\texttt{mast}\{v, w\}$ as in (a), we can either try a "diagonal," as in (b), or we can find a maximum weight matching between the sets of children, as in (c).

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

In [3] Amir and Keselman use a similar reduction from *Three-Dimensional Matching* to prove that the MAST problem on 3 or more trees is NP-Hard.

3 Ideas and Outline

3.1 Preliminaries

In this subsection, we will describe a rooted version of the SW algorithm. This algorithm will form the basis of our discussion of more efficient algorithms. For simplicity, all algorithms presented will compute only the maximal cardinality of an agreement set, i.e. the cardinality of the output set for the MAST-problem. However, after this cardinality has been found, the computation can easily be traced back in order to derive a concrete set. In this subsection, we will also introduce the main notation and terminology for the rest of the paper.

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 $\mathtt{p}(v)$ be the parent of v, $\mathtt{C}(v)$ be the set of its children, and $\mathtt{d}(v)$ 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.

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 this case $\mathtt{MAST}(\mathcal{T}_0, \mathcal{T}_1) = |\mathtt{A}| = n = 0$. Finally, fix an arbitrary left to the right ordering of the children of each node. Thus $\mathcal{T}_0$ and $\mathcal{T}_1$ are henceforth considered ordered.

The rôles of the evolutionary trees $\mathcal{T}_0$ and $\mathcal{T}_1$ are symmetric. In order to avoid unnecessary repetitions in our definitions, we will commonly use $\bar{0}$ to mean 1, and vice versa. Also, we introduce the generic term *opposing pair*, by which we refer to a pair $\{x, y\}$ where x is contained in $\mathcal{T}_i$ and y is contained in $\mathcal{T}_{\bar{i}}$. Here x and y might be of different types. For example, x might be a vertex of $\mathcal{T}_i$ while y is a subset of the vertices of $\mathcal{T}_{\bar{i}}$.

For any opposing pair $\{v, w\}$ of vertices, let $\mathtt{mast}\{v, w\}$ denote the MAST of the subtrees rooted respectively at v and w. Here the subtrees are understood to be topologically restricted to the intersection of their label sets. Thus, $\mathtt{mast}\{v, w\}$ is the MAST of $\mathcal{T}_0|B$ and $\mathcal{T}_1|B$ where $B = \mathtt{A}(\mathtt{t}(v)) \cap \mathtt{A}(\mathtt{t}(w))$.

The following lemma appears, with some minor modifications, in [21] and is the basis for their dynamic program approach to the unrooted version of this problem.

Lemma 3.1 ([21]) $\forall v \in V(\mathcal{T}_0), w \in V(\mathcal{T}_1)$,

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

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 where* $\mathtt{match}\{v, w\}$ *is the value of the maximum weight matching of the weighted bipartite graph* $((\mathtt{C}(v) \cup \mathtt{C}(w), \mathtt{C}(v) \times \mathtt{C}(w), \mathtt{mast}\{\cdot, \cdot\})$.

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 will turn out to be useful in guiding 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 Steel and Warnow paper [21]. The above cited complexities for their algorithms ($O(n^{4.5}\log n)$ for unbounded degrees and $O(n^2)$ for bounded degrees) apply to the unrooted case. However, they noticed the rooted case reduces to the unrooted case, and hence that these complexities carry over to the rooted case. In [7], our main result was to improve the complexity of the unbounded unrooted case to $O(n^2 c^{\sqrt{\log n}})$. We believe that the unrooted case is much harder than the rooted case and know of no $O(n^2)$ algorithm for this problem. Another variant is that of considering three or more trees. Amir and Keselman [3] showed that the MAST problem on three or more trees is *NP*-hard for trees of unbounded degree, while polynomial if just one of the trees has bounded degree. While the constant degree restriction may be suitable in some settings, the unbounded degree algorithm is important since there are many tree construction techniques available which place no restrictions on the degree of the trees produced [22].

Another problem related to our MAST problem is the tree homeomorphism problem. In [4], Chung presents an $O(n^{2.5})$ algorithm for the rooted tree homeomorphism problem of deciding whether one rooted tree (no leaf-labels) is a topological subtree of another. This algorithm is quite similar to the SW algorithm. Unfortunately, none of our optimizations to the SW algorithm apply to the tree homeomorphism problem since they all rely on the restriction that the agreement isomorphism should be leaf label preserving.

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 [12] 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 $G = (U \cup V, E, W : E \to \mathbb{N})$ 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 $W(e)$ leaf children to c_u in T_U and label them $< x, y, i >$, $1 \le i \le W(e)$. Add leaf 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 $\mathtt{A}(c_v) \cap \mathtt{A}(c_w) \subseteq B$.

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 [10] 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 [17], Kubicka *et al.* presented an $O(n^{(\frac{1}{2}+\epsilon)\log n})$ time algorithm for the binary MAST problem. Steel and Warnow [21] 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 [7] 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. To avoid the too costly quadratic complexity of these algorithms, researchers have either turned to heuristics (e.g. the BLAST program [2]) or to the design of *input sensitive* algorithms. Notable in the later class is the algorithm of Hunt and Szymanski [14]. This latter class of algorithms has the property that they speed up various dynamic programs to almost linear time in the *best case* but have at least quadratic time worst cases.

In this paper we use sparsity conditions to break the dynamic programming bottleneck and have sub-quadratic running times in the worst case. In fact, we show 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 (see [12] for definitions of matchings, etc.). 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 [12] 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. If the degrees are bounded, our algorithm runs in $O(nc^{\sqrt{\log n}}) = O(n^{1+o(1)})$ time, beating the previous $O(n^2)$ bound [21] for this case.

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 carefully balance the time spent at a single node pair in

[1] This complexity is understood to be modulo a class of "well-behaved" functions explicitly defined in Theorem 4.6.

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 [15, 24]. 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. [8, 9, 11, 19, 22, 20]). More recently, a variety of solutions have been examined rigorously ([1, 6, 16]). 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 [18]. In other settings, a particular method may be applied to different data sets for the same set of species [13] or on a single data set which has been permuted some number of times for statistical analysis [9]. The resulting trees are then compared in order to arrive at some consensus. Many consensus techniques have been proposed and are currently in use (see [5] for a review). One of the most extensively studied consensus methods was defined by Finden and Gordon [10] as follows.

Let A be a set of species. We will define an *evolutionary tree*, T, on A to be a rooted tree, with no degree 1 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 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?

Finden and Gordon's answer to this question involves the notion of a "topological restriction" of an evolutionary tree to a subset of the species. Given an evolutionary tree T on set A, and given $B \subseteq A$, then the *topological 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. More formally, first, given any rooted tree T, by a *topological subtree of T*, we mean a rooted tree U with no degree 1 nodes, obtained from a normal subtree S of T by replacing dipaths with single arcs. That is, U can be obtained from the subtree S by repetition of the following operation: if a vertex v has only one child w, we may delete ("jump") v, making the original parent of v the parent of w. Note that the topological subtree U is uniquely defined in terms of its leaf set. The full vertex set of U is the closure of the leaf set under the least common ancestor operation in T. Now, formally $T|B$ denotes the topological subtree of T whose leaves are the the leaves of T with labels in B. This restriction operator immediately implies the following similarity measure on trees.

Sparse Dynamic Programming for Evolutionary Tree Comparison*

Martin Farach†
Rutgers University

Mikkel Thorup‡
University of Copenhagen

January 9, 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 should be compared for consensus. It has become necessary to automate this process as the number of species under consideration has grown. We study one formalization of the problem: the *Maximum Agreement Subtree Problem* (`MAST`).

The `MAST` problem is 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 topological restrictions of $\mathcal{T}_0$ and $\mathcal{T}_1$ to B are isomorphic. In this paper, we will show that this problem reduces to Unary Weighted Bipartite Matching (`UWBM`) with an $O(n^{1+o(1)})$ additive overhead. We also show that `UWBM` reduces linearly to `MAST`. Thus, our algorithm is optimal unless `UWBM` can be solved in near-linear time. The overall running time of our algorithm is $O(n^{1.5}\log n)$, improving on the previous best algorithm which runs in $O(n^2)$. We also derive an $O(nc^{\sqrt{\log n}})$ time algorithm for the case of bounded degrees, where the previously best algorithm runs in $O(n^2)$, as in the unbounded case.

*A preliminary version of this paper appeared in the Foundations of Computer Science Conference, FOCS '94.

†`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.