The Maximum Agreement Subtree Problem for Binary Trees

Martin Farach*
Rutgers University

Teresa M. Przytycka†
Odense University

Mikkel Thorup‡
University of Copenhagen

February 1, 1995

Abstract

We consider the problem of computing the Maximum Agreement Subtree (a maximum common topological restriction) of two binary labeled trees. We show that the problem can be solved in $O(n \log^3 n)$ using a novel dynamic programming approach. This improves on the previous $O(nc^{\sqrt{\log n}})$-time algorithm. At the heart of our solution is an efficient algorithm for an independent problem which we call the Maximum Crossing (*MC*) problem, a problem similar in flavor to the Heaviest Common Subsequence problem.

1 Introduction

The Maximum Agreement Subtree arises in biology as a measure of consistency between two evolutionary trees. An *evolutionary tree* for a species set A is a rooted tree in which the leaves are uniquely labeled by the species in A, and the internal nodes represent ancestors. There are many methods for computing evolutionary trees [1, 5, 9, 11, 15, 17]. 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

*Department of Computer Science, Rutgers University, Piscataway, NJ 08855, USA. E-mail: farach@cs.rutgers.edu

†Department of Mathematics and Computer Science, Odense University, Campusvej 55, 5230 Odense M, Denmark. E-mail: przytyck@imada.ou.dk. Part of this work was done when this author was visiting DIMACS institute and a part when visiting the University of Warwick.

‡Department of Computer Science, University of Copenhagen, Universitetsparken 1, 2100 Kbh. Ø, Denmark. E-mail: mthorup@diku.dk.

different methods be applied to the data. The resulting trees are then compared in order to arrive at some consensus. One technique for obtaining such a consensus is to compute the so-called Maximum Agreement Subtree ($MAST$).

Define a *leaf labeled tree*, T, on label set L to be a rooted tree with no degree 1 nodes, such that the leaves of T are uniquely labeled with the elements of L. Thus evolutionary trees are leaf labeled trees, but we will use this less application specific name to emphasize that leaf labeled trees are used in many other settings for example to model clustering [4]. The set of leaf labels of a tree T is denoted $L(T)$. The *size*, n, of a tree is defined to be the number of leaves. Given a leaf labeled tree T on set L, and given $L' \subseteq L$, then the *topological restriction of T to L'*, written $T|L'$, is the tree with vertex set $\{\mathtt{lca}^T(a,b) | (a,b) \in L' \times L'\}$, where the arcs are defined such that for all $(a,b) \in L' \times L'$, $\mathtt{lca}^{T|L'}(a,b) = \mathtt{lca}^T(a,b)$. Here $\mathtt{lca}^T(x,y)$ is the least common ancestor of nodes x and y in T, for any tree T. Such a tree is uniquely determined by L'. More operationally, we get $T|L'$ from T by first removing all nodes without descendants in B, and then contracting any path of degree 1 nodes to a single edge. Given a set $L' \subseteq L(T)$ the tree $T|L'$ can be computed in $O(n)$ time [6].

Given two leaf labeled trees T_0, T_1 on the same set L, the *Maximum Agreement Subtree* problem ($MAST$) is to compute a maximum cardinality subset L' of L such that $T_0|L'$ and $T_1|L'$ are isomorphic. (Isomorphism of leaf labeled trees is assumed to preserve the labels.) Finden and Gordon [10] gave a heuristic method for computing the $MAST$ for two rooted binary trees. Their algorithm, which has a $O(n^5)$ running time, does not, however, guarantee an optimal solution. Subsequently, Kubicka *et al.* [16] presented an $O(n^{(\frac{1}{2}+\epsilon)\log n})$ time algorithm for the binary $MAST$ problem. The first polynomial time algorithm was given by Steel and Warnow [18]. (They also considered the unrooted version of the problem ($UMAST$), a problem which is also biologically important.) Their algorithm, which we will refer to as SW, is a dynamic programming approach which runs in $O(n^2)$ time on bounded degree unrooted trees and in $O(n^{4.5}\log n)$ time on unbounded degree unrooted trees. Moreover, they give a linear reduction from the rooted to the unrooted case, thus reporting the same bounds for the rooted case. Subsequently, Farach and Thorup showed an $O(n^2 c^{\sqrt{\log n}})$ time algorithm for the $UMAST$ problem [6] and an $O(n^{1.5}\log n)$ time algorithm for the rooted version of the problem [7]. The latter result is optimal in the sense that it is proved by giving a (slightly non-standard) reduction to unary weighted bipartite matching [12]. For binary trees, their algorithm runs in $O(nc^{\sqrt{\log n}})$ time.

In practice, evolutionary trees have small degree, and most frequently they are simply binary trees. Thus the MAST problem for binary trees is of particular interest. In this paper, we give an $O(n\log^3 n)$ time algorithm for the Maximum Agreement Subtree problem for binary trees.

As with the SW algorithm, our algorithm uses dynamic programming. The quadratic cost of the SW algorithm follows from the fact that it computes the local solution to the

MAST problem for every pair of rooted subtrees of the two input trees. All $o(n^2)$ dynamic programming algorithms for the *MAST* problem must find a sparse subset of significant local computations to perform. In [7], this was done implicitly by a fairly intricate recursive scheme specifically designed to give an optimal algorithm for the case of unbounded degrees. As mentioned above, for binary trees their recursion takes $O(nc^{\sqrt{\log n}})$ time. Our algorithm is a simple dynamic program tailored to binary trees. A key feature of our algorithm is that we can identify all the local computations initially before the dynamic program starts. We are thus able to radically simplify the method for actually computing the needed values, and therefore we have an algorithm which is both more efficient and much simpler. It solves the *MAST* problem for binary trees by performing $O(n \log^2 n)$ local computations in $O(n \log^3 n)$ total time. The technique presented in this paper can quite easily be generalized to give an $O(n\sqrt{d}\log^3 n)$ time algorithm for degree d bounded trees. However, for unbounded degrees, the technique of [7] is still preferable.

The main idea is to combine a decomposition of the trees into paths with dynamic programming algorithms for two other optimization problems: a variant of the Maximum Weight Common Subsequence Problem (which, because of the interpretation in terms of bipartite graphs used in this paper, we call the Maximum Non-Crossing Matching problem) and a problem that we call the Maximum Crossing problem. The definitions of these two problems are given later in this section.

The rest of the paper is organized as follows. In the following subsection, we define the Maximum Non-Crossing Matching and the Maximum Crossing Pair Problems. In the Section 2, we discuss the Maximum Crossing problem and give an efficient algorithm for solving it. In Section 3, we give our $O(n \log^3 n)$–time algorithm for the Maximum Agreement Subtree for binary trees. In the concluding remarks, we discuss possible generalizations to any bounded degree d.

1.1 Definitions of the Maximum Non-Crossing Matching and the Maximum Crossing Pair Problems

Both problems are defined in the terms of bipartite graphs. Let $B = (I_0 \cup I_1, E)$ be a weighted bipartite graph, where $I_0 = \{1, \ldots, n_0\}$, $I_1 = \{1, \ldots, n_1\}$, and the weights of edges are non-negative. Let $|E| = m$ and let $n = \max(n_0, n_1)$. Assume, without loss of generality, that $n \leq m$. The weight of edge (i, j) is denoted by $w(i, j)$. Extend the weight function to any pair (i, j), where $1 \leq i \leq n_0$ and $1 \leq j \leq n_1$ by letting $w(i, j) = 0$ for all $(i, j) \notin E$. An edge (i, j) *dominates* (i', j') if $i \leq i'$ and $j \leq j'$. Two pairs $(i, j), (i', j')$ *cross* if $i \leq i'$ and $j \geq j'$ or $i \geq i'$ and $j \leq j'$. The crossing between such a pair is *proper* if $i \neq i'$ and $j \neq j'$.

Nested Maximum Non-Crossing Matching with Cuts. The *Maximum Non-Crossing Matching* (*MNCM*) problem is: given a weighted bipartite graph B, find a maximum total

weight set of non-crossing edges. In this paper, we use a slightly more general version of the problem. The generalization goes in two ways. First, we are interesting in finding, for all edges (i,j), $MNCM(i,j)$, which is defined to be the $MNCM$ of the graph restricted to the edges that are dominated by (i,j). We call this version of the problem the *Nested Maximum Non-Crossing Matching* problem ($MNCM^*$). The standard algorithms to solve the $MNCM$ problem solve, in fact, the nested version of the problem. Our second extension is to assume that the graph also contains a special type of weighted edges, called *cut edges*. We are interested in the Non-Crossing Matching problem, subject to the restriction that if the matching contains a cut edge (i,j) then it cannot contain any edge that dominated by (i,j). We call this problem the Maximum Non-Crossing Matching problem with Cuts Problem and denote it by $MNCM_c$. Finally, the Nested Maximum Non-Crossing Matching with Cuts ($MNCM_c^*$) Problem is that of computing, for all edges (i,j), the value $MNCM_c(i,j)$, which, as before, is defined to be the $MNCM_c$ of the graph restricted to the edges dominated by (i,j). Let $cut(i,j)$ denote the weight of the maximum weight cut edge between i and j (if there is no cut edge between i and j, set $cut(i,j) = 0$).

The $MNCM_c^*$ problem can be solved in $O(n^2)$ time by dynamic programming using the following recurrence:

$$MNCM_c(i,j) = \begin{cases} 0 & \text{for } i < 0 \text{ or } j < 0 \\ \max \begin{cases} MNCM_c(i-1,j) \\ MNCM_c(i,j-1) \\ MNCM_c(i-1,j-1) + w(i,j) \\ cut(i,j) \end{cases} & \text{otherwise} \end{cases} \quad (1)$$

In the case of sparse graphs, the $O(m \log n)$ time (or $O(m \log\log n)$-time using integer priority queue operations of [14]) algorithm for $MNCM$ [13] extends trivially to the $MNCM_c^*$ problem.

Nested Maximum Crossing problem. Consider a pair (i,j), where $0 < i \leq n_0$, $0 < j \leq n_1$. We say the crossing between e_1, e_2 *is dominated* by (i,j) if e_1 and e_2 cross properly and both of them are dominated by (i,j). The Maximum Crossing (MC) problem is that of finding the maximum total weight pair of crossing edges. As in the case of Maximum Non-Crossing Matching, we are interested in the "nested" version of the problem (MC^*). Namely, for any (i,j) where $i \in I_0$, $j \in I_1$ compute $MC(i,j)$, which is defined to be the MC of the graph restricted to the edges that are dominated by (i,j).

The MC^* problem can also be solved in $O(n^2)$ time by dynamic programming, based on the following recurrence:

$$MC(i,j) = \begin{cases} 0 & \text{for } i = 0 \text{ or } j = 0 \\ \max \begin{cases} MC(i-1,j) \\ MC(i,j-1) \\ \max\{w(i',j)|i' < i\} + \max\{w(i,j')|j' < j\} \end{cases} & \text{otherwise} \end{cases} \quad (2)$$

In this paper, we give an $O(m \log n)$-time algorithm for the MC^* problem. In fact, we need to solve a slightly more general problem:

The Red-Green Crossing problem. Let B be a bipartite whose edges are colored with two colors: red and green. Order the set of edges, E, by lexicographic order. (We represent the edges as pairs such that the endpoint that belongs to I_0 is the first element of a pair.) A *red-green* crossing is a crossing between a red edge and a green edge such that the green edge is lexicographically larger than the red edge. Our goal is to compute the solution to the nested version of the following problem: find the maximum weight red-green crossing. We refer to this problem as to the Maximum Red-Green Crossing problem and abbreviate it by $MRGC^*$. The algorithm for $MRGC^*$ will be used in our algorithm for the Maximum Agreement Subtree problem.

The MC^* problem can be reduced to the colored version by representing each edge as a pair of edges of the same weight and different colors.

2 The algorithm for the Nested Maximum Red-Green Crossing problem

Without loss of generality, we assume that the degree of every I_1 node in the graph B is one. For assume that a node i has degree d and is adjacent to $(j_{r_1}, \ldots, j_{r_d})$. Then we obtain an equivalent problem by replacing i with d nodes $i_1, \ldots i_d$ of degree one such that i_k is adjacent to j_{r_k} and $w(i_k, j_{r_k}) = w(i, j_{r_k})$.

In the algorithm, we process the edges of B in lexicographic order. For each processed edge, e, we compute $MRGC(e)$ and store information about the edge e and about some crossings relevant for computing $MRGC$ values for future edges. We store the information in a balanced ordered rooted binary tree T with n leaves. The i^{th} leaf of T (in the left-to-right order) is vertex i of the set I_1. Thus T is a binary search tree over I_1. We say an edge $(i,j) \in I_0 \times I_1$ is *local* to a vertex v of T if j is a descendant of v. Information will be stored in some attributes – called $g(v), r(v)$, and $x(v)$ – of the vertices v in T.

We will typically access the information in T by following a path from a node to the root. In addition to the standard path, we will use what we call the left fringe and the right fringe. The *left fringe* of v in tree T is the sequence of nodes that are left children of the nodes on the

path from v to the root but which do not belong to the path itself. Thus the leaves descending from vertices in the left fringe of a leaf v are exactly the leaves that are smaller than v. The *right fringe* of v is defined symmetrically. For any node attribute a and node v of T we define $max_a(v)$ (resp. $max\_left_a(v)$) to be the maximum value of $a(u)$ over all nodes u on the path (resp. left fringe) from v to the root.

The main idea is to store in the tree T enough information to be able to compute quickly a derived attribute $pair(v)$, defined as follows on the nodes of the tree T:

For each vertex v define $pair(v)$ to be the weight of a maximum weight red-green crossing (e_1, e_2) among the edges processed so far such that e_1 is a red edge local to v. The importance of the function *pair* is explained in the following lemma:

Lemma 2.1 *If (i,j) is the next edge to be processed by the algorithm then*

$$MRGC(i,j) = max\_left_{pair}(j).$$

Proof: Assume that the maximum total weight crossing dominated by (i,j) is the crossing between edges e_1 and e_2. Then both e_i must be local to some node that belongs to the left fringe of j. Thus $MRGC(i,j) \leq max\_left_{pair}(j)$.

Assume now that $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$ is the pair that realizes the crossing reported in $max\_left_{pair}(j)$. Without loss of generality, let e_1 be a red edge local to some node v on the left fringe of j and e_2 be a lexicographically larger green edge. Since (i,j) is the next edge to be processed, we have that $i_1, i_2 \leq i$. Since e_1 is local to some node v on the left fringe of j, we have that $j_1 < j$, and thus e_1 is dominated by (i,j). Since e_2 crosses e_1 and e_2 is lexicographically larger than e_1, we have $j_2 < j_1 \leq j$ and thus e_2 is also dominated by (i,j). Therefore $MRGC(i,j) \geq max\_left_{pair}(j)$ and the lemma follows. □

In order to be able to compute values of the function *pair*, for every node v we will make the attributes $x(v)$, $r(v)$, and $g(v)$ satisfy the following invariants:

I1: $max_g(v)$ is the weight of heaviest green edge e such that

- e is lexicographically larger than all the red edges local to v, and
- e crosses *all* these red edges.

I2: $r(v)$ is the weight of a maximum red edge local to v.

I3: $x(v)$ is the maximum weight red-green crossing between any pair of edges local to v.

The invariants give us the following relationship to our previous derived attribute *pair*:

Lemma 2.2 *If the invariants* I1-I3 *are satisfied,*

$$pair(v) = \max\{x(v), max_g(v) + r(v)\}.$$

Proof: Note that, by I1 and I2, $max_g(v) + r(v)$ reports the maximal crossing between a red edge e that is local to v and a green edge that is lexicographically larger than e and which is not local to v. By $I3$, $x(v)$ is the maximum weight red-green crossing local to v. □

To update attributes along the paths of the tree T we use the following operations.

update, *update_right*: Given a value x, a leaf j, and an attribute a, procedure $update_a(x, j)$ ($update\_right_a(x, j)$) assigns, for every node u on the path from j (resp. the right fringe of j to the root), $a(u) \leftarrow\!- max\{a(u), x\}$.

propagate_down_left: Given a leaf j, $propagate\_down\_left_a(j)$, assigns, for every node u on the left fringe of j, $a(u) \leftarrow\!- max_a(u)$.

propagate_up: Given a leaf j and an attribute a, procedure $propagate\_up_a(j)$ changes the value of the attribute a of every node, u, on the path from j to the root to be the maximum value of a on the subpath from j to u.

$clear_g$: This procedure clears the path from j to the root by assigning zero to all g attributes on that path without changing the value of $max_g(j)$ for any other node in the tree. This is done by pushing the values on the path from j down to the left and right fringes before setting them to zero.

algorithm *MRGC*(B);

1. Construct tree T and initialize all its attributes to zero
2. **for** each edge (i, j) of weight w in lexicographic order **do**
   - 2.1 $propagate\_down\_left_g(j)$;
   - 2.2 $MRGC(i, j) \leftarrow\!- max\_left_{max(x, r+g)}(j)$
   - 2.3 **if** (i, j) is green **then**
     - 2.3.1. $update\_right_g(w, j)$
     - 2.3.2. **for** every node v on the right fringe of j **do** $x(parent(v)) \leftarrow\!- \max\{x(parent(v)), w + r(v)\}$
     - 2.3.3. $propagate\_up_x(j)$
   - 2.4 **else**
     - 2.4.2. $update_r(w, j)$
     - 2.4.2. $clear_g(j)$

Lemma 2.3 *The algorithm MRGC correctly computes the value of $MRGC(e)$, for every edge e of the graph B, provided the degree of each vertex in I_1 is one. The complexity of the algorithm is $O(m \log n)$.*

Proof: By Lemma 2.1 and Lemma 2.2, all we need to show is that points I1-I3 are the invariants of the algorithm. Initially all attributes are initialized to zero, thus the invariants are satisfied. Assume that the invariants are satisfied before processing of the edge (i,j). Let (i,j) be the next processed edge. Invariant I1 is guaranteed by steps 2.3.1 and 2.4.2. Invariant I2 is guaranteed by step 2.4.2. Invariant I3 is guaranteed by steps 2.3.1–2.3.2. To see the last fact, note that since the edges are considered in lexicographic order, a new red-green crossings can be introduced only together with a new green edge. If such a new edge creates a crossing which is local to a node v and is bigger than the crossing currently reported in $x(v)$, then it could be created in one of the two ways:

- the new crossing is also local to one of the children of v; then $x(v)$ is updated in step 2.3.2, or
- it is obtained by crossing the new edge with the maximum local green edge of the right child of v, provided this child is on the fringe of j. This type of crossings is tested for in step 2.3.1.

Each iteration takes $O(\log n)$ time, thus the cost of the algorithm $MRGC$ is $O(m \log n)$. □

In particular, we can now conclude:

Theorem 2.4 *Given a bipartite weighted graph, the $MRGC^*$ problem can be solved in $O(m \log n)$ time.*

3 The algorithm for the Maximum Agreement Subtree problem

Fix the two trees T_0, T_1 for which we want to solve the maximum agreement subtree problem. For any graph $G = (V, E)$, let $V(G) = V$. For any subtree T' of a rooted tree T, let $r(T')$ be the node in T' of minimum depth. For all $(v_0, v_1) \in V(T_0) \times V(T_1)$, let $MAST(v_0, v_1)$ denote the $MAST$ of the subtrees rooted respectively at v_0 and v_1. Here the subtrees are understood to be topologically restricted to the intersection of their label sets. More formally, $MAST(v_0, v_1) = MAST(T_0|B, T_1|B)$ where $B = L(v_0) \cap L(v_1)$, and where $L(v)$ is the label set descending from v.

The SW algorithm [18] is a dynamic programming approach that computes the $MAST(v_0, v_1)$ for all $O(n^2)$ pairs $(v_0, v_1) \in V(T_0) \times V(T_1)$. We will only compute $MAST(v_0, v_1)$ for $n \log^2 n$ of these pairs, but then we will pay an factor $O(\log n)$ for data structures related to the MCP and MNCM problems. Thus, we will replace the $\Theta(n^2)$ bound from the SW algorithm by an $O(n \log^3 n)$ bound.

For each internal vertex v in T_i $(i = 0, 1)$, let the *center child*, denoted $cntr(v)$ be the node with the maximum number of descending leaves, and let the *side child*, denoted $side(v)$ be the other child, with ties broken arbitrarily. Correspondingly, call the arc $(x, cntr(x))$ a *center arc*, and $(x, side(x))$ a *side arc*. By a *center path* we mean a maximum path using center arcs. Let $\prec$ order the center paths in each tree following the preordering of the roots of the center paths. Also, let $\prec$ be the corresponding lexicographic ordering of the center path pairs (P_0, P_1) where P_i is a center path in T_i. We will process the center path pairs, computing certain $MAST$-values in the order determined by $\prec$. We will use the recursive formula below, which is a simple reformulation of the one used in the SW algorithm [18]. For technical reasons, for any leaf v, we define $side(v) = cntr(v) = \perp$, where $\perp$ is a special "vertex" not belonging to any tree.

$$MAST(x_0, x_1) = \begin{cases} 0 \text{ if } x_0 = \perp \text{ or } x_1 = \perp. \text{ Otherwise:} \\ \max \begin{cases} a) 1 \text{ if } (x_0, x_1) \text{ is an interesting leaf pair.} \\ b) MAST(x_0, cntr(x_1)), MAST(cntr(x_0), x_1) \\ c) MAST(cntr(x_0), cntr(x_1)) + \underline{MAST(side(x_0), side(x_1))} \\ d) \underline{MAST(side(x_0), x_1)}, \underline{MAST(x_0, side(x_1))} \\ e) \underline{MAST(side(x_0), cntr(x_1))} + \underline{MAST(cntr(x_0), side(x_1))} \end{cases} \end{cases} \quad (3)$$

The underlining indicate vertex pairs belonging center path pairs preceding that of (x_0, x_1) in $\prec$. To see the relationship with the previous recurrence formulas 1 and 2, think of $cntr(x_i)$ as $x_i - 1$. In the following, we shall see that we only need to compute relatively few $MAST$-values.

Interesting and significant vertex pairs. For each internal vertex v in $T_i, i = 0, 1$, let $L^s(v)$ denote the label set descending from $side(v)$. For leaves v, $L^s(v) = L(v)$. Note that if r is the root of a center path P, then $\{L^s(v) | v \in V(P)\}$ is a partitioning of $L(r)$.

A vertex pair $(v_0, v_1) \in V(T_0) \times V(T_1)$ is *interesting* if $L^s(v_0) \cap L^s(v_1) \neq \emptyset$.

Lemma 3.1 *The are at most $n \log^2 n$ interesting vertex pairs, and they can be identified in time $O(n \log^2 n)$.*

Proof: Any interesting vertex pair is witnessed by a label, and since the path from a leaf to the root pass at most $\log n$ side arcs, each of the n labels can witness at most $\log^2 n$ interesting pairs. □

A center path pair (P_0, P_1) is *interesting* if it contains an interesting vertex pair, or equivalently, if $L(r(P_0)) \cap L(r(P_1))) \neq \emptyset$. The size of (P_0, P_1), denoted $||P_0, P_1||$, is the number of interesting vertex pairs $(v_0, v_1) \in V(P_0) \times V(P_1)$. Thus, by Lemma 3.1, $\sum_{(P_0, P_1)} ||P_0, P_1|| \leq n \log^2 n$. A vertex pair (v_0, v_1) in an interesting center path pair (P_0, P_1) is said to be *significant* if either

(i) (v_0, v_1) is interesting,

(ii) $(v_0, v_1) = (r(P_0), r(P_1))$, or

(iii) v_0 or v_1 is the root of its center path and the other node belongs to an interesting vertex pair.

In relation to recurrence 3, note the following relationship between interesting and significant vertex pairs:

Lemma 3.2 *If* $(v_0, v_1) \in V(P_0) \times V(P_1)$ *is interesting then* $(v_0, side(v_1))$, $(side(v_0), v_1)$, *and* $(side(v_0), side(v_1))$ *are all significant, and belong to center path pairs preceding* (P_0, P_1) *in* $\prec$.

Proof: For $i = 0,1$, let P_i be the path containing v_i, and let Q_i be the path with root $side(v_i)$. By definition (v_0, v_1) is interesting because $L^s(v_0) \cap L^s(v_1) \neq \emptyset$. Thus, v_0 is interesting with respect to some vertex in Q_1, so v_0 is interesting in (P_0, Q_1), so $(v_0, side(v_1))$ satisfies (iii) relative to $(P_0, Q_1) \prec (P_0, P_1)$. The case of $(side(v_0), v_1)$ is symmetric. Concerning $(side(v_0), side(v_1))$ we note that $(side(v_0), side(v_1))$ satisfies (ii) relative to $(Q_0, Q_1) \prec (P_0, P_1)$. □

In our dynamic program we want to compute the $MAST$-value for all significant vertex pairs. These $MAST$ values are stored in a binary search tree, thus they can be accessed in $O(\log n)$ time. Note that there are at most $3 \cdot ||P_0, P_1|| + 1$ significant pairs in (P_0, P_1). Uninteresting center path pairs have no significant vertex pairs, so by Lemma 3.1, in total there are only $O(n \log^2 n)$ significant vertex pairs. Also note that in $O(n \log^2 n)$ time, we can identify all significant vertex pairs for all interesting center path pairs. We will compute the significant vertex pairs for one interesting path pair at a time, following the lexicographic ordering given by $\prec$. More specifically we wish show:

Proposition 3.3 *Let* (P_0, P_1) *be an interesting center path pair. Given the MAST-values for all significant pairs belonging to center path pairs* $(Q_0, Q_1) \prec (P_0, P_1)$, *we can compute the MAST-values for all significant pairs in* (P_0, P_1) *in* $O(||P_0, P_1|| \log n)$ *time.*

Proof: The proof of the proposition essentially covers the rest of this section. Fix the interesting center path pair (P_0, P_1), and assume we have computed the $MAST$-values for all significant pairs belonging to center path pairs $(Q_0, Q_1) \prec (P_0, P_1)$. Thus, by Lemma 3.2, if $(v_0, v_1) \in V(P_0) \times V(P_1)$ is interesting, then $MAST(v_0, side(v_1))$, $MAST(side(v_0), v_1)$, and $MAST(side(v_0), side(v_1))$ are all known.

Modifying the recurrence formula. In order to relate to the $MNCM$ and MC recurrences 1 and 2, we enumerate the vertices in each P_i from 0 to $|P_i| - 1$ starting from the leaf. Thus

$x-1$ replaces $cntr(x)$, and we set $\perp = -1$. Now, rewrite recurrence 3 as:

$$MAST(x_0,x_1) = \begin{cases} 0 \text{ if } x_0 = -1 \text{ or } x_1 = -1. \text{ Otherwise:} \\ \max \begin{cases} a) 1 \text{ if } (x_0,x_1) \text{ is an interesting leaf pair.} \\ b) MAST(x_0, x_1-1), MAST(x_0-1, x_1) \\ c) MAST(x_0-1,x_1-1) + MAST(side(x_0), side(x_1)) \\ \quad \text{if } (x_0,x_1) \text{ is interesting.} \\ d) cut'(x_0,x_1) \text{ if } (x_0,x_1) \text{ is interesting.} \\ e) cut''(x_0,x_1) \\ \quad \text{if } (x_0+1,x_1+1) \text{ is interesting or } (x_0,x_1) \text{ is significant.} \end{cases} \end{cases} \quad (4)$$

where cut' and cut'' replace the expressions in the last two lines of (3) and are explained below. Note that our condition in c) of (x_0,x_1) being interesting is valid in the sense that otherwise $MAST(side(x_0), side(x_1)) = 0$ and then c) follows by applying b) twice. Set $m = ||P_0, P_1||$, that is, let m be the number of interesting vertex pairs in (P_0, P_1). Recall that the number of significant vertex pairs is bounded by $3m+1 = O(m)$. Assuming appropriate definitions of cut' and cut'', we now have an $MNCM_c^*$ problem with $O(m)$ cut-edges corresponding to a), d), and e), $O(m)$ normal "matching" edges corresponding to c), and $O(m)$ "query" edges (x_0,x_1) for which we want to know $MNCM(x_0,x_1)$; namely the significant vertex pairs. At the moment, we may have some vertices that are not incident with any of these edges. However, from the significant edges, we can identify all the matching edges, and clearly, in $O(m \log n)$ time, we can sort the involved vertices so as to skip the superfluous vertices. Afterwards, we have an $O(m)$ sized $MNCM_c^*$ problem, which we know that we can solve in time $O(m \log n)$. This time bound also includes the $O(m)$ accesses of $O(\log n)$ time to look up needed $MAST$ values.

The cut edges. We define

$$cut'(x_0,x_1) = \max\{MAST(side(x_0),x_1), MAST(x_0,side(x_1)\}. \quad (5)$$

The reason why in 3d) we only need to consider the case where (x_0,x_1) are interesting is that otherwise, for $\{i,\bar{i}\} = \{0,1\}$, $MAST(side(x_i),x_{\bar{i}}) = MAST(side(x_i),x_{\bar{i}}-1) \leq MAST(x_i,x_{\bar{i}}-1)$. In other words, if (x_0,x_1) is not interesting then b) overrules d).

The value $cut''(x_0,x_1)$ replaces the last line in the recurrence (3). Unfortunately, there may be $\Omega(n^2)$ non-zero values of the sums $MAST(side(x_0),x_1-1) + MAST(x_0-1,side(x_1))$. We define cut'' as follows:

$$cut''(x_0,x_1) = \max \begin{cases} b) cut''(x_0-1,x_1), cut''(x_0,x_1-1) \\ e) \max\{MAST(side(x_0),x'_1) \mid x'_1 < x_1 \text{ and } (x_0,x'_1) \text{ is interesting}\} \\ \quad + \max\{MAST(x'_0,side(x_1)) \mid x'_0 < x_0 \text{ and } (x'_0,x_1) \text{ is interesting}\} \end{cases} \quad (6)$$

To see that this definition of cut'' is correct, first recall that if (x_0,x_1) is not interesting, for $\{i,\bar{i}\} = \{0,1\}$, $MAST(side(x_i),x'_{\bar{i}}) = MAST(side(x_i),x'_{\bar{i}}-1)$. Hence

$$MAST(side(x_i),x_{\bar{i}}-1) = \max\{MAST(side(x_0),x'_{\bar{i}}) \mid x'_{\bar{i}} < x_{\bar{i}} \text{ and } (x_0,x'_{\bar{i}}) \text{ is interesting}\}.$$

Thus 6e) is equal to 3e).

The point in 6b) is that it can substitute for 3b). More precisely, all calls to 3b) immediately preceding a call to 3e) are translated into calls to 6b)—any other calls to 3b) are replaced by calls to 4b). Thus, our entry (x_0, x_1) into 4e) is either a start query meaning that (x_0, x_1) is significant, or it is following a call to c) implying that $(x_0 + 1, x_1 + 1)$ is interesting. Thus we may conclude that recurrence formulas 4,5 and 6 are equivalent to recurrence formula 3.

Now, recurrence formula 4 is equivalent to the recurrence 2 of the Nested Red-Green Maximum Crossing Pair problem ($MRGC^*$). For each interesting pair (x_0, x_1), we have a green edge (g, x_0, x_1) with weight $MAST(side(x_0), x_1)$ and a red edge (r, x_0, x_1) with weight $MAST(x_0, side(x_1))$. As before, we note that in $O(m \log n)$ time, we can skip all vertices not incident with any red or green edges, and afterwards, we have an $MRGC^*$ problem is of size $O(m)$. By Theorem 2.4, this problem can be solved in time $O(m \log n)$. This completes the computation of the $MAST$-values of all significant vertex pairs in (P_0, P_1), based on $MAST$-values of significant vertex pairs from center path pairs $(Q_0, Q_1) \prec (P_0, P_1)$. Thus follows Proposition 3.3. □

In conclusion, we have shown

Theorem 3.4 *The MAST problem on two binary trees can be solved in $O(n \log^3 n)$ time.*

Proof: By Lemma 3.1 there are at most $n \log^2 n$ interesting vertex pairs, and hence at most $n \log^2 n$ interesting center path pairs. These are sorted by the lexicographic ordering $\prec$ in time $O(n \log^3 n)$. Processing the path pairs (P_0, P_1) following this ordering, by Proposition 3.3, we can compute the $MAST$-values of all significant vertex pairs in (P_0, P_1) in $O(||P_0, P_1|| \log n)$ time, hence in $O(n \log^3 n)$ total time. At the end, we can return $MAST(r(T_0), r(T_1))$ which by case (ii) is a significant pair in the center path pair containing both roots. □

4 Concluding remarks

Consider, now, the case of trees with some small degree bound, d. Things become slightly more complicated in this case. The recurrence for $MAST(v_0, v_1)$ then involves a weighted bipartite matching, where the independent sets are the children of v_0 and v_1, and where the weight function is $MAST$. Without going into details, in [6, §2] it is shown that we can reduce these matchings so that in total they contain only $O(n \log^2 n)$ "interesting edges". Since the vertex sets in each matching are of size at most $2d$, using the best bound for weighted bipartite matching [12], we get that the matching work can be done in time $O(n\sqrt{d} \log^3 n)$, which is thus our bound for computing the $MAST$ of two trees with degree bounded by d. This should be contrasted with the previous bound of $O(n^{1+o(1)} + n\sqrt{d} \log n)$ from [7]. The $n^{o(1)}$ term hides an expression which is $\Omega(c^{\sqrt{\log n}})$. Thus our bound of $O(n\sqrt{d} \log^3 n)$ is worse for large d, but

better for small values of d, say, if $d = \text{polylog}\, n$. In particular, our result is superior for the common case of constant degree trees.

Our $n \log^2 n$ bound on the number of interesting vertex pairs (Lemma 3.1) generalize naturally to an $n \log^k n$ bound for k trees. It would be nice to see this observation used in a general $MAST$ algorithm for k trees. It should be noted, however, that the problem fundamentally changes character for $k > 2$. In [2] Amir and Keselman showed $MAST$ to be NP-hard for just 3 unbounded degree trees. Nevertheless, using an entirely different approach, they presented an $O(kn^{d+1} + n^{2d})$ bound for this problem [2, 3]. We have since improved this bound to $O(kn^3 + n^d)$ [8]. Thus the suggested generalization of the techniques in this paper would only be significant for small values of k.

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

[3] A. Amir and D. Keselman. Maximum agreement subtrees in multiple evolutionary trees - a correction. 1994. Prersonal Communication.

[4] J-P. Barthélemy and A. Guénoche. *Trees and Proximity Representations.* Wiley, New York, 1991.

[5] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, (13) 155–179, 1995.

[6] 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.

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

[8] M. Farach, T.M.Przytycka, and M. Thorup. Agreement of many bounded degree evolutionary trees. manuscript.

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

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

[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] G.Jacobson and K-P. Vo. Heaviest increasing/common subsequence problems. In *Proceedings of the Combinatorial Matching Conferences*, 1992.

[14] D.B. Johnson. A priority queue in which initialization and queue operations take $O(\log\log D)$ time. *Math. Systems Theory*, 15:295–309, 1982.

[15] 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.

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

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

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