

Finding a Maximum Compatible Tree for a Bounded Number of Trees with Bounded Degree is Solvable in Polynomial Time

Ganeshkumar G.¹ and Tandy Warnow^{1,2}

¹ Department of Computer Sciences, University of Texas, Austin, TX 78712;
{gsgk,tandy}@cs.utexas.edu

² Supported in part by the David and Lucile Packard Foundation.

Abstract. In this paper, we consider the problem of computing a maximum compatible tree for k rooted trees when the maximum degree of all trees is bounded. We show that the problem can be solved in $O(2^{2kd}n^k)$ time, where n is the number of taxa in each tree, and every node in every tree has at most d children. Hence, a maximum compatible tree for k unrooted trees can be found in $O(2^{2kd}n^{k+1})$ time.

1 Introduction

A “phylogeny” (or evolutionary tree) represents the evolutionary history of a set of species S by a rooted (typically binary) tree in which the leaves are labelled with elements from S , and unlabelled internal nodes.

For a variety of reasons, a typical outcome of a phylogenetic analysis of a dataset can consist of many different unrooted trees, and each tree represents an equally believable estimate of the true tree. Making sense of the set of these trees is then a challenging prospect.

There are two basic approaches that have been used for this problem. The first approach (and the most popular) represents the set of trees by a single tree on the full dataset (i.e. the “consensus tree”). Consensus tree techniques such as “strict consensus” and “majority consensus” are the most popular, and have the advantage that they are polynomial time. However, consensus tree methods have the disadvantage that they tend to create consensus trees that lack resolution (and this lack of resolution can be significant), meaning that the trees can contain high degree nodes. An alternate approach seeks a subset of the taxa on which all the trees agree (the “maximum agreement subset” (MAST) approach), which means that when restricted to this subset of the taxa, the trees are identical. By contrast with the consensus tree approach, the agreement subset approach is computationally intensive (unless at least one of the trees has low degree); furthermore, the size of the maximum agreement subset can be very small (David Swofford, personal communication).

In this paper we consider a different approach to this problem, in which we seek the largest subset of taxa on which all the trees are “compatible” (meaning that when restricted to this subset of taxa, the trees share a common refinement). The problem is suited for the case where the set of trees contains unresolved trees (such as can happen when returning a set of consensus trees, one for each phylogenetic island obtained during a search for the maximum parsimony or maximum likelihood tree). In such cases, it may return a larger subset of taxa than the maximum agreement subset approach, as illustrated in Figure 1. This “maximum compatible set” (MCS) problem is NP-hard for 6 or more trees [6]. We will denote by MCT the tree constructed on the MCS, and call it the maximum compatible tree. Occasionally, when the context is clear, we will use MAST to also denote the maximum agreement subtree.

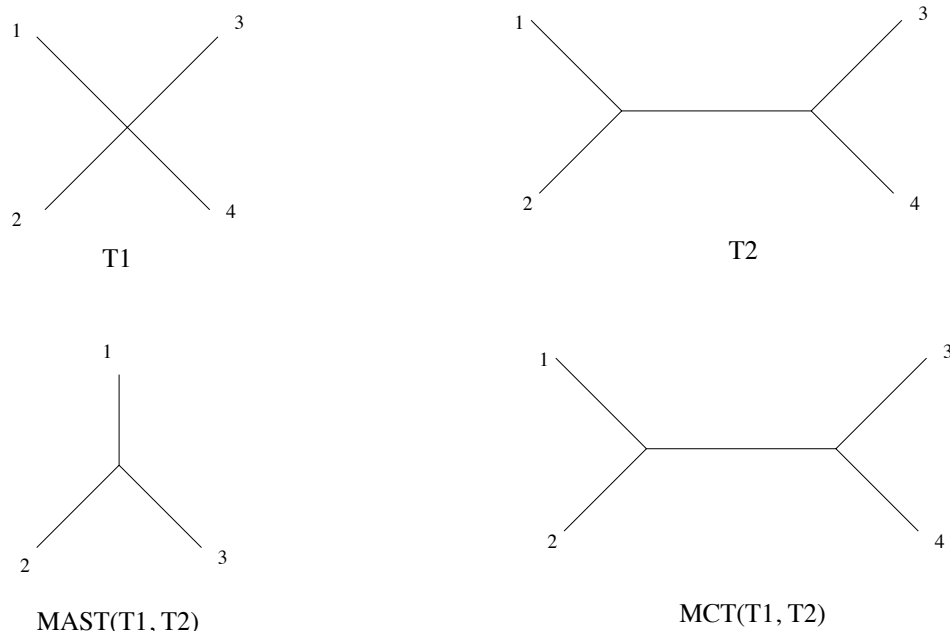


Fig. 1. The MCT of T1 and T2 has more leaves than the MAST.

Our result for the MCS problem is an algorithm for the two-tree MCS problem which runs in time $O(2^{4d}n^3)$ time, where $n = |S|$ and d is the maximum degree of the two unrooted trees. (The algorithm we present is a dynamic programming algorithm and is an extension of the earlier dynamic programming algorithm for the two-tree MAST problem in [4].) Thus, we show that the two-tree MCS problem is fixed-parameter tractable. We extend this algorithm to

the k -tree MCS problem in which all trees have bounded degree, obtaining an $O(2^{2kd} n^{k+1})$ algorithm.

The organization of the paper is as follows. In Section 2 we give some basic definitions (we will introduce and explain other terminology as needed). We then present the dynamic programming algorithm for two-tree MAST, and discuss the challenges in extending the algorithm to the two-tree MCS problem. In Section 3 we present the dynamic programming algorithm for the two-tree MCS problem, the proof of correctness, and running time analysis. We then show how we extend this algorithm to the k tree case. In Section 4 we discuss further work in the area.

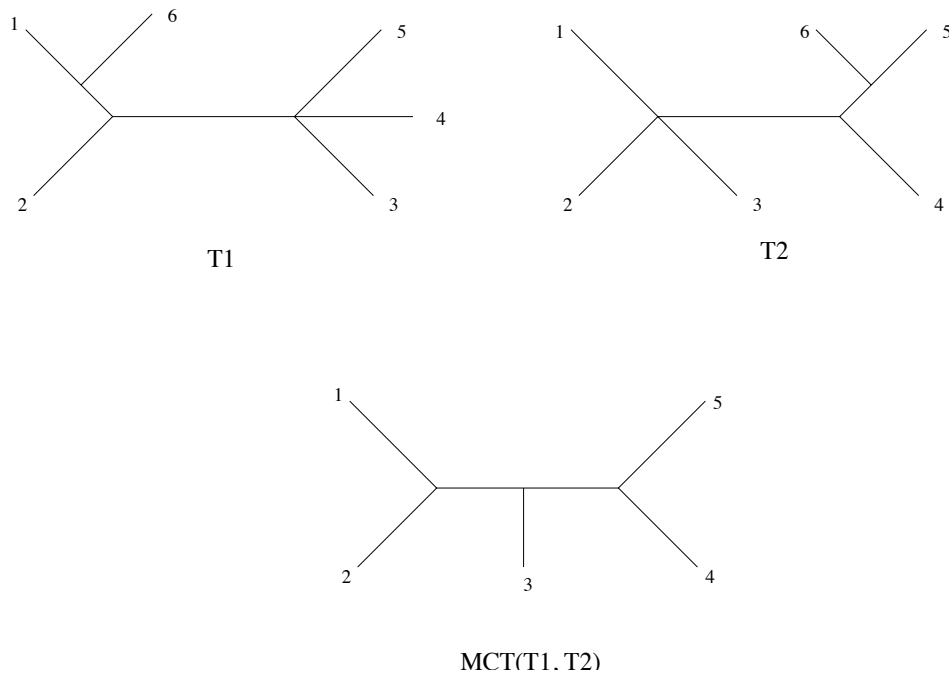


Fig.2. The MCT of Trees T1 and T2

2 Computing a Maximum Agreement Subtree of two Trees

In this section, we first present some definitions and then we describe the dynamic programming algorithm for computing the maximum agreement subtree of two trees, as given in [4]. The algorithm actually computes the cardinality of

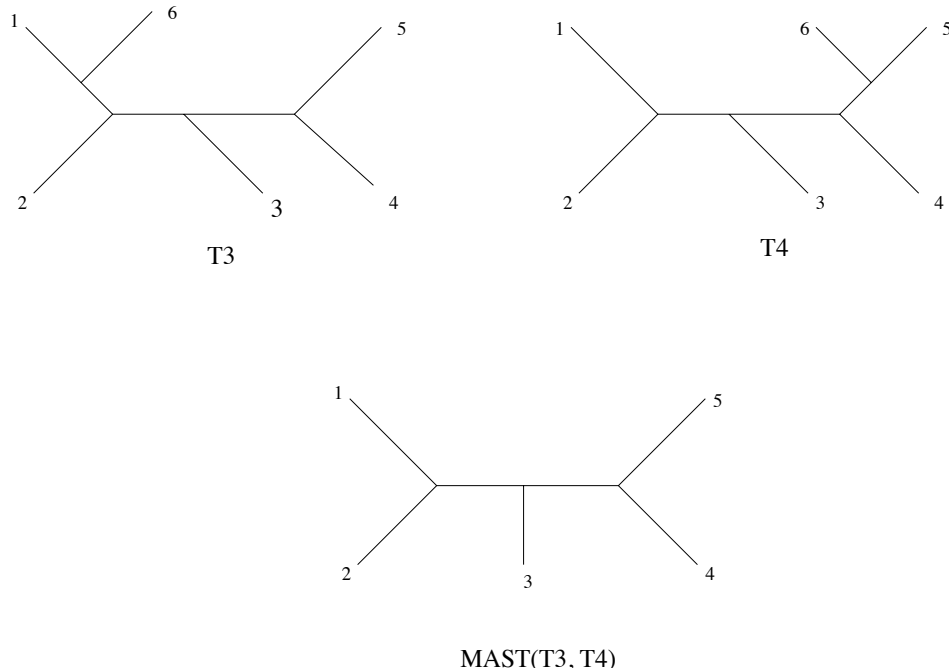


Fig. 3. The MAST of Trees T3 and T4

the maximum agreement subset but can be easily extended to give the maximum agreement subset (or equivalently the maximum agreement subtree).

Definition 1 *The restriction of a tree to a subset of the leaves.*

Given a leaf-labelled tree T on a set S , the restriction of T to a set $X \subseteq S$ is obtained by removing all leaves in $(S - X)$ and then suppressing all internal nodes of degree two. This restriction will be denoted by $T|_X$.

Definition 2 *Maximum Agreement Subset (MAST).*

Given a set $\Sigma = \{T_1, T_2, \dots, T_k\}$ of leaf-labelled trees on the set S , a maximum agreement subset for the set of trees Σ is a set X of maximum cardinality such that the trees $T_i|_X$ are all isomorphic. (The isomorphism must map leaves with the same label to each other).

Definition 3 *Maximum Compatible Subset (MCS).*

Given a set $\Sigma = \{T_1, T_2, \dots, T_k\}$ of leaf-labelled trees on the set S , a maximum compatible subset for the set of trees Σ is a set X of maximum cardinality such that the trees $T_i|_X$ all share a common refinement.

We will first assume that the two trees T and T' are both rooted (we will later discuss how to extend the algorithm for unrooted trees). The trees can have any degree, but both are leaf-labelled by the same set S of taxa.

Let v be a node in T , and denote by T_v the subtree of T rooted at v . Similarly denote by T'_w the subtree of T' rooted at a node w in T' . Denote by $MAST(T_v, T'_w)$ the number of leaves in a maximum agreement subset of T_v and T'_w . The dynamic programming algorithm operates by computing $MAST(T_v, T'_w)$ for all pairs of nodes (v, w) in $V(T) \times V(T')$, “bottom-up”.

We now describe the basic idea of the dynamic programming algorithm. First, the value $MAST(T_v, T'_w)$ is easy to compute when either v or w are leaves. Now consider the computation of $MAST(T_v, T'_w)$ where both v and w are not leaves, and let X be a maximum agreement subset of T_v and T'_w . The least common ancestor of X in T_v may be v , or it may be a node below v . Similarly, the least common ancestor of X in T'_w may be w or it may be a node below w . We have four cases to consider:

1. If the least common ancestor of X is below v in T and similarly below w in T' , then $|X| = MAST(v_i, w_j)$ for some pair (v_i, w_j) of nodes where v_i is a child of v and w_j is a child of w .
2. If the least common ancestor of X is below v in T but equal to w in T' , then $|X| = MAST(v_i, w)$ for some child v_i of v .
3. If the least common ancestor of X is below w in T' but equal to v in T , then $|X| = MAST(v, w_j)$ for some child w_j of w .
4. If the least common ancestor of X is v in T and w in T' , then X is the disjoint union of X_1, X_2, \dots, X_p where each X_i is a maximum agreement subset for some pair of subtrees T_{v_a} and T'_{w_b} (in which v_a is a child of v and w_b is a child of w). Furthermore, for each i and j , the pairs of subtrees associated to X_i and X_j are disjoint. Hence X is the maximum value of a matching in the weighted complete bipartite graph $G(A, B, E)$ in which $A = \{v_1, v_2, \dots, v_j\}$ (for the children of v), $B = \{w_1, w_2, \dots, w_p\}$ (for the children of w), and the weight of edge (v_i, w_j) is $MAST(v_i, w_j)$.

This discussion suggests a straightforward dynamic programming algorithm which involves computing $O(n^2)$ subproblems, each of which involves computing the maximum of a number of $O(d)$ values (where d is the maximum degree of any node in both T and T'). Each of these values in turn is easy to compute, though the maximum weighted bipartite matching of an $O(d)$ vertex graph takes $O(d^{2.5} \log d)$ time [3]. The running time analysis of the MAST algorithm given in [4] shows it is $O(n^{4.5} \log n)$ if d is not bounded, but $O(n^2)$ if d is bounded.

3 Algorithms for the MCS problem

3.1 Relation between MCS and MAST

We begin by observing the following:

Lemma 1. *Let T_1 and T_2 be two unrooted leaf-labelled trees. Let X be an MCS of T and T' . Then there exists a binary tree T'_1 refining T_1 and a binary tree T'_2 refining T_2 such that X is a MAST of T'_1 and T'_2 .*

Proof. Since X is a compatible subset of taxa for the pair of trees T_1 and T_2 , there is a common refinement T^* of $T_1|X$ and $T_2|X$. Hence we can refine T_1 , obtaining T'_1 , and refine T_2 , obtaining T'_2 , so that T'_1 restricted to X yields T^* , and similarly T'_2 restricted to X also yields T^* . Then X is an agreement subset of T'_1 and T'_2 . □

The above observation is illustrated in Figures 2 and 3.

This observation suggests an obvious algorithm for computing an MCS of two trees: for each way of refining the two trees into a binary tree, compute a MAST. However, this algorithm is computationally expensive, since the number of binary refinements of an n leaf tree with maximum degree d can be $O(4^d)$. Hence this brute force algorithm will not be acceptably fast.

3.2 Computing an MCS of two rooted trees

We now describe the dynamic programming algorithm for the maximum compatible set (MCS) of two rooted trees, both with degree bounded by d (by this we mean that every node has at most d children). As for the MAST problem, here too the algorithm computes the cardinality of an MCS, but can be easily extended to compute an MCS itself. This algorithm can easily be extended to produce a dynamic programming algorithm for computing an MCS of two unrooted trees, by computing an MCS of each of the n pairs of rooted trees (obtained by rooting the unrooted trees at each of the n leaves). Furthermore, we also show how to extend the algorithm to handle k rooted or unrooted trees.

The basic set of problems we need to compute must include the computation of an MCS of subtrees T_v and T'_w , for every pair of nodes v and w . (T_v denotes the subtree of T rooted at v , and similarly T'_w denotes the subtree of T' rooted at w .) We will also need to include the computation of MCS's of other pairs of trees, but begin our discussion with these MCS calculations.

Let T and T' be two rooted trees, and let v and w denote nodes in T and T' respectively. Let the children of v be v_1, v_2, \dots, v_p and the children of w be

w_1, w_2, \dots, w_q . Let X be the set of leaves involved in an MCS T^* of T_v and T'_w . Note that $T|X$ and $T'|X$ will only include those children of v and w which have some element(s) of X below them. Let A be the children of v included in $T|X$ and B be the children of w included in $T'|X$. (Note that X defines the sets A and B .)

Note also that any MCS of T and T' actually defines an agreement subset of some binary refinement of T and some binary refinement of T' (Lemma 1). Hence, T^* defines a binary refinement at the node v if $|A| > 1$, and a binary refinement at the node w if $|B| > 1$. In these cases, T^* defines a partition of the nodes in A into two sets, and a partition of the nodes in B into two sets.

There are four cases to consider:

1. $|A| = |B| = 1$, i.e. $A = \{v_i\}$ and $B = \{w_j\}$ for some i and j . In this case, any MCS of T_v and T'_w is an MCS of T_{v_i} and T'_{w_j} .
2. $|A| = 1$ and $|B| > 1$, i.e., any MCS of T_v and T'_w is an MCS of T_{v_i} and T'_w for some i .
3. $|A| > 1$ and $|B| = 1$, i.e., any MCS of T_v and T'_w is an MCS of T_v and T'_{w_j} for some j .
4. $|A| > 1$ and $|B| > 1$.

The analysis of the fourth case is somewhat complicated, and is the reason that we need additional subproblems. Recall that T^* defines a bipartition of A into $(A', A - A')$ and B into $(B', B - B')$. Further, recall that T^* is a binary tree with two subtrees off the root; we call these subtrees T_1 and T_2 . It can then be observed that T_1 is an MCS of the subtree of T_v obtained by restricting to the nodes below $A - A'$ and the subtree of T'_w obtained by restricting to the nodes below $B - B'$. Similarly, T_2 is an MCS of the subtree of T_v obtained by restricting to the nodes below A' and the subtree of T'_w obtained by restricting to the nodes below B' . Hence we need to define additional subproblems as follows. For each $A' \subset A$ define the tree $T(v, A')$ to be subtree of T_v obtained by deleting all the children of v (and their descendents) not in A' . Similarly define the tree $T'(w, B')$ to be the subtree of T'_w obtained by deleting all the children of w (and their descendents) not in B' . The construction of tree $T(v, A')$ is illustrated in Figure 4. Now define $MCS(v, A', w, B')$ to be the size of an MCS of $T(v, A')$ and $T'(w, B')$ ¹. From the above discussion it follows that:

$MCS(v, A', w, B')$ is the maximum of:

$$- \max\{MCS(v, A', w_j, Children(w_j)) : w_j \text{ a child of } w\},$$

¹ The size of a tree is taken to mean the number of leaves in the tree.

- $\max\{MCS(v_i, Children(v_i), w, B') : v_i \text{ a child of } v\}$,
- $\max\{MCS(v, A'', w, B'') + MCS(v, A' - A'', w, B' - B'') : A'' \text{ and } B'' \text{ are non-empty proper subsets of } A' \text{ and } B', \text{ respectively.}\}$.

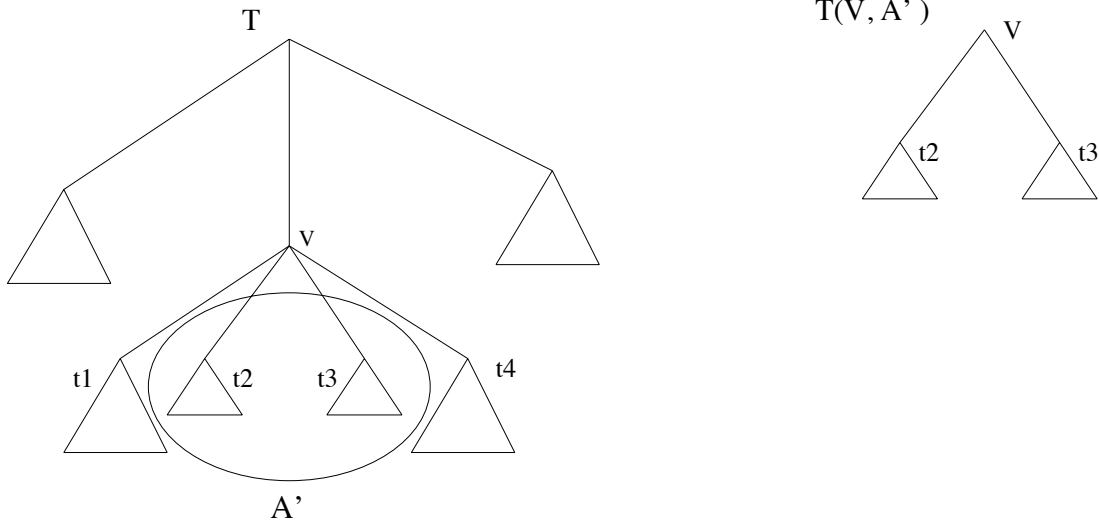


Fig.4. The tree T , the set A' and the tree $T(v, A')$

The computation of these subproblems follows the obvious partial ordering, in which $MCS(v, A, w, B)$ must follow $MCS(v', A', w', B')$ if both of the following conditions holds:

- v lies above v' or [$v = v'$ and $A' \subseteq A$].
- w lies above w' or [$w = w'$ and $B' \subseteq B$].

The base cases, in which v is a leaf or w is a leaf, are easy to compute, and equal 1 or 0. For example, if v is a leaf then necessarily $A = \emptyset$, and so $MCS(v, \emptyset, w, B) = 1$ if $v \in T'(w, B)$, and 0 otherwise.

Running time analysis There are $O(2^d n)$ trees $T(v, A)$, and hence $O(2^{2d} n^2)$ subproblems. The computation of $MCS(v, A, w, B)$ involves computing the maximum of $2d + 2^{2d}$ values, and hence takes $O(2^{2d})$ time. Hence the running time is $O(2^{4d} n^2)$.

3.3 Algorithm for the MCS problem of k rooted trees with bounded degree

We now show how to extend the analysis to k rooted trees. In this case, the subproblems are $2k$ -tuples of the form $MCS(v_1, A_1, v_2, A_2, \dots, v_k, A_k)$ where v_i is a node in T_i and $A_i \subset \text{Children}(v_i)$. Hence there are $O(2^{kd} n^k)$ subproblems. Computing each subproblem involves taking the maximum of $O(kd + 2^{kd})$ values. Hence the running time for the algorithm is $O(2^{2kd} n^k)$ time.

3.4 Extension to unrooted trees.

For each of the algorithms described, extending the algorithm to handle unrooted trees involves rooting each of the trees at each of the n leaves (for each leaf all the trees are rooted at that leaf), and then finding the best rooting. This is based on the observation that given a leaf l , the size of an MCS of the trees rooted at l is the maximum size of a compatible set for the unrooted trees that includes l (while rooting the trees at the leaf l , l itself must be excluded from the trees. Hence the size of a maximum compatible set for the unrooted trees that includes l will be actually one more than the size of an MCS of the trees rooted at l). Since there are n leaves, this multiplies the running time by n .

Hence:

Theorem 1. *We can compute an MCS of k unrooted trees in which each tree has degree at most $d + 1$ in $O(2^{2kd} n^{k+1})$ time.*

4 Future Work

To conclude we point out that many questions about the MCS problem remain unsolved. We know that MCS is NP-hard for 6 trees with unbounded degree, but we do not know the minimum number of trees for which MCS becomes hard. In particular, we do not know if MCS is NP-hard or polynomial for two trees. It also remains to be seen if there are any approximation algorithms for the problem, or exact algorithms when only some of the trees have bounded degree.

References

1. A. Amir and D. Kesselman. Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms. *Proceedings of the 35th IEEE FOCS, Santa Fe, 1994*.
2. C.R. Finden and A.D. Gordon. Obtaining common pruned trees, *Journal of Classification*, 2, 255-276 (1985).
3. H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.* 18 (5) (1989) pp. 1013-1036.

4. M. Steel and T. Warnow. Kaikoura tree theorems: computing the maximum agreement subtree. *Information Processing Letters*, 48, 77-82 (1993).
5. M. Farach Colton, T.M. Przytycka, M.Thorup. On the Agreement of Many Trees. *Information Processing Letters* 55 (1995), 297–301.
6. A.M. Hamel and M.A. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Research Report No. 114, Department of Mathematics and Statistics, University of Canterbury*, Christchurch, New Zealand, 1994.