MAXIMUM AGREEMENT SUBTREE IN A SET OF EVOLUTIONARY TREES - Metrics and Efficient Algorithms*

Amihood Amir[†] Dmitry Keselman[‡]

Georgia Tech and Bar-Ilan University

Georgia Tech and Simons Technologies

Abstract

The *maximum agreement subtree* approach is one method of reconciling different evolutionary trees for the same set of species. An agreement subtree enables choosing a subset of the species for whom the restricted subtree is *equivalent* (under a suitable definition) in all given evolutionary trees.

Recently, dynamic programming ideas were used to provide polynomial time algorithms for finding a maximum homeomorphic agreement subtree of *two* trees. Generalizing these methods to sets of more than two trees yields algorithms that are exponential in the number of trees. Unfortunately, it turns out that in reality one is usually presented with more than two trees, sometimes as many as thousands of trees.

In this paper we prove that the maximum homeomorphic agreement subtree problem is $\mathcal{NP}$-complete for three trees with unbounded degrees. We then show an approximation algorithm of time $O(kn^5)$ for choosing the species that are *not* in a maximum agreement subtree of a set of k trees. Our approximation is guaranteed to provide a set that is no more than 4 times the optimum solution.

While the set of evolutionary trees may be large in practice, the trees usually have very small degrees, typically no larger than three. We develop a new method for finding a maximum agreement subtree of k trees, of which one has degree bounded by d. This new method enables us to find a maximum agreement subtree in time $O(kn^{d+1} + n^{2d})$.

1 Introduction

One of the methods for classifying hierarchical relations between different objects is by representing them in a tree [12]. In particular, trees have been used to represent evolutionary splits among species (cf. [11, 18, 3]). Different methods of classification may lead to different trees. It is natural to try to resolve differing evolutionary trees in a manner that will increase our confidence in the results.

*A preliminary version of this paper appeared in the proceedings of FOCS 94 [1]

†Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat Gan, ISRAEL; (972-3)531-8770; amihood@bimacs.cs.biu.ac.il; Partially supported by NSF grant CCR-92-23699 and Israel Ministry of Science and Arts grant 6297.

‡College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280; (404) 894-9678; dmitry@cc.gatech.edu.

There are two ways one can go about handling different evolutionary trees for the same species. One may try to construct a *consensus tree* that is "close" to all given trees. This direction was taken by several researchers (e.g. [15]). Another direction, the one that concerns us in this paper, is extracting a maximum set of species about whom we are confident. This method, introduced by Gordon [10], involves obtaining a *maximum agreement subtree.*

There are several alternate way to define a maximum agreement subtree. One approach was taken by Finden and Gordon [7].

Definition: Let $S = \{s_1, \ldots, s_n\}$ be a set of *labels.* An $S - labeled$ tree T is a tree with n leaves, each labeled with a distinct element of S, i.e. no two leaves have the same label. Let T be an S-labeled tree and let $S' \subseteq S$. $H(T, S')$ is the minimal homeomorphic subtree of T (all degree 2 nodes are contracted) containing exactly the leaves labeled by S'.

Let $T_1, \ldots, T_k$ be S-labeled trees. A *maximum homeomorphic agreement subtree* of $T_1, \ldots, T_k$ ($MHT(T_1, \ldots, T_k)$) is a maximum cardinality set $S' \subseteq S$ such that $H(T_1, S') = H(T_2, S') = \cdots = H(T_k, S')$. We will also refer to the problem of finding a maximum homeomorphic agreement subtree as *the MHT problem.*

We will interchangeably refer to a MHT as a *tree* or *set of labels.* There is no ambiguity since a set of labels uniquely defines the contracted subtree whose leaves are exactly the given set, if that set of labels is in a homeomorphic agreement subtree.

The most general possible way to view agreement subtree is by assigning each edge an *interval weight,* i.e. the conjectured **range** of time it took to evolve along this edge. This idea is very similar to the "graph-sandwich" of Farach, Kannan and Warnow [4]. Informally, the maximum interval weight agreement subtree (MIWT) of k S-labeleld trees $T_1, \ldots, T_k$ is a maximum set $S' \subseteq S$ such that $\forall s, t \in S'$ the distance between s and t is within the allowable range in all trees $T_1, \ldots, T_k$. A precise definition is provided in section 7.

Finden and Gordon [7] gave a heuristic algorithm for the MHT problem for **two trees** T_1, T_2. Their result is not guaranteed to be a $MHT(T_1, T_2)$. An algorithm of complexity $O(n^{1/2+\epsilon \log_2 n})$ was presented by Kubicka et. al. [14]. This algorithm is also for the MHT problem of two trees. This result was improved by Warnow and Steel [17] to finding $MHT(T_1, T_2)$ in time $O(n^2)$ for bounded degree trees and $O(n^{4.5} \log n)$ for unbounded degree trees. A similar method was independently found by Goddard *et al.* [9] for finding $MHT(T_1, T_2)$ of binary trees in time $O(n^2)$. Farach and Thorup [6] further improved these results to $O(n^2 c^{\sqrt{\log n}})$ for unbounded degree trees.

The algorithms of [17, 6, 9] only handle the restricted case of two trees. They can be generalized, but then they become exponential in the number of trees. In reality, sometimes **thousands** of trees need to be considered [9]. As for the maximum interval weight agreement subtree, there are currently no solutions to this problem.

In this paper we present the first known solution to the realistic versions of the maximum agreement subtree problem. In addition, we use novel methods for tackling this problem. The main idea behind our algorithms is to use divide and conquer rather than dynamic programming. A naive top-down algorithm would quickly reach exponential time, but we exploit some subtle internal properties of tree partitions in a manner that assures an efficient algorithm. Surprisingly, the algorithm is simple and easily programmable.

The main contributions of this paper are the following:

- We show that for unbounded degree trees, $MHT(T_1, T_2, T_3)$ and $MIWT(T_1, T_2, T_3)$ are $\mathcal{NP}$-complete.
- We present an approximation algorithm for finding a set of leaves that are *not* in a MHT of k unbounded

degree evolutionary trees. Our algorithm runs in time $O(kn^5)$. The approximation factor of our algorithm is 4.

- For trees $T_1, \ldots, T_k$, where at least one of which has degree bounded by d we present algorithms for finding $MHT(T_1, \ldots, T_k)$ and $MIWT(T_1, \ldots, T_k)$ in time $O(kn^{d+1} + n^{2d})$.

This paper is organized as follows. In section 2, we prove the $\mathcal{NP}$-completeness of the $MHT(T_1, T_2, T_3)$ and $MIWT(T_1, T_2, T_3)$ problem for unbounded degree trees. In section 3 we give the approximation algorithm for the MHT problem of k unbounded degree trees. In section 4, we define a very simple version of the MIWT problem (the maximum isomorphic agreement subtree or MIT problem) and give an efficient algorithm for trees of bounded degree. In section 5 we show the algorithm for computing a MHT of k trees, one of which has degree bounded by d. In section 6 we consider the special case of the MIWT problem where all weight ranges are a single weight (the MWT problem). We give an efficient algorithm for finding $MWT(T_1, \ldots, T_k)$ where at least one of the trees has a small bounded degree. In section 7 we provide an efficient algorithm for the maximum interval agreement subtree of a set of trees. We conclude with open problems and future research.

2 Multiple Unbounded Degree Trees

We show that finding the $MHT(T_1, T_2, T_3)$ is an $\mathcal{NP}$-complete problem. (the same proof applies to $MIWT(T_1, T_2, T_3)$). First define the decision problem:

Homeomorphic Agreement Subtree of 3 Unbounded degree Trees (3-HUT).

INSTANCE: Three S-labeled trees T_1, T_2, T_3 of unbounded degree, where $S = \{s_1, \ldots, s_n\}$; integer $i \leq n$.

QUESTION: Is there a subset $S' \subseteq S$ of size i such that $H(T_1, S') = H(T_2, S') = H(T_3, S')$?

Theorem 1: The 3-HUT problem is $\mathcal{NP}$-complete.

Proof: It is clearly in $\mathcal{NP}$. We will reduce the 3-Dimensional Matching problem (3DM) [8] to 3-HUT. In 3DM the input is a set $M \subseteq W \times X \times Y$ where W, X, Y are disjoint q element sets. We need to answer if there is a set $M' \subseteq M$ of size q where no two elements of M' agree in any coordinate.

Construct three trees T_1, T_2, T_3 as follows. Each tree T_i has a root r_i, and each root has q children. The children of r_1 correspond to the elements of W, the children of r_2 correspond to the elements of X and the children of r_3 correspond to the elements of Y. Take $S = M$. For every element $e = \langle w, x, y\rangle \in M$ attach a child labeled e to the node that corresponds to w in T_1, to the node that corresponds to x in T_2 and to the node that corresponds to y in T_3.

Assuming that the roots r_1, r_2, r_3 appear in a maximum homeomorphic subtree, then any two elements of M that agree on one or two coordinates will not be on a homeomorphic subtree of all three trees. We force the roots to be in a maximum homeomorphic subtree, by adding to each root $q+1$ children labeled $x_1, \ldots, x_{q+1}$, where the x_i are new and distinct symbols. Thus it is easy to see that there is an M' of size q iff there is a homeomorphic subtree of all three trees of size $2q+1$. $\square$

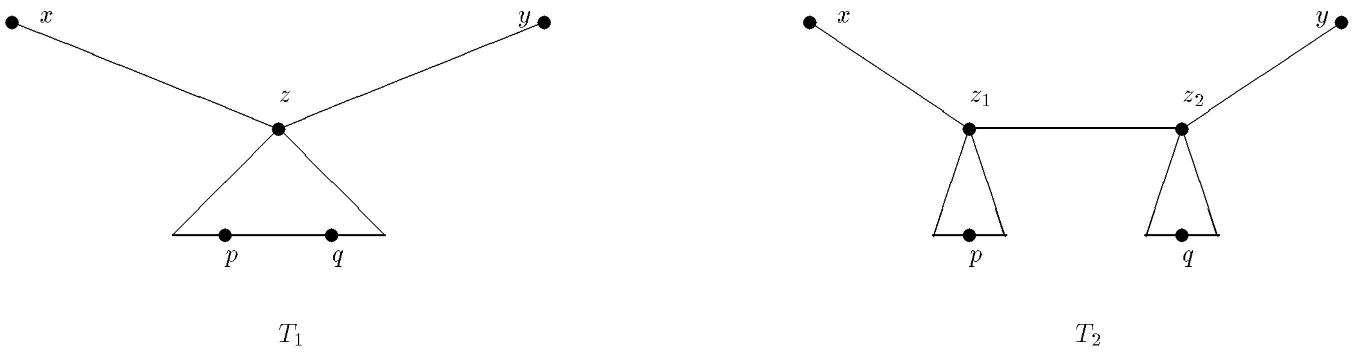


Figure 1:

3 An Approximation Algorithm

We have seen that constructing the set of leaves whose restricted subtree is homeomorphic to their restricted subtree in all trees is an $\mathcal{NP}$-hard problem. We now provide an approximation algorithm for constructing the set of leaves that are not in a MHT. The approximation algorithm is based on the following property, which was first proved by Bandelt and Dress [2]. We present the theorem and a new proof.

Theorem 2: Two evolutionary trees are homeomorphic iff all 4-leaf subtrees generate homeomorphic subtrees.

Proof: One direction is immediate. We will prove that if all 4-leaf subtrees generate homeomorphic subtrees then the trees are homeomorphic.

Call a vertex of degree greater than 2 a *non-2-vertex*. Call two leaves *twins* if the path connecting them has at most one intermediate non-2-vertex. Every tree contains at least one pair of twins. This can be seen by the following argument. Consider a path with the largest number of non-2-vertices. Let the leaf ends of this path be a and b. Let v be the closest non-2-vertex to a (a similar argument works for b). There is a path from v to a, to b and to at least some other leaf c. In the path from v to a and from v to c there are no non-2-vertices, otherwise the path from a to b would not be maximal. Therefore a and c are twins.

Proceed with proving the theorem by induction on the number of leaves n. If $n \leq 4$ the theorem is trivially true. For $n > 4$ let x and y be a pair of twins in tree T_1 whose intermediate non-2-vertex is z. We claim that x and y are twins in tree T_2 as well. Otherwise, let z_1 and z_2 be two non-2-vertices on the path from x to y in T_2. Since both are non-2-vertices, there exist leaves p and q such that p branches off z_1 and q branches off z_2. Thus in the subtree generated by x, y, p, q in T_2, x and y are separated by 2 non-2-vertices, whereas in the subtree generated by x, y, p, q in T_1 they are separated by only one non-2-vertex. Hence these two subtrees can not be homeomorphic. See figure 1.

Now remove the path from z to y in both trees. Inductively, the resulting two trees are homeomorphic. Clearly, then, adding the path from z to y in both T_1 and T_2 will still yield two homeomorphic trees. □

Let $S4$ be the set of all 4-element subsets of S that do not generate a homeomorphic subtree in all trees. $S4$ can be constructed in time $O(kn^5)$. A set $S' \subset S$ is a *cover* of $S4$ if $\forall \{a, b, c, d\} \in S4 \exists e \in S'$ such that $e \in \{a, b, c, d\}$. S' is a *minimum cover* if S' is a cover of $S4$ and every other cover is at least as large as S'.

Theorem 2 implies that if S' is a minimum cover of $S4$ then the homeomorphic restriction of $S - S'$ in all trees is a maximum agreement homeomorphic subtree.

The following algorithm is similar to the approximation algorithm for vertex cover [8].

Approximation Algorithm

1. $S_a \leftarrow \emptyset$
2. While $S4 \neq \emptyset$ do
   - 2.1 Choose an element $\alpha \in S4$
   - 2.2 $S_a \leftarrow S_a \cup \alpha$
   - 2.3 Delete from $S4$ all elements β such that $\alpha \cap \beta \neq \emptyset$

end Algorithm

Algorithm Time: $O(|S4|) = O(n^4)$ (The construction of $S4$ requires an additional $O(kn^5)$ time).

Theorem 3: The set S_a produced by the approximation algorithm is a cover of $S4$ and is at most 4 times the size of a minimum cover of $S4$.

Proof: S_a is clearly a cover. Let A be the set of all elements of $S4$ chosen by step 2.1. For any $\alpha_1, \alpha_2 \in A$, $\alpha_1 \cap \alpha_2 = \emptyset$. Thus any cover of $S4$ has at least $|A|$ elements. S_a has $4|A|$ elements. □

4 Maximum Isomorphic Agreement Subtree

For simplicity's sake, we start by presenting an algorithm for the MIWT problem for k trees, where the edge weights are uniformly 1. We call this the Maximum Isomorphic Agreement Subtree Problem (MIT), formally defined below. The general MIWT problem is handled in section 7.

Definition: Let $T_1, \ldots, T_k$ be S-labeled trees. A *maximum isomorphic agreement subtree* of $T_1, \ldots, T_k$ ($MIT(T_1, \ldots, T_k)$) is a maximum set $S' \subseteq S$ such that $\forall s, t \in S'$ the distance between s and t is the same in all trees $T_1, \ldots, T_k$.

The following tree property, due to Smolenskii [16], establishes that the subtrees induced by $MIT(T_1, \ldots, T_k)$ in $T_1, \ldots, T_k$ are indeed isomorphic.

Theorem 4: Two labeled trees are isomorphic iff the distance between any two leaves with corresponding labels is the same in both trees.

To further elucidate our idea, let us restrict ourselves to the case of binary trees $T_1, \ldots, T_k$. We also introduce the following notation. Assume we fix a node labeled r as the root. Since r appears in all our trees, we may now consider $T_1, \ldots, T_k$ as directed trees rooted at r. Since all edges are now directed, there is a unique subtree rooted at every node. Let x be a node in T_i. Denote the subtree rooted at x by $T_i(x)$. We denote the distance of node x from the root as a superscript, i.e. if $d(x, r) = p$ then we write x^p.

The main idea behind our algorithm is to use a top-down approach. We will show that the structure of our problem prevents an exponential time blow-up.

The first crucial observation we make is that there is at least one label in $MIT(T_1,\ldots,T_k)$. If we had *apriori* knowledge of such a label, say r, we could root all the trees at r. This fixes the direction of the edges, so for the next levels in the recursion there is only one possible root for every subtree.

The following lemma is also an important factor in understanding our algorithm.

Lemma 5: Let s and t be leaves of $MIT(T_1,\ldots,T_k)$. Suppose there is a tree T_i and $x_i^p \in T_i$ such that $s,t \in T_i(x_i^p)$. Then for every $j=1,\ldots,k$ there exists a node $x_j^p \in T_j$ for which $s,t \in T_j(x_j^p)$.

Proof: Consider the meeting point of the three pairwise paths of the leaves s,t and r. Since s,t,r are leaves of $MIT(T_1,\ldots,T_k)$, then Theorem 4 says their distances from each other are the same in all the trees $T_1,\ldots,T_k$. These distances $d(s,t)$, $d(s,r)$ and $d(t,r)$ determine that the distance from r to the meeting point is $(d(r,s)+d(r,t)-d(s,t))/2 = p$ in all trees. $\square$

Algorithm Outline

for $r = s_1$ to s_n do

    root all trees at r.

    find $MIT(T_1,\ldots,T_k)$ (A recursive algorithm for rooted MIT follows).

choose the largest of all n rooted MIT's.

end Algorithm

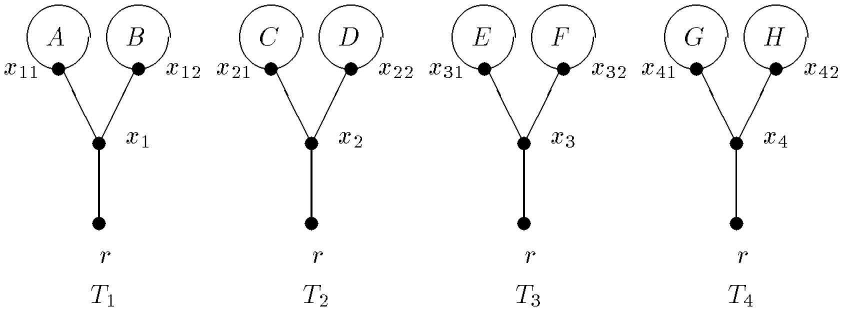
Before we proceed to the algorithm for rooted MIT, we define the concept of *thread intersection*. This is a key concept that appears in many of our algorithms. It is properties of these intersections that prevent an exponential time blowup of the algorithms.

We first intuitively describe the concept. Suppose a set of k nodes $x_i \in T_i,\ i=1,...,k$ is given. Each of these nodes may have several children. A thread is a choice of one son of each of the x_i's.

Definition: Let $\{x_1,\ldots,x_k\}$ be a fixed set of nodes where $x_i \in T_i,\quad i=1,\ldots,k$, and let $L(x_i) \subseteq S$ be the set of labels in $T(x_i)$ (the subtree rooted at x_i). Let $\{x_{i1},...,x_{ip_i}\}$ be the set of children of x_i.
A k-tuple $\sigma_{\langle x_1,\ldots,x_k\rangle} = \sigma[1],...,\sigma[k]$ where $\sigma[i] \in \{1,...,p_i\}$ is called a *thread* for nodes $x_1,...,x_k$. (In other words, a thread for a set of k nodes is a pointer from each of the k nodes to a specific one of its children.) Wherever the set of nodes $\{x_1,\ldots,x_k\}$ is clear from the text we will drop the index $\langle x_1,\ldots,x_k\rangle$ and denote the thread simply by σ. A *non-empty* intersection $\cap_{i=1}^{k} L(x_{i\sigma[i]})$ is a *thread intersection.*

For an example see figure 2.

Let A be the union of all thread intersections for a fixed set of nodes. The following simple observation guarantees that the non-empty thread intersections form a partition of A.



The thread $\sigma = \langle 1,1,2,1\rangle$ for nodes x_1, x_2, x_3, x_4 defines thread intersection $A \cap C \cap F \cap G$.
The thread $\overline{\sigma} = \langle 2,2,1,2\rangle$ for nodes x_1, x_2, x_3, x_4 defines thread intersection $B \cap D \cap E \cap H$.

Figure 2: Examples of threads.

Lemma 6: Let σ_1, σ_2 be threads for nodes $x_1, \ldots, x_k$. If $\sigma_1 \neq \sigma_2$ then

$$\left(\bigcap_{i=1}^{k} L(x_{i\sigma_1[i]})\right) \cap \left(\bigcap_{i=1}^{k} L(x_{i\sigma_2[i]})\right) = \emptyset.$$

Proof: If $\sigma_1 \neq \sigma_2$ then there is a j such that $\sigma_1[j] \neq \sigma_2[j]$.
$\cap_{i=1}^{k} L(x_{i\sigma_1[i]}) \subseteq L(x_{j\sigma_1[j]})$
$\cap_{i=1}^{k} L(x_{i\sigma_2[i]}) \subseteq L(x_{j\sigma_2[j]})$
But $\sigma_1[j] \neq \sigma_2[j]$ therefore $L(x_{j\sigma_1[j]}) \cap L(x_{j\sigma_2[j]}) = \emptyset$, thus

$$\left(\bigcap_{i=1}^{k} L(x_{i\sigma_1[i]})\right) \cap \left(\bigcap_{i=1}^{k} L(x_{i\sigma_2[i]})\right) = \emptyset. \qquad \square$$

The idea behind our algorithm is the straightforward one. Assume we are guaranteed that node r is in a MIT. We now root all the trees at r. Theorem 4 and lemma 5 now assure us that if other nodes appear in the MIT then the child x_i of r in each tree T_i, $i = 1, \ldots, k$ has to be in the MIT also.

Thus a naive way of constructing the MIT is by considering all possible combinations of children of the x_i's, $i = 1, \ldots, k$. In particular, if all the trees are binary trees, assume the children of x_i are x_{i1} and x_{i2}. For each thread σ of the nodes $x_1, \ldots, x_k$ recursively find a MIT T_1 of $\cap_{i=1}^{k} L(x_{i\sigma[i]})$ and a MIT T_2 of $\cap_{i=1}^{k} L(x_{i\overline{\sigma}[i]})$. Construct a tree whose root is r, its child is x_1 and make the roots of T_1 and T_2 the children of x_1. Each such tree is an isomorphic agreement subtree. The largest of them is a maximal isomorphic agreement subtree.

The naive algorithm described above has exponential running time since there are 2^k combinations to check at each level of the recursion. The algorithm below is essentially the same as the naive algorithm but avoids the need to check an exponential number of thread combinations by exploiting the fact that the threads form a partition hence there are only very few non-empty thread intersections. We only check the non-empty ones.

Recursive Algorithm for Finding Rooted MIT of Binary Trees

1. prune all trees to include only the labels that are equidistant to r in all trees.

   The trees are now of the following form: Tree T_i has root r. r has one child x_i^1. x_i^1 has two children x_{i1}^2 and x_{i2}^2. The rest of the tree is rooted at x_{i1}^2 and x_{i2}^2. We would like to recurse and choose a maximal isomorphic agreement subtrees of all combinations of children. However, there are 2^k combinations, which we can not afford. The partition property of thread intersections means there are at most $n-1$ non-empty candidate intersections.

2. construct the i disjoint thread intersections $\{S_1, \ldots, S_i\}$ of $S - \{r\}$ $(i \leq n-1)$.
   Details of the construction appear in the implementation of step 2 below.

   Let the thread intersections be of sizes $n_1, \ldots, n_i$, respectively. $(\sum_{j=1}^{i} n_j \leq n-1.)$ Let T_{jl} be the minimal subtree of T_j rooted at x_j and containing exactly the leaves in $T_j \cap S_l$. In the next level of the recursion, x_j has only the single child appearing in the chosen thread. The role of r will be played by x_j. We continue using only the non-root leaves of the tree as our counting basis, with the root never contributing.

   construct i tuples $\langle T_{11}, \ldots, T_{k1} \rangle, \ldots, \langle T_{1i}, \ldots, T_{ki} \rangle$.

   recursively find the MIT of each of the i tuples.

3. The MIT will be composed of putting together the MITs of different thread intersections. Since our trees are binary trees, a choice of thread σ forces at most one possibility for its counterpart, the thread $\bar{\sigma}$ such that $\bar{\sigma}[i] = 3 - \sigma[i]$. Note that $\bar{\sigma}$ may be empty either because of an empty intersection or even because some x_i has only a single child.
   Let $MIT(\sigma) = MIT(T_{1t}, T_{2t}, ..., T_{kt})$ where $S_t = \cap_{l=1}^{k} L(x_{l\sigma[l]})$.
   Pair all threads into $\{\langle \sigma_1, \bar{\sigma_1} \rangle, ..., \langle \sigma_{i'}, \bar{\sigma_{i'}} \rangle\}$ where each pair has at least one thread with a non-empty intersection. It is clear that $i' \leq i$. (Recall that i is the number of non-empty disjoint thread intersections found in Step 2.)
   Each $MIT(\sigma)$ is a binary tree whose root has a single child. Let $MIT(\langle \sigma, \bar{\sigma} \rangle)$ be the tree resulting from merging the roots of $MIT(\sigma)$ and $MIT(\bar{\sigma})$ into a single node that has two children. This node is the single child of r.

   choose $MIT(T_1, \ldots, T_k)$ as the largest of $\{MIT(\langle \sigma_1, \bar{\sigma_1} \rangle), ..., MIT(\langle \sigma_{i'}, \bar{\sigma_{i'}} \rangle)\}$.

end Algorithm

Correctness: The algorithm's main action takes place in step 2. The fact that the MIT is an isomorphic subtree of all trees means that each of $x_1^1, x_2^1, \ldots, x_k^1$ is the child of r in the MIT restricted to $T_1, \ldots, T_k$, respectively. The MIT is then the best combination of children of the x_i's. The partition property of the thread intersections limits the number of recursive cases that need to be considered to $i < n$ rather than 2^k since all other thread intersections are necessarily empty.

Step 3 is correct because in a binary tree every choice of right or left, forces a single remaining option. □

Implementation of Step 2: Our goal is to decide in polynomial time which of the 2^k possible threads gives a non-empty thread intersection. When constructing a thread, at each of the k trees we make one choice. Either we choose a 1 (if x_{i1} is in the thread) or a 2 (x_{i2} in the thread). Constructing all these choices is equivalent to a breadth-first-search of the complete depth-k binary tree where each node has a left son labeled 1 and a right son labeled 2. Visiting child e, $e \in \{1, 2\}$ at level d means adding x_{de} to the thread. We will in fact be intersecting with $L(x_{de})$. We are guaranteed that the non-empty sets are disjoint at every node at every fixed level. We do not continue the search at any branch where the empty set is reached. The sum of all elements in the sets for each level is at most $n-1$, thus the total time to construct step 2 is $O(kn)$.

Algorithm Time: The time for computing the MIT of k rooted trees is given by the following recurrence:

$$f(1) = 1$$

$$f(n) = f(n_1) + f(n_2) + \cdots + f(n_i) + kn, \quad \text{where } \sum_{l=1}^{i} n_l = n.$$

The closed form is $f(n) = O(kn^2)$.

Total Algorithm Time: Since the rooted MIT algorithm has to be run n times, the total time is $O(kn^3)$.

It is easy to see that the the above algorithm will also work for trees where T_1 has degree bounded by $d > 2$. The only use we made of the fact that the trees are binary was in step 3. In the binary tree case, choosing a σ, fixes $\overline{\sigma}$ for purposes of recursively constructing a MIT. Now, however, we need to make sure that we pick the maximum of the choice of all non-conflicting threads, where a thread σ_1 conflicts with a thread σ_2 of the same nodes if $\exists i$ such that $\sigma_1[i] = \sigma_2[i]$. Obviously, two conflicting threads can not be joined together in the same agreement subtree. There are going to be $\binom{n}{d-1}$ possibilities. The non-conflicting threads may be put together in a brute-force fashion. Each of the $\binom{n}{d-1}$ possibilities requires $O(kd)$ time for verification of non-conflict (since there are at most $d-1$ threads in a combination, and each thread is of length k). Thus all non conflicting threads can be generated in time $O(k(d-1)n^{d-1}/(d-1)!) = O(kn^{d-1})$. We now choose the best MIT over all nonconflicting thread combinations.

The recurrence is now:

$$f(1) = 1$$

$$f(n) = f(n_1) + f(n_2) + \cdots + f(n_i) + kn^{d-1}, \quad \text{where } \sum_{l=1}^{i} n_l = n.$$

Thus the total algorithm time for degree bounded by d trees is $O(kn^{d+1})$. It should be noted that the algorithm does not require all trees to have bounded degree. It will work even for the case where only **one** tree has boundeed degree d. However, if all trees have degree bounded by d then each set of $d-2$ threads forces a unique $d-1^{th}$ thread (as in the binary tree case above) and the total algorithm time is then $O(kn^d)$.

5 Maximum Homeomorphic Agreement Subtree

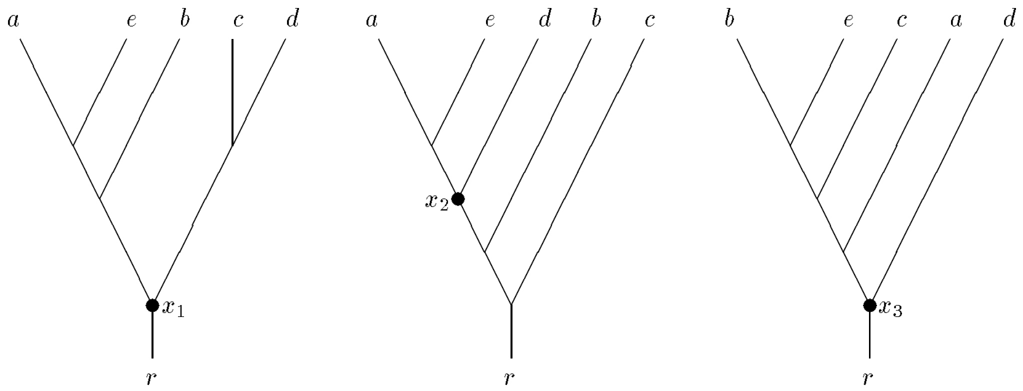
We now consider the task of finding a maximum homeomorphic agreement subtree of k trees, where the degree of at least one of the trees is bounded.

We follow the same ideas as before. We basically use a naive brute-force algorithm that normally produces an exponential time solution. However, we will define special sets of labels - *maximal decomposable sets* - and show that considering these sets alone is sufficient for finding a MHT. Since there is only a polynomial number of such sets, our algorithm's time is therefore polynomial.

Before presenting the algorithm, we need some more definitions and observations.

Notation: Let $A \subseteq S$. We denote the tree $MHT(H(T_1, A), ..., H(T_k, A))$ by $MHT(A)$. To avoid unnecessary notation, we will also use the notation $MHT(A)$ to denote the set of labels of $MHT(A)$.

Definition: A set $A \subseteq S$ is *decomposable* if there exist $x_1 \in T_1, ..., x_k \in T_k$ such that each $x_i,\ i = 1, ..., k$ has at least p children which can be labeled $x_{i1}, ..., x_{ip},\ p \geq 2$, and such that $A = A_1 \cup \cdots \cup A_p$, where A_j are mutually disjoint nonempty sets and $A_j \subseteq L(x_{ij}),\ j = 1, ..., p;\ i = 1, ..., k$. We say that *$A$ decomposes to* $A_1 \cup \cdots \cup A_p$, and that $x_1, ..., x_k$ are the *roots of the decomposition.*



$\{a, e, d\}$ is a maximal decomposable set that decomposes to $\{a, e\} \cup \{d\}$.
The roots of the decomposition are x_1, x_2, x_3.

Figure 3: Example of a decomposable set.

In words, there is a node in each one of the trees, where the label set is split among p children in the same fashion. See figure 3.

Note that the number of decomposable sets may still be exponential. Consider as an example two identical complete binary trees. Every label subset with more than two leaves is a decomposable set. We would like to limit the number of label sets that need to be considered.

Definition: Let A be a decomposale set which decomposes to $A_1 \cup \cdots \cup A_p$. A is a *maximal decomposable* set if there does not exist a decomposable set B such that A is a proper subset of B and such that B decomposes to $B_1 \cup \cdots \cup B_q$, $q \geq p$ where every A_i, $i = 1, ..., p$ is a subset of a distinct B_j.

An alternate way of viewing maximal decomposable sets is by considering the roots of the decomposition. A set of nodes $x_1 \in T_1, ..., x_k \in T_k$ defines a set of maximal decomposable sets. Each of these maximal decomposable sets is defined by a maximum p ($p \geq 2$) and an ordered list of p children of each x_i, which we will label $x_{i1}, ..., x_{ip}$ (although note that the order of these nodes is not necessarily their left-to-right order in the trees). If $\cap_{i=1}^{k} L(x_{ij}) \neq \emptyset$, $j = 1, ..., p$ then $\cup_{j=1}^{p} \cap_{i=1}^{k} L(x_{ij})$ is a maximal decomposable set.

The maximality, then, is the fact that this set is the largest decomposable set defined by the given set of nodes. This limitation reduces the number of sets from exponential to polynomial, as can be seen from the following theorem.

Theorem 7: Let $T_1, ..., T_k$ be a set of trees where T_1 has degree bounded by d. Assume the trees are rooted at a common label r. Then there are $O(n^{d-1})$ maximal decomposable sets.

Proof: We know that T_1 is a degree d tree, so in our case, no set can decompose to more than $d-1$ subsets. Observe that if A decomposes to $A_1 \cup \cdots \cup A_{d-1}$, then for every $(d-1)$-tuple $\langle a_1, a_2, ..., a_{d-1} \rangle$ where $a_i \in A_i$, $i = 1, ..., d-1$, the lowest common ancestor of the $d-1$ elements in each of the k tree is the node defined as the root of the decomposition in that tree. If, however, we are considering a maximal decomposable set, then each such tuple uniquely defines the set. The reason is that the LCA of the tuple finds the root of the decomposition, and the maximum subset in each of the $d-1$ children that appears in all k trees is the maximal decomposition. □

The following observation guarantees the polynomial time of our algorithm.

Observation 8: If $A \subseteq S$ is a maximal decomposable set decomposable to $A = A_1 \cup \cdots \cup A_p$ then $MHT(A) = MHT(A_1) \cup \cdots \cup MHT(A_p)$. If A is not decomposable then $MHT(A) = \max\{MHT(B) | B \subseteq A \text{ and } B \text{ is maximal decomposable}\}$

Proof: Let A be maximal decomposable, and let H be the homeomorphic restriction of T_i to $A \cup \{r\}$. The root of H is r. Each x_i, the root of the decomposition in T_i, is the restriction of the only child of r in H. It is clear that the only chance of getting a non-empty MHT is by matching equal subsets to each other. The maximum number we can get is by matching *all* equal subsets to each other.

If A is not maximal decomposable then for each i, $i = 1, \ldots, k$, let x_i be the node in T_i that is the restriction of r's child in the MHT. Because $x_1, \ldots, x_k$ appear in the MHT then they are either leaves or each has children $x_{i1}, \ldots, x_{ip}$, $p > 1$ where $L(x_{ij}) \cap L(x_{lj}) \neq \emptyset$. Thus if B is the maximal decomposable set of A for nodes $x_1, \ldots, x_k$ then $MHT(A) = MHT(B)$ and $B \subseteq A$. □

The above observation means that it is sufficient to compute the MHT of maximal decomposable sets.

We now present an algorithm for finding the MHT of a set of trees where tree T_1 (which we may assume has the smallest degree) has degree bounded by d. As in the previous sections we present the part that assumes root r. The general algorithm chooses as the MHT the largest tree produced by each of the rooted cases.

For simplicity of the presentation, we will separate the algorithm into two parts. One part that constructs all maximal decomposable subsets of $S - \{r\}$, and a second part that constructs the MHT of all maximal decomposable subsets.

Algorithm for Constructing all Maximal Decomposable Subsets (degree bounded by d tree)

For each $(d-1)$-tuple $\langle a_1, ..., a_{d-1} \rangle$ of elements from $S - \{r\}$ do:

1. For every tree T_i, $i = 1, ..., k$ find if there is $x_i \in T_i$ that has $d-1$ distinct children $x_{i1}, ..., x_{i(d-1)}$ where $a_j \in L(x_{ij})$, $i = 1, ..., d-1$.
2. If such an x_i is found for every T_i, $i = 1, ..., k$ then construct the maximal sets $A_1, ..., A_{d-1}$ such that $a_j \in A_j \subseteq L(x_{ij})$, $i = 1, ..., k$; $j = 1, ..., d-1$.
   Take $A = A_1 \cup \cdots A_{d-1}$ as a maximal decomposable set.

end

end Algorithm

Time: The algorithm loops n^{d-1} times. It takes time $O(kn)$ to find the x_i's (step 1) and to construct the A_i's (step 2). The total time is $O(kn^d)$.

Correctness: By theorem 7.

We are now ready to present the algorithm for finding MHT of k trees one of which has degree bounded by d.

Algorithm for Rooted MHT (degree bounded by d tree)

1. Construct all n^{d-1} maximal decomposable sets.
2. Order the maximal decomposable sets by the subset relation.
3. The base subsets (those that have no maximal decomposable subsets) are sets $A = A_1 \cup \cdots \cup A_{d-1}$ where each of the A_i's, $i = 1, ..., d-1$, is a singleton set $\{a_i\}$. $MHT(A)$ is A.
   For subsequent maximal decomposable subsets, construct the MHT by the relation $MHT(A) = MHT(A_1) \cup$

$MHT(A_2) \cup \cdots \cup MHT(A_{d-1})$. For any maximal decomposable A_i, we have already computed the MHT. If any of the A_i is not maximal, its MHT is that of its largest maximal decomposable subset.
$MHT(S)$ is $MHT(A)$, where A is the largest maximal decomposable subset of S.

end Algorithm

Time: Step 1 was already seen to take time $O(n^d)$. For steps 2 and 3, there are $O(n^{d-1})$ maximal decomposable sets. comparing every two sets takes time $O(n)$. A straightforward implementation of steps 2 and 3 is by a pairwise comparison of all maximal decomposable sets. This can be done in time $O(n^{2d-1})$. The total algorithm time for rooted trees is then $O(kn^d + n^{2d-1})$. For unrooted trees the time is $O(kn^{d+1} + n^{2d})$.

Correctness: Clear by Observation 8.

6 Maximum Weighted Agreement Subtree

A MHT of a set of trees, produces a tree whose **shape** is common to all trees. However, this does not take into account the conjectured **time** each evolutionary split took. Our final goal is a tree where the distances between nodes are conjectured, i.e. given as a **range** of numbers. However, we start with the case where the distances between nodes in each of the evolutionary trees are given exactly. This case is a special case of the maximum interval subtree discussed in the next section. However, we treat is separately for two reasons: 1) It introduces some concepts that are used later in the interval case. 2) It has a faster algorithm than the general case.

Definition: Let $T_1, \ldots, T_k$ be S-labeled trees with integer weights on the edges. A *maximum weighted agreement subtree* of $T_1, \ldots, T_k$ ($MWT(T_1, \ldots, T_k)$) is a maximum set $S' \subseteq S$ such that $\forall s, t \in S'$ the distance between s and t (the sum of the weights on the edges of the path between them) is the same in all trees $T_1, \ldots, T_k$. We will also refer to the problem of finding a maximum weighted agreement subtree as *the MWT problem.*

This is a generalization of the MIT problem (there every edge had weight 1). However, we can not immediately reduce the MWT problem to the MIT problem by replacing every weight-i edge by i weight-1 edges, since then the problem size may grow exponentially.

Let T_1 and T_2 be two S-labeled trees with integer weights on the edges. T_1 and T_2 are *w-isomorphic* if the distance between every two labeled edges is the same in both trees.

Theorem 9: Two labeled trees with no *2-vertices* (vertices of degree 2) are w-isomorphic iff they are isomorphic as leaf-labeled trees and corresponding edges have the same weight.

Proof: For rational weights, this is an immediate corollary of theorem 4. □

Theorem 9 asserts that in different trees of the given set, subtrees corresponding to the MWT may differ only in edges incident to 2-vertices. In addition, the sum of the weights of a maximal chain of 2-vertices (whose end points are not 2-vertices) must be the same for all corresponding pairs of non-2-vertices in all the trees.

For the sake of exposition we again assume that all our trees are binary trees. The algorithm outline is generally unchanged from the MIT case. The only exceptions are the following necessary changes.

- As in the MIT case, the recursion always treats rooted trees where the root has a single child x and that child has two children x_1 and x_2. Unlike the MIT case, our root is **always** taken to be r and the weight of the edge from r to x is the sum of the weights from r to x in the original tree.
- If the children of r in all the trees have the same distance to r the case is handled exactly like the MIT algorithm.
- If there are at least two children of r with different distances to r, let i be the tree with the smallest distance $d(x_i, r)$. Because of Theorem 9, it is impossible for x_i to be in a MWT as a 3-vertex. Therefore we split the tree $T_i(x_i)$ into two trees $T_i(x_{i1})$ and $T_i(x_{i2})$, where x_{i1} and x_{i2} are x_i's children. We now have two MWT problems of smaller size.

Termination: The algorithm terminates since a tree split always reduces the size of the remaining trees. Since there are kn nodes initially, and every split reduces the size of the trees by at least one node, there can be no more than kn splits.

Time: The time complexity of the algorithm does not change. Although we may split the problem several times during the course of the algorithm, the sizes of the problems are reduced with each split. Because of convexity of the time function, the complexity is worst when there is no split. This is exactly the MIT case.

Total Algorithm Time: $O(kn^3)$.

Total Algorithm Time for degree bounded by d Trees: $O(n^d + kn^3)$.

7 Maximum Interval Agreement Subtree

A MHT of k trees provides, in a sense, the greatest number of leaves whose underlying tree structure has a similar shape. A MWT provides a maximum set of leaves where the exact distances between them are preserved. The big problem is that the exact distances are not usually known. We would like a metric that gives us the advantage of both structure and "approximate" distance. We approximate the distance between two nodes by a pair of numbers $[a, b]$ where $a \leq b$. This pair represents the interval between a and b and its meaning is that the distance between the edge's head and tail is within that interval. We are interested in the largest set of leaves that has an underlying tree with edge weights in the intersection of all appropriate intervals. Formally:

Definition: Let $T_1, \ldots, T_k$ be S-labeled trees. Assume that every edge is labeled by a pair of numbers $[a, b]$, where $a \leq b$. We call such trees *S-labeled interval labeled trees.* An *instantiation* T' of an S-labeled interval labeled tree T is an S-labeled weighted tree (as defined in section 6) where all nodes and edges of T' are equal to the nodes and edges of T and where every edge of T' is labeled by a number in the interval label of that edge in T.

A *maximum interval agreement subtree* of $T_1, \ldots, T_k$ ($MIWT(T_1, \ldots, T_k)$) is a maximum set $S' \subseteq S$ such that $MIWT(T_1, \ldots, T_k)$ is $MWT(T'_1, \ldots, T'_k)$, where T'_i is an instantiation of T_i for $i = 1, \ldots, k$. We refer to the problem of finding a maximum interval agreement subtree as *the MIWT problem.*

Note that all previous agreement metrics are special cases of MIWT. In MIT, we assume all intervals are $[1, 1]$. In MWT, any edge weight a is interval $[a, a]$. In MHT, we assume all intervals are $[1, n-1]$. We presented the case separately for historical and methodological reasons.

The key observation for the MIWT problem is that theorem 9 guarantees that a MIWT of k trees is also a homeomorphic agreement subtree of those trees (because it is a MWT of some instantiation of them). Once this observation is made the following algorithm follows naturally. Before presenting the algorithm, we need one more definition.

Definition: The *distance interval* between node x and y in an interval labeled tree is the interval $[z, w]$ where z is the sum of the smaller elements of each of the pairs on the path from x to y, and w is the sum of the larger elements.

Algorithm for MIWT

We modify the MHT algorithm to return MIWT. The only necessary modification is the following:

1. For each homeomorphic agreement subtree T' handled by the algorithm do
   (a) for each tree T_i, $\quad i = 1, \ldots, k$ do
       for every path in T_i corresponding to an edge in T' compute the distance interval of that path
   (b) { Every edge in T' now has k distance intervals associated with it. }
       for every edge in T' where the k distance intervals have a non-empty intersection, discard the edge and the entire subtree rooted to its head

end Algorithm

Theorem 10: The above algorithm computes a MIWT of k trees in time $O(kn^{d+1} + n^{2d})$, where there is at least one tree of degree bounded by d.

Proof: Correctness follows immediately from the observation that every MIWT is also a homeomorphic agreement subtree.

We need to prove the time bounds. The added modification can be implemented in time $O(kn)$ and needs to be added for every maximal decomposale set. Since there are $O(n^d)$ such sets, the asymptotic time complexity of the algorithm does not change, and is $O(kn^{d+1} + n^{2d})$. □

8 Future Work

Although the $\mathcal{NP}$-completeness proof of the unbounded degree case prepares us for having the degree d as an exponent in our time complexity, it would be nice to reduce the time for finding the MWT to $O(c^d p(n, k))$, where p is a polynomial, rather than $O(n^d)$.

We presented here an approximation for the problem of finding the complement of the MHT. In [13] it was proven that the MHT problem for three trees with unbounded degree can not be approximated within ratio n^ϵ for any constant $\epsilon \leq 1$.

For the case of bounded degree trees, the time was improved by [5].

From a graph theoretic aspect, it would be interesting to find a maximum agreement subtree in the sense of maximizing the number of edges, while using true contraction. The number is no less than n because every tree can be contracted to a star. However, again in [13] it was shown that this prolem is $\mathcal{NP}$-hard even for two trees.

An ambitious problem is to define "tree closeness" metrics and efficiently find the closest tree to a set of trees keeping all the leaves.

Acknowledgements

We wish to thank Mike Steel for his great help in refining this paper, Bob Robinson for kindly sending us reference material on the subject, and two anonymous referee who were very fast and insightful.

References

[1] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms. *Proc. FOCS 94*, pages 758–769, 1994.

[2] H. J. Bandelt and A. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Adv. Appl. Math*, 7:309–343, 1986.

[3] W. H. E. Day. Optimal algorithms for comparing trees with labeled leaves. *Journal of Classification*, 2:7–28, 1985.

[4] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Proc. 25th Annual ACM Symposium on the Theory of Computing*, pages 137–145, 1993.

[5] M. Farach, T. M. Przytycka, and M. Thorup. Computing the agreement of trees with bounded degrees. *Proc. 3rd European Symposium on Algorithms*, 1995.

[6] M. Farach and M. Thorup. Fast comparison of evolutionary trees. *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 481–488, 1994.

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

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Co., 1979.

[9] W. Goddard, E. Kubicka, G. Kubicki, and F. R. McMorris. Agreement subtrees, metric and consensus for labeled binary trees. DIMACS Workshop on Partitioning Data Sets, 1993.

[10] A. D. Gordon. A measure of the agreement between rankings. *Biometrika*, 66:7–15, 1979.

[11] A. D. Gordon. On the assessment and comparison of classifications. In R. Tomassone, editor, *Analyse de Données et Informatique*, pages 149–160. INRIA, 1980.

[12] J. A. Hartigan. *Clustering algorithms*. New York: John Wiley, 1975.

[13] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. manuscript, 1995.

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

[15] D. A. Neumann. Faithful consensus methods for n-trees. *Bulletin of Mathematical Biology*, 63:271–287, 1983.

[16] E. A. Smolenskii. *Jurnal Vicisl. Mat. i Matem. Fiz*, 2(2):371–372, 1962.

[17] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

[18] M. A. Steel and D. Penny. Distributions of tree comparison metrics - some new results. *Syst. Biol.*, 42:126–141, 1993.