

# Approximation and Special Cases of Common Subtrees and Editing Distance<sup>\*</sup>

Magnús M. Halldórsson<sup>1</sup> and Keisuke Tanaka<sup>2</sup>

<sup>1</sup> Science Institute, University of Iceland  
IS-107 Reykjavik, Iceland  
mmh@rhi.hi.is

<sup>2</sup> School of Information Science,  
Japan Advanced Institute of Science and Technology – Hokuriku  
Tatsunokuchi, Ishikawa 923-12, Japan  
tanaka@jaist.ac.jp

**Abstract.** Given two rooted, labeled, unordered trees, the *common subtree* problem is to find a bijective matching between subsets of vertices of the trees of maximum cardinality which preserves labels and ancestry relationship. The *tree editing distance* problem is to determine the least cost sequence of additions, deletions and changes that converts a tree into another given tree. Both problems are known to be hard to approximate within some constant factor in general.

We present polynomial algorithms for several special classes of trees as well as a tighter approximation hardness proof, which together comes close to characterizing the complexity of both problems on all interesting special classes of trees. We also present the first approximation algorithm with non-trivial approximation ratios. In particular, we achieve a ratio of  $\log^2 n$ , where  $n$  is the number of vertices in the trees.

## 1 Introduction

An important part of computer science is in detecting and efficiently recognizing similarities among data sets or changes thereof. A natural measure of the difference between data sets is in the minimum cost sequence of atomic changes (or editing operations) that transform one into another.

One of the more pervasive structures of data are trees. We consider in this paper the *tree editing distance* problem and the *largest common (hereditary) subtree* problem, which model the difference and the similarity of two trees, respectively.

Given two rooted, labeled, unordered trees, the *common subtree* problem is to find a bijective matching between subsets of vertices of the trees of maximum cardinality which preserves labels and ancestry relationship.

---

<sup>\*</sup> Work done in large part while the first author visited JAIST, first as PFU Visiting Chair, and later supported by JAIST Foundation. Also work done in part while the second author visited NTT Telecommunication Networks Laboratories under the supervision of Toshihiko Yamakami.

The *tree editing distance* problem is to determine the least cost sequence of additions, deletions and changes that converts one tree into the other input tree. Deletion of a vertex  $v$  not only removes  $v$  but makes the parent of  $v$  become the new parent of the children of  $v$ . Addition of a vertex  $v$  similarly makes  $v$  become the parent of some subset of the children of the new parent of  $v$ . A change operation changes only the label of a vertex. Each of these operation carry a cost function that depends on the label of the respective vertices.

These problems have applications in various fields. Telecommunications networks are often in the form of rooted, labeled, unordered trees, and the rapid changes in active element have forced the introduction of automatic recognition in network design systems. Such similarity recognition problem occur also frequently in biocomputing, since different sequences (e.g. molecules, RNA) of the same type are never completely identical.

The editing distance and common subtree problems of two trees have been shown to be NP-hard, and thus no efficient algorithm are to be expected. The current paper deals with two ways of dealing with the intractability of the problem: efficient algorithms for important special classes of trees, and approximation algorithms with good performance bounds for general trees.

**Special classes of trees** Given the NP-hardness of the common subtree problem, it is valuable to study the complexity when the inputs are restricted to important special classes of trees.

We consider the following restricted classes of trees:

**Star** Trees where the root is the only internal node.

**Few internal nodes** Trees with some constant number of internal nodes. A superclass of *stars*.

**Few branching nodes** A branching node is a node of degree three or more. A superclass of *few internal nodes*, and of paths.

**Generalized Caterpillar** Trees where all branching nodes are lie on a single path. A superclass of *stars*, *paths*, and trees with at most 3 branching nodes.

**Small height** Trees whose height is bounded by a constant. A superclass of *few internal nodes*.

We present algorithms based on dynamic programming that solve the general editing distance problem for the classes of *few internal nodes*, *few branching nodes* and *generalized caterpillars*. For the other direction, we show that the common subtree problem and the editing problem are NP-hard, as well hard to approximate, if one tree is of height one (viz. a star) and the other is of height two.

These results characterize completely the solvability of the common subtree problem of trees from any combination of these classes. Namely, the problem is polynomial solvable iff either both trees are from one of the first four classes listed above, or if one tree has bounded number of leaves.

**Approximation** Approximation algorithms are heuristics that compute solutions that are not necessarily optimal but are guaranteed to be within some ratio from the optimal bound. The *performance ratio* of an approximation algorithm

for the common subtree problem is the maximum, over all pairs of input trees, of the ratio between the size of the optimal common subtree and the size of the common subtree found by the algorithm.

We present the first approximation algorithm with sub-polynomial performance ratio. The ratio we achieve is  $\log^2 n$ . We also give an incomparable bound, that is twice the height of the shorter tree.

**Previous results** On computing the largest common subtree or editing distance between unordered labeled trees, the following results are known. It was shown in [3] that these problems are MAX SNP-hard for the general case, and thus cannot be approximated within some constant slightly larger than one. Previously, the problem had been shown NP-hard; even the restricted problem of deciding if the former tree was fully included in the latter tree. (Our reduction shows that the latter problem is also hard to approximate.)

A result for a special class of trees appears in [4], which contains an efficient polynomial-time algorithm for the case when one tree is a string (i.e. has only one leaf) or has a bounded number of leaves.

The only approximation result we are aware of is an incomplete manuscript of Yamaguchi [2]. It contains an algorithm for which a performance ratio of  $O(OPT/\log OPT)$  for the largest common subtree problem is claimed, where  $OPT$  is the size of the optimal solution. We note that this ratio is incomparable to our bounds, as it can be superior when the optimal solution is very small.

**Outline of paper** In Section 2 we present polynomial algorithms for special classes of trees, that apply both for the common subtree as well as the editing distance problems. We then give a tightened hardness result in Section 3. Section 4 contains several approximation algorithms for common subtrees of general trees. We then end with discussion of extensions to more general types of graphs.

## 2 Special classes of trees

We give in this section algorithms for computing the editing distance between two trees belonging to several special classes of trees. These can then easily be modified to output as well the largest common subtree.

We start with definitions of tree editing operations and their measures, and the equivalent concept of a mapping between subsets of nodes of the trees. We then give an algorithm based on dynamic programming for computing the editing distance between two trees with bounded number of internal nodes. These are then modified for the more general cases of trees with bounded number of branching nodes, and for generalized caterpillars.

**Editing operations and editing distance** We consider three kinds of operations: a) deleting a node  $v$ , where  $v$  is removed from the tree while the children of  $v$  become the children of the parent of  $v$ ; b) adding a node  $v$ , which is the complement of deleting, as  $v$  will become a parent of the subset of the children of the node that becomes the parent of  $v$ , and c) changing the label of a node.

These editing operations carry a cost function, that may depend on both the type of operation as well as the value of the label added/deleted/changed. We denote them by  $del(u)$ ,  $add(v)$ , and  $chg(u, v)$ , and assume that they satisfy a

*distance metric.* The editing distance between  $T_1$  and  $T_2$  is defined as the total cost of a minimum cost sequence of edit operations that transform  $T_1$  into  $T_2$ .

**Mappings** The editing distance is equivalent to finding a bijective (one-to-one) matching between subsets of the vertices of  $T_1$  and  $T_2$  that preserves the ancestor-descendant relationship.

For a mapping  $M$ , let  $I$  denote the non-participating vertices in  $T_1$ , and  $J$  the non-participating vertices in  $T_2$ . The cost of the mapping  $M$  is defined as:

$$\sum_{(u,v) \in M} \text{chg}(u,v) + \sum_{u \in I} \text{del}(u) + \sum_{v \in J} \text{add}(v).$$

The largest common subtree problem is the natural complement of the editing distance problem, where we try to *maximize* the weight of the vertices that are not changed (or maximize the savings over deleting all vertices from  $T_1$  and adding all from  $T_2$ ).

$$\text{Weight of common subtree} = \frac{1}{2} \left( \sum_{u \in T_1} \text{del}(u) + \sum_{v \in T_2} \text{add}(v) - \sum_{(u,v) \in M} \text{chg}(u,v) \right)$$

Some of the algorithms apply only to the unweighted case, where addition and deletion cost one unit and changes cost two units.

**Notation** Let  $F$  be a forest, i.e. a collection of trees, and let  $|F|$  denote the number of nodes in  $F$ . Let  $\text{roots}(F)$  denote a set of the roots of all the trees in  $F$ , and  $\text{tree}(u)$  denote the subtree rooted at  $u$ . Notice that, if  $T$  is a tree and  $v$  is the root of  $T$ , let  $T - v$  represent the forest obtained from  $T$  by removing  $v$ . The editing distance between  $F_1$  and  $F_2$  is denoted by  $\text{dist}(F_1, F_2)$ .

## 2.1 The case of bounded number of internal nodes

Let the two trees be  $T_1$  and  $T_2$  with constant number of internal nodes. Let  $F_1$  and  $F_2$  be two subforest obtained from  $T_1$  and  $T_2$  by deleting nodes, respectively. We now introduce an observation and a lemma which assure the validity of our algorithms.

Let  $M$  be one of the minimum cost mappings from  $F_1$  to  $F_2$  which induces the minimum length of edit operations. Let  $\overline{M}$  denote the set of vertices that does not appear in the matching. The following observation is obvious. Since, for example, if  $u \notin M$ , then root node  $u$  maps to no node and  $u$  should be deleted. When  $u$  is deleted from subtree  $\text{tree}(u)$  in  $F_1$ ,  $\text{tree}(u)$  becomes subforest  $\text{tree}(u) - u$ .

**Observation 1.**

$$\text{dist}(F_1, F_2) = \begin{cases} \sum_{u \in F_1} \text{del}(u) + \sum_{v \in F_2} \text{add}(v) & \text{if } F_1 = \emptyset \text{ or } F_2 = \emptyset, \\ \text{dist}(F_1 - u, F_2) + \text{del}(u) & \text{if } \exists u, u \in \text{roots}(F_1) \cap \overline{M} \\ \text{dist}(F_1, F_2 - v) + \text{add}(v) & \text{if } \exists v, v \in \text{roots}(F_2) \cap \overline{M} \end{cases}$$

Here we consider the following condition:

$$u \text{ is a leaf or } u \in M, \text{ for all } u \in \text{roots}(F_1) \cup \text{roots}(F_2) \quad (1)$$

We can show the following (proof omitted).

**Lemma 2.** *Under (1), if  $|roots(F_1)| \leq |roots(F_2)|$ , then each root in  $F_1$  maps to some root in  $F_2$ ; otherwise, each root in  $F_2$  is mapped from some root in  $F_1$ .*

**Algorithm** We now present a recursive algorithm **dist** with its sub-algorithm **distsub**, where  $I = \{u | u \in roots(T_1) \text{ and } u \text{ is an internal node}\}$  and  $J = \{v | v \in roots(T_2) \text{ and } v \text{ is an internal node}\}$ .

Algorithm **dist** recursively calculates distances considering three major cases: one of the two forests is empty, there exists an internal root node which is not in  $M$ , and all internal root nodes are in  $M$ .

```

dist( $F_1, F_2$ )
  if ( $F_1 = \emptyset$  or  $F_2 = \emptyset$ ) then
    return( $\sum_{u \in F_1} del(u) + \sum_{v \in F_2} add(v)$ )
  else
    Calculate  $dist(F_1 - u, F_2) + del(u)$  for each  $u \in I$ 
    Calculate  $dist(F_1, F_2 - v) + add(v)$  for each  $v \in J$ 
    Calculate  $distsub(F_1, F_2)$ 
    return the minimum of the three cases
end

```

Algorithm **distsub** treats the case that all internal root nodes are in  $M$ . It constructs a balanced bipartite graph, with vertices for each root of the forests along with padding vertices corresponding to deleting and inserting operations. The weight of a pair of roots then depends on if both roots are matched in  $M$ , or if one of the roots is not matched (in which case it is not an internal node). The solution is obtained by computing a maximum weight matching on this graph.

```

distsub( $F_1, F_2$ )
   $U \leftarrow roots(F_1)$ 
   $V \leftarrow roots(F_2)$ 
  W.l.o.g., assume  $|U| \leq |V|$ 
  if  $|U| < |V|$  then add  $|V| - |U|$  additional vertices to  $U$ .
   $E \leftarrow \{(u, v) | u \in U, v \in V\}$ 
  for each  $u \in U, v \in V$  do
     $w(u, v) \leftarrow \begin{cases} dist(tree(u) - u, tree(v) - v) + chg(u, v) & \text{if } u \in roots(F_1) \text{ and } v \in roots(F_2) \\ add(v) & \text{if } u \notin roots(F_1) \text{ and } v \text{ is a leaf in } F_2 \\ del(u) & \text{if } v \notin roots(F_2) \text{ and } u \text{ is a leaf in } F_1 \\ \infty & \text{otherwise} \end{cases}$ 
  return( $MinimumWeightBipartiteMatching(U, V, E, w)$ )
end

```

Then, we can compute the editing distance between the two trees by executing  $dist(T_1, T_2)$ . Correctness is obvious from Observation 1 and Lemma 2.

**Complexity** Let  $\tau(t_1, t_2)$  be the time complexity of the algorithm **dist** on  $T_1$  and  $T_2$  with  $t_1$  and  $t_2$  internal nodes, respectively. Then,

$$\begin{aligned}\tau(t_1, t_2) &\leq t_1 \cdot \tau(t_1 - 1, t_2) + t_2 \cdot \tau(t_1, t_2 - 1) + t_1 \cdot t_2 \cdot \tau(t_1 - 1, t_2 - 1) + O(n^3) \\ &= O(t_1! \cdot t_2! \cdot n^3).\end{aligned}$$

This is polynomial for  $t_1, t_2$  as large as  $O(\log n / \log \log n)$ .

## 2.2 The case of a bounded number of branching nodes

Let  $T_1$  and  $T_2$  be trees with constant number of branching nodes. Let  $F_1$  and  $F_2$  be two subforest obtained from  $T_1$  and  $T_2$  by deleting nodes, respectively.

The algorithm for the case for two trees of a bounded number of internal nodes can be extended to this case.

Let  $forest(U)$  denote a subforest containing all trees rooted at  $u \in U$ . Let  $b(u)$  be the nearest branching descendant such that there is no branching node between the root  $u$  and its descendant  $b(u)$ , and let  $children(b(u))$  be a set of children of  $b(u)$ . If the root  $u$  itself is a branching node, then  $b(u) = u$ . We denote all nodes between  $u$  and  $b(u)$  by  $path(u)$ .

We use the modified versions of the algorithm **dist** and the sub-algorithm **distsub**. We let  $I = \{u | u \in roots(T_1) \text{ and } u \text{ has a branching descendant}\}$  and  $J = \{v | v \in roots(T_2) \text{ and } v \text{ has a branching descendant}\}$ , and use  $path(u)$  instead of internal node  $u$ . We modify a few lines of the algorithms. We use:

$$dist(path(u), \emptyset) \quad \text{and} \quad dist(\emptyset, path(v))$$

instead of  $del(u)$  and  $add(v)$ , and we use

Calculate

$$dist(path(u) \cup tree(s), tree(v)) + dist(forest(children(b(u)) - \{s\}), \emptyset)$$

for each  $s \in children(b(u))$

Calculate

$$dist(tree(u), path(v) \cup tree(v)) + dist(\emptyset, forest(children(b(v)) - \{s\}))$$

for each  $s \in children(b(v))$

Calculate  $dist(tree(u) - path(u), tree(v) - path(v)) + dist(path(u), path(v))$   
return the minimum of the three cases

instead of  $dist(tree(u) - u, tree(v) - v) + chg(u, v)$ .

The distance between a path  $S$  and a tree  $T$  can be calculated in time  $O(|S| \cdot |T|)$  [4]. We use this method to calculate  $dist(path(u), tree(v))$  and  $dist(tree(u), path(v))$  in the algorithm.

**Theorem 3.** *The algorithm correctly calculates the distance between  $T_1$  and  $T_2$  in polynomial time.*

*Proof.* Correctness can be checked similarly as for the previous case. Notice that the nodes in  $path(u)$  of  $F_1$  can map to both  $path(v)$  and the nodes in one of the subtrees in  $\bigcup_{s \in children(b(v))} tree(s)$ . This can be handled in the modified version of the algorithm.

Let  $t_1$  and  $t_2$  be the number of branching nodes in  $T_1$  and  $T_2$ , respectively. Let  $\tau(t_1, t_2)$  be the time complexity of the algorithm **dist** with inputs  $T_1$  and  $T_2$ . Then,

$$\begin{aligned} \tau(t_1, t_2) &\leq t_1 \cdot \tau(t_1 - 1, t_2) + t_2 \cdot \tau(t_1, t_2 - 1) + t_1 \cdot t_2 \cdot \tau(t_1 - 1, t_2 - 1) \\ &\quad + t_1 \cdot t_2 \cdot O(n^2) + t_1 \cdot n \cdot \tau(t_1, t_2 - 1) + t_2 \cdot n \cdot \tau(t_1 - 1, t_2) + O(n^3) \\ &\leq (t_1! \cdot t_2! + t_1^{t_2} + t_2^{t_1} + O(n^2)) \cdot O(n^3) = O(n^{3 \cdot (t_1 + t_2)}). \end{aligned}$$

Since  $t_1$  and  $t_2$  are constants, the running time of the algorithm is polynomial. ■

### 2.3 The case of two generalized caterpillars

We here consider the case that each tree is a *generalized caterpillar*, which is a tree all whose branching nodes are on a unique path from the root to a leaf. Let  $T_1$  and  $T_2$  be generalized caterpillars. For this case, we use the same algorithm for two trees with a bounded number of branching nodes, while the analysis of the complexity is different. Correctness can be checked similarly as for the previous case.

**Theorem 4.** *The algorithm runs in time  $O(n^4)$  on two caterpillars.*

*Proof.* Let  $t_1$  and  $t_2$  be the (unbounded) number of branching nodes in  $T_1$  and  $T_2$ , respectively. Let  $\tau(t_1, t_2)$  be the time complexity of the algorithm **dist** with inputs  $T_1$  and  $T_2$ . Observing that there is always a unique non-path tree (i.e. generalized caterpillar) in each subforest of  $F_1$  and  $F_2$ , it follows that

$$\begin{aligned} \tau(t_1, t_2) &\leq \tau(t_1 - 1, t_2) + \tau(t_1, t_2 - 1) + \tau(t_1 - 1, t_2 - 1) \\ &\quad + |\text{roots}(F_1)| \cdot |\text{roots}(F_2)| \cdot O(n_1 \cdot n_2) + O(n^3), \end{aligned}$$

where  $n_1$  and  $n_2$  are the sizes of subtrees in  $F_1$  and  $F_2$  to be matched, respectively. Since  $|F_1| = |F_2| = n$ ,  $\tau(t_1, t_2) \leq O(n) \cdot (O(n^2) + O(n^3)) = O(n^4)$ . ■

## 3 Hardness

Zhang and Jiang [3] showed that the common subtree problem was NP-hard to approximate within some constant factor, even when one tree had a single branching node. However, the number of internal nodes and the height of the trees was not bounded. We present an approximation-preserving reduction that holds for a more restricted class of trees.

**Theorem 5.** *Let  $T_1$  be restricted to stars, and  $T_2$  restricted to trees of height 2. There is a fixed  $\epsilon > 0$ , such that it is NP-hard to distinguish between the following two cases: a)  $T_1$  is a subtree of  $T_2$ , and b) the maximum common subtree of  $T_1$  and  $T_2$  contains at most  $(1 - \epsilon)|T_1|$  vertices.*

**Reduction** We reduce from the problem 3-Set Packing-3:

*Given:* Finite set  $S$  and a collection  $C = \{C_1, C_2, \dots, C_m\}$  of subsets of  $S$  of size three, such that each element in  $S$  appears at most three times in a set in  $C$ .

*Find:* A collection of maximum cardinality of disjoint sets from  $C$ .

This problem is known to be hard to approximate within some constant factor greater than one [1] (see results on 3-DM). More precisely, there exists a  $\gamma > 0$ , such that it is NP-hard to decide whether there is an *exact cover* of  $(S, C)$ , or if no set packing contains more than  $n/3 - \gamma n$  sets.

Denote the elements of  $C_i$  by  $l_{i,1}, l_{i,2}, l_{i,3}$ . Denote the elements of  $S$  by  $l_1, l_2, \dots, l_n$ , with  $n$  being divisible by 3. Let  $x$  be a new element not in  $S$ .

We construct two labeled trees as follows:  $T_1$  is a star with  $n + (m - n/3)$  rays (vertices of degree one): one for each label  $l_i, i = 1, \dots, n$ , and  $m - n/3$  with the label  $x$ .  $T_2$  is of three levels: the root which is unlabeled;  $m$  internal nodes at level 2, all labeled with  $x$  and representing the sets  $C_i$ ; and,  $3m$  leaves, one for each element  $l_{i,j}$  of the sets  $C_i, i = 1, 2, \dots, m, j = 1, 2, 3$ . A subtree rooted by a node at level 2 is called a *clause*.

An exact cover of  $(S, C)$  yields a matching of all the leaves of  $T_1$ : using the leaves of  $n/3$  subtrees of  $T_2$ , along with the  $m - n/3$  remaining internal nodes labeled  $x$ .

On the other hand, suppose that we can match  $n + (m - n/3) - (\gamma/2)n$  non-root vertices from  $T_1$  and  $T_2$ . Without loss of generality, these include all the  $l_1, l_2, \dots, l_n$ , along with  $m - n/3 - (\gamma/2)n$  of the  $x$ -labeled vertices. That leaves  $n/3 + (\gamma/2)n$  internal nodes in  $T_2$  that are not matched and thus their leaves can be matched. Since we match all  $n$  vertices, all three leaves are matched in at least  $n/3 - \gamma n$  clauses. These induce a set packing in  $(S, C)$ , and we would be deciding an NP-hard problem.

Since  $m \leq n$ , the approximation ratio  $\frac{n+(m-n/3)}{n+(m-n/3)-(\gamma/2)n}$  is at most  $10/(10 - 3\gamma)$ . Hence, if we can approximate the common subtree problem within  $10/(10 - 3\gamma)$ , then  $P = NP$ . This completes the proof of Theorem 5.

Observe that our construction has a *gap location* at 1. Namely, it is hard to determine if all the nodes can be matched, or if only a certain constant fraction of them. That implies that the complementary problem, the editing distance problem with unit costs, is also hard to approximate for the same class of trees.

## 4 Approximation Algorithms

We present two approximation algorithms for the maximum common subtree problem in this section, that are based on partitioning the input trees into easy subproblems.

**Common subforests vs. common subtrees** We shall be searching for common *subforests* in the inputs. By trying all  $n^2$  choices for roots, this gives at least as good bounds for the common subtree problem as well.

**Approximation via partitioning** Suppose that we partition a forest into induced subgraphs; that is, each vertex is contained in exactly one partitioning class and the subgraph induced by the vertices within a class also forms a forest, under the ancestry relationship.

If we find solutions within each class (by matching the other input forest to the subgraph induced by each class), we can choose the largest one as the output.

This is because solutions to the common subforest problem are *hereditary* under vertex removal; subsets of solutions are also solutions. The optimal solution would be at most the union of optimal solutions of the classes. Hence, if we can find optimal solutions of each of the classes, the performance ratio attained would be at most the number of classes.

We generalize this argument to partitions of both input forests.

**Proposition 6.** *Suppose we partition input forests  $F$  and  $F'$  into induced subgraphs  $F_1, F_2, \dots, F_k$  and  $F'_1, F'_2, \dots, F'_l$ , and compute  $\rho$ -approximate solutions of the maximum common subforest of  $F_i$  and  $F'_j$ , for  $i = 1, \dots, k$ ,  $j = 1, \dots, l$ . Then the largest of these approximates the maximum weight common subforest of  $F$  and  $F'$  within a factor of  $\rho \cdot k \cdot l$ .*

*Proof.* Let  $OPT(X, Y)$  denote the size of the optimal solution on trees  $X$  and  $Y$ , and  $HEU(X, Y)$  the size of the approximate solution computed. Then,

$$HEU(F, F') \geq \frac{1}{\rho} \max_{i,j} OPT(F_i, F'_j) \geq \frac{1}{\rho k l} \sum_{i,j} OPT(F_i, F'_j) \geq \frac{1}{\rho k l} OPT(F, F'). \blacksquare$$

**Approximation in terms of the height** We first give a simple greedy heuristic for the case of a zero-height forest (set of independent vertices) vs. a general forest, with a performance ratio of 2.

```

Given an independent set  $I$  and a forest  $F$ :
Repeat until  $F$  is empty
  Pick any leaf  $v$  of  $F$  and delete it from  $F$ 
  if ( $v$  can be matched with a vertex  $w$  of  $I$ ) then
    Add  $v$  to solution
    Delete  $v$  and its ancestors from  $F$ 
    Delete  $w$  from  $I$ 
end

```

Each vertex  $v$  added to the solution can eliminate at most two vertices from the optimal solution out of  $I$ :  $w$ , and a node that the optimal solution matched to some ancestor of  $v$ . Hence, the ratio attained is 2.

This yields a performance ratio for general trees that is asymptotic to the height of the shorter tree.

**Theorem 7.** *The (unweighted) common subtree problem can be approximated within a factor of  $2 \min(ht(F_1), ht(F_2))$ .*

*Proof.* We can partition the shorter forest  $F$  into  $height(F)$  zero-height forests, and apply Proposition 6 to attain a ratio of  $2height(F)$ .  $\blacksquare$

**Approximation in terms of the number of vertices** We now obtain a  $\log^2 n$  performance ratio, by partitioning both input forests into at most  $\log n$  different *linear forests* each, solving each pair optimally, and referring to Proposition 6. In fact, we get a more general ratio in terms of the total number of branching nodes in the trees.

A linear forest is one where each component is a path. Solving the common subforest problem of two linear forests is equivalent to the common subtree problem of trees with a single branching node. That we have shown how to do in polynomial time in the previous section.

**Lemma 8.** *A tree with  $b$  branching nodes can be partitioned into  $\log b$  linear forests.*

*Proof.* We apply the following bottom-up greedy strategy until the tree is empty:

Include in the class the nodes on the path from each leaf up to, but not including, the nearest branching node.

After each iteration, the new leaves are exactly the minimally lowest branching nodes. At least two leaves are descendants of each branching nodes, hence the number of leaves is halved in each step. The number of steps, and the number of classes, is therefore at most  $\lfloor \log_2 b + 1 \rfloor$ . This bound is tight on complete binary trees. ■

Applying Proposition 6, Lemma 8 and the algorithm for single branching nodes, we obtain the following approximation.

**Corollary 9.** *Let  $T_1$  and  $T_2$  be trees with  $b_1$  and  $b_2$  branching nodes, respectively. The weighted common subtree of  $T_1$  and  $T_2$  can be approximated within a factor of  $\log b_1 \cdot \log b_2$ .*

We can generalize this to the case of finding common subtrees (subforests) in  $t$  trees,  $t$  constant.

**Theorem 10.** *The weighted common subtree problem of  $t$  trees can be approximated within a factor of  $t \log^t n$ .*

*Proof.* Partitioning each tree into  $\log n$  linear forests, we obtain  $\log^t n$  subproblems. Each is an instance of a weighted  $t$ -dimensional matching, which can be approximated within a factor of  $t$  by a greedy procedure in time  $n^t$ . The ratio then follows from Proposition 6. ■

## Acknowledgments

We thank Atsuko Yamaguchi for valuable discussions.

## References

1. CRESCENZI, P., AND KANN, V. A compendium of NP optimization problems. Dynamic on-line survey available at <ftp.nada.kth.se>, Oct. 1995.
2. YAMAGUCHI, A. Approximation algorithm for the maximum common hereditary subtree problem. Manuscript. (In Japanese), Nov. 1994.
3. ZHANG, K., AND JIANG, T. Some MAX SNP-hard results concerning unordered labeled trees. *Inf. Process. Lett.* 42 (1994), 133–139.
4. ZHANG, K., STATMAN, R., AND SHASHA, D. On the editing distance between unordered labeled trees. *Inf. Process. Lett.* 42 (1992), 133–139.