Title	Approximation and parameterized algorithms for common subtrees and edit distance between unordered trees
Author(s)	Akutsu, Tatsuya; Fukagawa, Daiji; Halldórsson, Magnús M.; Takasu, Atsuhiro; Tanaka, Keisuke
Citation	Theoretical Computer Science (2013), 470: 10-22
Issue Date	2013-01
URL	http://hdl.handle.net/2433/169691
Right	© 2012 Elsevier B.V.
Type	Journal Article
Textversion	author

Approximation and Parameterized Algorithms for Common Subtrees and Edit Distance between Unordered Trees *

Tatsuya Akutsu
Bioinformatics Center, Institute for Chemical Research, Kyoto University,
Uji, Kyoto 611-0011, Japan.

Daiji Fukagawa†
Faculty of Culture and Information Science, Doshisha University,
Kyoto 610-0394, Japan.

Magnús M. Halldórsson‡
ICE-TCS, School of Computer Science, Reykjavik University,
101 Reykjavik, Iceland.

Atsuhiro Takasu
National Institute of Informatics,
Tokyo 101-8430, Japan.

Keisuke Tanaka§
Department of Mathematical and Computing Sciences, Tokyo Institute of Technology,
Tokyo, Japan.

Abstract

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

We tackle these problems from two perspectives: giving exact algorithms, either for special cases or in terms of some parameters; and approximation algorithms and hardness of approximation. We present a parameterized algorithm in terms of the number of branching nodes that solves both problems and yields polynomial algorithms for several special classes of trees. This is complemented with a tighter APX-hardness proof that holds when the trees are of height one and two, respectively. Furthermore, we present the first approximation algorithms for both problems. In particular, for the common subtree problem for t trees,

*This paper is based on conference papers that appeared in [12] and [14]. TA and MMH are corresponding authors.

†Work done in part at National Institute of Informatics, Japan.

‡Work done in part at Japan Advanced Institute of Science and Technology and at the Science Institute, University of Iceland.

§Work done in part while visiting NTT Telecommunication Networks Laboratories and at Japan Advanced Institute of Science and Technology.

we present an algorithm achieving a $t \log_2(b_{OPT} + 1)$ ratio, where b_{OPT} is the number of branching nodes in the optimal solution. We also present constant factor approximation algorithms for both problems in the case of bounded height trees.
keywords: Tree edit distance; Approximation algorithms; Parameterized algorithms; Dynamic programming; Unordered trees

1 Introduction

A large area of computer science involves detecting and efficiently recognizing similarities among data sets or changes thereof. A natural measure of the difference between data sets is the minimum cost sequence of atomic changes (or editing operations) that transform one data set into another. Trees are perhaps the most pervasive structures of data. We consider in this paper the *tree edit 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 *largest common subtree* problem is to find a partial bijective matching of maximum cardinality between nodes of the trees that preserves labels and ancestry relationship. The *tree edit distance* problem is to determine the least cost sequence of insertions, deletions and substitutions that converts one tree into the other input tree. Deletion of a node v not only removes v but makes the parent of v become the new parent of the children of v. Insertion of a node v similarly makes v become the parent of some subset of the children of the new parent of v. A substitution operation changes only the label of a node. Each of these operations carries a cost function that depends on the label of the respective nodes.

These problems have numerous applications. In particular, such similarity recognition problems occur frequently in bioinformatics, since various biological objects and biological networks, including glycans [7], vascular networks [25] and cell lineage [16], can be represented as rooted, labeled, unordered trees. Comparison of and searching for XML data is also naturally modeled as tree edit computation of unordered trees [19].

The edit distance and common subtree problems of two ordered trees are known to be polynomially solvable [11, 23]. In particular, Demaine et al. developed an $O(n^3)$ time algorithm for the edit distance problem[1] and showed an $\Omega(n^3)$ lower bound for the family of decomposition strategy algorithms [11], where n denotes the size of a larger input tree. In order to cope with this barrier, Akutsu et al. developed an $O(n^2)$ time algorithm that approximates the unit cost edit distance within a factor of $O(n^{3/4})$ for bounded degree trees [3].

On the other hand, for two unordered trees, the edit distance and common subtree problems have been shown to be NP-hard [28]. We consider here three ways of dealing with the intractability of the problems: efficient algorithms for important special classes of trees, parameterized algorithms, and approximation algorithms with good performance bounds for general trees.

Special classes of trees and parameterized algorithms Given the NP-hardness of the common subtree problem, we study the complexity when the inputs are restricted to natural special classes of trees. We consider the following classes of trees:

Stars Trees where the root is the only internal node.

[1] This algorithm can also be applied to the common subtree problem.

Caterpillars Trees where all nodes are either on, or adjacent to nodes on, a particular path. These are the graphs of *pathwidth* 2. A superclass of *stars* and *paths*.

Moths Trees where all branching nodes (nodes with two or more children) lie on a single path from the root to a leaf.[2] A superclass of *caterpillars*.

Also, we consider the following parameters of trees:

- Number of *leaves*,
- Number of *branching nodes*,
- *Height* of the tree.

Several other parameters have been considered. The number of branching nodes *dominates* the number of *internal* nodes (in that the former is always at most the latter), and thus our algorithmic results for the former apply to the latter. The number of branching nodes also dominates the number of leaves. Both the number of leaves and internal nodes dominate the number of *nodes*. The situation for the parameter *maximum degree* is fully understood: the problem is in P if one tree has degree 2 (i.e., is a path) [27], but NP-hard if at least one has degree 3 or more [28].

We present an $O(2^{b_1+b_2}|T_1||T_2|\Delta)$-time algorithm based on dynamic programming that solves both the tree edit and common subtree problems, where T_1 and T_2 are two input trees, $|T_i|$ and b_i respectively denote the number of nodes and number of branching nodes in T_i, and Δ is the maximum number of children of a node in either tree. This means that both tree edit and common subtree problems are fixed-parameter tractable in terms of the branching number parameter. We also present a polynomial time algorithm for the classes of caterpillars and moths. For the other direction, we show that the common subtree problem and the tree edit problem are NP-hard, as well as hard to approximate, if one tree is of height one (viz. a star) and the other is of height two.

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 of the size of the optimal common subtree to the size of the common subtree found by the algorithm. In contrast, the performance ratio for the tree edit problem is the maximum of the ratio of the edit distance found by the algorithm to the optimal one.

For both tree edit and common subtree problems for unordered trees, we present the first approximation algorithms. For the common subtree problem for t trees, we present an approximation algorithm which achieves performance ratio of $t\lg(b_{OPT}+1)$, where b_{OPT} is the number of branching nodes in the optimal solution and $\lg(x) = \log_2(x)$. We also give an incomparable bound, that is t times the branching height of the optimal solution. For the tree edit distance problem, we present approximation algorithms with performance ratios of $O(n/\log n)$ and $2h+2$, respectively, where h is the height of the taller tree.

[2] Although moths are usually defined for undirected graphs, we only consider moths as rooted trees in this paper.

Although most approximability results are proved for the unweighted model (for common subtree) and the unit cost model (for tree edit), all the algorithms here achieve the same order of performance ratios if editing costs (resp. weights) are positive integers (resp. non-negative integers) bounded by a constant. The omitted proofs are straight-forward extensions of those for the unit cost model.

Previous results The following results are known about computing the largest common subtree or tree edit distance between unordered labeled trees. Both problems are MAX SNP-hard even for bounded degree trees [27], and thus cannot be approximated within some constant slightly larger than one. Recently, Hirata et al. showed that the tree edit problem remains MAX SNP-hard even for trees of height 2 [15]. However, one of the preliminary versions of this paper [14] appeared much earlier than [15] and our hardness result holds for a more restricted case (one input tree can be a star).

As for parameterized algorithms, Akutsu et al. developed an $O(2.62^k \cdot poly(|T_1|, |T_2|))$ time exact algorithm under the unit cost model when the edit distance is bounded by k [4]. Shasha et al. developed an $O(4^{\ell_1} 4^{\ell_2} \min(\ell_1, \ell_2) \cdot |T_1| \cdot |T_2|)$ time algorithm [21], where ℓ_1 and ℓ_2 are the number of leaves in trees T_1 and T_2, respectively. Although our parameterized algorithm has some similarity with theirs, it is considerably more efficient. Recently, Akutsu et al. developed an $O(1.26^{|T_1|+|T_2|})$ time algorithm for the general case and an $O((1+\epsilon)^{|T_1|+|T_2|})$ time algorithm for bounded degree trees over a fixed alphabet, where ϵ is any positive constant [6]. Although the former algorithm works in $O(2^{b_1+b_2} poly(|T_1|, |T_2|))$ time if measured by the number of branching nodes, it is more complicated than our parameterized algorithm.

It is known that restricted problems, *tree inclusion* and *tree alignment*, can be solved in polynomial time if the maximum degree is bounded by a constant, where the former is to decide whether T_2 is obtained from T_1 by deletion operations only [18], and the latter is to find the shortest sequence of insertions followed by substitutions followed by deletions that transforms T_1 into T_2 [17]. However, both problems are NP-hard for unbounded degree trees [17, 18]. For more about restricted cases of the common subtree problem and the tree edit problem, see [9].

In some studies, the largest common subtree is defined as a largest common connected subgraph of given trees. This problem can be solved in polynomial time for two trees, but is NP-hard for three trees and is hard to approximate in general [5]. Extensive studies have been done on common subtrees (e.g., agreement subtree) of phylogenetic trees [22]. However, our problems are very different because leaf labels have special and important roles in such studies.

Outline of paper In Section 3 we present polynomial time algorithms for special classes of trees, that apply both for the common subtree problem and for the tree edit problem. We then give a tightened hardness result in Section 4. Sections 5 and 6 respectively contain approximation algorithms for common subtrees and tree edit of general trees.

2 Definitions

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.

Editing operations and edit 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 children of the parent of v; b) inserting a node v – the complement of deleting – where v becomes the parent of a subset of the children of the new parent of v, and c) substituting the label of a node. It is to be noted that deletion or insertion of the root is not allowed because deletion of the root may decompose a tree into a forest.

These editing operations carry a cost function, that may depend on both the type of operation as well as the value of the label deleted/inserted/substituted. We denote them by $del(u), ins(v)$, and $sub(u, v)$, and assume they form a *distance metric*, satisfying non-negativity and transitivity. The edit 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 As shown by Zhang et al. [28], the tree edit problem is equivalent to finding the minimum-cost bijective mapping between subsets of the nodes of T_1 and T_2 that preserves the ancestor-descendant relationship:

> If u is mapped to v and u' to v', then u is an ancestor of u' in T_1 iff v is an ancestor of v' in T_2.

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

$$Cost(M) = \sum_{(u,v)\in M} sub(u, v) + \sum_{u\in I} del(u) + \sum_{v\in J} ins(v).$$

The largest common subtree problem is the natural complement of the tree edit problem, where the objective is to *maximize* the weight of the nodes that are not substituted, or equivalently, maximize the savings over deleting all nodes from T_1 and inserting all from T_2. We generalize the concept by allowing the common subtree to involve label substitutions; this functions as a subtractive cost. Let $w(u, v)$ denote the weight for a matched node pair (u, v). Then, the largest common subtree problem is to find a mapping M maximizing

$$Weight(M) \;=\; \sum_{(u,v)\in M} w(u, v).$$

If we define $w(u, v)$ by

$$w(u, v) = del(u) + ins(v) - sub(u, v),$$

to capture the cost of not matching u with v, we have the relationship [4]

$$Cost(M) = \sum_{u\in V(T_1)} del(u) + \sum_{v\in V(T_2)} ins(v) - Weight(M).$$

Since the first and second terms of the right hand side of this equation are determined only by input trees, it means that the minimum editing cost is determined by the maximum weight and vice versa. The tree edit and common subtree problems are thus equivalent in terms of optimization, but their approximative behavior is different given the different measures. In the common subtree problem, the roots are allowed to correspond to other nodes or not appear in M at all.

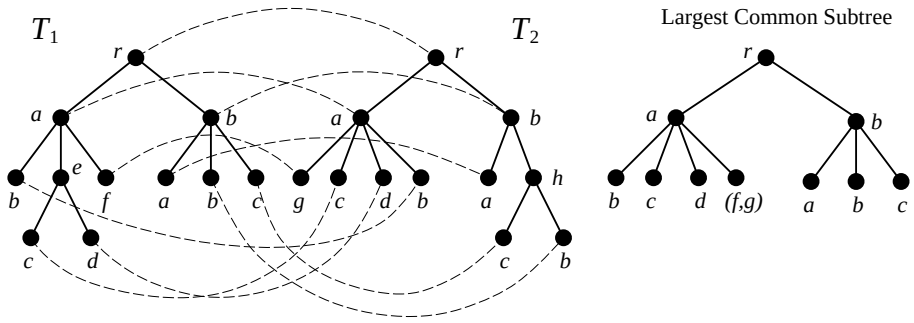

Figure 1: Example of tree edit and common subtree. Dashed curves denote a mapping.

The most common cost/weight models are the unit cost model for tree edit and the unweighted model for common subtree. In the former model, each of insertion, deletion, and substitution costs one unit. In the latter model, each of insertion and deletion costs one unit and substitution costs two units, which means that the weight is measured by the number of node pairs with identical labels appearing in M.[3] It is to be noted that $w(u, v) \geq 0$ holds for all u, v if it is defined via the distance metric although we need not assume this property in the parameterized algorithms for the common subtree problem.

Fig. 1 gives an example of tree edit and common subtree. In this example, T_2 is obtained from T_1 by a sequence of deletion of a node labeled e, substitution of a node labeled f, and insertion of a node labeled h. The corresponding cost is 3 under the unit cost model. Since this cost is the minimum over all possible editing sequences, the tree edit distance between T_1 and T_2 is 3. The corresponding common subtree is shown in the right-hand side. Its weight is 9 under the unweighted model, which is equal to the number of nodes, excluding (f, g) whose weight is 0. This tree is the largest common subtree under the unweighted model because its weight is the maximum among all possible common subtrees. It is also the largest common subtree under the weighting scheme corresponding to the unit cost model.

In this paper, we describe algorithms only for computing the weight of a largest or approximate common subtree or an optimal or approximate tree edit distance. However, the corresponding common tree or edit sequence can be obtained by using the standard traceback technique for dynamic programming without increasing the order of the time complexity.

The notion of the maximum common subtree can be extended to the one for t input trees. In this case, a mapping is extended to the set of t-tuples that preserves the ancestor-descendant relationship and the weight function is extended to

$$w(x_1, \ldots, x_t)$$

where x_i is a node in the ith input tree T_i.

Notation For a forest F, i.e. a collection of trees including the cases that F is a tree, let $V(F)$ and $|F|$ denote the set of nodes in F and the number of nodes in F, respectively. For a node

[3]We can generalize the problem to node weights, i.e. each node has an associated weight, and the objective (of the common subtree problem) is to maximize the sum of the matched nodes (nodes of different weights can be matched). This generalizes label weights, but our approximation algorithms at least still apply.

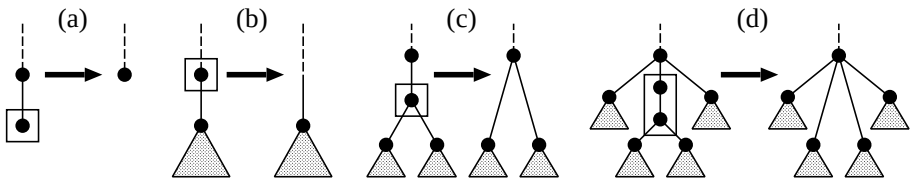

Figure 2: (a-c) Three types of ordinary deletions for $W_{con}(T_1, T_2)$. (d) Super-deletion.

v in F, $child(v)$, $anc(v)$, $des(v)$, $F(v)$, and $F - F(v)$ denote the set of children of v, the set of ancestors of v including v, the set of descendants of v excluding v, the subtree induced by v and its descendants, and the forest obtained from F by removing $F(v)$, respectively.[4] For a tree T, $root(T)$ and $ht(T)$ respectively denote the root of T and the height of T, where the height of a tree is the length of the longest simple path from the root to a leaf.

3 Exact Algorithms

3.1 Parameterized algorithm

In this subsection, we present a parameterized algorithm for the common subtree problem, which also solves the algorithm for the tree edit problem. In order to obtain the largest common subtree, we implicitly consider all possible ways of deleting nodes from both input trees via a combination of dynamic programming and exhaustive search for branching nodes.

First we present the dynamic programming part, which has some similarity with the algorithm for constrained edit distance [26]. For two trees T_1 and T_2, let $W_{con}(T_1, T_2)$ denote the weight of the largest common subtree between T_1 and T_2 which is obtained by deleting nodes from T_1 and T_2 under the condition that deletion operations can be repeatedly applied only to the following types of nodes:

(a) leaf,

(b) node having only one child, or

(c) node whose parent has only one child,

where $root(T_1)$ need not correspond to $root(T_2)$. These three types of deletions are illustrated in Fig. 2 (a-c) and are called *ordinary deletions*. A node that satisfies none of these three criteria can eventually be deleted once it satisfies one of them. A whole subtree can be deleted by repeated deletion of leaves. The same holds in a similar fashion for all nodes not in a given subtree.

In order to describe the deletions that are not possible, let the *branching path* P_v for a branching node v consist of v and its ancestors up to but not including its nearest branching node ancestor. The deletion of all nodes of a branching path will be referred to as a *super-deletion*. It is illustrated in Fig. 2 (d).

Lemma 3.1 *$W_{con}(T_1, T_2)$ can be computed in $O(|T_1||T_2|\Delta)$ time.*

[4] Although $anc(v)$ contains v, we do not call v an ancestor of v.

Proof. For a node u, let $\hat{u}$ denote the unique highest branching node among nodes in $des(u) \cup \{u\}$ if any. For convenience, let $W_{con}(u, v)$ denote $W_{con}(T_1(u), T_2(v))$, for a pair of nodes u in T_1, v in T_2. For each pair of branching nodes (u, v), we construct a weighted bipartite graph $G_{u,v}$ with vertex partitions $child(u)$ and $child(v)$, and let the weight of an edge between $u' \in child(u)$ and $v' \in child(v)$ be $W_{con}(u', v')$. Let $W_M(u, v)$ be the weight of the maximum weight matching in $G_{u,v}$.

Then, $W_{con}(u, v)$ can be computed using the following dynamic programming formula:

$$W_{con}(u,v) = \max \begin{cases} \max(0, w(u,v)), \\ \max(0, w(u,v)) + \max_{u' \in des(u), v' \in des(v)} W_{con}(u', v'), \\ \max(0, w(u,v)) + W_M(\hat{u}, \hat{v}), \\ \max_{v' \in des(v)} W_{con}(u, v'), \\ \max_{u' \in des(u)} W_{con}(u', v), \end{cases}$$

where taking the maximum over an empty set results in a zero value.[5] As for implementation details, this procedure can be executed in an arbitrary order from leaves to the roots; for example, we can use a double **for** loop where the inner loop processes $V(T_2)$ and the outer loop processes $V(T_1)$, both in postorder.

The meanings of the first, second and fourth lines are illustrated in Fig. 3, and that of the third line is illustrated in Fig. 4, where the fifth line is symmetric to the fourth line. It is to be noted that these lines are not exclusive; for example, the second line is achieved by repeated application of the fourth and fifth lines if $w(u, v) = 0$. It should also be noted that the weight of the resulting common subtree is given by $W_{con}(root(T_1), root(T_2))$ and a node pair (u, v) is contained in the resulting common subtree if $w(u, v)$ contributes to the total weight $W_{con}(root(T_1), root(T_2))$.[6]

Now we prove the correctness of the algorithm. Suppose that M is the mapping giving the largest common subtree T_c under deletion operations of type (a)-(c) in Fig. 2. Since insertion operations are defined as complement of deletion operations, we can assume that the largest common subtree is obtained from each of the input trees by deletion operations followed by substitution operations and each node in T_c can be represented by a pair of nodes in $V(T_1) \times V(T_2)$. We consider the following three cases.

(i) $(u, v) \in V(T_1) \times V(T_2)$ appears in T_c as a leaf.
Since all the descendants of u and v are deleted, this case is covered by the first line of the recurrence.

(ii) $(u, v) \in V(T_1) \times V(T_2)$ has only one child (u', v') in T_c.
This case is covered by the second line of the recurrence.

(iii) $(u, v) \in V(T_1) \times V(T_2)$ has multiple children in T_c.
This case is basically covered by the third line. However, some child u_i of u may correspond to a descendant v'_j of a child v_i of v. In this case, mapping between u_i and v'_j is covered by the fourth line. The symmetric case is covered by the fifth line. There also exists a

[5] This procedure may produce a common forest if $w(u, v) < 0$ for some u, v. However, we can cope with such a case by computing $W'_{con}(u, v)$ (in addition to $W_{con}(u, v)$) in which $\max(0, w(u, v))$ is replaced by $w(u, v)$ while keeping $W_{con}(-, -)$ in the right-hand side as is.

[6] There may exist multiple common subtrees giving the same weight. In such a case, (u, v) is contained in one of such common subtrees.

case where a pair of descendants (u'_i, v'_j) appears in T_c in place of (u_i, v_j). In this case, mapping between u'_i and v'_j is covered by the second line.

Although the same case may be covered by multiple combinations of the recurrence, it does not affect the correctness because the best solution is selected by means of 'max' operations. We can also see that super-deletions are not possible by the above recurrence because super-deletions introduce merge of children of different nodes. As mentioned in Section 2, the largest common subtree and the sequence of edit operations can be extracted from $W_{con}(u, v)$s using the standard traceback technique for dynamic programming. Therefore, the correctness of the algorithm is proved.

We can achieve the same result more efficiently by replacing $des(u)$ and $des(v)$ respectively by $child(u)$ and $child(v)$ in the above dynamic programming formulation, since such nodes can be covered by combination of the second, fourth and fifth lines.

In assessing the complexity, let us first set aside the costs of the weighted matching computation, W_M. The cost of computing $W_{con}(u, v)$, for a particular pair (u, v), is proportional to $|child(u)| \cdot |child(v)| + O(1)$. The total cost is then at most

$$\begin{aligned}
&\sum_{u \in V(T_1), v \in V(T_2)} \{|child(u)| \cdot |child(v)| + O(1)\} \\
&\leq \left(\sum_{u \in V(T_1)} |child(u)| \right) \cdot \left(\sum_{v \in V(T_2)} |child(v)| \right) + O(|T_1||T_2|) \\
&= O(|T_1||T_2|) \ .
\end{aligned}$$

Since maximum weight bipartite matching can be computed in $O(n_1 n_2 \max(n_1, n_2))$ time for a bipartite graph with partitions of sizes n_1 and n_2 [1], the total time required for computing all $W_M(\hat{u}, \hat{v})$'s is asymptotically

$$\begin{aligned}
&\sum_{u \in V(T_1), v \in V(T_2)} |child(u)| \cdot |child(v)| \cdot \max(|child(u)|, |child(v)|) \\
&\leq \Delta \sum_{u \in V(T_1), v \in V(T_2)} |child(u)| \cdot |child(v)| \\
&\leq \Delta |T_1||T_2| \ .
\end{aligned}$$

□

Next we combine the above dynamic programming algorithm with exhaustive search for branching nodes.

Theorem 3.2 *The common subtree and tree edit problems can be solved in $O(2^{b_1+b_2}|T_1||T_2|\Delta)$ time.*

Proof. Let $B(T)$ denote the set of branching nodes in a rooted unordered tree T. For each $B \subseteq B(T)$, $T \ominus B$ denotes the tree obtained from T by performing super-deletion operations for all nodes in B. Since each super-deletion does not propagate beyond the lowest branching ancestor, $T \ominus B$ is uniquely determined regardless of the order of super-deletion operations.

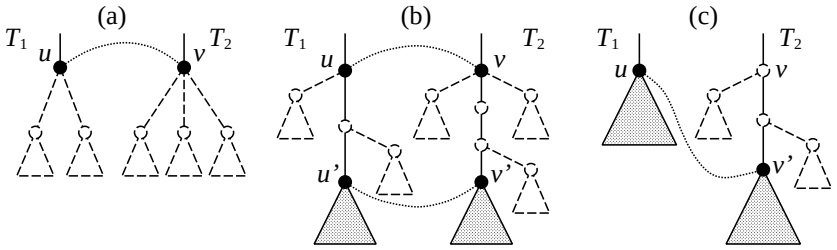

Figure 3: Explanation of the recurrence for computing $W_{con}(u, v)$. (a), (b), and (c) correspond to the first, second, and fourth lines of RHS of the recurrence, respectively. While circles and triangles with dashed lines mean deleted nodes and subtrees, respectively. Dotted curves mean that two nodes may correspond to each other.

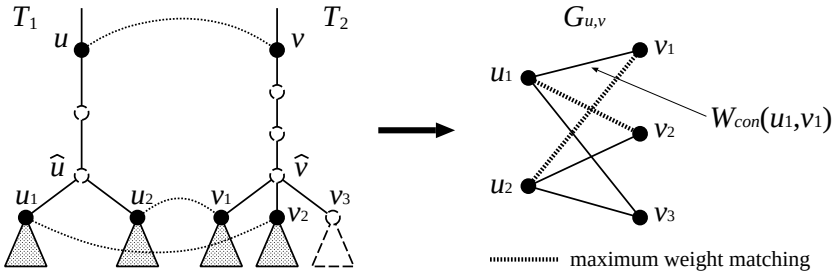

Figure 4: Explanation of the third RHS of the recurrence for computing $W_{con}(u, v)$. Mapping between the children of $\hat{u}$ and the children of $\hat{v}$ is computed by maximum bipartite matching. While circles and triangles with dashed lines mean deleted nodes and subtrees, respectively. Dotted curves mean that two nodes may correspond to each other.

We then compute the weight of the largest common subtree between T_1 and T_2 by

$$\max_{B_1 \subseteq B(T_1), B_2 \subseteq B(T_2)} W_{con}(T_1 \ominus B_1, T_2 \ominus B_2) \ .$$

The correctness of the algorithm is proved as follows. Recall that deletions of type (a)-(d) cover all types of deletions. Therefore, it is enough to prove that any subtree T_c of T_1 can be obtained by a sequence of super-deletions followed by ordinary deletions, where the same argument holds for T_2. To this end, we first mark all nodes in T_1 not appearing in T_c. For each branching node v, let $A(v)$ denote the set of non-branching nodes between v and the lowest branching ancestor of v. We determine B_1 by

$$B_1 = \{v |\ v \text{ is a marked branching node with no unmarked node in } A(v)\}.$$

Then, we can obtain T_c by super-deletion operations for nodes in B_1 followed by deletion operations for the remaining marked nodes (see Fig. 5). The time complexity directly follows from the above expression and Lemma 3.1. □

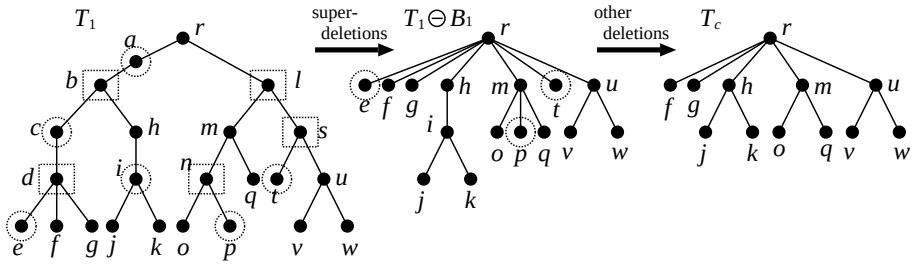

Figure 5: Determination of nodes for super-deletions. Nodes shown by dotted rectangles and dotted circles are marked nodes, where B_1 consists of nodes shown by dotted rectangles. Node i is not eliminated by super-deletion because it has an unmarked non-branching parent (i.e., node h).

3.2 The case of two moths

We here consider the case when both trees are moths, which we recall are trees whose branching nodes are on a single path, called the *backbone*, from the root to a leaf.

Theorem 3.3 *Common subtree and tree edit problems are polynomially solvable on two moths.*

Proof. Let u and v respectively be nodes in T_1 and T_2 outside the backbones. For each such pair (u, v), let $W_{path}(u, v)$ denote the weight of the largest common subtree between $T_1(u)$ and $T_2(v)$, where u and v need not correspond to each other. It is straight-forward to see that since $T_1(u)$ and $T_2(v)$ are paths, $W_{path}(u, v)$ can be computed using the standard sequence alignment algorithm in time $O(|T_1| \cdot |T_2|)$.

Let $(u_1, u_2, \ldots, u_h)$ and $(v_1, v_2, \ldots, v_k)$ be the backbones in T_1 and T_2 respectively, where u_1 and v_1 are the roots.

For each pair (u_i, v_j), we compute the weight of the largest common subtree between $T_1(u_i)$ and $T_2(v_j)$, denoted $W_{bb}(u_i, v_j)$, where u_i and v_j need not correspond to each other.

For $i < j$, let $CHD(u_i, u_j)$ be the set of children of $u_i, u_{i+1}, \ldots, u_{j-1}$ excluding nodes in the backbone (see Fig. 6); formally,

$$CHD(u_i, u_j) = \{w | \exists k, i \leq k < j, w \text{ is a child of } u_k \text{ but } w \neq u_{k+1}\} .$$

$CHD(v_i, v_j)$ is defined analogously. For $i' > i$ and $j' > j$, we construct a weighted bipartite graph with vertex-partition $CHD(u_i, u_{i'})$ and $CHD(v_j, v_{j'})$ by letting $W_{path}(u, v)$ be the weight of an edge between $u \in CHD(u_i, u_{i'})$ and $v \in CHD(v_j, v_{j'})$. Let $W_M(u_i, u_{i'}, v_j, v_{j'})$ be the weight of a maximum weight matching in this bipartite graph.

Then, $W_{bb}(u_i, v_j)$ can be computed by

$$W_{bb}(u_i, v_j) = \max(0, w(u_i, v_j)) + \max_{i'>i, j'>j} \{W_{bb}(u_{i'}, v_{j'}) + W_M(u_i, u_{i'}, v_j, v_{j'})\},$$

where taking the maximum over an empty set results in a zero value.

The correctness of the algorithm is seen by observing that all possible ways of deletions are taken into account by this dynamic programming algorithm. Since the maximum weight matching can be computed in polynomial time [1], the algorithm also works in polynomial time. □

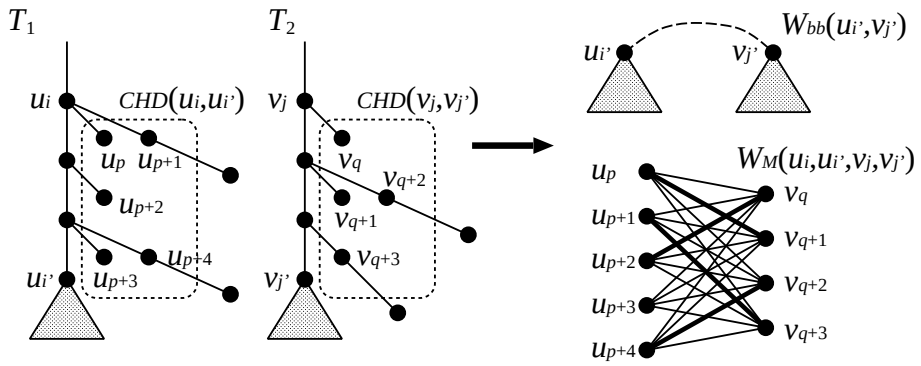

Figure 6: Illustration of the dynamic programming algorithm for moths.

4 Hardness

Zhang and Jiang [27] 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 4.1 *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 $(1-\epsilon)|T_1|$ nodes.*

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

> *Given:* Finite set S and a collection $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of subsets of S of size three, such that each element in S appears at most three times in a set in $\mathcal{C}$.
> *Find:* A collection of maximum cardinality of disjoint sets from $\mathcal{C}$.

This problem is known to be hard to approximate within some constant factor greater than one; more precisely, there exists a $\gamma > 0$, such that it is NP-hard to decide whether there is an *exact cover* of $(S, \mathcal{C})$, or if no set packing contains more than $n/3 - \gamma n$ sets [20].

Denote the elements of C_i by $l_{i,1}, l_{i,2}, l_{i,3}$. Denote the elements of S by $l_1, l_2, \ldots, l_n$, 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 $N_1 = n + (m - n/3)$ rays (nodes of degree one): one for each label l_i, $i = 1, \ldots n$, 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, $i = 1, 2, \ldots, m$, $j = 1, 2, 3$. A subtree rooted by a node at level 2 is called a *clause*.

An exact cover of $(S, \mathcal{C})$ yields a matching of all the leaves of T_1, with all the subtrees of T_2 and the $m - n/3$ remaining internal nodes labeled x.

On the other hand, suppose that we can match $Q = N_1 - (\gamma/2)n$ non-root nodes from T_1 and T_2. Without loss of generality, these include all the $l_1, l_2, \ldots, l_n$, along with $Q - n = m - n/3 - (\gamma/2)n$ of the x-labeled nodes. 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 nodes, all three leaves are matched in at least $n/3 - \gamma n$ clauses. These induce a set packing in $(S, \mathcal{C})$, and we would be deciding an NP-hard problem.

Since $m \leq n$, it holds that $N_1 \leq n + n - n/3 = 5n/3$. Thus, $Q \leq (1 - \frac{\gamma}{2} \cdot \frac{3}{5})N_1$. Hence, the claim of the theorem holds for $\epsilon = \frac{3\gamma}{10}$. □

Corollary 4.2 *The common subtree problem is APX-hard, even for trees of height 1 and 2, respectively.*

Observe that our construction has a *gap location* at 1, in the terminology of [20]. Namely, it is hard to determine if all the nodes can be matched, or if only a certain constant fraction of them. That has implication for related problems.

Corollary 4.3 *The tree edit distance problem is APX-hard, even when using unit cost for two trees of height 1 and 2, respectively.*

Proof. Let $N_1 = n + (m - n/3)$ and $N_2 = 4m$ be the number of non-root nodes of trees T_1 and T_2 respectively. Let OPT_{ED} denote the edit distance between T_1 and T_2 and OPT_{CS} be the cardinality of the maximum common subtree. Note that $OPT_{ED} = N_1 + N_2 - 2 \cdot OPT_{CS}$. Theorem 4.1 shows that it is NP-hard to distinguish between two cases: when $OPT_{CS} = N_1 + 1$ and when $OPT_{CS} \leq N_1 - \frac{\gamma}{2}n$. This implies that it is NP-hard to distinguish when $OPT_{ED} = N_2 - N_1$ and when $OPT_{ED} \geq N_2 - N_1 + \gamma n$. Since $N_2 - N_1 = 3m - 2n/3$ and $n \geq m$, it holds that $n = \frac{9}{7}n - \frac{2}{7}n \geq \frac{9m-2n}{7} = \frac{3}{7}(N_2 - N_1)$. Hence, it is hard to distinguish between when $OPT_{ED} = N_2 - N_1$ and when $OPT_{ED} \geq (1 + \frac{3}{7}\gamma)(N_2 - N_1)$. Thus, obtaining an approximation factor of $1 + \frac{3}{7}\gamma$ is NP-hard. □

5 Approximation Algorithms for Common Subtrees

We give an approximation algorithm for the problem of finding a maximum common subtree of t trees, where t is a constant. This is obtained by an approximation-preserving polynomial-time reduction to a special case of the maximum independent set problem in graphs.

We first address a special class of trees, so-called spiders, and give an approximation algorithm for finding a maximum common subtree that is restricted to be a spider. We then show that any tree is bound to contain a large spider, thus our algorithm actually attains a good performance for the general common subtree problem.

Common subforests vs. common subtrees We shall formulate our problem as one regarding forests, where both the inputs and the common substructure can be disconnected. These are actually polynomially equivalent problems. One can always restructure an input forest by adding a dummy node as root. Similarly, when searching for a common subtree, we can try all possible matchings of roots (at most n^t choices, on t input trees), and then restrict the instances, and the pattern, to the forests induced by proper descendants of the matched roots.

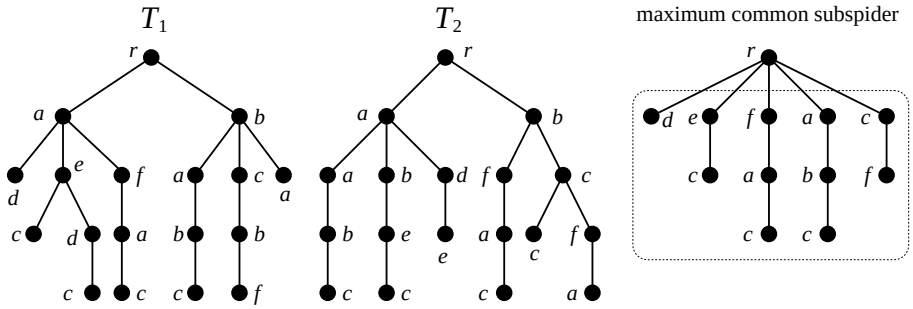

Figure 7: Example of a maximum common subspider for T_1 and T_2. The corresponding maximum common linear subforest is shown by dotted curves.

5.1 Common linear subforests and subspiders

A *linear forest* is one where each component is a path. A *spider* is tree where all non-root nodes are of degree at most 2. Alternatively, a spider consists of a root on top of a linear forest.

We shall focus on the problem of approximating the maximum common linear subforest of t input forests, referred to as the t-CLS problem (see Fig. 7 for 2-CLS). By the above discussion, this is equivalent to finding a maximum common subspider. We show that t-CLS can be reduced to a special case of the maximum independent set problem in so-called t-union graphs. We then show that any tree – including the optimal common subtree – must contain a large spider.

A graph $G = (V, E)$ is a *t-union graph* if there is a collection $G_i = (V, E_i)$, $i = 1, 2, \ldots, t$, where each G_i is an interval graph, such that $E = E_1 \cup E_2 \cup \cdots \cup E_t$. Each node in V then corresponds to an ordered t-tuple of intervals, referred to as a t-interval.

The t-WMIS problem is as follows: Given a union graph G and the corresponding collection of t-intervals, with positive weights on the t-intervals, find the maximum weight independent set of vertices in G, i.e., a subcollection of the t-intervals all of whose constituting intervals are disjoint. We are particularly interested in the special case, t-LamWMIS, where the graph G_i are containment interval graphs, i.e., when the set of intervals forms a laminar family. This problem is also known as *Tree Constrained t-partite Matching* [10].

Proposition 5.1 *There is an approximation preserving reduction from t-CLS to t-LamWMIS.*

Proof. We shall prove this for the case $t = 2$ (see Fig. 8); the extension to larger t is straightforward.

Let F_1, F_2 be the input forests. We first form an arbitrary ordering of the input forests, i.e., form a total order among the children of each internal node. This induces a numbering of the leaves respecting the total order; we shall assume the leaves of different forests are labeled differently.

Label each node v of an input tree with the interval $I_v = [l_v, r_v]$, where l_v (r_v) is the smallest (largest) leaf label in the subtree rooted by v, respectively. For each leaf x_1 in F_1, each ancestor a_1 of x_1 (possibly $a_1 = x_1$), each leaf x_2 in F_2 and each ancestor a_2 of x_2, we construct a 2-interval $J_{x_1,a_1,x_2,a_2} = \{I_{a_1}, I_{a_2}\}$. Let P_1 be the path from x_1 to a_1, and P_2 the path from x_2 to a_2. The weight $w(J_{x_1,a_1,x_2,a_2}) = LCS(P_1, P_2)$, the maximum weight common subsequence of

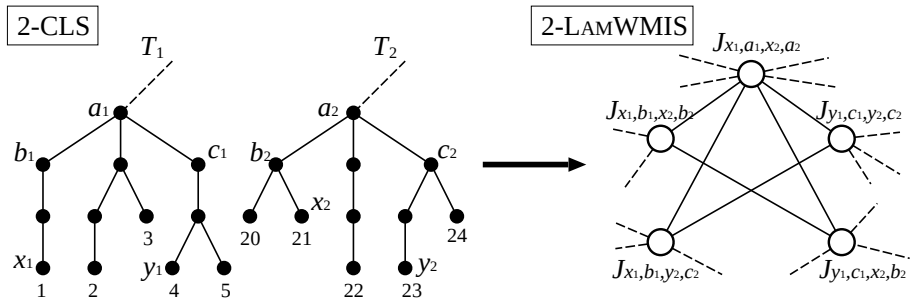

Figure 8: Reduction from 2-CLS to 2-LAMWMIS. In this example, a part of G is shown, and each 2-interval and the corresponding vertex in G are identified, where $J_{x_1,a_1,x_2,a_2} = \{[1,5],[20,24]\}$, $J_{x_1,b_1,x_2,b_2} = \{[1,1],[20,21]\}$, $J_{x_1,b_1,y_2,c_2} = \{[1,1],[23,24]\}$, $J_{y_1,c_1,y_2,c_2} = \{[4,5],[23,24]\}$, $J_{y_1,c_1,x_2,b_2} = \{[4,5],[20,21]\}$.

P_1 and P_2. Let $\mathcal{I}_{F_1,F_2}$ be the collection of all such 2-intervals. Observe that the collection of intervals that form the basis of $\mathcal{I}_{F_1,F_2}$ is a laminar family.

We now argue that any solution to t-LAMWMIS on $\mathcal{I}_{F_1,F_2}$ yields a solution to t-CLS on F_1 and F_2 with same objective value. This implies the proposition. Let $\mathcal{I}'$ be a solution to t-LAMWMIS on $\mathcal{I}_{F_1,F_2}$; i.e., it consists of a subcollection of 2-intervals whose underlying intervals are disjoint. Any 2-interval J is a pairing of a path in F_1 with a path in F_2. Furthermore, for any two 2-intervals J, J' in $\mathcal{I}_{F_1,F_2}$, the corresponding paths in F_1 and F_2 must be disjoint, since the constituting intervals are disjoint. Hence, $\mathcal{I}'$ induces a valid solution to t-CLS, and it is easily seen that the value of each 2-interval corresponds to the contribution that the corresponding matched paths make to the objective function of t-CLS. □

The t-WMIS problem is approximable within a $2t$-factor [8], while the special case of t-LAMWMIS was recently shown to be approximable within a t-factor [10].

Theorem 5.2 t-CLS *(and equivalently the maximum common subspider problem on t input trees) is approximable within a factor t.*

5.2 Relating subspiders to tree size

To relate the linear forest approximation to the original common subtree problem, we show that any forest, in particular the optimal common subforest, must contain a relatively large linear forest.

We define a parameterized class of trees called *s-moths*. They are defined recursively as follows.

Definition 5.3 *The 0-moths are the paths. The s-moths are the trees in which the minimal subgraph containing all branching nodes forms an s'-moth, for some $s' \leq s-1$.*

In particular, moths are 1-moths.

Lemma 5.4 *Let T be an s-moth with b branching nodes. Then T can be partition into trees T_1 and T_2, where T_1 is a linear forest and T_2 is an $(s-1)$-moth with at most $\lfloor (b-1)/2 \rfloor$ branching nodes.*

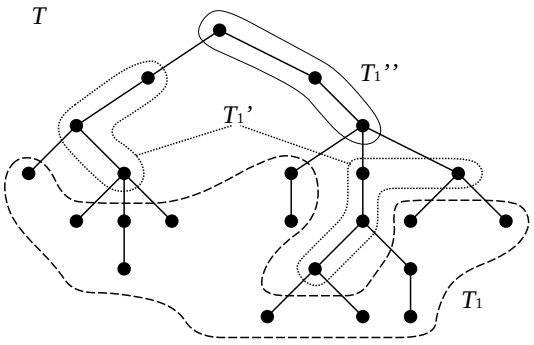

Figure 9: Decomposition of a tree into linear forests. In this example, T is decomposed into T_1, T_1', and T_1'', where T_1'', $T_1' \cup T_1''$, and T are 0-, 1-, and 2-moths.

Proof. Following the definition of an s-moth, let T_2 be the minimal subgraph containing all branching nodes in T and T_1 be the linear forest induced by the remaining vertices (see Fig. 9). It is clear that T_2 is an $(s-1)$-moth.

For branching nodes v and v' in T, we say that v is a *branching child* of v' if v is a proper descendant of v' and there is no other branching node on the path from v to v'. Let $d(v)$ denote the number of branching children of a branching node v in T. Let $B_2(T)$ denote the set of branching nodes in T with at least two branching children. A key observation is that the set of branching nodes in T_2 is exactly $B_2(T)$.

Each branching node in T, except for the root, is a branching child of a single branching node. The total number of branching children of nodes in $B_2(T)$ is then at most $b-1$. Thus,

$$2|B_2(T)| \leq \sum_{v \in B_2(T)} d(v) \leq b-1 \ .$$

It follows that the number $|B_2(T)|$ of branching nodes in T_2 is at most $(b-1)/2$. □

Lemma 5.4 has several straightforward implications.

Corollary 5.5 *Let T be a rooted tree with moth number $m(T)$, and let $\hat{T}$ be T without its root. Then $\hat{T}$ can be partitioned into $m(T)$ linear forests.*

Corollary 5.6 *Let T be a vertex-weighted tree of weight $w(T)$ and with moth number $m(T)$. T contains a sub-spider of weight at least $w(T)/m(T)$.*

Define the *branching height* of a tree to be the maximum number of branching nodes on a root-leaf path. The following observation follows also from Lemma 5.4.

Observation 5.7 *The moth number $m(T)$ of a tree T with b branching nodes and branching height h is at most* $\min(\lfloor \lg b + 1 \rfloor, h)$.

Let b_{OPT} (h_{OPT}) denote to the branching number (branching height) of an optimal common subtree. By combining Thm. 5.2, Cor. 5.6, and Obs. 5.7, we obtain the following performance guarantee for the general common subtree problem.

Theorem 5.8 *There is a $t \cdot \lg(b_{OPT}+1)$-approximation algorithm for the maximum weight common subtree problem of t trees. A $t \cdot h_{OPT}$-ratio also holds.*

It can be shown that $b_{OPT} \leq (|T_{OPT}|-1)/2$. Also, clearly, $|T_{OPT}| \leq \min_i |T_i|$, the cardinality of the smallest input tree.

The only previously known approximation ratio was $O(\log^t n)$ [14], where n is the size of the largest input tree. Also, a factor of twice the height of the smaller *input* tree (for the case of two trees) was also given in [14], improved to 1.5 times the height in [2].

6 Approximation Algorithms for Edit Distance

As mentioned in the Introduction, Akutsu et al. developed an $O(2.62^k \cdot poly(n))$-time algorithm that computes the exact tree edit distance OPT_{ED} when it is no more than k [4], which can also decide whether or not $OPT_{ED} \leq k$ holds. By using it, we obtain the following theorem.

Theorem 6.1 *The unordered tree edit distance can be approximated within an $O(n/\log n)$-factor under the unit cost model, where n is the size of the larger input tree.*

Proof. Let $K = c \log n$, where c is an arbitrary constant. By applying the algorithm in [4], we can compute the exact distance if $OPT_{ED} \leq K$. Otherwise, we compute the approximate edit distance by deleting all non-root nodes in T_1, changing the label of the root (if necessary), and inserting all non-root nodes in T_2.

Let APX_{ED} be the cost obtained by the above procedure. In the former case, $APX_{ED} = OPT_{ED}$ holds. In the latter case, $APX_{ED} \leq 2n$ holds, where $OPT_{ED} > K$. By combining these two, $APX_{ED} \leq \frac{2n}{c \log n} \cdot OPT_{ED}$. □

The above theorem can be extended for the cases in which the cost of each editing operation is a positive integer no larger than some constant d because the algorithm in [4] can work for such a case and OPT_{ED} is bounded by dn for some constant d.

We next consider bounded height trees. For a pair of trees t and T, let $\phi_t(T)$ denote the number of $T(v)$'s isomorphic to t; in other words, $\phi_t(T) = |\{v \in V(T) | T(v) \approx t\}|$, where $T_1 \approx T_2$ denotes that T_1 is isomorphic to T_2. Let $\mathcal{S}_n$ be the set of all possible trees of size at most n. Then, we consider the feature vector $\phi(T)$ which is a vector of positive integers defined by $\phi(T) = (\phi_t(T))_{t \in \mathcal{S}_n}$ (see Fig. 10). Although the number of dimensions of $\phi(T)$ is exponential in n, we only consider the non-zero elements. Since the number of non-zero elements is at most n for each T, $\phi(T)$ can be represented in polynomial size.

For a feature vector $\phi(T)$ on $\mathcal{S}_n$, let $\|\phi(T)\|_1$ denote the L_1 norm of $\phi(T)$. Hereafter, we will show that $OPT_{ED} \leq \|\phi(T_1) - \phi(T_2)\|_1 \leq (2h+2) \cdot OPT_{ED}$ holds, where h is the maximum height of T_1 and T_2. It is to be noted that the distance $\|\phi(T_1) - \phi(T_2)\|_1$ is equivalent to the bottom-up distance in [24]. As shown in that paper, the corresponding bottom-up mapping is a special case of the edit distance mapping and the time complexity of computing the bottom-up distance is linear in the number of nodes. However, no bound on the approximation ratio of bottom-up distance against the original tree edit distance is given in [24], which is the main result of this section.

Lemma 6.2 $\|\phi(T_1) - \phi(T_2)\|_1 \leq (2h+2) \cdot OPT_{ED}$.

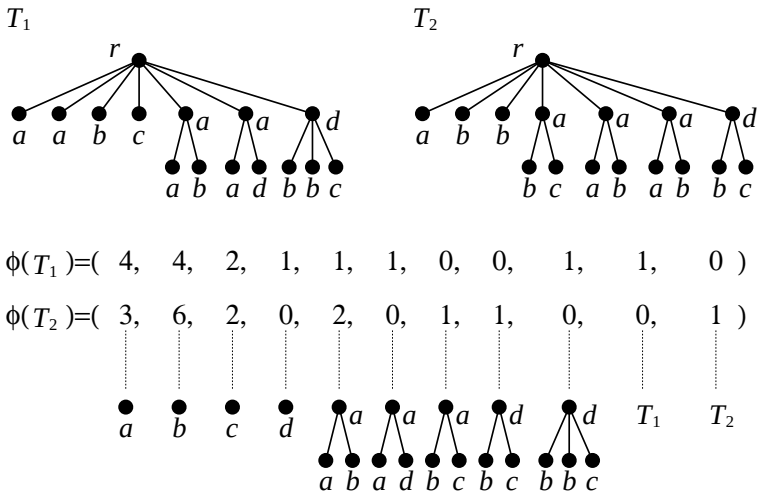

Figure 10: Feature vectors for two trees. Only coordinates whose values are positive for at least one of T_1 and T_2 are shown.

Proof. Let $o_1, \ldots, o_m$ be an optimal sequence of edit operations which converts T_1 into T_2, and let $T^0(= T_1), T^1, \ldots, T^m(= T_1)$ be the corresponding sequence of intermediate trees, where $m = OPT_{ED}$ holds. That is, for $j = 1, 2, \ldots, m$, the tree T^j is obtained by applying an edit operation o_j to T^{j-1}. We assume without loss of generality that $ht(T^j) \leq h$ holds for all $j = 1, \ldots, m$ since it is known that any optimal sequence of edit operations can be arranged so that all deletions precede all insertions [23].

We consider how the feature vector changes according to each operation.

Substitution: Let v be the substituted node in T^{j-1} and T^j. Then, ϕ_t decreases by at most one for each $t = T^{j-1}(x)$ with $x \in anc(v)$, and $\phi_{t'}$ increases by at most one for each $t' = T^j(x)$ with $x \in anc(v)$. Therefore, $\|\phi(T^{j-1}) - \phi(T^j\|_1 \leq 2|anc(v)| \leq 2h + 2$ holds.

Deletion: Let v be the deleted node in T^{j-1} and let w be its parent. Then, ϕ_t decreases by at most one for each $t = T^{j-1}(x)$ with $x \in anc(v)$, and $\phi_{t'}$ increases by at most one for each $t' = T^j(x)$ with $x \in anc(w)$. Therefore, $\|\phi(T^{j-1}) - \phi(T^j)\|_1 \leq |anc(v)| + |anc(w)| \leq 2|anc(v)| \leq 2h + 2$ holds.

Insertion: Since insertion is the complement of deletion, $\|\phi(T^{j-1}) - \phi(T^j)\|_1 \leq 2h + 2$ holds.

Since each o_j is one of the above, we have $\|\phi(T_1) - \phi(T_2)\|_1 \leq \sum_{i=1}^{m} \|\phi(T^{j-1}) - \phi(T^j)\|_1 \leq (2h + 2) \cdot OPT_{ED}$. □

Lemma 6.3 $OPT_{ED} \leq \|\phi(T_1) - \phi(T_2)\|_1$.

Proof. We show that there exists a sequence of editing operations of length at most $\|\phi(T_1) - \phi(T_2)\|_1$ which transforms T_1 into T_2.

Since the roots are not deleted or inserted and $\phi(T)$ changes by 2 according to a substitution of the root whose editing cost is 1 (< 2), we assume without loss of generality that the labels of the roots of T_1 and T_2 are the same.

We first note that if $\|\phi(T_1) - \phi(T_2)\|_1 = 0$, $T_1 \approx T_2$ clearly holds and thus we need no editing operation.

Otherwise, we repeatedly delete nodes from T_1 and T_2 as follows until the resulting trees become isomorphic.

If there exists any pair (u, v) such that $T_1(u) \approx T_2(v)$, $u \in \mathit{child}(\mathit{root}(T_1))$, and $v \in \mathit{child}(\mathit{root}(T_2))$ hold, we remove $T_1(u)$ and $T_2(v)$ respectively from T_1 and T_2, where these nodes are not counted as 'deleted' but rather as non-candidates for deletions. Since $\phi(T_1(u)) = \phi(T_2(v))$ holds, we have

$$\phi(T_1 - T_1(u)) - \phi(T_2 - T_2(v)) = (\phi(T_1) - \phi(T_1(u))) - (\phi(T_2) - \phi(T_2(v))) = \phi(T_1) - \phi(T_2).$$

We greedily repeat this procedure until there does not exist such a pair (u, v). Let T_1' and T_2' be the resulting trees.

Let v be a node such that $|T_i'(v)|$ $(i = 1, 2)$ is the largest among children of $\mathit{root}(T_1')$ and $\mathit{root}(T_2')$. We assume without loss of generality that v is a child of $\mathit{root}(T_1')$. Then we delete v from T_1' and let T_1'' be the resulting tree. Since v is a child with the largest $|T_i'(v)|$, $\phi_t(T_1'') = \phi_t(T_1') - 1$ and $\phi_t(T_2') = 0$ hold for $t = T_1'(v)$. Therefore, $\|\phi(T_1'') - \phi(T_2')\|_1 = \|\phi(T_1') - \phi(T_2')\|_1 - 1$ holds.

Suppose that $u_1, u_2, \ldots, u_{k_1}$ and $v_1, v_2, \ldots, v_{k_2}$ are deleted in these orders respectively from T_1 and T_2. It is to be noted that $k_1 + k_2 = \|\phi(T_1) - \phi(T_2)\|_1$ holds. We construct an editing sequence such that $u_1, u_2, \ldots, u_{k_1}$ are deleted from T_1 in this order and then $v_{k_2}, v_{k_2-1}, \cdots, v_1$ are inserted to the resulting tree in this order. It is straight-forward to see that this sequence is a valid editing sequence of length $\|\phi(T_1) - \phi(T_2)\|_1$ and transforms T_1 into T_2. □

Combining Lemma 6.2 and Lemma 6.3, we can see that $OPT_{ED} \leq \|\phi(T_1) - \phi(T_2)\|_1 \leq (2h + 2) \cdot OPT_{ED}$ holds.

Theorem 6.4 *The unordered tree edit distance can be approximated within a factor of* $2h + 2$ *under the unit cost model, where* h *is the maximum height of two input trees.*

Acknowledgments

We thank Atsuko Yamaguchi for valuable discussions.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993.

[2] T. Akutsu, D. Fukagawa, and A. Takasu, Improved approximation of the largest common subtree of two unordered trees of bounded height, *Information Processing Letters* 109 (2008) 165–170.

[3] T. Akutsu, D. Fukagawa, and A. Takasu, Approximating tree edit distance through string edit distance, *Algorithmica* 57 (2010) 325–348.

[4] T. Akutsu, D. Fukagawa, A. Takasu, and T. Tamura, Exact algorithms for computing tree edit distance between unordered trees, *Theoretical Computer Science* 421 (2011) 352–364.

[5] T. Akutsu and M. M. Halldórsson, On the approximation of largest common subtrees and largest common point sets, *Theoretical Computer Science* 233 (2000) 33–50.

[6] T. Akutsu, T. Tamura, D. Fukagawa, and A. Takasu, Efficient exponential time algorithms for edit distance between unordered trees, in: *Proceedings of the 23rd Annual Symposium on Combinatorial Pattern Matching*, in LNCS, vol. 7354, Springer, 2012, pp. 360–372.

[7] K. F. Aoki, A. Yamaguchi, N. Ueda, T. Akutsu, H. Mamitsuka, S. Goto, and M. Kanehisa, KCaM (KEGG Carbohydrate Matcher): A software tool for analyzing the structures of carbohydrate sugar chains, *Nucleic Acids Research* 32 (2004) W267–W272.

[8] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai and I. Shapira, Scheduling Split Intervals, *SIAM J. Comput.*,36 (2006) 1–15.

[9] P. Bille, A survey on tree edit distance and related problem, *Theoretical Computer Science* 337 (2005) 217–239.

[10] S. Canzar, K. M. Elbassioni, G. W. Klau, and J. Mestre, On tree-constrained matchings and generalizations, in: *Proceedings of the 38th International Colloquium on Automata, Languages and Programming*, in LNCS, vol. 6755, Springer, 2011, pp. 98–109.

[11] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, An optimal decomposition algorithm for tree edit distance, *ACM Transactions on Algorithms* 6 (2009) 1.

[12] D. Fukagawa, T. Akutsu, and A. Takasu, Constant factor approximation of edit distance of bounded height unordered trees, in: *Proceedings of the 16th International Symposium on String Processing Information Retrieval*, in LNCS, vol. 5721, Springer, 2009, pp. 7–17.

[13] M. M. Halldórsson, Approximations of weighted independent set and hereditary subset problems, *Journal of Graph Algorithms and Applications* 4 (2000) 1.

[14] M. M. Halldórsson and K. Tanaka, Approximation and special cases of common subtrees and editing distance, in: *Proceedings of the 7th International Symposium on Algorithms and Computation*, in: LNCS, vol. 1178, Springer, 1996, pp. 75–84.

[15] K. Hirata, Y. Yamamoto, and T. Kuboyama, Improved MAX SNP-hard results for finding an edit distance between unordered trees, in: *Proceedings of the 22nd Annual Symposium on Combinatorial Pattern Matching*, in: LNCS, vol. 6661, Springer, 2011, pp. 402-415.

[16] Y. Horesh, R. Mehr, and R. Unger, Designing an A* algorithm for calculating edit distance between rooted-unordered trees, *Journal of Computational Biology* 6 (2006) 1165–1176.

[17] T. Jiang, L. Wang, and K. Zhang, Alignment of trees - an alternative to tree edit, *Theoretical Computer Science* 143 (1995) 137–148.

[18] P. Kilpeläinen and H. Mannila, Ordered and unordered tree inclusion, *SIAM Journal on Computing* 24 (1995) 340–356.

[19] D. Milano, M. Scannapieco, and T. Catarci, Structure-aware XML object identification, *Data Engineering Bulletin* 29 (2006) 67–74.

[20] E. Petrank, The Hardness of Approximation: Gap Location. *Computational Complexity* 4 (1994) 133–157.

[21] D. Shasha, J. T.-L. Wang, K. Zhang, K., and F. Y. Shih, Exact and approximate algorithms for unordered tree matching, *IEEE Transactions on Systems, Man, and Cybernetics* 24 (1994) 668–678.

[22] W.-K. Sung, *Algorithms in Bioinformatics. A Practical Approach*, CRC Press, 2010.

[23] K.-C. Tai, The tree-to-tree correction problem, *Journal of ACM* 26 (1979) 422–433.

[24] G. Valiente, An efficient bottom-up distance between trees, in: *Proceeding of the 8th International Symposium on String Processing Information Retrieval*, IEEE, 2001, pp. 212–219.

[25] K.-C. Yu, E. L. Ritman, and W. E. Higgins, System for the analysis and visualization of large 3D anatomical trees, *Computers in Biology and Medicine* 27 (2007) 1802–1830.

[26] K. Zhang, A constrained edit distance between unordered labeled trees, *Algorithmica* 15 (1996) 205–222.

[27] K. Zhang and T. Jiang, Some MAX SNP-hard results concerning unordered labeled trees, *Information Processing Letters* 49 (1994) 249–254.

[28] K. Zhang, R. Statman, and D. Shasha, On the editing distance between unordered labeled trees, *Information Processing Letters* 42 (1992) 133–139.