Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 337 (2005) 217–239

Theoretical Computer Science

www.elsevier.com/locate/tcs

A survey on tree edit distance and related problems☆

Philip Bille*

The IT University of Copenhagen, Rued Langgardsvej 7, DK-2300 Copenhagen S, Denmark

Received 23 March 2004; received in revised form 23 December 2004; accepted 30 December 2004
Communicated by A. Apostolico

Abstract

We survey the problem of comparing labeled trees based on simple local operations of deleting, inserting, and relabeling nodes. These operations lead to the tree edit distance, alignment distance, and inclusion problem. For each problem we review the results available and present, in detail, one or more of the central algorithms for solving the problem.

Keywords: Tree matching; Tree edit distance; Tree alignment; Tree inclusion

1. Introduction

Trees are among the most common and well-studied combinatorial structures in computer science. In particular, the problem of comparing trees occurs in several diverse areas such as computational biology, structured text databases, image analysis, automatic theorem proving, and compiler optimization [43,55,22,24,16,35,56]. For example, in computational biology, computing the similarity between trees under various distance measures is used in the comparison of RNA secondary structures [55,18].

☆ This work is part of the DSSCV project supported by the IST Programme of the European Union (IST-2001-35443).

* Tel.: +45 72 18 50 00; fax: +45 72 18 50 01.
E-mail address: beetle@itu.dk.

0304-3975/$ - see front matter
doi:10.1016/j.tcs.2004.12.030

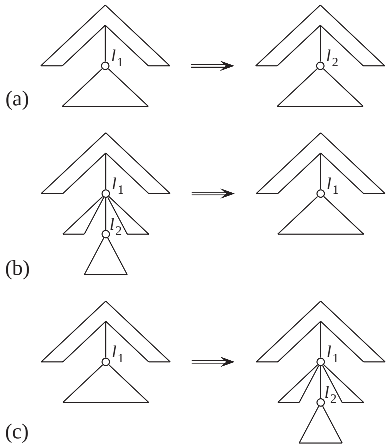

Fig. 1. (a) A relabeling of the node label l_1 to l_2. (b) Deleting the node labeled l_2. (c) Inserting a node labeled l_2 as the child of the node labeled l_1.

Let T be a rooted tree. We call T a *labeled tree* if each node is a assigned a symbol from a fixed finite alphabet Σ. We call T an *ordered tree* if a left-to-right order among siblings in T is given. In this paper we consider matching problems based on simple primitive operations applied to labeled trees. If T is an ordered tree these operations are defined as follows:

Relabel: Change the label of a node v in T.

Delete: Delete a non-root node v in T with parent v', making the children of v become the children of v'. The children are inserted in the place of v as a subsequence in the left-to-right order of the children of v'.

Insert: The complement of delete. Insert a node v as a child of v' in T making v the parent of a consecutive subsequence of the children of v'.

Fig. 1 illustrates the operations. For unordered trees the operations can be defined similarly. In this case, the insert and delete operations works on a *subset* instead of a subsequence. We define three problems based on the edit operations. Let T_1 and T_2 be labeled trees (ordered or unordered).

Tree edit distance: Assume that we are given a *cost function* defined on each edit operation. An *edit script* S between T_1 and T_2 is a sequence of edit operations turning T_1 into T_2. The cost of S is the sum of the costs of the operations in S. An *optimal edit script* between T_1 and T_2 is an edit script between T_1 and T_2 of minimum cost and this cost is the *tree edit distance*. The *tree edit distance problem* is to compute the edit distance and a corresponding edit script.

Tree alignment distance: Assume that we are given a cost function defined on pair of labels. An *alignment* A of T_1 and T_2 is obtained as follows. First we insert nodes labeled with *spaces* into T_1 and T_2 so that they become isomorphic when labels are ignored. The resulting trees are then *overlaid* on top of each other giving the alignment A, which is a tree where each node is labeled by a pair of labels. The *cost* of A is the sum of costs of all pairs of opposing labels in A. An *optimal alignment* of T_1 and T_2 is an alignment of minimum cost and this cost is called the *alignment distance* of T_1 and T_2. The *alignment distance problem* is to compute the alignment distance and a corresponding alignment.

Tree inclusion: T_1 is *included* in T_2 if and only if T_1 can be obtained by deleting nodes from T_2. The *tree inclusion problem* is to determine if T_1 is included in T_2.

In this paper we survey each of these problems and discuss the results obtained for them. For reference, Table 1 summarizes most of the available results. All of these and a few others are covered in the text. The tree edit distance problem is the most general of the problems. The alignment distance corresponds to a kind of restricted edit distance, while tree inclusion is a special case of both the edit and alignment distance problem. Apart from these simple relationships, interesting variations on the edit distance problem has been studied leading to a more complex picture.

Both the ordered and unordered version of the problems are reviewed. For the unordered case, it turns out that all of the problems in general are NP-hard. Indeed, the tree edit distance and alignment distance problems are even MAX SNP-hard [4]. However, under various interesting restrictions, or for special cases, polynomial time algorithms are available. For instance, if we impose a *structure preserving* restriction on the unordered tree edit distance problem, such that disjoint subtrees are mapped to disjoint subtrees, it can be solved in polynomial time. Also, unordered alignment for constant degree trees can be solved efficiently.

For the ordered version of the problems polynomial time algorithms exists. These are all based on the classic technique of *dynamic programming* (see, e.g., [9, Chapter 15]) and most of them are simple combinatorial algorithms. Recently, however, more advanced techniques such as fast matrix multiplication have been applied to the tree edit distance problem [8].

The survey covers the problems in the following way. For each problem and variations of it we review results for both the ordered and unordered version. This will, in most cases, include a formal definition of the problem, a comparison of the available results and a description of the techniques used to obtain the results. More importantly, we will also pick one or more of the central algorithms for each of the problems and present it in almost full detail. Specifically, we will describe the algorithm, prove that it is correct, and analyze its time complexity. For brevity, we will omit the proofs of a few lemmas and skip over some less important details. Common for the algorithms presented in detail is that, in most cases, they are the basis for more advanced algorithms. Typically, most of the algorithms for one of the above problems are refinements of the same dynamic programming algorithm.

The main technical contribution of this survey is to present the problems and algorithms in a common framework. Hopefully, this will enable the reader to gain a better overview and deeper understanding of the problems and how they relate to each other. In the literature, there are some discrepancies in the presentations of the problems. For instance, the ordered edit distance problem was considered by Klein [25] who used edit operations on edges. He presented an algorithm using a reduction to a problem defined on balanced parenthesis strings. In contrast, Zhang and Shasha [55] gave an algorithm based on the postorder numbering on trees. In fact, these algorithms share many features which become apparent if considered in the right setting. In this paper we present these algorithms in a new framework bridging the gap between the two descriptions.

Another problem in the literature is the lack of an agreement on a definition of the edit distance problem. The definition given here is by far the most studied and in our opinion the most natural. However, several alternatives ending in very different distance measures have been considered [30,45,38,31]. In this paper we review these other variants and compare them to our definition. We should note that the edit distance problem defined here is sometimes referred to as the *tree-to-tree correction problem*.

This survey adopts a *theoretical* point of view. However, the problems above are not only interesting mathematical problems but they also occur in many practical situations and it is important to develop algorithms that perform well on *real-life* problems. For practical issues see, e.g., [49,46,40].

We restrict our attention to *sequential* algorithms. However, there has been some research in parallel algorithms for the edit distance problem, e.g., [55,53,41].

This summarizes the contents of this paper. Due to the fundamental nature of comparing trees and its many applications several other ways to compare trees have been devised. In this paper, we have chosen to limit ourselves to a handful of problems which we describe in detail. Other problems include *tree pattern matching* [27,10] and [16,35,56], *maximum agreement subtree* [19,11], *largest common subtree* [2,20], and *smallest common supertree* [34,13].

1.1. Outline

In Section 2 we give some preliminaries. In Sections 3, 4, and 5 we survey the tree edit distance, alignment distance, and inclusion problems, respectively. We conclude in Section 6 with some open problems.

2. Preliminaries and notation

In this section we define notations and definitions that we use throughout the paper. For a graph G we denote the set of nodes and edges by $V(G)$ and $E(G)$, respectively. Let T be a rooted tree. The root of T is denoted by $\mathrm{root}(T)$. The *size* of T, denoted by $|T|$, is $|V(T)|$. The *depth* of a node $v \in V(T)$, $\mathrm{depth}(v)$, is the number of edges on the path from v to $\mathrm{root}(T)$. The *in-degree* of a node v, $\deg(v)$ is the number of children of v. We extend these definitions such that $\mathrm{depth}(T)$ and $\deg(T)$ denotes the maximum depth and degree, respectively, of any node in T. A node with no children is a leaf and otherwise an internal node. The number of leaves of T is denoted by $\mathrm{leaves}(T)$. We denote the parent of node v by $\mathrm{parent}(v)$. Two nodes are siblings if they have the same parent. For two trees T_1 and T_2, we will frequently refer to $\mathrm{leaves}(T_i)$, $\mathrm{depth}(T_i)$, and $\deg(T_i)$ by L_i, D_i, and I_i, $i = 1, 2$.

Let θ denote the empty tree and let $T(v)$ denote the subtree of T rooted at a node $v \in V(T)$. If $w \in V(T(v))$ then v is an ancestor of w, and if $w \in V(T(v))\backslash\{v\}$ then v is a proper ancestor of w. If v is a (proper) ancestor of w then w is a (proper) descendant of v. A tree T is *ordered* if a left-to-right order among the siblings is given. For an ordered tree T with root v and children $v_1, \ldots, v_i$, the *preorder traversal* of $T(v)$ is obtained by visiting v and then recursively visiting $T(v_k)$, $1 \leqslant k \leqslant i$, in order. Similarly, the *postorder traversal* is obtained by first visiting $T(v_k)$, $1 \leqslant k \leqslant i$, and then v. The *preorder number* and *postorder number* of a node $w \in T(v)$, denoted by $\mathrm{pre}(w)$ and $\mathrm{post}(w)$, is the number of nodes preceding w in the preorder and postorder traversal of T, respectively. The nodes to the *left* of w in T is the set of nodes $u \in V(T)$ such that $\mathrm{pre}(u) < \mathrm{pre}(w)$ and $\mathrm{post}(u) < \mathrm{post}(w)$. If u is to the left of w then w is to the *right* of u.

A forest is a set of trees. A forest F is ordered if a left-to-right order among the trees is given and each tree is ordered. Let T be an ordered tree and let $v \in V(T)$. If v has children $v_1, \dots, v_i$ define $F(v_s, v_t)$, where $1 \leqslant s \leqslant t \leqslant i$, as the forest $T(v_s), \dots, T(v_{\mathrm{r}})$. For convenience, we set $F(v) = F(v_1, v_i)$.

We assume throughout the paper that labels assigned to nodes are chosen from a finite alphabet Σ. Let $\lambda \notin \Sigma$ denote a special *blank* symbol and define $\Sigma_\lambda = \Sigma \cup \lambda$. We often define a *cost function*, $\gamma : (\Sigma_\lambda \times \Sigma_\lambda) \backslash (\lambda, \lambda) \to \mathbb{R}$, on pairs of labels. We will always assume that γ is a distance metric. That is, for any l_1,l_2,$l_3 \in \Sigma_\lambda$ the following conditions are satisfied:
1. $\gamma(l_1, l_2) \geqslant 0,\ \gamma(l_1, l_1) = 0$,
2. $\gamma(l_1, l_2) = \gamma(l_2, l_1)$,
3. $\gamma(l_1, l_3) \leqslant \gamma(l_1, l_2) + \gamma(l_2, l_3)$.

3. Tree edit distance

In this section we survey the tree edit distance problem. Assume that we are given a *cost function* defined on each edit operation. An *edit script S* between two trees T_1 and T_2 is a sequence of edit operations turning T_1 into T_2. The cost of S is the sum of the costs of the operations in S. An *optimal edit script* between T_1 and T_2 is an edit script between T_1 and T_2 of minimum cost. This cost is called the *tree edit distance*, denoted by $\delta(T_1, T_2)$. An example of an edit script is shown in Fig. 2.

The rest of the section is organized as follows. First, in Section 3.1, we present some preliminaries and formally define the problem. In Section 3.2 we survey the results obtained for the ordered edit distance problem and present two of the currently best algorithms for the problem. The unordered version of the problem is reviewed in Section 3.3. In Section 3.4 we review results on the edit distance problem when various *structure-preserving* constraints are imposed. Finally, in Section 3.5 we consider some other variants of the problem.

3.1. Edit operations and edit mappings

Let T_1 and T_2 be labeled trees. Following [43] we represent each edit operation by $(l_1 \to l_2)$, where $(l_1, l_2) \in (\Sigma_\lambda \times \Sigma_\lambda) \backslash (\lambda, \lambda)$. The operation is a relabeling if $l_1 \neq \lambda$ and $l_2 \neq \lambda$, a deletion if $l_2 = \lambda$, and an insertion if $l_1 = \lambda$. We extend the notation such that $(v \to w)$ for nodes v and w denotes $(\mathrm{label}(v) \to \mathrm{label}(w))$. Here, as with the labels, v or w may be λ. Given a metric cost function γ defined on pairs of labels we define the cost of an edit operation by setting $\gamma(l_1 \to l_2) = \gamma(l_1, l_2)$. The cost of a sequence $S = s_1, \dots, s_k$ of operations is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. The edit distance, $\delta(T_1, T_2)$, between T_1 and T_2 is formally defined as:

$$\delta(T_1, T_2) = \min\{\gamma(S) \mid S \text{ is a sequence of operations transforming } T_1 \text{ into } T_2\}.$$

Since γ is a distance metric δ becomes a distance metric too.

An *edit distance mapping* (or just a *mapping*) between T_1 and T_2 is a representation of the edit operations, which is used in many of the algorithms for the tree edit distance problem.

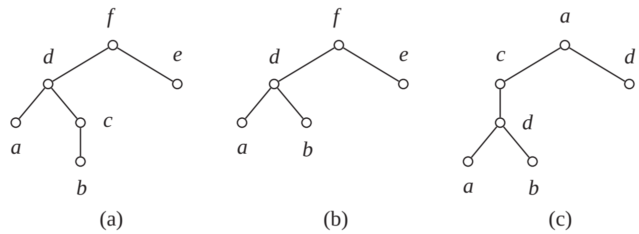

Fig. 2. Transforming (a) into (c) via editing operations. (a) A tree. (b) The tree after deleting the node labeled c. (c) The tree after inserting the node labeled c and relabeling f to a and e to d.

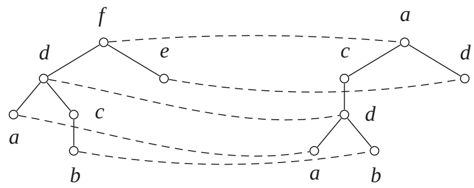

Fig. 3. The mapping corresponding to the edit script in Fig. 2.

Formally, define the triple (M, T_1, T_2) to be an *ordered edit distance mapping* from T_1 to T_2, if $M \subseteq V(T_1) \times V(T_2)$ and for any pair $(v_1, w_1), (v_2, w_2) \in M$:
1. $v_1 = v_2$ iff $w_1 = w_2$ (one-to-one condition).
2. v_1 is an ancestor of v_2 iff w_1 is an ancestor of w_2 (ancestor condition).
3. v_1 is to the left of v_2 iff w_1 is to the left of w_2 (sibling condition).

Fig. 3 illustrates a mapping that corresponds to the edit script in Fig. 2. We define the *unordered edit distance mapping* between two unordered trees as the same, but without the sibling condition. We will use M instead of (M, T_1, T_2) when there is no confusion. Let (M, T_1, T_2) be a mapping. We say that a node v in T_1 or T_2 is *touched by a line* in M if v occurs in some pair in M. Let N_1 and N_2 be the set of nodes in T_1 and T_2, respectively, not touched by any line in M. The cost of M is given by

$$\gamma(M) = \sum_{(v,w)\in M} \gamma(v \to w) + \sum_{v\in N_1} \gamma(v \to \lambda) + \sum_{w\in N_2} \gamma(\lambda \to w).$$

Mappings can be composed. Let T_1, T_2, and T_3 be labeled trees. Let M_1 and M_2 be a mapping from T_1 to T_2 and T_2 to T_3, respectively. Define

$$M_1 \circ M_2 = \{(v, w) \mid \exists u \in V(T_2) \text{ such that } (v, u) \in M_1 \text{ and } (u, w) \in M_2\}.$$

With this definition it follows easily that $M_1 \circ M_2$ itself becomes a mapping from T_1 to T_3. Since γ is a metric, it is not hard to show that a minimum cost mapping is equivalent to the edit distance:

$$\delta(T_1, T_2) = \min\{\gamma(M) \mid (M, T_1, T_2) \text{ is an edit distance mapping}\}.$$

Hence, to compute the edit distance we can compute the minimum cost mapping. We extend the definition of edit distance to forests. That is, for two forests F_1 and F_2, $\delta(F_1, F_2)$

denotes the edit distance between F_1 and F_2. The operations are defined as in the case of trees, however, roots of the trees in the forest may now be deleted and trees can be merged by inserting a new root. The definition of a mapping is extended in the same way.

3.2. General ordered edit distance

The ordered edit distance problem was introduced by Tai [43] as a generalization of the well-known *string edit distance problem* [48]. Tai presented an algorithm for the ordered version using $O(|T_1||T_2||L_1|^2|L_2|^2)$ time and space. Subsequently, Zhang and Shasha [55] gave a simple algorithm improving the bounds to $O(|T_1||T_2|\min(L_1, D_1)\min(L_2, D_2))$ time and $O(|T_1||T_2|)$ space. This algorithm was modified by Klein [25] to get a better worst-case time bound of $O(|T_1|^2|T_2|\log|T_2|)$ [1] under the same space bounds. We present the latter two algorithms in detail below. Recently, Chen [8] has presented an algorithm using $O(|T_1||T_2| + L_1^2|T_2| + L_1^{2.5}L_2)$ time and $O((|T_1| + L_1^2)\min(L_2, D_2) + |T_2|)$ space. Hence, for certain kinds of trees the algorithm improves the previous bounds. This algorithm is more complex than all the above and uses results on fast matrix multiplication. Note that in the above bounds we can exchange T_1 with T_2 since the distance is symmetric.

3.2.1. A simple algorithm

We first present a simple recursion which will form the basis for the two dynamic programming algorithms we present in the next two sections. We will only show how to compute the edit distance. The corresponding edit script can be easily obtained within the same time and space bounds. The algorithm is due to Klein [25]. However, we should note that the presentation given here is somewhat different. We believe that our framework is more simple and provides a better connection to previous work.

Let F be a forest and v be a node in F. We denote by $F - v$ the forest obtained by deleting v from F. Furthermore, define $F - T(v)$ as the forest obtained by deleting v and all descendants of v. The following lemma provides a way to compute edit distances for the general case of forests.

Lemma 1. *Let F_1 and F_2 be ordered forests and γ be a metric cost function defined on labels. Let v and w be the rightmost (if any) roots of the trees in F_1 and F_2, respectively. We have,*

$$\begin{aligned}
&\delta(\theta, \theta) = 0,\\
&\delta(F_1, \theta) = \delta(F_1 - v, \theta) + \gamma(v \to \lambda),\\
&\delta(\theta, F_2) = \delta(\theta, F_2 - w) + \gamma(\lambda \to w),\\
&\delta(F_1, F_2) = \min \begin{cases} \delta(F_1 - v, F_2) + \gamma(v \to \lambda),\\ \delta(F_1, F_2 - w) + \gamma(\lambda \to w),\\ \delta(F_1(v), F_2(w)) + \delta(F_1 - T_1(v), F_2 - T_2(w)) + \gamma(v \to w). \end{cases}
\end{aligned}$$

[1] Since the edit distance is symmetric this bound is in fact $O(\min(|T_1|^2|T_2|\log|T_2|, |T_2|^2|T_1|\log|T_1|)$. For brevity we will use the short version.

Proof. The first three equations are trivially true. To show the last equation consider a minimum cost mapping M between F_1 and F_2. There are three possibilities for v and w:

Case 1: v is not touched by a line. Then $(v, \lambda) \in M$ and the first case of the last equation applies.

Case 2: w is not touched by a line. Then $(\lambda, w) \in M$ and the second case of the last equation applies.

Case 3: v and w are both touched by lines. We show that this implies $(v, w) \in M$. Suppose (v, h) and (k, w) are in M. If v is to the right of k then h must be to right of w by the sibling condition. If v is a proper ancestor of k then h must be a proper ancestor of w by the ancestor condition. Both of these cases are impossible since v and w are the rightmost roots and hence $(v, w) \in M$. By the definition of mappings the equation follows. $\square$

Lemma 1 suggests a dynamic programming algorithm. The value of $\delta(F_1, F_2)$ depends on a constant number of subproblems of smaller size. Hence, we can compute $\delta(F_1, F_2)$ by computing $\delta(S_1, S_2)$ for all pairs of subproblems S_1 and S_2 in order of increasing size. Each new subproblem can be computed in constant time. Hence, the time complexity is bounded by the number of subproblems of F_1 times the number of subproblems of F_2.

To count the number of subproblems, define for a rooted, ordered forest F the (i, j)-*deleted subforest*, $0 \leqslant i + j \leqslant |F|$, as the forest obtained from F by first deleting the rightmost root repeatedly j times and then, similarly, deleting the leftmost root i times. We call the $(0, j)$-deleted and $(i, 0)$-deleted subforests, for $0 \leqslant j \leqslant |F|$, the *prefixes* and the *suffixes* of F, respectively. The number of (i, j)-deleted subforests of F is $\sum_{k=0}^{|F|} k = \mathrm{O}(|F|^2)$, since for each i there are $|F| - i$ choices for j.

It is not hard to show that all the pairs of subproblems S_1 and S_2 that can be obtained by the recursion of Lemma 1 are deleted subforests of F_1 and F_2. Hence, by the above discussion the time complexity is bounded by $\mathrm{O}(|F_1|^2|F_2|^2)$. In fact, fewer subproblems are needed, which we will show in the next sections.

3.2.2. Zhang and Shasha's algorithm

The following algorithm is due to Zhang and Shasha [55]. Define the *keyroots* of a rooted, ordered tree T as follows:

$$\mathrm{keyroots}(T) = \{\mathrm{root}(T)\} \cup \{v \in V(T) \mid v \text{ has a left sibling}\}.$$

The *special* subforests of T is the forests $F(v)$, where $v \in \mathrm{keyroots}(T)$. The *relevant subproblems of T with respect to the keyroots* is the prefixes of all special subforests $F(v)$. In this section we refer to these as the *relevant subproblems*.

Lemma 2. *For each node $v \in V(T)$, $F(v)$ is a relevant subproblem.*

It is easy to see that, in fact, the subproblems that can occur in the above recursion are either subforests of the form $F(v)$, where $v \in V(T)$, or prefixes of a special subforest of T. Hence, it follows by Lemma 2 and the definition of a relevant subproblem, that to compute $\delta(F_1, F_2)$ it is sufficient to compute $\delta(S_1, S_2)$ for all relevant subproblems S_1 and S_2 of T_1 and T_2, respectively.

The relevant subproblems of a tree T can be counted as follows. For a node $v \in V(T)$ define the *collapsed depth* of v, $\mathrm{cdepth}(v)$, as the number of keyroot ancestors of v. Also, define $\mathrm{cdepth}(T)$ as the maximum collapsed depth of all nodes $v \in V(T)$.

Lemma 3. *For an ordered tree T the number of relevant subproblems, with respect to the keyroots is bounded by* $\mathrm{O}(|T|\mathrm{cdepth}(T))$.

Proof. The relevant subproblems can be counted using the following expression:

$$\sum_{v \in \mathrm{keyroots}(T)} |F(v)| < \sum_{v \in \mathrm{keyroots}(T)} |T(v)| = \sum_{v \in V(T)} \mathrm{cdepth}(v) \leqslant |T|\mathrm{cdepth}(T)$$

Since the number prefixes of a subforest $F(v)$ is $|F(v)|$ the first sum counts the number of relevant subproblems of $F(v)$. To prove the first equality note that for each node v the number of special subforests containing v is the collapsed depth of v. Hence, v contributes the same amount to the left and right side. The other equalities/inequalities follow immediately. □

Lemma 4. *For a tree T,* $\mathrm{cdepth}(T) \leqslant \min\{\mathrm{depth}(T), \mathrm{leaves}(T)\}$

Thus, using dynamic programming it follows that the problem can be solved in $\mathrm{O}(|T_1||T_2| \min\{D_1, L_1\} \min\{D_2, L_2\})$ time and space. To improve the space complexity we carefully compute the subproblems in a specific order and discard some of the intermediate results. Throughout the algorithm we maintain a table called the *permanent table* storing the distances $\delta(F_1(v), F_2(w))$, $v_1 \in V(F_1)$ and $w_2 \in V(F_2)$, as they are computed. This uses $\mathrm{O}(|F_1||F_2|)$ space. When the distances of all special subforests of F_1 and F_2 are available in the permanent table, we compute the distance between all prefixes of F_1 and F_2 in order of increasing size and store these in a table called the *temporary table*. The values of the temporary table that are distances between special subforests are copied to the permanent table and the rest of the values are discarded. Hence, the temporary table also uses at most $\mathrm{O}(|F_1||F_2|)$ space. By Lemma 1 it is easy to see that all values needed to compute $\delta(F_1, F_2)$ are available. Hence,

Theorem 1 (*Zhang and Shasha [55]*). *For ordered trees T_1 and T_2 the edit distance problem can be solved in time* $\mathrm{O}(|T_1||T_2| \min\{D_1, L_1\} \min\{D_2, L_2\})$ *and space* $\mathrm{O}(|T_1||T_2|)$.

3.2.3. Klein's algorithm

In the worst case, that is for trees with linear depth and a linear number of leaves, Zhang and Shasha's algorithm of the previous section still requires $\mathrm{O}(|T_1|^2|T_2|^2)$ time as the simple algorithm. In [25] Klein obtained a better worst-case time bound of $\mathrm{O}(|T_1|^2|T_2| \log |T_2|)$. The reported space complexity of the algorithm is $\mathrm{O}(|T_1|^2|T_2| \log |T_2|)$ which is significantly worse than the algorithm of Zhang and Shasha. However, according to Klein [23] this algorithm can also be improved to $\mathrm{O}(|T_1||T_2|)$.

The algorithm is based on an extension of the recursion in Lemma 1. The main idea is to consider all of the $\mathrm{O}(|T_1|^2)$ deleted subforests of T_1 but only $\mathrm{O}(|T_2| \log |T_2|)$ deleted subforests of T_2. In total the worst-case number of subproblems is thus reduced to the desired bound above.

A key concept in the algorithm is the decomposition of a rooted tree T into disjoint paths called *heavy paths*. This technique was introduced by Harel and Tarjan [15]. We define the *size* a node $v \in V(T)$ as $|T(v)|$. We classify each node of T as either *heavy* or *light* as follows. The root is light. For each internal node v we pick a child u of v of maximum size among the children of v and classify u as heavy. The remaining children are light. We call an edge to a light child a *light edge*, and an edge to a heavy child a *heavy edge*. The *light depth* of a node v, $\mathrm{ldepth}(v)$, is the number of light edges on the path from v to the root.

Lemma 5 (*Harel and Tarjan [15]*). *For any tree T and any* $v \in V(T)$, $\mathrm{ldepth}(v) \leqslant \log|T| + \mathrm{O}(1)$.

By removing the light edges T is partitioned into heavy paths.

We define the *relevant subproblems of T with respect to the light nodes* below. We will refer to these as *relevant subproblems* in this section. First fix a heavy path decomposition of T. For a node v in T we recursively define the relevant subproblems of $F(v)$ as follows: $F(v)$ is relevant. If v is not a leaf, let u be the heavy child of v and let l and r be the number of nodes to the left and to the right of u in $F(v)$, respectively. Then, the $(i, 0)$-deleted subforests of $F(v)$, $0 \leqslant i \leqslant l$, and the (l, j)-deleted subforests of $F(v)$, $0 \leqslant j \leqslant r$ are relevant subproblems. Recursively, all relevant subproblems of $F(u)$ are relevant.

The relevant subproblems of T with respect to the light nodes is the union of all relevant subproblems of $F(v)$ where $v \in V(T)$ is a light node.

Lemma 6. *For an ordered tree T the number of relevant subproblems with respect to the light nodes is bounded by* $\mathrm{O}(|T|\,\mathrm{ldepth}(T))$.

Proof. Follows by the same calculation as in the proof of Lemma 3. □

Also note that Lemma 2 still holds with this new definition of relevant subproblems. Let S be a relevant subproblem of T and let v_l and v_r denote the leftmost and rightmost root of S, respectively. The *difference node* of S is either v_r if $S - v_\mathrm{r}$ is relevant or v_l if $S - v_\mathrm{l}$ is relevant. The recursion of Lemma 1 compares the rightmost roots. Clearly, we can also choose to compare the leftmost roots resulting in a new recursion, which we will refer to as the *dual* of Lemma 1. Depending on which recursion we use, different subproblems occur. We now give a modified dynamic programming algorithm for calculating the tree edit distance. Let S_1 be a deleted tree of T_1 and let S_2 be a relevant subproblem of T_2. Let d be the difference node of S_2. We compute $\delta(S_1, S_2)$ as follows. There are two cases to consider:

1. If d is the rightmost root of S_2 compare the rightmost roots of S_1 and S_2 using Lemma 1.
2. If d is the leftmost root of S_2 compare the leftmost roots of S_1 and S_2 using the dual of Lemma 1.

It is easy to show that in both cases the resulting smaller subproblems of S_1 will all be deleted subforests of T_1 and the smaller subproblems of S_2 will all be relevant subproblems of T_2. Using a similar dynamic programming technique as in the algorithm of Zhang and Shasha we obtain the following:

Theorem 2 (*Klein [25]*). *For ordered trees T_1 and T_2 the edit distance problem can be solved in time and space* $O(|T_1|^2|T_2|\log|T_2|)$.

Klein [25] also showed that his algorithm can be extended within the same time and space bounds to the *unrooted ordered edit distance problem* between T_1 and T_2, defined as the minimum edit distance between T_1 and T_2 over all possible roots of T_1 and T_2.

3.3. General unordered edit distance

In the following section we survey the unordered edit distance problem. This problem has been shown to be NP-complete [58,50,57] even for binary trees with a label alphabet of size 2. The reduction is from the Exact Cover by 3-Sets problem [12]. Subsequently, the problem was shown to be MAX SNP-hard [54]. Hence, unless $P = NP$ there is no PTAS for the problem [4]. It was shown in [58] that for special cases of the problem polynomial time algorithms exists. If T_2 has one leaf, i.e., T_2 is a sequence, the problem can be solved in $O(|T_1||T_2|)$ time. More generally, there is an algorithm running in time $O(|T_1||T_2| + L_2!3^{L_2}(L_2^3 + D_1^2)|T_1|)$. Hence, if the number of leaves in T_2 is logarithmic the problem can be solved in polynomial time.

3.4. Constrained edit distance

The fact that the general edit distance problem is difficult to solve has led to the study of restricted versions of the problem. In [51,52] Zhang introduced the *constrained edit distance*, denoted by δ_c, which is defined as an edit distance under the restriction that disjoint subtrees should be mapped to disjoint subtrees. Formally, $\delta_c(T_1, T_2)$ is defined as a minimum cost mapping (M_c, T_1, T_2) satisfying the additional constraint, that for all $(v_1, w_1), (v_2, w_2), (v_3, w_3) \in M_c$:

- $\mathrm{nca}(v_1, v_2)$ is a proper ancestor of v_3 iff $\mathrm{nca}(w_1, w_2)$ is a proper ancestor of w_3.

According to [29], Richter [37] independently introduced the *structure respecting edit distance* δ_s. Similar to the constrained edit distance, $\delta_s(T_1, T_2)$ is defined as a minimum cost mapping (M_s, T_1, T_2) satisfying the additional constraint, that for all $(v_1, w_1), (v_2, w_2), (v_3, w_3) \in M_s$ such that none of v_1, v_2, and v_3 is an ancestor of the others,

- $\mathrm{nca}(v_1, v_2) = \mathrm{nca}(v_1, v_3)$ iff $\mathrm{nca}(w_1, w_2) = \mathrm{nca}(w_1, w_3)$.

It is straightforward to show that both of these notions of edit distance are equivalent. Henceforth, we will refer to them simply as the constrained edit distance. As an example consider the mappings of Fig. 4. (a) is a constrained mapping since $\mathrm{nca}(v_1, v_2) \neq \mathrm{nca}(v_1, v_3)$ and $\mathrm{nca}(w_1, w_2) \neq \mathrm{nca}(w_1, w_3)$. (b) is not constrained since $\mathrm{nca}(v_1, v_2) = v_4 \neq \mathrm{nca}(v_1, v_3) = v_5$, while $\mathrm{nca}(w_1, w_2) = w_4 = \mathrm{nca}(w_1, w_3)$. (c) is not constrained since $\mathrm{nca}(v_1, v_3) = v_5 \neq \mathrm{nca}(v_2, v_3)$, while $\mathrm{nca}(w_1, w_3) = v_5 \neq \mathrm{nca}(w_2, w_3) = w_4$.

In [51,52] Zhang presents algorithms for computing minimum cost constrained mappings. For the ordered case he gives an algorithm using $O(|T_1||T_2|)$ time and for the unordered case he obtains a running time of $O(|T_1||T_2|(I_1 + I_2)\log(I_1 + I_2))$. Both use space $O(|T_1||T_2|)$. The idea in both algorithms is similar. Due to the restriction on the mappings fewer subproblem need to be considered and a faster dynamic programming algorithm is obtained. In

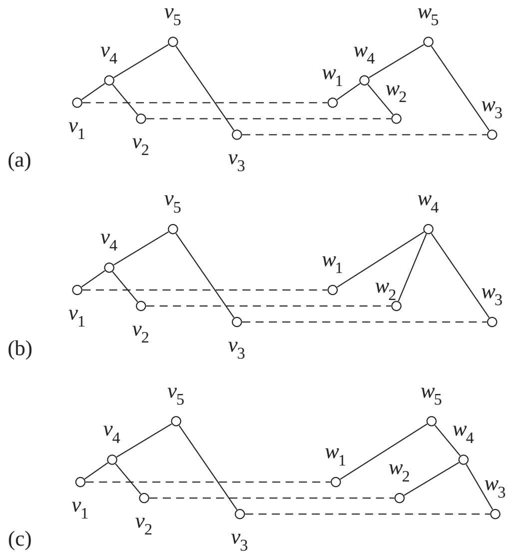

Fig. 4. (a) A mapping which is constrained and less-constrained. (b) A mapping which is less-constrained but not constrained. (c) A mapping which is neither constrained nor less-constrained.

the ordered case the key observation is a reduction to the string edit distance problem. For the unordered case the corresponding reduction is to a maximum matching problem. Using an efficient algorithm for computing a minimum cost maximum flow Zhang obtains the time complexity above. Richter presented an algorithm for the ordered constrained edit distance problem, which uses $O(|T_1||T_2|I_1 I_2)$ time and $O(|T_1|D_2 I_2)$ space. Hence, for small degree, low depth trees this algorithm gives a space improvement over the algorithm of Zhang.

Recently, Lu et al. [29] introduced the *less-constrained edit distance*, δ_l, which relaxes the constrained mapping. The requirement here is that for all $(v_1, w_1), (v_2, w_2), (v_3, w_3) \in M_l$ such that none of v_1, v_2, and v_3 is an ancestor of the others,

- $\mathrm{depth}(\mathrm{nca}(v_1, v_2)) \geqslant \mathrm{depth}(\mathrm{nca}(v_1, v_3))$ and also $\mathrm{nca}(v_1, v_3) = \mathrm{nca}(v_2, v_3)$ if and only if $\mathrm{depth}(\mathrm{nca}(w_1, w_2)) \geqslant \mathrm{depth}(\mathrm{nca}(w_1, w_3))$ and $\mathrm{nca}(w_1, w_3) = \mathrm{nca}(w_2, w_3)$.

For example, consider the mappings in Fig. 4. (a) is less-constrained because it is constrained. (b) is not a constrained mapping, however, the mapping is less-constrained since $\mathrm{depth}(\mathrm{nca}(v_1, v_2)) > \mathrm{depth}(\mathrm{nca}(v_1, v_3))$, $\mathrm{nca}(v_1, v_3) = \mathrm{nca}(v_2, v_3)$, $\mathrm{nca}(w_1, w_2) = \mathrm{nca}(w_1, w_3)$, and $\mathrm{nca}(w_1, w_3) = \mathrm{nca}(w_2, w_3)$. (c) is not a less-constrained mapping since $\mathrm{depth}(\mathrm{nca}(v_1, v_2)) > \mathrm{depth}(\mathrm{nca}(v_1, v_3))$ and $\mathrm{nca}(v_1, v_3) = \mathrm{nca}(v_2, v_3)$, while $\mathrm{nca}(w_1, w_3) \neq \mathrm{nca}(w_2, w_3)$.

In paper [29] an algorithm for the ordered version of the less-constrained edit distance problem using $O(|T_1||T_2|I_1^3 I_2^3(I_1 + I_2))$ time and space is presented. For the unordered version, unlike the constrained edit distance problem, it is shown that the problem is NP-complete. The reduction used is similar to the one for the unordered edit distance problem. It is also reported that the problem is MAX SNP-hard. Furthermore, it is shown that there is no absolute approximation algorithm [2] for the unordered less-constrained edit distance problem unless P = NP.

3.5. Other variants

In this section we survey results for other variants of edit distance. Let T_1 and T_2 be rooted trees. The *unit cost edit distance* between T_1 and T_2 is defined as the number of edit operations needed to turn T_1 into T_2. In [41] the ordered version of this problem is considered and a fast algorithm is presented. If u is the unit cost edit distance between T_1 and T_2 the algorithm runs in $O(u^2 \min\{|T_1|, |T_2|\} \min\{L_1, L_2\})$ time. The algorithm uses techniques from Ukkonen [47] and Landau and Vishkin [28].

In [38] Selkow considered an edit distance problem where insertions and deletions are restricted to leaves of the trees. This edit distance is sometimes referred to as the *1-degree edit distance*. He gave a simple algorithm using $O(|T_1||T_2|)$ time and space. Another edit distance measure where edit operations work on subtrees instead of nodes was given by Lu [30]. A similar edit distance was given by Tanaka in [45,44]. A short description of Lu's algorithm can be found in [42].

4. Tree alignment distance

In this section we consider the alignment distance problem. Let T_1 and T_2 be rooted, labeled trees and let γ be a metric cost function on pairs of labels as defined in Section 2. An alignment A of T_1 and T_2 is obtained by first inserting nodes labeled with λ (called *spaces*) into T_1 and T_2 so that they become isomorphic when labels are ignored, and then *overlaying* the first augmented tree on the other one. The *cost* of a pair of opposing labels in A is given by γ. The cost of A is the sum of costs of all opposing labels in A. An *optimal alignment* of T_1 and T_2, is an alignment of T_1 and T_2 of minimum cost. We denote this cost by $\alpha(T_1, T_2)$. Fig. 5 shows an example (from [18]) of an ordered alignment.

The tree alignment distance problem is a special case of the tree editing problem. In fact, it corresponds to a restricted edit distance where all insertions must be performed before any deletions. Hence, $\delta(T_1, T_2) \leqslant \alpha(T_1, T_2)$. For instance, assume that all edit operations have cost 1 and consider the example in Fig. 5. The optimal sequence of edit operations is achieved by deleting the node labeled *e* and then inserting the node labeled *f*. Hence, the edit distance is 2. The optimal alignment, however, is the tree depicted in (c) with a value of 4. Additionally, it also follows that the alignment distance does not satisfy the triangle inequality and hence it is not a distance metric. For instance, in Fig. 5 if T_3 is T_1 where the node labeled *e* is deleted, then $\alpha(T_1, T_3) + \alpha(T_3, T_2) = 2 > 4 = \alpha(T_1, T_2)$.

It is a well-known fact that edit and alignment distance are equivalent in terms of complexity for sequences, see, e.g., Gusfield [14]. However, for trees this is not true which we will show in the following sections. In Section 4.1 and Section 4.2 we survey the results for the ordered and unordered tree alignment distance problem, respectively.

---

[2] An approximation algorithm A is *absolute* if there exists a constant $c > 0$ such that for every instance I, $|A(I) - \mathrm{OPT}(I)| \leqslant c$, where $A(I)$ and $\mathrm{OPT}(I)$ are the approximate and optimal solutions of I, respectively [33].

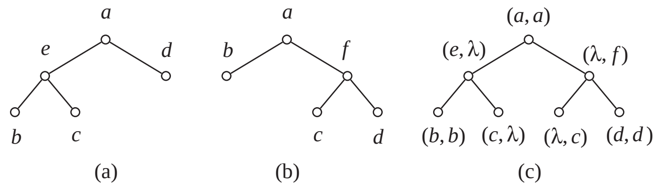

Fig. 5. (a) Tree T_1. (b) Tree T_2. (c) An alignment of T_1 and T_2.

4.1. Ordered tree alignment distance

In this section we consider the ordered tree alignment distance problem. Let T_1 and T_2 be two rooted, ordered and labeled trees. The ordered tree alignment distance problem was introduced by Jiang et al. in [18]. The algorithm presented there uses $O(|T_1||T_2|(I_1 + I_2)^2)$ time and $O(|T_1||T_2|(I_1+I_2))$ space. Hence, for small degree trees, this algorithm is in general faster than the best known algorithm for the edit distance. We present this algorithm in detail in the next section. Recently, in [17], a new algorithm was proposed designed for *similar* trees. Specifically, if there is an optimal alignment of T_1 and T_2 using at most s spaces the algorithm computes the alignment in time $O((|T_1|+|T_2|)\log(|T_1|+|T_2|)(I_1+I_2)^4 s^2)$. This algorithm works in a way similar to the fast algorithms for comparing similar sequences, see, e.g., Section 3.3.4 in [39]. The main idea is to speedup the algorithm of Jiang et al. by only considering subtrees of T_1 and T_2 whose sizes differ by at most $O(s)$.

4.1.1. Jiang, Wang, and Zhang's algorithm

In this section we present the algorithm of Jiang et al. [18]. We only show how to compute the alignment distance. The corresponding alignment can easily be constructed within the same complexity bounds. Let γ be a metric cost function on the labels. For simplicity, we will refer to nodes instead of labels, that is, we will use (v, w) for nodes v and w to mean $(\text{label}(v), \text{label}(w))$. Here, v or w may be λ. We extend the definition of α to include alignments of forests, that is, $\alpha(F_1, F_2)$ denotes the cost of an optimal alignment of forest F_1 and F_2.

Lemma 7. *Let $v \in V(T_1)$ and $w \in V(T_2)$ with children $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$, respectively. Then,*

$$\alpha(\theta, \theta) = 0,$$

$$\alpha(T_1(v), \theta) = \alpha(F_1(v), \theta) + \gamma(v, \lambda),$$

$$\alpha(\theta, T_2(w)) = \alpha(\theta, F_2(w)) + \gamma(\lambda, w),$$

$$\alpha(F_1(v), \theta) = \sum_{k=1}^{i} \alpha(T_1(v_k), \theta),$$

$$\alpha(\theta, F_2(w)) = \sum_{k=1}^{j} \alpha(\theta, T_2(w_k)).$$

Lemma 8. *Let $v \in V(T_1)$ and $w \in V(T_2)$ with children $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$, respectively. Then,*

$$\alpha(T_1(v), T_2(w)) = \min \begin{cases} \alpha(F_1(v), F_2(w)) + \gamma(v, w), \\ \alpha(\theta, T_2(w)) + \min_{1 \leqslant r \leqslant j}\{\alpha(T_1(v), T_2(w_r)) - \alpha(\theta, T_2(w_r)\}, \\ \alpha(T_1(v), \theta) + \min_{1 \leqslant r \leqslant i}\{\alpha(T_1(v_{\mathrm{r}}), T_2(w)) - \alpha(T_1(v_{\mathrm{r}}), \theta)\}. \end{cases}$$

Proof. Consider an optimal alignment A of $T_1(v)$ and $T_2(w)$. There are four cases: (1) (v, w) is a label in A, (2) (v, λ) and (k, w) are labels in A for some $k \in V(T_1)$, (3) (λ, w) and (v, h) are labels in A for some $h \in V(T_2)$ or (4) (v, λ) and (λ, w) are in A. Case (4) need not be considered since the two nodes can be deleted and replaced by the single node (v, w) as the new root. The cost of the resulting alignment is by the triangle inequality at least as small.

Case 1: The root of A is labeled by (v, w). Hence,

$$\alpha(T_1(v), T_2(w)) = \alpha(F_1(v), F_2(w)) + \gamma(v, w)$$

Case 2: The root of A is labeled by (v, λ). Hence, $k \in V(T_1(w_s))$ for some $1 \leqslant r \leqslant i$. It follows that,

$$\alpha(T_1(v), T_2(w)) = \alpha(T_1(v), \theta) + \min_{1 \leqslant r \leqslant i}\{\alpha(T_1(v_{\mathrm{r}}), T_2(w)) - \alpha(T_1(v_{\mathrm{r}}), \theta)\}$$

Case 3: Symmetric to case 2. □

Lemma 9. *Let $v \in V(T_1)$ and $w \in V(T_2)$ with children $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$, respectively. For any s, t such that $1 \leqslant s \leqslant i$ and $1 \leqslant t \leqslant j$,*

$$\alpha(F_1(v_1, v_s), F_2(w_1, w_t)) = \min \begin{cases} \alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_{t-1})) + \alpha(T_1(v_s), T_2(w_t)), \\ \alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_t)) + \alpha(T_1(v_s), \theta), \\ \alpha(F_1(v_1, v_s), F_2(w_1, w_{t-1})) + \alpha(\theta, T_2(w_t)), \\ \gamma(\lambda, w_t) + \min_{1 \leqslant k < s}\{\alpha(F_1(v_1, v_{k-1}), F_2(w_1, w_{t-1})) \\ \quad + \alpha(F_1(v_k, v_s), F_2(w_k))\}, \\ \gamma(v_s, \lambda) + \min_{1 \leqslant k < t}\{\alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_{k-1})) \\ \quad + \alpha(F_1(v_s), F_2(w_k, w_t))\}. \end{cases}$$

Proof. Consider an optimal alignment A of $F_1(v_1, v_s)$ and $F_2(w_1, w_t)$. The root of the rightmost tree in A is labeled either (v_s, w_t), (v_s, λ) or (λ, w_t).

Case 1: The label is (v_s, w_t). Then the rightmost tree of A must be an optimal alignment of $T_1(v_s)$ and $T_2(w_t)$. Hence,

$$\alpha(F_1(v_1, v_s), F_2(w_1, w_t)) = \alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_{t-1})) + \alpha(T_1(v_s), T_2(w_t)).$$

Case 2: The label is (v_s, λ). Then $T_1(v_s)$ is a aligned with a subforest $F_2(w_{t-k+1}, w_t)$, where $0 \leqslant k \leqslant t$. The following subcases can occur:

Subcase 2.1 ($k = 0$): $T_1(v_s)$ is aligned with $F_2(w_{t-k+1}, w_t) = \theta$. Hence,

$$\alpha(F_1(v_1, v_s), F_2(w_1, w_t)) = \alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_t)) + \alpha(T_1(v_s), \theta).$$

Subcase 2.2 ($k = 1$): $T_1(v_s)$ is aligned with $F_2(w_{t-k+1}, w_t) = T_2(w_t)$. Similar to case 1.

Subcase 2.3 ($k \geqslant 2$): The most general case. It is easy to see that:

$$\begin{aligned}\alpha(F_1(v_1, v_s), F_2(w_1, w_t)) = {} & \gamma(v_s, \lambda) + \min_{1 \leqslant r < t} (\alpha(F_1(v_1, v_{s-1}), F_2(w_1, w_{k-1}))) \\ & + \alpha(F_1(v_s), F_2(w_k, w_t)).\end{aligned}$$

Case 3: The label is (λ, w_t). Symmetric to case 2. □

This recursion can be used to construct a bottom-up dynamic programming algorithm. Consider a fixed pair of nodes v and w with children $v_1, \ldots, v_i$ and $w_1, \ldots, w_j$, respectively. We need to compute the values $\alpha(F_1(v_h, v_k), F_2(w))$ for all $1 \leqslant h \leqslant k \leqslant i$, and $\alpha(F_1(v), F_2(w_h, w_k))$ for all $1 \leqslant h \leqslant k \leqslant j$. That is, we need to compute the optimal alignment of $F_1(v)$ with each subforest of $F_2(w)$ and, on the other hand, compute the optimal alignment of $F_2(w)$ with each subforest of $F_1(v)$. For any s and t, $1 \leqslant s \leqslant i$ and $1 \leqslant t \leqslant j$, define the set:

$$A_{s,t} = \{\alpha(F_1(v_s, v_p), F_2(w_t, w_q)) \mid s \leqslant p \leqslant i, t \leqslant q \leqslant j\}.$$

To compute the alignments described above we need to compute $A_{s,1}$ and $A_{1,t}$ for all $1 \leqslant s \leqslant i$ and $1 \leqslant t \leqslant j$. Assuming that values for smaller subproblems are known it is not hard to show that $A_{s,t}$ can be computed, using Lemma 9, in time $\mathrm{O}((i-s)\cdot(j-t)\cdot(i-s+j-t)) = \mathrm{O}(ij(i+j))$. Hence, the time to compute the $(i+j)$ subproblems, $A_{s,1}$ and $A_{1,t}$, $1 \leqslant s \leqslant i$ and $1 \leqslant t \leqslant j$, is bounded by $\mathrm{O}(ij(i+j)^2)$. It follows that the total time needed for all nodes v and w is bounded by:

$$\begin{aligned}&\sum_{v \in V(T_1)} \sum_{w \in V(T_2)} \mathrm{O}(\deg(v)\deg(w)(\deg(v)+\deg(w))^2) \\ &\quad \leqslant \sum_{v \in V(T_1)} \sum_{w \in V(T_2)} \mathrm{O}(\deg(v)\deg(w)(\deg(T_1)+\deg(T_2))^2) \\ &\quad \leqslant \mathrm{O}\left((I_1+I_2)^2 \sum_{v \in V(T_1)} \sum_{w \in V(T_2)} \deg(v)\deg(w)\right) \\ &\quad \leqslant \mathrm{O}(|T_1||T_2|(I_1+I_2)^2).\end{aligned}$$

In summary, we have shown the following theorem.

Theorem 3 (*Jiang et al. [18]*). *For ordered trees T_1 and T_2, the tree alignment distance problem can be solved in* $\mathrm{O}(|T_1||T_2|(I_1+I_2)^2)$ *time and* $\mathrm{O}(|T_1||T_2|(I_1+I_2))$ *space.*

4.2. Unordered tree alignment distance

The algorithm presented above can be modified to handle the unordered version of the problem in a straightforward way [18]. If the trees have bounded degrees the algorithm still runs in $O(|T_1||T_2|)$ time. This should be seen in contrast to the edit distance problem which is MAX SNP-hard even if the trees have bounded degree. If one tree has arbitrary degree unordered alignment becomes NP-hard [18]. The reduction is, as for the edit distance problem, from the Exact Cover by 3-Sets problem [12].

5. Tree inclusion

In this section we survey the tree inclusion problem. Let T_1 and T_2 be rooted, labeled trees. We say that T_1 is *included* in T_2 if there is a sequence of delete operations performed on T_2 which makes T_2 isomorphic to T_1. The *tree inclusion problem* is to decide if T_1 is included in T_2. Fig. 6(a) shows an example of an ordered inclusion. The tree inclusion problem is a special case of the tree edit distance problem: If insertions all have cost 0 and all other operations have cost 1, then T_1 can be included in T_2 if and only if $\delta(T_1, T_2) = 0$. According to [7] the tree inclusion problem was initially introduced by Knuth [26, exercise 2.3.2-22].

The rest of the section is organized as follows. In Sections 5.1 we give some preliminaries and in Sections 5.2 and 5.3 we survey the known results on ordered and unordered tree inclusion, respectively.

5.1. Orderings and embeddings

Let T be a labeled, ordered, and rooted tree. We define an ordering of the nodes of T given by $v \prec v'$ iff $\text{post}(v) < \text{post}(v')$. Also, $v \preceq v'$ iff $v \prec v'$ or $v = v'$. Furthermore, we extend this ordering with two special nodes $\bot$ and $\top$ such that for all nodes $v \in V(T)$, $\bot \prec v \prec \top$. The *left relatives*, $\text{lr}(v)$, of a node $v \in V(T)$ is the set of nodes that are to the left of v and similarly the *right relatives*, $\text{rr}(v)$, are the set of nodes that are to the right of v.

Let T_1 and T_2 be rooted labeled trees. We define an *ordered embedding* (f, T_1, T_2) as an injective function $f : V(T_1) \to V(T_2)$ such that for all nodes $v, u \in V(T_1)$,

- $\text{label}(v) = \text{label}(f(v))$ (label preservation condition).
- v is an ancestor of u iff $f(v)$ is an ancestor of $f(u)$ (ancestor condition).
- is to the left of u iff $f(v)$ is to the left of $f(u)$ (sibling condition).

Hence, embeddings are special cases of mappings (see Section 3.1). An *unordered embedding* is defined as above, but without the sibling condition. An embedding (f, T_1, T_2) is *root-preserving* if $f(\text{root}(T_1)) = \text{root}(T_2)$. Fig. 6(b) shows an example of a root-preserving embedding.

5.2. Ordered tree inclusion

Let T_1 and T_2 be rooted, ordered and labeled trees. The ordered tree inclusion problem has been the attention of much research. Kilpeläinen and Mannila [22] (see also [21]) presented

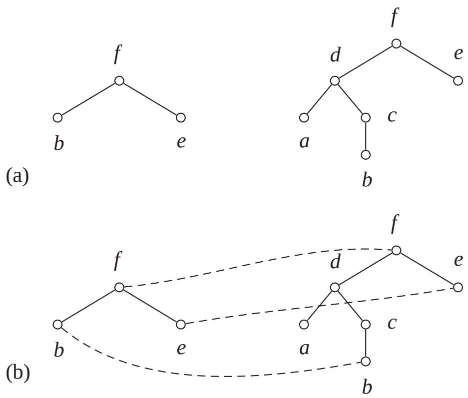

Fig. 6. (a) The tree on the left is included in the tree on the right by deleting the nodes labeled *d*, *a* and *c*. (b) The embedding corresponding to (a).

the first polynomial time algorithm using $\mathrm{O}(|T_1||T_2|)$ time and space. Most of the later improvements are refinements of this algorithm. We present this algorithm in detail in the next section. In [21] a more space efficient version of the above was given using $\mathrm{O}(|T_1|D_2)$ space. In [36] Richter gave an algorithm using $\mathrm{O}(|\Sigma_{T_1}||T_2| + m_{T_1,T_2}D_2)$ time, where Σ_{T_1} is the alphabet of the labels of T_1 and m_{T_1,T_2} is the set *matches*, defined as the number of pairs $(v, w) \in T_1 \times T_2$ such that $\mathrm{label}(v) = \mathrm{label}(w)$. Hence, if the number of matches is small the time complexity of this algorithm improves the $(|T_1||T_2|)$ algorithm. The space complexity of the algorithm is $\mathrm{O}(|\Sigma_{T_1}||T_2| + m_{T_1,T_2})$. In [7] a more complex algorithm was presented using $\mathrm{O}(L_1|T_2|)$ time and $\mathrm{O}(L_1 \min\{D_2, L_2\})$ space. In [3] an efficient average case algorithm was given.

5.2.1. Kilpeläinen and Mannila's algorithm

In this section we present the algorithm of Kilpeläinen and Mannila [22] for the ordered tree inclusion problem. Let T_1 and T_2 be ordered labeled trees. Define $R(T_1, T_2)$ as the set of root-preserving embeddings of T_1 into T_2. We define $\rho(v, w)$, where $v \in V(T_1)$ and $w \in V(T_2)$:

$$\rho(v, w) = \min_{\prec}(\{w' \in rr(w) \mid \exists f \in R(T_1(v), T_2(w'))\} \cup \{\top\}).$$

Hence, $\rho(v, w)$ is the closest right relative of w which has a root-preserving embedding of $T_1(v)$. Furthermore, if no such embedding exists $\rho(v, w)$ is $\top$. It is easy to see that, by definition, T_1 can be included in T_2 if and only if $\rho(v, \bot) \neq \top$. The following lemma shows how to search for root preserving embeddings.

Lemma 10. *Let v be a node in T_1 with children $v_1, \ldots, v_i$. For a node w in T_2, define a sequence $p_1, \ldots, p_i$ by setting $p_1 = \rho(v_1, \max_{\prec} \mathrm{lr}(w))$ and $p_k = \rho(v_k, p_{k-1})$, for $2 \leqslant k \leqslant i$. There is a root-preserving embedding f of $T_1(v)$ in $T_2(v)$ if and only if* $\mathrm{label}(v) = \mathrm{label}(w)$ *and $p_i \in T_2(w)$, for all $1 \leqslant k \leqslant i$.*

Proof. If there is a root-preserving embedding between $T_1(v)$ and $T_2(w)$ it is straightforward to check that there is a sequence p_i, $1 \leqslant i \leqslant k$ such that the conditions are satisfied. Conversely, assume that $p_k \in T_2(w)$ for all $1 \leqslant k \leqslant i$ and $\text{label}(v) = \text{label}(w)$. We construct a root-preserving embedding *f* of $T_1(v)$ into $T_2(w)$ as follows. Let $f(v) = w$. By definition of ρ there must be a root-preserving embedding f^k, $1 \leqslant k \leqslant i$, of $T_1(v_k)$ in $T_2(p_k)$. For a node u in $T_1(v_k)$, $1 \leqslant k \leqslant i$, we set $f(u) = f^k(u)$. Since $p_k \in \text{rr}(p_{k-1})$, $2 \leqslant k \leqslant i$, and $p_k \in T_2(w)$ for all *k*, $1 \leqslant k \leqslant i$, it follows that *f* is indeed a root-preserving embedding. □

Using dynamic programming it is now straightforward to compute $\rho(v, w)$ for all $v \in V(T_1)$ and $w \in V(T_2)$. For a fixed node v we traverse T_2 in reverse postorder. At each node $w \in V(T_2)$ we check if there is a root-preserving embedding of $T_1(v)$ in $T_2(w)$. If so we set $\rho(v, q) = w$, for all $q \in lr(w)$ such that $x \preccurlyeq q$, where *x* is the next root-preserving embedding of $T_1(v)$ in $T_2(w)$.

For a pair of nodes $v \in V(T_1)$ and $w \in V(T_2)$ we test for a root-preserving embedding using Lemma 10. Assuming that values for smaller subproblems has been computed, the time used is $\mathrm{O}(\deg(v))$. Hence, the contribution to the total time for the node w is $\sum_{v \in V(T_1)} \mathrm{O}(\deg(v)) = \mathrm{O}(|T_1|)$. It follows that the time complexity of the algorithm is bounded by $\mathrm{O}(|T_1||T_2|)$. Clearly, only $\mathrm{O}(|T_1||T_2|)$ space is needed to store ρ. Hence, we have the following theorem,

Theorem 4 (*Kilpeläinen and Mannila [22]*). *For any pair of rooted, labeled, and ordered trees T_1 and T_2, the tree inclusion problem can be solved in $\mathrm{O}(|T_1||T_2|)$ time and space.*

5.3. Unordered tree inclusion

In [22] it is shown that the unordered tree inclusion problem is NP-complete. The reduction used is from the Satisfiability problem [12]. Independently, Matoušek and Thomas [32] gave another proof of NP-completeness.

An algorithm for the unordered tree inclusion problem is presented in [22] using $\mathrm{O}(|T_1| I_1 2^{2 I_1} |T_2|)$ time. Hence, if I_1 is constant the algorithm runs in $\mathrm{O}(|T_1||T_2|)$ time and if $I_1 = \log |T_2|$ the algorithm runs in $\mathrm{O}(|T_1| \log |T_2| |T_2|^3)$.

6. Conclusion

We have surveyed the tree edit distance, alignment distance, and inclusion problems. Furthermore, we have presented, in our opinion, the central algorithms for each of the problems. There are several open problems, which may be the topic of further research. We conclude this paper with a short list proposing some directions.

- For the unordered versions of the above problems some are NP-complete while others are not. Characterizing exactly which types of mappings that gives NP-complete problems for unordered versions would certainly improve the understanding of all of the above problems.
- The currently best worst-case upper bound on the ordered tree edit distance problem is the algorithm of [25] using $\mathrm{O}(|T_1|^2|T_2| \log |T_2|)$. Conversely, the quadratic lower bound for

Table 1
Results for the tree edit distance, alignment distance, and inclusion problem listed according to variant

Variant	Type	Time	Space	Reference
Tree edit distance				
General	O	$O(\|T_1\|\|T_2\|D_1^2D_2^2)$	$O(\|T_1\|\|T_2\|D_1^2D_2^2)$	[43]
General	O	$O(\|T_1\|\|T_2\|\min(L_1, D_1)\min(L_2, D_2))$	$O(\|T_1\|\|T_2\|)$	[55]
General	O	$O(\|T_1\|^2\|T_2\|\log\|T_2\|)$	$O(\|T_1\|\|T_2\|)$	[25]
General	O	$O(\|T_1\|\|T_2\| + L_1^2\|T_2\| + L_1^{2.5}L_2)$	$O((\|T_1\| + L_1^2)\min(L_2, D_2) + \|T_2\|)$	[8]
General	U	MAX SNP-hard		[54]
Constrained	O	$O(\|T_1\|\|T_2\|)$	$O(\|T_1\|\|T_2\|)$	[51]
Constrained	O	$O(\|T_1\|\|T_2\|I_1I_2)$	$O(\|T_1\|\|D_2I_2)$	[37]
Constrained	U	$O(\|T_1\|\|T_2\|(I_1 + I_2)\log(I_1 + I_2))$	$O(\|T_1\|\|T_2\|)$	[52]
Less-constrained	O	$O(\|T_1\|\|T_2\|I_1^3I_2^3(I_1 + I_2))$	$O(\|T_1\|\|T_2\|I_1^3I_2^3(I_1 + I_2))$	[29]
Less-constrained	U	MAX SNP-hard		[29]
Unit-cost	O	$O(u^2\min(\|T_1\|, \|T_2\|)\min(L_1, L_2))$	$O(\|T_1\|\|T_2\|)$	[41]
1-degree	O	$O(\|T_1\|\|T_2\|)$	$O(\|T_1\|\|T_2\|)$	[38]
Tree alignment distance				
General	O	$O(\|T_1\|\|T_2\|(I_1 + I_2)^2)$	$O(\|T_1\|\|T_2\|(I_1 + I_2))$	[18]
General	U	MAX SNP-hard		[18]
Similar	O	$O((\|T_1\| + \|T_2\|)\log(\|T_1\| + \|T_2\|)(I_1 + I_2)^4s^2)$	$O((\|T_1\| + \|T_2\|)\log(\|T_1\| + \|T_2\|)(I_1 + I_2)^4s^2)$	[17]
Tree inclusion				
General	O	$O(\|T_1\|\|T_2\|)$	$O(\|T_1\|\min(D_2L_2))$	[21]
General	O	$O(\|\Sigma_{T_1}\|\|T_2\| + m_{T_1,T_2}D_2)$	$O(\|\Sigma_{T_1}\|\|T_2\| + m_{T_1,T_2})$	[36]
General	O	$O(L_1\|T_2\|)$	$O(L_1\min(D_2L_2))$	[7]
General	U	NP-hard		[22,32]

D_i, L_i, and I_i denotes the depth, the number of leaves, and the maximum degree, respectively, of T_i, $i = 1, 2$. The type is either O for ordered or U for unordered. The value u is the unit cost edit distance between T_1 and T_2 and the value s is the number of spaces in the optimal alignment of T_1 and T_2. The value Σ_{T_1} is set of labels used in T_1 and m_{T_1,T_2} is the number of pairs of nodes in T_1 and T_2 which have the same label.

the longest common subsequence problem [1] problem is the best general lower bound for the ordered tree edit distance problem. Hence, a large gap in complexity exists which needs to be closed.

- Several meaningful edit operations other than the above may be considered depending on the particular application. Each set of operations yield a new edit distance problem for which we can determine the complexity. Some extensions of the tree edit distance problem have been considered [6,5,24].

Acknowledgements

Thanks to Inge Li Gørtz and Anna Östlin for proof reading and helpful discussions.

References

[1] A.V. Aho, J.D. Ullman, D.S. Aho, J.D. Hirschberg, Bounds on the complexity of the longest common subsequence problem, J. ACM 1 (23) (1976) 1–12.

[2] T. Akutsu, M.M. Halldórsson. On the approximation of largest common point sets and largest common subtrees, in: Proc. 5th Ann. Internat. Symp. on Algorithms and Computation, Lecture Notes in Computer Science (LNCS), Vol. 834. Springer, Berlin, 1994, pp. 405–413.

[3] L. Alonso, R. Schott, 1993. On the tree inclusion problem, in: Proc. Mathematical Foundations of Computer Science, 1993, pp. 211–221.

[4] A. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy, Proof verification and hardness of approximation problems, in: Proc. 33rd IEEE Symp. on the Foundations of Computer Science (FOCS), 1992, pp. 14–23.

[5] S. Chawathe, H. Garcia-Molina, Meaningful change detection in structured data, in: Proc. ACM SIGMOD International Conference on Management of Data, Tuscon, Arizona, 1997, pp. 26–37.

[6] S.S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, Change detection in hierarchically structured information, in: Proc. ACM SIGMOD Internat. Conf. on Management of Data, Montréal, Québec, June 1996, pp. 493–504.

[7] W. Chen, More efficient algorithm for ordered tree inclusion, J. Algorithms 26 (1998) 370–385.

[8] W. Chen, New algorithm for ordered tree-to-tree correction problem, J. Algorithms 40 (2001) 135–158.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, Cambridge, MA, 2001.

[10] M. Dubiner, Z. Galil, E. Magen, Faster tree pattern matching, in: Proc. 31st IEEE Symp. on the Foundations of Computer Science (FOCS), 1990, pp. 145–150.

[11] M. Farach, M. Thorup, Fast comparison of evolutionary trees, in: Proc. 5th Ann. ACM-SIAM Symp. on Discrete Algorithms, 1994, pp. 481–488.

[12] M.J. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, New York, 1979.

[13] A. Gupta, N. Nishimura, Finding largest subtrees and smallest supertrees, Algorithmica 21 (1998) 183–210.

[14] D. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, Cambridge, 1997.

[15] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.

[16] C.M. Hoffmann, M.J. O'Donnell, Pattern matching in trees, J. ACM 29 (1) (1982) 68–95.

[17] J. Jansson, A. Lingas, A fast algorithm for optimal alignment between similar ordered trees, in: Proc. 12th Ann. Symp. Combinatorial Pattern Matching (CPM), Lecture Notes of Computer Science (LNCS). Vol. 2089, Springer, Berlin, 2001.

[18] T. Jiang, L. Wang, K. Zhang, Alignment of trees—an alternative to tree edit, Theoret. Comput. Sci. 143 (1995).

[19] D. Keselman, A. Amir, Maximum agreement subtree in a set of evolutionary trees—metrics and efficient algorithms, in: Proc. 35th Ann. Symp. on Foundations of Computer Science (FOCS), 1994, pp. 758–769.

[20] S. Khanna, R. Motwani, F.F. Yao, Approximation algorithms for the largest common subtree problem, Technical Report, Stanford University, 1995.

[21] P. Kilpeläinen, Tree matching problems with applications to structured text databases, Ph.D. Thesis, Department of Computer Science, University of Helsinki, November 1992.

[22] P. Kilpeläinen, H. Mannila, Ordered and unordered tree inclusion, SIAM J. Comput. 24 (1995) 340–356.

[23] P. Klein, 2002. Personal communication.

[24] P. Klein, S. Tirthapura, D. Sharvit, B. Kimia, A tree-edit-distance algorithm for comparing simple, closed shapes, in: Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA), 2000, pp. 696–704.

[25] P.N. Klein, Computing the edit-distance between unrooted ordered trees, in: Proc. 6th Ann. European Symp. on Algorithms (ESA), Springer, Berlin, 1998, pp. 91–102.

[26] D.E. Knuth, The Art of Computer Programming, Vol. 1, Addison-Wesley, Reading, MA, 1969.

[27] S.R. Kosaraju, Efficient tree pattern matching, in: Proc. 30th IEEE Symp. on the Foundations of Computer Science (FOCS), 1989, pp. 178–183.

[28] G.M. Landau, U. Vishkin, Fast parallel and serial approximate string matching, J. Algorithms 10 (1989) 157–169.

[29] S.Y. Lu, A tree-to-tree distance and its application to cluster analysis, IEEE Trans. Pattern Anal. Mach. Intell. 1 (1979) 219–224.

[30] S.Y. Lu, A tree-matching algorithm based on node splitting and merging, IEEE Trans. Pattern Anal. Mach. Intell. 6 (2) (1984) 249–256.

[31] C.L. Lu, Z.-Y. Su, C.Y., Tang, A new measure of edit distance between labeled trees, in: Proc. 7th Ann. Internat. Conf. on Computing and Combinatorics (COCOON), Lecture Notes in Computer Science (LNCS). Vol. 2108, Springer, Berlin, 2001.

[32] J. Matoušek, R. Thomas, On the complexity of finding iso- and other morphisms for partial k-trees, Discrete Math. 108 (1992) 343–364.

[33] R. Motwani, Lecture Notes on Approximation Algorithms, Vol. 1. Technical Report STAN-CS-92-1435, Department of Computer Science, Stanford University, 1992.

[34] N. Nishimura, P. Ragde, D.M. Thilikos, Finding smallest supertrees under minor containment, Internat. J. Found. Comput. Sci. 11 (3) (2000) 445–465.

[35] R. Ramesh, I.V. Ramakrishnan, Nonlinear pattern matching in trees, J. ACM 39 (2) (1992) 295–316.

[36] T. Richter, A new algorithm for the ordered tree inclusion problem, in: Proc. 8th Ann. Symp. on Combinatorial Pattern Matching (CPM), Lecture Notes of Computer Science (LNCS), Vol. 1264, Springer, Berlin, 1997, pp. 150–166.

[37] T. Richter, A new measure of the distance between ordered trees and its applications, Technical Report 85166-cs. Department of Computer Science, University of Bonn, 1997.

[38] S.M. Selkow, The tree-to-tree editing problem, Inform. Process. Lett. 6 (6) (1977) 184–186.

[39] J. Setubal, J. Meidanis, Introduction to Computational Biology, PWS Publishing Company, MA, 1997.

[40] D. Shasha, J.T.-L. Wang, H. Shan, K. Zhang, Atreegrep: approximate searching in unordered trees, in: Proc. 14th Internat. Conf. on Scientific and Statistical Database Management, 2002, pp. 89–98.

[41] D. Shasha, K. Zhang, Fast algorithms for the unit cost editing distance between trees, J. Algorithms 11 (1990) 581–621.

[42] D. Shasha, K. Zhang, Approximate tree pattern matching, in: Pattern Matching in String, Trees and Arrays, Oxford University, Oxford, 1997, pp. 341–371.

[43] K.-C. Tai, The tree-to-tree correction problem, J. ACM 26 (1979) 422–433.

[44] E. Tanaka, A note on a tree-to-tree editing problem, Internat. J. Pattern Recogn. Artif. Intell. 9 (1) (1995) 167–172.

[45] E. Tanaka, K. Tanaka, The tree-to-tree editing problem, Internat. J. Pattern Recogn. Artif. Intell. 2 (2) (1988) 221–240.

[46] S. Tirthapura, D. Sharvit, P. Klein, B.B. Kimia, Indexing based on edit-distance matching of shape graphs, in: Proc. SPIE Internat. Symp. Voice, Video and Data Communications, 1998, pp. 91–102.

[47] E. Ukkonen, Finding approximate patterns in strings, J. Algorithms 6 (1985) 132–137.

[48] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. ACM 21 (1974) 168–173.

[49] J.T.-L. Wang, K. Zhang, K. Jeong, D. Shasha, A system for approximate tree matching, IEEE Trans. Knowl. Data Eng. 6 (4) (1994) 559–571.

[50] K. Zhang, The Editing Distance Between Trees: Algorithms and Applications, Ph.D. Thesis, Department of Computer Science, Courant Institute, 1989.

[51] K. Zhang, Algorithms for the constrained editing problem between ordered labeled trees and related problems, Pattern Recognition 28 (1995) 463–474.

[52] K. Zhang, A constrained edit distance between unordered labeled trees, Algorithmica 15 (3) (1996) 205–222.

[53] K. Zhang, Efficient parallel algorithms for tree editing problems, in: Proc. 7th Ann. Symp. Combinatorial Pattern Matching (CPM), Lecture Notes in Computer Science, Vol. 1075, 1996, pp. 361–372.

[54] K. Zhang, T. Jiang, Some MAX SNP-hard results concerning unordered labeled trees, Inform. Process. Lett. 49 (1994) 249–254.

[55] K. Zhang, D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, SIAM J. Comput. 18 (1989) 1245–1262.

[56] K. Zhang, D. Shasha, J.T.L. Wang, Approximate tree matching in the presence of variable length don't cares, J. Algorithms 16 (1) (1994) 33–66.

[57] K. Zhang, R. Statman, D. Shasha, On the editing distance between unordered labeled trees, Technical Report 289, Department of Computer Science, The University of Western Ontario, 1991.

[58] K. Zhang, R. Statman, D. Shasha, On the editing distance between unordered labeled trees, Inform. Process. Lett. 42 (1992) 133–139.