A New Measure of the Distance between Ordered Trees and its Applications*

Thorsten Richter
Department of Computer Science IV, University of Bonn
Roemerstr. 164, 53117 Bonn, Germany
e–mail: richter@cs.uni-bonn.de

May 31, 1997

Abstract

The problem of computing the distance between two trees T_1 and T_2 ([14], [18]), also known as *tree editing problem*, is a generalization of the problem of computing the distance between two strings ([8], [9], [16]) to labeled ordered trees. One view of the distance between two trees T_1 and T_2 is that of a *mapping*. A mapping M from T_1 to T_2 is a partial one-to-one map from the nodes of T_1 to the nodes of T_2 that preserves the ancestor relations and the left-to-right ordering of the nodes. In [18] an algorithm is given that computes the distance between two ordered trees T_1 and T_2 in time $O(|T_1| \cdot |T_2| \cdot \min(\textsc{depth}(T_1), \textsc{leaves}(T_1)) \cdot \min(\textsc{depth}(T_2), \textsc{leaves}(T_2)))$ and in space $O(|T_1| \cdot |T_2|)$. In this paper we define a new measure of the distance between two ordered trees T_1 and T_2 that is based on a restricted kind of mapping which we call *structure respecting*. In a structure respecting mapping M from T_1 to T_2 for all (v_1, w_1), (v_2, w_2), $(v_3, w_3) \in M$ the additional condition $\textsc{lca}(v_1, v_2) = \textsc{lca}(v_1, v_3) \iff \textsc{lca}(w_1, w_2) = \textsc{lca}(w_1, w_3)$ holds, so that a structure respecting mapping preserves more of the structure of the trees involved than a general mapping. We then present a simple dynamic programming algorithm that computes a minimum cost structure respecting mapping between two ordered trees T_1 and T_2 in time $O(\textsc{degree}(T_1) \cdot \textsc{degree}(T_2) \cdot |T_1| \cdot |T_2|)$ and in space $O(\textsc{degree}(T_1) \cdot \textsc{depth}(T_1) \cdot |T_2|)$.

1 Introduction and Motivation

The problem of computing the distance between two trees T_1 and T_2 ([14], [18]), also known as *tree editing problem*, is a generalization of the problem of computing the distance between two strings ([8], [9], [16]) to labeled trees. The trees under consideration in the tree editing problem are ordered, i. e. the left-to right order of the children of a node is significant. Without this assumption, i. e. if considering unordered trees, the tree editing problem becomes $\mathcal{NP}$-complete [19].

The edit operations available in the tree editing problem are changing, deleting and inserting a node. To these operations costs are assigned that depend on the labels of the

*This work was supported by the DFG under grant Bl 320/2 - 1

nodes involved. The problem is to find a sequence of such operations transforming a tree T_1 into a tree T_2 with minimum cost. The distance between the trees T_1 and T_2 is then defined to be the cost of such a sequence.

Another view of the distance between two trees T_1 and T_2 is that of a *mapping*. A mapping M from T_1 to T_2 is a partial one-to-one map from the nodes of T_1 to the nodes of T_2 that preserves the ancestor relations and the left-to-right ordering of the nodes. The pairs of inverse image and image of a mapping are called connections. Interpreting a connection as a change operation, a node of T_1 not contained in the domain of M as a delete operation, and a node of T_2 not contained in the range of M as an insert operation, one can define the cost of a mapping to be the sum of the costs of these edit operations. The cost of a mapping with minimum cost is equal to the cost of an edit sequence with minimum cost, so that both views of the distance between two trees are equivalent.

In [18] an algorithm is given that computes the distance between two ordered trees T_1 and T_2 in time $O(|T_1| \cdot |T_2| \cdot \min(\text{DEPTH}(T_1), \text{LEAVES}(T_1)) \cdot \min(\text{DEPTH}(T_2), \text{LEAVES}(T_2)))$ and in space $O(|T_1| \cdot |T_2|)$.

An application of the tree editing problem is the comparison of RNA secondary structures (see [11], [12]). Here an RNA secondary structure is modelled as an ordered tree, such that one can use an algorithm for the tree editing problem to compare two of these structures.

Another application of ordered labeled trees occurs in syntax theory. To compare two sentential forms of a (context-free) grammar, one can compute the distance between their parse trees. In [13] this was applied to the problem of syntactic error recovery and correction for programming languages, for example. Similar techniques could be applied in the analysis of natural language. One can describe the syntactic structure of a natural language sentence in terms of its parse tree (see [17]). Then one can compare a new sentence with other sentences stored in a database by computing the distance between their parse trees, for example.

In this paper we define a new measure of the distance between two ordered trees T_1 and T_2 that is based on a restricted kind of mapping which we call *structure respecting*. In a structure respecting mapping M from T_1 to T_2 for all (v_1, w_1), (v_2, w_2), $(v_3, w_3) \in M$ the additional condition

$$\text{LCA}(v_1, v_2) = \text{LCA}(v_1, v_3) \iff \text{LCA}(w_1, w_2) = \text{LCA}(w_1, w_3)$$

holds, so that a structure respecting mapping preserves more of the structure of the trees involved than a general mapping. This can be interesting when one compares the syntactic structure of (natural language) sentences, for example. The restriction to structure respecting mappings could give more significant results in this case.

For every structure respecting mapping there is still a corresponding edit sequence with the same cost, but on the other hand there is no longer a corresponding structure respecting mapping for every edit sequence. We present a dynamic programming algorithm that computes a minimum cost structure respecting mapping between two ordered trees T_1 and T_2 in time $O(\text{DEGREE}(T_1) \cdot \text{DEGREE}(T_2) \cdot |T_1| \cdot |T_2|)$ and in space $O(\text{DEGREE}(T_1) \cdot \text{DEPTH}(T_1) \cdot |T_2|)$.

This paper is organized as follows. In Section 2 we review the general tree editing problem and introduce the main concepts used within. In Section 3 our new measure of the distance between trees is introduced and its main properties are shown. Section 4 is devoted to the computation of this new distance and Section 5 presents an algorithm

in pseudo-code fashion, that computes it. In Section 6 we show the correctness of the algorithm and consider its complexity. In Section 7 we apply our new concept of the distance between two ordered trees to approximate tree matching. Finally in Section 8 we briefly discuss our results.

2 The General Tree Editing Problem

The trees we consider in this paper are always ordered and labeled. The label of a node v is denoted by $\text{LABEL}(v)$. We assume the labels of the nodes to be chosen from a finite alphabet Σ. The root of tree T is denoted by $\text{ROOT}(T)$. Furthermore $T[v]$ is the subtree of T with root v and if v has k children $v_1, v_2, \ldots, v_k$, then $F[v_{i_1} \ldots v_{i_2}]$, $1 \leq i_1 \leq i_2 \leq k$, denotes the subforest of $T[v]$ which consists of the subtrees $T[v_{i_1}]$ to $T[v_{i_2}]$. We sometimes use $F[v]$ as an abbreviation of $F[v_1 \ldots v_k]$, i. e. of the subforest $T[v] \setminus \{v\}$.

In this section we review the general tree editing problem and introduce the main concepts - edit operations, edit sequences and mappings - used within.

2.1 Edit Operations and Edit Sequences

A first view of the general tree editing problem is based on the concept of a set of edit operations on trees to which costs are assigned. Applying such an edit operation manipulates a tree locally. If two trees T and T' are given and their distance is to be computed, the tree T is transformed to the tree T' step-by-step by applying an appropriate sequence of edit operations. The distance between T and T' is then defined to be the cost of an edit sequence with minimum cost which transforms T to T'. The details are as follows.

Definition 1 (Edit operations) *Let $T = (V, E)$ be a labeled ordered tree. Then we consider the following* edit operations *on T.*

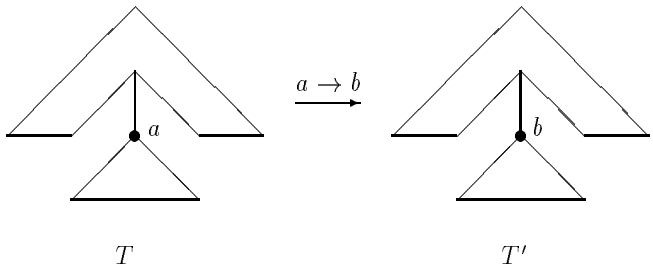

Figure 1: *A change operation changing the label of a node from a to b.*

- Change. *Changing the label of a node from $a \in \Sigma$ to $b \in \Sigma$ (this includes the case $a = b$). Such a change operation is denoted by $a \to b$ (this notation is potentially ambiguous, since many nodes can have the same label. However which node is meant will always be clear from the context). Figure 1 illustrates a change operation.*
- Delete. *Deleting a node labeled by $a \in \Sigma$. All children of the deleted node become children of its parent, i. e. the deleted node is substituted by the sequence of its*

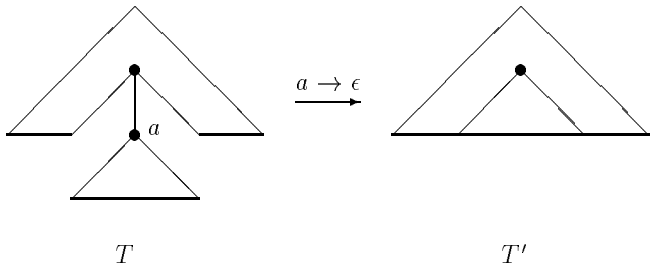

Figure 2: *A delete operation deleting a node with label a.*

children. Such a delete operation is denoted by $a \to \varepsilon$ *(*$\varepsilon \notin \Sigma$ *denotes the empty word). Figure 2 illustrates a delete operation.*

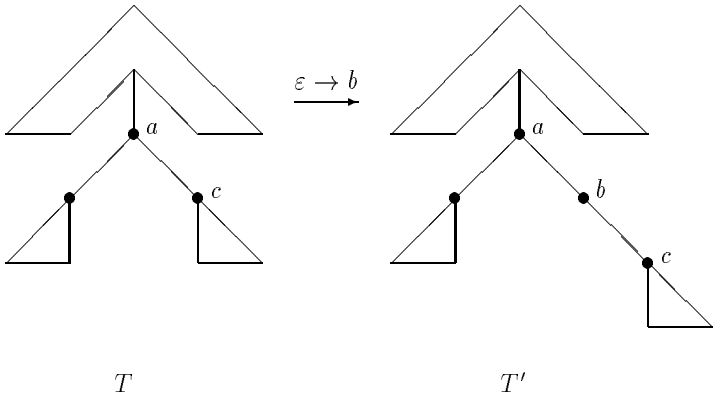

Figure 3: *An insert operation inserting a node with label b.*

- Insert. *Inserting a node labeled by* $b \in \Sigma$ *as the child of a node with label* $a \in \Sigma$*. A consecutive sequence of the children of the latter become the children of the inserted node. Which sequence this is depends on the context. An insert operation is denoted by* $\varepsilon \to b$*. Figure 3 illustrates an insert operation that inserts a node with label b as a child of the node with label a. Here the right child of the latter becomes the child of the inserted node.*

The set of all such edit operations is denoted by OP, formally

$$OP \;=\; \{a \to b \mid a, b \in \Sigma \cup \{\varepsilon\}\} \setminus \{\varepsilon \to \varepsilon\}.$$

If T' *is the tree which results from the tree* T *by applying the edit operation* $op \in OP$*, we say that op* transforms *the tree* T *to the tree* T'*, which is represented by* $T \overset{op}{\to} T'$*.* □

Definition 2 (Edit sequences) *A sequence* $S = (op_1, op_2, \ldots, op_k)$*,* $k \in \mathcal{N}_0$*, of edit operations* $op_i \in OP$*,* $1 \le i \le k$*, is called an* edit sequence*, if there are trees* T_i*,* $0 \le i \le k$*,*

such that the operation op_i transforms the tree T_{i-1} to the tree T_i, $1 \leq i \leq k$.

Then we say that the edit sequence S transforms *the tree $T = T_0$ to the tree $T_k = T'$, which is represented by $T \stackrel{S}{\Rightarrow} T'$.* □

Definition 3 (Costs of edit operations and edit sequences) *We assign to each edit operation $op \in OP$ a* cost *$\gamma(op)$, such that*

(a) $\gamma(op) \geq 0 \ \ \forall\ op \in OP$,

(b) $\gamma(a \to a) = 0 \ \ \forall\, a \in \Sigma$,
$\gamma(a \to b) > 0 \ \ \forall\, a, b \in \Sigma \cup \{\varepsilon\},\ a \neq b$,

(c) $\gamma(a \to b) = \gamma(b \to a) \ \ \forall\, a, b \in \Sigma \cup \{\varepsilon\}$,

(d) $\gamma(a \to c) \leq \gamma(a \to b) + \gamma(b \to c) \ \ \forall\, a, b, c \in \Sigma \cup \{\varepsilon\}$.

Hence the cost function γ is a metric.

The cost *$\gamma(S)$ of an edit sequence S is then defined to be the sum of the costs of the edit operations of S, formally*

$$\gamma(S) \ = \ \sum_{i=1}^{|S|} \gamma(op_i).$$

□

2.2 Mappings between Trees

Another view of the general tree editing problem is based on partial one-to-one maps between trees which are called mappings. To each mapping M a cost is assigned which depends on the nodes of the trees "covered" by M. The distance between the two trees is then defined to be the cost of a mapping between them with minimum cost. The details are as follows.

Definition 4 (Mapping) *Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two labeled ordered trees. A* mapping *M from T_1 to T_2 is a set*

$$M \ \subseteq \ V_1 \times V_2,$$

such that for all $(v_1, w_1), (v_2, w_2) \in M$ the following conditions are satisfied:

(1) $v_1 = v_2 \iff w_1 = w_2$,

(2) *v_1 is an ancestor of v_2* $\iff$ *w_1 is an ancestor of w_2,*

(3) *v_1 is to the left of v_2* $\iff$ *w_1 is to the left of w_2.*

A pair $(v, w) \in M$ is called a connection *between v and w. Then we say that v is* connected to *w, and vice versa.* □

Definition 5 (Not feasible and expandable) *Any set $N \subseteq V_1 \times V_2$ that violates one of the conditions of a mapping is called* not feasible. *A mapping M from T to T' is called* not expandable *if the addition of an arbitrary connection to M causes M to be not feasible, formally if*

$$\forall (v, w) \in (V_1 \times V_2 \setminus M): \quad M \cup \{(v, w)\} \text{ is not feasible};$$

and called expandable, *otherwise.* □

Condition (1) of a mapping says that a mapping can be considered as a partial one-to-one map from the nodes of T_1 to the nodes of T_2. Therefore we can use the following notation for the domain and the range of a mapping M

$$\begin{aligned} D_M(T_1) &:= \{v \in V_1 \mid \exists w \in V_2 : (v,w) \in M\} \subseteq V_1, \\ R_M(T_2) &:= \{w \in V_2 \mid \exists v \in V_1 : (v,w) \in M\} \subseteq V_2. \end{aligned}$$

If $v \in D_M(T_1)$ and $w \in R_M(T_2)$, respectively, we say that v and w, respectively, are *covered* by the mapping M. Otherwise v and w, respectively, are *not covered* by M.

A *substructure* of a tree is obtained by deleting none or more nodes of the tree (in the sense of a delete operation defined above). A substructure of a tree is in general a forest in which the ancestor relations as well as the left-to-right ordering of the nodes are preserved.

Condition (2) of a mapping ensures that the connections of a mapping M preserve the ancestor relations of T_1 and T_2. Condition (3) ensures that the left-to-right ordering of the nodes is preserved. Therefore a mapping M from T_1 to T_2 induces a substructure $S_M(T_1)$ of T_1 in the following manner. The nodes of the substructure are the nodes of T_1 that are covered by M, i. e. the nodes in the domain $D_M(T_1)$ of M. For two nodes $v_1, v_2 \in D_M(T_1)$ the node v_1 is the parent v_2 in the substructure, iff v_1 is ancestor of v_2 in T_1 and there is no node u on the path from v_1 to v_2 in T_1 that is covered by M. This means the parent of a node in the substructure is its lowest ancestor in T_1 that is also contained in the substructure. Finally for two edges (v, v_1) (v, v_2) of the substructure the node v_1 is to the left of the node v_2 in the substructure, iff it is to the left of v_2 in T_1. This means that the order of the substructure reflects the order of the tree.

Analogously a mapping M also induces a substructure $S_M(T_2)$ of T_2, the nodes of which are the nodes in the range $R_M(T_2)$ of M. By the definition of a mapping we have that the two substructures $S_M(T_1)$ and $S_M(T_2)$ of T_1 and T_2, respectively, are isomorphic. They only differin the labels of the nodes. Hence we need not distinguish between the induced substructure of T_1 and that of T_2 in the following. Instead we can speak of a single substructure induced by M which is denoted by $S_M(T_1, T_2)$. It represents the parts of the trees T_1 and T_2 which are "similar" with respect to the mapping M.

Formally we can define the substructure induced by a mapping as follows.

Definition 6 (Substructure induced by a mapping) *Let M be a mapping from T_1 to T_2. Then the* substructure of T_1 and T_2 induced by a mapping M *is the forest $S_M(T_1, T_2) = (V_M, E_M)$ where*

(1) $V_M = D_M(T_1)$,

(2) $(v_1, v_2) \in E_M \iff \left[\begin{array}{c} v_1, v_2 \in V_M \ \wedge\ v_1 \textit{ is ancestor of } v_2 \textit{ in } T_1 \wedge \\ \nexists u \in V_M : u \textit{ is on the path from } v_1 \textit{ to } v_2 \textit{ in } T_1 \end{array} \right]$,

(3) $\forall (v, v_1), (v, v_2) \in E_M : \left[\begin{array}{c} v_1 \textit{ is in } F_1(M) \textit{ to the left of } v_2 \iff \\ v_1 \textit{ is in } T_1 \textit{ to the left of } v_2 \end{array} \right]$.

Note that we could have defined the substructure in terms of T_2 as well. □

Now we define the cost of a mapping, so that we are able to search for a mapping with minimal cost.

Definition 7 (Cost of a mapping) *The* cost $\gamma(M)$ *of a mapping* M *from* T_1 *to* T_2 *is defined as follows*

$$\begin{aligned}\gamma(M) \quad = \quad & \sum_{(v,w)\in M} \gamma(\text{LABEL}(v) \to \text{LABEL}(w)) \\ & + \sum_{v\in V_1 \setminus D_M(T_1)} \gamma(\text{LABEL}(v) \to \varepsilon) \quad + \sum_{w\in V_2 \setminus R_M(T_2)} \gamma(\varepsilon \to \text{LABEL}(w)).\end{aligned}$$

□

Note that because of condition (d) in Definition 3 an expandable mapping M does not need to be considered when seeking for a minimal cost mapping, since there is a corresponding not expandable mapping M', whose cost is at most as high as the cost of M. The cost of M is in general lower than the cost of M'. They are equal only if, with respect to the labels of the nodes of the expanding connection (v, w) in condition (d) of Definition 3, equality holds. Then the delete operation and the insert operation are together as expensive as the corresponding change operation. In the following we interpret such a pair of delete and insert operation as a change operation. Then we can say that an expandable mapping cannot have minimal cost.

2.3 The General Distance between Trees

The following lemma says that both views of the distance between two trees are equivalent in the sense that they define the same distance.

Lemma 1 ([18]) *Given an edit sequence* S *that transforms the tree* T *to the tree* T'*, there exists a mapping* M *from* T *to* T' *such that* $\gamma(M) \leq \gamma(S)$*.*

Conversely, for any mapping M *from* T *to* T'*, there exists an edit sequence* S *transforming* T *to* T' *such that* $\gamma(S) = \gamma(M)$*.* □

The general distance between two trees is defined as follows.

Definition 8 (The general distance between two trees) *The* general distance $\delta(T, T')$ *between two trees* T *and* T' *is the cost of a edit sequence with minimum cost that transforms* T *to* T'*, formally*

$$\delta(T, T') \quad = \quad \min\{\gamma(S) \mid T \overset{S}{\Rightarrow} T'\}.$$

□

Note that due to Lemma 1 we could have defined the general distance between two trees as well in terms of mappings:

$$\delta(T, T') \quad = \quad \min\{\gamma(M) \mid M \text{ is a mapping from } T \text{ to } T'\}.$$

The first algorithm for computing the general distance between two trees was given by Tai in [14]. It considers the nodes of both tree in preorder and has a running time of

$$O(|T| \cdot |T'| \cdot \text{DEPTH}(T)^2 \cdot \text{DEPTH}(T')^2).$$

This complexity was improved by Zhang and Shasha in [18]. The basic idea of their algorithm is to consider the nodes of the trees in (left-to-right) postorder. This is as follows. Let $T = (\{v_1, \ldots, v_n\}, E)$ and $T' = (\{w_1, \ldots, w_m\}, E')$ be the given trees, where the subscripts of the nodes correspond to their left-to-right postorder numbers. Then the algorithm iteratively computes the distances $\textsc{treedist}(v_i, w_j)$ between the subtrees $T[v_i]$ and $T'[w_j]$ of T and T', respectively, for all $1 \leq i \leq n, 1 \leq j \leq m$ in increasing order.

Let $\textsc{Left}(T)$ denote the leftmost leaf of a tree T, i. e. the node of T with the smallest postorder number, and let $\textsc{forestdist}(v_{i_1} \ldots v_{i_2}, w_{j_1} \ldots w_{j_2})$ denote the distance between the ordered forests that consist of the nodes v_{i_1} to v_{i_2} and w_{j_1} to w_{j_2}, respectively. Then a subtree distance $\textsc{treedist}(v_i, w_j)$ is computed by computing the distances $\textsc{forestdist}(\textsc{Left}(T[v_i]) \ldots v_{i_1}, \textsc{Left}(T'[w_j] \ldots w_{j_1})$ iteratively for increasing $i_1 \leq i$ and $j_1 \leq j$. Since the computation of a single forest distance can be done in constant time, the computation of $\textsc{treedist}(v_i, w_j)$ altogether takes time $O(|T[v_i]| \cdot |T'[w_j]|)$.

The crucial observation of Zhang and Shasha was that they do not have to compute the subtree distance $\textsc{treedist}(v_i, w_j)$ for every pair (v_i, w_j) separately, but that some distances can be obtained as a byproduct of the computation of other distances. Let the set *LR_keyroots* of a tree T be defined as follows:

$$LR\_keyroots(T) = \{v_k \mid \text{there is no } k' > k \text{ such that } \textsc{Left}(T[v_k]) = \textsc{Left}(T[v_{k'}])\}.$$

That is, if v_k is in $LR\_keyroots(T)$ then either v_k is the root of T or v_k has a left sibling. Zhang and Shasha show that a subtree distance $\textsc{treedist}(v_i, w_j)$ has to be computed separately only if $v_i \in LR\_keyroots(T)$ and $w_j \in LR\_keyroots(T')$. So they obtain an overall time complexity of

$$O(\sum_{v \in \text{LR\_keyroots}(T)} \sum_{w \in \text{LR\_keyroots}(T')} |T[v]| \cdot |T'[w]|).$$

By a clever analysis of this formula they get the following result.

Theorem 1 ([18]) *The general distance $\delta(T, T')$ between two labeled ordered trees T and T' can be computed in time*

$$O(|T| \cdot |T'| \cdot \min(\textsc{depth}(T), \textsc{leaves}(T)) \cdot \min(\textsc{depth}(T'), \textsc{leaves}(T')))$$

and in space

$$O(|T| \cdot |T'|).$$

□

Within the same time and space bound a mapping that yields the distance computed can be constructed.

Zhang and Shasha also give a parallel implementation of their algorithm. It has a time complexity of $O(|T| + |T'|)$ using $O(\min(|T|, |T'|) \cdot \textsc{leaves}(T) \cdot \textsc{leaves}(T'))$ processors.

3 The Structure Respecting Distance

Now we introduce a new measure of the distance between two trees that is based on a restricted kind of mapping.

Definition 9 (Structure respecting mapping) *Let T_1 and T_2 be two labeled ordered trees. Then a mapping M from T_1 to T_2 is called* structure respecting, *if for all (v_1, w_1), (v_2, w_2), $(v_3, w_3) \in M$, such that none of the nodes v_1, v_2 and v_3 is an ancestor of one of the others, the additional condition*

$$\text{LCA}_{T_1}(v_1, v_2) = \text{LCA}_{T_1}(v_1, v_3) \iff \text{LCA}_{T_2}(w_1, w_2) = \text{LCA}_{T_2}(w_1, w_3)$$

is satisfied, where LCA *denotes the lowest common ancestor.* □

The terms *feasible* and *expandable* apply in an obvious manner to structure respecting mappings. The *cost* of a structure respecting mapping is defined analogously to the cost of a general mapping. In case of a structure respecting mapping only one direction of Lemma 1 holds.

Fact 1 *For any structure respecting mapping M from T_1 to T_2 there exists an edit sequence S transforming T_1 to T_2 such that $\gamma(S) = \gamma(M)$.* □

With respect to structure respecting mappings we now define the structure respecting distance between two trees.

Definition 10 (Structure respecting distance between trees) *Let T_1 and T_2 be two labeled ordered trees. Then the* structure respecting distance $\Delta(T_1, T_2)$ *between T_1 and T_2 is defined as follows*

$$\Delta(T_1, T_2) \quad := \quad \min\{\gamma(M) \mid M \text{ is a structure respecting mapping from } T_1 \text{ to } T_2\}.$$

A structure respecting mapping M with $\gamma(M) = \Delta(T_1, T_2)$ is called a minimum cost mapping. □

Before considering the computation of the structure respecting distance, we look at the properties of a structure respecting mapping.

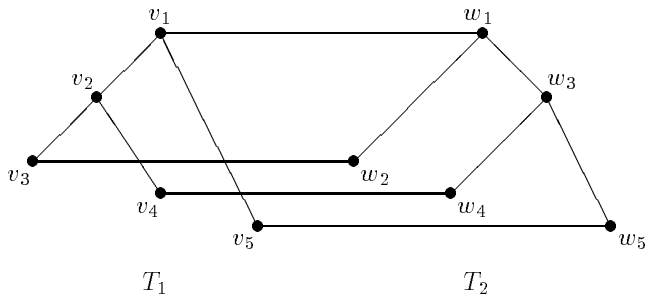

Figure 4: *The mapping M_1.*

Example 1 *As a first example of a not structure respecting mapping, consider the mapping M_1 shown in Figure 4 (in the figures we represent the connections of a mapping by a horizontal line connecting the nodes involved). M_1 is not structure respecting, because*

$$\text{LCA}(v_5, v_3) = v_1 = \text{LCA}(v_5, v_4),$$

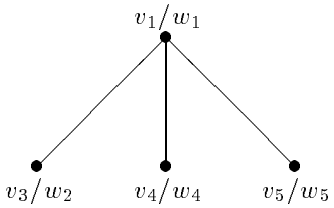

Figure 5: *The substructure* $S_{M_1}(T_1, T_2)$.

but

$$\text{LCA}(w_5, w_2) = w_1 \neq w_3 = \text{LCA}(w_5, w_4).$$

In Figure 5 the substructure $S_{M_1}(T_1, T_2)$ *of the trees* T_1 *and* T_2 *from Figure 4 induced by the mapping* M_1 *is shown.*

Note that the connection (v_4, w_4) *is "responsible" for the mapping* M_1 *not being structure respecting. While* v_4 *is in* T_1 *structurally closer to* v_3 *than to* v_5, w_4 *is in* T_2 *structurally closer to* w_5 *than to* w_2. *In the substructure* $S_{M_1}(T_1, T_2)$ *induced by* M_1 *the node* v_4/w_4 *is structurally equally close to* v_3/w_2 *and to* v_5/w_5, *so information contained in the structure of* T_1 *and* T_2 *is lost in the induced substructure.* □

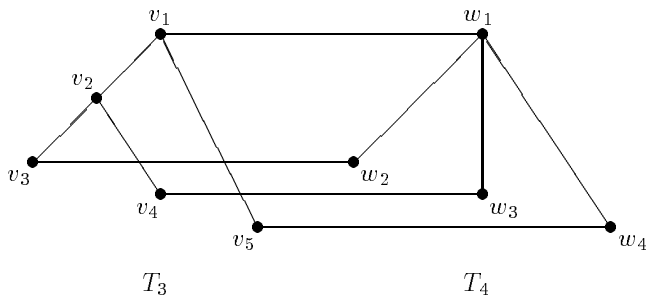

Figure 6: *The mapping* M_2.

Example 2 *As another example of a not structure respecting mapping consider the mapping* M_2 *shown in Figure 6.* M_2 *is not structure respecting, because*

$$\text{LCA}(v_3, v_4) = v_2 \neq v_1 = \text{LCA}(v_3, v_5),$$

but

$$\text{LCA}(w_2, w_3) = w_1 = \text{LCA}(w_2, w_4).$$

In Figure 7 the substructure $S_{M_2}(T_3, T_4)$ *of the trees* T_3 *and* T_4 *from Figure 6 induced by the mapping* M_2 *is shown. Note that this substructure is isomorphic to that of Example 1. Note further that this situation can only occur when considering trees with degree* > 2. □

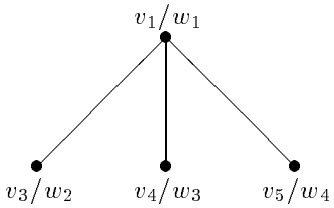

Figure 7: *The substructure* $S_{M_2}(T_3, T_4)$.

The examples have shown that the substructure of two k-nary trees induced by a general mapping can have degree higher than k. On the contrary, the following theorem shows that a substructure of two k-nary trees induced by a not expandable structure respecting mapping is always at most k-nary.

Theorem 2 *Let T_1 and T_2 be two labeled ordered k-nary trees with $k \geq 2$, and let M be a not expandable structure respecting mapping from T_1 to T_2. Then the substructure $S_M(T_1, T_2) = (V_M, E_M)$ of T_1 and T_2 induced by M is at most k-nary.*

PROOF. We show the following: If $S_M(T_1, T_2)$ is (at least) $(k+1)$-nary, then M is either expandable or not structure respecting.

Let be $v \in V_M$, such that v has in $S_M(T_1, T_2)$ (at least) $k+1$ children x_i, $1 \leq i \leq k+1$, in that order. Let v and the x_i's be nodes of T_1, and let w, y_j, $1 \leq j \leq k+1$, be the corresponding nodes of T_2.

Since T_1 is k-nary, there must be two nodes x_r, x_{r+1}, $1 \leq r < k+1$, for which either

(1) $\text{LCA}(x_r, x_{r+1}) = \widehat{x_1} \neq \widehat{x_2} = \text{LCA}(x_{r-1}, x_r)$

or

(2) $\text{LCA}(x_r, x_{r+1}) = \widehat{x_1} \neq \widehat{x_2} = \text{LCA}(x_{r+1}, x_{r+2})$.

Each of these cases is now considered in turn.

(1) Since $\widehat{x_1}$ and $\widehat{x_2}$ are both ancestors of x_r, one must be ancestor of the other. Hence we have to distinguish two subcases.

(a) $\widehat{x_1}$ *is ancestor of* $\widehat{x_2}$. Let y_{r-1}, y_r and y_{r+1} be the nodes of T_2, to which x_{r-1}, x_r and x_{r+1}, respectively, are connected. Let $\widehat{y_1} = \text{LCA}(y_r, y_{r+1})$ and $\widehat{y_2} = \text{LCA}(y_{r-1}, y_r)$. Since both $\widehat{y_1}$ and $\widehat{y_2}$ are ancestors of y_r, one of the following cases must hold.

(i) $\widehat{y_1} = \widehat{y_2}$. This situation is shown in Figure 8. Here we have

$$\text{LCA}(x_{r-1}, x_r) = \widehat{x_2} \neq \widehat{x_1} = \text{LCA}(x_{r-1}, x_r),$$

but

$$\text{LCA}(y_{r-1}, y_r) = \widehat{y_1} = \text{LCA}(y_{r-1}, y_r).$$

Hence M is not structure respecting.

(ii) $\widehat{y_1}$ *is ancestor of* $\widehat{y_2}$. This situation is shown in Figure 9. Here we have that M is expandable by the connection $(\widehat{x_2}, \widehat{y_2})$.

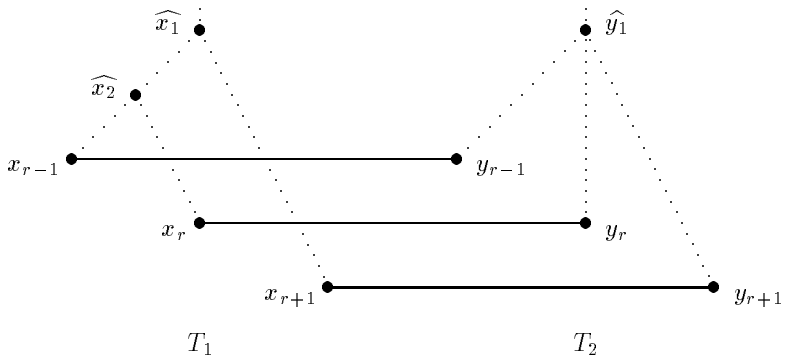

Figure 8: *Case (1.a.i) of Theorem 2.*

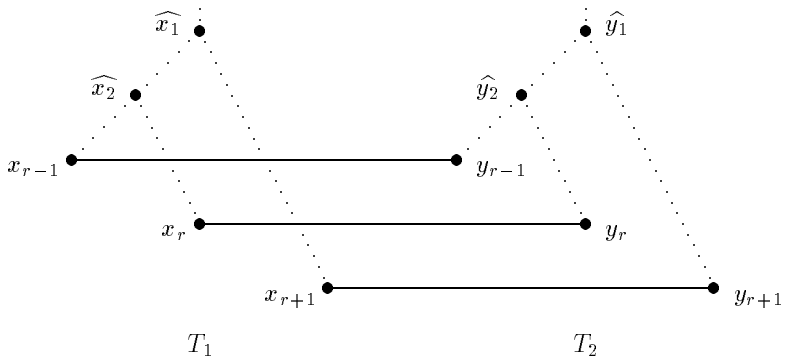

Figure 9: *Case (1.a.ii) of Theorem 2.*

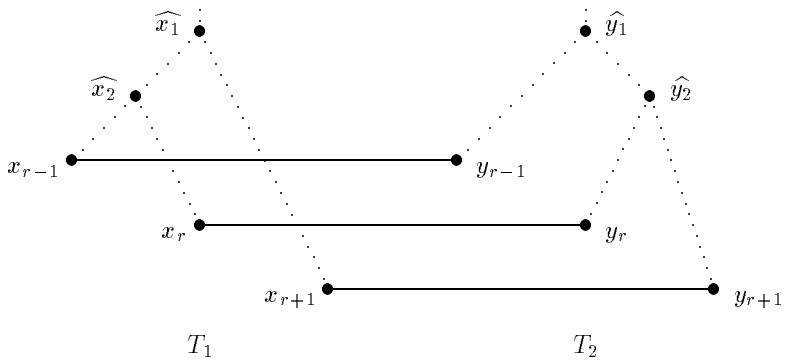

Figure 10: *Case (1.a.iii) of Theorem 2.*

(iii) $\widehat{y_2}$ *is ancestor of* $\widehat{y_1}$. This situation is shown in Figure 10. Here we have

$$\text{LCA}(x_{r-1}, x_r) = \widehat{x_2} \neq \widehat{x_1} = \text{LCA}(x_{r-1}, x_{r+1}),$$

but

$$\text{LCA}(y_{r-1}, y_r) = \widehat{y_2} = \text{LCA}(y_{r-1}, y_{r+1}).$$

Hence M is not structure respecting.

(b) $\widehat{x_2}$ *is ancestor of* $\widehat{x_1}$. Let y_{r-1}, y_r and y_{r+1} be the nodes of T_2, to which x_{r-1}, x_r and x_{r+1}, respectively, are connected. Let $\widehat{y_1} = \text{LCA}(y_r, y_{r+1})$ and $\widehat{y_2} = \text{LCA}(y_{r-1}, y_r)$. Since both $\widehat{y_1}$ and $\widehat{y_2}$ are ancestors of y_r, one of the following cases must hold.

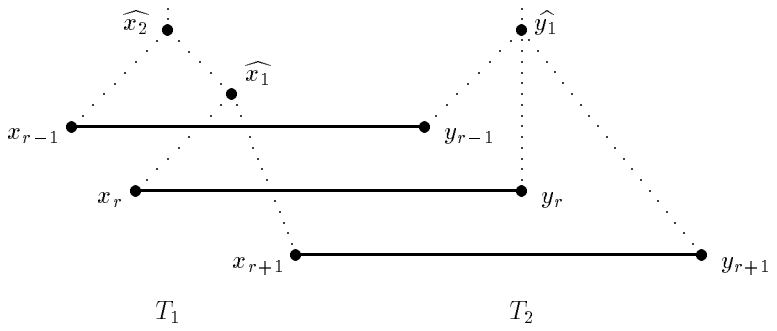

Figure 11: *Case (1.b.i) of Theorem 2.*

(i) $\widehat{y_1} = \widehat{y_2}$. This situation is shown in Figure 11. Here we have

$$\text{LCA}(x_r, x_{r-1}) = \widehat{x_2} \neq \widehat{x_1} = \text{LCA}(x_r, x_{r+1}),$$

but

$$\text{LCA}(y_r, y_{r-1}) = \widehat{y_1} = \text{LCA}(y_r, y_{r+1}).$$

Hence M is not structure respecting.

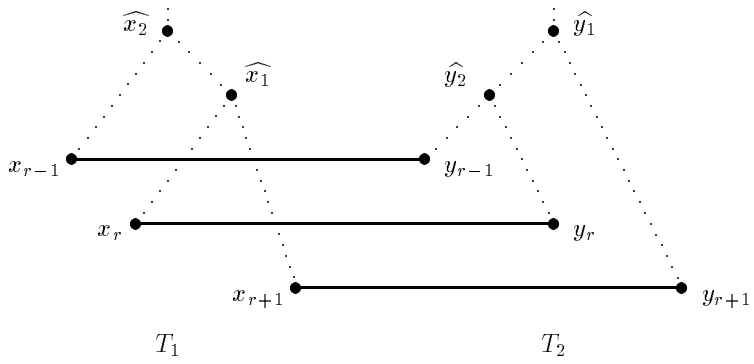

Figure 12: *Case (1.b.ii) of Theorem 2.*

(ii) $\widehat{y_1}$ *is ancestor of* $\widehat{y_2}$. This situation is shown in Figure 12. Here we have

$$\text{LCA}(x_{r-1}, x_r) = \widehat{x_2} = \text{LCA}(x_{r-1}, x_{r+1}),$$

but

$$\text{LCA}(y_{r-1}, y_r) = \widehat{y_2} \neq \widehat{y_1} = \text{LCA}(y_{r-1}, y_{r+1}).$$

Hence M is not structure respecting.

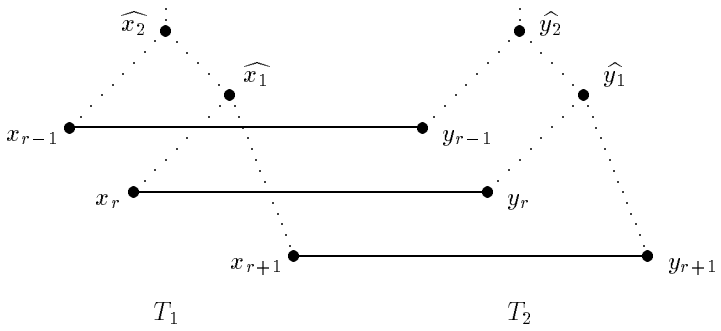

Figure 13: *Case (1.b.iii) of Theorem 2.*

(iii) $\widehat{y_2}$ *is ancestor of* $\widehat{y_1}$. This situation is shown in Figure 13. Here we have that M is expandable by the connection $(\widehat{x_1}, \widehat{y_1})$.

(2) Since $\widehat{x_1}$ and $\widehat{x_2}$ are both ancestors of x_{r+1}, one must be ancestor of the other. Hence we have to distinguish two subcases.

(a) $\widehat{x_1}$ *is ancestor of* $\widehat{x_2}$. Let y_r, y_{r+1} and y_{r+2} be the nodes of T_2, to which x_r, x_{r+1} and x_{r+2}, respectively, are connected. Let $\widehat{y_1} = \text{LCA}(y_r, y_{r+1})$ and $\widehat{y_2} = \text{LCA}(y_{r+1}, y_{r+1})$. Since both $\widehat{y_1}$ and $\widehat{y_2}$ are ancestors of y_{r+1}, one of the following cases must hold.

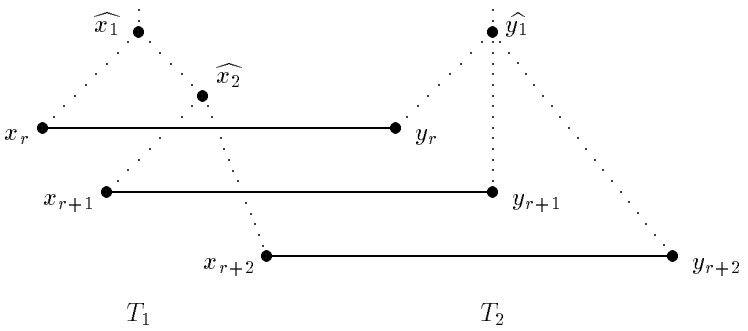

Figure 14: *Case (2.a.i) of Theorem 2.*

(i) $\widehat{y_1} = \widehat{y_2}$. This situation is shown in Figure 14. It is obviously analogous to the situatation in Case (1.b.i).

The Cases (ii) and (iii) are analogous to the Cases (1.b.iii) and (1.b.ii), respectively.

(b) $\widehat{x_2}$ *is ancestor of* $\widehat{x_1}$. This case is analogous to Case (1.a). □

In the case of binary trees the following theorem holds, which shows that a not expandable general mapping between binary trees that induces a binary substructure is structure respecting.

Theorem 3 *Let T_1 and T_2 be two labeled ordered binary trees, let M be a not expandable general mapping from T_1 to T_2, and let the substructure $S_M(T_1, T_2) = (V_M, E_M)$ of T_1 and T_2 induced by M be at most binary. Then M is structure respecting.*

PROOF. Assume that M is not structure respecting, i. e. that there are connections (x_1, y_1), (x_2, y_2), $(x_3, y_3) \in M$, such that none of the nodes x_1, x_2 and x_3 is an ancestor of one of the others, and

$$\text{LCA}(x_1, x_2) = \text{LCA}(x_1, x_3),$$

but

$$\text{LCA}(y_1, y_2) \neq \text{LCA}(y_1, y_3)$$

(the reverse case "$\text{LCA}(y_1, y_2) = \text{LCA}(y_1, y_3)$, but ..." is symmetrical). We can assume that the numbering of the nodes corresponds to their left-to-right ordering, because we can force this by renumbering the nodes and by switching the order of the nodes simultaneously in both trees.

Let $\widehat{x_1} = \text{LCA}(x_1, x_2) = \text{LCA}(x_1, x_3)$ and $\widehat{x_2} = \text{LCA}(x_2, x_3)$. Since T_1 is binary, $\widehat{x_2}$ must be a proper descendant of $\widehat{x_1}$.

Let $\widehat{y_1} = \text{LCA}(y_1, y_2)$ and $\widehat{y_2} = \text{LCA}(y_1, y_3)$. Since both $\widehat{y_1}$ and $\widehat{y_2}$ are ancestors of y_1, $\widehat{y_1} \neq \widehat{y_2}$ and y_2 is to the left of y_3, $\widehat{y_2}$ has to be a proper ancestor of $\widehat{y_1}$.

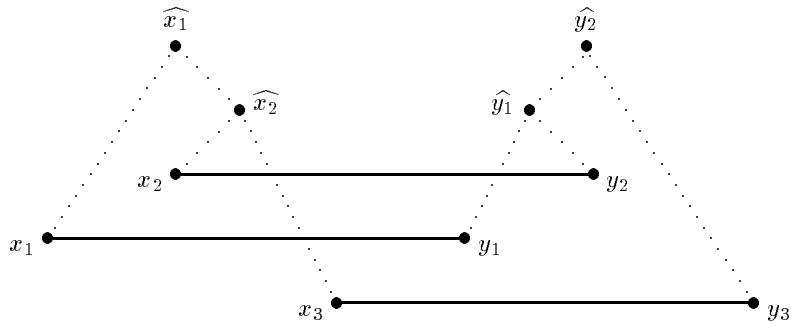

Figure 15: *The disposition of the three connections in Theorem 3.*

Then the three connections of M under consideration are disposed as shown in Figure 15. Note that at least one of $\widehat{x_1}$ and $\widehat{y_2}$ has to be covered by M, because M is not expandable. However, we do not distinguish the resulting cases, because in any case the following argumentation applies.

Since $S_M(T_1, T_2)$ is binary, there must be a common ancestor $\hat{v}$ of x_2 and x_3 that is a proper descendant of $\widehat{x_1}$ and that is covered by M. With regard to the ancestor condition of a mapping, the node $\hat{w}$, to which $\hat{v}$ is connected, has to be a common ancestor of y_2 and y_3, i. e. the node $\widehat{y_2}$ or an ancestor of it. Then $\hat{w}$ is an ancestor of y_1, but $\hat{v}$ is not an ancestor of x_1. This contradicts the definition of a mapping. □

Note that the assumption that M is not expandable in the theorem is necessary. Note further that the theorem does not claim that a minimum cost general mapping is structure respecting, since such a mapping can be ternary or of higher degree.

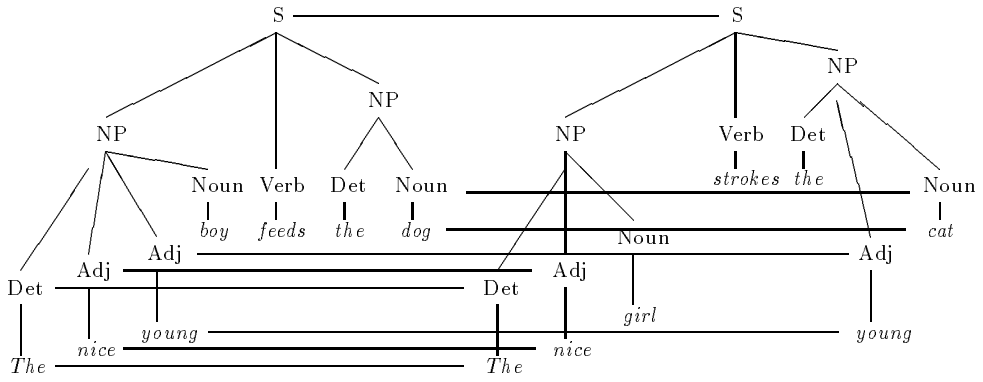

Figure 16: *An example comparison of parse trees with a general mapping.*

We conclude this section with an example that demonstrates the usefulness of structure respecting mappings in practice.

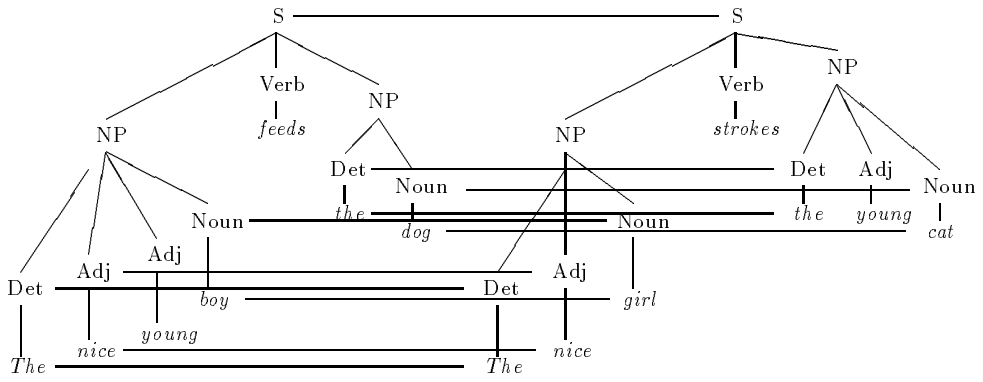

Figure 17: *An example comparison of parse trees with a structure respecting mapping.*

Example 3 *In Figure 16 parse trees of the sentences* The nice young boy feeds the dog *and* The nice girl strokes the young cat *and a general (not necessarily minimum cost)*

mapping between them are given. This mapping is not structure respecting because of the connections between the adjective young *in both sentences. These connections mix the left und the right noun phrases* NP*. This could not happen in a structure respecting mapping. For example, the structure respecting mapping between these two sentences shown in Figure 17 connects the left noun phrases and the right noun phrases separately (in this figure we have omitted connections between the* NP *nodes and the verbs for clarity). Obviously, this mapping reflects more of syntactic similarity between the two sentences.*

This example has shown that it can be useful to consider the parse trees when one compares two (natural language) sentences to determine their syntactic similarity. Furthermore, it has demonstrated that the restriction to structure respecting mappings could give more significant results in these comparisons. □

4 Computation of the Structure Respecting Distance

4.1 Introduction

Let $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ be two labeled ordered trees. For $v \in V_1$ and $w \in V_2$ we denote the structure respecting distance between the subtrees $T_1[v]$ and $T_2[w]$ by $\textsc{Tdist}(v, w)$. Let v have the children v_i, $1 \leq i \leq k_1$ and w the children w_j, $1 \leq j \leq k_2$. Then we denote by $\textsc{Fdist}(v_{i_1} \ldots v_{i_2}, w_{j_1} \ldots w_{j_2})$ the structure respecting distance between the subforests $F_1[v_{i_1} \ldots v_{i_2}]$, $1 \leq i_1 < i_2 \leq k_1$, and $F_2[w_{j_1} \ldots w_{j_2}]$, $1 \leq j_1 < j_2 \leq k_2$, where the distance between forests is defined analogously to the distance between trees; especially the left-to-right ordering of the trees of a forest is significant. To denote the distance beween all subforests of $T_1[v]$ and $T_2[w]$, i. e. the distance $\textsc{Fdist}(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$, we often use $\textsc{Fdist}(v, w)$ as an abbreviation.

To compute the structure respecting distance $\Delta(T_1, T_2)$ between the trees T_1 and T_2, we consider their nodes in ascending left-to-right postorder and compute at any time the distance between the corresponding subtrees of T_1 and T_2 using intermediate results computed previously. I. e. we compute the values of $\textsc{Fdist}(v, w)$ and $\textsc{Tdist}(v, w)$ for all pairs $(v, w) \in V_1 \times V_2$ by considering the nodes of each tree in ascending left-to-right postorder. The basis for these computations is the distances $\textsc{Fdist}(v, \textsc{nil})$, $\textsc{Tdist}(v, \textsc{nil})$, $\textsc{Fdist}(\textsc{nil}, w)$ and $\textsc{Tdist}(\textsc{nil}, w)$ (NIL denotes the empty tree) between the subforests and subtrees of T_1 and T_2, respectively, and an empty tree. These values are used if a subforest or a subtree of T_1 (of T_2) is completely deleted (inserted). Here we have the following fact.

Fact 2 *Let v be a node of T_1 with children v_i, $1 \leq i \leq k_1$, and w be a node of T_2 with children w_j, $1 \leq j \leq k_2$. Then we have*

(1) $\textsc{Fdist}(\textsc{nil}, \textsc{nil}) = \textsc{Tdist}(\textsc{nil}, \textsc{nil}) = 0$;

(2) $\textsc{Fdist}(v, \textsc{nil}) = \sum_{1 \leq i \leq k_1} \textsc{Tdist}(v_i, \textsc{nil})$,
$\textsc{Tdist}(v, \textsc{nil}) = \gamma(\textsc{label}(v) \to \epsilon) + \textsc{Fdist}(v, \textsc{nil})$;

(3) $\textsc{Fdist}(\textsc{nil}, w) = \sum_{1 \leq j \leq k_2} \textsc{Tdist}(\textsc{nil}, w_j)$,
$\textsc{Tdist}(\textsc{nil}, w) = \gamma(\epsilon \to \textsc{label}(w)) + \textsc{Fdist}(\textsc{nil}, w)$. □

Given these values we can devote ourselves to the computation of $\textsc{Tdist}(v, w)$ and $\textsc{Fdist}(v, w)$.

4.2 Computation of TDIST

Given the subtrees $T_1[v]$ and $T_2[w]$, we want to compute their structure respecting distance TDIST(v, w). We assume in our description that v has the children v_i, $1 \leq i \leq k_1$, and w the children w_j, $1 \leq j \leq k_2$.

The following lemma limits the number of cases to be considered in the computation of TDIST(v, w).

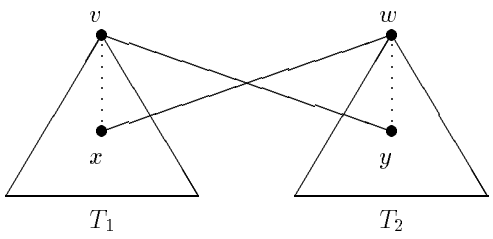

Figure 18: *Case (b) of Lemma 2.*

Lemma 2 *Let M be a mapping from $T_1[v]$ to $T_2[w]$. Then we have*

(a) if v and w are both not covered by M, then M is expandable;

(b) if v and w are both covered by M, then $(v, w) \in M$.

PROOF.

(a) If v and w are not covered by M, then $M \cup \{(v, w)\}$ is obviously a feasible mapping.

(b) Assume that v is connected to a node y in $F_2[w_1 \ldots w_{k_2}]$ and that w is connected to a node x in $F_1[v_1 \ldots v_{k_1}]$, as shown in Figure 18. Then x is a descendant of v and y is a descendant of w. Hence we have for (v, y) and (x, w) that x is a descendant of v, but w is not a descendant of y. This contradicts the definition of a feasible mapping. □

Hence computing the distance TDIST(v, w) we have to consider only the following three cases:

(1) $v \in D_M(T_1[v])$, $w \notin R_M(T_2[w])$;

(2) $v \notin D_M(T_1[v])$, $w \in R_M(T_2[w])$:

(3) $(v, w) \in M$.

Each of these cases is now considered in turn.

Case (1). $v \in D_M(T_1[v]), w \notin R_M(T_2[w])$. In this case the node v, but not the node w, is covered by M. We have the following lemma.

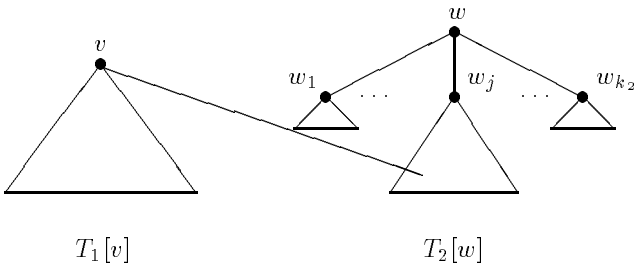

Figure 19: *The situation of Lemma 3.*

Lemma 3 *If M is a minimum cost structure respecting mapping from $T_1[v]$ to $T_2[w]$ and M covers the node v but not the node w, then the cost of M is*

$$\gamma(M) = \textsc{Tdist}(\textsc{nil}, w) + \min_{1 \le j \le k_2} \{\textsc{Tdist}(v, w_j) - \textsc{Tdist}(\textsc{nil}, w_j)\}.$$

Proof. Since v is not connected to w, v has to be connected to a node in subtree $T_2[w_j]$, $1 \le j \le k_2$, of $T_2[w]$ as shown in Figure 19. Then there can only be connections between $T_1[v]$ and $T_2[w_j]$. Hence the node w and all of its subtrees except $T_2[w_j]$ have to be inserted. The costs of these operations are

$$\textsc{Tdist}(\textsc{nil}, w) - \textsc{Tdist}(\textsc{nil}, w_j).$$

Since M is a minimum cost mapping from $T_1[v]$ to $T_2[w]$, M is also a minimum cost mapping from $T_1[v]$ to $T_2[w_j]$. Hence, considering M as a mapping from $T_1[v]$ to $T_2[w_j]$ we get

$$\gamma(M) = \textsc{Tdist}(v, w_j).$$

Considering M again as a mapping from $T_1[v]$ to $T_2[w]$, we get altogether

$$\gamma(M) = \textsc{Tdist}(\textsc{nil}, w) + \textsc{Tdist}(v, w_j) - \textsc{Tdist}(\textsc{nil}, w_j).$$

Note that either $\textsc{Tdist}(v, w_j)$ or $\textsc{Tdist}(\textsc{nil}, w_j)$ may be larger. But, since M is a minimum cost mapping, $T_2[w_j]$ must be that subtree of $T_2[w]$ for which the difference $\textsc{Tdist}(v, w_j) - \textsc{Tdist}(\textsc{nil}, w_j)$ is smallest. Hence we have

$$\gamma(M) = \textsc{Tdist}(\textsc{nil}, w) + \min_{1 \le j \le k_2} \{\textsc{Tdist}(v, w_j) - \textsc{Tdist}(\textsc{nil}, w_j)\}.$$

□

Note that if there is no minimum cost mapping from $T_1[v]$ to $T_2[w]$ covering the node v but not the node w, then the distance $\textsc{Tdist}(v, w_j)$ may correspond to a mapping M that does not cover the node v. Hence M considered as a mapping from $T_1[v]$ to $T_2[w]$ may be expandable. But this does not matter, because in this case the cost of M is beaten by the cost of a mapping considered in **Case (2)** or in **Case (3)**.

Case (2). $v \notin D_M(T_1[v]), w \in R_M(T_2[w])$. In this case the node w, but not the node v, is covered by M. Analogously to **Case (1)**, here we have the following lemma.

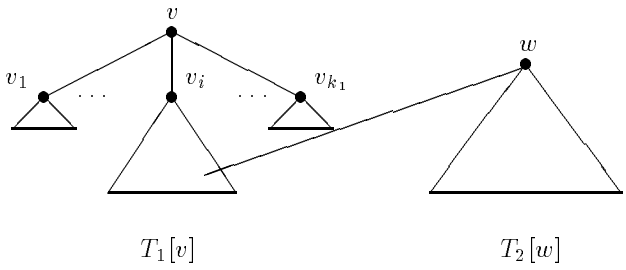

Figure 20: *The situation of Lemma 4.*

Lemma 4 *If M is a minimum cost structure respecting mapping $T_1[v]$ to $T_2[w]$ and M covers the node w but not the node v, then the cost of M is*

$$\gamma(M) = \textsc{Tdist}(v, \textsc{nil}) + \min_{1 \le i \le k_1} \{\textsc{Tdist}(v_i, w) - \textsc{Tdist}(v_i, \textsc{nil})\}.$$

Proof. Analogously to Lemma 3. □

Note that if there is no minimum cost mapping from $T_1[v]$ to $T_2[w]$ covering the node w but not the node v, then the distance $\textsc{Tdist}(v_l, w)$ may correspond to a mapping M that does not cover the node w. Hence M considered as a mapping from $T_1[v]$ to $T_2[w]$ may be expandable. But this does not matter, because in this case the cost of M is beaten by the cost of a mapping considered in **Case (1)** or in **Case (3)**.

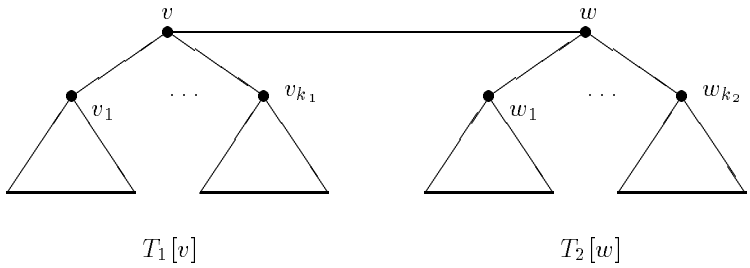

Figure 21: *The general situation in* **Case (3)**.

Case (3). $(v, w) \in M$. In this case the node v as well as the node w is covered by the mapping M. Hence all mappings which have to be considered include the connection (v, w), as shown in Figure 21. We denote the minimum cost of such a mapping from $T_1[v]$ to $T_2[w]$ by $\textsc{Tdist}_3(v, w)$. This distance is composed of the cost $\gamma(\textsc{label}(v) \to \textsc{label}(w))$ of the connection (v, w) and the distance $\textsc{Fdist}(v, w)$ between the forests $F_1[v_1 \ldots v_{k_1}]$ and $F_2[w_1 \ldots w_{k_2}]$.

In computing the distance $\textsc{Fdist}(v, w)$ the following lemmata limit the number of cases to be considered.

Lemma 5 *Let M be a mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$. If there are $1 \leq i_1 < i_2 \leq k_1$ and $1 \leq j_1 < j_2 \leq k_2$, such that there are connections in M both between $T_1[v_{i_1}]$ and $T_2[w_{j_2}]$ and between $T_1[v_{i_2}]$ and $T_2[w_{j_1}]$, then M is not feasible.*

PROOF. Let (x_1, y_1) be an arbitrary connection between $T_1[v_{i_1}]$ and $T_2[w_{j_2}]$, and let

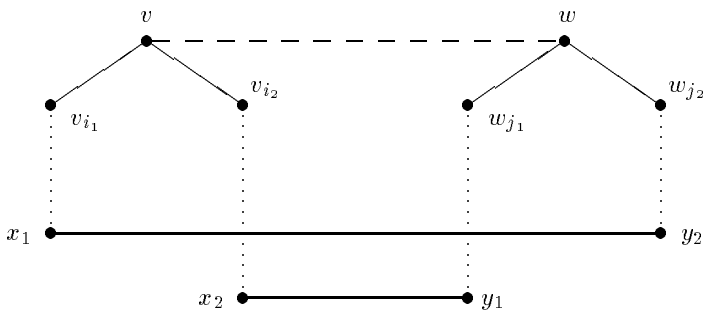

Figure 22: *The situation of Lemma 5.*

(x_2, y_2) be an arbitrary connection between $T_1[v_{i_2}]$ and $T_2[w_{j_1}]$, as shown in Figure 22. Then x_1 is to the left of x_2, and y_1 is to the right of y_2. This contradicts the definition of a feasible mapping. □

The following lemma generalizes Example 1.

Lemma 6 *Let M be a mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$, and let M contain connections between $T_1[v_{i_1}]$ and $T_2[w_{j_1}]$ and between $T_1[v_{i_2}]$ and $T_2[w_{j_2}]$, $1 \leq i_1 < i_2 \leq k_1$ and $1 \leq j_1 < j_2 \leq k_2$. If there are additional connections in M either*

(a) between $T_1[v_{i_2}]$ and $T_2[w_{j_1}]$

or

(b) between $T_1[v_{i_1}]$ and $T_2[w_{j_2}]$,

then M is not structure respecting.

PROOF.

(a) Let (x_1, y_1) be an arbitrary connection between $T_1[v_{i_1}]$ and $T_2[w_{j_1}]$, (x_2, y_2) be an arbitrary connection between $T_1[v_{i_2}]$ and $T_2[w_{j_1}]$, and (x_3, y_3) be an arbitrary connection between $T_1[v_{i_2}]$ and $T_2[w_{j_2}]$. Then y_1 is to the left of y_2 and x_2 is to the left of x_3 as shown in Figure 23. Let $\hat{y}$ be the lowest common ancestor of y_1 and y_2. Then we have

$$\text{LCA}(x_1, x_2) = v = \text{LCA}(x_1, x_3),$$

but

$$\text{LCA}(y_1, y_2) = \hat{y} \neq w = \text{LCA}(y_1, y_3).$$

(b) Analogously, see Figure 24. □

The following two lemmata generalize Example 2.

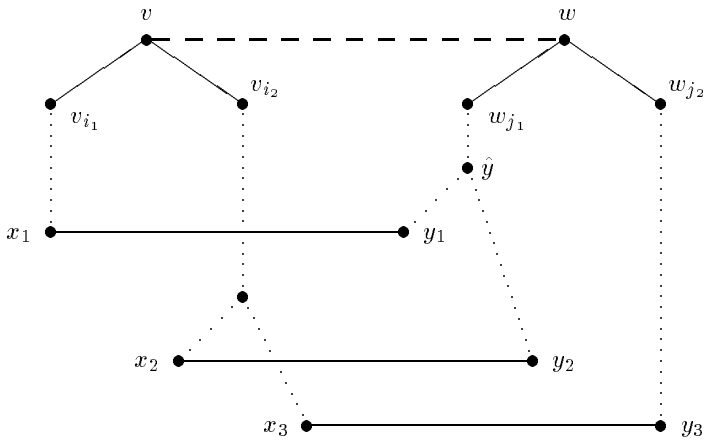

Figure 23: *Case (a) of Lemma 6.*

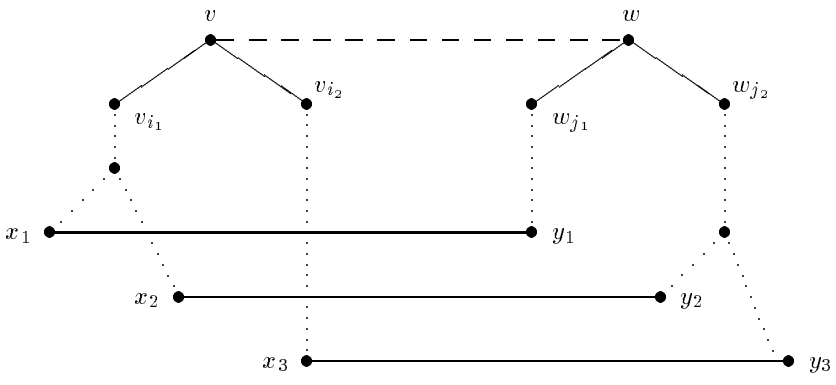

Figure 24: *Case (b) of Lemma 6.*

Lemma 7 *Let M be a mapping from $F_1[v_1 \dots v_{k_1}]$ to $F_2[w_1 \dots w_{k_2}]$, $k_2 > 2$, and let M contain connections between $T_1[v_l]$ and $T_2[w_{l_1}]$ and between $T_1[v_l]$ and $T_2[w_{l_2}]$, $1 \leq l \leq k_1$, $1 \leq l_1 < l_2 \leq k_2$. If there are additional connections in M either*

(a) between $T_1[v_{i_1}]$ and $T_2[w_{j_1}]$, $1 \leq i_1 < l$, $1 \leq j_1 < l_1$

or

(b) between $T_1[v_{i_2}]$ and $T_2[w_{j_2}]$, $l < i_2 \leq k_1$, $l_2 < j_2 \leq k_2$,

then M is not structure respecting.
PROOF.

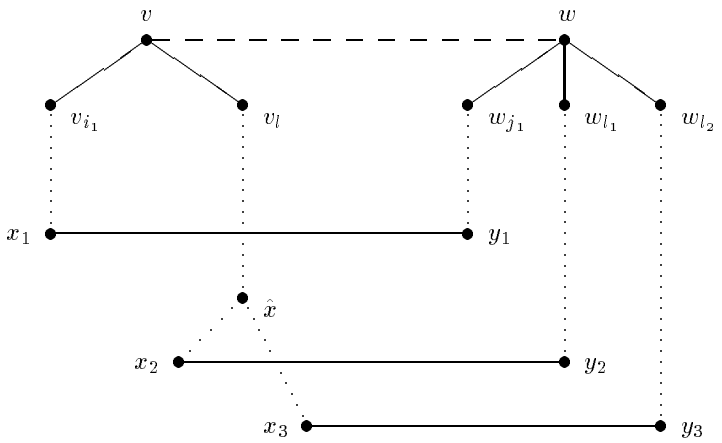

Figure 25: *Case (a) of Lemma 7.*

(a) Let (x_1, y_1) be an arbitrary connection between $T_1[v_{i_1}]$ and $T_2[w_{j_1}]$, (x_2, y_2) be an arbitrary connection between $T_1[v_l]$ and $T_2[w_{l_1}]$, and (x_3, y_3) be an arbitrary connection between $T_1[v_l]$ and $T_2[w_{l_2}]$, as shown in Figure 25. Let $\hat{x}$ be the lowest common ancestor of x_2 and x_3. Then we have

$$\text{LCA}(x_2, x_1) = v \neq \hat{x} = \text{LCA}(x_2, x_3),$$

but

$$\text{LCA}(y_2, y_1) = w = \text{LCA}(y_2, y_3).$$

(b) Analogously, see Figure 26. □

Symmetrically, we have the following lemma, the cases of which are illustrated in Figures 27 and 28, respectively.

Lemma 8 *Let M be a mapping from $F_1[v_1 \dots v_{k_1}]$, $k_1 > 2$, to $F_2[w_1 \dots w_{k_2}]$, and let M contain connections between $T_1[v_{l_1}]$ and $T_2[w_l]$ and between $T_1[v_{l_2}]$ and $T_2[w_l]$, $1 \leq l_1 < l_2 \leq k_1$, $1 \leq l \leq k_2$. If there are additional connections in M either*

(a) between $T_1[v_{i_1}]$ and $T_2[w_{j_1}]$, $1 \leq i_1 < l_1$, $1 \leq j_1 < l$

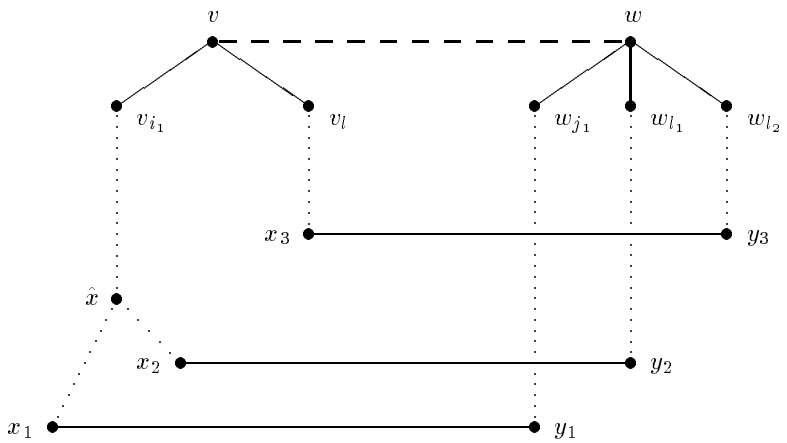

Figure 26: *Case (b) of Lemma 7.*

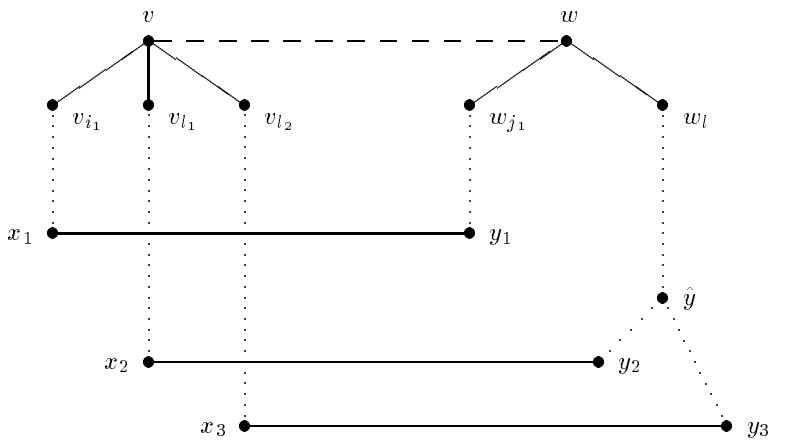

Figure 27: *Case (a) of Lemma 8.*

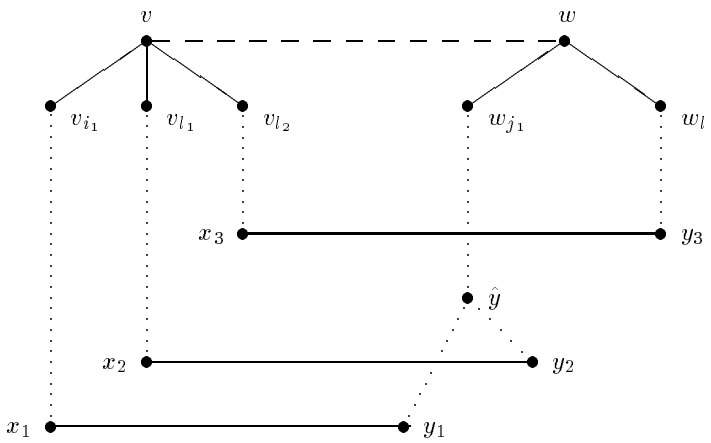

Figure 28: *Case (b) of Lemma 8.*

or

(b) between $T_1[v_{i_2}]$ *and* $T_2[w_{j_2}]$, $l_2 < i_2 \leq k_1$, $l < j_2 \leq k_2$,

then M *is not structure respecting.* □

The following corollary is an immediate consequence of these two lemmata.

Corollary 1 *Let* M *be a structure respecting mapping from* $F_1[v_1 \ldots v_{k_1}]$ *to* $F_2[w_1 \ldots w_{k_2}]$. *Then there can be at most one subtree of* $T_1[v_1], \ldots T_1[v_{k_1}], T_2[w_1], \ldots T_2[w_{k_2}]$ *that is mapped onto more than one other subtree by* M. □

We call a mapping M from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$ a *left star mapping* if there is a subtree $T_1[v_i]$, $1 \leq i \leq k_1$ that is mapped onto more than one subtree of $T_2[w]$, and a *right star mapping* if there is a subtree $T_2[w_j]$, $1 \leq j \leq k_2$ that is mapped onto more than one subtree of $T_1[v]$. The node v_i or w_j is then called the *center node* of the mapping. Note that at most one of $v_1, \ldots v_{k_1}, w_1 \ldots w_{k_2}$ can be a center node of a mapping. Furthermore, the other subtrees of $T_1[v]$ or $T_2[w]$) cannot be covered by the mapping if v_i or w_j is a center node.

If none of $v_1, \ldots v_{k_1}, w_1 \ldots w_{k_2}$ is a center node, then the mapping is composed of submappings between pairs of single subtrees of $T_1[v]$ and $T_2[w]$. This means that every subtree of $T_1[v_1], \ldots T_1[v_{k_1}], T_2[w_1], \ldots T_2[w_{k_2}]$ is mapped onto at most one other subtree. We call such a mapping a *fork mapping*.

To compute $\textsc{Fdist}(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$ we first consider the fork mappings from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$. The minimum cost of such a mapping is denoted by $\textsc{Fdist}_F(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$. To compute $\textsc{Fdist}_F(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$ we consider the subtrees of $T_1[v]$ and $T_2[w]$ from the left to the right, i. e. we compute $\textsc{Fdist}_F(v_1 \ldots v_i, w_1 \ldots w_j)$ step-by-step for ascending i and j. Here we have the following lemma.

Lemma 9 *The cost of a minimum cost fork mapping from $F_1[v_1 \dots v_i]$, $1 \leq i \leq k_1$ to $F_2[w_1 \dots w_j]$, $1 \leq j \leq k_2$, is*

$$\textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_j) = \min \left\{ \begin{array}{l} \textsc{Tdist}(v_i, w_j) + \textsc{Fdist}_F(v_1 \dots v_{i-1}, w_1 \dots w_{j-1}), \\ \textsc{Tdist}(v_i, \textsc{nil}) + \textsc{Fdist}_F(v_1 \dots v_{i-1}, w_1 \dots w_j), \\ \textsc{Tdist}(\textsc{nil}, w_j) + \textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_{j-1}) \end{array} \right\}.$$

Proof. Since $\textsc{Tdist}(v_i, w_j) \leq \textsc{Tdist}(v_i, \textsc{nil}) + \textsc{Tdist}(\textsc{nil}, w_j)$ by definition of the cost function γ, we have to distinguish in the computation of $\textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_j)$ only the following three cases.

(a) *$T_1[v_i]$ and $T_2[w_j]$ are both covered by the mapping.* Then $T_1[v_i]$ must be mapped onto $T_2[w_j]$ due to Lemma 5. Hence we combine a fork mapping from $F_1[v_1 \dots v_{i-1}]$ to $F_2[w_1 \dots w_{j-1}]$ with a mapping from $T_1[v_i]$ to $T_2[w_j]$. Such a mapping M with minimum cost has the cost

$$\textsc{Tdist}(v_i, w_j) + \textsc{Fdist}_F(v_1 \dots v_{i-1}, w_1 \dots w_{j-1}).$$

(b) *Only $T_1[v_i]$ is covered by the mapping.* Then $T_2[w_j]$ has to be inserted and $T_1[v_i]$ is mapped to a subtree $T_2[w_l]$, $1 \leq l < j$. Hence we combine a special fork mapping from $F_1[v_1 \dots v_i]$ to $F_2[w_1 \dots w_{j-1}]$ with the insertion of $T_2[w_j]$. As every fork mapping from $F_1[v_1 \dots v_i]$ to $F_2[w_1 \dots w_{j-1}]$ is also a fork mapping from $F_1[v_1 \dots v_i]$ to $F_2[w_1 \dots w_j]$, we can cover this case by the cost

$$\textsc{Tdist}(\textsc{nil}, w_j) + \textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_{j-1}).$$

If $\textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_{j-1})$ corresponds to a mapping M that does not cover $T_1[v_i]$, then M considered as a mapping from $F_1[v_1 \dots v_i]$ to $F_2[w_1 \dots w_j]$ is expandable. But then this cost is beaten by the cost of case (a).

(c) *Only $T_2[w_j]$ is covered by the mapping.* Analogously to case (b) this case is covered by the cost

$$\textsc{Tdist}(v_i, \textsc{nil}) + \textsc{Fdist}_F(v_1 \dots v_{i-1}, w_1 \dots w_j).$$

If $\textsc{Fdist}_F(v_1 \dots v_{i-1}, w_1 \dots w_j)$ corresponds to a mapping M that does not cover $T_2[w_j]$, then M considered as a mapping from $F_1[v_1 \dots v_i]$ to $F_2[w_1 \dots w_j]$ is expandable. But then this cost is again beaten by the cost of case (a). □

Note that all subdistances used in the formula of the lemma have already been computed before computing $\textsc{Fdist}_F(v_1 \dots v_i, w_1 \dots w_j)$.

Now we consider the star mappings. The minimum cost of a star mapping from $F_1[v_1 \dots v_{k_1}]$ to $F_2[w_1 \dots w_{k_2}]$ is denoted by $\textsc{Fdist}_S(v_1 \dots v_{k_1}, w_1 \dots w_{k_2})$.

Let v_i, $1 \leq i \leq k_1$, be the center node of a left star mapping. Then $T_1[v_i]$ is mapped onto a subforest $F_2[j_1 \dots j_2]$, $1 \leq j_1 < j_2 \leq k_2$, and the other subtrees of $T_1[v]$ are deleted. The costs for these deletions are

$$\textsc{Fdist}(v_1 \dots v_{k_1}, \textsc{nil}) - \textsc{Tdist}(v_i, \textsc{nil}).$$

Next we have the following lemma.

Lemma 10 *Let M be a left star mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_{j_1} \ldots w_{j_2}]$, $1 \leq j_1 < j_2 \leq k_2$, with center node v_i, $1 \leq i \leq k_1$. Then the center node is not covered by M.*
PROOF. Assume that M covers the center node v_i, i. e. v_i is connected to a node $\hat{w}$ which is in a subtree $T_2[w_l]$, $j_1 \leq l \leq j_2$. Then M can only contain connections between $T_1[v_i]$ and $T_2[w_l]$. Hence M cannot be a star mapping. □

Consequently a left star mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_{j_1} \ldots w_{j_2}]$ with center node v_i is a mapping from $F_1[v_{i1} \ldots v_{ik_3}]$ to $F_2[w_{j_1} \ldots w_{j_2}]$, where v_{i1} to v_{ik_3} are the children of v_i. On the other hand, every structure respecting mapping from $F_1[v_{i1} \ldots v_{ik_3}]$ to $F_2[w_{j_1} \ldots w_{j_2}]$ is also a structure respecting mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_{j_1} \ldots w_{j_2}]$. Hence we have for the cost $\gamma(M)$ of a minimum cost star mapping M from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$ with center node v_i the following formula.

$$\begin{aligned}
\gamma(M) \;=\; & \textsc{Fdist}(v_1 \ldots v_{k_1}, \textsc{nil}) - \textsc{Tdist}(v_i, \textsc{nil}) + \\
& \gamma(\textsc{label}(v_i) \to \varepsilon) + \\
& \min_{1 \leq j_1 < j_2 \leq k_2} \{\textsc{Fdist}(v_{i1} \ldots v_{ik_3}, w_{j_1} \ldots w_{j_2}) + && (1) \\
& \qquad \textsc{Fdist}(\textsc{nil}, w_1 \ldots w_{j_1-1}) + && (2) \\
& \qquad \textsc{Fdist}(\textsc{nil}, w_{j_2+1} \ldots w_{k_2})\} && (3)
\end{aligned}$$

The costs over which we minimize in lines (4.1) to (4.3) are the costs of certain mappings from $F_1[v_{i1} \ldots v_{ik_3}]$ to $F_2[w_1 \ldots w_{k_2}]$. Since every structure respecting mapping from $F_1[v_{i1} \ldots v_{ik_3}]$ to $F_2[w_1 \ldots w_{k_2}]$ is on the other hand also a structure respecting mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$, we can replace the lines (4.1) to (4.3) by $\textsc{Fdist}(v_{i1} \ldots v_{ik}, w_1 \ldots w_k)$. Hence we have

$$\begin{aligned}
\gamma(M) \;=\; & \textsc{Fdist}(v_1 \ldots v_k, \textsc{nil}) - \textsc{Tdist}(v_i, \textsc{nil}) + \\
& \gamma(\textsc{label}(v_i) \to \varepsilon) + \textsc{Fdist}(v_i, w).
\end{aligned}$$

If we minimize this cost over all v_i we get the cost of a minimum cost left star mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$. In the case of right star mappings, an analogous argumentation applies.

Altogether we have the following lemma for the computation of $\textsc{Fdist}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$.

Lemma 11 *The cost of a minimum cost star mapping from $F_1[v_1 \ldots v_{k_1}]$ to $F_2[w_1 \ldots w_{k_2}]$ is*

$$\begin{aligned}
& \textsc{Fdist}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}) = \min\{ \\
& \quad \min_{1 \leq i \leq k_1} \{\textsc{Fdist}(v, \textsc{nil}) - \textsc{Tdist}(v_i, \textsc{nil}) + && (4) \\
& \qquad \gamma(\textsc{label}(v_i) \to \varepsilon) + \textsc{Fdist}(v_i, w)\}, \\
& \quad \min_{1 \leq j \leq k_2} \{\textsc{Fdist}(\textsc{nil}, w) - \textsc{Tdist}(\textsc{nil}, w_j) + && (5) \\
& \qquad \gamma(\varepsilon \to \textsc{label}(w_j)) + \textsc{Fdist}(v, w_j)\}\}.
\end{aligned}$$

□

Note that the formula in line (4.4) corresponds to the left star mappings and the formula in line (4.5) to the right star mappings. Note further that all subdistances used in this formula have, again, already been computed before computing $\textsc{Fdist}_S(v_1 \ldots v_{k_1}, w_i \ldots w_{k_2})$.

Summarizing, we get the following results for Case **(3)**.

Corollary 2

(a) The cost of a minimum cost mapping from $F_1[v]$ to $F_2[w]$ is

$$\textsc{Fdist}(v,w) = \min \left\{ \begin{array}{l} \textsc{Fdist}_F(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}), \\ \textsc{Fdist}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}). \end{array} \right\}$$

(b) The cost of a minimum cost mapping from $T_1[v]$ to $T_2[w]$ that includes the connection (v,w) is

$$\textsc{Tdist}_3(v,w) = \gamma(\textsc{label}(v) \rightarrow \textsc{label}(w)) + \textsc{Fdist}(v,w).$$

□

4.3 Computation of a Mapping

Up to now we have only shown how to compute the structure respecting distance between the two trees T_1 and T_2. But it is natural to ask for a mapping from T_1 to T_2 that yields this distance. It is easy to extend our algorithm so that it constructs such a mapping, too. For example, one may store at any time the combination of immediate results which yields the distance $\textsc{Tdist}(v,w)$ between the subtrees $T_1[v]$ and $T_2[w]$. Then one can reconstruct top-down a mapping that yields this distance within the same time bounds. We will not give the details here.

5 Implementation of the Algorithm

In this section we sketch an implementation of our algorithm *StructureRespectingDistance* for computing the structure resepcting distance between ordered trees.

In Figure 29 the algorithm is given in pseudo-code fashion. There we assume that the **input** consists of two labeled ordered trees $T_1 = (\{v_1, v_2, \ldots v_{n1}\}, E_1)$ and $T_2 = (\{w_1, w_2, \ldots w_{n2}\}, E_2)$. Furthermore, we assume that the nodes of both trees are numbered in left-to-right postorder and that there are for every node u pointers $\textsc{child}(u,i)$, $1 \leq i \leq \textsc{degree}(u)$, that point to the ith child of u (if a child is not present, the corresponding pointer has the value NIL). If this information is not available, some preprocessing is required. The **ouput** of the algorithm is the structure respecting distance $\Delta(T_1, T_2)$ between T_1 and T_2.

In the algorithm two global arrays TDIST and FDIST are used to hold the distances between the subtrees and the subforests of T_1 and T_2. In lines (1) to (3) these arrays are initialized by the distances to empty trees according to Fact 2. The main computation is done in lines (4) to (6). In the outer **for**-loop the nodes of T_1 are traversed in ascending left-to-right postorder, and in the inner **for**-loop the nodes of T_2 are traversed in ascending left-to-right postorder. In line (6) a procedure *ComputeDist* is called, which computes the values of TDIST and FDIST for its argument pair (v,w). We consider an implementation

```
program StructureRespectingDistance
    begin
(1)     FDIST(NIL, NIL) := 0;
        TDIST(NIL, NIL) := 0;
(2)     for v := v_1 to v_{n1} do
            FDIST(v, NIL) := Σ_{i=1}^{DEGREE(v)} TDIST(CHILD(v, i), NIL);
            TDIST(v, NIL) := γ(LABEL(v) → ε) + FDIST(v, NIL)
        od;
(3)     for w := w_1 to w_{n2} do
            FDIST(NIL, w) := Σ_{j=1}^{DEGREE(w)} TDIST(NIL, CHILD(w, j));
            TDIST(NIL, w) := γ(ε → LABEL(w)) + FDIST(NIL, w)
        od;
(4)     for v := v_1 to v_{n1} do
(5)         for w := w_1 to w_{n2} do
(6)             ComputeDist(v, w)
            od
        od;
(7)     output TDIST(ROOT(T_1), ROOT(T_2))
    end
```

Figure 29: *The algorithm for computing the structure respecting tree distance.*

of this procedure below. Finally in line (7) the computed distance between T_1 and T_2 is given out.

```
procedure ComputeDist(v, w)
  begin
    k_1 := DEGREE(v);
    k_2 := DEGREE(w);
(1) TDIST_1(v, w) := TDIST(NIL, w)+
        min_{1≤j≤k_2}{TDIST(v, CHILD(w, j)) − TDIST(NIL, CHILD(w, j))};
(2) TDIST_2(v, w) := TDIST(v, NIL)+
        min_{1≤i≤k_1}{TDIST(CHILD(v, i), w) − TDIST(CHILD(v, i), NIL)};
(3) for i := 1 to k_1 do
      for j := 1 to k_2 do
        FDIST_F(CHILD(v, 1) ... CHILD(v, i), CHILD(w, 1) ... CHILD(w, j)) := min{
          TDIST(CHILD(v, i), CHILD(w, j))+
        FDIST_F(CHILD(v, 1) ... CHILD(v, i − 1), CHILD(w, 1) ... CHILD(w, j − 1)),
          TDIST(CHILD(v, i), NIL)+
        FDIST_F(CHILD(v, 1) ... CHILD(v, i − 1), CHILD(w, 1) ... CHILD(w, j)),
          TDIST(NIL, CHILD(w, j))+
        FDIST_F(CHILD(v, 1) ... CHILD(v, i), CHILD(w, 1) ... CHILD(w, j − 1))}
        od
      od;
(4) FDIST_S(CHILD(v, 1) ... CHILD(v, k_1), CHILD(w, 1) ... CHILD(w, k_2)) := min{
      min_{1≤i≤k_1}{FDIST(v, NIL) − TDIST(CHILD(v, i), NIL) +
        γ(LABEL(CHILD(v, i)) → ε) + FDIST(v_i, w)},
      min_{1≤j≤k_2}{FDIST(NIL, w) − TDIST(NIL, CHILD(w, j) +
        γ(ε → LABEL(CHILD(w, j))) + FDIST(v, w_j)}};
(5) FDIST(v, w) := min{
        FDIST_F(CHILD(v, 1) ... CHILD(v, k_1), CHILD(w, 1) ... CHILD(w, k_2)),
        FDIST_S(CHILD(v, 1) ... CHILD(v, k_1), CHILD(w, 1) ... CHILD(w, k_2))};
(6) TDIST_3(v, w) := γ(LABEL(v) → LABEL(w)) + FDIST(v, w));
(7) TDIST(v, w) := min{TDIST_1(v, w), TDIST_2(v, w), TDIST_3(v, w)}
  end
```

Figure 30: *The procedure* ComputeDist.

Figure 30 sketches an implementation of the procedure *ComputeDist*. In lines (1) and (2) of this procedure, the cases **(1)** and **(2)**, respectively, of the computation of TDIST discussed in Subsection 4.1 are covered. In the **for**-loops of line (3) the minimum cost of a fork mapping from $F_1[v_1 \ldots v_{\text{DEGREE}(v)}]$ to $F_2[w_1 \ldots w_{\text{DEGREE}(w)}]$ is computed according to Lemma 9. In line (4) the minimum cost of a star mapping from $F_1[v_1 \ldots v_{\text{DEGREE}(v)}]$ to $F_2[w_1 \ldots w_{\text{DEGREE}(w)}]$ is computed according to Lemma 11. In line (5) $\text{FDIST}(v, w)$ is computed according to Corollary 2.(a). Then this value is used to compute $\text{TDIST}_3(v, w)$ in line (6) according to Corollary 2.(b). Finally, in line (7) $\text{TDIST}(v, w)$ is computed. Here

the three terms of which the minimum is computed correspond to the three cases of the computation of TDIST discussed in Subsection 4.1. Note that the values of $\text{TDIST}_1(v, w)$, $\text{TDIST}_2(v, w)$, $\text{TDIST}_3(v, w)$, $\text{FDIST}_F(v, w)$ and $\text{FDIST}_S(v, w)$ have to be stored only locally within this procedure.

6 Analysis of the Algorithm

6.1 Correctness of the Algorithm

Now we show the correctness of our algorithm. The following fact establishes a relation between structure respecting mappings between trees (forests) and structure respecting mappings between their subtrees (subforests).

Fact 3 *Let x be a node of $T_1[v]$ and let y be a node of $T_2[w]$. Then we have that*

(a) a structure respecting mapping from $T_1[x]$ to $T_2[y]$ also is a structure respecting mapping from $T_1[v]$ to $T_2[w]$;

(b) a structure respecting mapping from $F_1[x]$ to $F_2[y]$ also is a structure respecting mapping from $F_1[v]$ to $F_2[w]$.

PROOF.

(a) Going from $T_1[x]$ to $T_1[v]$ and from $T_2[y]$ to $T_2[w]$ does not change the lowest common ancestors of nodes of $T_1[x]$ and of $T_2[y]$.

(b) Analogously. □

The following lemma says that the algorithm only considers structure respecting mappings.

Lemma 12 *When computing the distance between T_1 and T_2 the algorithm* StructureRespectingDistance *only considers structure respecting mappings from T_1 to T_2.*
PROOF (by induction on the calls of the procedure *ComputeDist*). We show that the distance between the subtrees $T_1[v]$ and $T_2[w]$ computed by *ComputeDist* relies solely on structure respecting mappings.

(i) In the first call of *ComputeDist* v and w are the leftmost leaves of T_1 and T_2, respectively, so that no mappings between subtrees are used. Furthermore, mappings between single nodes are obviously structure respecting.

(ii) The subtree distances used in the further calls of *ComputeDist* rely on structure respecting mappings by induction hypothesis. What remains to show is that their combinations lead to structure respecting mappings from $T_1[v]$ to $T_2[w]$. To do so, we look at the cases of the computation of $\text{TDIST}(v, w)$ one-by-one.

In **Case (1)** and **Case (2)** only one subtree distance is used. Hence the resulting mapping from $T_1[v]$ to $T_2[w]$ is structure respecting according to Fact 3.(a).

In **Case (3)** we consider the fork mappings first.

Let M be a fork mapping from $F_1[v]$ to $F_2[w]$. Then M is composed of structure respecting submappings between pairs of single subtrees of $F_1[v]$ and $F_2[w]$. What remains to show is that the union of these mappings is also structure respecting.

Let v have the children v_i, $1 \leq ik_1$ and w the children w_j, $1 \leq j \leq k_2$. Let M_1 be a submapping of M from $T_1[v_{i_1}]$ to $T_2[w_{j_1}]$, let M_2 be the submapping of M from $T_1[v_{i_2}]$ to $T_2[w_{j_2}]$ and let M_3 be the submapping of M from $T_1[v_{i_3}]$ to $T_2[w_{j_3}]$, where $1 \leq i_1 < i_2 < i_3 \leq k_1$ and $1 \leq j_1 < j_2 < j_3 \leq k_2$. We now show that all triples (x_1, y_1), (x_2, y_2), (x_3, y_3) of connections of $\hat{M} = M_1 \cup M_2 \cup M_3$ do not violate the property structure respecting. The three connections can be distributed as follows.

(1) *The three connections are in the same submapping M_i, $1 \leq i \leq 3$.* Then the three connections do not violate the property structure respecting because each M_i is a structure respecting mapping.

(2) *Only two connections are in the same submapping.* Let $(x_1, y_1), (x_2, y_2)$ be in M_1 and (x_3, y_3) be in M_2. Then we have

$$\text{LCA}(x_1, x_2) \neq \text{LCA}(x_1, x_3) \quad \text{and} \quad \text{LCA}(y_1, y_2) \neq \text{LCA}(y_1, y_3),$$

$$\text{LCA}(x_2, x_1) \neq \text{LCA}(x_2, x_3) \quad \text{and} \quad \text{LCA}(y_2, y_1) \neq \text{LCA}(y_2, y_3),$$

and

$$\text{LCA}(x_3, x_1) = v = \text{LCA}(x_3, x_2) \quad \text{and} \quad \text{LCA}(y_3, y_1) = w = \text{LCA}(y_3, y_2),$$

so these connections do not violate the property structure respecting. An analogous argumentation applies if (x_1, y_1) is in M_1 and $(x_2, y_2), (x_3, y_3)$ are in M_2.

(3) *Each connection is in a different submapping.* Let (x_1, y_1) be in M_1, (x_2, y_2) be in M_2, and (x_3, y_3) be in M_3. Then we have

$$\text{LCA}(x_1, x_2) = v = \text{LCA}(x_1, x_3) \quad \text{and} \quad \text{LCA}(y_1, y_2) = w = \text{LCA}(y_1, y_3),$$

$$\text{LCA}(x_2, x_1) = v = \text{LCA}(x_2, x_3) \quad \text{and} \quad \text{LCA}(y_2, y_1) = w = \text{LCA}(y_2, y_3),$$

and

$$\text{LCA}(x_3, x_1) = v = \text{LCA}(x_3, x_2) \quad \text{and} \quad \text{LCA}(y_3, y_1) = w = \text{LCA}(y_3, y_2),$$

so these connections do not violate the property structure respecting either.

In the case of star mappings only one subforest distance is used. Hence the resulting mapping from $F_1[v]$ to $F_2[w]$ is structure respecting according to Fact 3.(b). □

Now we have to show that the algorithm considers all eligible mappings. This is done by the following lemma. But first we show that the required subtree distances are available at any time.

Fact 4 *When* ComputeDist(v, w) *is called, the required subtree and subforest distances between $T_1[v]$ and $T_2[w]$ are available.*

PROOF. Immediately from the fact that the algorithm considers the nodes of each tree in ascending left-to-right postorder. □

Lemma 13 *When computing the distance between T_1 and T_2 the algorithm* StructureRespectingDistance *considers all structure respecting mappings from T_1 to T_2 that can have minimal cost.*

PROOF (by induction on the calls of the procedure *ComputeDist*). We show that the procedure *ComputeDist* considers all structure respecting mappings that can have minimal cost when computing the distance between the subtrees $T_1[v]$ and $T_2[w]$ and the distance between the subforests $F_1[v]$ and $F_2[w]$.

(i) In the first call of *ComputeDist* v and w are the leftmost leaves T_1 and T_2, respectively. The only not expandable mapping from $T_1[v]$ to $T_2[w]$ consists solely of the connection (v, w). The only mapping from $F_1[v]$ to $F_2[w]$ is the empty mapping. Both mappings are obviously considered.

(ii) In the following calls of *ComputeDist* the required subtree and subforest distances are available according to Fact 4. By induction hypothesis all structure respecting mappings that can have minimal cost have been considered in their computation. It remains to show that when computing $\text{TDIST}(v, w)$ and $\text{FDIST}(v, w)$, respectively, all of their combinations in question are considered.

By Lemma 2 we have that in the computation of $\text{TDIST}(v, w)$ only the cases considered in *ComputeDist* have to be considered. The correct treatment of **Case (1)** and **(2)** follows immediately from Lemma 3 and Lemma 4, respectively.
In **Case (3)** Lemma 5, Lemma 6, and Corollary 1 reduce the number of the combinations to be considered. The correct treatment of the remaining combinations follows from Lemma 9 and Lemma 11. □

Now the correctness of our algorithm follows immediately.

Theorem 4 *The algorithm* StructureRespectingDistance *computes the structure respecting distance between two labeled ordered trees.* □

6.2 Complexity of the Algorithm

Finally we consider the time and space complexity of our algorithm.

Theorem 5 *The time complexity of the algorithm* StructureRespectingDistance *is*

$$O(\text{DEGREE}(T_1) \cdot \text{DEGREE}(T_2) \cdot |T_1| \cdot |T_2|).$$

PROOF. In the algorithm the procedure *ComputeDist* is called for every pair (v, w) of nodes from $T_1 \times T_2$ exactly once. The most expensive part of *ComputeDist* is the computation of the minimum cost of a fork mapping from $F_1[v]$ to $F_2[w]$, which takes $O(\text{DEGREE}(v) \cdot \text{DEGREE}(w))$ time. □

For analyzing the space complexity of our algorithm, we assume that the algorithm traverses in an outer loop the nodes of T_1 and in an inner loop the nodes of T_2 in ascending left-to-right postorder. This means the algorithm computes for a node v of T_1 consecutively the distances $\text{TDIST}(v, w)$ and $\text{FDIST}(v, w)$ for all nodes w of T_2.

Lemma 14 *The distances* $\text{TDIST}(v, w)$ *and* $\text{FDIST}(v, w)$ *have to be stored only until the algorithm has computed the distances* $\text{TDIST}(\text{PARENT}(v), \text{PARENT}(w))$ *and* $\text{FDIST}(\text{PARENT}(v), \text{PARENT}(w))$.

PROOF. The distances $\text{TDIST}(v, w)$ and $\text{FDIST}(v, w)$ are only required in the computation of the distances $\text{TDIST}(v, \text{PARENT}(w))$ and $\text{FDIST}(v, \text{PARENT}(w))$, $\text{TDIST}(\text{PARENT}(v), w)$ and $\text{FDIST}(\text{PARENT}(v), w)$, $\text{TDIST}(\text{PARENT}(v), \text{PARENT}(w))$ and $\text{FDIST}(\text{PARENT}(v), \text{PARENT}(w))$. Among these, the latter two distances are computed last. □

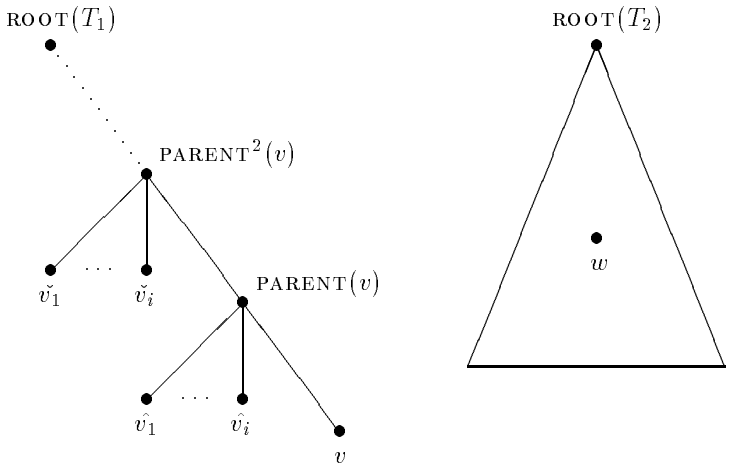

Figure 31: *The situation of Theorem 6.*

Theorem 6 *The space complexity of the algorithm* StructureRespectingDistance *is*

$$O(\text{DEGREE}(T_1) \cdot \text{DEPTH}(T_1) \cdot |T_2|).$$

PROOF. We consider a run of the outer loop for a node v of T_1. Within this run, the inner loop traverses all nodes w of T_2 and uses the distances $\text{TDIST}(v_1, w)$ to $\text{TDIST}(v_k, w)$ and $\text{FDIST}(v_1, w)$ to $\text{FDIST}(v_k, w)$ for all w in T_2, where v_1 to v_k are the children of v. During this run the algorithm computes the distances $\text{TDIST}(v, w)$ and $\text{FDIST}(v, w)$ for all w in T_2. Hence, the algorithm computes $O(|T_2|)$ distances within this run

If v has the left siblings $\hat{v}_1, \ldots \hat{v}_i$, then the algorithm uses in the later run of the outer loop for the parent of v the distances $\text{TDIST}(\hat{v}_1, w)$ to $\text{TDIST}(\hat{v}_i, w)$ and $\text{FDIST}(\hat{v}_1, w)$ to $\text{FDIST}(\hat{v}_i)$ for all w in T_2, which have been computed previously. If the parent of v has the left siblings $\check{v}_1, \ldots \check{v}_i$, then the algorithm uses in the later run of the outer loop for the parent of the parent of v the distances $\text{TDIST}(\check{v}_1, w)$ to $\text{TDIST}(\check{v}_i, w)$ and $\text{FDIST}(\check{v}_1, w)$ to $\text{FDIST}(\check{v}_i)$ for all w in T_2, and so on, as illustrated in Figure 31.

Altogether the algorithm has to store for later runs of the outer loop at most $O(\text{DEGREE}(T_1) \cdot \text{DEPTH}(v))$ distances to all subtrees of T_2. □

Note that to meet this space complexity in a real implementation of the algorithm, one cannot use global arrays to hold the values of TDIST and FDIST. Instead, one has

to use temporary structures to hold the values of $\text{TDIST}(v, w)$ and $\text{FDIST}(v, w)$ for every pair (v, w). These structures have to be created when the pair (v, w) is considered by the algorithm, and they can be deleted as soon as the pair $(\text{PARENT}(v), \text{PARENT}(w))$ has been considered.

Note further that, if we want to reconstruct a mapping that yields the distance computed, we have to store the combination of immediate results which yields the computed distance for every pair of subtrees, so that we get a space complexity of $O(|T_1| \cdot |T_2|)$.

7 Approximate Tree Matching

This section is devoted to approximate tree matching. Approximate tree matching is a generalization of approximate string matching which is closely related to tree editing. Actually, it can be seen as an application of tree distance measures.

In the the following we first review approximate string and tree matching. Then we show how the concecpts used in these problems can be transformed to the structure respecting distance between ordered trees.

7.1 Approximate String Matching

Approximate string matching was first introduced by Sellers [10]. In this problem a pattern string P and a target string S are given. Then one looks for the substrings of S which most resemble P, where resemblance is defined in terms of string distance (see [8], [9] and [16]). Sellers defines two variants of this problem that differ in the exact definition of resemblance.

In the first variant, which we call *global approximate string matching*, one looks for the substrings of S whose distance to the pattern string is minimum among all substrings. Formally, this global resemblance is defined as follows.

Definition 11 (Global resemblance) *A substring $S[i_1 \ldots i_2]$, $1 \leq i_1 \leq i_2 \leq |S|$, of the target string* most resembles *the pattern string P* globally, *if*

$$\delta_S(P, S[i_1 \ldots i_2]) \leq \delta_S(P, S[j_1 \ldots j_2])$$

for all $1 \leq j_1 \leq j_2 \leq |S|$, where δ_S denotes the string distance. □

In [10] an algorithm is given that solves this problem in time $O(|P| \cdot |T|)$. The essential difference between this algorithm and the standard dynamic programming algorithm for the string editing problem is that the former allows the removal of a prefix of the target string without any cost.

A version of this problem is the *k differences problem* (see [15], [3], [7], for example). Here one asks whether there is a substring of the target string whose distance to the pattern string is at most k. Clearly one can solve this problem with Sellers' algorithm, because this algorithm computes the distance between the pattern string and every substring of the target string. In the following, some algorithms have been developed that improve the quadratic time bound of Sellers' algorithm (see [3] for a survey and further references).

Note that these global approximate string matching problems can also be considered as generalizations of the classic string matching problem (see [1] and [6] for the classic

algorithms).

In the second variant of Sellers, which we call *local approximate string matching*, one looks for the substrings of S whose distance to the pattern string is minimum among their sub- and superstrings. Formally, this local resemblance is defined as follows.

Definition 12 (Local resemblance) *A substring* $S[i_1 \ldots i_2], 1 \leq i_1 \leq i_2 \leq |S|$ *of the target string* most resembles *the pattern string* P locally, *if*

$$\delta_S(P, S[i_1 \ldots i_2]) \leq \delta_S(P, S[h_1 \ldots h_2])$$

for all $i_1 \leq h_1 \leq h_2 \leq i_2$ *and*

$$\delta_S(P, S[i_1 \ldots i_2]) \leq \delta_S(P, S[j_1 \ldots j_2])$$

for all $j_1 \leq i_1 \leq i_2 \leq j_2$. □

In [10] an algorithm is given that solves this problem in time $O(|P| \cdot |T|)$, too. The main difference from the algorithm for the global case is that the former needs two distance matrices to cancel other substrings which overlap from the left and from the right, respectively.

7.2 Approximate Tree Matching in the General Case

As well as approximate string matching, which is based on the string editing problem, one can define approximate tree matching, which is based on the tree editing problem. This is done by Zhang and Shasha in [18] as follows.

The removal of a prefix of a string is generalized by the removal of a collection of subtrees. To do so, two new operations at a node are introduced.

- *Removing* at a node w means removing the entire subtree $T[w]$;
- *Pruning* at a node w means removing all descendants of w, i. e. removing the subforest $F[w]$.

Note that removing is the more general operation, as pruning can be done by removing at all children of w. On the other hand, pruning never eliminates the entire tree. Note further that both operations reduce to the removal of a prefix of a string, when the string is considered in "postorder", i. e. in reverse order.

A *subtree set* $\text{STS}(T)$ of a tree T is a subset of its nodes such that for all $v_1, v_2 \in \text{STS}(T)$ neither is an ancestor of the other. $R(T, \text{STS}(T))$ is defined to be the tree T with removing at all nodes in $\text{STS}(T)$, and $P(T, \text{STS}(T))$ is defined to be the tree T with pruning at all nodes in $\text{STS}(T)$.

Now one can give two variants of approximate tree matching. Given a pattern tree P and a target tree T, one wants to compute either

$$\text{RDIST}(P, T[w]) = \min_{\text{STS subtree set}} \{\delta_T(P, R(T[w], \text{STS}(T[w])))\}$$

or

$$\textsc{Pdist}(P, T[w]) = \min_{\text{STS subtree set}} \{\delta_T(P, P(T[w], \textsc{sts}(T[w])))\}$$

for all $w \in T$, where δ_T denotes the general distance between two trees. We call RDIST the *removing distance* and PDIST the *pruning distance*. $\textsc{Rdist}(P, T[w])$ and $\textsc{Pdist}(P, T[w])$, respectively, is the distance between the pattern tree and that subtree of $T[w]$ which most resembles the pattern tree. By minimizing over all $w \in T$ one gets the distance between the pattern tree and that subtree of the target tree, that most resembles the pattern tree.

In [18] Zhang and Shasha present algorithms for computing the removing distance and the pruning distance, respectively, that are based on their algorithm for the general distance between trees, and that also have a running time of $O(|P| \cdot |T| \cdot \min(\textsc{depth}(P), \textsc{leaves}(P)) \cdot \min(\textsc{depth}(T), \textsc{leaves}(T)))$.

Finally, note that the removing distance as well as the pruning distance corresponds to the global resemblance of substrings. Hence they can also be seen as generalizations of tree pattern matching (see [4], [5] and [2]).

7.3 The Structure Respecting Removing Distance

In this subsection we apply the structure respecting distance between two ordered trees to approximate tree matching, where we restrict ourselves to the removing distance. Formally we can define this problem as follows.

Definition 13 (Structure respecting removing distance) *Let P and T be two labeled ordered trees. P is called the* pattern tree *and T the* target tree*. Then the task is to compute*

$$\textsc{RTdist}(P, T[w]) = \min_{\text{STS subtree set}} \{\Delta(P, R(T[w], \textsc{sts}(T[w])))\}$$

for all $w \in T$, where Δ denotes the structure respecting distance between two ordered trees. We call RTDIST *the* structure respecting removing distance. □

If we minimize the structure respecting removing distance over all $w \in T$, we get the subtrees of the target tree that most resemble the pattern tree globally. Furthermore, we can define local resemblance between trees on the basis of the structure respecting removing distance as follows.

Definition 14 (Local resemblance between trees) *A subtree $T[w]$ of the target tree T* most resembles *the pattern tree P* locally*, if for all subtrees $T[w_{sub}]$ of $T[w]$*

$$\textsc{RTdist}(P, T[w]) \leq \textsc{RTdist}(P, T[w_{sub}])$$

and for all supertrees $T[w_{super}]$ of $T[w]$

$$\textsc{RTdist}(P, T[w]) \leq \textsc{RTdist}(P, T[w_{super}])$$

holds. □

Note that this definition applies to the general case as well. But it does not reduce to local resemblance between strings when the strings are considered in reverse order as trees. This is because every suffix of a string is either a "subtree" or a "supertree" of every other suffix. Nevertheless, we think that our definition of local resemblance between trees is a natural generalization of the concept of local resemblance to trees.

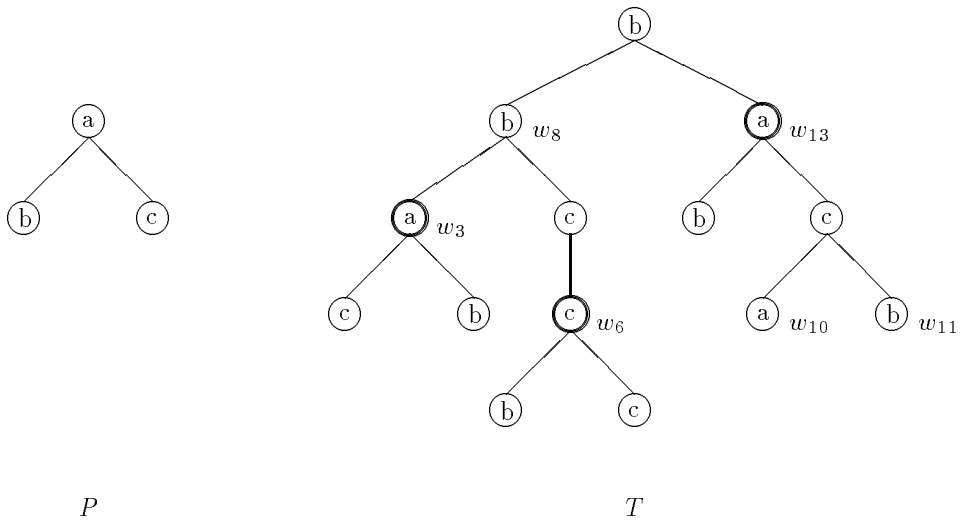

Figure 32: *Global and local resemblance between trees.*

Example 4 *As an example, consider the pattern tree P and the target tree T shown in Figure 32. Here $T[w_{13}]$ most resembles the pattern tree globally, because $T[w_{10}]$ and $T[w_{11}]$ can be removed without cost, so that its distance to the pattern tree is zero. $T[w_{13}]$ also most resembles the pattern tree locally because its distance is the only absolute minimum. This means that neither a subtree nor a supertree of $T[w_{13}]$ can most resemble the pattern tree locally. Hence the other subtrees of the target tree that most resemble the pattern tree locally must be subtrees of $T[w_8]$. If we define $\gamma(a \to b)$ to be one, if $a \neq b$, and zero, if $a = b$, then we have that $T[w_3]$ and $T[w_6]$ are the other subtrees of the target tree that most resemble the pattern tree locally.* □

7.4 Computation of the Removing Distance

Now we look at the computation of the structure respecting removing distance. As our algorithm for computing the structure respecting distance also computes the distance between all pairs of subtrees, we can use it as a template and change it only where necessary.

In the computation of RTDIST(v, w) in the procedure *ComputeDist* we have to take into account that any subtree of the target tree T may be removed without any cost. To cope, we consider each of the three cases of the computation of TDIST discussed in Subsection 4.2 in turn. Again we assume that v has the children v_i, $1 \leq i \leq k_1$, and w the children w_j, $1 \leq j \leq k_2$.

Case (1). $v \in D_M(P[v]), w \notin R_M(T[w])$. In this case the node v, but not the node w, is covered by the mapping M. This means that the node v has to be connected to a node in a subtree $T[w_j]$, $1 \leq j \leq k_2$, of $T[w]$. The other subtrees of $T[w]$ may be removed without any cost. Note that we have to insert the node w separately. Hence the formula of Lemma

3 changes to

$$\gamma(M) = \gamma(\varepsilon \to \text{LABEL}(w)) + \min_{1 \le j \le k_2} \{\text{RTDIST}(v, w_j)\}.$$

Case (2). $v \notin D_M(P[v]), w \in R_M(T[w])$. In this case the node w, but not the node v, is covered by the mapping M. This means that the node w has to be connected to a node in a subtree $P[v_i]$, $1 \le i \le k_1$, of $P[v]$. The other subtrees of $P[v]$ have to be deleted. Since possible removings of subtrees of $T[w]$ have already been considered in the computation of the subdistances, the formula of Lemma 4 remains unchanged:

$$\gamma(M) = \text{RTDIST}(v, \text{NIL}) + \min_{1 \le i \le k_1} \{\text{RTDIST}(v_i, w) - \text{RTDIST}(v_i, \text{NIL})\}.$$

Case (3). $(v, w) \in M$. In this case the node v, as well as the node w, is covered by the mapping M. Hence all mappings which have to be considered include the connection (v, w). We denote the minimum cost of such a mapping from $P[v]$ to $T[w]$ by $\text{RTDIST}_3(v, w)$. This distance is composed of the cost $\gamma(\text{LABEL}(v_i) \to \text{LABEL}(w_j))$ of the connection (v, w) and the distance $\text{RFDIST}(v, w)$ between the forests $F_P[v_1 \dots v_{k_1}]$ and $F_T[w_1 \dots w_{k_2}]$. In its computation we now have to take into account that the subtrees of w may be removed without cost.

To compute $\text{RFDIST}(v_1 \dots v_{k_1}, w_1 \dots w_{k_2})$ we again consider the fork mappings from $F_P[v_1 \dots v_{k_1}]$ to $F_T[w_1 \dots w_{k_2}]$ first. The minimum cost of such a mapping is denoted by $\text{RFDIST}_F(v_1 \dots v_{k_1}, w_1 \dots w_{k_2})$. In its computation we consider the subtrees of $P[v]$ and $T[w]$ again from the left to the right. Here the following lemma corresponds to Lemma 9 of Subsection 4.2.

Lemma 15 *The cost of a minimum cost fork mapping from $F_1[v_1 \dots v_i]$, $1 \le i \le k_1$ to $F_2[w_1 \dots w_j]$, $1 \le j \le k_2$, is*

$$\begin{aligned}\text{RFDIST}_F(v_1 \dots v_i, w_1 \dots w_j) = \min\{ & \\ & \text{RTDIST}(v_i, w_j) + \text{RFDIST}_F(v_1 \dots v_{i-1}, w_1 \dots w_{j-1}), \\ & \text{RTDIST}(v_i, \text{NIL}) + \text{RFDIST}_F(v_1 \dots v_{i-1}, w_1 \dots w_j), \\ & \text{RTDIST}(v_i, \text{NIL}) + \text{RFDIST}_F(v_1 \dots v_{i-1}, w_1 \dots w_{j-1}), \\ & \text{RFDIST}_F(v_1 \dots v_i, w_1 \dots w_{j-1})\}.\end{aligned}$$

PROOF. First, $\text{RTDIST}(v_i, w_j) \le \text{RTDIST}(v_i, \text{NIL}) + \text{RTDIST}(\text{NIL}, w_j)$ may not hold here, because $\text{RTDIST}(\text{NIL}, w_j)$ is zero and it may be cheaper to delete a subtree than to map it to another subtree. Hence we have to distinguish four cases here.

(a) *$P[v_i]$ and $T[w_j]$ are both covered by the mapping.* Then such a mapping M with minimum cost has the cost

$$\text{RTDIST}(v_i, w_j) + \text{RFDIST}_F(v_1 \dots v_{i-1}, w_1 \dots w_{j-1}),$$

as in Lemma 9.

(b) *Only $P[v_i]$ is covered by the mapping.* Then $T[w_j]$ can be removed without any cost. Hence we can cover this case by the cost

$$\text{RFDIST}_F(v_1 \dots v_i, w_1 \dots w_{j-1}).$$

(c) *Only $T[w_j]$ is covered by the mapping.* Analogously to Lemma 9, this case is covered by the cost

$$\text{RTDIST}(v_i, \text{NIL}) + \text{RFDIST}_F(v_1 \ldots v_{i-1}, w_1 \ldots w_j).$$

(d) *Neither of $P[v_i]$ and $T[w_j]$ is covered by the mapping.* Since $T[w_j]$ can be removed without any cost, we effectively combine a fork mapping from $F_P[v_1 \ldots v_{i-1}]$ to $F_T[w_1 \ldots w_{j-1}]$ with the deletion of $P[v_i]$ in this case. Hence this case is covered by the cost

$$\text{RTDIST}(v_i, \text{NIL}) + \text{RFDIST}_F(v_1 \ldots v_{i-1}, w_1 \ldots w_{j-1}).$$

□

Now we consider the star mappings from $F_P[v_1 \ldots v_{k_1}]$ to $F_T[w_1 \ldots w_{k_2}]$. The minimum cost of such a mapping is denoted by $\text{RFDIST}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2})$.

In the case of a left star mapping, we can adopt the formula in line (4.4) of Lemma 11 in Subsection 4.2:

$$\min_{1 \le i \le k_1} \{\text{RFDIST}(v, \text{NIL}) - \text{RTDIST}(v_i, \text{NIL}) + \gamma(\text{LABEL}(v_i) \to \varepsilon) + \text{RFDIST}(v_i, w)\},$$

since possible removings of subtrees of $T[w]$ have already been considered in the computation of the subdistances $\text{RFDIST}(v_i, w)$.

In the case of right star mappings, we have to take into account that subtrees of $T[w]$ may be removed without cost. Hence the formula in line (4.5) of Lemma 11 changes to

$$\min_{1 \le j \le k_2} \{\gamma(\varepsilon \to \text{LABEL}(w_j)) + \text{RFDIST}(v, w_j)\}.$$

Summarizing, we have the following result for the computation of $\text{RFDIST}_S(v_1 \ldots v_i, w_1 \ldots w_j)$.

Corollary 3 *The cost of a minimum cost star mapping from $F_P[v_1 \ldots v_{k_1}]$ to $F_T[w_1 \ldots w_{k_2}]$ is*

$$\begin{aligned}\text{RFDIST}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}) = \min\{ & \\ & \min_{1 \le i \le k_1}\{\text{RFDIST}(v, \text{NIL}) - \text{RTDIST}(v_i, \text{NIL}) + \\ & \quad \gamma(\text{LABEL}(v_i) \to \varepsilon) + \text{RFDIST}(v_i, w)\}, \\ & \min_{1 \le j \le k_2}\{\gamma(\varepsilon \to \text{LABEL}(w_j)) + \text{FDIST}(v, w_j)\}\}.\end{aligned}$$

□

Further summarizing, we get the following results for Case **(3)**.

Corollary 4

(a) The cost of a minimum cost structure respecting mapping from $F_P[v]$ to $F_T[w]$ is

$$\text{RFDIST}(v, w) = \min \left\{ \begin{array}{l} \text{RFDIST}_F(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}), \\ \text{RFDIST}_S(v_1 \ldots v_{k_1}, w_1 \ldots w_{k_2}). \end{array} \right\}$$

(b) The cost of a minimum cost structure respecting mapping from $P[v]$ to $T[w]$ that includes the connection (v, w) is

$$\text{RTDIST}_3(v, w) = \gamma(\text{LABEL}(v) \to \text{LABEL}(w)) + \text{RFDIST}(v, w).$$

□

Altogether we have the following result for the computation of $\text{RTDIST}(v, w)$.

Corollary 5 *For the structure respecting removing distance between $P[v]$ and $T[w]$ it holds that*

$$\begin{aligned}\text{RTDIST}(v, w) = \min\{ & \\ & \gamma(\varepsilon \to \text{LABEL}(w)) + \min_{1 \le j \le k_2}\{\text{RTDIST}(v, w_j)\}, \\ & \text{RTDIST}(v, \text{NIL}) + \min_{1 \le i \le k_1}\{\text{RTDIST}(v_i, w) - \text{RTDIST}(v_i, \text{NIL})\}, \\ & \text{RTDIST}_3(v, w)\}.\end{aligned}$$

□

Furthermore, we do not have to initialize the RFDIST and the RTDIST array with distances between subforests and subtrees of the target tree T and empty subtrees, as these values are not used anymore. The other initializations remain unchanged.

Since we have used our algorithm for the structure respecting distance as a template, it should be clear that the resulting algorithm for the structure respecting removing distance is correct and runs within the same time bounds. Hence we have the following result.

Corollary 6 *The structure respecting removing distance between two labeled ordered trees P and T can be computed in time $O(\text{DEGREE}(P) \cdot \text{DEGREE}(T) \cdot |P| \cdot |T|)$.* □

Having computed the structure respecting removing distance, we can determine the subtrees of the target tree which most resemble the pattern tree globally by minimizing the removing distance over all $w \in T$ and choosing those subtrees of T which yield this minimum.

To determine the subtrees of the target tree which most resemble the pattern tree locally, a little more work is necessary. For every node $w \in T$ we compute

$$\text{RTDIST}_{sub}(w) :=$$

$$\begin{cases} \infty, & \text{if } w \text{ is a leaf,} \\ \min\{\text{RTDIST}(P, T[\tilde{w}] \mid \tilde{w} \text{ is a proper descendant of } w\}, & \text{otherwise;} \end{cases}$$

and

$$\text{RTDIST}^{super}(w) :=$$

$$\begin{cases} \infty, & \text{if } w \text{ is the root of } T, \\ \min\{\text{RTDIST}(P, T[\tilde{w}] \mid \tilde{w} \text{ is a proper ancestor of } w\}, & \text{otherwise.} \end{cases}$$

The values of $\text{RTDIST}_{sub}(w)$ and $\text{RTDIST}^{super}(w)$ can be computed by traversing the target tree in postorder and reverse postorder, respectively.

Then $T[w]$ is a subtree of T that most resembles the pattern tree P locally, iff

$$\text{RTDIST}(P, T[w]) \leq \text{RTDIST}_{sub}(w)$$

and

$$\text{RTDIST}(P, T[w]) \leq \text{RTDIST}^{super}(w).$$

Altogether we have the following result.

Theorem 7 *Let P and T be two labeled ordered trees. Then the subtrees of T which most resemble P globally or locally with respect to the structure respecting removing distance can be determined in time $O(\text{DEGREE}(P) \cdot \text{DEGREE}(T) \cdot |P| \cdot |T|)$.* □

8 Conclusion and Further Work

We have defined a new measure of the distance between two labeled ordered trees, which we have called *structure respecting distance*. It preserves more of the structure of the trees involved. This can be useful when one compares the structure of the parse trees of (natural language) sentences to determine their similarity. Then we have presented a simple dynamic programming algorithm that computes this new distance efficiently. The computation of this new distance requires less time and space than the computation of the general distance. Hence we think that our restriction of the general distance is justified. Finally we have applied the new measure of distance to approximate tree matching, where we have introduced a concept of local resemblance between trees. Next we would like to give a parallel implementation of our algorithms.

Finally we would like to improve the time complexity of the general tree editing problem. By comparing our algorithm for the structure respecting distance with the algorithm of Zhang and Shasha for the general distance, one notices that our algorithm computes the subtree distance for every pair of subtrees separately, whereas the algorithm of Zhang and Shasha computes the subtree distance only for certain pairs of subtrees separately. On the other hand, our algorithm computes the distance between the subtrees $T[v]$ and $T'[w]$ in time $O(\text{DEGREE}(v) \cdot \text{DEGREE}(w))$, whereas the algorithm of Zhang and Shasha needs time $O(|T[v]| \cdot |T'[w]|)$ to compute the corresponding subtree distance. Maybe it is possible to combine the advantages of both algorithms in a new algorithm for the general tree editing problem.

Acknowledgement. I would like to thank Prof. Dr. N. Blum for helpful discussions on this work.

References

[1] R. S. Boyer and J. S. Moore, *A Fast String Matching Algorithm*, CACM **20** (1977), pp. 262 - 272.

[2] M. Dubiner, Z. Galil and E. Magen, *Faster Tree Pattern Matching*, Proc. 31st FOCS (1990), pp. 145 - 150.

[3] Z. Galil and R. Giancarlo, *Data Structures and Algorithms for Approximate String Matching*, J. Complexity **4** (1988), pp. 33 - 72.

[4] C. M. Hoffmann and M. J. O'Donnell, *Pattern Matching in Trees*, JACM **29** (1982), pp. 68 - 95.

[5] S. R. Kosaraju, *Efficient Tree Pattern Matching*, Proc. 30th FOCS (1989), pp. 178 - 183.

[6] D. E. Knuth, J. H. Morris and V. R. Pratt, *Fast Pattern Matching in Strings*, SIAM J. Comput. **6** (1977), pp. 323 - 350.

[7] G. M. Landau and U. Vishkin, *Fast Parallel and Serial Approximate String Matching*, J. Algorithms **10** (1989), pp. 157 - 169.

[8] V. I. Levenshtein, *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Phys. Dokl. **6** (1966), pp. 707 - 710.

[9] P. H. Sellers, *On the Theory and Computation of Evolutionary Distances*, SIAM J. Appl. Math. **26** (1974), pp. 787 - 793.

[10] P. H. Sellers, *The Theory and Computation of Evolutionary Distances: Pattern Recognition*, J. Algorithms **1** (1980), pp. 359 - 373.

[11] B. A. Shapiro, *An Algorithm for comparing Multiple RNA Secondary Structures*, Comput. Appl. Biosci. **4** (1988), pp. 387 -393.

[12] B. A. Shapiro and K. Zhang, *Comparing Multiple RNA Secondary Structures Using Tree Comparisons*, Comput. Appl. Biosci. **6** (1990), pp. 309 - 318.

[13] K.-C. Tai, *Syntactic Error Correction in Programming Languages*, Ph. D. Th., Dept. Comptr. Sci., Cornell U., Ithaca, N.Y., 1977.

[14] K.-C. Tai, *The Tree-to-Tree Correction Problem*, JACM **26** (1979), pp. 422 - 433.

[15] E. Ukkonen, *Finding Approximate Pattern in Strings*, J. Algorithms **6** (1985), pp. 132 - 137.

[16] R. A. Wagner and M. J. Fischer, *The String-to-String Correction Problem*, JACM **21** (1974), pp. 168 - 173.

[17] T. Winograd, *Language as a Cognitive Process*, Vol. 1: *Syntax*, Chapter 3, Addison-Weslwy, Reading, MA, 1983, pp. 72 - 132.

[18] K. Zhang and D. Shasha, *Simple Fast Algorithms for the Editing Distance between Trees and Related Problems*, SIAM J. Comput. **18** (1989), pp. 1245 - 1262.

[19] K. Zhang, R. Statman and D. Shasha, *On the editing distance between unordered labeled trees*, IPL **42** (1992), pp. 133 - 139.