Ordered Tree Edit Distance with Merge and Split Operations

Philip Bille

IT University Technical Report Series **TR-2003-35**

ISSN 1600–6100 **September 2003**

ISSN 1600–6100

ISBN 87-7949-048-4

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone:	**+45 38 16 88 88**
Telefax:	**+45 38 16 88 99**
Web	**`www.it-c.dk`**

Ordered Tree Edit Distance with Merge and Split Operations

Philip Bille*

September 29, 2003

Abstract

Comparing trees is a fundamental problem in computer science. In particular, the ordered tree edit distance problem, defined as the problem of comparing ordered and labeled trees based on the cost and number of edit operations needed to transform a tree T_1 into another tree T_2, arise in many applications. For the simple edit operations of inserting, deleting and relabeling a node the problem is a well-studied problem and algorithms with $o(n^4)$ time complexity exists. In this paper we extend the set of operations with merge and split operations. We argue that this new problem naturally generalize the old problem and we provide polynomial time algorithms for solving it.

1 Introduction

Comparing trees is a fundamental problem in computer science in various areas such as computational biology, structured text databases, image analysis, automatic theorem proving and compiler optimization [Tai79, ZS89, KM95, KTSK00, HO82, RR92, ZSW94]. In particular, the *tree edit distance problem* — the problem of comparing trees based on the cost and number of simple local operations needed to transform a tree T_1 into another tree T_2 — has be studied extensively [Sel77, Tai79, ZS89, ZSS92, ZJ94, Zha95, Zha96, Kle98, KTSK00, LST01, Che01].

Let T be a rooted tree. We call T a *labeled tree* if each node is a assigned a symbol from a fixed finite alphabet Σ. We call T an *ordered tree* if a left-to-right order among siblings in T is given. In this paper we consider edit distance problems based on simple primitive operations applied to rooted, ordered and labeled trees. The operations are defined below. We assume that all of the operations preserve the left-to-right order, that is unless otherwise stated, if v is to the left of w before an operation then v will also be to the left of w after the operation, for any pair of nodes v and w in T.

relabel Change the label of a node v in T.

delete Delete a non-root node v in T, making the children of v become the children of the parent of v.

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

horizontal-merge Merge a consecutive subsequence of siblings $v_1, \dots, v_s$ into a single node v. The children of $v_1, \dots, v_s$ become the children of v.

horizontal-split The complement of horizontal-merge. Split a node v into a consecutive sequence of siblings $v_1, \dots, v_s$. The children of v become children of $v_1, \dots, v_s$.

*The IT University of Copenhagen, Glentevej 67, DK-2400 Copenhagen NV, Denmark. Email: `beetle@it-c.dk`.
This work is part of the DSSCV project supported by the IST Programme of the European Union (IST-2001-35443).

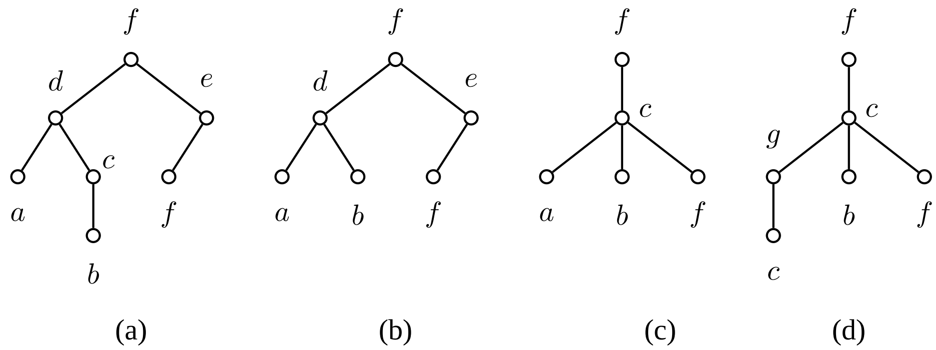

Figure 1: Transforming (a) into (d) via editing operations. (a) A tree. (b) The tree after deleting the node labeled c. (c) The tree after a horizontal-merge of the nodes labeled d and e into a node labeled c (d) The tree after a vertical-split of the node labeled a into the nodes labeled c and g. Conversely, we can transform (d) to (a) via a vertical-merge, a horizontal-split and an insert operation.

vertical-merge Merge a sequence of nodes $v_1, \dots, v_s$, where $\mathrm{parent}(v_{i+1}) = v_i$, $1 \leq i < s$, into a single node v. The children of $v_1, \dots, v_s$ not in the sequence become the children of v.

vertical-split The complement of vertical-merge. Split a node v into a sequence of nodes $v_1, \dots, v_s$, where $\mathrm{parent}(v_{i+1}) = v_i$, $1 \leq i < s$. The children of v become the children of the sequence $v_1, \dots, v_s$.

For unordered trees the operations can be defined similarly. In this case, the insert, delete, merge and split operations works on *subsets* of nodes instead of subsequences. An example of the above edit operations applied to ordered trees is shown in Figure 1.

We can define a tree edit distance problem for any subset O of the above operations. Let T_1 and T_2 be rooted, ordered and labeled trees. Assume that we are given a cost defined on each edit operation in O. An *edit script* S between T_1 and T_2 is a sequence of edit operations from O turning T_1 into T_2. The cost of S is the sum of the costs of the operations in S. An *optimal edit script* between T_1 and T_2 is an edit script between T_1 and T_2 of minimum cost and this cost is the *tree edit distance with respect to* O. The *tree edit distance problem with respect to* O is to compute the edit distance with respect to O and the corresponding edit script.

Let $|T_1|$, D_1 and L_1 denote the number of nodes, the maximum depth and the number of leaves in T_1 respectively and similarly define $|T_2|$, D_2 and L_2 for T_2. The edit distance problem with respect to the relabel, delete and insert operations, which we call the *standard edit distance problem*, is a well studied problem. The ordered version was introduced by Tai [Tai79] as a generalization of the well-known *string edit distance problem* [WF74]. The currently fastest algorithms are due Zhang and Shasha [ZS89] using $O(|T_1||T_2|\min(L_1, D_1)\min(L_2, D_2))$ time and $O(|T_1||T_2|)$ space, [Kle98] using $O(|T_1|^2|T_2|\log|T_2|)$ and $O(|T_1||T_2|)$ space and Chen [Che01] using $O(|T_1||T_2| + L_1^2|T_2| + L_1^{2.5}L_2)$ time and $O((|T_1| + L_1^2)\min(L_2, D_2) + |T_2|)$ space.

The unordered version of the problem has been shown to be NP-complete [ZSS92] and even MAX-SNP hard [ZJ94]. Hence, unless P=NP there is no PTAS for the problem [ALM+92].

All of the above algorithms compute the standard edit distance problem use the classic technique of *dynamic programming* (see, *e.g.*, [CLRS01, Chapter 15]). Furthermore, the algorithms are based on a reduction to *edit distance mappings*. An edit distance mapping is a compact representation of an edit script which may be viewed as a set of lines from nodes in T_1 to nodes in T_2. Each line corresponds to an edit operation. In this paper we introduce several new types of edit distance mappings which generalize the previous definition. This leads to edit distance problems extended with the above merge and split operations. Specifically, we consider the *horizontal edit distance problem* and the *vertical edit distance problem*, defined as the edit distance problem with respect to relabel, delete, insert, horizontal-merge and horizontal-split operations and the edit distance problem with respect to the relabel, delete, insert, vertical-merge and vertical-split operation respectively. We call these problems the *merge edit distance problems*. Define a merge edit distance problem to be k-way, for some integer $k > 1$, if no node in T_2 is the result of merging more than k nodes in T_1 and no node in T_1 is split into more than k nodes in T_2.

Our main result in this paper is that under some restrictions the k-way horizontal edit distance problem, for any constant k, and the vertical edit distance problem can be solved in polynomial time. Our algorithms all use dynamic programming to compute an optimal mapping. We only show how to compute the cost of the edit distances, however, the corresponding edit scripts can easily be found within the same time and space bounds given here.

1.1 Related work

Several other extension of the standard edit distance problem have been considered. In [KTSK00] Klein *et al.* developed an edit distance specifically for computing distances between ordered trees representing closed shapes in the plane. This edit distance also includes a type of merge operation. However, this operation is simpler than ours and involves deleting a subtree rooted at one of the nodes participating in the merge. Chawathe *et al.* [CRGMW96] considered an edit distance for ordered trees with a subtree move operation which moves and entire subtree from one node to another. Building on this work an algorithm for unordered trees is given in [CGM97]. This algorithm further extends the set of operations with a subtree copy operation which copies an entire subtree from one node to another node. Both of the algorithms in [CRGMW96, CGM97] are *heuristic*, that is, they do not guarantee that the solution they produce is the optimal. Interestingly, [CGM97] proposes merge operations as the ones in this paper, but does not consider how to implement these.

Instead of extending the standard edit distance problem with new operations, some restrictions have also been considered. In [Sel77, Zha96, Zha95, LST01] only edit scripts with various structural properties are considered. For a survey on tree edit distances and related problems see [Bil03].

1.2 Outline

In Section 2 we present the fundamental notation and definitions used throughout the paper. Section 3 formally defines the edit operations and the edit distance problems. Furthermore, the concept of mappings is presented. In Section 4 and 5 we present the algorithms for the horizontal and vertical edit distance problems respectively.

2 Preliminaries and notation

In this section we define notations and definitions we will use throughout the paper. For a graph G we denote the set of nodes and edges by $V(G)$ and $E(G)$ respectively. A forest is a set of trees. Let F be a forest. The *size* of F, denoted by $|F|$, is $|V(F)|$. A node with no children is a leaf and otherwise an internal node. We denote the parent of node v by $\mathrm{parent}(v)$. Two nodes are siblings if they have the same parent. Define θ to be the empty forest. For forests we allow the delete operation to be performed on roots. If a root $v \in V(F)$ with children $v_1, \ldots, v_s$ is deleted then $v_1, \ldots, v_s$ become roots in F in the place of v. Let $T(v)$ denote the subtree of F rooted at a node $v \in V(F)$ and let $F(v)$ denote the forest obtained by deleting v from $T(v)$. If $w \in V(T(v))$ then v is an ancestor of w, and if $w \in V(F(v))$ then v is a proper ancestor of w. If v is a (proper) ancestor of w then w is a (proper) descendant of v. A *vertical path* is a simple path from a node v to a node $w \in T(v)$. Let p be a vertical path from a node v to a node $w \in T(v)$ and define $V(p)$ to be the set of nodes on this path including v and w. If $u \in V(T(v))$ then u is a descendant of p and if $u \in V(T(v))\backslash V(p)$ then u is a proper descendant of p. A vertical path p' with topmost node u is a (proper) descendant of p if u is a (proper) descendant of p.

A tree T is a *labeled tree* if each node is a assigned a symbol from a fixed finite alphabet Σ. We say that T is *ordered* if a left-to-right order among the siblings is given. A forest F is ordered if a left-to-right order among the trees is given and each tree is ordered. Throughout the text we assume unless otherwise stated that any tree is rooted, ordered and labeled and any forest is an ordered forest consisting of rooted, ordered and labeled trees.

Let F be a forest and define the (i, j)-*deleted subforest* of F, $0 \leq i+j \leq |F|$, as the forest obtained from F by first deleting the rightmost root repeatedly j times and then, similarly, deleting the leftmost root i times. We call the $(0, j)$-deleted and $(j, 0)$-deleted subforests, for $0 \leq j \leq |F|$, the *prefixes* and the *suffixes* of F respectively. The number of (i, j)-deleted subforests of F is $\sum_{k=0}^{|F|} k = O(|F|^2)$, since for each i there are $|F| - i$ choices for j. Let v be any node in $V(F)$. We denote by $F - v$ the forest obtained by deleting v from F. Define $F[v]$ as the maximal prefix of F not containing v or any descendant of v. Similarly, define $F\{v\}$ as the minimal suffix of F containing v. Thus, $V(F[v]) \cap V(F\{v\}) = \emptyset$ and, if v is a root, $V(F[v])\backslash V(F\{v\}) = V(F)$. The nodes to the *left* of v are the nodes $w \in V(F[v])$ and the nodes to the *right* v are the nodes $u \in V(F\{v\}) \cap T(v)$. For any two forests F_1 and F_2 defined by the sequence of trees $T_{1_1}, \ldots, T_{1_s}$ and $T_{2_1}, \ldots, T_{2_t}$ respectively, we define $F_1 \bullet F_2$ as the sequence of trees $F = T_{1_1}, \ldots, T_{1_s}, T_{2_1}, \ldots, T_{2_t}$.

3 Edit operations and edit mappings

In this section we formally define the edit operations and the various edit distance problems. Throughout the section let F_1 and F_2 be ordered, labeled forests with labels from a finite alphabet Σ. We use the symbol λ to denote a special *null* node not in any forest and also a special *null* symbol $\lambda \notin \Sigma$. Define $V(F)_\lambda = V(F) \cup \lambda$ for any forest F and $\Sigma_\lambda = \Sigma \cup \lambda$. The *label* of a node $v \in V(F)$ is denoted by $\mathrm{label}(v)$ and the label of the node λ is the symbol λ.

Following [Tai79] we represent each edit operation by a set of pairs $(v_1, v_2) \in (V(F_1)_\lambda \times V(F_2)_\lambda)\backslash\{(\lambda, \lambda)\}$, often written $(v_1 \to v_2)$, where v_1 is a node in F_1 or λ and v_2 is a node in F_2 or λ. A single pair $(v_1 \to v_2)$ is a relabeling if $v_1 \neq \lambda$ and $v_2 \neq \lambda$, a deletion if $v_2 = \lambda$ and an insertion if $v_1 = \lambda$. A set of pairs $(v_{1_1} \to v_2), \ldots, (v_{1_s} \to v_2)$ is a horizontal-merge if $v_{1_1}, \ldots, v_{1_s}$ are siblings and consecutive in the left-to-right order of F_1 and a vertical-merge if $\mathrm{parent}(v_{i+1}) = v_i$, $1 \leq i < s$. Similarly, we represent the split operations by a set of pairs $(v_1 \to v_{2_1}), \ldots, (v_1 \to v_{2_t})$. Furthermore, for subsets $\mathbf{v_1} \subseteq V(F_1)_\lambda$ and $\mathbf{v_2} \subseteq V(F_2)_\lambda$, we define a shorthand notation for a set of pairs:

$$(\mathbf{v_1} \to \mathbf{v_2}) = \{(v_1 \to v_2) \mid (v_1, v_2) \in \mathbf{v_1} \times \mathbf{v_2}\}.$$

In general, we will use boldface letters to denote subsets of nodes. Note that by definition any edit operation can be written as $(\mathbf{v_1} \to \mathbf{v_2})$ for appropiate subsets $\mathbf{v_1} \subseteq V(F_1)_\lambda$ and $\mathbf{v_2} \subseteq V(F_2)_\lambda$. We say that any node v in F_1 or F_2 that occurs in a pair that is part of a edit operation, *participates* in that operation. An *edit script* between F_1 and F_2 is a sequence of edit operations turning F_1 into F_2. A *legal edit script* is an edit script $S = s_1, \ldots, s_i$ such that for any operation s_j, $1 \leq j \leq i$, we have that:

- If s_j is a horizontal- or vertical-merge operation resulting in a node v, then v does not participate in any of the operations $s_{j+1}, \ldots, s_i$.

- If s_j is a horizontal- or vertical-split operation splitting a node v, then v does not participate in any of the operations $s_1, \ldots, s_{j-1}$.

In the rest of the paper we will only consider legal edit scripts. Hence, unless otherwise stated, we will implictly refer to a legal edit script when we write edit script.

We assume that we are given a cost function $\gamma : (\Sigma_\lambda \times \Sigma_\lambda)\backslash\{(\lambda, \lambda)\} \rightarrow \mathbb{R}$, on pairs of labels. This cost should be a distance metric, that is, for any labels l_1,l_2,$l_3 \in \Sigma_\lambda$ the following conditions are satisfied:

1. $\gamma(l_1, l_2) \geq 0$, $\gamma(l_1, l_1) = 0$.
2. $\gamma(l_1, l_2) = \gamma(l_2, l_1)$.
3. $\gamma(l_1, l_3) \leq \gamma(l_1, l_2) + \gamma(l_2, l_3)$.

We define $\gamma(v_1 \rightarrow v_2) = \gamma(\text{label}(v_1), \text{label}(v_2))$, where $v_1 \in V(F_1)_\lambda$ and $v_2 \in V(F_2)_\lambda$. The cost of an edit operation $\gamma(\mathbf{v_1} \rightarrow \mathbf{v_2})$ is given by $\sum_{(v_1 \rightarrow v_2) \in (\mathbf{v_1} \rightarrow \mathbf{v_2})} \gamma(v_1 \rightarrow v_2)$. Note that for a legal edit script we have $\gamma(\mathbf{v_1} \rightarrow \mathbf{v_3}) \leq \gamma(\mathbf{v_1} \rightarrow \mathbf{v_2}) + \gamma(\mathbf{v_2} \rightarrow \mathbf{v_3})$, which does not holds in general. This is the primary reason for only considering legal edit scripts.

The cost of a sequence $S = s_1, \ldots, s_k$ of operations is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. The edit distance with respect to O between F_1 and F_2, denoted $\delta_O(F_1, F_2)$, is formally defined as:

$$\delta_O(F_1, F_2) = \min\{\gamma(S) \mid S \text{ is a sequence of edit operations from } O \text{ transforming } F_1 \text{ into } F_2\}.$$

If no sequence of operations from O transforms F_1 to F_2 we define $\delta_O(F_1, F_2) = \infty$.

A *mapping* between F_1 and F_2 is a representation of an edit script between F_1 and F_2, which is used in many of the algorithms for the tree edit distance problem. We define a *mapping* between F_1 and F_2 to be a a triple (M, F_1, F_2), such that $M \subseteq V(F_1) \times V(F_2)$. When there is no confusion we will simply use M to denote the mapping. For subsets of nodes $\mathbf{v_1} \subseteq V(F_1)$ and $\mathbf{v_2} \subseteq V(F_1)$ we define the sets:

$$\text{map}(\mathbf{v_1}) = \{v_2 \in V(F_2) \mid \exists (v_1, v_2) \in M \text{ such that } v_1 \in \mathbf{v_1}\}$$
$$\text{map}(\mathbf{v_2}) = \{v_1 \in V(F_1) \mid \exists (v_1, v_2) \in M \text{ such that } v_2 \in \mathbf{v_2}\}$$

We extend the notation by setting $\text{map}(v_1) = \text{map}(\{v_1\})$ and $\text{map}(v_2) = \text{map}(\{v_2\})$ for any nodes $v_1 \in V(F_1)$ and $v_2 \in V(F_2)$. We define three types of mappings: We say that M is a *one-to-one* mapping if, for any pair $(v_1, v_2) \in M$, $\text{map}(v_1) = \{v_2\}$ and $\text{map}(v_2) = \{v_1\}$, a *many-to-one* mapping if, for any pair $(v_1, v_2) \in M$, $\text{map}(\text{map}(v_1)) = \{v_1\}$ or $\text{map}(\text{map}(v_2)) = \{v_2\}$ and otherwise M is a *many-to-many* mapping. If M is a many-to-one mapping we will often write $(\mathbf{v_1}, \mathbf{v_2}) \in M$ if $\mathbf{v_1} = \{v_1\}$ and $\mathbf{v_2} = \text{map}(v_1)$ or $\mathbf{v_2} = \{v_2\}$ and $\mathbf{v_1} = \text{map}(v_2)$.

We say that a node v in F_1 or F_2 is *touched by a line* in M if v occurs in some pair in M. Let N_1 and N_2 be the set of nodes in F_1 and F_2 respectively not touched by any line in M. The cost of M is given by:

$$\gamma(M) = \sum_{(v_1, v_2) \in M} \gamma(v_1 \rightarrow v_2) + \sum_{v_1 \in N_1} \gamma(v_1 \rightarrow \lambda) + \sum_{v_2 \in N_2} \gamma(\lambda \rightarrow v_2)$$

Mappings can be composed. Let F_1, F_2 and F_3 be forests and let M_1 and M_2 be a mapping from F_1 to F_2 and from F_2 to F_3 respectively. Define

$$M_1 \circ M_2 = \{(\mathbf{v_1}, \mathbf{v_3}) \mid \exists \mathbf{v_2} \in V(F_2) \text{ such that } (\mathbf{v_1}, \mathbf{v_2}) \in M_1 \text{ and } (\mathbf{v_2}, \mathbf{v_3}) \in M_2\}$$

Note that a if M_1 and M_2 are one-to-one mappings $M_1 \circ M_2$ is a one-to-one mapping. In general, the composition of two many-to-one mappings is a many-to-many mapping. If $\gamma(\mathbf{v_1} \rightarrow \mathbf{v_3}) \leq \gamma(\mathbf{v_1} \rightarrow$

$\mathbf{v_2}) + \gamma(\mathbf{v_2} \to \mathbf{v_3})$, for any pairs $(\mathbf{v_1}, \mathbf{v_2}) \in M_1$ and $(\mathbf{v_2}, \mathbf{v_3}) \in M_2$, we say that M_1 and M_2 are *compatible*.

Lemma 1 *For any three forests F_1, F_2 and F_3 and compatible many-to-one mappings (M_1, F_1, F_2) and (M_2, F_2, F_3),*

$$\gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2)$$

Proof. Let N_1 and N_3 be the set of nodes in F_1 and F_3 respectively not touched by a line in $(M_1 \circ M_2, F_1, F_3)$. For a node $v_1 \in V(F_1)$ there are two cases to consider. If $v_1 \in N_1$ then either v_1 is not touched by a line in M_1 or $(v_1, v_2) \in M_1$ and v_2 is not touched by a line in M_2, for some $v_2 \in V(F_2)$. By the triangle inequality $\gamma(v_1 \to \lambda) \leq \gamma(v_1 \to v_2) + \gamma(v_2 \to \lambda)$. If $(v_1, v_3) \in M_1 \circ M_2$ for some node $v_3 \in V(F_3)$, then let $(\mathbf{v_1} \to \mathbf{v_3}) \in M_1 \circ M_2$, be the pair such that $v_1 \in \mathbf{v_1}$ and $v_3 \in \mathbf{v_3}$. Since M_1 and M_2 are compatible we have that $\gamma(\mathbf{v_1} \to \mathbf{v_3}) \leq \gamma(\mathbf{v_1} \to \mathbf{v_2}) + \gamma(\mathbf{v_2} \to \mathbf{v_3})$, for any pairs $(\mathbf{v_1}, \mathbf{v_2}) \in M_1$ and $(\mathbf{v_2}, \mathbf{v_3}) \in M_2$. Equivalently, the result holds for any node $v_3 \in V(F_3)$ and hence the lemma follows. □

For each edit distance problem we study in this paper there is a corresponding minimum cost mapping with the same cost as the edit distance. For completeness and comparison we first present the mapping used for the standard edit distance problem and then define mappings for the merge edit distance problems.

A *standard edit distance mapping*, M_e, between F_1 and F_2 is a one-to-one mapping such that for all pairs $(v_1, v_2), (w_1, w_2) \in M_e$:

- v_1 is a proper descendant of v_2 iff w_1 is a proper descendant of w_2. (descendant condition)
- v_1 is to the left of v_2 iff w_1 is to the left of w_2. (sibling condition)

By the definition of M_e and since γ is a metric, it is not hard to show that a minimum cost standard edit distance mapping is equivalent to the standard edit distance:

Lemma 2 ([Tai79]) *For any forest F_1 and F_2, the standard edit distance, $\delta_e(F_1, F_2)$, satisfies:*

$$\delta_e(T_1, T_2) = \min\{\gamma(M_e) \mid (M_e, F_1, F_2) \textit{ is a standard edit distance mapping}\}.$$

Let (M, F_1, F_2) be a many-to-one mapping. We define M to be *normal*, if for all pairs $(\mathbf{v_1}, \mathbf{v_2})$, $(\mathbf{w_1}, \mathbf{w_2}) \in M$, either all nodes in $\mathbf{v_1}$ are descendants (ancestors) of nodes in $\mathbf{w_1}$ or all nodes in $\mathbf{v_1}$ are to the left (right) $\mathbf{w_1}$ and the equivalent conditions also hold for $\mathbf{v_2}$ and (w_2). We say that M is *horizontal*, if for any pair $(\mathbf{v_1}, \mathbf{v_2}) \in M$, no pair of nodes in $\mathbf{v_1}$ or $\mathbf{v_2}$ are a descendant of each other. Similarly, we say that M is *vertical* if no pair of nodes in $\mathbf{v_1}$ or $\mathbf{v_2}$ are to the left and right of each other. Note that if M is a one-to-one mapping it is both vertical and horizontal. For a horizontal mapping the leftmost and rightmost node in $\mathbf{v}$, where $\mathbf{v}$ is either $\mathbf{v_1}$ or $\mathbf{v_2}$, is well-defined and we denote these by $\mathrm{right}(\mathbf{v})$ and $\mathrm{left}(\mathbf{v})$ respectively. Similarly, for a vertical mapping the topmost and bottommost node is denoted by $\mathrm{top}(\mathbf{v})$ and $\mathrm{bottom}(\mathbf{v})$. Furthermore, we define $\mathrm{path}(\mathbf{v})$ for a vertical mapping to be the path from $\mathrm{top}(\mathbf{v})$ to $\mathrm{bottom}(\mathbf{v})$. If $\mathbf{v} = \{v\}$ then $\mathrm{right}(\mathbf{v}) = \mathrm{left}(\mathbf{v}) = \mathrm{top}(\mathbf{v}) = \mathrm{bottom}(\mathbf{v}) = v$. If $|\mathbf{v_1}| \leq k$ and $|\mathbf{v_2}| \leq k$, for all pairs $(\mathbf{v_1}, \mathbf{v_2}) \in M$ and some positive integer k, the mapping is k-way. Finally, we can properly define the many-to-one mappings that correspond to the merge edit distance problems.

A *merge edit distance mapping*, M_m, between F_1 and F_2 is a normal many-to-one mapping such that for all pairs $(\mathbf{v_1}, \mathbf{v_2})$, $(\mathbf{w_1}, \mathbf{w_2}) \in M_m$,

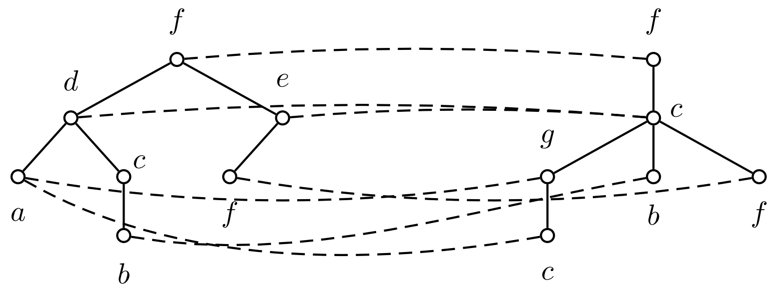

Figure 2: The mapping corresponding to the edit script in Figure 1. The mapping is a merge edit distance mapping but it is neither horizontal nor vertical.

- All nodes in $\mathbf{v_1}$ are proper descendants of nodes in $\mathbf{w_1}$ iff all nodes in $\mathbf{v_2}$ are proper descendants of nodes in $\mathbf{w_2}$. (merge descendant condition)
- All nodes in $\mathbf{v_1}$ is to the left of all $\mathbf{w_1}$ iff all nodes in $\mathbf{v_2}$ is to the left of all nodes in $\mathbf{w_2}$. (merge sibling condition)

Note that if M_m is one-to-one the definition is equivalent to the definition of standard edit mappings, and hence, the definition generalize standard edit mappings. If M_m is horizontal or vertical then M_m is a horizontal or vertical edit distance mapping respectively. The mapping corresponding to the edit operations in Figure 1 is shown in Figure 2.

Lemma 3 *For forests F_1 and F_2 the horizontal and vertical edit distance, $\delta_h(F_1, F_2)$ and $\delta_v(F_1, F_2)$ satisfies:*

$$\delta_h(F_1, F_2) = \min\{M_h \mid M_h \textit{ is a horizontal edit distance mapping}\}$$
$$\delta_v(F_1, F_2) = \min\{M_v \mid M_v \textit{ is a vertical edit distance mapping}\}$$

Proof. We show the lemma for the horizontal edit distance. The vertical part follows by the same argument. Let $S = s_1, \ldots, s_i$ be an minimum cost horizontal edit distance script between F_1 and F_2. We show that there exists a horizontal edit distance mapping M_h such that $\gamma(M_h) \leq \gamma(S)$ by induction on i. If $i = 1$ construct the mapping corresponding to the pairs of s_1 of the form $(v_1 \to v_2)$ representing the edit operation. For any type of operation s_1 we clearly obtain a horizontal edit distance mapping of the same cost as s_1. Let $S_1 = s_1, \ldots, s_{i-1}$ and assume that there exists a mapping M_1 such that $\gamma(M_1) \leq \gamma(S_1)$. Let M_2 be the mapping corresponding to edit operation s_i. Since $s_1, \ldots, s_i$ is a legal edit script it follows that M_1 and M_2 are compatible and by the definition of horizontal edit distance mappings $M_1 \circ M_2$ is also a horizontal edit distance mapping. Furthermore, by Lemma 1 we have that,

$$\gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2) \leq \gamma(S_1) + \gamma(S_2) = \gamma(S).$$

Conversely, for any horizontal edit distance mapping M_h, we can construct a sequence S of edit operation indicated by the mapping. For each pair $(\mathbf{v_1}, \mathbf{v_2}) \in M_h$ perform the relabel, merge or split operation $(\mathbf{v_1} \to \mathbf{v_2})$, then delete all nodes not touched by a line in F_1 and then insert all nodes not touched by a line in F_2. Hence, $\gamma(S) = \gamma(M_h)$ and the lemma follows. □

From the above lemma we immediately have that a minimum cost k-way horizontal (vertical) edit distance mapping is equal to the k-way horizontal (vertical) edit distance. Note that without the restriction to legal edit scripts, Lemma 3 would not hold and the problem would not reduce to finding a many-to-one mapping.

4 Horizontal merges and splits

In this section we show how to compute the horizontal edit distance using dynamic programming. We describe in detail the algorithm for the two-way horizontal edit distance and subsequently describe how to generalize this to k-way edit distances, for any integer k, $k \geq 1$. The algorithm computes the cost of a minimum cost horizontal edit distance mapping but it is straightforward to also compute the mapping and the corresponding edit script without changing the asymptotic running time or space usage of the algorithm. For a forest F and nodes $w, v \in V(F)$, let $\sigma(F, w, v)$ denote the cost of deleting the set nodes that are to the right of w and to the left of v in F.

Lemma 4 *Let F_1 and F_2 be forests with rightmost roots v_1 and v_2 respectively. The two-way horizontal merge edit distance, δ_h^2, satisfies the recurrence:*

$$
\begin{aligned}
\delta_h^2(\theta, \theta) &= 0 \\
\delta_h^2(F_1, \theta) &= \delta_h^2(F_1 - v_1, \theta) + \gamma(v_1 \to \lambda) \\
\delta_h^2(\theta, F_2) &= \delta_h^2(\theta, F_2 - v_2) + \gamma(\lambda \to v_2) \\
\delta_h^2(F_1, F_2) &= \min \begin{cases}
\delta_h^2(F_1 - v_1, F_2) + \gamma(v_1 \to \lambda) \\
\delta_h^2(F_1, F_2 - v_2) + \gamma(\lambda \to v_2) \\
\delta_h^2(F_1(v_1), F_2(v_2)) + \delta_h^2(F_1[v_1], F_2[v_2]) + \gamma(v_1 \to v_2) \\
\displaystyle\min_{w_1 \in V(F_1[v_1])} \delta_h^2(F_1[w_1], F_2[v_2]) + \delta_h^2(F_1(w_1) \bullet F_1(v_1), F_2(v_2)) \\
\qquad\qquad + \sigma(F_1, w_1, v_1) + \gamma(w_1 \to v_2) + \gamma(v_1 \to v_2) \\
\\
\displaystyle\min_{w_2 \in V(F_1[v_2])} \delta_h^2(F_1[v_1], F_2[w_2]) + \delta_h^2(F_1(v_1), F_2(w_2) \bullet F_2(v_2)) \\
\qquad\qquad + \sigma(F_2, w_2, v_2) + \gamma(v_1 \to w_2) + \gamma(v_1 \to v_2)
\end{cases}
\end{aligned}
$$

Proof. The first three equations are trivially true. To show the last equation consider a minimum cost two-way horizontal mapping M_h^2 between F_1 and F_2. Let N_1 and N_2 be the set of nodes in F_1 and F_2 respectively not touched by a line in M_h^2. There are three possibilities for v_1 and v_2:

Case 1: v_1 is not touched by a line. Then $v_1 \in N_1$ and hence,

$$\delta_h^2(F_1, F_2) = \delta_h^2(F_1 - v_1, F_2) + \gamma(v_1 \to \lambda).$$

Case 2: v_2 is not touched by a line. Then $v_2 \in N_2$ and hence,

$$\delta_h^2(F_1, F_2) = \delta_h^2(F_1, F_2 - v_2) + \gamma(\lambda \to v_2).$$

Case 3: v_1 and v_2 are both touched by lines. We show that this implies that $(v_1, v_2) \in M_h^2$. Let $a = \mathrm{right}(\mathrm{map}(v_1))$ and $b = \mathrm{right}(\mathrm{map}(v_2))$ and assume that $v_1 \neq b$ and $v_2 \neq a$. If a is to the left of v_2 then by the merge sibling condition v_1 must be to the left of b, which is impossible since no nodes are to the left of v_1 and v_2. If a is a proper descendant of v_2 then by the merge descendant condition v_1 must be a proper descendant of b, which is impossible since v_1 and v_2 are roots. Hence, $(v_1, v_2) \in M_h^2$.

Let $(\mathbf{v_1}, \mathbf{v_2}) \in M_h^2$ denote the pair such that $v_1 \in \mathbf{v_1}$ and $v_2 \in \mathbf{v_2}$. Since the M_h^2 is a two-way mapping there are three subcases to consider:

(i) $\mathbf{v_1} = \{v_1\}$ and $\mathbf{v_2} = \{v_2\}$. Hence,

$$\delta_h^2(F_1, F_2) = \delta_h^2(F_1(v_1), F_2(v_2)) + \delta_h^2(F_1[v_1], F_2[v_2]) + \gamma(v_1 \to v_2).$$

(ii) $\mathbf{v_1} = \{w_1, v_1\}$, for some w_1 to the left of v_1, and $\mathbf{v_2} = \{v_2\}$. Then all proper descendants of w_1 and v_1 must be mapped to proper descendants of v_2, and all nodes to the right of w_1 and to the left of v_1 must be deleted. Hence,

$$\begin{aligned}\delta_h^2(F_1, F_2) = \delta_h^2(F_1[w_1], F_2[v_2]) + \delta_h^2(F_1(w_1) \bullet F_1(v_1), F_2(v_2)) \\ + \sigma(F_1, w_1, v_1) + \gamma(w_1 \to v_2) + \gamma(v_1 \to v_2).\end{aligned}$$

(iii) $\mathbf{v_1} = \{v_1\}$ and $\mathbf{v_2} = \{w_2, v_2\}$, for some w_2 to the left of v_2. As above,

$$\begin{aligned}\delta_h^2(F_1, F_2) = \delta_h^2(F_1[v_1], F_2[w_2]) + \delta_h^2(F_1(v_1), F_2(w_2) \bullet F_2(v_2)) \\ + \sigma(F_2, w_2, v_2) + \gamma(v_1 \to w_2) + \gamma(v_1 \to v_2).\end{aligned}$$

Taking the minimum over all possible values of w_1 and w_2 and over all of the above cases the lemma follows. $\square$

The recurrence in Lemma 4 suggests a dynamic program. The value $\delta_h^2(F_1, F_2)$ depends on a number of subproblems of smaller size. Hence, we can compute $\delta_h^2(F_1, F_2)$ by computing the value of each subproblem in order of increasing size. Let $w_1, v_1 \in V(F_1)$ and $w_2, v_2 \in V(F_2)$, where w_1 is to the left v_1 and w_2 is to the left v_2. By Lemma 4 the subproblems (S_1, S_2) are of the following three forms:

1. S_1 is a prefix of $F_1(v_1)$ and S_2 is a prefix of $F_2(v_2)$, for any pair of nodes $v_1 \in V(F_1)$ and $v_2 \in V(F_2)$.

2. S_1 is a prefix of $F_1(w_1) \bullet F_1(v_1)$ and S_2 is a prefix of $F_2(v_2)$, for any nodes $w_1, v_1 \in V(F_1)$, where w_1 is to the left of v_1, and $v_2 \in V(F_2)$.

3. S_1 is a prefix of $F_1(v_1)$ and S_2 is a prefix of $F_2(w_2) \bullet F_2(v_2)$, for any nodes $v_1 \in V(F_1)$ and $w_2, v_2 \in V(F_2)$, where w_2 is to the left of v_2.

We count the number of subproblems as follows. For the first kind there are $O(|F_1|)$ and $O(|F_2|)$ choices for v_1 and v_2 respectively and for each choice there are $O(|F_1|)$ and $O(|F_2|)$ prefixes of $F_1(v_1)$ and $F_2(v_2)$. Hence, in total there are $O(|F_1|^2|F_2|^2)$ subproblems of the first kind. Similarly, for the second and third kind there are $O(|F_1|^3|F_2|^2)$ and $O(|F_1|^2|F_2|^3)$ subproblems respectively. By Lemma 4 each subproblem depends on at most $O(|F_1| + |F_2|)$ subproblems and thus the total time to compute $\delta_h^2(F_1, F_2)$ is $O((|F_1|^3|F_2|^2 + |F_1|^2|F_2|^3)(|F_1| + |F_2|)) = O(n^6)$, where $n = \max(|F_1|, |F_2|)$.

Theorem 1 *Let F_1 and F_2 be ordered forests and let $n = \max(|F_1|, |F_2|)$. The two-way horizontal merge edit distance (for legal edit scripts), $\delta_h^2(F_1, F_2)$, can be computed in time and space $O(n^6)$.*

It is straightforward to generalize Theorem 1 to handle k-way horizontal merge edit distances. In this case we need to compute all problems of the form $(F_1(v_{1_1}) \bullet \cdots \bullet F_2(v_{1_s}), F_2(v_2))$ and $(F_1(v_1), F_2(v_{2_t}) \bullet \cdots \bullet F_2(v_{2_j}))$, where v_{1_i} and v_{2_j} is to the left of $v_{1_{i+1}}$ and $v_{2_{j+1}}$ respectively, $1 \leq i < s$, $1 \leq j < t$ and $s, t \leq k$. This gives a total of $O(|F_1|^{k+1}|F_2|^2)$ and $O(|F_1|^2|F_2|^{k+1})$ subproblems of the second and third kind respectively. Each subproblem depends on $O(|F_1|^{k-1} + |F_2|^{k-1})$ subproblems and hence the total the time to compute $\delta_h^k(F_1, F_2)$ is $O((|F_1|^{k+1}|F_2|^2 + |F_1|^2|F_2|^{k+1})(|F_1|^{k-1} + |F_2|^{k-1})) = O(n^{2k+2})$, for any $k > 1$.

Theorem 2 *Let F_1 and F_2 be ordered forests and let $n = \max(|F_1|, |F_2|)$. The k-way horizontal merge edit distance (for legal edit scripts), $\delta_h^k(F_1, F_2)$, can be computed in time and space $O(n^{2k+2})$.*

5 Vertical merges and splits

In this section we show how to compute the vertical edit distance. Let F_1 and F_2 be forests. For nodes $w_1, v_1 \in V(F_1)$, such that $w_1 \in F(v_1)$ and $v_2 \in V(F_2)$, define $\rho(F_1, w_1, v_1, v_2)$ as the cost of a minimum cost vertical edit distance mapping between the vertical path from v_1 to w_1, without v_1 and w_1, and the single node v_2. Hence, by definition of the vertical edit distance, each node u_1 on the path is either not touched by a line or u_1 is mapped to v_2. Equivalently, define $\rho(F_2, w_2, v_2, v_1)$.

Lemma 5 *Let F_1 and F_2 be forests with rightmost roots v_1 and v_2 respectively. The vertical merge edit distance, δ_v, satisfies the recurrence:*

$$
\begin{aligned}
\delta_v(\theta, \theta) &= 0 \\
\delta_v(F_1, \theta) &= \delta_h(F_1 - v_1, \theta) + \gamma(v_1 \to \lambda) \\
\delta_v(\theta, F_2) &= \delta_h(\theta, F_2 - v_2) + \gamma(\lambda \to v_2) \\
\delta_v(F_1, F_2) &= \min \begin{cases}
\delta_v(F_1 - v_1, F_2) + \gamma(v_1 \to \lambda) \\
\delta_v(F_1, F_2 - v_2) + \gamma(\lambda \to v_2) \\
\delta_v(F_1(v_1), F_2(v_2)) + \delta_v(F_1[v_1], F_2[v_2]) + \gamma(v_1 \to v_2) \\
\min\limits_{w_1 \in V(F_1(v_1))} \{\delta_v(F_1[v_1], F_2[v_2]) \\
\qquad + \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s}), F_2(v_2)) \\
\qquad + \gamma(w_1 \to v_2) + \gamma(v_1 \to v_2) + \rho(F_1, w_1, v_1, v_2)\} \\
\min\limits_{w_2 \in V(F_2(v_2))} \{\delta_v(F_1[v_1], F_2[v_2]) \\
\qquad + \delta_v(F_1(v_1), T_2(z_{2_1}) \bullet \cdots \bullet T_2(z_{2_t})) \\
\qquad + \gamma(v_1 \to w_2) + \gamma(v_1 \to v_2) + \rho(F_2, w_2, v_2, v_1)\}
\end{cases}
\end{aligned}
$$

where $z_{1_1}, \ldots, z_{1_s}$ and $z_{2_1}, \ldots, z_{2_t}$ is the set of children (ordered from left to right) of the path between v_1 and w_1 and v_2 and w_2 respectively.

Proof. The first three equations are trivially true. To show the last equation consider a minimum cost vertical mapping M_v between F_1 and F_2. Let N_1 and N_2 be the set of nodes in F_1 and F_2 respectively not touched by a line in M_v. There are three possibilities for v_1 and v_2:

Case 1: v_1 is not touched by a line. Then $v_1 \in N_1$ and hence,

$$\delta_v(F_1, F_2) = \delta_v(F_1 - v_1, F_2) + \gamma(v_1 \to \lambda).$$

Case 2: v_2 is not touched by a line. Then $v_2 \in N_2$ and hence,

$$\delta_v(F_1, F_2) = \delta_v(F_1, F_2 - v_2) + \gamma(\lambda \to v_2).$$

Case 3: v_1 and v_2 are both touched by lines. We show that this implies that $(v_1, v_2) \in M_v$. Let $a = \mathrm{top}(\mathrm{map}(v_1))$ and $b = \mathrm{top}(\mathrm{map}(v_2))$ and assume that $v_1 \neq b$ and $v_2 \neq a$. If a is to the left of v_2 then v_1 must be to the left of b by the vertical sibling condition. If a is a proper descendant of $\mathrm{path}(\mathbf{v_2})$ then v_1 must be a proper descendant of b by the vertical descendant condition. Both cases are impossible since v_1 and v_2 are the rightmost roots and hence $(v_1, v_2) \in M_v$.

Let $(\mathbf{v_1}, \mathbf{v_2}) \in M_v$ denote the pair such that $v_1 \in \mathbf{v_1}$ and $v_2 \in \mathbf{v_2}$. There are three subcases to consider:

(i) $\mathbf{v_1} = \{v_1\}$ and $\mathbf{v_2} = \{v_2\}$. As above,

$$\delta_v(F_1, F_2) = \delta_v(F_1(v_1), F_2(v_2)) + \delta_v(F_1[v_1], F_2[v_2]) + \gamma(v_1 \to v_2).$$

(ii) $|\mathbf{v_1}| > 1$, $w_1 = \mathrm{bottom}(\mathbf{v_1})$ and $\mathbf{v_2} = \{v_2\}$. Then all proper descendants of $\mathrm{path}(\mathbf{v_1})$ are mapped to proper descendants of v_2 and hence,

$$\begin{aligned}\delta_v(F_1, F_2) = {} & \delta_v(F_1[v_1], F_2[v_2]) + \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s}), F_2(v_2)) \\ & + \gamma(w_1 \to v_2) + \gamma(v_1 \to v_2) + \rho(F_1, w_1, v_1, v_2).\end{aligned}$$

(iii) $\mathbf{v_1} = \{v_1\}$, $|\mathbf{v_2}| > 1$ and $w_2 = \mathrm{bottom}(\mathbf{v_2})$. As above,

$$\begin{aligned}\delta_v(F_1, F_2) = {} & \delta_v(F_1[v_1], F_2[v_2]) + \delta_v(F_1(v_1), T_2(z_{2_1}) \bullet \cdots \bullet T_2(z_{2_t})) \\ & + \gamma(v_1 \to w_2) + \gamma(v_1 \to v_2) + \rho(F_2, w_2, v_2, v_1).\end{aligned}$$

Taking the minimum over all possible values of w_1 and w_2 and over all of the above cases the lemma follows. □

Lemma 6 *Let F_1 and F_2 be ordered trees and let $z_{1_1}, \ldots, z_{1_s}$ and $z_{2_1}, \ldots, z_{2_t}$ be ordered sequences of nodes from left to right in F_1 and F_2 respectively. Then,*

$$\delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s}), F_2) =$$
$$\min \begin{cases} \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_{s-1}}), F_2) + \delta_v(T_1(z_{1_s}), \theta) \\ \min_{w_2 \in V(F_2)} \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_{s-1}}), F_2[w_2]) + \delta_v(T_1(z_{1_s}), F_2\{w_2\}) \end{cases}$$
$$\delta_v(F_1, T_1(z_{2_1}) \bullet \cdots \bullet T_1(z_{2_t})) =$$
$$\min \begin{cases} \delta_v(F_1, T_2(z_{2_1}) \bullet \cdots \bullet T_2(z_{2_{t-1}})) + \delta_v(\theta, T_1(z_{2_t})) \\ \min_{w_1 \in V(F_1)} \delta_v(F_1[w_1], T_2(z_{2_1}) \bullet \cdots \bullet T_2(z_{2_{t-1}})) + \delta_v(F_1\{w_1\}, T_2(z_{2_t})) \end{cases}$$

Proof. We give the proof for the first equation. Consider a minimum vertical edit distance mapping M_v between $T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s})$ and F_2. If no node in $T_1(z_{1_s})$ is touched by a line in M_v,

$$\delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s}), F_2) = \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_{s-1}}), F_2) + \delta(T_1(z_{1_s}), \theta).$$

Conversely. let $\mathrm{path}(\mathbf{w_2})$ be the leftmost path in F_2 that is mapped to nodes in T_{1_s}. By the vertical sibling condition all nodes in $F_2[\mathrm{top}(\mathbf{w_2})]$ must map to nodes in $T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_{s-1}})$, while all nodes in $F_2\{\mathrm{top}(\mathbf{w_2})\}$ must map to nodes in T_{1_s}. Hence,

$$\delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_s}), F_2) = \delta_v(T_1(z_{1_1}) \bullet \cdots \bullet T_1(z_{1_{s-1}}), F_2[w_2]) + \delta_v(T_1(z_{1_s}), F_2\{w_2\}).$$

Setting $w_2 = \mathrm{top}(\mathbf{w_2})$ and taking the minimum over all possible values of w_2 the equation follows. The second equation can be shown symmmetrically and hence the lemma follows. □

The recurrence in Lemma 5 and 6 suggests a dynamic program. The value $\delta_v(F_1, F_2)$ depends on a number of subproblems of smaller size. Hence, we can compute $\delta_v(F_1, F_2)$ by computing the value of each subproblem in order of increasing size. From Lemma 5 it follows that the subproblems (S_1, S_2) are of the following three forms:

1. S_1 is a prefix of $F_1(v_1)$ and S_2 is a prefix of $F_2(v_2)$, for any pair of nodes $v_1 \in V(F_1)$ and $v_2 \in V(F_2)$.

2. $S_1 = T_1(z_1) \bullet \cdots \bullet T_2(z_{1_s})$, where $z_{1_1}, \ldots, z_{1_s}$ are the children of a vertical path and S_2 is a deleted subforest of F_2.

3. S_1 is a deleted subforest of F_1 and S_2 is $T_2(z_{2_1}) \bullet \cdots \bullet T_2(z_{2_t})$, where $z_{2_1}, \ldots, z_{2_t}$ are the children of a vertical path.

We count the number of subproblems as follows. For the first kind note that S_1 and S_2 in particular are a deleted subforests of F_1 and F_2 respectively. Inspecting Lemma 6 each subproblem of the second and third kind reduce to subproblems, where S_1 is a subtree of F_1 and S_2 is a deleted subforest of F_2 or S_1 is a deleted subforest of F_1 and S_2 is a subtree. Hence, any subproblem is of the form (S_1, S_2), where S_1 and S_2 are deleted subforests of F_1 and F_2 respectively. In total there are $O(|F_1|^2|F_2|^2)$ subproblems. The value of $\delta_v(F_1, F_2)$ depends on $O(|F_1| + |F_2|)$ subproblems which in turn depend on $O(|F_1||F_2| + |F_2||F_1|)$ subproblems. Hence, the total time to compute $\delta_v(F_1, F_2)$ is at most $O((|F_1|^2|F_2|^2)(|F_1| + |F_2|)(|F_1||F_2| + |F_2||F_1|)) = O(n^7)$, where $n = \max(|F_1|, |F_2|)$.

Theorem 3 *Let F_1 and F_2 be forests and let $n = max(|F_1|, |F_2|)$. The vertical edit distance (for legal edit scripts), $\delta_v(F_1, F_2)$, can be computed in time and space $O(n^7)$.*

References

[ALM+92] A. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. In *Proceedings of the 33rd IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 14–23, 1992.

[Bil03] Philip Bille. Tree edit distance, alignment distance and inclusion. Technical Report TR-2003-23, IT University of Copenhagen, 2003.

[CGM97] S. Chawathe and H. Garcia-Molina. Meaningful change detection in structured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–37, Tuscon, Arizona, 1997.

[Che01] Weimin Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40:135–158, 2001.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, second edition*. MIT Press, 2001.

[CRGMW96] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, Montréal, Québec, June 1996.

[HO82] Christoph M. Hoffmann and Michael J. O'Donnell. Pattern matching in trees. *Journal of the Association for Computing Machinery (JACM)*, 29(1):68–95, 1982.

[Kle98] P.N. Klein. Computing the edit-distance between unrooted ordered trees. In *Proceedings of the 6th annual European Symposium on Algorithms (ESA) 1998.*, pages 91–102. Springer-Verlag, 1998.

[KM95] Pekka Kilpeläinen and Heikki Mannila. Ordered and unordered tree inclusion. *SIAM Journal of Computing*, 24:340–356, 1995.

[KTSK00] Philip Klein, Srikanta Tirthapura, Daniel Sharvit, and Ben Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 696–704, 2000.

[LST01] Chin Lung Lu, Zheng-Yao Su, and Chuan Yi Tang. A new measure of edit distance between labeled trees. In *Proceedings of the 7th Annual International Conference on Computing and Combinatorics (COCOON)*, volume 2108 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2001.

[RR92] R. Ramesh and I.V. Ramakrishnan. Nonlinear pattern matching in trees. *Journal of the Association for Computing Machinery (JACM)*, 39(2):295–316, 1992.

[Sel77] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, 1977.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery (JACM)*, 26:422–433, 1979.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21:168–173, 1974.

[Zha95] Kaizhong Zhang. Algorithms for the constrained editing problem between ordered labeled trees and related problems. *Pattern Recognition*, 28:463–474, 1995.

[Zha96] Kaizhong Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3):205–22, 1996.

[ZJ94] Kaizhong Zhang and Tao Jiang. Some MAX SNP-hard results concerning unordered labeled trees. *Information Processing Letters*, 49:249–254, 1994.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262, 1989.

[ZSS92] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42:133–139, 1992.

[ZSW94] Kaizhong Zhang, Dennis Shasha, and Jason T. L. Wang. Approximate tree matching in the presence of variable length don't cares. *Journal of Algorithms*, 16(1):33–66, 1994.