INFORMATION SCIENCES **77**, 253–273 (1994)

Constrained Tree Editing

B. JOHN OOMMEN

and

WILLIAM LEE

School of Computer Science, Carleton University, Ottawa, Ont., Canada K1S 5B6

ABSTRACT

The distance between two ordered labeled trees is considered to be the minimum sum of the weights associated with the edit operations (insertion, deletion, and substitution) required to transform one tree to another. The problem of computing this distance and the optimal transformation using no edit constraints has been studied in the literature [3, 4, 7–9, 11]. In this paper, we consider the problem of transforming one tree T_1 to another tree T_2 using any arbitrary edit constraint involving the number and type of edit operations to be performed. An algorithm to compute this constrained distance is presented. If for a tree T, Span(T) is defined as the Min{Depth(T), Leaves(T)}, the time and space complexities of this algorithm are:

$$\text{Time: } O\Big(|T_1| * |T_2| * \big(\text{Min}\{|T_1|, |T_2|\}\big)^2 * \text{Span}(T_1) * \text{Span}(T_2)\Big)$$

$$\text{Space: } O\big(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\}\big).$$

---

1. INTRODUCTION

Trees, graphs, and webs are typically considered as a multidimensional generalization of strings. Among these different structures, trees are considered to be the most important "nonlinear" structures in computer science, and the tree-editing problem has been studied since 1976.

Similar to the string-editing problem [1, 2, 5–7, 10], the tree-editing problem concerns the determination of the distance between two trees as measured by the minimum cost sequence of edit operations. Typically, the edit sequence considered includes the substitution, insertion, and deletion

655 Avenue of the Americas, New York, NY 10010 0020-0255/94/$7.00

of nodes needed to transform one tree into the other. Applications of the tree-editing problem can be found in the theory of amino-acid sequence comparison, pattern recognition, and in the parsing of sentences from a grammar. For example, the secondary structure of RNA is a single strand of nucleotides which folds back onto itself into a shape that is topologically a tree [7, 11]. This strand influences the translation rates from RNA to proteins. A comparison of these structures yields information about the comparative functionality of different RNAs.

Unlike the string-editing problem which has been well developed, only a few results have been published concerning the tree-editing problem. In 1979, Selkow [8] presented a tree-editing algorithm in which insertions and deletions were only restricted to the leaves. Later, Tai [9] presented another scheme in which insertions/deletions were prohibited at the root. The algorithm of Lu [3], on the other hand, solved the problem for trees of depth two. The best known algorithm for solving the general tree-editing problem is the one due to Zhang and Shasha [11].

All of the above algorithms considered the editing of one tree, say T_1, and transforming it to another, T_2, with the edit processes being absolutely unconstrained. In this paper, we consider the problem of editing T_1 to T_2 subject to any general edit constraint. This constraint can be arbitrarily complex as long as it is specified in terms of the number and type of edit operations to be included in the optimal edit sequence. Some examples of constrained editing are presented below:

(a) What is the optimal way of editing T_1 to T_2 using no more than k deletions?

(b) How can we optimally transform T_1 to T_2 using exactly k substitutions?

(c) Is it possible to transform T_1 to T_2 using exactly k_i insertions, k_e deletions, and k_s substitutions? If so, what is the distance between T_1 to T_2 subject to this constraint?

In this paper, we present a consistent method of specifying arbitrary edit constraints. This method is analogous (but not identical) to the one shown in [5] and is specified by a constraint set, τ. We will then discuss the computation of $D_\tau(T_1, T_2)$, the edit distance between T_1 and T_2 subject to this constraint. A similar algorithm to achieve this was independently reported in [12]. However, the differences between our scheme and the latter will be discussed in a later section.

With regard to applications, just as the **constrained** string-editing algorithm [5, 6] has been used successfully to solve the noisy **subsequence** problem, we believe that to obtain acceptable recognition rates for subsequence trees, one must resort to **constrained** edit distances between the trees and not merely their unconstrained distances. We are currently investigating this in the pattern recognition of noisy (garbled) subsequence trees, and in analyzing biochemical structures.

2. NOTATIONS AND DEFINITIONS

A tree will be represented in terms of the postorder numbering of its nodes, where the postorder tree traversal is defined by the following recursive steps:

1. Visit each of the subtrees from left to right in the postorder sequence.
2. Visit the node.

The sequence of nodes generated by the postorder traversal is called the **postorder sequence** of the tree. Zhang and Shasha [11] have shown the advantages of this ordering over the other well-known orderings.

Let $T[i]$ be the ith node in the tree according to the left-to-right postorder numbering, and let $\delta(i)$ represent the postorder number of the leftmost leaf descendant of the subtree rooted at $T[i]$. Note that when $T[i]$ is a leaf, $\delta(i)=i$. $T[i \cdots j]$ represents the postorder forest induced by nodes $T[i]$ to $T[j]$ inclusive of tree T. $T[\delta(i) \cdots i]$ will be referred to as Tree(i). Size(i) is the number of nodes in Tree(i). An example of these terms is shown pictorially in Figure 1.

Finally, the father of $T[i]$ is denoted as $f(i)$ and $f^0(i)=i$, $f^1(i)=f(i)$, $f^2(i)=f(f^1(i))$, and so on, and using this, we define the set of ancestors of i as Anc(i) as:

$$\text{Anc}(i) = \{f^k(i) \mid 0 \leqslant k \leqslant \text{Depth}(i)\}.$$

2.1. *EDIT OPERATIONS AND DISTANCE BETWEEN TREES*

Let λ be the null node. It is distinct from μ, the null tree. An edit operation on a tree is either a node insertion, a node deletion, or a substitution of one node by another. Symbolically, an edit operation is represented as: $x \to y$ where x and y can either be a node value or λ. $x=\lambda$ and $y \neq \lambda$ represents an insertion; $x \neq \lambda$ and $y=\lambda$ represents a deletion; and $x \neq \lambda$ and $y \neq \lambda$ represents a substitution. Note that the case of $x=\lambda$ and $y=\lambda$ has not been defined because it is not needed. The formal definitions of these operations are described below.

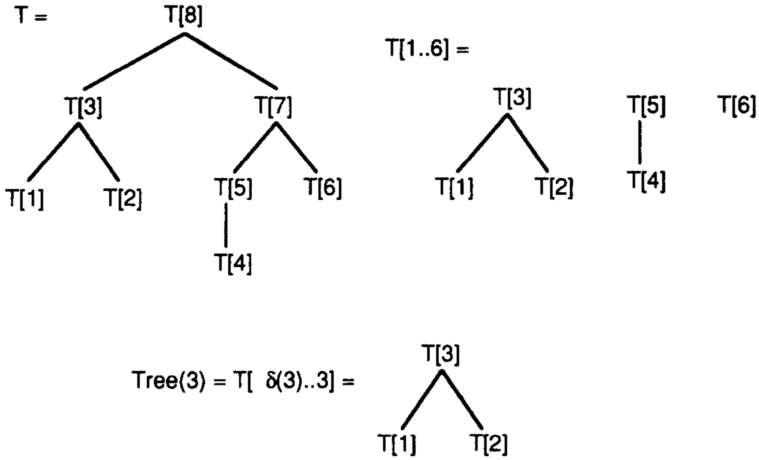

Fig. 1. An example of a tree, a postorder forest, a subtree, and the associated notations.

(i) Insertion of node x.

Node x will be inserted as a son of some node u of T. It may either be inserted with no sons or take as sons any subsequence of the sons of u. Formally, if u has sons $u_1, u_2, \ldots, u_k$, then for some $0 \leqslant i \leqslant j \leqslant k$, node u in the resulting tree will have sons $u_1, \ldots, u_i, x, u_j, \ldots, u_k$, and node x will have no sons if $j = i+1$, or else have sons $u_{i+1}, \ldots, u_{j-1}$. An example of this is in Figure 2.

(ii) A deletion of node y from a tree T. (See Figure 3).

If node y has sons $y_1, y_2, \ldots, y_k$ and node u, the father of y, has sons $u_1, u_2, \ldots, u_j$ with $u_i = y$, then node u in the resulting tree obtained by the

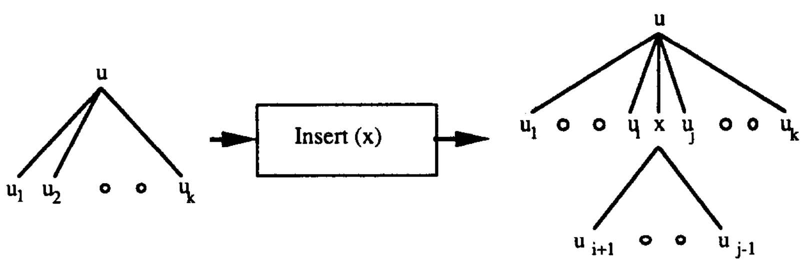

Fig. 2. An example of the insertion of a node.

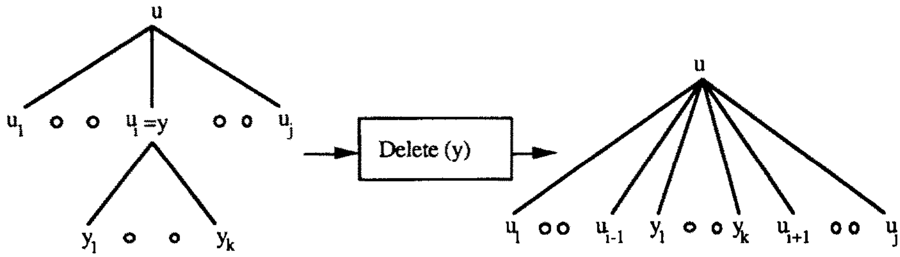

Fig. 3. An example of the deletion of a node.

deletion will have sons $u_1, u_2, \ldots, u_{i-1}, y_1, y_2, \ldots, y_k, u_{i+1}, \ldots, u_j$. The deletion of the root is not allowed if the root has more than one son.

(iii) Substitution of node x by node y in T.

In this case, node y in the resulting tree will have the same father and sons as node x in the original tree. An example of substitution is shown in Figure 4.

Let $d(x,y) \geqslant 0$ be the cost of transforming node x to node y. If $x \neq \lambda \neq y$, $d(x,y)$ will represent the cost of substitution of node x by node y. Similarly, $x \neq \lambda$, $y = \lambda$ and $x = \lambda$, $y \neq \lambda$ will represent the cost of deletion and insertion of node x and y, respectively. We assume that:

$$d(x,y) \geqslant 0; \qquad d(x,x) = 0; \tag{1}$$

$$d(x,y) = d(y,x); \tag{2}$$

and

$$d(x,z) \leqslant d(x,y) + d(y,z) \tag{3}$$

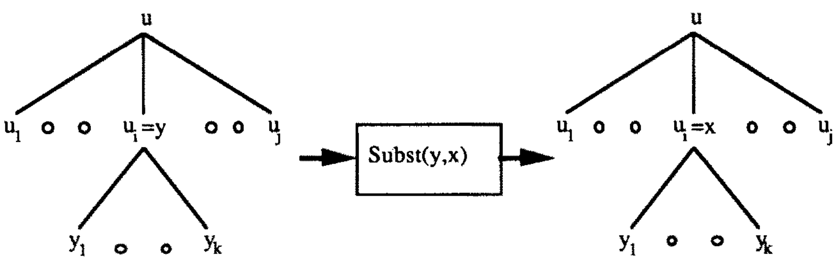

Fig. 4. An example of the substitution of a node by another.

where (3) ensures that no sequence of edit operations can achieve, at lower cost, the same effect as a single operation, and is thus a "triangular" inequality constraint.

Let S be a sequence $s_1, \ldots, s_k$ of edit operations. An S-derivation from A to B is a sequence of trees $A_0, \ldots, A_k$ such that $A = A_0$, $B = A_k$, and $A_{i-1} \to A_i$ via s_i for $1 \leqslant i \leqslant k$. We extend $d(*, *)$ to the sequence S by assigning $W(S) = \sum_{i=1}^{|S|} d(s_i)$. With the introduction of $W(S)$, the distance between T_1 and T_2 can be defined as follows:

$$D(T_1, T_2) = \mathrm{Min}\{W(S) | S \text{ is an } S\text{-derivation transforming } T_1 \text{ to } T_2.\}$$

It is easy to observe that:

$$D(T_1, T_2) \leqslant d(T_1[|T_1|], T_2[|T_2|]) + \sum_{i=1}^{|T_1|-1} d(T_1[i], \lambda) + \sum_{j=1}^{|T_2|-1} d(\lambda, T_2[j]).$$

2.2. *MAPPINGS BETWEEN TREES*

A Mapping is a description of how a sequence of edit operations transforms T_1 into T_2. A pictorial representation of a mapping is given in Figure 5. Informally, in it the following hold:

(i) Lines connecting $T_1[i]$ and $T_2[j]$ correspond to substituting $T_1[i]$ by $T_2[j]$.

(ii) Nodes in T_1 not touched by any line are to be deleted.

(iii) Nodes in T_2 not touched by any line are to be inserted.

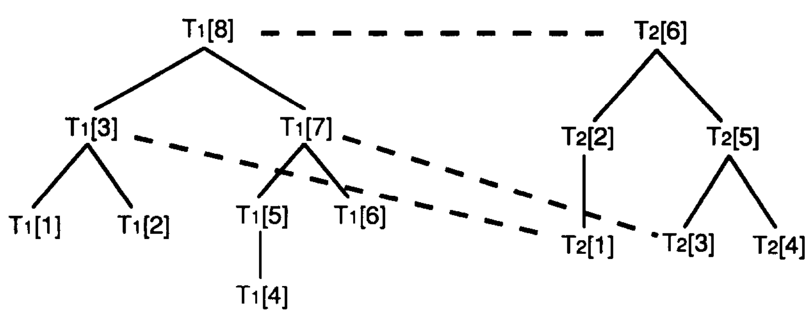

Fig. 5. An example of a mapping.

Formally, a mapping is a triple (M, T_1, T_2), where M is any set of pairs of integers (i, j) satisfying:

(i) $1 \leqslant i \leqslant |T_1|$, $1 \leqslant j \leqslant |T_2|$;

(ii) For any pair of (i_1, j_1) and (i_2, j_2) in M,

(a) $i_1 = i_2$ if and only if $j_1 = j_2$ (one-to-one).

(b) $T_1[i_1]$ is to the left of $T_1[i_2]$ if and only if $T_2[j_1]$ is to the left of $T_2[j_2]$. This is referred to as the Sibling Property.

(c) $T_1[i_1]$ is an ancestor of $T_1[i_2]$ if and only if $T_2[j_1]$ is an ancestor of $T_2[j_2]$. This is referred to as the Ancestor Property.

Whenever there is no ambiguity, we will use M to represent the triple (M, T_1, T_2), the mapping from T_1 to T_2. Let I, J be sets of nodes in T_1 and T_2, respectively, not touched by any lines in M. Then we can define the cost of M as follows:

$$\operatorname{cost}(M) = \sum_{(i,j)\in M} d(T_1[i], T_2[j]) + \sum_{i\in I} d(T_1[i], \lambda) + \sum_{j\in J} d(\lambda, T_2[j]).$$

Since mappings can be composed to yield new mappings [9, 11], the relationship between a mapping and a sequence of edit operations can now be specified.

LEMMA I. *Given S, an S-derivation $s_1, \ldots, s_k$ of edit operations from T_1 to T_2, there exists a mapping M from T_1 to T_2 such that* $\operatorname{cost}(M) \leqslant W(S)$. *Conversely, for any mapping M, there exists a sequence of editing operations such that* $W(S) = \operatorname{cost}(M)$.

Proof. Same as the proof of Lemma 2 in [11]. ∎

Due to the above lemma, we obtain

$$D(T_1, T_2) = \operatorname{Min}\{\operatorname{cost}(M) \mid M \text{ is a mapping from } T_1 \text{ to } T_2.\}$$

Thus, to search for the minimal cost edit sequence, we need to only search for the optimal mapping.

3. EDIT CONSTRAINTS

Consider the problem of editing T_1 to T_2, where $|T_1| = N$ and $|T_2| = M$. Editing a postorder-forest of T_1 into a postorder-forest of T_2 using exactly i insertions, e deletions, and s substitutions corresponds to editing $T_1[1 \cdots e+s]$ into $T_2[1 \cdots i+s]$. To obtain bounds on the magnitudes of variables,

i, e, s, we observe that they are constrained by the sizes of trees T_1 and T_2. Thus, if $r = e + s$, $q = i + s$, and $R = \text{Min}\{N, M\}$, these variables will have to obey the following constraints:

$$\max\{0, M - N\} \leqslant i \leqslant q \leqslant M; \qquad 0 \leqslant e \leqslant r \leqslant N; \qquad 0 \leqslant s \leqslant R.$$

Values of (i, e, s) which satisfy these constraints are termed *feasible values* of the variables. Let

$$H_i = \{ j | \max\{0, M - N\} \leqslant j \leqslant M \}, \qquad H_e = \{ j | 0 \leqslant j \leqslant N \}, \quad \text{and}$$

$$H_s = \{ j | 0 \leqslant j \leqslant \text{Min}\{ M, N \} \}.$$

H_i, H_e, and H_s are called the set of *permissible values* of i, e, and s.

Theorem I specifies the feasible triples for editing $T_1[1 \cdots r]$ to$T_2[1 \cdots q]$.

THEOREM I. *To edit $T_1[1 \cdots r]$, the postorder-forest of T_1 of size r, to $T_2[1 \cdots q]$, the postorder-forest of T_2 of size q, the set of feasible triples is given by* $\{(q - s, r - s, s) | 0 \leqslant s \leqslant \text{Min}\{M, N\}\}$.

Proof. Consider the constraints imposed on feasible values of i, e, and s. Since we are interested in editing $T_1[1 \cdots r]$ to $T_2[1 \cdots q]$, we have to consider only those triples (i, e, s) in which $i + s = r$ and $e + s = q$. But, the number of substitution can take any value from 0 to $\text{Min}\{r, q\}$. Therefore, for every value of s in this range, the feasible triples (i, e, s) must have exactly $r - s$ deletions since $r = e + s$. Similarly, the triples (i, e, s) must have exactly $q - s$ insertions since $q = s + i$. The result follows. ∎

An edit constraint is specified in terms of the number and type of edit operations that are required in the process of transforming T_1 to T_2. It is expressed by formulating the number and type of edit operations in terms of three sets Q_i, Q_e, and Q_s which are subsets of the sets H_i, H_e, and H_s defined above. The following examples could clarify the issue:

(a) To edit T_1 to T_2 performing no more than k deletions, the sets Q_s and Q_i are both equal to $\varnothing$, the null set, and $Q_e = \{j | j \in H_e,\ j \leqslant k\}$.

(b) To edit T_1 to T_2 performing exactly k_i insertions, k_e deletions, and k_s substitutions yields $Q_i = \{k_i\} \cap H_i$, $Q_e = \{k_e\} \cap H_e$, and $Q_s = \{k_s\} \cap H_s$.

THEOREM II. *Every edit constraint specified for the process of editing T_1 to T_2 is a unique subset of H_s.*

Proof. Let the constraint be specified by the sets Q_i, Q_e, and Q_s. Every element $j \in Q_i$ requires editing to be performed using exactly j insertions. Since $|T_2| = M$, from Theorem I, this requires that the number of substitutions be $M-j$. Similarly, if $j \in Q_e$, the edit transformation must constain exactly j deletions. Since $|T_1| = N$, Theorem I requires that $N-j$ substitutions be performed. Let

$$Q_e^* = \{N-j \mid j \in Q_e\} \quad \text{and} \quad Q_i^* = \{M-j \mid j \in Q_i\}.$$

Thus, for any constraint, the number of substitutions permitted is $Q_s \cap Q_e^* \cap Q_i^* \subseteq H_s$. ■

For example, let T_1 and T_2 be the trees shown in Figure 5. Suppose we want to transform T_1 to T_2 by performing at most 5 insertions, at least 3 substitutions, and exactly 3 deletions. Then

$$Q_i = \{0,1,2,3,4,5\}, \qquad Q_e = \{3\}, \quad \text{and} \quad Q_s = \{3,4,5,6\}.$$

Hence, $Q_e^* = \{5\}$, and $Q_i^* = \{1,2,3,4,5,6\}$, yielding $\tau = Q_s \cap Q_e^* \cap Q_i^* = \{5\}$. Hence, the optimal transformation must contain **exactly** 5 substitutions.

We shall refer to the edit distance subject to the constraint τ as $D_\tau(T_1, T_2)$. By definition, $D_\tau(T_1, T_2) = \infty$ if $\tau = \emptyset$, the null set. We now consider the computation of $D_\tau(T_1, T_2)$.

4. CONSTRAINED TREE-EDITING ALGORITHM

Since edit constraints can be written as unique subsets of H_s, we denote the distance between forest $T_1[i' \cdots i]$ and forest $T_2[j' \cdots j]$ subject to the constraint that exactly s substitutions are performed by Const_F_$Wt(T_1[i' \cdots i], T_2[j' \cdots j], s)$ or, more precisely, by Const_F_$Wt([i' \cdots i], [j' \cdots j], s)$. The distance between $T_1[1 \cdots i]$ and $T_2[1 \cdots j]$ subject to this constraint is given by Const_F_$Wt(i, j, s)$ since the starting index of both trees is unity. As opposed to this, the distance between the subtree rooted at i and the subtree rooted at j subject to the same constraint is given by Const_T_$Wt(i, j, s)$. The difference between Const_F_Wt and Const_T_Wt is subtle. Indeed,

$$\text{Const\_}T\text{\_}Wt(i,j,s) = \text{Const\_}F\text{\_}Wt(T_1[\delta(i) \cdots i], T_2[\delta(j) \cdots j], s).$$

These weights obey the following properties.

LEMMA II (a). *Let* $i_1 \in \text{Anc}(i)$ *and* $j_1 \in \text{Anc}(j)$; *then*

(i) Const_F_$Wt(\mu, \mu, 0) = 0$.

(ii) Const_F_$Wt(T_1[\delta(i_1) \cdots i], \mu, 0) =$ Const_F_$Wt(T_1[\delta(i_1) \cdots i-1], \mu, 0) + d(T_1[i], \lambda)$.

(iii) $\text{Const\_}F\text{\_}Wt(\mu, T_2[\delta(j_1)\cdots j], 0) = \text{Const\_}F\text{\_}Wt(\mu, T_2[\delta(j_1)\cdots j-1], 0) + d(\lambda, T_2[j])$.

(iv) $\text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], 0)$

$$= \text{Min}\begin{cases} \text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i-1], T_2[\delta(j_1)\cdots j], 0) + d(T_1[i], \lambda) \\ \text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j-1], 0) + d(\lambda, T_2[j]). \end{cases}$$

Proof. Case (i) requires no edit operations. In cases (ii) and (iii), the distance corresponds to the cost of deleting and inserting nodes in $T_1[\delta(i_1)\cdots i]$ and $T_2[\delta(j_1)\cdots j]$, respectively. In case (iv), since no substitution is allowed, the minimum cost mapping can be extended to $T_1[i]$ and $T_2[j]$ by either inserting $T_2[j]$ or deleting $T_1[i]$ only. Hence, the lemma. ■

LEMMA II (b). *Let* $i_1 \in \text{Anc}(i)$ *and* $j_1 \in \text{Anc}(j)$; *then*

(i) $\text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i], \mu, s) = \infty \quad \text{if } s > 0.$
(ii) $\text{Const\_}F\text{\_}Wt(\mu, T_2[\delta(j_1)\cdots j], s) = \infty \quad \text{if } s > 0.$
(iii) $\text{Const\_}F\text{\_}Wt(\mu, \mu, s) = \infty \quad \text{if } s > 0.$

Proof. Obvious since $s > 0$. ■

THEOREM III. *Let* $i_1 \in \text{Anc}(i)$ *and* $j_1 \in \text{Anc}(j)$; *then*

$$\text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], s)$$

$$= \text{Min}\begin{cases} \text{Const\_}F\text{\_}Wt([\delta(i_1)\cdots i-1], [\delta(j_1)\cdots j], s) + d(T_1[i], \lambda) \\ \text{Const\_}F\text{\_}Wt([\delta(i_1)\cdots i], [\delta(j_1)\cdots j-1], s) + d(\lambda, T_2[j]) \\ \displaystyle\min_{1 \leqslant s_2 \leqslant \text{Min}\{\text{Size}(i); \text{Size}(j); s\}} \begin{cases} \text{Const\_}F\text{\_}Wt([\delta(i_1)\cdots \delta(i)-1], \\ \quad [\delta(j_1)\cdots \delta(j)-1], s-s_2) \\ \quad + \text{Const\_}F\text{\_}Wt([\delta(i)\cdots i-1], \\ \quad [\delta(j)\cdots j-1], s_2-1) \\ \quad + d(T_1[i], T_2[j]) \end{cases} \end{cases}$$

Proof. We are trying to find a minimum cost mapping M between $T_1[\delta(i_1)\cdots i]$ and $T_2[\delta(j_1)\cdots j]$ using exactly s substitutions. The map can be extended to $T_1[i]$ and $T_2[j]$ in the following three ways:

(i) If $T_1[i]$ is not touched by any line in M, then $T_1[i]$ is to be deleted. Thus, since the number of substitutions in $\text{Const\_}F\text{\_}Wt(\cdot,\cdot,\cdot)$ remains unchanged, we have:

$$\begin{aligned}&\text{Const\_}F\_Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], s)\\&\quad=\text{Const\_}F\_Wt(T_1[\delta(i_1)\cdots i-1],\\&\qquad\qquad T_2[\delta(j_1)\cdots j], s)+d(T_1[i],\lambda).\end{aligned}$$

(ii) If $T_2[j]$ is not touched by any line in M, then $T_2[j]$ is to be inserted. Again, since the number of substitutions in Const_F_$Wt(\cdot,\cdot,\cdot)$ remains unchanged, we have:

$$\begin{aligned}&\text{Const\_}F\_Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], s)\\&\quad=\text{Const\_}F\_Wt(T_1[\delta(i_1)\cdots i],\\&\qquad\qquad T_2[\delta(j_1)\cdots j-1], s)+d(\lambda, T_2[j]).\end{aligned}$$

(iii) Consider the case when both $T_1[i]$ and $T_2[j]$ are touched by lines in M. Let (i,k) and (h,j) be the respective lines, i.e., (i,k) and $(h,j)\in M$. If $\delta(i_1)\leqslant h\leqslant\delta(i)-1$, then i is to the right of h, and so k, must be to the right of j by virtue of the sibling property of M. But this is impossible in $T_2[\delta(j_1)\cdots j]$ since j is the rightmost sibling in $T_2[\delta(j_1)\cdots j]$. Similarly, if i is a proper ancestor of h, then k must be a proper ancestor of j by virtue of the ancestor property of M. This is again impossible since $k\leqslant j$. So h has to equal i. By symmetry, k must equal j, so $(i,j)\in M$.

Now, by the ancestor property of M, any node in the subtree rooted at $T_1[i]$ can only be touched by a node in the subtree rooted at $T_2[j]$. This situation is depicted by Figure 6.

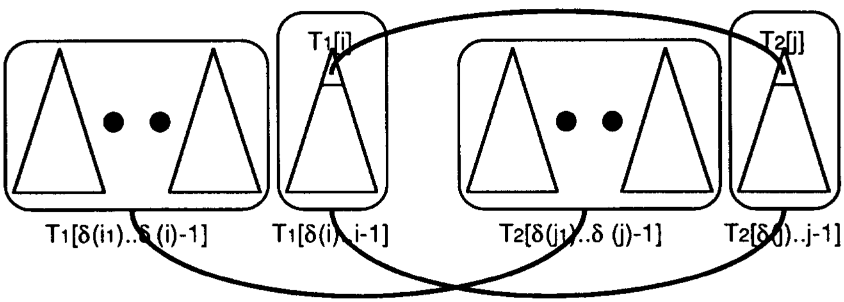

Fig. 6. Case 3 of Theorem III.

Since exactly s substitutions must be performed in this transformation, the total number of substitutions used in the subtransformation from $T_1[\delta(i_1)\cdots\delta(i)-1]$ to $T_2[\delta(j_1)\cdots\delta(j)-1]$ and the subtransformation from $T_1[\delta(i)\cdots i-1]$ to $T_2[\delta(j)\cdots j-1]$ must be equal to $s-1$ (the last substitution being the operation $T_1[i]\rightarrow T_2[j]$). Let s_2-1 be the number of substitutions used in the subtransformation from $T_1[\delta(i)\cdots i-1]$ to $T_2[\delta(j)\cdots j-1]$; then s_2 can take any value between 1 to $\mathrm{Min}\{\mathrm{Size}(i),\mathrm{Size}(j),s\}$. Hence, we have:

$$\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i],T_2[\delta(j_1)\cdots j],s)$$

$$= \min_{1\leqslant s_2\leqslant \mathrm{Min}\{\mathrm{Size}(i);\mathrm{Size}(j);s\}} \left\{ \begin{array}{l} \mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots\delta(i)-1], \\ T_2[\delta(j_1)\cdots\delta(j)-1],s-s_2) \\ \quad +\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i)\cdots i-1], \\ T_2[\delta(j)\cdots j-1],s_2-1) \\ \quad +d(T_1[i],T_2[j]) \end{array} \right.$$

Since these three cases exhaust the possible ways for yielding $\mathrm{Const\_}F\mathrm{\_}Wt\ (\delta(i_1)\cdots i,\delta(j_1)\cdots j,s)$, the minimum of these three costs yields the result. ■

It is easy to construct a recursive algorithm by using the above theorem. However, both the time and space complexities of the algorithm will be prohibitively large. Note that the main handicap with Theorem III is that when substitutions are involved, the quantity $\mathrm{Const\_}F\mathrm{\_}Wt\ (T_1[\delta(i_1)\cdots i],T_2[\delta(j_1)\cdots j],s)$ between the forests $T_1[\delta(i_1)\cdots i]$ and $T_2[\delta(j_1)\cdots j]$ is computed using the Const_*F*_*Wt*s of the forests $T_1[\delta(i_1)\cdots\delta(i)-1]$ and $T_2[\delta(j_1)\cdots\delta(j)-1]$ and the Const_*F*_*Wt*s of the remaining forests $T_1[\delta(i)\cdots i-1]$ and $T_2[\delta(j)\cdots j-1]$. If we note that, under certain conditions, the removal of a subforest leaves us with an entire tree, the computation is simplified. Thus, if $\delta(i)=\delta(i_1)$ and $\delta(j)=\delta(j_1)$ (i.e., both i and i_1 and j and j_1 span the same subtree), the subforests from $T_1[\delta(i_1)\cdots\delta(i)-1]$ and $T_2[\delta(j_1)\cdots\delta(j)-1]$ do not get included in the computation. However, if this is not the case, the $\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i],T_2[\delta(j_1)\cdots j],s)$ can be considered as a combination of the $\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots\delta(i)-1],T_2[\delta(j_1)\cdots\delta(j)-1],s-s_2))$ and the tree weight between the trees rooted at i and j, respectively, which is $\mathrm{Const\_}T\mathrm{\_}Wt(i,j,s_2)$. This is formally proved below.

THEOREM IV. *Let* $i_1 \in \mathrm{Anc}(i)$ *and* $j_1 \in \mathrm{Anc}(j)$. *Then the following is true*: *If* $\delta(i)=\delta(i_1)$ *and* $\delta(j)=\delta(j_1)$, *then*

$$\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], s)$$

$$=\mathrm{Min}\begin{cases}\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i-1], \\ T_2[\delta(j_1)\cdots j], s)+d(T_1[i],\lambda) \\ \mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i], \\ T_2[\delta(j_1)\cdots j-1], s)+d(\lambda, T_2[j]) \\ \mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots \delta(i)-1], \\ T_2[\delta(j_1)\cdots \delta(j)-1], s-1)+d(T_1[i], T_2[j])\end{cases}$$

otherwise,

$$\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i-1], T_2[\delta(j_1)\cdots j], s)$$

$$=\mathrm{Min}\begin{cases}\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i-1], \\ T_2[\delta(j_1)\cdots j], s)+d(T_1[i],\lambda) \\ \mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots i], \\ T_2[\delta(j_1)\cdots j-1], s)+d(\lambda, T_2[j]) \\ \underset{1\leqslant s_2\leqslant \mathrm{Min}\{\mathrm{Size}(i);\mathrm{Size}(j)s\}}{\mathrm{Min}}\begin{cases}\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots \delta(i)-1], \\ T_2[\delta(j_1)\cdots \delta(j)-1], s-s_2 \\ +\mathrm{Const\_}T\mathrm{\_}Wt(i,j,s_2)\end{cases}\end{cases}$$

Proof. By Theorem III, if $\delta(i)=\delta(i_1)$ and $\delta(j)=\delta(j_1)$, then the forests $T_1[\delta(i_1)\cdots \delta(i)-1]$ and $T_2[\delta(j_1)\cdots \delta(j)-1]$ are both empty. Thus,

$$\mathrm{Const\_}F\mathrm{\_}Wt(T_1[\delta(i_1)\cdots \delta(i)-1], T_2[\delta(j_1)\cdots \delta(j)-1], s-s_2)$$

$$=\mathrm{Const\_}F\mathrm{\_}Wt(\mu,\mu,s-s_2)$$

which is equal to zero if $s_2=s$, or is equal to ∞ if $s_2<s$. The first part of the theorem follows.

For the second part, since the distance is the cost of the minimal cost mapping, we know that:

$$\text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots i], T_2[\delta(j_1)\cdots j], s)$$
$$\leqslant \text{Const\_}F\text{\_}Wt(T_1[\delta(i_1)\cdots \delta(i)-1],$$
$$T_2[\delta(j_1)\cdots \delta(j)-1], s-s_2) + \text{Const\_}T\text{\_}Wt(i,j,s_2).$$

This is because the latter formula represents a particular mapping of $T_1[\delta(i)\cdots i]$ to $T_2[\delta(j)\cdots j]$ in which the forest $T_1[\delta(i_1)\cdots \delta(i)-1]$ is transformed into the forest $T_2[\delta(j_1)\cdots \delta(j)-1]$, and subsequently the tree rooted at i is transformed into the tree rooted at j. Thus, the second term in the above expression is a Const_*T*_*Wt* and not a Const_*F*_*Wt*. For the same reason we have:

$$\text{Const\_}T\text{\_}Wt(i,j,s_2)$$
$$\leqslant \text{Const\_}F\text{\_}Wt(T_1[\delta(i)\cdots i-1], T_2[\delta(j)\cdots j-1], s_2-1)$$
$$+ d(T_1[i], T_2[j]).$$

Theorem III and these two inequalities justify the substituting of Const_*T*_*Wt* (i, j, s_2) for the corresponding Const_*F*_*Wt* expressions, and the result follows. ■

Theorem IV suggests that we can use a dynamic programming flavored algorithm to solve the constrained tree-editing problem. First of all, note that the second part of Theorem IV suggests that if we are to compute the quantity Const_*T*_*Wt* (i_1, j_1, s), we have to have available the quantities Const_*T*_*Wt* (i, j, s_2) for all i and j and for all feasible values of $0 \leqslant s_2 \leqslant s$, where the nodes i and j are all the descendants of i_1 and j_1, except nodes on the path from i_1 to $\delta(i_1)$ and the nodes on the path from j_1 to $\delta(j_1)$. Furthermore, the theorem asserts that the distances associated with the nodes which are on the path from i_1 to $\delta(i_1)$ get computed (as a byproduct[1]) in the process of computing the Const_*F*_*Wt* between the trees rooted at i_1 and j_1. Indeed, the set of nodes for which the computation of Const_*T*_*Wt* must be done independently before the Const_*T*_*Wt* associated with their ancestors can be computed is called the set of "Essential_Nodes," and these are merely those nodes for which the computation would involve the second case of Theorem IV as opposed to

[1]The reason why this is obtained as a byproduct will be clear when the algorithm is formally presented. Indeed, whenever a Const_*F*_*Wt* is computed, if the forests are trees, it is retained as a Const_*T*_*Wt*.

the first. Thus, the Const_T_Wt can be computed for the entire tree if Const_T_Wt of the Essential_Nodes is computed, and using this stored value, the rest of the Const_T_Wts can be computed. This suggests a bottom-up approach for computing the Const_T_Wt between all pairs of subtrees. To formally present the algorithm, we define the set Essential_Nodes[2] of tree T as:

$$\text{Essential\_Nodes}(T) = \{k | \text{there exists no } k' > k \text{ such that } \delta(k) = \delta(k')\}.$$

That is, if k is in Essential_Nodes(T), then either k is the root or k has a left sibling. Intuitively, this set will be the roots of all subtrees of tree T that need separate computations. Thus, for the trees in Figure 5, Essential_Nodes(T_1) = {2, 6, 7, 8} and Essential_Nodes(T_2) = {4, 5, 6}.

The function $\delta()$ and the set Essential_Nodes() can be computed in linear time. We assume that these are stored in arrays δ[ ], and Essential_Nodes [ ], respectively. Furthermore, we assume that the elements in Essential_Nodes [ ] are sorted as per the postorder representation.

ALGORITHM T_Weights

INPUT: Trees T_1 and T_2 and the set of elementary edit distances.

OUTPUT: Const_T_Wt (i, j, s), $1 \leqslant i \leqslant |T_1|$, $1 \leqslant j \leqslant |T_2|$, and $1 \leqslant s \leqslant \text{Min}\{|T_1|, |T_2|\}$.

ASSUMPTION: **Preprocess** ($\mathbf{T_1}, \mathbf{T_2}$) yields the δ[ ] and Essential_Nodes [ ] arrays for both trees. These quantities are assumed to be global.

BEGIN

Preprocess (T_1, T_2);
FOR $i' = 1$ to |Essential_Nodes$_1$[ ]|**DO**
  FOR $j' = 1$ to |Essential_Nodes$_2$[ ]|**DO**
    i = Essential_Nodes$_1$[i'];
    j = Essential_Nodes$_2$[j'];

[2]The set of nodes which we refer to as the set of Essential_Nodes happens to be exactly the same as the set of nodes defined in [11] as the LR_keyroots set. Although these sets are identical, the implication of a node being in the sets is slightly different. In [11], $i \in LR$_keyroots[T_1] and $j \in LR$_keyroots[T_2] implies that the corresponding tree weights associated with the trees rooted at these nodes need precomputation. In our case, $i \in$ Essential_Nodes[T_1] and $j \in$ Essential_Nodes[T_2] implies that the corresponding **constrained** tree weights rooted at these trees need precomputation, and that this precomputation must be achieved for all feasible values of s which are relevant.

Compute_Const_T_Wt (i, j);
ENDFOR
ENDFOR
END.

In the succeeding computation, we shall attempt to evaluate Const_T_Wt (i, j, s) and store it in a **permanent** three-dimensional array Const_T_Wt. From Theorem IV, we observe that to compute the quantity Const_T_Wt (i, j, s), the quantities which are involved are precisely the terms Const_F_Wt $([\delta(i) \cdots h], [\delta(j) \cdots k], s')$ defined for a particular input pair (i, j), where h and k are the internal nodes of $\text{Tree}_1(i)$ and $\text{Tree}_2(j)$ satisfying $\delta(i) \leqslant h \leqslant i$, $\delta(j) \leqslant k \leqslant j$, and where s' is in the set of feasible values and satisfies $0 \leqslant s' \leqslant s = \text{Min}\{|\text{Tree}_1(i)|, |\text{Tree}_2(j)|\}$. Our intention is to store **these** values using a single **temporary** three-dimensional array Const_F_$Wt[\cdot,\cdot,\cdot]$. But in order to achieve this, it is clear that the base indices of the temporary three-dimensional array Const_F_$Wt[\cdot,\cdot,\cdot]$ will have to be adjusted each time the procedure is invoked so as to permit us the possibility of utilizing the **same** memory allocations repeatedly for every computation. This is achieved by assigning the base values b_1 and b_2 as $b_1 = \delta_1(i) - 1$ and $b_2 = \delta_2(j) - 1$.

Thus, for a particular input pair (i, j), the same memory allocations Const_F_$Wt[\cdot,\cdot,\cdot]$ can be used to store the values in each phase of the computation by assigning:

$$\text{Const\_}F\text{\_}Wt[x_1, y_1, s']$$
$$= \text{Const\_}F\text{\_}Wt\ \big([\delta(i) \cdots \delta(i) + x_1 - 1], [\delta(j) \cdots \delta(j) + y_1 - 1], s'\big)$$

for all $1 \leqslant x_1 \leqslant i - \delta(i) + 1$, $1 \leqslant y_1 \leqslant j - \delta(j) + 1$.

Consequently, we note that for every x_1, y_1, and s' in any intermediate step in the algorithm, the quantity Const_T_Wt () that has to be stored in the permanent array can be obtained by incorporating these base values again, and has the form Const_T_$Wt[x_1 + b_1, y_1 + b_2, s']$.

After the array Const_T_$Wt[\cdot,\cdot,\cdot]$ has been computed, the distance $D_\tau(T_1, T_2)$ between the trees T_1 and T_2 subject to the constraint τ can be directly evaluated using the ALGORITHM Constrained_Tree_Distance presented thereafter.

ALGORITHM Compute_Const_T_Wt
INPUT: Indexes i, j and the quantities assumed global in **T_Weights.**
OUTPUT: Const_T_$Wt[i_1, j_1, s]$, $\delta_1(i) \leqslant i_1 \leqslant i$, $\delta_2(j) \leqslant j_1 \leqslant j, 0 \leqslant s \leqslant \text{Min}\{\text{Size}(i), \text{Size}(j)\}$.

BEGIN

$N = i - \delta_1(i) + 1$; /* size of subtree rooted at $T_1[i]$ */

$M = j - \delta_2(j) + 1$; /* size of subtree rooted at $T_2[j]$ */

$R = \text{Min}\{M, N\}$

$b_1 = \delta_1(i) - 1$; /* adjustment for nodes in subtree rooted at $T_1[i]$ */

$b_2 = \delta_2(j) - 1$; /* adjustment for nodes in subtree rooted at $T_2[j]$ */

Const_F_Wt[0][0][0] = 0; /* Initialize Const_F_Wt */

FOR $x_1 = 1$ to N **DO**

Const_F_$Wt[x_1][0][0]$

$= Const$_F_Wt$[x_1 - 1][0][0] + d(T_1[x_1 + b_1] \to \lambda)$;

Const_T_$Wt[x_1 + b_1][0][0]$ = Const_F_$Wt[x_1][0][0]$;

ENDFOR

FOR $y_1 = 1$ to M **DO**

Const_F_$Wt[0][y_1][0]$ = Const_F_$Wt[0][y_1 - 1][0] + d(\lambda \to T_2[y_1 + b_2])$;

Const_T_$Wt[0][y_1 + b_2][0]$ = Const_F_$Wt[0][y_1][0]$;

ENDFOR

FOR $s = 1$ to R **DO**

Const_F_$Wt[0][0][s] = \infty$;

Const_T_$Wt[0][0][s]$ = Const_F_$Wt[0][0][s]$;

ENDFOR

FOR $x_1 = 1$ to N **DO**

FOR $y_1 = 1$ to M **DO**

Const_F_$Wt[x_1][y_1][0]$

$$= \text{Min}\begin{cases} \text{Const\_}F\text{\_}Wt[x_1][y_1 - 1][0] + d(\lambda \to T_2[y_1 + b_2]) \\ \text{Const\_}F\text{\_}Wt[x_1 - 1][y_1][0] + d(T_1[x_1 + b_1] \to \lambda) \end{cases}$$

Const_T_$Wt[x_1 + b_1][y_1 + b_2][0]$ = Const_F_$Wt[x_1][y_1][0]$;

ENDFOR

ENDFOR

FOR $x_1 = 1$ to N **DO**

FOR $s = 1$ to R **DO**

Const_F_$Wt[x_1][0][s] = \infty$;

Const_T_$Wt[x_1 + b_1][0][s]$ = Const_F_$Wt[x_1][0][s]$;

ENDFOR

ENDFOR

FOR $y_1 = 1$ to M **DO**

FOR $s = 1$ to R **DO**

Const_F_$Wt[0][y_1][s] = \infty$;

Const_T_$Wt[0][y_1 + b_2][s]$ = Const_F_$Wt[0][y_1][s]$;

ENDFOR

```
ENDFOR
FOR x_1 = 1 to N DO
   FOR y_1 = 1 to M DO
      FOR s = 1 to R DO
         IF δ_1(x_1 + b_1) = δ_1(x) and δ_2(y_1 + b_2) = δ_2(y) THEN
            Const_F_Wt[x_1][y_1][s]
                   ⎧ Const_F_Wt[x_1 - 1][y_1][s] + d(T_1[x_1 + b_1] → λ)
              = Min⎨ Const_F_Wt[x_1][y_1 - 1][s] + d(λ → T_2[y_1 + b_2])
                   ⎪ Const_F_Wt[x_1 - 1][y_1 - 1][s - 1]
                   ⎩    + d(T_1[x_1 + b_1] → T_2[y_1 + b_2])
            Const_T_Wt[x_1 + b_1][y_1 + b_2][s] = Const_F_Wt[x_1][y_1][s];
         ELSE
            Const_F_Wt[x_1][y_1][s]
                 ⎧ Const_F_Wt[x_1 - 1][y_1][s] + d(T_1[x_1 + b_1] → λ)
                 ⎪ Const_F_Wt[x_1][y_1 - 1][s] + d(λ → T_2[y_1 + b_2])
            = Min⎨                        ⎧ Const_F_Wt[a][[b][s - s_2]]
                 ⎪          Min           ⎨
                 ⎩ 1 ≤ s_2 ≤ Min{c; d; s} ⎩    + Const_T_Wt[x_1 + b_1][y_1 + b_2][s_2]
              where a = δ_1(x_1 + b_1) - 1 - b_1, b = δ_2(y_1 + b_2) - 1 - b_2,
                    c = Size(x_1 + b_1) and d = Size(y_1 + b_2).
            ENDIF
         ENDFOR
      ENDFOR
   ENDFOR
END.
```

ALGORITHM Constrained_Tree_Distance

INPUT: The array Const_T_$Wt[\cdot,\cdot,\cdot]$ computed using Algorithm **T_Weights**, and the constraint set τ.

OUTPUT: The constrained distance $D_\tau(T_1, T_2)$.

```
BEGIN
      D_τ(T_1, T_2) = ∞;
      FOR all s ∈ τ DO
         D_τ(T_1, T_2) = Min{D_τ(T_1, T_2), Const_T_Wt[|T_1|][|T_2|][s]}
      ENDFOR
END.
```

THEOREM V. *The basic algorithm is correct.*

Proof. The proof follows along the lines of the proof for the unconstrained distance [11]. We prove that the invariants hold for all (i, j) such that $i \in$ Essential_Nodes(T_1) and $j \in$ Essential_Nodes(T_2):

(i) Immediately before the computation of Const_T_$Wt(i,j,s)$ for all valid values of s, all distances Const_T_$Wt(h,k,s')$ are available, where $\delta(i) \leqslant h \leqslant i$, $\delta(j) \leqslant k \leqslant j$, $0 \leqslant s' \leqslant \text{Min}\{\text{Size}(h), \text{Size}(k)\}$, and either $\delta(i) \neq \delta(h)$ or $\delta(j) \neq \delta(k)$. This is true because the values of h and k are either contained in Essential_Nodes(T_1) and Essential_Nodes(T_2), respectively, or can be computed from them. Thus, Const_T_$Wt(h,k,s')$ is available if h is in the subtree of Tree(i), but not in the path from $\delta(i)$ to i, and k is in the subtree of Tree(j), but not in the path from $\delta(j)$ to j.

(ii) After the computation of Const_T_$Wt(i, j, s)$, every Const_T_$Wt(h,k,s')$ is available, where $\delta(i) \leqslant h \leqslant i$, $\delta(j) \leqslant k \leqslant j$, and $0 \leqslant s' \leqslant \text{Min}\{\text{Size}(h), \text{Size}(k)\}$. Thus, every Const_$T$_$Wt(h,k,s')$ is available, including those nodes in the path from $\delta(i)$ to i, and in the path from $\delta(j)$ to j.

We will show that if (i) is true, then (ii) is true. From Theorem IV and (i), we know that all required subtree-to-subtree distances are available. We compute each Const_T_$Wt(h,k,s')$, where $\delta(h) = \delta(i)$, $\delta(k) = \delta(j)$, and $0 \leqslant s' \leqslant \text{Min}\{\text{Size}(h), \text{Size}(k)\}$ using the **IF** part of Theorem IV, and subsequently include them in the permanent array Const_T_Wt. So (ii) holds.

Let us show that (i) always holds. Suppose $\delta(h) \neq \delta(i)$. Let h' be the lowest ancestor of h such that $h' \in$ Essential_Nodes (T_1). Clearly, such an ancestor exists since the root of T_1 is in Essential_Nodes [ ]. Since $\delta(h') = \delta(h) \neq \delta(i)$, we conclude that $h' \neq i$. Further, $i \in$ Essential_Nodes (T_1), we have $h' \leqslant i$. Combining the latter two assertions, we obtain $h' < i$. Similarly, we have $k' < j$. This means that Const_T_$Wt(h',k',s'')$ will be computed for all valid values of s'' before Const_T_$Wt(i,j,s)$ because elements in Essential_Nodes (T_1) and Essential_Nodes (T_2) are stored in their increasing orders. Hence, Const_T_$Wt(h,k,s')$ is available for all valid values of s' after Const_T_$Wt(h',k',s'')$ is computed for all valid values of s''. So (i) always holds, and this proves the theorem. ■

Note that our algorithm is similar in spirit to the one independently reported in [12], although the latter, just as in [5], deals with the problem by utilizing the number of insertions permitted as the "free" variable. In this case, however, we have chosen to use the number of substitutions as the free variable. This makes it differ from the philosophies of both [5] and [12], but helps us to retain the same underlying principles of tree editing as in [11], in which the substitution operation has to be handled differently from both the insertion and deletion operations.

The space required by our scheme is obviously $O(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\})$. To analyze the time complexity, we use the following results which are a consequence of the equivalence of the sets Essential–Nodes(T) and

LR_keyroots(T) defined in [11] (see the proof of Lemma 6 of [11]):

(i) Cardinality (Essential_Nodes(T)) $\leqslant$ |Leaves(T)|
(ii)

$$\sum_{i=1}^{|\text{Essential\_Nodes}(T)|} \text{Size}(i) \leqslant |T| * \text{Min}\{\text{Depth}(T), \text{Leaves}(T)\} \quad (4)$$

THEOREM VI. *If* Span(T) *is the* Min{Depth(T), Leaves(T)}, *the time complexity of our algorithm is*:

$$O\left(|T_1| * |T_2| * \left(\text{Min}\{|T_1|, |T_2|\}\right)^2 * \text{Span}(T_1) * \text{Span}(T_2)\right).$$

Proof. We first observe that the preprocessing takes linear time. Also, note that if the array Const_T_Wt is computed, the constrained tree-editing distance, $D_\tau(*, *)$, can be computed using Algorithm Constrained_Tree_Distance in time $|\tau|$, which in the worst case is $O(\text{Min}\{|T_1|, |T_2|\})$ which is also linear. Hence, the dominant term involves computing the array Const_T_$Wt(i, j, s)$ for all relevant i, j, and s. This algorithm involving the subtrees rooted at i and j involves: $O(\text{Size}(i) * \text{Size}(j) * \text{Min}\{\text{Size}(i), \text{Size}(j)\}^2)$ computation. Therefore, the time required is:

$$\sum_{i=1}^{|\text{Essential\_Nodes}(T_1)|} \sum_{j=1}^{|\text{Essential\_Nodes}(T_2)|} \text{Size}(i) * \text{Size}(j) * \text{Min}\{\text{Size}(i), \text{Size}(j)\}^2$$

$$\leqslant \text{Min}\{|T_1|, |T_2|\}^2 * \sum_{i=1}^{|\text{Essential\_Nodes}(T_1)|} \text{Size}(i) * \sum_{j=1}^{|\text{Essential\_Nodes}(T_2)|} \text{Size}(j).$$

The result follows as a consequence of (4). ■

5. CONCLUSIONS

In this paper, we have considered the problem of editing a tree T_1 to a tree T_2 subject to a set of specified edit constraints. The edit constraint is fairly arbitrary, and can be specified in terms of the number and type of edit operations desired in optimal transformation just as in the case of strings [5, 6]. Given the trees T_1 and T_2, an intermediate quantity which is the array of constrained edit distances Const_T_$Wt(i, j, s)$ can be computed using dynamic programming, whence $D_\tau(T_1, T_2)$ can be evaluated in linear time. If for a tree T, Span(T) is defined as the Min{Depth(T), Leaves(T)}, the scheme to compute this array requires $O(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\}^2 * \text{Span}(T_1) * \text{Span}(T_2))$ time. The space required for this computation is cubic.

We are currently investigating the use of constrained edit distances between trees in the pattern recognition of noisy (garbled) trees, and in analyzing biochemical structures.

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. A preliminary version of some of these results is found in the Proceedings of the Second International Computer Sciences Conference: Data and Knowledge Engineering: Theory and Applications, Hong Kong, December 1992, pp. 409–415.

REFERENCES

1. R. L. Kashyap and B. J. Oommen, A common basis for similarity measures involving two strings, *Inter. J. Computer Math.* 13:17–40 (1983).
2. R. L. Kashyap and B. J. Oommen, Spelling correction using probabilistic methods, *Pattern Recognition Letters* 2:147–154 (1984).
3. S. Y. Lu, A tree-to-tree distance and its application to cluster analysis, *IEEE Trans. Pattern Anal. and Mach. Intell.* PAMI-1(2):219–224 (1979).
4. S. Y. Lu, A tree-matching algorithm based on node splitting and merging, *IEEE Trans. Pattern Anal. and Mach. Intell.* PAMI-6(2):249–256 (1984).
5. B. J. Oommen, Constrained string editing, *Inform. Sci.* 40:267–284 (1986).
6. B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. Pattern Anal. and Mach. Intell.* PAMI-9(5):676–685 (1987).
7. D. Sankoff and J. B. Kruskal, *Time Warps, String Edits, and Macromolecules: Theory and Practice of Sequence Comparison*, Addison-Wesley, 1983.
8. S. M. Selkow, The tree-to-tree editing problem, *Inform. Process. Letters* 6(6):184–186 (1977).
9. K. C. Tai, The tree-to-tree correction problem, *J. Assoc. Comput. Mach.* 26:422–433 (1979).
10. R. A. Wagner and M. J. Fischer, The string-to-string correction problem, *J. Assoc. Comput. Mach.* 21:168–173 (1974).
11. K. Zhang and D. Shasha, Simple fast algorithms for the editing distance between trees and related problems, *SIAM J. Comput.* 18(6):1245–1262 (1989).
12. K. Zhang, Constrained string and tree editing distance, *Proceedings of the IASTED International Symposium*, New York, 1990, pp. 92–95.

Received 23 April 1992; revised 15 June 1992, 19 November 1992