CONSTRAINED TREE EDITING

B. John Oommen and William Lee

SCS-TR-199, DECEMBER 1991

School of Computer Science, Carleton University
Ottawa, Canada, KIS 5B6

CONSTRAINED TREE EDITING*

B. John Oommen and William Lee
School of Computer Science
Carleton University
Ottawa ; 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 [3,4,7,8,9,11]. 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 :

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

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

* Partially supported by the Natural Sciences and Engineering Research Council of Canada.

I. INTRODUCTION

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

Similar to the string-to-string editing problem [1, 2, 5-7, 10], the tree-to-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 of nodes needed to transform one tree into the other. Possible applications of the tree-to-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. As an example, consider the secondary structure comparisons of R.N.A. in the study of macromolecules. It is well known that the secondary structure of R.N.A. 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 R.N.A. to proteins. Thus comparisons among these secondary structures are necessary to understand the comparative functionality of different R.N.As.

Unlike the string-editing problem which has been well-developed, only 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. In 1979, Tai [9] presented another algorithm in which insertions and deletions could take place at any node within the tree except the root. The algorithm of Lu [3], on the other hand, did not solve this problem for trees of more than two levels. 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 consider the editing of one tree, say T_1, and transforming it to another tree 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 transformation. For the sake of clarity we present some examples of constrained editing.

EXAMPLE I.

Below are some constrained editing problems :

(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 will first present a consistent method of specifying arbitrary edit constraints. This method is analogous to the one shown in [5] which 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. The latter quantity is computed by first evaluating elements of a three-dimensional array using dynamic programming techniques, and then combining certain elements of this three-dimensional array to yield $D_\tau(T_1, T_2)$. If Span(T) is defined as the Min{Depth(T), Leaves(T)}. It will be shown that the algorithm requires $O(|T_1| * |T_2| * Span(T_1) * Span(T_2))$ time and $O(|T_1| * |T_2| * Min\{|T_1|, |T_2|\})$ space. Using this three-dimensional array, the optimal sequence of edit operations required to edit T_1 to T_2 subject to the constraint τ can then be obtained by backtracking.

II. NOTATIONS AND DEFINITIONS

2.1 Notation

A tree will be represented in terms of the postorder numbering of its nodes. Zhang and Shasha [11] have shown the advantages of this ordering over the other well-known orderings. Let T[i] be the i^{th} 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..j] represents the postorder forest induced by nodes T[i] to T[j] inclusive, of tree T. T[δ(i)..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 I.

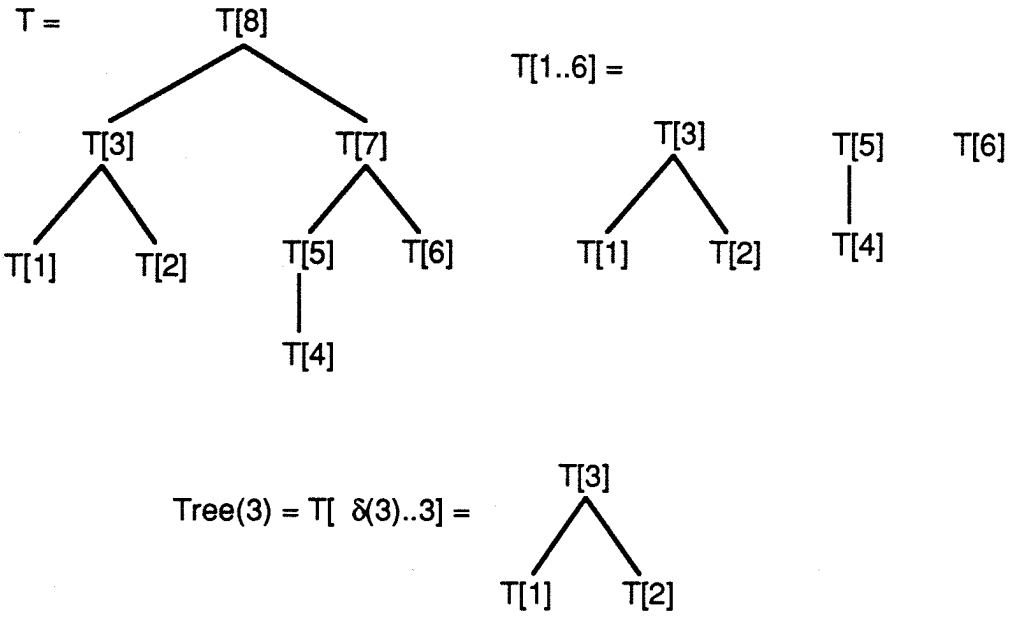

Figure I : An example of a tree, a postorder forest, a subtree, and the associated notations.

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 :

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

In the body of the paper we will use Figure I extensively to explain various terms and notational details.

2.2 Edit Operations and Distance between trees

Let λ be the null node distinct from μ, the null tree. An edit operation on a tree is either an insertion of a node, a deletion of a node, or a substitution of one node by another. In terms of notation, an edit operation is represented symbolically as : $x \rightarrow y$ where x and y can either be a node value or λ, the null node. $x = \lambda$ 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 each operation are described as follows :

(i) An insertion of node x into tree T.

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 $0 \leq i \leq j \leq k$, 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 insertion is shown in Figure II.

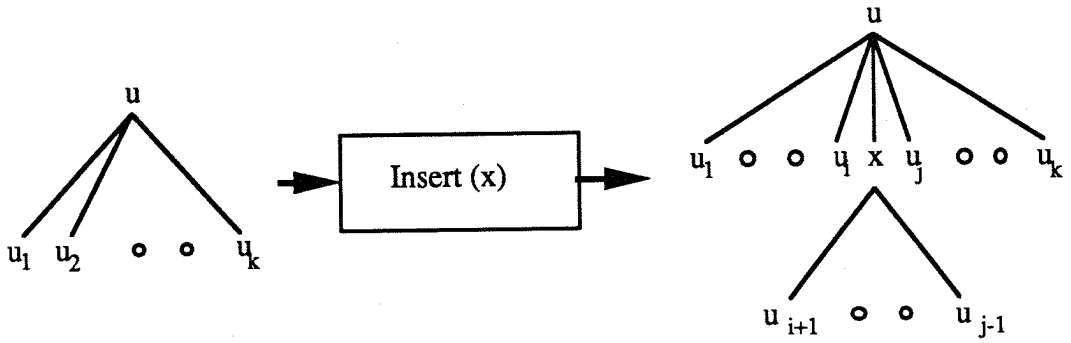

Figure II : An example of the insertion of a node.

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

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 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. An example of deletion is shown in the following figure.

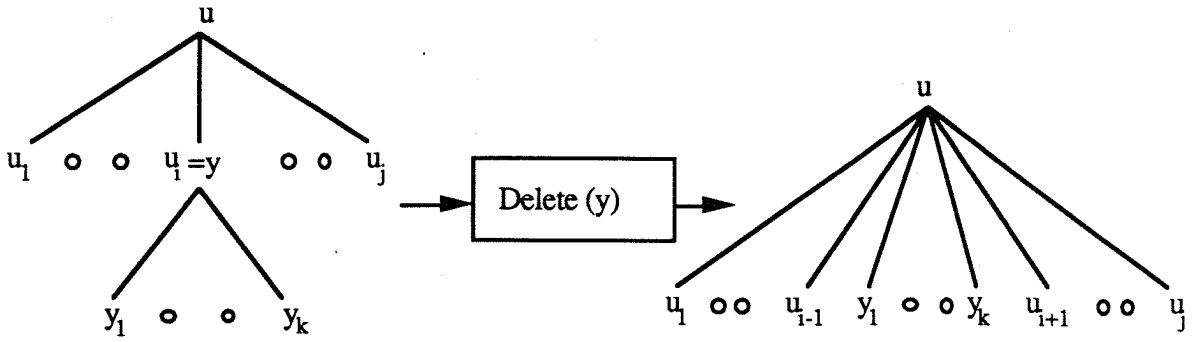

Figure III : An example of the deletion of a node.

(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 IV.

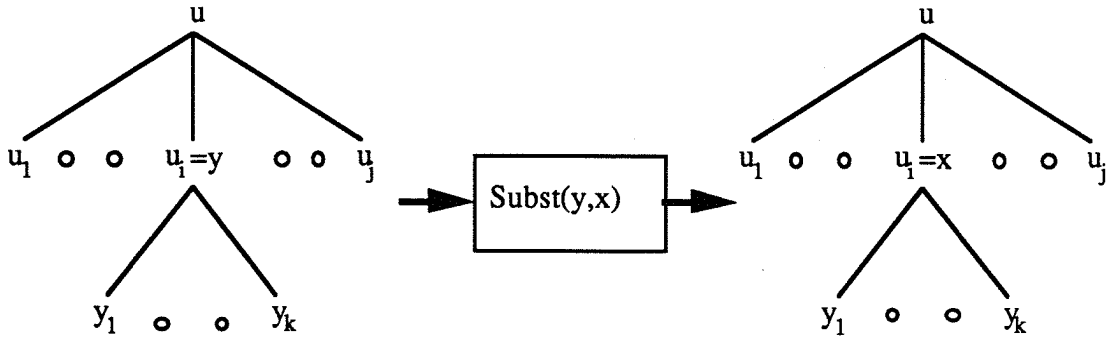

Figure IV : An example of the substitution of a node by another.

Let $d(x,y) \geq 0$ 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. Without loss of generality, we assume that

$$d(x,y) \geq 0;\ d(x,x) = 0; \qquad (1)$$

$$d(x,y) = d(y,x); \qquad (2)$$

and $$d(x,z) \leq d(x,y) + d(y,z) \qquad (3)$$

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 $s_1, ..., s_k$ of edit operations. An S-derivation from A to B is a sequence of trees $A_0, ..., A_k$ such that $A = A_0$, $B = A_k$, and $A_{i-1} \rightarrow A_i$ via s_i for $1 \leq i \leq k$. 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 :

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

It is easy to observe that :

$$D(T_1,T_2) \leq d(T_1[|T_1|],T_2[|T_2|]) + \sum_{i=1}^{|T_1|-1} d(T_1[i],\lambda) + \sum_{j=1}^{|T_2|-1} d(\lambda,T2[j])$$

by considering the following S-derivation or edit sequence : delete all nodes of T_1 except the root of T_1; substitute the root of T_1 by the root of T_2; finally insert back all nodes of T_2 except the root. Clearly this is an upper bound for the value of $D(T_1,T_2)$.

2.3 Mappings between Trees

A Mapping is a description of how a sequence of edit operations transforms T_1 into T_2. Informally, we can describe a mapping as follows :

(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.

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

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

(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.

A pictorial representation of a mapping is given in Figure V for an example sequence. Note that T_1 is exactly the same tree used in Figure I. 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 :

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

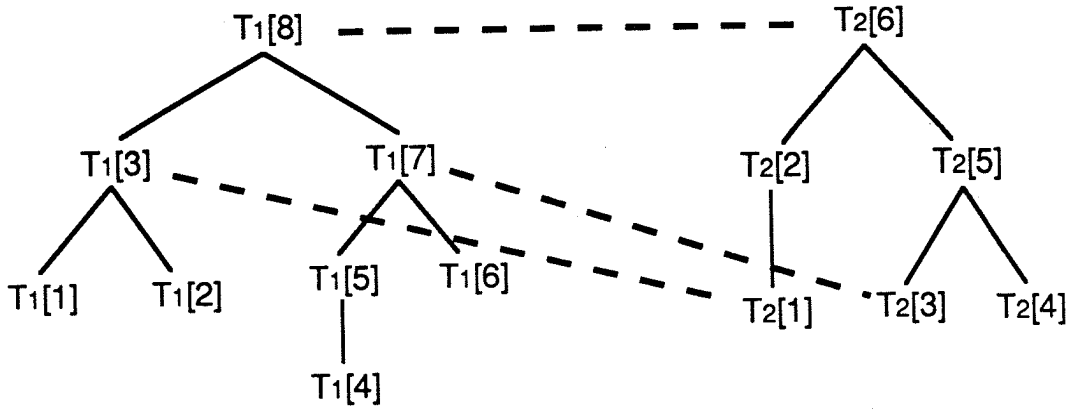

Figure V : An example of a mapping.

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 $s_1, ..., s_k$ of edit operations from T_1 to T_2, there exists a mapping M from T_1 to T_2 such that cost (M) ≤ W(S). Conversely, for any mapping M, there exists a sequence of editing operations such that W(S) = cost (M).

PROOF : Same as the proof of Lemma 2 in [11]. ****

Due to the above lemma, we obtain

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

In other words, the problem of searching for the minimal cost edit sequence has been reduced to a search for the minimal cost mapping.

III. EDIT CONSTRAINTS

3.1 Permissible and Feasible Edit Operations

Consider the problem of editing T_1 to T_2, where $|T_1| = N$ and $|T_2| = M$. Suppose we edit a postorder-forest of T_1 into a postorder-forest of T_2 using exactly i insertions, e deletions, and s substitutions. Since the number of edit operations has been specified, this corresponds to editing $T_1[1..e+s]$ into $T_2[1..i+s]$.

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=Min{N,M}, these variables will have to obey the following constraints :

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

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

$$H_i = \{j \mid \max\{0,M-N\} \le j \le M\},$$
$$H_e = \{j \mid 0 \le j \le N\},$$
$$H_s = \{j \mid 0 \le j \le \text{Min}\{M,N\}\}.$$

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

The following theorem specifies the permitted forms of feasible triples which are encountered in editing $T_1[1..r]$, the postorder-forest of T_1 of size r, to $T_2[1..q]$, the postorder-forest of T_2 of size q.

THEOREM I.

To edit $T_1[1..r]$, the postorder-forest of T_1 of size r, to $T_2[1..q]$, the postorder-forest of T_2 of size q, the set of feasible triples is given by :

$$\{(q-s, r-s, s) \mid 0 \le s \le \text{Min}\{M,N\}\}.$$

PROOF :

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

3.2 Specification of Edit Constraints

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 in last subsection. We will now clarify this using the three constraints given in the following example.

EXAMPLE II.

(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 ϕ, the null set. Further,

$$Q_e = \{j \mid j \in H_e, j \le k\}$$

(b) To edit T_1 to T_2 performing exactly k_i insertions, k_e deletions, and k_s substitutions yields

$$Q_i = \{k_i\} \cap H_i, \quad Q_e = \{k_e\} \cap H_e, \quad \text{and} \quad Q_s = \{k_s\} \cap H_s.$$ ****

THEOREM II.

Every edit constraint specified for the process of editing T_1 to T_2 can be written as 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 1, this requires that the number of substitutions be M-j.

Similarly, if $j \in Q_e$, the edit transformation must contain exactly j deletions. Since $|T_1| = N$, Theorem 1 requires that N-j substitutions be performed. Let

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

Clearly, for any arbitrary constraint, the number of substitutions that are permitted is given by

$$Q_s \cap Q_e^* \cap Q_i^*$$

which is obviously a subset of H_s. ****

Example III.

Let T_1 and T_2 be 2 trees as shown in Figure V. Suppose we want to transform T_1 to T_2 by performing at most 5 insertions, at least 3 substitutions and exactly 1 deletion. Then

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

Hence

$$Q_e^* = \{3\}, \qquad Q_i^* = \{1,2,3,4,5,6\}$$

Thus,

$$\tau = Q_s \cap Q_e^* \cap Q_i^* = \{3\}.$$

Hence, the optimal transformation must contains ***exactly*** 3 substitutions. ****

At this juncture, it is appropriate to distinguish between the constrained string editing problem and the constrained tree editing problem. The points of similarity are few. First of all, the similarity includes the technique by which the permissible and feasible edit operations can be defined. Indeed, by virtue of this similarity, Theorem I can be seen to be analogous to the corresponding theorem (Theorem I of [5]) in the constrained string editing problem. The second similarity involving the specification of the edit constraints. In the case of the constrained string editing problem, this set was defined in terms of the subset of the number of the feasible insertions. It is interesting to note that it could, just as effectively, have been specified as a subset of the feasible substitutions. In this case, we have opted to use the latter formulation primarily because the postorder representation permits us to always increase the number of substitutions performed more directly.

Although the similarities are few, the differences between these two problems are far more pronounced. First of all, it should be noted that the constrained tree editing problem is not even well-defined if the mappings are not consistent. Thus, for every edit S-derivation, not only must the number of substitutions be feasible, but also the corresponding mapping must be well-defined. Additionally and more importantly, two stringent constraints, namely the sibling and the parent relationship must also be satisfied. Thus, the constrained tree editing problem can be considered as a "conditionally optimalized" constrained string editing problem in which the postorder string of T_1 is edited to the postorder string of T_2 subject to H_s and further subject to the sibling/parent conditions. This is the reason why the problem currently studied is far more complex.

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 $\tau = \phi$, the null set. This is merely a way of expressing that it is impossible to edit T_1 to T_2 subject to the constraint τ. We now consider the computation of $D_\tau(T_1,T_2)$.

IV. CONSTRAINED TREE EDITING ALGORITHM

Since edit constraints can be written as unique subsets of H_s, we will denote the distance between forest $T_1[i'..i]$ and forest $T_2[j'..j]$ subject to the constraint that exactly s substitutions are performed by Const_F_Wt($T_1[i'..i],T_2[j'..j]$,s) or more precisely by Const_F_Wt([i'..i],[j'..j],s) if the context is clear. The distance between $T_1[1..i]$ and $T_2[1..j]$ subject to the constraint that exactly s substitutions are used is sometimes denoted 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 constraint that exactly s substitutions are used is denoted by Const_T_Wt(i,j,s). The difference between Const_F_Wt and Const_T_Wt is subtle. Indeed,

$$\text{Const\_T\_Wt}(i,j,s) = \text{Const\_F\_Wt}(T_1[\delta(i)..i],T_2[\delta(j)..j],s),$$

and compares segments of T_1 and T_2 which are exactly trees.

4.2 The Constrained Editing Algorithm

To develop the Constrained Tree Editing Algorithm, we now present the following results :

LEMMA II (a)

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j), then

(i) $\text{Const\_F\_Wt}(\mu,\mu,0) = 0.$

(ii) $\text{Const\_F\_Wt}(T_1[\delta(i_1)..i],\mu,0) = \text{Const\_F\_Wt}(T_1[\delta(i_1)..i-1],\mu,0) + d(T_1[i],\lambda).$

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

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

PROOF :

Case (i) requires no edit operations. In cases (ii) and (iii), the distance corresponds to the cost of deleting and inserting nodes in $T_1[\delta(i_1)..i]$ and $T_2[\delta(j_1)..j]$, 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, and the minimum of these two costs will be the desired distance. ****

LEMMA II (b).

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j), then

(i) Const_F_Wt $(T_1[\delta(i_1)..i],\mu,s) = \infty$ if $s > 0$.

(ii) Const_F_Wt $(\mu,T_2[\delta(j_1)..j],s) = \infty$ if $s > 0$.

(iii) Const_F_Wt $(\mu,\mu,s) = \infty$ if $s > 0$.

PROOF :

If $s > 0$, there must be exactly s nodes in both trees which are involved in the substitution. Since, none of the above three cases satisfies this, all of the distances equal ∞, implying that it is impossible to edit the trees subject to this constraint. ****

THEOREM III.

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j), then

$$\text{Const\_F\_Wt}\,(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s)$$

$$= \text{Min}\begin{cases} \text{Const\_F\_Wt}([\delta(i_1)..i\text{-}1],[\delta(j_1)..j],s) + d(T_1[i],\lambda) \\ \text{Const\_F\_Wt}([\delta(i_1)..i],[\delta(j_1)..j\text{-}1],s) + d(\lambda,T_2[j]) \\ \underset{1\le s_2\le \text{Min}\{\text{Size(i)};\text{Size(j)}\}}{\text{Min}} \begin{cases} \text{Const\_F\_Wt}([\delta(i_1)..\delta(i)\text{-}1],[\delta(j_1)..\delta(j)\text{-}1],s\text{-}s_2) \\ + \text{Const\_F\_Wt}([\delta(i)..i\text{-}1],[\delta(j)..j\text{-}1],s_2\text{-}1) \\ + d(T_1[i],T_2[j]) \end{cases} \end{cases}$$

PROOF :

We are trying to find a minimum cost mapping M between $T_1[\delta(i_1)..i]$ and $T_2[\delta(j_1)..j]$ such that exactly s substitutions are performed. 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 Const_F_Wt(.,.,.) remains unchanged, we have :
Const_F_Wt$(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s)$ = Const_F_Wt$(T_1[\delta(i_1)..i\text{-}1],T_2[\delta(j_1)..j],s) + d(T_1[i],\lambda)$.

(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(.,.,.) remains unchanged, we have :

$\text{Const\_F\_Wt}(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s) = \text{Const\_F\_Wt}\ (T_1[\delta(i_1)..i],T_2[\delta(j_1)..j-1],s) + d(\lambda,T_2[j])$.

Observe that in both the above cases the numbers of substitutions done remains unchanged.

(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 $\delta(i_1) \le h \le \delta(i)-1$, 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 $T_2[\delta(j_1)..j]$ since j is the rightmost sibling in $T_2[\delta(j_1)..j]$. 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 $k \le j$. So h has to equal to 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 the following figure:

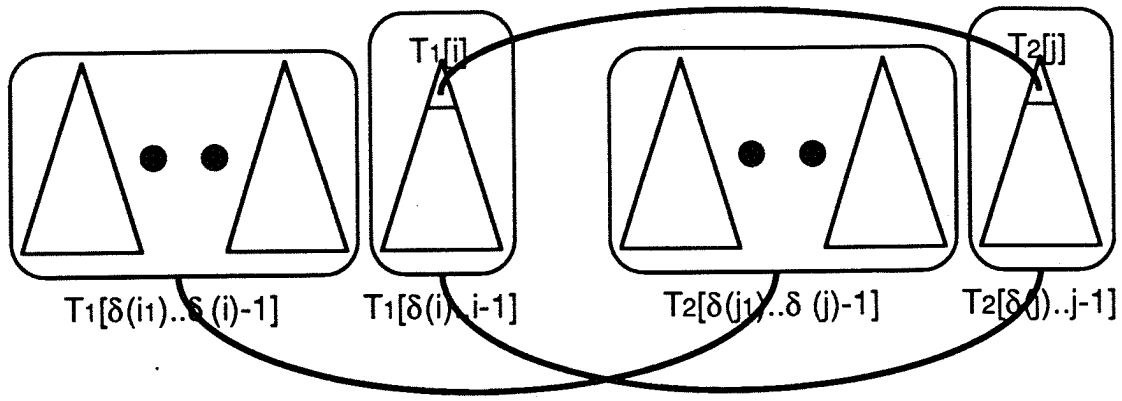

Figure VI : Case 3 of Theorem III.

Since exactly s substitutions must be performed in this transformation, the total number of substitutions used in the sub-transformation from $T_1[\delta(i_1)..\delta(i)-1]$ to $T_2[\delta(j_1)..\delta(j)-1]$ and the sub-transformation from $T_1[\delta(i)..i-1]$ to $T_2[\delta(j)..j-1]$ 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 sub-transformation from $T_1[\delta(i)..i-1]$ to $T_2[\delta(j)..j-1]$, then s_2 can take any value between 1 to $\text{Min}\{\text{Size}(i),\text{Size}(j),s\}$. Hence, we have :

$$\text{Const\_F\_Wt}\ (T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s)$$

$$= \min_{1 \le s_2 \le \text{Min}\{\text{Size}(i);\text{Size}(j);s\}} \begin{cases} \text{Const\_F\_Wt}\ (T_1[\delta(i_1)..\delta(i)-1],T_2[\delta(j_1)..\delta(j)-1],s-s_2) \\ + \text{Const\_F\_Wt}\ (T_1[\delta(i)..i-1],T_2[\delta(j)..j-1],s_2-1) \\ + d(T_1[i],T_2[j]) \end{cases}$$

Since these three cases exhaust the possible ways for yielding $\text{Const\_F\_Wt}\ (\delta(i_1)..i,\delta(j_1)..j,s)$, 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 - probably of an octic time complexity. Improvements in both the time and space complexities can be obtained by utilizing the following result involving the quantity Const_T_Wt(.,.,.) described earlier.

The main handicap with Theorem III is that when substitutions are involved, the quantity Const_F_Wt ($T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s$) evaluated between the forests $T_1[\delta(i_1)..i]$ and $T_2[\delta(j_1)..j]$ is computed using the Const_F_Wts of the forests $T_1[\delta(i_1)..\delta(i)-1]$ and $T_2[\delta(j_1)..\delta(j)-1]$ and incorporating them with the Const_F_Wts of the remaining forests $T_1[\delta(i)..i-1]$ and $T_2[\delta(j)..j-1]$. But if we observe that under certain conditions the removal of a sub-forest leaves us with an ***entire tree***, the computation need not be so extensive. Thus, if $\delta(i) = \delta(i_1)$ and $\delta(j) = \delta(j_1)$ (which means that both i and i_1, and j and j_1 span the same subtree), clearly the subforests from $T_1[\delta(i_1)..\delta(i)-1]$ and $T_2[\delta(j_1)..\delta(j)-1]$ do not get included in the computation. On the other hand if this is not the case, the Const_F_Wt ($T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s$) can be considered as a combination of the Const_F_Wt($T_1[\delta(i_1)..\delta(i)-1],T_2[\delta(j_1)..\delta(j)-1],s-s_2$)) and the tree weight between the trees rooted at i and j respectively, which is Const_T_Wt (i,j,s_2). Informally speaking, this is exactly what the following theorem claims.

THEOREM IV.

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j), then the following is true :

If $\delta(i) = \delta(i_1)$ and $\delta(j) = \delta(j_1)$ then

$$\text{Const\_F\_Wt}(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s) = \text{Min}\begin{cases} \text{Const\_F\_Wt}(T_1[\delta(i_1)..i-1],T_2[\delta(j_1)..j],s) + d(T_1[i],\lambda) \\ \text{Const\_F\_Wt}(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j-1],s) + d(\lambda,T_2[j]) \\ \text{Const\_F\_Wt}(T_1[\delta(i_1)..\delta(i)-1],T_2[\delta(j_1)..\delta(j)-1],s-1) + d(T_1[i],T_2[j]) \end{cases}$$

otherwise,

$$\text{Const\_F\_Wt}(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j],s) = \text{Min}\begin{cases} \text{Const\_F\_Wt}(T_1[\delta(i_1)..i-1],T_2[\delta(j_1)..j],s) + d(T_1[i],\lambda) \\ \text{Const\_F\_Wt}(T_1[\delta(i_1)..i],T_2[\delta(j_1)..j-1],s) + d(\lambda,T_2[j]) \\ \underset{1\le s_2\le \text{Min}\{\text{Size(i)};\text{Size(j)};s\}}{\text{Min}} \begin{cases} \text{Const\_F\_Wt}(T_1[\delta(i_1)..\delta(i)-1],T_2[\delta(j_1)..\delta(j)-1],s-s_2) \\ + \text{Const\_T\_Wt}(i,j,s_2) \end{cases} \end{cases}$$

PROOF :

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

Const_F_Wt ($T_1[\delta(i_1)..\delta(i)-1]$, $T_2[\delta(j_1)..\delta(j)-1],s-s_2$) = Const_F_Wt ($\phi,\phi,s-s_2$)

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:

$$\text{Const\_F\_Wt}\ (T_1[\delta(i_1)..i], T_2[\delta(j_1)..j], s) \leq \text{Const\_F\_Wt}\ (T_1[\delta(i_1)..\delta(i)-1], T_2[\delta(j_1)..\delta(j)-1], s-s_2) + \text{Const\_T\_Wt}\ (i,j,s_2).$$

This is because the latter formula represents a particular mapping of $T_1[\delta(i)..i]$ to $T_2[\delta(j)..j]$ in which the forest $T_1[\delta(i_1)..\delta(i)-1]$ is transformed into the forest $T_2[\delta(j_1)..\delta(j)-1]$ 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 an Const_F_Wt. For the same reason, we have :

$$\text{Const\_T\_Wt}\ (i,j,s_2) \leq \text{Const\_F\_Wt}\ (T_1[\delta(i)..i-1], T_2[\delta(j)..j-1], s_2-1) + d(T_1[i], T_2[j]).$$

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 $0 \leq s_2 \leq s$, 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 by-product[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 the first. Thus, the Const_T_Wt can be computed for the entire tree if Const_T_Wt of the Essential_Nodes are computed, and using this stored values, the rest of the Const_T_Wts can be computed. This suggest a bottom-up approach for computing the Const_T_Wt between all pairs of subtrees. To formally present the algorithm, we define the set of Essential_Nodes described above.

Definition : We define the set Essential_Nodes[2] of tree T as follows :

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

---

[1]The reason why this is obtained as a by-product will be clear when the algorithm is formally presented. Indeed, whenever an Const_F_Wt is computed, if the forests are trees, it is retained as a Const_T_Wt.

[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.

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. As an example, consider the trees as shown in Figure 5, the Essential_Nodes(T_1) = {2,6,7,8} and the Essential_Nodes(T_2) = {4,5,6}.

It is easy to see that the function $\delta()$ and the set Essential_Nodes() can be computed in linear time. We assume that the results are stored in arrays δ [] and Essential_Nodes [] respectively. Furthermore, we assume that the elements in array Essential_Nodes [] are sorted and stored in an increasing order 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), $1 \le i \le |T_1|$, $1 \le j \le |T_2|$, and $1 \le s \le \text{Min}\{|T_1|, |T_2|\}$.
ASSUMPTION : The algorithm assume a procedure **Preprocess (T_1,T_2)** which preprocesses the trees and yields the δ [] and Essential_Nodes [] arrays for both trees. These quantities are assumed to be global. It also invokes the procedure **Compute_Const_T_Wt** given subsequently.

```
BEGIN
        Preprocess (T1,T2) ;
        FOR i' = 1 to | Essential_Nodes1[] | DO
            FOR j' = 1 to | Essential_Nodes2[] | DO
                i = Essential_Nodes1 [i'];
                j = Essential_Nodes2 [j'];
                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 ([δ(i)..h], [δ(j)..k], s') defined for a particular input pair (i, j), where h and k are the internal nodes of Tree_1(i) and Tree_2(j) satisfying, $\delta(i) \le h \le i$, $\delta(j) \le k \le j$, and where s' is in the set of feasible values and satisfies $0 \le s' \le s = \text{Min}\{|\text{Tree}_1(i)|, |\text{Tree}_2(j)|\}$. Our intention is store ***these*** values using a single ***temporary*** three-dimensional array Const_F_Wt [.,.,.]. But in order to achieve this, it is clear that the base indices of the temporary three-dimensional array Const_F_Wt [.,.,.] 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 follows :

$$b_1 = \delta_1(i) - 1, \quad \text{and} \quad b_2 = \delta_2(j) - 1.$$

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

$$\text{Const\_F\_Wt } [x_1,y_1,s'] = \text{Const\_F\_Wt } ([\delta(i)..\delta(i)+x_1-1], [\delta(j)..\delta(j)+y_1-1], s')$$

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

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 [.,.,.] 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 $[i_1, j_1, s]$, $\delta_1(i) \le i_1 \le i$, $\delta_2(j) \le j_1 \le j$, $0 \le s \le \text{Min}\{\text{Size}(i),\text{Size}(j)\}$.
BEGIN

```
N = i - δ1(i) + 1;                  /* size of subtree rooted at T1[i] */
M = j - δ2(j) + 1;                  /* size of subtree rooted at T2[j] */
R = Min{M, N};
b1 = δ1(i) - 1;                     /* adjustment for nodes in subtree rooted at T1[i] */
b2 = δ2(j) - 1;                     /* adjustment for nodes in subtree rooted at T2[j] */

Const_F_Wt [0][0][0] = 0;

FOR x1 = 1 to N DO
    Const_F_Wt [x1][0][0] = Const_F_Wt [x1-1][0][0] + d(T1[x1+b1]→λ);
    Const_T_Wt [x1+b1][0][0] = Const_F_Wt [x1][0][0];
ENDFOR

FOR y1 = 1 to M DO
    Const_F_Wt [0][y1][0] = Const_F_Wt [0][y1-1][0] + d(λ→T2[y1+b2]);
    Const_T_Wt [0][y1+b2][0] = Const_F_Wt [0][y1][0];
ENDFOR

FOR s = 1 to R DO
    Const_F_Wt [0][0][s] = ∞;
    Const_T_Wt [0][0][s] = Const_F_Wt [0][0][s];
ENDFOR

FOR x1 = 1 to N DO
    FOR y1 = 1 to M DO
        Const_F_Wt [x1][y1][0]
            = Min { Const_F_Wt [x1][y1-1][0] + d(λ→T2[y1+b2])
                  { Const_F_Wt [x1-1][y1][0] + d(T1[x1+b1]→λ)
        Const_T_Wt [x1+b1][y1+b2][0] = Const_F_Wt [x1][y1][0];
    ENDFOR
ENDFOR
```

```
FOR x1 = 1 to N DO
    FOR s = 1 to R DO
        Const_F_Wt [x1][0][s] = ∞;
        Const_T_Wt [x1+b1][0][s] = Const_F_Wt [x1][0][s];
    ENDFOR
ENDFOR

FOR y1 = 1 to M DO
    FOR s = 1 to R DO
        Const_F_Wt [0][y1][s] = ∞;
        Const_T_Wt [0][y1+b2][s] = Const_F_Wt [0][y1][s];
    ENDFOR
ENDFOR

FOR x1 = 1 to N DO
    FOR y1 = 1 to M DO
        FOR s = 1 to R DO
            IF δ1(x1+b1) = δ1(x) and δ2(y1+b2) =δ2(y) THEN
                Const_F_Wt [x1][y1][s]
                    = Min { Const_F_Wt [x1-1][y1][s] + d(T1[x1+b1]→λ)
                          { Const_F_Wt [x1][y1-1][s] + d(λ→T2[y1+b2])
                          { Const_F_Wt [x1-1][y1-1][s-1]
                          {   + d(T1[x1+b1]→T2[y1+b2])
                Const_T_Wt [x1+b1][y1+b2][s] = Const_F_Wt [x1][y1][s];
            ELSE
                Const_F_Wt [x1][y1][s]
                    = Min { Const_F_Wt [x1-1][y1][s] + d(T1[x1+b1]→λ)
                          { Const_F_Wt [x1][y1-1][s] + d(λ→T2[y1+b2])
                          {        Min        { Const_F_Wt [a][[b][s-s2]
                          { 1≤s2≤Min{c; d; s} {   + Const_T_Wt [x1+b1][y1+b2][s2]

                where  a = δ1(x1+b1) -1-b1, b = δ2(y1+b2)-1-b2,
                       c = Size(x1+b1) and d = Size(y1+b2).
            ENDIF
        ENDFOR
    ENDFOR
ENDFOR
END.
```

ALGORITHM Constrained_Tree_Distance
INPUT : The array Const_T_Wt [.,.,.] computed using Algorithm **T_Weights**, and the constraint set τ.
OUTPUT : The constrained distance $D_\tau(T_1,T_2)$.
BEGIN

```
    Dτ(T1,T2) = ∞;
    FOR all s ∈ τ DO
        Dτ(T1,T2) = Min {Dτ(T1,T2), Const_T_Wt [|T1|][|T2|][s]}
    ENDFOR
```

END.

THEOREM V.

The basic algorithm is correct.

PROOF :

We will prove that the following two invariants hold for all pairs (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 $\delta(i) \leq h \leq i$, $\delta(j) \leq k \leq j$, $0 \leq s' \leq$ Min{Size(h),Size(k)}, and either $\delta(i) \neq \delta(h)$ or $\delta(j) \neq \delta(k)$. This, of course, 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. In other words, 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) Immediately after the computation of Const_T_Wt (i,j,s), all distances Const_T_Wt (h,k,s') are available, where $\delta(i) \leq h \leq i$, $\delta(j) \leq k \leq j$, and $0 \leq s' \leq$ Min{Size(h),Size(k)}. In other words, all Const_T_Wt (h,k,s') are 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 $0 \leq s' \leq$ Min{Size(h),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 $h' \leq i$. Combing 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. ****

V. TIME AND SPACE COMPLEXITIES

We shall now analyze the time and space complexities of the algorithm. From [11], we have the following result :

LEMMA III.

For every tree T,

Cardinality (Essential_Nodes (T)) ≤ |Leaves(T)|

PROOF :

Due to the equivalence of the sets Essential_Nodes(T) and LR_keyroots(T) of [11], the proof is identical to the proof of Lemma 6 of [11]. ****

This leads us to our primary complexity result.

THEOREM VI.

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 :

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

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

PROOF :

For space complexity, we use a permanent array for Const_T_Wt and a temporary array for Const_F_Wt. Each of these two arrays requires space $O(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\})$ since the first component of these arrays is $O(|T_1|)$, the second component of these arrays is $O(|T_2|)$, and the last component of these arrays is the maximum number of substitution that are feasible, which is $O(\text{Min}\{|T_1|, |T_2|\})$.

With regard to the time complexity, we first observe that the preprocessing takes linear time to precompute the arrays δ[] and Essential_Nodes[] for both trees. 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 constrained distance dynamic programming algorithm involving the subtrees rooted at i and j, respectively, involves : $O(\text{Size(i)} * \text{Size(j)} * \text{Min}\{\text{Size(i)}, \text{Size(j)}\}^2)$ computation. Therefore, total time required is :

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

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

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

Since the sets Essential_Nodes is identical to the set defined in [11] as the LR_keyroots, we see from Theorem 2 of [11] that :

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

Since Span(T) is equal to Min{Depth(T), Leaves(T)}, the time complexity of the algorithm is $O(|T_1| * |T_2| * \text{Min}\{|T_1|,|T_2|\}^2 * \text{Span}(T_1) * \text{Span}(T_2))$, and the theorem is proved ****

VI. 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. The way by which constraint τ can be specified has been proposed. Also, the technique to compute $D_\tau(T_1,T_2)$, the edit distance subject to constraint τ, has been presented. This set, τ, is typically a subset of the number of possible substitutions.

Given the trees T_1 and T_2, $D_\tau(T_1,T_2)$ the array of constrained edit distances Const_T_Wt (i,j,s) can be computed using dynamic programming, where Const_T_Wt (i,j,s) is the constrained distance between the subtrees rooted at i and j subject to the constraint that exactly s substitutions are performed. 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 $O(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\})$. From the array Const_T_Wt(i,j,s) the constrained edit distance $D_\tau(T_1,T_2)$ can be obtained in linear time.

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

REFERENCES

1. R. L. Kashyap and B. J. Oommen, "A common basic for similarity measures involving two strings", Intern. J. Computer Math. vol 13 : pp17-40 (1983).
2. R. L. Kashyap and B. J. Oommen, "Spelling correction using probabilistic methods", Pattern Recognition Letter, vol 2 : pp147-154 (1984).
3. S. Y. Lu, "A tree-to-tree distance and its application to cluster analysis", I.E.E.E. Trans. Pattern Anal. and Mach. Intell., vol : PAMI 1, No. 2, pp219-224 (1979).
4. S. Y. Lu, "A tree-matching algorithm based on node splitting and merging", I.E.E.E. Trans. Pattern Anal. and Mach. Intell., vol : PAMI 6, No 2, pp249-256 (1984).
5. B. J. Oommen, "Constrained string editing", Inform. Sci. vol 40 : pp267-284 (1986).
6. B. J. Oommen, "Recognition of noisy subsequences using constrained edit distance", I.E.E.E. Trans. on Pattern Anal. and Mach. Intell., vol PAMI 9, No. 5 : pp676-685 (1987).
7. D. Sankoff and J. B. Kruskal, "Time wraps, string edits, and macromolecules : Theory and practice of sequence comparsion", Addison-Wesley, 1983.
8. S. M. Selkow, "The tree-to-tree editing problem", Inform. Process. Letters, vol 6, No. 6, pp184-186 (1977).
9. K. C. Tai, "The tree-to-tree correction problem", J. Assoc. Comput. Mach., vol 26, pp422-433 (1979).
10. R. A. Wagner and M. J. Fischer, "The string-to-string correction problem", J. ACM, vol 21 : pp168-173 (1974).
11. K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems", SIAM J. Comput, vol 18, No. 6, pp1245-1262 (1989).

School of Computer Science, Carleton University
Recent Technical Reports

SCS-TR-159 **Epsilon-Optimal Stubborn Learning Mechanisms**
J.P.R. Christensen and B.J. Oommen, June 1989

SCS-TR-160 **Disassembling Two-Dimensional Composite Parts Via Translations**
Doron Nussbaum and Jörg-R. Sack, June 1989

SCS-TR-161 (revised) **Recognizing Sources of Random Strings**
R.S. Valiveti and B.J. Oommen, January 1990
Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989

SCS-TR-162 **An Adaptive Learning Solution to the Keyboard Optimization Problem**
B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989

SCS-TR-163 **Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time**
L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989

SCS-TR-164 **A Survey of Algorithms for Handling Permutation Groups**
M.D. Atkinson, January 1990

SCS-TR-165 **Key Exchange Using Chebychev Polynomials**
M.D. Atkinson and Vincenzo Acciaro, January 1990

SCS-TR-166 **Efficient Concurrency Control Protocols for B-tree Indexes**
Ekow J. Otoo, January 1990

SCS-TR-167 **A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem**
B.J. Oommen, January 1990

SCS-TR-168 **Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule**
R.S. Valiveti and B.J. Oommen, January 1990

SCS-TR-169 **Trade-Offs in Non-Reversing Diameter**
Hans L. Bodlaender, Gerard Tel and Nicola Santoro, February 1990

SCS-TR-170 **A Massively Parallel Knowledge-Base Server using a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-171 **Parallel Processing of Quad Trees on the Hypercube (and PRAM)**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-172 **A Note on the Load Balancing Problem for Coarse Grained Hypercube Dictionary Machines**
Frank Dehne and Michel Gastaldo, May 1990

SCS-TR-173 **Self-Organizing Doubly-Linked Lists**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-174 **A Presortedness Metric for Ensembles of Data Sequences**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-175 **Separation of Graphs of Bounded Genus**
Ljudmil G. Aleksandrov and Hristo N. Djidjev, May 1990

SCS-TR-176 **Edge Separators of Planar and Outerplanar Graphs with Applications**
Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora and Imrich Vrto, May 1990

SCS-TR-177 **Representing Partial Orders by Polygons and Circles in the Plane**
Jeffrey B. Sidney and Stuart J. Sidney, July 1990

SCS-TR-178 **Determining Stochastic Dependence for Normally Distributed Vectors Using the Chi-squared Metric**
R.S. Valiveti and B.J. Oommen, July 1990

SCS-TR-179 **Parallel Algorithms for Determining K-width-Connectivity in Binary Images**
Frank Dehne and Susanne E. Hambrusch, September 1990

SCS-TR-180 **A Workbench for Computational Geometry (WOCG)**
P. Epstein, A. Knight, J. May, T. Nguyen, and J.-R. Sack, September 1990

SCS-TR-181 **Adaptive Linear List Reorganization under a Generalized Query System**
R.S. Valiveti, B.J. Oommen and J.R. Zgierski, October 1990

SCS-TR-182 **Breaking Substitution Cyphers using Stochastic Automata**
B.J. Oommen and J.R. Zgierski, October 1990

SCS-TR-183 **A New Algorithm for Testing the Regularity of a Permutation Group**
V. Acciaro and M.D. Atkinson, November 1990

SCS-TR-184 **Generating Binary Trees at Random**
M.D. Atkinson and J.-R. Sack, December 1990

SCS-TR-185 **Uniform Generation of Combinatorial Objects in Parallel**
M.D. Atkinson and J.-R. Sack, January 1991

SCS-TR-186 **Reduced Constants for Simple Cycle Graph Separation**
Hristo N. Djidjev and Shankar M. Venkatesan, February 1991

SCS-TR-187 **Multisearch Techniques for Implementing Data Structures on a Mesh-Connected Computer**
Mikhail J. Atallah, Frank Dehne, Russ Miller, Andrew Rau-Chaplin, and Jyh-Jong Tsay, February 1991

SCS-TR-188 **Generating and Sorting Jordan Sequences**
Out of print
Alan Knight and Jörg-Rüdiger Sack, March 1991

SCS-TR-189 **Probabilistic Estimation of Damage from Fire Spread**
Charles C. Colbourn, Louis D. Nel, T.B. Boffey and D.F. Yates, April 1991

SCS-TR-190 **Coordinators: A Mechanism for Monitoring and Controlling Interactions Between Groups of Objects**
Wilf R. LaLonde, Paul White, and Kevin McGuire, April 1991

SCS-TR-191 **Towards Decomposable, Reusable Smalltalk Windows**
Kevin McGuire, Paul White, and Wilf R. LaLonde, April 1991

SCS-TR-192 **PARASOL: A Simulator for Distributed and/or Parallel Systems**
John E. Neilson, May 1991

SCS-TR-193 **Realizing a Spatial Topological Data Model in a Relational Database Management System**
Ekow J. Otoo and M.M. Allam, August 1991

SCS-TR-194 **String Editing with Substitution, Insertion, Deletion, Squashing and Expansion Operations**
B John Oommen, September 1991

SCS-TR-195 **The Expressiveness of Silence: Optimal Algorithms for Synchronous Communication of Information**
Una-May O'Reilly and Nicola Santoro, October 1991

SCS-TR-196 **Lights, Walls and Bricks**
J. Czyzowicz, E. Rivera-Campo, N. Santoro, J. Urrutia and J. Zaks, October 1991

SCS-TR-197 **A Brief Survey of Art Gallery Problems in Integer Lattice Systems**
Evangelos Kranakis and Michel Pocchiola, November 1991

SCS-TR-198 **On Reconfigurability of Systolic Arrays**
Amiya Nayak, Nicola Santoro, and Richard Tan, November 1991

SCS-TR-199 **Constrained Tree Editing**
B. John Oommen and William Lee, December 1991