On the Editing Distance Between Trees and Related Problems

by

Kaizhong Zhang

Dennis Shasha

Ultracomputer Note 122

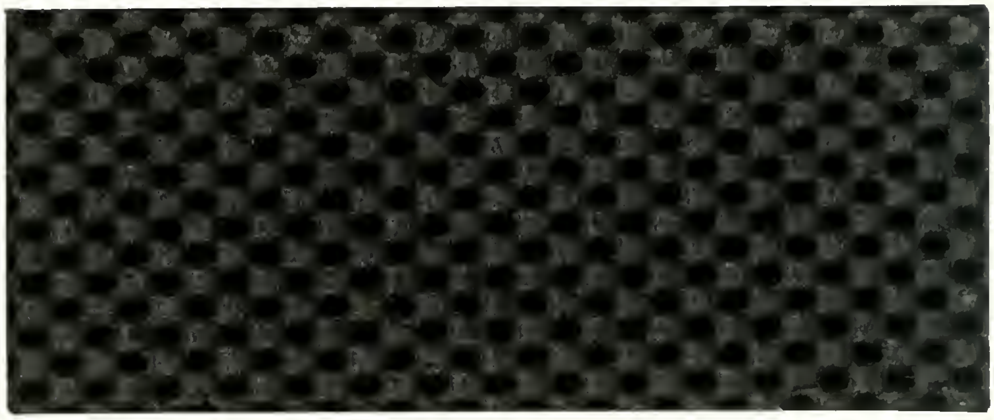
NYU Computer Science Technical Report 310

August, 1987

Ultracomputer Research Laboratory

NYU COMPSCI TR-310 c.1
Zhang, Kaizhong
On the editing distance
between trees and related
problems

New York University
Courant Institute of Mathematical Sciences
Division of Computer Science
251 Mercer Street, New York, NY 10012



On the Editing Distance Between Trees and Related Problems

by

Kaizhong Zhang

Dennis Shasha

Ultracomputer Note 122

NYU Computer Science Technical Report 310

August, 1987

This work was supported in part by the National Science Foundation under grant number DCR8501611, by the Office of Naval Research under grant number N00014-85-K-0046, and by the Supercomputing Research Center.

ABSTRACT

Since a tree can represent a scene description, a grammar parse, a structural description, and many other phenomena, comparing trees is a way to compare scenes, parses and so on. We consider the distance between two trees to be the (weighted) number of edit operations (insert, delete, and modify) to transform one tree to another. Then, we consider the following kinds of questions:
1. What is the distance between two trees?
2. What is the minimum distance between T_1 and T_2 with an arbitrary subtree removed? What if zero or more subtrees can be removed from T_2? A specialization of these algorithms solves the problem of approximate tree matching.
3. Given k, are T_1 and T_2 within distance k of one another and if so what is their distance?

We present a postorder dynamic programming algorithm to solve question 1 in sequential time $O(|T_1| \times |T_2| \times depth(T_1) \times depth(T_2))$ compared with $O(|T_1| \times |T_2| \times (depth(T_1))^2 \times (depth(T_2))^2)$ for the best previous algorithm due to Tai. Further, our algorithm can be parallelized to give time $O(|T_1| + |T_2|)$. Our approach extends to answer question 2 giving an algorithm of the same complexity. For question 3, a variant of our distance algorithm yields an $O(k^2 \times \min(|T_1|, |T_2|) \times \min(depth(T_1), depth(T_2)))$ algorithm.

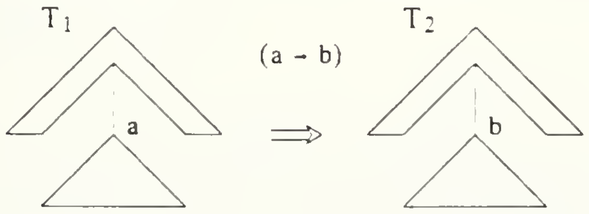
1. Editing distance between two trees

Ordered labeled trees are trees in which the left-to-right order among siblings is significant. The distance between such trees is a generalization of the distance between strings.

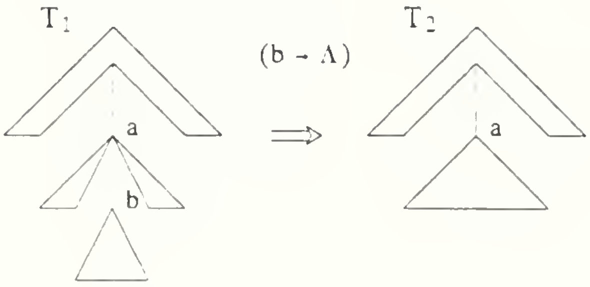
The following definitions are basically from [T-79, WF-74]. Let T_1 and T_2 be two trees with N_1 and N_2 nodes respectively. Suppose that we have an ordering for each tree and $T_1[i]$ means the ith node of tree T_1 in the given ordering. $T[i..j]$ is the subtree of T whose nodes are numbered i to j inclusive. If $i > j$, then $T[i..j] = \emptyset$. An edit operation is a pair $(a,b) \neq (\Lambda,\Lambda)$ of strings of length less than or equal to 1 and is usually written as $a \to b$. We call $a \to b$ a change operation if $a \neq \Lambda$ and $b \neq \Lambda$; a delete operation if $b = \Lambda$; and an insert operation if $a = \Lambda$.

Let us consider these three kinds of operations.

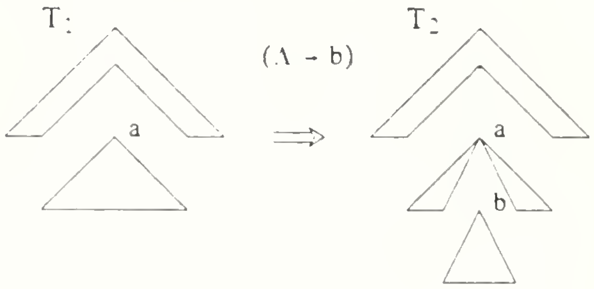
1. Change: To change one node label to another.



2. Delete: To delete a node.
(All children of the deleted node b become children of the parent a.)



3. Insert: To insert a node.
(A consecutive sequence of siblings among the children of a become the children of b.)



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 $1 \le i \le k$.

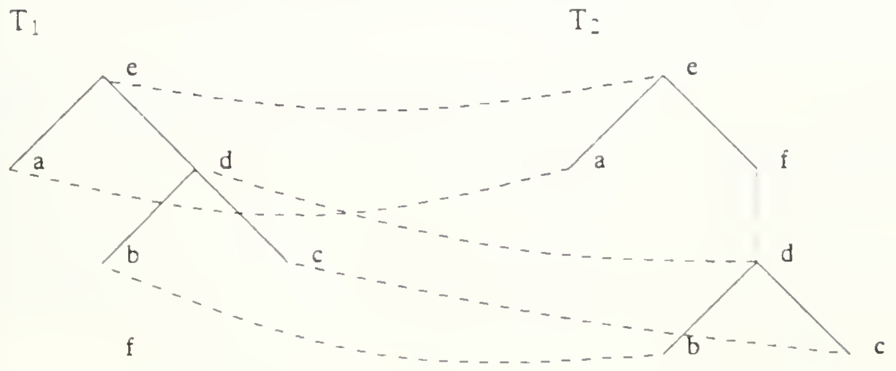
Let γ be an cost function which assigns to each edit operation $a \to b$ a nonnegative real number $\gamma(a \to b)$. This cost can be different for different nodes, so it can be used to give greater weights to, for example, the higher nodes in a tree than to lower nodes.

We constrain γ to be a distance metric. That is,
i) $\gamma(a \to a) = 0$;
ii) $\gamma(a \to b) = \gamma(b \to a)$; and
iii) $(a \to c) \le \gamma(a \to b) - \gamma(b \to c)$.

We extend γ to the sequence S by letting $\gamma(S) = \sum_{i=1}^{i=k} \gamma(s_i)$. Formally the distance between T_1 and T_2 is defined as:
$\delta(T_1,T_2) = \min\{\gamma(S) \mid S$ is an edit operation sequence taking T_1 to $T_2\}$. The definition of γ makes this a distance metric also.

The edit operations give rise to a mapping. Intuitively, a mapping is a description of how a sequence of edit operations transform T_1 into T_2, ignoring the order in which edit operations are applied.

Consider the following diagram of a mapping:



A dotted line from $T_1[i]$ to $T_2[j]$ indicates that $T_1[i]$ should be changed to $T_2[j]$ if $T_1[i] \neq T_2[j]$, or that $T_1[i]$ remains unchanged if $T_1[i]=T_2[j]$. The nodes of T_1 not touched by a dotted line are to be deleted and the nodes of T_2 not touched are to be inserted. The mapping shows a way to transform T_1 to T_2.

Formally we define a triple (M,T_1,T_2) to be a mapping from T_1 to T_2, where M is any set of pair of integers (i,j) satisfying:[1]

(1) $1 \le i \le N_1$, $1 \le j \le N_2$;
(2) For any pair of (i_1,j_1) and (i_2,j_2) in M,
  (a) $i_1=i_2$ iff $j_1=j_2$
  (b) $T_1[i_1]$ is to the left of $T_1[i_2]$ iff $T_2[j_1]$ is to the left of $T_2[j_2]$
  (c) $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$

We will use M instead of (M,T_1,T_2) if there is no confusion. Let M be a mapping from T_1 to T_2. Let I and J be the sets of nodes in T_1 and T_2, respectively, not touched by any line in M. Then we can define the cost of M:

$$\gamma(M)=\sum_{(i,j)\in M}\gamma(T1[i]\rightarrow T_2[j])+\sum_{i\in I}\gamma(T1[i]\rightarrow\Lambda)+\sum_{j\in J}\gamma(\Lambda\rightarrow T_1[j])$$

Mappings can be composed. Let M_1 be a mapping from T_1 to T_2 and let M_2 be a mapping from T_2 to T_3. Define
$M_1 \circ M_2 = \{(i,j) \mid \exists\ k \text{ s.t. } (i,k) \in M_1 \text{ and } (k,j) \in M_2\}$

Lemma 1:
(1) $M_1 \circ M_2$ is a mapping
(2) $\gamma(M_1 \circ M_2) \le \gamma(M_1) + \gamma(M_2)$

Proof:

(1). is clear from the definition of mapping.
(2). Let M_1 be the mapping from T_1 to T_2 and I_1 and J_1 be the corresponding deletion and insertion sets. Let M_2 be the mapping from T_2 to T_3 and I_2 and J_2 be the corresponding deletion and insertion sets. Let $M_1 \circ M_2$ be the composed mapping from T_1 to T_3 and let I and J be the corresponding deletion and insertion sets.

Three general situations occur. $(i,j) \in M_1 \circ M_2$, $i \in I$, or $j \in J$. In each case this corresponds to an editing operation $\gamma(x \rightarrow y)$ where x and y may be nodes or may be Λ. In all such cases, the

[1] - Note that our definition of mapping is different from the definition in [T-79]. We believe that our definition is more natural because it does not depend on any traversal ordering of the tree.

triangle inequality on the distance metric γ ensures that $\gamma(x \to y) \leq \gamma(x \to z) + \gamma(z \to y)$. □

The relation between a mapping and a sequence of edit operation is as follows:

Lemma 2:

Given S, a sequence $s_1, \ldots, s_k$ of edit operations from T_1 to T_2, there exists a mapping M from T_1 to T_2 such that $\gamma(M) \leq \gamma(S)$.

Proof:

This can be proved by induction on k. The base case is k=1. This case is correct, because any editing operation preserves the ancestor and sibling relationships in the mapping. In general case, let S_1 be the sequence $s_1, \ldots, s_{k-1}$ of edit operation. There exist a mapping M_1 such that $\gamma(M_1) \leq \gamma(S_1)$. Let M_2 be the mapping for s_k. Now from lemma 1, we have following.
$\gamma(M_1 \circ M_2) \leq \gamma(M_1) + \gamma(M_2) \leq \gamma(S)$. □

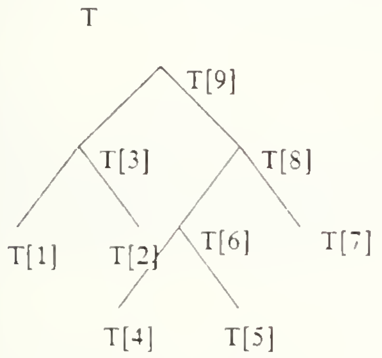
Hence, $\delta(T_1, T_2) = \min\{\gamma(M) | M$ is a mapping from T_1 and $T_2\}$

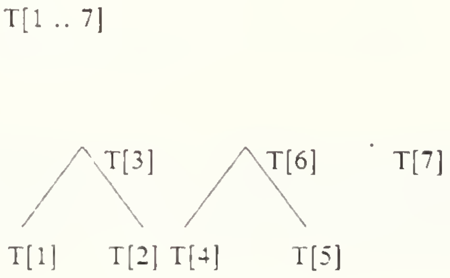
There has been previous work on this problem. Tai [T-79] gave the previous best algorithm for the problem. [Z-83] is an improvement of [T-79], giving the same sequential time as our algorithm. Our new algorithm is simpler than [Z-83], gives a better parallel time, and extends to related problems. The algorithm of Lu [L-79] does not extend to trees of more than a two levels.

2. A simple new algorithm

This algorithm, unlike [T-79], [L-79], and [Z-83], will, in its intermediate steps, consider the distance between two ordered forests. At first sight one may think that this will complicate the work for us, but it will in fact make matters easier.

We use a postorder numbering of the nodes in the trees. In the postordering, $T_1[1 .. i]$ and $T_2[1 .. j]$ will generally be forests as in the following figure. (The edges are those in the subgraph of the tree induced by the vertices.) Fortunately, the definition of mapping for these induced ordered forests is the same as for trees.





Notation. Assume that tree T is numbered by post order. $l(i)$ is the number of the leftmost leaf descendant of the subtree rooted at $T[i]$. When $T[i]$ is a leaf, $l(i)=i$. $p(i)$ is the number of the parent of node $T[i]$. We define $p^0(i)=i$, $p^1(i)=p(i)$, $p^2(i)=p(p^1(i))$ and so on. Let L_i be the depth of node i. (We take the depth of the root to be 1.) Let $anc(i)=\{p^k(i) | 0 \le k \le L_i\}$.

2.1. A first attempt

We first attempt to solve the problem as it is solved for strings. We try to compute $D(T_1[1 .. i],T_2[1 .. j])$, where $1 \le i \le N_1$ and $1 \le j \le N_2$. There are three cases:

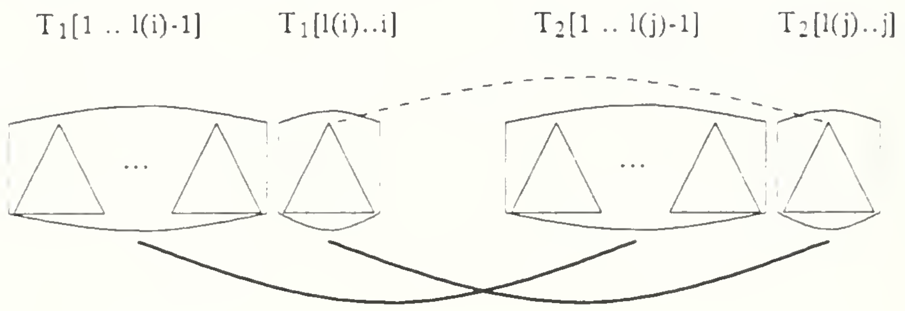
Case 1. $T_1[i]$ is not touched by a line in M.
In this case $D(T_1[1 .. i],T_2[1 .. j]) = D(T_1[1 .. i-1],T_2[1 .. j]) + \gamma(T_1[i] \to \Lambda)$.

Case 2. $T_2[j]$ is not touched by a line in M.
In this case $D(T_1[1 .. i],T_2[1 .. j]) = D(T_1[1 .. i],T_2[1 .. j-1]) + \gamma(\Lambda \to T_2[j])$.

Case 3. $T_1[i]$ and $T_2[j]$ are touched by lines in M.
As we explain later, (i,j) must be in M and any node in the subtree rooted at $T_1[i]$ can only be touched by a node in the subtree rooted at $T_2[j]$.

Hence $D(T_1[1 .. i],T_2[1 .. j]) =$
$D(T_1[1 .. l(i)-1],T_2[1 .. l(j)-1]) + D(T_1[l(i) .. i-1],T_2[l(j) .. j-1]) + \gamma(T_1[i] \to T_2[j])$ Recall that if $i > j$, then $T[i..j] = \varnothing$.

The following figure shows the situation.
(The solid lines indicate the pairs of structures
whose distances must be calculated.)



Case 3 expresses the crucial difference between strings and trees. It says that in order to compute the distance between the forests up to $T_1[i]$ and $T_2[j]$, we need the distance between the subtrees rooted at those nodes. (Failing to recognize this caused Lu's algorithm to come to grief.) Calculating it requires knowing $D(T_1[l(i) .. i-1], T_2[l(j) .. j-1])$, which our first attempt did not compute.

In general, we compute $D(T_1[i_1 .. i], T_2[j_1 .. j])$, where
$i_1 \in \{l(p^0(i)), l(p^1(i)), l(p^2(i)), \ldots, 1\}$ and
$j_1 \in \{l(p^0(j)), l(p^1(j)), l(p^2(j)), \ldots, 1\}$.

Note here for tree T with N nodes, node 1 is $l(N)$ -- the leftmost descendant of the root and the first node visited in the postorder traversal -- which equals $l(p^{L_i-1}(i))$, where L_i is the depth of node $T[i]$.

2.2. New Algorithm

We first present three lemmas and then give our new algorithm.

Recall that $anc(i) = \{p^k(i) \mid 0 \le k \le L_i\}$

Lemma 3:
(1) If $i_1 \in anc(i)$, either $T_1[l(i_1) .. i-1]$ is empty or $i_1 \in anc(i-1)$.
(2) If $i_1 \in anc(i)$, either $T_1[l(i_1) .. l(i)-1]$ is empty or $i_1 \in anc(l(i)-1)$.
(3) If $i_1 \in anc(i)$, either $T_1[l(i) .. i-1]$ is empty or $i \in anc(i-1)$.

Proof:

(1). Suppose $i_1 \in anc(i)$. Because of the post-order numbering, $p(i-1) \in anc(i)$. If $i_1 \ge p(i-1)$, then either $i_1 \in anc(i-1)$ or i_1 is in a subtree to the right of $p(i-1)$. But the second is impossible since i_1 and $p(i-1)$ are both ancestors of i. If $i_1 < p(i-1)$, then $T_1[l(i_1) .. i-1]$ is empty.

(2). Suppose $i_1 \in anc(i)$. If $l(i_1) = l(i)$, then $T_1[l(i_1) .. l(i)-1]$ is empty, If $l(i_1) < l(i)$, then $i_1 \in anc(p(l(i)-1))$. So, $i_1 \in anc(l(i)-1)$. By the post-ordering and the ancestor assumption, $l(i_1) > l(i)$ is impossible.

(3). Suppose $i_1 \in anc(i)$. If $l(i) = i$, $T_1[l(i) .. i-1]$ is empty. If $l(i) < i$, then $p(i-1) = i$. So, $i \in anc(i-1)$. By the post-ordering, $l(i) > i$ is impossible. □

Lemma 4 deals with empty trees or forests.

Lemma 4:

(i) $D(\varnothing,\varnothing) = 0$

(ii) $D(T_1[l(i_1)\ ..\ i],\varnothing) = D(T_1[l(i_1)\ ..\ i-1],\varnothing) + \gamma(T_1[i] \to \Lambda)$

(iii) $D(\varnothing,T_2[l(j_1)\ ..\ j]) = D(\varnothing,T_2[l(j_1)\ ..\ j-1]) + \gamma(\Lambda \to T_2[j])$

where $i_1 \in anc(i)$ and $j_1 \in anc(j)$

Proof:

(i) requires no edit operation. In (ii) and (iii), the distances correspond to the cost of deleting or inserting the nodes in $T_1[l(i_1)\ ..\ i]$ and $T_2[l(j_1)\ ..\ j])$ respectively. □

Lemma 5 deal with the general situation.

Lemma 5:

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j]) = \min\{$

$D(T_1[l(i_1)\ ..\ i-1],T_2[l(j_1)\ ..\ j]) + \gamma(T_1[i] \to \Lambda)$,

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j-1]) + \gamma(\Lambda \to T_2[j])$,

$D(T_1[l(i_1)\ ..\ l(i)-1],T_2[l(j_1)\ ..\ l(j)-1]) + D(T_1[l(i)\ ..\ i-1],T_2[l(j)\ ..\ j-1]) + \gamma(T_1[i] \to T_2[j])\ \}$

where $i_1 \in anc(i)$ and $j_1 \in anc(j)$

Proof:

Consider a minimum cost mapping M such that $\gamma(M) = D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j])$. There are three cases.

Case 1:

$T_1[i]$ is not touched by a line in M. In this case

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j]) =$

$D(T_1[l(i_1)\ ..\ i-1],T_2[l(j_1)\ ..\ j]) + \gamma(T_1[i] \to \Lambda)$

Case 2:

$T_2[j]$ is not touched by a line in M. In this case

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j]) =$

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j-1]) + \gamma(\Lambda \to T_2[j])$

Case 3:

$T_1[i]$ and $T_2[j]$ are both touched by lines in M. Since a mapping must preserve ancestor and sibling relationships, they must touch each other, i.e. $(i,j) \in M$. For the same reason, any node in the subtree rooted at $T_1[i]$ can only be touched by a node in the subtree rooted at $T_2[j]$ and vice versa. Hence we have

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j]) =$

$D(T_1[l(i_1)\ ..\ l(i)-1],T_2[l(j_1)\ ..\ l(j)-1]) + D(T_1[l(i)\ ..\ i-1],T_2[l(j)\ ..\ j-1]) + \gamma(T_1[i] \to T_2[j])$

$D(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j])$ is just the minimum of the above three values. □

We are now ready to to give our new algorithm.

Algorithm Basic Distance

```
begin
 $D(\emptyset,\emptyset)=0$

  for $i:=1$ to $N_1$
    for $i_1 \in anc(i)$
      $D(T_1[l(i_1)..i],\emptyset)=D(T_1[l(i_1)..i-1],\emptyset)-\gamma(T_1[i]\to\Lambda)$

  for $j:=1$ to $N_2$
    for $j_1 \in anc(j)$
      $D(\emptyset,T_2[l(j_1)..j])=D(\emptyset,T_2[l(j_1)..j-1])-\gamma(\Lambda\to T_2[j])$

  for $i:=1$ to $N_1$
    for $j:=1$ to $N_2$
      for $i_1 \in anc(i)$
        for $j_1 \in anc(j)$ begin
          $D(T_1[l(i_1)..i],T_2[l(j_1)..j])=\min\{$
   $D(T_1[l(i_1)..i-1],T_2[l(j_1)..j])-\gamma(T_1[i]\to\Lambda),$
   $D(T_1[l(i_1)..i],T_2[l(j_1)..j-1])-\gamma(\Lambda\to T_2[j]),$
   $D(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1])-D(T_1[l(i)..i-1],T_2[l(j)..j-1])-\gamma(T_1[i]\to T_2[j])\}$
        end;
end
```

Theorem 1: Algorithm Basic Distance correctly computes tree distance.

Proof:

From lemma 3 we know that all the distance terms used in the right hand side of the equations have been computed previously. From lemma 4 and lemma 5, we know that the formulas used in above algorithm is correct. □

3. Some aspects of our algorithm

3.1. Complexity

Let us consider the time and space complexity of our algorithm.

By definition of L_i, $|anc(i)| = L_i$. Therefore the time and space complexity is at most $O(\sum_{i=1}^{i=N_1} L_i \cdot \sum_{j=1}^{j=N_2} L_j)$. If we substitute for $L_i \times L_j$ its maximum $L_1 \times L_2$, we obtain the following.

The time and space complexity is $O(N_1 \times N_2 \times L_1 \times L_2)$. This is an improvement over the $O(N_1 \times N_2 \times L_1^2 \times L_2^2)$ complexity of [T-79].

3.2. From trees to strings

The analysis in previous section is pessimistic. For special trees such as strings, the depth terms disappear altogether, as can be seen by observing the following. If we define $lanc(i) = \{l(i_1) | i_1 \in anc(i)\}$, then we can rewrite the algorithm as follows:

```
 for i:= 1 to $N_1$
   for j:= 1 to $N_2$
     for $i_1 \in lanc(i)$
          for $j_1 \in lanc(j)$ begin
$D(T_1[i_1 .. i], T_2[j_1 .. j]) = \min\{$
  $D(T_1[i_1 .. i-1], T_2[j_1 .. j]) + \gamma(T_1[i] \to \Lambda)$,
  $D(T_1[i_1 .. i], T_2[j_1 .. j-1]) + \gamma(\Lambda \to T_2[j])$,
  $D(T_1[i_1 .. l(i)-1], T_2[j_1 .. l(j)-1]) + D(T_1[l(i) .. i-1], T_2[l(j) .. j-1]) + \gamma(T_1[i] \to T_2[j])\}$
        end;
```

So, the complexity should be

$$O(\sum_{i=1}^{i=N_1} |lanc(i)| \times \sum_{j=1}^{j=N_2} |lanc(j)|).$$

So, if the parent of node i has only i as a child, then $lanc(i) = lanc(p(i))$. It is not the actual depth of i that matters, therefore, but its "collapsed depth", i.e. the number distinct leftmost descendants of nodes on the path from i to the root.

In the important extreme case of a string, every node has one child, so every node has collapsed depth of 1. Since $|lanc(i)| = 1$, time and space complexity of $O(N_1 \times N_2)$.

This is a nice property. This means that our algorithm is not only a generalization of the string algorithm to trees but also that when the input is really a string the complexity is the same as that of the best available algorithm for the general problem of string distance. The algorithms in [T-79] and [Z-83] do not have this property.

3.3. Parallel Implementation

A transformation of our algorithm to a parallel one has time complexity $O(N_1 + N_2)$ while [T-79] and [Z-83] have time complexity $O((N_1 + N_2) \times (L_1 + L_2))$. Our algorithm uses $O(\min(N_1, N_2) \times L_1 \times L_2)$ processors. Our strategy is to compute, in parallel, all distances $D(T_1[l(i_1)..i]$,

$T_2[l(j_1)..j])$ for which $i + j = k$.

We now present the parallel algorithm. (When the PARBEGIN - PAREND construct surrounds one or more for loops, it means that every setting of the iterators in the enclosed for loops can be executed in parallel. The semantics are those of the sequential program ignoring this construct.)

Algorithm Parallel Distance

```
begin
$D(\emptyset,\emptyset)=0$

for $i:=1$ to $N_1$
  PARBEGIN
    for $i_1 \in anc(i)$
      $D(T_1[l(i_1)..i],\emptyset)=D(T_1[l(i_1)..i-1],\emptyset)+\gamma(T_1[i]\to\Lambda)$
  PAREND

for $j:=1$ to $N_2$
  PARBEGIN
    for $j_1 \in anc(j)$
      $D(\emptyset,T_2[l(j_1)..j])=D(\emptyset,T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j])$
  PAREND

for $k:=2$ to $N_1+N_2$
  PARBEGIN
    for $i := \max(1,k-N_1)$ to $\min(k-1,N_2)$ do begin
        $j := k-i$,
     for $i_1 \in anc(i)$
      for $j_1 \in anc(j)$ do
      $D(T_1[l(i_1)..i],T_2[l(j_1)..j])=\min\{$
   $D(T_1[l(i_1)..i-1],T_2[l(j_1)..j])+\gamma(T_1[i]\to\Lambda)$,
   $D(T_1[l(i_1)..i],T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j])$,
   $D(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1])+D(T_1[l(i)..i-1],T_2[l(j)..j-1])+\gamma(T_1[i]\to T_2[j])\}$
   end
  PAREND
end
```

Theorem 2: Parallel Distance Algorithm is correct and has time complexity $O(N_1+N_2)$.

Proof:

The first two initializations are the same as in the Basic Distance algorithm. Let us consider the general case. For i and j within PARBEGIN and PAREND, $i+j=k$. We now show that all the terms used in the min expressions have been previously computed (so there are no interdependencies among the terms calculated for a given value of k). In the first term $i-1+j=k-1<k$. In the second term $i+j-1=k-1<k$. In the third term $l(i)-1+l(j)-1 \leq i-1+j-1=k-2<k$. In the fourth term $i-1+j-1=k-2<k$.

Since no sequential loop is executed more than N_1+N_2 times, the Parallel Distance algorithm has time complexity $O(N_1+N_2)$. □

4. k-distance problem

Here we show how to solve a specialization of tree distance in better time.

Given two trees T_1 and T_2 and a number k, we would like to know if the distance $\delta(T_1,T_2)$ is less than k or not and in case it is less than k to give the actual value of $\delta(T_1,T_2)$. For simplicity, we assume that the cost for insert and delete is 1. (An easy generalization to arbitrary costs for insertion and deletion is to take the minimum insert or delete cost c and then replace k by k/c everywhere, including in the complexity measures.)

Though we can use our general tree distance algorithm to solve the above problem, we have a more efficient method. Our algorithm has time complexity $O(k \times \min(N_1,N_2) \times L_1 \times L_2)$ or $O(k^2 \times \min(N_1,N_2) \times \min(L_1,L_2))$. Note that if k is small, this is a big improvement over the Basic Distance Algorithm.

As always, we take our inspiration from strings. The following algorithm is a simple $O(k \times \min(N_1,N_2))$ algorithm for the k-distance problem among strings.

```
for i:= 1 to $N_1$
  for j:= max(i − k,1) to min(i+k,$N_2$) begin
    if $T_1[i] = T_2[j]$
      then d:= 0
      else d:= 1;
    D(i,j)= min{ D(i,j − 1)+1,D(i − 1,j)+1,D(i − 1,j − 1)+d }
  end;
```

The idea is that we do not need to compute any D(i,j) such that $|i-j|>k$. The reason is that for those D(i,j), $D(i,j) \geq k$. Hence such terms will not be useful in any later computation.

The difficulty in the tree case is that even if $D(T_1[1..i],T_2[1..j])$ is greater than k, $D(T_1[l(i_1)..i],T_2[l(j_1)..j])$ may be smaller than k, where $i_1 \in anc(i)$ and $j_1 \in anc(j)$. Our next lemma shows that we don't have to worry about such terms.

Lemma 6:

If $D(T_1[1..i],T_2[1..j])>k$, then in any mapping from $T_1[1..i']$ to $T_2[1..j']$ such that $D(T_1[1..i'],T_2[1..j'])\leq k$, no minimal mapping from $T_1[l(i_1)..i]$ to $T_2[l(j_1)..j]$ will be used (i.e. it will not be a submapping).[2]

Proof:

By contradiction. If a minimal mapping from $T_1[1..i']$ to $T_2[1..j']$ uses any minimal mapping from $T_1[l(i_1)..i]$ to $T_2[l(j_1)..j]$, then from the conditions a mapping must follow we know that $T_1[1..l(i_1)-1]$ must map to $T_2[1..l(j_1)-1]$. Therefore, $D(T_1[1..l(i_1)-1],T[1..l(j_1)-1]) + D(T_1[l(i_1)..i],T_2[l(j_1)..j]) \geq D(T_1[1..i],T_2[1..j])>k$ This would imply that $D(T_1[1..i'],T_2[1..j'])>k$. □

Now it is easy to see how the algorithm works. We only compute $D(T_1[l(i_1)..i],T_2[l(j_1)..j]$ when $|i-j|\leq k$. In the computation if we need to use $D(T_1[l(i_1)..i],T_2[l(j_1)..j]$ such that $|i-j|>k$, we just substitute the value $k+1$. So, the general step of the algorithm becomes:

[2] - This lemma applies to general costs, so we may do better by using it instead of replacing k by k/c as we proposed above.

```
for i:= 1 to $N_1$
  for j:= max(i − k,1) to min(i+k,$N_2$)
    for $i_1 \in$ anc(i)
      for $j_1 \in$ anc(j) begin
        inner loop computation from Basic Distance Algorithm
```

The complexity is clearly $O(k \times \min(N_1,N_2) \times L_1 \times L_2)$. But we can do better. For each $T_1[l(i_1)..i]$, there are at most 2k terms from $T_2[l(j_1)..j]$ such that $D(T_1[l(i_1)..i],T_2[l(j_1)..j] \leq k$. Therefore we can manage to have an algorithm with complexity $O(k^2 \times \min(N_1,N_2) \times \min(L_1,L_2))$.

```
for i:= 1 to $N_1$
  for j:= max(i − k,1) to min(i+k,$N_2$)
    for $i_1 \in$ anc(i)
      for $j_1 \in$ anc(j)
              such that $|(i - l(i_1)) - (j - l(j_1))| \leq k$
        inner loop computation from Basic Distance Algorithm
```

To take advantage of other heuristics, the following lemma is useful.

Lemma 7:

If $D(T_1[1..i],T_2[1..j]) - D(T_1[i+1..N_1],T_2[j+1..N_2]) > k$, then in computing $D(T_1,T_2) \leq k$ $D(T_1[l(i_1)..i],T_2[l(j_1)..j]$ will not be useful. □

In general, to use the lemma 6 one must compute 2k diagonals whereas using lemma 7 only k diagonals are needed. Another way to use this lemma is to estimate $D[T_1[1..i],T_2[1..j]$ and $D[T_1[i+1..N_1],T_2[j+1..N_2]$ using less expensive heuristics such as string matching or label counting and then to disregard unhelpful intermediate mappings.

The parallel time for this algorithm is the same as for the Basic Distance algorithm but only $O(k \times L_1 \times L_2)$ are needed.

5. Distance algorithm as a general technique

Many problems in strings can be solved with dynamic programming. Similarly, our algorithm not only applies to tree distance but also provides a way to do dynamic programming for a variety of tree problems.

Here is the general pattern (assuming a postorder traversal):

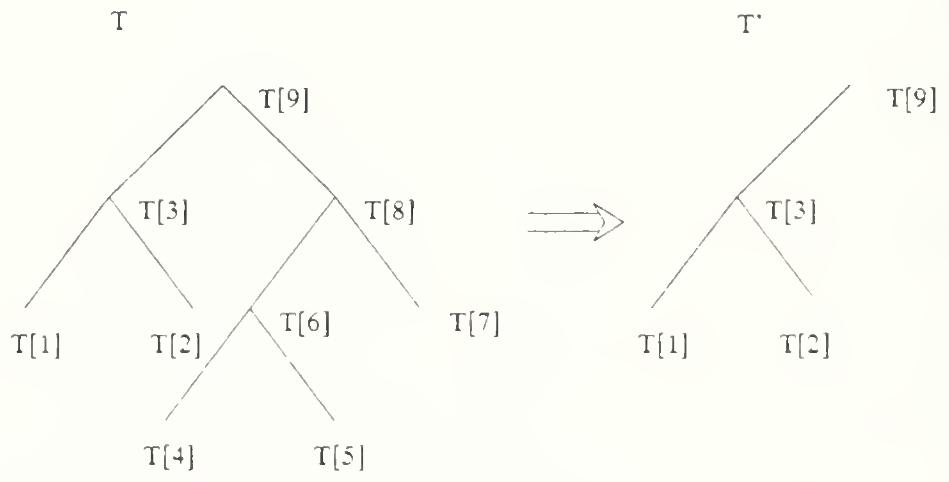
```
empty_initialization

for i:=1 to $N_1$
  for $i_1 \in$ anc(i)
    left_initialization

for j:=1 to $N_2$
  for $j_1 \in$ anc(j) begin
    right_initialization

for i:=1 to $N_1$
  for j:=1 to $N_2$
    for $i_1 \in$ anc(i)
      for $j_1 \in$ anc(j) begin
          general_term_computation
```

In the next two sections, we give four examples of apparently more complex problems that can be solved by the same technique and in the same (serial and parallel) time and space complexity as the Basic Distance algorithm.

6. Removals at a vertex

6.1. Single Remove subtrees from one tree

In this section, we consider the calculation of the minimum distance between two trees with a subtree removed from one of them.



Remove subtree rooted at T[8]

The problem is as follows: Given trees T_1 and T_2, we want to know what is the minimum distance between T_1 with a subtree removed and T_2.

Let $DR1(T_1[l(i_1) .. i], T_2[l(j_1) .. j])$ denote the minimum distance between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ such that one subtree is removed from $T_1[l(i_1) .. i]$. The following initialization and general term computation steps will give us an algorithm. Note D() is the distance in the sense of the Basic Distance Algorithm.

Algorithm Single Subtree Removal

```
empty_initialization:
$DR1(\emptyset,\emptyset)=\infty$

left_initialization:
$DR1(T_1[l(i_1)..i],\emptyset)=\min\{$
  $D(T_1[l(i_1)..l(i)-1],\emptyset),$
  $DR1(T_1[l(i_1)..i-1],\emptyset)+\gamma(T_1[i]\to\Lambda)\}$

right_initialization:
$DR1(\emptyset,T_2[l(j_1)..j])=\infty$

general_term_computation
$DR1(T_1[l(i_1)..i],T_2[l(j_1)..j])=\min\{$
  $D(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..j]),$
  $DR1(T_1[l(i_1)..i-1],T_2[l(j_1)..j])+\gamma(T_1[i]\to\Lambda),$
  $DR1(T_1[l(i_1)..i],T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j]),$
  $DR1(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1])+D(T_1[l(i)..i-1],T_2[l(j)..j-1])+\gamma(T_1[i]\to T_2[j]),$
  $D(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1])+DR1(T_1[l(i)..i-1],T_2[l(j)..j-1])+\gamma(T_1[i]\to T_2[j])\}$
```

Lemma 8: Single Subtree Removal algorithm is correct.

Proof:

First the empty_initialization and right_initialization are correct because no subtree can be removed from an empty tree.

For the left_initialization there are two cases. Either subtree $T_1[l(i)..i]$ should be removed, in which case $DR1(T_1[l(i_1)..i],\emptyset)=D(T_1[l(i_1)..l(i)-1],\emptyset)$; or a subtree in $T_1[l(i_1)..i-1]$ should be removed, in which case $DR1(T_1[l(i_1)..i],\emptyset)=DR1(T_1[l(i_1)..i-1],\emptyset)+\gamma(T_1[i]\to\Lambda)\}$

Hence the left_initialization is correct.

Now let us consider the general_term_computation.

Case (1): subtree $T_1[l(i)..i]$ is removed. So, $DR1(T_1[l(i_1)..i],T_2[l(j_1)..j])=$
$D(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..j])$

Case (2): subtree $T_1[l(i)..i]$ is not removed. Consider the best mapping between $T_1[l(i_1)..i]$ and $T_2[l(j_1)..j]$ with one subtree removed from $T_1[l(i_1)..i]$.

There are three subcases.

subcase 1: i is not in the mapping. In this case,
$DR1(T_1[l(i_1)..i],T_2[l(j_1)..j])=$
$DR1(T_1[l(i_1)..i-1],T_2[l(j_1)..j])+\gamma(T_1[i]->\Lambda),$

subcase 2: j is not in the mapping. In this case,
$DR1(T_1[l(i_1)..i],T_2[l(j_1)..j])=$
$DR1(T_1[l(i_1)..i],T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j]),$

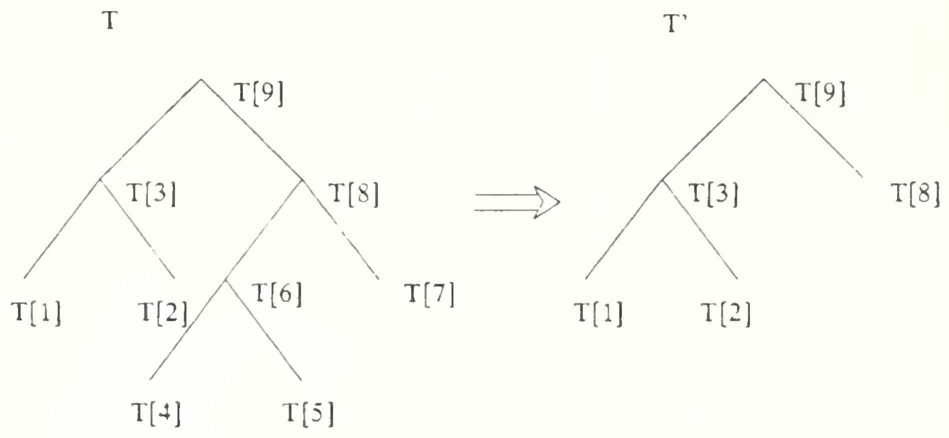
subcase 3: i and j are both in the mapping.
In this case there are two different situation.
3a: subtree is from $T_1[l(i_1)..l(i)-1]$. In this case,
$DR1(T_1[l(i_1)..i],T_2[l(j_1)..j])=$

$DR1(T_1[l(i_1) .. l(i)-1], T_2[l(j_1) .. l(j)-1]) - D(T_1[l(i) .. i-1], T_2[l(j) .. j-1]) - \gamma(T_1[i]\text{-}T_2[j])$

3b: subtree is removed from $T_1[l(i) .. i-1]$. In this case,
$DR1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$D(T_1[l(i_1) .. l(i)-1], T_2[l(j_1) .. l(j)-1]) - DR1(T_1[l(i) .. i-1], T_2[l(j) .. j-1]) - \gamma(T_1[i]\text{-}T_2[j])$ }.
□

6.2. Prune subtrees from one tree

In this section, we consider a similar problem, the calculation of the minimum distance between two trees with a pruning at a node of one of the trees. By *pruning at* T[i], we mean removing of all the proper descendants of T[i] but keeping T[i] itself. (Thus, a pruning never eliminates the entire tree.)



Pruning at T[8] -- remove all its proper descendants

Formally: Given trees T_1 and T_2, we want to know what is the minimum distance between T_1 that has been pruned at some node and T_2.

A naive application of our distance algorithm would require $O(N_1 \times$ the time to run the tree distance algorithm). We now give a algorithm to do it directly.

We need the following (slightly counterintuitive) definition. A *pruning for* $T_1[l(i_1) .. i]$ can mean
i) there is a pruning at one node in $T_1[l(i_1) .. i]$; or
ii) there is a pruning at p(i), but this is only allowed if all the proper descendants of p(i) are in $T_1[l(i_1) .. i]$.

We denote the condition by the predicate *arewithin*(i, i_1). This holds if and only if $l(i_1) \leq l(p(i))$ and i is the rightmost child of p(i).

Let $DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j])$ denote the minimum distance between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ such that there is a pruning for $T_1[l(i_1) .. i]$.

The following initialization and general term computation steps will give us an algorithm. Again, D() is the distance in the sense of the Basic Distance algorithm

Algorithm Single Prune

```
  empty_initialization:
  $DP1(\emptyset,\emptyset)=\infty$

  left_initialization:
  $DP1(T_1[l(i_1)..i],\emptyset)=$
    $DP1(T_1[l(i_1)\ ..\ i-1],\emptyset)+\gamma(T_1[i]\to\Lambda)$
  if arewithin$(i,i_1)$ then
    $DP1(T_1[l(i_1)..i],\emptyset)=\min\{$
      $DP1(T_1[l(i_1)..i],\emptyset),$
      $D(T_1[l(i_1)..l(p(i))-1],\emptyset)\}$

  right_initialization:
  $DP1(\emptyset,T_2[l(j_1)..j])=\infty$

  general_term_computation
  $DP1(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j])=\min\{$
    $DP1(T_1[l(i_1)\ ..\ i-1],T_2[l(j_1)\ ..\ j])+\gamma(T_1[i]\to\Lambda),$
    $DP1(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j-1])+\gamma(\Lambda\to T_2[j]),$
    $DP1(T_1[l(i_1)\ ..\ l(i)-1],T_2[l(j_1)\ ..\ l(j)-1])+D(T_1[l(i)\ ..\ i-1],T_2[l(j)\ ..\ j-1])+\gamma(T_1[i]\to T_2[j]),$
    $D(T_1[l(i_1)\ ..\ l(i)-1],T_2[l(j_1)\ ..\ l(j)-1])+DP1(T_1[l(i)\ ..\ i-1],T_2[l(j)\ ..\ j-1])+\gamma(T_1[i]\to T_2[j])\ \}$
  if arewithin$(i,i_1)$ then
    $DP1(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j])=\min\{$
      $DP1(T_1[l(i_1)\ ..\ i],T_2[l(j_1)\ ..\ j]),$
      $D(T_1[l(i_1)\ ..\ l(p(i))-1],T_2[l(j_1)\ ..\ j])\}$
```

Lemma 9: Algorithm Single Prune is correct.

Proof:

First the empty_initialization and right_initialization are correct because no pruning can occur on an empty tree.

For the left_initialization there are two cases. If all descendants of $p(i)$ are in $T_1[l(i_1)\ ..\ i]$, we can prune at $p(i)$. That means we remove $T_1[l(p(i))\ ..\ i]$. In this case $DP1(T_1[l(i_1)\ ..\ i],\emptyset)=D(T_1[l(i_1)\ ..\ l(p(i))-1],\emptyset)$. Otherwise, we can prune for $T_1[l(i_1)\ ..\ i-1]$, giving cost $DP1(T_1[l(i_1)..i],\emptyset)=DR1(T_1[l(i_1)\ ..\ i-1],\emptyset)+g(T_1[i]\to\Lambda)\}$.

Hence the left_initialization is correct.

Now let us consider the general_term_computation.

First there are two cases:
Case (1): $T_1[l(p(i)) .. i]$ is removed. So,
$DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$\quad D(T_1[l(i_1) .. l(p(i)) - 1], T_2[l(j_1) .. j])$
Note: this case is conditional, depending on if all the descendants of $p(i)$ are in $T_1[l(i_1) .. i]$, i.e. arewithin(i, i_1).

Case (2): $T_1[l(p(i)) .. i]$ is not removed.
Consider the best mapping between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ with a pruning at a node in $T_1[l(i_1) .. i]$.
There are three subcases.

subcase 1: i is not in the mapping. In this case,
$DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$\quad DP1(T_1[l(i_1) .. i - 1], T_2[l(j_1) .. j]) + \gamma(T_1[i] \to \Lambda)$,

subcase 2: j is not in the mapping. In this case,
$DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$\quad DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j - 1]) + \gamma(\Lambda \to T_2[j])$.

subcase 3: i and j are both in the mapping.
In this case there are two different situations.

3a: There is a pruning for $T_1[l(i_1) .. l(i) - 1]$. In this case,
$DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$\quad DP1(T_1[l(i_1) .. l(i) - 1], T_2[l(j_1) .. l(j) - 1]) + D(T_1[l(i) .. i - 1], T_2[l(j) .. j - 1]) + \gamma(T_1[i] \to T_2[j])$

3b: There is a pruning for $T_1[l(i) .. i - 1]$. In this case,
$DP1(T_1[l(i_1) .. i], T_2[l(j_1) .. j]) =$
$\quad D(T_1[l(i_1) .. l(i) - 1], T_2[l(j_1) .. l(j) - 1]) + DP1(T_1[l(i) .. i - 1], T_2[l(j) .. j - 1]) + \gamma(T_1[i] \to T_2[j])$ }

□

7. Approximate tree matching

We consider here approximate tree matching. Hoffman and O'Donnell [HO-82] have proposed an algorithm for exact tree matching. To generalize the problem, we first consider approximate matching [S-80, U-83, U-85, LV-86] for strings. The problem is to compute, for each i, the minimum number of editing operations between the "pattern" string PAT[1..|PAT|] and the "text string" TEXT[1..i] with a prefix removed (from TEXT). (Intuitively, the algorithm finds the "occurrence" in TEXT that most closely matches PAT.)

To study this problem to trees, we must generalize the notion of prefix. For us, a prefix will mean a collection of subtrees. These subtrees can be arbitrary or can arise as a result of zero or more prunings (section 6.2). We consider each generalization in turn.

7.1. Remove any number of subtrees from TEXT tree

The problem is as follows: Given trees T_1 and T_2, we want to know what is the minimum distance between T_1 with zero or more subtrees removed and T_2.

Let $DR(T_1[l(i_1) .. i], T_2[l(j_1) .. j])$ denote the minimum distance between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ with zero or more subtrees removed from $T_1[l(i_1) .. i]$.

Algorithm Many Subtree Removal

```
empty_initialization:
$DR(\varnothing,\varnothing)=0$

left_initialization:
$DR(T_1[l(i_1)..i],\varnothing)=0$

right_initialization:
$DR(\varnothing,T_2[l(j_1)..j])=DR(\varnothing,T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j])$

general_term_computation
$DR(T_1[l(i_1)..i],T_2[l(j_1)..j])=\min\{$
  $DR(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..j])$,
  $DR(T_1[l(i_1)..i-1],T_2[l(j_1)..j])+\gamma(T_1[i]\to\Lambda)$,
  $DR(T_1[l(i_1)..i],T_2[l(j_1)..j-1])+\gamma(\Lambda\to T_2[j])$,
  $DR(T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1])+DR(T_1[l(i)..i-1],T_2[l(j)..j-1])+\gamma(T_1[i]\to T_2[j])\ \}$
```

Lemma 10: Algorithm Many Subtree Removal is correct.

Proof:

First we show that the initialization is correct. The empty-initialization and the right_initialization is the same as in the tree distance algorithm. The left-initialization $DR(T_1[l(i_1)..i],\varnothing)=0$ is correct, because we can remove all of $T_1[l(i_1)..i]$.

For the general term $DR(T_1[l(i_1) .. i],T_2[l(j_1) .. j])$, we ask first whether the subtree $T_1[l(i) .. i]$ is removed or not. If it is removed, then the distance should be $DR(T_1[l(i_1) .. l(i) - 1],T_2[l(j_1) .. j])$ giving the first term of the minimization. Otherwise, consider the mapping between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ after we perform an optimal removal of subtrees of $T_1[l(i_1) .. i]$. To compute this mapping, we have the same three cases as in the tree distance algorithm. Hence the general term should be the minimum of above four terms. □

7.2. Prune at any number of nodes from the TEXT tree

Given trees T_1 and T_2, we want to know what is the minimum distance between T_1 and T_2 when there have been zero or more prunings at nodes of T_1.

Let $DP(T_1[l(i_1) .. i], T_2[l(j_1) .. j])$ denote the minimum distance between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ with zero or more prunings for $T_1[l(i_1) .. i]$. (Refer to section 6.2 for the definition of "pruning for".) The following initialization and general term computation steps will give us an algorithm to solve our problem.

Algorithm Many Prunings

```
empty_initialization:
DP($\emptyset$,$\emptyset$)=0

left_initialization:
DP($T_1[l(i_1)..i],\emptyset$)=
  DP($T_1[l(i_1)..i-1],\emptyset$)+$\gamma(T_1[i]\to\Lambda)$
if arewithin($i,i_1$) then
  DP($T_1[l(i_1)..i],\emptyset$)=
    DP($T_1[l(i_1)..l(p(i))-1],\emptyset$)

right_initialization:
DP($\emptyset,T_2[l(j_1)..j]$)=DP($\emptyset,T_2[l(j_1)..j-1]$)+$\gamma(\Lambda\to T_2[j])$

general_term_computation
DP($T_1[l(i_1)..i],T_2[l(j_1)..j]$)= min{
  DP($T_1[l(i_1)..i-1],T_2[l(j_1)..j]$)+$\gamma(T_1[i]\to\Lambda)$,
  DP($T_1[l(i_1)..i],T_2[l(j_1)..j-1]$)+$\gamma(\Lambda\to T_2[j])$,
  DP($T_1[l(i_1)..l(i)-1],T_2[l(j_1)..l(j)-1]$)+DP($T_1[l(i)..i-1],T_2[l(j)..j-1]$)+$\gamma(T_1[i]\to T_2[j])$ }
if arewithin($i,i_1$) then
  DP($T_1[l(i_1)..i],T_2[l(j_1)..j]$)= min{
    DP($T_1[l(i_1)..i],T_2[l(j_1)..j]$),
    DP($T_1[l(i_1)..l(p(i))-1],T_2[l(j_1)..j]$)}
```

Lemma 11: Algorithm Many Prunings is correct.

Proof:

First we show that the initialization is correct. The empty-initialization and the right_initialization is the same as in the tree distance algorithm. For left-initialization, if we can remove $T_1[l(p(i)) .. i]$ (prune at $p(i)$) then $DP(T_1[l(i_1)..i],\emptyset)=DP(T_1[l(i_1)..l(p(i))-1],\emptyset)$. Otherwise $DP(T_1[l(i_1)..i],\emptyset)=DP(T_1[l(i_1)..i-1],\emptyset)+\gamma(T_1[i]\to\Lambda)$. Hence the left_initialization is correct.

For the general term $DP(T_1[l(i_1) .. i],T_2[l(j_1) .. j])$, we ask first whether $T_1[l(p(i)) .. i]$ is removed or not. If it is removed, then the distance should be $DP(T_1[l(i_1) .. l(p(i))-1],T_2[l(j_1) .. j])$ giving the first term of the minimization. Otherwise, consider the mapping between $T_1[l(i_1) .. i]$ and $T_2[l(j_1) .. j]$ after we perform an optimal number of prunings for $T_1[l(i_1) .. i]$. Now we have the same three cases as in the tree distance algorithm. Hence the general term should be the minimum of above four terms. □

Let us now condiser the problem of approximate tree matching. In above algorithm, let T_1 be TEXT and T_2 be PAT and set all cost be 1. We have now a algorithm for approximate tree matching. The result is in D(TEXT[l(i) .. i],PAT[1 .. N_2]), where $1 \le i \le N_1$. Note that if i is not only child of its parent, we need to check if D($\emptyset$,PAT[1 ..N_2]) is smaller than D(TEXT[l(i) .. i],PAT[1 .. N_2]).

8. Conclusion

We present a simple dynamic programming algorithm for tree distance which
1. has better time and space complexity than any in the literature;
2. is efficiently parallelizable;
3. can be specialized to the k-distance problem for trees with much improved efficiency; and
4. is generalizable to approximate tree matching problems.

Our research suggests two broad avenues for further algorithmic work. First, we would like to generalize these algorithms to unordered trees. Second, we would like to consider distance metrics other than editing distance

Acknowledgement

We would like to thank Bob Hummel for helpful discussions.

References

HO-82.
C. M. Hoffmann and M. J. O'Donnell, "Pattern matching in trees," *J. ACM* **29, No. 1** pp. 68-95 (1982).

L-79.
S. Y. Lu, "A tree-to-tree distance and its application to cluster analysis," *IEEE Trans. on PAMI* **PAMI-1, No. 2** pp. 219-224 (1979).

LV-86.
G. M. Landau and U. Vishkin, "Introducing efficient parallelism into approximate string matching and a new serial algorithm," *Proc. 18th ACM Syposium on Theory of Computing*, pp. 220-230 (1986).

S-80.
P.H. Sellers, "The theory and computation of evolutionary distances," *J. of Algorithm* **1** pp. 359-373 (1980).

T-79.
Kuo-Chung Tai, "The tree-to-tree correction problem," *J. ACM* **26** pp. 422-433 (1979).

U-83.
E. Ukkonen, "On approximate string matching," *Proc. Int. Conf. Found. Comp. Theor., Lecture notes in Computer Science* **158** pp. 487-495 Springer-Verlag, (1983).

U-85.
E. Ukkonen, "Finding approximate pattern in strings," *J. of Algorithms* **6** pp. 132-137 (1985).

WF-74.
R. Wagner and M. Fisher, "The string-to-string correction problem," *J. ACM* **21** pp. 168-178 (1974).

Z-83.
Kaizhong Zhang, *An algorithm for computing similarity of trees*, Technical Report, Mathematics Department Peking University 1983.

NYU COMPSCI TR-310 c.1
Zhang, Kaizhong
On the editing distance
between trees and related
problems

NYU COMPSCI TR-310 c.1
Zhang, Kaizhong
On the editing distance
between trees and related
problems

JUN 2 8 1988	T L Wang

This book may be kept
FOURTEEN DAYS
A fine will be charged for each day the book is kept overtime.

JAN 2 5 1993

GAYLORD 142 PRINTED IN U S A