Editing Distance in Trees

Sourav Chakraborty*
Chennai Mathematical Institute, Chennai, India
sourav@cmi.ac.in

[*Abstract*: We will define the problem of editing distance in trees. But since it is known that the problem of editing distance in unordered trees is NP-complete so it is unlikely to find efficient algorithm for the problem. So we try to solve the problem by putting various restrictions on the structure of the trees or on the kind of edit operations we are allowed. We will give efficient polynomial type algorithms for this problem with various restriction imposed on the trees and we will see some related problems. Finally we will try to use new techniques to improve the algorithms and to generalize our algorithms for bigger classes of graphs.]

1 Introduction

The problem of *editing distance in strings* is a very important problem mainly due to its application to various other branches of study. The problem is as follows:-

Let S_1 and S_2 be two strings over the alphabet Σ. We are allowed to do three kinds of operations on the first string. We can change a letter in the string to another letter or we can delete a letter in the string or we can insert a letter at any point in the string. Using these three operations we have to change the first string S_1 to the second string S_2. The problem is, what is the minimum number of such operations required.

This problem, other than being very interesting by its own self, is very useful in various other branches of study including molecular biology, chemistry, data handling and computer programming. In fact polynomial time algorithms have been found for this problem. The best known algorithm uses $o(|S_1| \times |S_2|)$ time.

But more generalized version of this problem can be more useful and more interesting. In the next few sections we will introduce the problem of editing distance in trees and try to solve it efficiently. It is our goal to generalize this problems to planar graphs and then finally to graphs in general.

*This is the study done by me when I visited Paris as a part of the exchange program between Chennai Mathematical Institute, Chennai, India and École Normale Supérieure, Paris, France.

2 Editing Distance in Trees

Just like the problem of editing distance in strings we are given two *labeled* trees T_1 and T_2 and three kinds of operations that can be done on the first tree. A *labelled tree* is a tree in which there is a label in each node of the tree.

The three operations that can be done on the first tree are as follows:-

(1) Change the label in a node to another label.
(2) Delete a node and all its children becomes the children of the parent of the deleted node.
(3) Insert a node at any place and a (consecutive) sequence of siblings becomes its children.

Remark 1 *The word "consecutive" in the third operation mentioned above only has meaning when we are working with ordered trees, that is in trees in which the left to right ordering of the nodes are important.*

There is a cost associated to each of the above operations. And the problem is to change the tree T_1 to T_2 in minimum total cost possible. The cost is actually a metric. Even in case of *editing distance in strings* often different costs are associated to different edit operations. And then the problem is to change one string to other using total cost. Even this problem in strings has been solved in polynomial time and best known algorithm for this also uses only $o(|N_1| \times |N_2|)$ time complexity.

Actually this problem is $NP - complete$[5] when no restriction on the structure of the trees are imposed. So we will put appropriate restrictions on the structure of the trees so that we get efficient algorithms to solve the problem.

Remark 2 *Due to the fact that the problem of editing distance in unordered labeled trees is NP-Complete, the problem of editing distance in general graphs or even in general planar graph also becomes NP-hard.*

First we will assume that the trees with which we will work are ordered (definition already given in the remark above). Then we will put some restriction on the kind of operations we are allowed. This gives us nice algorithms with better time complexity. Finally we will show that with the constrained edit operations we can even give efficient algorithms for unordered tree.

But before we do our algorithms it is important to set our notations appropriately.

Notations

First of all we use γ as a cost function. $\gamma(a \to b)$ is the cost of the operation of changing the node a to node b. But if a is Λ then it means inserting node b and if the b is Λ then it means deleting node a.

By abuse of notation we also define the cost of editing distance also as γ. Thus $\gamma(A, B)$ means the minimum cost of editing A to B, where A and B are forests.

We will denote trees by T_1 and T_2 and the ith node by $T_1[i]$. And we denote the set of nodes in tree T_1 by N_1.

The subtree of T_1 rooted at node i is denoted by $Tree_1[i]$ and the forest got by deleting the root from the subtree rooted at i is denoted by $Forest_1[i]$.

Also for our help we denote $\gamma(Tree_1[i], Tree_2[j])$ by $Treedist(i, j)$.

We will be using post-ordering in numbering the nodes of the trees. So for $i \leq j$, $F[i..j]$ will denote the forest induced by the nodes numbered i to j. $T_1[i]$ will denote the node with number i. And $l(i)$ will denote the smallest numbered node in the tree $Tree_1[i]$.

So after setting our notations correctly we can proceed to do our algorithms.

2.1 Editing Distance for Ordered Trees

This algorithm is from the paper "*Simple fast algorithms for the editing distance between trees and related problems*"[1] by Kaizhong Zhang and Dennis Shasha.

In this algorithm we assume that the trees are ordered, that is the left to right ordering of the nodes are important.

First of all we will show a relation between the edit distance and a mapping. Given trees T_1 and T_2 a mapping is a subset of $N_1 \times N_2$ (where N_i is the nodes of tree T_i) with the following properties:-

(1) The mapping is one-to-one.
(2) It preserves the left to right ordering.
(3) It preserves the ancestor ordering.

Now given a mapping M we can associate a cost to it by:-

$\gamma(M) = \Sigma_{(i,j)\in M}\gamma(T_1[i] \to T_2[j]) + \Sigma_{i\in N_1}\gamma(T_1[i] \to \Lambda) + \Sigma_{j\in N_2}\gamma(\Lambda \to T_2[j])$

Now we will see that a mapping is equivalent to an edit distance.

It is easy to see that by the metric property of the cost of edit operations the mapping also becomes a metric.

First of all a mapping gives rise to a canonical edit distance and the cost of the edit distance is the same as that of the edit mapping. Again since an individual edit operation is one-one and also preserves left-right ordering and ancestor ordering so it becomes a mapping. Hence by induction and using the property that the mapping is a metric we can see that given a edit distance there is a mapping of cost less than or equal to the edit distance. Thus mapping and edit distance are equivalent. So from now on we will work with mappings instead of edit distance.

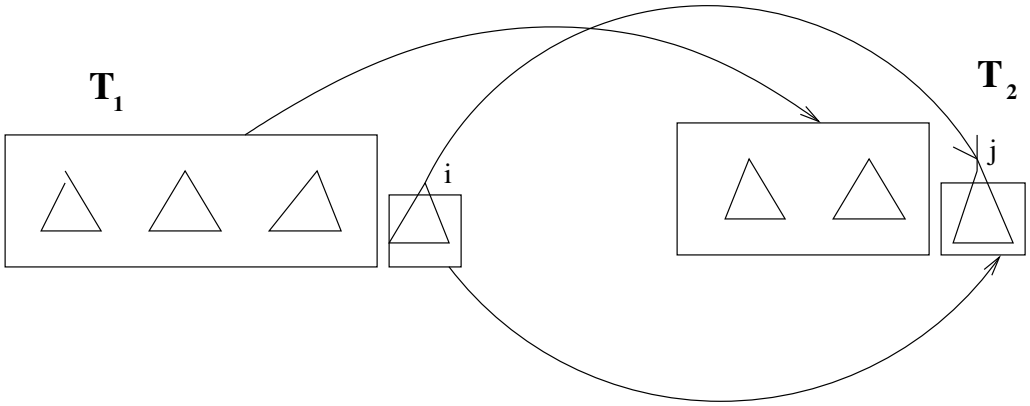
Properties of the mappings on Ordered Trees The idea is to apply dynamical programming to make the programming efficient. So we will try to find some properties so that dynamical programming can be used.

Lemma 1 *Let i_1 be an ancestor of i and j_1 be an ancestor of j. Then, $\gamma(l(i_1)..i, l(j_1)..j)$ is the minimum of the following:*

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

Proof: Since the mapping preserves left-right ordering and ancestor ordering so in case of $\gamma(l(i_1)..i, l(j_1)..j)$ if both $T_1[i]$ and $T_2[j]$ are in the mapping then the only possibility is that $T_1[i]$

is mapped to $T_2[j]$ that is the third case. Otherwise either $T_1[i]$ has to be deleted (the first case) or $T_2[j]$ has to be inserted (the second case). □



From the above lemma it follows that if $l(i) = l(i_1)$ and $l(j) = l(j_1)$ then
$\gamma(l(i_1)..i, l(j_1)..j)$ is the minimum of the following:

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

else
$\gamma(l(i_1)..i, l(j_1)..j)$ is the minimum of the following:

(1) $\gamma(l(i_1)..i-1, l(j_1)..j) + \gamma(T_1[i] \rightarrow \Lambda)$
(2) $\gamma(l(i_1)..i, l(j_1)..j-1) + \gamma(\Lambda \rightarrow T_2[j])$
(3) $\gamma(l(i_1)..l(i)-1, l(j_1)..l(j)-1) + Treedist(i,j)$

So we observe that to compute $Treedist(i_1, j_1)$ we need to know the $Treedist(i,j)$ for all i, j such that i is in the subtree rooted at i_1 and j is in the subtree rooted at j_1. In fact we have to find the $Treedist(i,j)$ in a bottom up fashion. Then computing $Treedis(i,j)$ for all i, j takes constant steps.

Also we can observe that $Treedist(i,j)$ is got as a byproduct of $Treedist(i_1, j_1)$ if $l(i)$ is same as $l(i_1)$ and $l(j)$ is same as $l(j_1)$.

So we can use the above lemma to get our algorithm. Let I be the set of nodes in T_1 such that no other node in the tree has the same leftmost child and similarly let J be the set of nodes in T_2 such that no other node in the tree has the same leftmost child. Then in the algorithm we have to compute $Treedist(i,j)$ for all $i \in I$ and for all $j \in J$.

Complexity Since we have to store the result of $Treedist(i,j)$, we need $o(|N_1| \times |N_2|)$ space.

Now, for the case of time complexity since we find the $Treedist(i,j)$ of all the $i \in I$ and $j \in J$, and the time complexity will be $\Sigma_{i \in I} \Sigma_{j \in J} (Size\ of\ tree\ rooted\ at\ i) \times (Size\ of\ tree\ rooted\ at\ j)$.

This is actually $o(|N_1| \times |N_2| \times min[depth(T_1), leaves(T_1)] \times min[depth(T_1), leaves(T_1)])$.

So we have a pretty efficient algorithm for edit distance in ordered tree. But now we will try to see if by putting some other constrained we can remove the constrained of left-right "*order*" in the trees.

2.2 Constrained Editing Distance in Ordered Trees

After having got an efficient algorithm by putting the restriction of ordering in the trees we would like to see if we can get a more efficient algorithm by putting some other restriction other than the restriction of ordering. But for that we first put another constraint and in the next section we will see that it would be possible to remove the restriction of ordering by keeping the new constraint. This time the constraint is not on the structure of the tree but on the mapping.

The constraint we impose is that different subtrees will go to different subtrees and different subtrees will come from different subtrees. The exact formal definition can be found in the paper *Algorithms for the Constrained Editing Distance between Ordered Labeled Trees and Related Problems*[2] by Kaizhong Zhang.

With the above constraint we will find some important properties of mapping. First of all it is trivially seen that the mapping gives a new definition for distance and it is clear that the distance is also a metric.

Lemma 2 *Let* $t_1[i_1], t_1[i_2], ..., t_1[i_n]$ *be the children of the node* $T_1[i]$ *and* $t_2[j_1], t_2[j_2], ..., t_2[j_m]$ *be the children of the node* $T_2[j]$ *then* $\gamma(Tree_1[i], Tree_2[j])$ *is the minimum of the following three.*

(1) $\gamma(\phi, Tree_2[j]) + min_{1 \le t \le m}[\gamma(Tree_1[i], Tree_2[j_t]) - \gamma(\phi, Tree_2[j_t])]$
(2) $\gamma(Tree_1[i], \phi) + min_{1 \le s \le n}[\gamma(Tree_1[i_s], Tree_2[j]) - \gamma(Tree_1[i_s], \phi)]$
(3) $\gamma(Forest_1[i], Forest_2[j]) + \gamma(T_1[i] \rightarrow T_2[j])$

Proof:- The lemma follows from the definition of constrained mapping. What the constrained mapping says that either the $Tree_1[i]$ is mapped to $Tree_2[j]$ by sending the $T_1[i]$ to $T_2[j]$ and the remaining forest in the first tree to the remaining forest in the second tree (which is the third case) or $T_1[i]$ will be mapped to a subtree rooted at a children of $T_2[j]$ and the remaining of the second tree is created (which is the first case) or the vise versa (which is the second case). □

Now let SM be a *Special Mapping* from a forest to another forest in which the ordered of the tree in the trees is maintained. So we can speak of the minimum cost of such a map.

Based on the same argument of the previous lemma we get the following lemma:-

Lemma 3 $\gamma(Forset_1[i], Forest_2[j])$ *is the minimum of the following.*

(1) $\gamma(\phi, Forest_2[j]) + min_{1 \le t \le n}[\gamma(Forest_1[i], Forest_2[j_t]) - \gamma(\phi, Forest_2[j_t])]$
(2) $\gamma(Forest_1[i], \phi) + min_{1 \le s \le m}[\gamma(Forest_1[i_s], Forest_2[j]) - \gamma(Forest_1[i_s], \phi)]$
(3) $min[\gamma[SM(Forest_1[i], Forest_2[j])]$

Using the above two lemmas we have got an algorithm if we can find out an efficient algorithm to find out the SM. But that turns out to be identical to the *editing distance in strings*. Actually we have two sequence of trees and we have to change the first sequence to the other. Here the cost of changing a label is same as changing tree at that place to the other. So using the two lemmas and the problem of editing distance in strings we have got a algorithm.

Complexity Since we have to store $\gamma(Tree_1[i], Tree_2[j])$ for all i, j and also $\gamma(Forest_1[i], Forest_2[j])$ for all i, j so we need $o(|N_1| \times |N_2|)$ space.

For the time complexity we can see that for computing $SM(Forest_1[i], Forest_2[j]$ we need $o(n_i, n_j)$ time where n_i is the number of children of $T_1[i]$.

So the time complexity is $\Sigma_{i \in N_1} \Sigma_{j \in N_2} (n_i \times n_j)$ which by easy computation is seen to be $o(|N_1| \times |N_2|)$ which is an improvement to our earlier algorithm that we did in the last section.

2.3 Constrained Editing Distance in Unordered Trees

In section we show that using the constrained editing distance it is possible to get efficient algorithms even without the restriction of ordering on the trees. For this we just need to change our last section a little bit. This algorithm is from the paper *A constrained editing distance between unordered labeled trees*[3] by Kaizhong Zhang.

The *Lemma*2 of the last section holds good without the restriction of left-right ordering on the trees. But the *Lemma*3 uses the fact that we are working with ordered trees and so we used the *Special Mapping* among the forests. But when we are working with unordered trees let us give a matching from tree $Tree_1[i]$ to tree $Tree_2[j]$ and let us match $T_1[i]$ to $T_2[j]$. Then by the definition of constrained editing distance we have to individual subtrees $Tree_1[i_s]$ mapped to $Tree_2[j_t]$ for some i and j but ordered of the subtrees need not be preserved. Let us call this *Restricted Special Mapping RSM*.

Lemma 4 *$\gamma(Forset_1[i], Forest_2[j])$ is the minimum of the following.*

(1) $\gamma(\phi, Forest_2[j]) + min_{1 \leq t \leq n}[\gamma(Forest_1[i], Forest_2[j_t]) - \gamma(\phi, Forest_2[j_t])]$

(2) $\gamma(Forest_1[i], \phi) + min_{1 \leq s \leq m}[\gamma(Forest_1[i_s], Forest_2[j]) - \gamma(Forest_1[i_s], \phi)]$

(3) $min[\gamma[RSM(Forest_1[i], Forest_2[j])]$

Now we have to give an efficient algorithm to calculate the RSM. It turns out that it is similar to the Max flow problem.

Let $\mathcal{A} = \{i_1, ..., i_m, e\}$ and let $\mathcal{B} = \{j_1, ..., j_n, f\}$. We have to find a set of lines from $\mathcal{A}$ to $\mathcal{B}$ with all i_s and j_t using exactly once and e and f acting as empty trees. So we can make a network using these and cost of a line form i_s to j_t is $\gamma(Tree_1[i_s], Tree_2[j_t])$ and a line from e to j_t has cost $\gamma(\phi, Tree_2[j_t])$. Similarly the cost of a line from i_s to f is $\gamma(Tree_1[i_s], \phi)$ and the cost of line from e to f is 0.

So we have got a network and we have to find out the maximum flow using minimum cost. And for this efficient algorithms are already known. So we have got an efficient algorithm to solve the RSM.

So we have got an efficient algorithm to solve the constrained editing distance problem for unordered trees.

This is a very important observation as it says that although the Editing Distance Problem in trees is $NP - Complete$ for unordered trees, the constrained editing problem for unordered trees is in P. This gives us some hope of solving the problem for bigger classes of graphs using this constrained edit operations.

3 Further Studies

Our main aim is to define properly the problem of *Editing distance in Graphs*. But before that we would like to try the problem in trees from a different angle and see whether that gives us any help to generalize the algorithm to graphs or even to planar graphs.

Instead of using dynamical programming we would like to try the problem from a different angle. So for that a program was written implementing the algorithm for *Editing Distance in Ordered Trees*. The program was then run on small inputs. That is the trees with number of edges 3. The trees were ordered in a canonical way. Then the *editing distance* between all pairs of trees with number of edges 3 was computed using the algorithm. It gave a matrix. Similar thing was done for trees with number of edges 4. The idea was to observe the matrixes and see if the matrixes can be produced by some other better way, if possible by some algebraic equation.

After some discussion certain progress was made as an algebraic expression was found for the first row of the matrixes. This was done by using *Murphy's Bases of Hecke Algebras*.

The study is in progress and new results are expected and we hope to get a better algorithm. In fact since we will be using the algebraic properties of the trees so it may also be possible to generalize this idea to larger classes of graphs like the planar graph (but obviously with some kind of restrictions on the structure of the graphs or on the edit operations).

4 Conclusion

As we see that due to the fact that the problem of editing distance in unordered trees is NP-complete it is a very challenging and interesting problem to find out polynomial time algorithm of this problem by imposing various restrictions. The problem on graphs or on planar graphs promises to be more interesting. The recent idea of using algebra to solve the problem may be useful and help us to solve it.

Since the actual problem is in NP-complete so it might be good to try out *approximation algorithm* to solve it. But unfortunately it is known that the problem is even Max SNP-hard[4], meaning no good approximation algorithm can be expected of it. But still by imposing certain restrictions some approximation algorithm has been made even for the problem of *approximate graph matching* which is a problem very closely related to the problem of *editing distance in graphs*[6].

So this problem is a highly open problem with lots of scope of work on this problem.

5 Acknowledgment

I will like to thank Dr Anne Micheli for introducing me to this subject and helping me get a clear idea of the subject by listening to my lectures carefully. I would also like to thank Dr Dominique Rossin, Dr Dominique Poulalhon and Dr Daniel Krob and Dr Anne Micheli for the discussions we had on this problem which helped me understand this subject in a much better way. I will like to thank Stephane Fischler and other people of École Normale Supérieure for helping me a lot during my stay in Paris. Finally I thank my institute Chennai Mathematical Institute, Chennai, India without whom I would not have got the opportunity to come to Paris and do this study.

References

[1] K.Zhang and D.Shasha, *Simple fast algorithm for the editing distance between trees and related problems*, published in *SIAM J.Computing* 18(6), 1245-1262(1989).

[2] K.Zhang, *Algorithms for the Constrained Editing Distance Between Ordered Labeled Trees and Related Problems*, published in *Technical Report* No.361, Department of Computer

Science, University of Western Ontario, 1993.

[3] K.Zhang, *A constrained editing distance between unordered labeled trees*, published in *Algorithmica* (1996) 15: 205-222..

[4] K.Zhang and Tao Jiang, *Some Max SNP – hard results concerning unordered labeled trees*, published in *Inform.Process.Lett.*, 49(1994), 249-254.

[5] K.Zhang, R.Statman and D.Shasha, *On the editing distance between unorderd labeled trees*, published in *Inform.Process.Lett.*, 42(1992), 133-139.

[6] K. Zhang, Jason T.L.Wang and Gung-Wei Chirn, *Algorithms for Approximate Graph Matching* published in *Information Sciences* 82, 45-74(1995).