The Noisy Subsequence Tree Recognition Problem

B. J. Oommen[1] and R. K. S. Loke

School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6.
e-mail address : oommen@scs.carleton.ca

ABSTRACT

In this paper we consider the problem of recognizing ordered labeled trees by processing their noisy subsequence-trees which are "patched-up" noisy portions of their fragments. We assume that we are given H, a finite dictionary of ordered labeled trees. X^* is an unknown element of H, and U is any arbitrary *subsequence-tree* of X^*. We consider the problem of estimating X^* by processing Y - a noisy version of U. We do this by sequentially comparing Y with every element X of H, the basis of comparison being the constrained edit distance between two trees [OL94], where the constraint implicitly captures the properties of the corrupting mechanism ("channel") which noisily garbles U into Y. Experimental results which involve manually constructed trees of sizes between 25 and 35 nodes and which contain an average of 21.8 errors per tree demonstrate that the scheme has about 92.8% accuracy. Similar experiments for randomly generated trees yielded an accuracy of 86.4%. To our knowledge this is the first reported solution to the problem.

I. INTRODUCTION

In this paper, we consider the following problem : Suppose we have a finite dictionary of labeled ordered trees, H. Let X^* be any tree from H. U is an *arbitrary* Subsequence-Tree (SuT) of X^* obtained by randomly deleting nodes from it. The resultant tree (called a subsequence-tree or SuT of X^*) is further subjected to substitution, insertion and deletion errors yielding the *Noisy Subsequence-Tree* (NSuT), Y. Our aim is then to identify the original tree, X^*, by processing Y.

Unlike the string-editing[2] problem, only few results have been published concerning the tree-editing problem. In 1977 Selkow [Se77, SK83] presented a tree editing algorithm in which insertions and deletions were only restricted to the leaves. Tai [Ta79] in 1979 presented another algorithm in which insertions and deletions could take place at any node within the tree except the root. The algorithm of Lu [Lu79] 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 [ZS89]. Also, to the best of our knowledge, in all the papers published till the mid-90's, the literature primarily contains only one numeric inter-tree dissimilarity measure - their pairwise "distance" measured by the minimum cost edit sequence.

---

[1]Senior Member IEEE.

[2]The literature on string editing is extensive. We refer the readers to the book written by Sankoff and Kruskal [SK83] and the proceedings of the recent symposia on Combinatorial Pattern Matching (CPM) for the state of the art techniques in sequence processing.

The literature on the comparison of trees is otherwise scanty : Zhang *et al.* [SZ90] have suggested how tree comparison can be done for ordered and unordered labeled trees using tree alignment as opposed to the edit distance utilized elsewhere [ZS89]. The question of comparing trees with variable length don't care operations was solved by Zhang *et. al.* [ZSW92]. Besides these, the results concerning unordered trees are primarily complexity results [ZSS92] - editing unordered trees with bounded degrees is shown to be NP-hard in [ZSS92] and even MAX SNP-hard in [ZJ94]. The most recent results concerning tree comparisons is probably the one due to Oommen *et. al.* [OZL96] where the authors defined and formulated an abstract measure of comparison, $\Omega(T_1, T_2)$, between two trees.

The problem of comparing a tree with one of its possible subtrees or SuTs has almost not been studied in the literature at all. The only reported results for comparing trees in this setting have involved constrained tree distances [OL94] and indeed, this we will be foundational basis for the PR of noisy SuTs.

The primary contribution of the paper is the application of the constrained tree distance for the NSuT recognition problem. This is achieved by considering the information about the noise characteristics of the channel which garbles a tree. Indeed, these characteristics are translated into edit constraints whence a constrained tree editing algorithm can be invoked to perform the classification. Besides these, our paper suggests a new perspective for generalized computation models. Unfortunately, we cannot extend the results concerning subsequence correction to this present problem because, as opposed to strings [Oo87], the topological structure of the underlying graph *prohibits* the two-dimensional generalizations of the corresponding computations. Thus, inter-tree computations require the simultaneous maintenance of *meta-tree* considerations represented as the parent and sibling properties of the respective trees, which are completely ignored in the case of linear structures (e.g., strings). The proofs of the results claimed are omitted here but included in [OL94].

II. NOTATIONS AND DEFINITIONS

II.1 Notation

Let N be an alphabet and N^* be the set of trees whose nodes are elements of N. Let μ be the null tree, which is distinct from λ, the null label not in N. $\tilde{N} = N \cup \{\lambda\}$. A tree $T \in N^*$ with M nodes is said to be of size $|T|=M$, and is represented as the postorder numbering of its nodes. See [ZS89] for the advantages of this ordering.

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]. Thus, 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). The father of i is denoted as f(i). If $f^0(i) = i$, the node $f^k(i)$ can be recursively defined as $f^k(i) = f(f^{k-1}(i))$. The set of ancestors of i is : $Anc(i) = \{f^k(i) \mid 0 \le k \le Depth(i)\}$.

II.2 Elementary Edit Operations and Sub-Trees

An edit operation on a tree is either an insertion, a deletion 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 label or λ, the null label. $x = \lambda$ 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 -- it is not needed. The formal definitions of these follow.

Insertion of node x into tree T : Node x will be inserted as a son of some node u of T. If u has sons $u_1,u_2,..,u_k$, then for some $0 \leq i \leq j \leq k$, node u in the resulting tree will have sons $u_1,...,u_i$, x, $u_j,...,u_k$, and node x will have no sons if $j = i+1$, or else have sons $u_{i+1},...,u_{j-1}$.

Deletion of node y from a tree T : If node y has sons $y_1,y_2,...,y_k$ and node u, the father of y, has sons $u_1,u_2,...,u_j$ with $u_i=y$, then node u in the resulting tree obtained by the deletion will have sons $u_1,u_2,...,u_{i-1},y_1,y_2,...,y_k,u_{i+1},...,u_j$.

Substitution of node x by node y in T : Node y in the resulting tree will have the same father and sons as node x in the original tree.

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. We assume that it is reflexive (although this is not mandatory), and that it obeys 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 the inter-node edit distance $d(.,.)$ to the sequence S as :

$$W(S) = \sum_{i=1}^{|S|} d(s_i) .$$

With the introduction of W(S), the distance between T_1 and T_2 is :

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

II.3 Mappings between Trees

A Mapping is a description of how a sequence of edit operations transforms T_1 into T_2. 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 $T_2[j_2]$

(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 $T_2[j_2]$.

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. Then we can define the cost of M as follows :

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

Since mappings can be composed to yield new mappings [Ta79, ZS89], 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) $\leq$ W(S). Also, for any mapping M, there is a sequence of operations such that W(S) = cost(M).

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

III. EDIT CONSTRAINTS

Consider the problem of editing T_1 to T_2, where $|T_1| = N$ and $|T_2| = M$. Editing a postorder-forest of T_1 into a postorder-forest of T_2 using exactly i insertions, e deletions, and s substitutions, corresponds to editing $T_1[1..e+s]$ into $T_2[1..i+s]$. Bounds on the magnitudes of variables i, e, s, are constrained by the sizes of trees. If r=e+s, q=i+s, and R=Min{N,M}, these variables will obey the following constraints:

(a) $\max\{0,M-N\} \leq i \leq q \leq M$, (b) $0 \leq e \leq r \leq N$, and (c) $0 \leq s \leq R$.

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

We define :

(a) $H_i = \{j \mid \max\{0,M-N\} \leq j \leq M\}$, (b) $H_e = \{j \mid 0 \leq j \leq N\}$, and

(c) $H_S = \{j \mid 0 \leq j \leq \text{Min}\{M,N\}\}$.

H_i, H_e, and H_S are called the set of *permissible values* of i, e, and s. Theorem I specifies the feasible triples for editing $T_1[1..r]$ to $T_2[1..q]$.

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

♦♦♦

Theorem II.

Every edit constraint specified for the process of editing T_1 to T_2 is a unique subset of H_S.

♦♦♦

We refer to the 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. We now consider the computation of $D_\tau(T_1,T_2)$.

IV. CONSTRAINED TREE EDITING

Since edit constraints can be written as unique subsets of H_S, we 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). The distance between T_1[1..i] and T_2[1..j] subject to this constraint is given by Const_F_Wt(i,j,s) since the starting index of both trees is unity. The distance between the subtree rooted at i and the subtree rooted at j subject to the same constraint is given by Const_T_Wt(i,j,s). The difference between Const_F_Wt and Const_T_Wt is subtle. Indeed, Const_T_Wt(i,j,s) = Const_F_Wt(T_1[δ(i)..i],T_2[δ(j)..j],s). These weights obey the following properties proved in [OL94].

Lemma II

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j). Then

(i) Const_F_Wt(μ,μ,0) = 0.

(ii) Const_F_Wt(T_1[δ(i_1)..i],μ,0) = Const_F_Wt(T_1[δ(i_1)..i-1],μ,0) + d(T_1[i],λ).

(iii) Const_F_Wt(μ,T_2[δ(j_1)..j],0) = Const_F_Wt(μ,T_2[δ(j_1)..j-1],0) + d(λ,T_2[j]).

(iv) Const_F_Wt(T_1[δ(i_1)..i],T_2[δ(j_1)..j],0)

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

(v) Const_F_Wt(T_1[δ(i_1)..i],μ,s) = ∞ if s > 0.

(vi) Const_F_Wt(μ,T_2[δ(j_1)..j],s) = ∞ if s > 0.

(vii) Const_F_Wt(μ,μ,s) = ∞ if s > 0.

♦♦♦

Lemma II essentially states the properties of the constrained distance when either s is zero or when either of the trees is null. These are thus "basis" cases which can be used in any recursive computation of the distance. For the non-basis cases we have to consider the scenarios when the trees are non-empty and when the constraining parameter, s, is strictly positive. The recursive property of Const_F_Wt is given by Theorem III.

Theorem III.

Let $i_1 \in$ Anc(i) and $j_1 \in$ Anc(j). Then

Const_F_Wt(T_1[δ(i_1)..i],T_2[δ(j_1)..j],s)

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

♦♦♦

Theorem III naturally leads to a recursive algorithm, except that its time and space complexities will be prohibitively large. However, under certain conditions, if the removal of a sub-forest leaves us with an entire tree, the computation is simplified. Thus, if $\delta(i)=\delta(i_1)$ and $\delta(j)=\delta(j_1)$ (i.e., i and i_1, j and j_1 span the same subtree), 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. If not, 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 $Const\_T\_Wt(i,j,s_2)$ as below.

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

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

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

Otherwise,

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

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

♦♦♦

Theorem IV suggests that we can use a dynamic programming flavored algorithm to solve the constrained tree editing problem. The theorem also 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 in the process of computing the Const_F_Wt between the trees rooted at i_1 and j_1. These distances are obtained as a by-product because whenever an Const_F_Wt is computed, if the forests are trees, it is retained as a Const_T_Wt. 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 Essential_Nodes, and these are nodes for which the computation would involve the second case of Theorem IV as opposed to the first. We define the set Essential_Nodes of tree T as :

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

Intuitively, this set will be the roots of all subtrees of tree T that need separate computations. Thus, the Const_T_Wt can be computed for the entire tree if Const_T_Wt of the Essential_Nodes are computed, and using these stored values, the rest of the Const_T_Wts can be computed. Using Theorem IV we can now develop a bottom-up approach for computing the Const_T_Wt between all pairs of subtrees.

Note that the function $\delta()$ and the set Essential_Nodes() can be computed in linear time.

We shall now compute Const_T_Wt(i, j, s) and store it in a *permanent* three-dimensional array Const_T_Wt. In the interest of brevity the algorithms used in this paper are omitted here, but can be found in [OZL98]. The correctness of Algorithm T_Weights is proven in detail in [OL94].

As a result of invoking Algorithm T_Weights (which repeatedly invokes Algorithm Compute_Const_T_Wt for all pertinent values of i and j) we will have computed the constrained inter-tree edit distance between T_1 and T_2 subject to the constraint that the number of substitutions performed is s, for all feasible substitutions, s. The space required by the above algorithm is obviously $O(|T_1| * |T_2| * \text{Min}\{|T_1|, |T_2|\})$. If Span(T) is the Min{Depth(T), Leaves(T)}, the algorithm's time complexity is $O(|T_1| * |T_2| * (\text{Min}\{|T_1|, |T_2|\})^2 * \text{Span}(T_1) * \text{Span}(T_2))$.

V. THE NOISY SUBSEQUENCE TREE RECOGNITION PROBLEM

V.1 Principles Used in Solving the Noisy Subsequence-Tree Recognition Problem

Using the foundational concepts of constrained edit distances explained in the previous sections, we are now in a position to present our solution to the Noisy Subsequence Tree (NSuT) Recognition Problem. We assume that a "Transmitter" intends to transmit a tree X^* which is an element of a finite dictionary of trees, H. However, it opts to transmit one of its subsequence trees, U by randomly delete nodes from X^*. The transmission of U is across a noisy channel which is capable of introducing substitution, deletion and insertion errors at the nodes. We also assume that the tree itself is transmitted as a two dimensional entity (and not merely string representations). The receiver receives Y, a noisy version of U. We now show how we recognize X^* from Y. To render the problem tractable, we assume that *some of* the properties of the channel can be observed. We assume that L, the expected number of substitutions introduced in transmission, can be estimated.

Since U can be an arbitrary subsequence tree of X^*, it is obviously meaningless to compare Y with every $X \in H$ using any known unconstrained tree editing algorithm. Clearly, before we compare Y to the individual tree in H, we have to use the additional information obtainable from the noisy channel. Also, since the specific number of substitutions (or insertions/deletions) introduced in any *specific* transmission is unknown, it is reasonable to compare any $X \in H$ and Y subject to the constraint that the number of substitutions that actually took place is its best estimate which in this case is the expected value, L. One could therefore use the set {L} as the constraint set to effectively compare Y with any $X \in H$. Since the latter set can be quite restrictive, we have opted to perform the comparison using a constraint set which is a superset of {L} marginally larger than {L}. We utilized the set {L-1, L, L+1}. Since the size of the set is still a constant, there is no significant increase in the computation times. This is the rationale for our recognition algorithm (Algorithm *RecognizeSubsequenceTrees*) given in the [OZL98].

Note that 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 given in [OL94] which essentially minimizes Const_T_Wt over all the values of 's' found in the constraint set.

V.2 Experimental Results

The technique developed in the previous sections was rigorously tested to verify its capability in the recognition of NSuTs. The experiments conducted were for two different data sets which were artificially generated. We have used "relatively long" character sequences using benchmark results involving keyboard character errors. These results are sufficient to demonstrate the power of the strategy to recognize noisy subsequence trees. The results we have obtained for simulated trees are remarkable. As mentioned earlier, to our knowledge, these are the first reported results which demonstrate that a tree can indeed be recognized by processing the information resident in one of its noisy subsequence trees. The details of the experimental set-ups and the results obtained follow.

V.2.1 Tree Representation

In the implementation of the algorithm we have opted to represent the tree structures of the patterns studied as parenthesized lists in a post-order fashion. Thus, a tree with root 'a' and children B, C and D is represented as a parenthesized list L = (B C D 'a') where B, C and D can themselves be trees in which cases the embedded lists of B, C and D are inserted in L. A specific example of a tree (taken from our dictionary) and its parenthesized list representation is given in Figure I below.

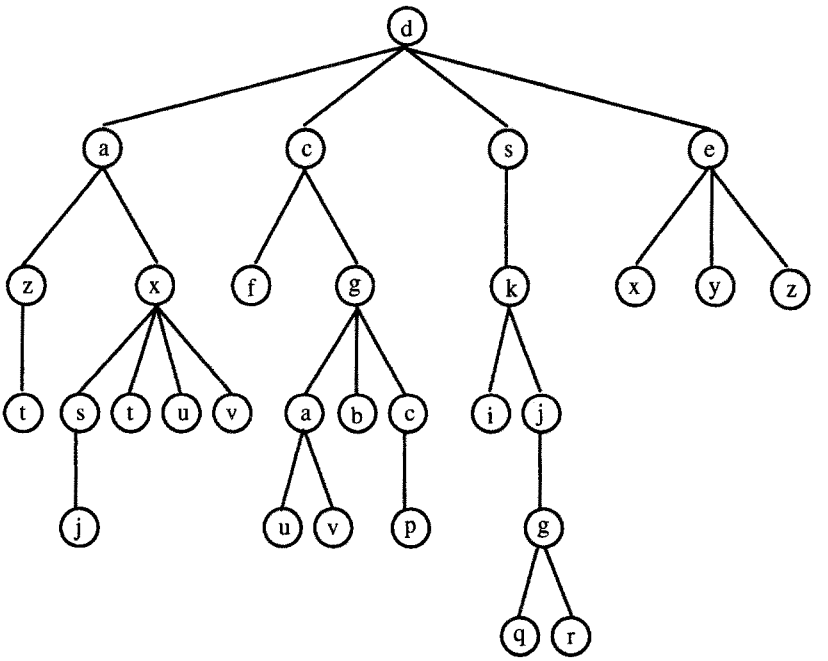

Figure I: A tree from the finite dictionary H. Its associated list representation is as follows:
((((t)z)(((j)s)(t)(u)(v)x)a)(((f)(((u)(v)a)(b)((p)c)g)c)(((i)(((q)(r)g)j)k)s)((x)(y)(z)e)d)

V.3 Experiment Setup for Data Set A

In our first experimental set-up the dictionary, H, consisted of 25 *manually* constructed trees which varied in sizes from 25 to 35 nodes. An example of a tree in H is given in Figure I above. To generate a NSuT for the testing process, a tree X^* (unknown to the classification algorithm) was chosen. Nodes from X^* were first randomly deleted producing a subsequence tree, U. In our experimental set-up the probability of deleting a node was set to be 60%. Thus although the average size of each tree in the dictionary was 29.88, the average size of the resulting subsequence trees was only 11.95.

The garbling effect of the noise was then simulated as follows. A subsequence tree U, was subjected to additional substitution, insertion and deletion errors which deforms the tree. This was effectively achieved by passing the string representation through a channel causing substitution, insertion and deletion errors analogous to the one used to generate the noisy subsequences in [Oo87] and which has recently been formalized by Oommen and Kashyap [OK96]. However, as opposed to merely mutating the string representations as in [OK96] the reader should observe that we are manipulating the underlying *list* representation of the *tree*. This involves ensuring the maintenance of the parent/sibling consistency properties of a tree - which is not trivial. In our specific scenario, the alphabet involved was the English alphabet, and the conditional probability of inserting any character $a \in A$ given that an insertion occurred was assigned the value 1/26 and the probability of a deletion was 1/20. The table of probabilities for substitution (the confusion matrix) was based on the proximity of the character keys on a standard QWERTY keyboard [Oo86, Oo87, OK96]. In our experiments ten NSuTs were generated for each tree in H yielding a test set of 250 NSuTs. The average number of tree deforming operations done per tree was 3.84. A typical example of the NSuTs generated, its associated subsequence tree and the tree in the dictionary which it originated from is given below in Figure II. Table I gives the average number of errors involved in the mutation of a subsequence tree, U. Indeed, after considering the noise effect of deleting nodes from X^* to yield U, the overall average number of errors involved is 21.76.

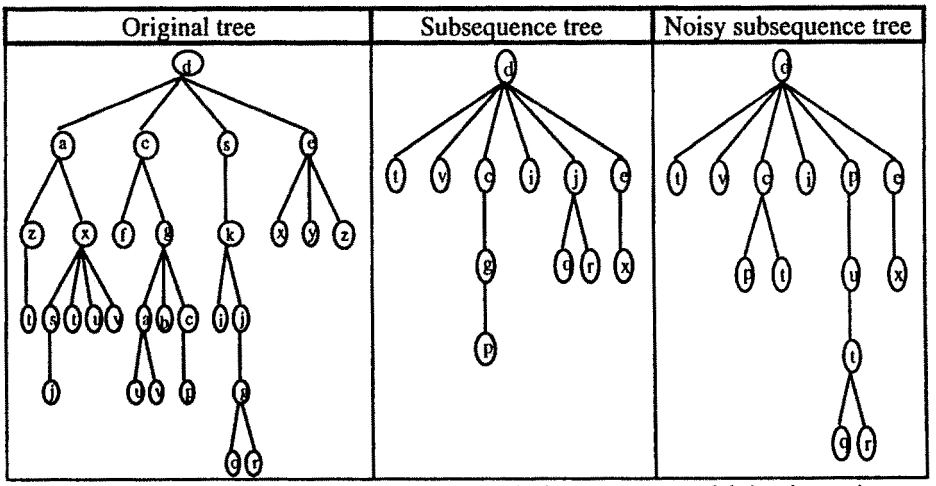

Figure II : Example of the original trees, the associated subsequence trees and their noisy versions.

Type of errors	Number of Errors	Average error per tree
Insertion	493	1.972
Deletion	313	1.252
Substitution	153	0.612
Total average error		3.836

Table I : The statistics associated with Data Set (A) of NSuTs used in the experiments.

The results that were obtained were remarkable. 232 out of 250 NsuTs were correctly recognized, implying an accuracy of 92.80%. This is quite overwhelming considering the fact that we are dealing with 2-dimensional objects with an unusually high (about 73%) error rate at the node and structural level.

V.4 Experiment Setup for Data Set B

In the second experimental set-up, the dictionary, H, consisted of 100 trees which were generated randomly. The tree structure for an element in H was obtained by randomly generating a parenthesized expression using the following stochastic context-free grammar G, where,

G = <N, A, G, P>, where, N = {T, S, $} is the set of non-terminals, A is the set of terminals - the English alphabet, G is the stochastic grammar with associated probabilities, P, given below :

$T \rightarrow (S\$)$	with probability 1,
$S \rightarrow (SS)$	with probability p_1,
$S \rightarrow (S\$)$	with probability $1-p_1$,
$S \rightarrow (\$)$	with probability p_2,
$S \rightarrow \lambda$	with probability $1-p_2$, where λ is the null symbol,
$\$ \rightarrow a$	with probability 1, where $a \in A$ is a letter of the alphabet.

Note that a smaller value of p_1 yields a more tree-like representation, and a larger value of p_1 a more string-like representation. In our experiments the values of p_1 and p_2 were set to be 0.3 and 0.6 respectively. The sizes of the trees varied from 27 to 35 nodes.

Once the tree structure was generated, the actual substitution of '$' with the terminal symbols was achieved by using the benchmark textual data set used in recognizing noisy subsequences [Oo87]. Each '$' symbol in the parenthesized list was replaced by the next character in the string. Thus, for example, one parenthesized expression for a tree obtained using the above tree generation process was :

((((((((((($)$)$)(($)$)$)$)$)$)(((((($)($)$)$)$)(((($)($)(($)$)$)$)$)$)$)$)$)$)

The '$"s in the string are now replaced by terminal symbols to yield the following list:

((((((((((((i)n)t)h)((i)s)s)e)c)t)((((((i)o)((n)w)e)c)a)((((l)c)((u)l)(((a)t)e)t)h)e)a)p)o)s)

The actual underlying tree for this string can be deduced from Section V.2.1.

The process as described in Section V.3 was used to generate the NsuTs. The average size of the resulting subsequence trees was only 13.42 instead of 31.45 for the original trees in the dictionary. In our experiments five NSuTs were generated for

each tree in H yielding a set of 500 NSuTs. The average number of tree deforming operations done per tree was 3.77. Table II gives the average number of errors involved in the mutation of a subsequence tree, U. The overall average number of errors was 21.8.

Type of errors	Number of errors	Average error per tree
Insertion	978	1.956
Deletion	601	1.202
Substitution	306	0.612
Total average error		3.770

Table II : The statistics associated with Data Set (A) of NSuTs used in the experiments.

Out of the 500 noisy subsequence trees tested, 432 were correctly recognized, which implies an accuracy of 86.4%. The power of the scheme is obvious considering the fact we are dealing with 2-dimensional objects with an unusually high (about 69.32%) error rate and the fact that the corresponding uni-dimensional problem (which only garbled the strings and did not mutate the *structure*) gave an accuracy of 95.4% [Oo87]. Additional experimental results are found in the unabridged paper, and are omitted here in the interest of brevity.

VIII. CONCLUSIONS

In this paper, we have considered the problem of recognizing trees by processing their noisy subsequence trees. The solution we propose (which, to our knowledge, is the first reported solution) was based on the theoretical work done by Oommen and Lee [OL94] involving constrained tree editing. Given a noisy subsequence tree Y of an unknown tree X^* in H, the technique which we propose estimates X^* by computing the constrained edit distance between every X in H and Y, based on the actual garbling properties of the error generating mechanism.

We have rigorously tested our algorithm experimentally for data sets which are manually and randomly generated. The experimental results obtained are remarkable considering the level of noise introduced and the fact that the garbling mechanism mutates the string *and the underlying structure* which stores the trees. We are currently working on applying these techniques to recognizing biological molecules from their *fragments*.

Acknowledgements : This work was partially supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[Lu79] S. Y. Lu, "A tree-to-tree distance and its application to cluster analysis", *IEEE Trans. Pattern Anal. and Mach. Intell.*, Vol. PAMI 1, No. 2: pp. 219-224 (1979).

[Oo87] B. J. Oommen, "Recognition of noisy subsequences using constrained edit distances", *IEEE Trans. Pattern Anal. and Mach. Intell.*, Vol. PAMI 9, No. 5 : pp. 676-685 (1987).

[OK96] B. J. Oommen and R. L. Kashyap, "A formal theory for optimal and information theoretic syntactic pattern recognition". (To appear in *Pattern Recognition*).

[OL94] B. J. Oommen, and W. Lee, "Constrained Tree Editing", *Information Sciences*, Vol. 77 No. 3,4: pp. 253-273 (1994).

[OL97] B. J. Oommen, and W. Lee, "Constrained Tree Editing", *Information Sciences*, Vol. 77 No. 3,4: pp. 253-273 (1994).

[OZL98] B. J. Oommen and R. K. S Loke, "On the Recognition of Noisy Subsequence Trees". Unabridged version of this paper.

[SK83] D. Sankoff and J. B. Kruskal, *Time wraps, string edits, and macromolecules : Theory and practice of sequence comparison*, Addison-Wesley, (1983).

[Se77] S. M. Selkow, "The tree-to-tree editing problem", *Inform. Proc. Let.*, Vol. 6, pp. 184-186 (1977).

[SZ90] B. Shapiro and K. Zhang, "Comparing multiple RNA secondary structures using tree comparisons", *Comput. Appl. Biosci.* vol. 6, no. 4, 309-318 (1990).

[Ta79] K. C. Tai, "The tree-to-tree correction problem", *J. Assoc. Comput. Mach.*, Vol. 26 : pp. 422-433 (1979).

[ZJ94] K. Zhang and T. Jiang, "Some MAX SNP-hard results concerning unordered labeled trees", *Information Processing Letters,* 49, 249-254 (1994).

[ZS89] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems", *SIAM J. Comput.* Vol. 18, No. 6 : pp. 1245-1262 (1989).

[ZSW92] K. Zhang, D. Shasha and J. T. L. Wang, "Fast serial and parallel approximate tree matching with VLDC's", *Proc. of the 1992 Symposium on Combinatorial Pattern Matching,* CPM92, 148-161 (1992).