RECOGNITION OF NOISY SUBSEQUENCES USING CONSTRAINED EDIT DISTANCES

B. John Oommen

SCS-TR-95
June 1986

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

This research was partially supported by NSERC.

RECOGNITION OF NOISY SUBSEQUENCES USING CONSTRAINED EDIT DISTANCES+

B. John Oommen*

ABSTRACT

Let X^* be any unknown word from a finite dictionary H. Let U be any arbitrary **subsequence** of X^*. We consider the problem of estimating X^* by processing Y which is a **noisy** version of U. We do this by defining the constrained edit distance between $X \in H$ and Y subject to any arbitrary edit constraint involving the number and type of edit operations to be performed. An algorithm to compute this constrained edit distance has been presented. Although in general the algorithm has a cubic time complexity, within the framework of our solution the algorithm possesses a quadratic time complexity. Recognition using the constrained edit distance as a criterion demonstrates a remarkable accuracy. Experimental results which involve strings of lengths between 40 and 80 and which contain an average of 26.547 errors per string demonstrates that the scheme has about 99.5% accuracy.

List of keywords : String Correction, Subsequence Correction, Substring Correction, Constrained Editing, Levenshtein Metric.

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

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

I. INTRODUCTION

A common problem in text editing is that of finding the occurrence of a given substring in a file. This is usually done to locate one's bearings in the file or to replace the occurrence of one substring by another. This problem is termed as the exact substring matching problem.

With no loss of generality, the file can be considered as a sequence of words from a finite dictionary. If the occurrence of a certain substring, U, is sought for, the exact substring matching problem aims to obtain the set of all words in the dictionary which contains U as a **contiguous substring**. Many algorithms have been proposed to solve this problem, the best of which have sub-linear time complexities.

One mishap that often occurs is that the substring sought for, U, is noisily represented, either due to mistyping or ignorance of spelling. Let the noisy version of U be Y. If Y is a contiguous substring of some words in the dictionary, the output of an exact substring matching algorithm will be the set of those particular words. On the other hand, if Y is not a contiguous substring of any word in the dictionary, any substring matching algorithm will give the user no information about the occurrence of the substring sought for. However, in such a case the algorithm due to Kashyap and Oommen [15] will yield the set of strings in H which contains a contiguous substring that is closest to Y. The metric used in [15] to measure the distance between two strings is the Levenshtein metric that has been extensively studied in the literature [2,10,11,14,16,18, 19,21, 23,28,32]. The power of the technique described in [15] to correct noisy substrings was demonstrated using a subset of the 1023 most common English words.

In this paper we consider a far more general problem. Let us assume that a sender intends to transmit a string $X^* \in H$. However, rather than send the entire string X^* he chooses to (randomly or otherwise) delete characters from it, and merely transmit a **subsequence,** U, of X^*. Note that any substring consisting of the consecutive characters of U, need not be a contiguous substring of X^*. The string U is transmitted through a noisy channel and is further subjected to substitution, deletion and insertion errors. The receiver receives Y, which is a garbled version of U. We aim to present an

accurate recognition technique to recognize X^* by merely processing Y.

To clarify situations, we shall consider an example from one of the experiments we have performed. Let us suppose X^* is the following string:

X^* = sincetheadventofthedigitalcomputertherehasbeenaconstante

Notice that the length of X^* is 55. Let us suppose the sender deletes various portions of X^* to form the string U, where,

U = dventofttalcomrehenaco.

Observe that in this case U is a subsequence of X^*. Also note that 34 of the 55 characters have been deleted and the resultant string U is transmitted through a noisy channel. The received string Y contains additional substitution, deletion and insertion errors and is received as

Y = senoftalxehbaco

In this case, between X^* and Y there is an overall number of 44 substitution, deletion and insertion errors. Our aim is to recognize X^* by processing Y. It is not inappropriate to add that the technique which we have proposed in this paper indeed recognizes it from a dictionary of 100 words of varying lengths.

Besides having applications in file editing, the noisy substring matching problem has also potential applications in the area of information retrieval. Consider a data base which consists of many records each identified by a distinct keyword. Let us suppose that the system managing the data base permits the user to access a record by merely referring to any arbitrary subsequence of the keyword. In such a case the program that solves the noisy subsequence matching problem can be used to process a noisy (incorrectly spelled) version of any subsequence of the keyword, and locate a record characterized by the entire keyword.

Before we proceed, to view our work in the right perspective, we shall briefly review the various string correcting algorithms available in the literature. The first problem extensively studied was the one in which Y was an inexact version of an **entire** string in H. Pioneering researchers tried to solve this problem by reducing it to an exact string matching problem. Both the words in H and Y were abbreviated based on some simple rules [5]. Y was then defined as "similar" to $X \in H$ if the abbreviations

for X and Y **exactly** matched. However, if there was no exact match for the abbreviation of Y, the words in H were reckoned as totally dissimilar to Y. Though this technique has been tested for an airlines reservation system [see 11], its application is limited, especially in the cases when H consists of words of varying lengths. This led to a more scientific approach involving the quantification of the dissimilarity (or similarity) between two strings.

Damareau [6] was probably the first researcher to observe that most of the errors that were found in strings were either a single substitution, deletion, insertion or reversal error. Thus the question of computing the dissimilarity between strings was reduced to comparing them using these edit transformations. Since the reversal edit operation is easily modelled as a sequence of a single insertion and deletion, and since this simplification can drastically change the complexity of the comparison problem, most of the research in this area has been directed towards comparing strings using substitution, deletion and insertion edit operations.

The first major breakthrough in comparing strings using these three elementary edit transformations was the concept of the Levenshtein metric introduced in coding theory [18], and its computation. The Levenshtein distance between two strings is defined as the minimum number of edit operations required to transform one string into another. Okuda et al [23] extended this concept by weighting the edit operations, and many other researchers among whom are Wagner and Fischer [32] generalized it by using edit distances which are symbol dependent. The latter distance is termed as the Generalized Levenshtein Distance (GLD). One of the advantages of the Generalized Levenshtein Distance (GLD) is that it can be made a metric if the individual edit distances obey some constraints [23]. Wagner and Fischer [32] also proposed an efficient algorithm for computing this distance by utilizing the concepts of dynamic programming. This algorithm has been proved to be the optimal algorithm for the infinite alphabet case. Various amazingly similar versions of this algorithm are also available in the literature, a review of which can be found in [28]. Masek and Paterson [21] improved the latter algorithm for the finite alphabet case.

Closely related to these algorithms are the ones proposed to compute the Longest Common Subsequence (LCS) of two strings due to Hirschberg [12,27] Hunt

and Szymanski [13] and Needleman and Wunsch [27]. Bounds on the complexity of the LCS problem have been given by Aho et al [1]. String correction using the GLD as a criterion has been used for individual strings [19,23] dictionaries treated as generalized tries [14,16] and for grammars [10]. Besides these deterministic techniques, various probabilistic methods have been studied in the literature [9,10,17,19,26,29-31]. A comparative study of these techniques highlighting their individual merits and demerits is presented in detail in [17].

The only paper known to us dealing with the correction of noisy substrings and which reports experimental evidence is the one due to Kashyap and Oommen [15]. Clearly, the problem that we study here is much more complex inasmuch as the number of subsequences of a string is **much** larger than the number of substrings. In this regard, we believe that the results which are presented here indicate the value of our contribution to the field of pattern recognition.

I.1 Proposed Solution

The solution that we propose involves computing a quantity which we shall term as "the constrained edit distance" between Y, the received string, and every $X \in H$.

All of the above mentioned algorithms consider the editing of one string, say X, to transform it to Y with the edit process being absolutely unconstrained. Sankoff [27] was the one who pioneered the study of constrained string editing. The only algorithm presented in the literature which investigated this problem is due to him. His algorithm is a LCS algorithm which involves a specialized constraint that has its application in the comparison of Amino Acid sequences.

The general constrained editing problem involves editing X to Y subject to any arbitrary edit constraint. This edit constraint can be arbitrarily complex so long as it is specified in terms of the number and type of the edit operations to be included in the optimal edit transformation. For the sake of clarity we give below some examples of constrained editing.

EXAMPLE I.

Here are some typical constrained editing problems.

(a) What is the optimal way of editing X to Y using no more than k insertions?

(b) How can we optimally transform X to Y using exactly k substitutions?

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

The problem of constrained string editing was studied by us in [24] and two algorithms presented to compute the constrained edit distance and the best edit sequence for a given constraint respectively [24]. In [24] we presented a consistent method of specifying any arbitrary edit constraint, τ. We then discussed the computation of $D_\tau(X,Y)$, the edit distance between X and Y subject to this constraint. The latter quantity is computed by first evaluating the elements of a three-dimensional array W(.,.,.) using dynamic programming techniques, and then combining certain elements of W(.,.,.) to yield $D_\tau(X,Y)$. Using the array W(.,.,.) the optimal sequence of edit operations required to edit X to Y subject to the constraint τ, can also be obtained by backtracking. It was shown that the algorithm presented in [24] required $0(|X|.|Y|.\min(|X|,|Y|))$ time and space.

Apart from defining and solving the problem of constrained string editing in its most general framework, a major contribution of [24] is that of using the concepts of dynamic programming to compute a quantity that does not inherently possess any known recursively computable properties. The quantity which we are referring to is indeed the constrained edit distance $D_\tau(X,Y)$. However, it was shown that unlike the latter quantity, the array W(.,.,.) is closely related to it and can be computed using dynamic programming in a fairly straightforward way. This technique is quite analogous to the computations in a control system in which the output is evaluated in terms of state variables which are computable in real time.

In this paper we shall show that the general constrained editing problem can be solved in cubic time and in quadratic space. We shall then consider the case when the

constraint **set** τ is a constant singleton set {T} where T is much smaller than the lengths of the strings. In this case, we shall show that the distance $D_\tau(X,Y)$ can be computed in quadratic time and space. We conclude the paper with experimental results that demonstrate the power of the strategy proposed in recognizing noisy subsequences.

In this paper, for the sake of brevity, results proven elsewhere shall merely be alluded to, unless their exclusion degrades the clarity of the paper.

I.2. Notation

Let A be any finite alphabet and A^* be the set of strings over A. θ is the null symbol, $\theta \notin A$. Let $\hat{A} = A \cup \{\theta\}$. $\hat{A}$ is referred to as the Appended Alphabet. A string $\underset{\sim}{X} \in \hat{A}^*$ of the form $X = x_1x_2...x_N$, where every $x_i \in \hat{A}$, is said to be of length $|X| = N$. Its prefix of length i will be written as X_i, $i < N$. Upper case symbols represent strings and lower case symbols represent elements of the alphabet under consideration.

Let Z' be any element in $\hat{A}^*$, the set of strings over $\hat{A}$. The Compression Operator, C, is a mapping from $\hat{A}^*$ to A^*. C(Z') is Z' with all occurrences of the symbol θ removed from Z'. Note that C preserves the order of the non-θ symbols in Z'. For example, if Z' = fθoθr, C(Z') = for.

1.3 The Set of Elementary Edit Distances d(.,.) and the Set Γ (X,Y)

$d(.,.) : \hat{A} \times \hat{A} \rightarrow R^+$, is a function whose arguments are a pair of symbols belonging to $\hat{A}$, the appended alphabet, and whose range is the set of nonnegative real numbers; $d(\theta, \theta)$ is undefined and is not needed. The elementary distance d(a,b) can be interpreted as the distance associated with transforming 'a' to 'b', for a, b $\in \hat{A}$. Thus,

(a) $d(x_i, y_j)$ is the distance associated with substituting y_j for x_i , x_i , $y_j \in A$.

(b) $d(x_i, \theta)$ is the distance associated with deleting $x_i \in A$.

(c) $d(\theta, y_j)$ is the distance associated with inserting $y_j \in A$.

Note that $|\Gamma(x,y)|$ depends only on $|X|$ and $|Y|$, and not on the actual strings X and Y themselves. Further, observe that the transformation of a symbol $a \in A$ to itself is also **considered as an operation** in the arbitrary pair $(X', Y') \in \Gamma(X,Y)$.

EXAMPLE II.

Let X = f and Y = go. Then,

$\Gamma(X, Y) = \{ (f\theta, go), (\theta f, go), (f\theta\theta, \theta go), (\theta f\theta, g\theta o), (\theta\theta f, go\theta) \}$

In particular, the pair $(\theta f, go)$ represents the operations of inserting the 'g' and replacing the 'f' by an 'o'.

Since the Generalized Levenshtein Distance (GLD) [2,10,11, 14,16,18, 19,21,23,28,32] between X and Y is the minimum of the sum of the edit distances associated with the edit operations required to transform X to Y, this distance, written as D(X,Y), has the expression,

$$D(X,Y) = \underset{(X',Y') \in \Gamma(X,Y)}{\text{Min}} \left[\sum_{1 \le i \le |X'|} \{ d(x'_i , y'_i) \} \right] \qquad (2)$$

Since the transformation of a symbol $a \in A$ to itself is considered as a substitution operation, an application in which this is a nonoperation must specify $d(a,a)=0$ for all $a \in A$.

II. EDIT CONSTRAINTS

II. 1 Permissible and Feasible Edit Opertations

Consider the problem of editing X to Y, where $|X| = N$ and $|Y| = M$. Suppose we edit a prefix of X into a prefix of Y, using exactly i insertions, e deletions (or erasures) and s substitutions. Since the **number** of edit operations are specified, this corresponds to editing $X_{e+s} = x_1 \ldots x_{e+s}$, the prefix of X of length e+s, into

$Y_{i+s}=y_1 \cdots y_{i+s}$, the prefix of Y of length i+s.

To obtain bounds on the magnitudes of the variables i, e and s, we observe that they are constrained by the lengths of the strings X and Y. Thus, if r=e+s, q=i+s and R=Min [M, N], these variables will have to obey the following obvious constraints.

$$\text{Max}[0,M-N] \leq i \leq q \leq M$$

$$0 \leq e \leq r \leq N$$

$$0 \leq s \leq \text{Min}[M,N]$$

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

$$H_i = \{ j \mid \text{Max}[0,M-N] \leq j \leq M \},$$

$$H_e = \{ j \mid 0 \leq j \leq N \}, \text{ and}$$

$$H_S = \{ j \mid 0 \leq j \leq \text{Min}\,[M, N] \} \qquad (3)$$

H_i, H_e and H_S are called the set of **permissible** values of i, e and s. Observe that a triple (i,e,s) is feasible if apart from $i \in H_i$, $e \in H_e$, and $s \in H_S$, the following is satisfied:

$$i + s \leq M, \quad \text{and} \quad e + s \leq N. \qquad (4)$$

The following theorem specifies the permitted forms of the feasible triples encountered on editing X_r, the prefix of X of length r, to Y_q, the prefix of Y of length q.

Theorem I.

To edit X_r, the prefix of X of length r, to Y_q, the prefix of Y of length q, the set of feasible triples is given by:

$$\{ (i, r-q+i, q-i) \mid \text{Max}\,[0, q-r] \leq i \leq q \}.$$

PROOF. Consider the constraints imposed on feasible values of i, e and s. Since we are interested in the editing X_r to Y_q we have to consider only those triples (i, e, s) in

$j \in Q_e$ requires the editing to be performed using exactly j deletions. From Theorem I, since $|X| = N$, this requires that the number of substitutions is N-j. Further, since $|Y|=M$, the number of insertions is forced to be M-N+ j.

Similarly, if $j \in Q_s$, the edit transformations must contain exaclty j substitutions. Since $|Y| = M$ and $|X| = N$, Theorem I requires that M-j insertions and N-j deletions are performed.

Let $Q_e^* = \{ M\text{-}N + j \mid j \in Q_e\}$, and, $Q_s^* = \{M\text{-}j \mid j \in Q_s\}$

Clearly, for any arbitrary constraint the number of insertions that are permitted is given by a set of integers which is obtained by intersecting Q_i, Q_e^* and Q_s^*, which is obviously a subset of H_i.

***

REMARKS:

1. The set referred to above which describes the constraint and which is the subset of H_i shall, in future, be written as τ.

2. The edit constraint can just as easily be written as a subset of H_s or H_e. We have chosen to describe τ as a subset of H_i for reasons which will be clear later.

3. Observe that the converse of Theorem II is not true. Verbal descriptions of many edit constraints could lead to the same set τ.

EXAMPLE IV.

Let X = for and Y = fa. Suppose we want to transform X to Y by performing at least 1 insertions, at most 1 substitutions and exactly 2 deletions. Then,

$Q_i = \{1,2\}$, $Q_e = \{2\}$ and $Q_s = \{0, 1\}$.

Hence, $Q_e^* = \{ 1 \}$, and $Q_s^* = \{1,2\}$. Thus,

$$\tau = Q_i \cap Q_e^* \cap Q_s^* = \{1\}.$$

Hence the optimal transformations must contain **exactly** one insertion. Some

candidate edit transformations are given by the following subset of $\Gamma(X, Y)$:

$$\{ (\theta for, f\theta\theta a),\ (f\theta or, fa\theta\theta),\ (fo\theta r, f\theta a\theta),\ (for\theta, f\theta\theta a) \}$$

Note that every pair in the above subset corresponds to at least one insertion, at most 1 substitution and exaclty 2 deletions.

***

We shall refer to the edit distance subject to the constraint τ as $D_\tau(X, Y)$. By definition, $D_\tau(X,Y) = \infty$ if $\tau = \varnothing$. This is merely a simple way of expressing that it is impossibe to edit X to Y subject to the constraint τ. We shall now consider the computation of $D_\tau(X,Y)$.

III. W: THE ARRAY OF CONSTRAINED EDIT DISTANCES

Let W (i,e,s) be the constrained edit distance associated with editing X_{e+s} to Y_{i+s} subject to the constraint that exaclty i insertions, e deletions and s substitutions are performed in the process of editing. As before, let r=e+s and q=i+s. Let $\Gamma_{i,e,s}(X,Y)$ be the subset of the pairs in $\Gamma(X_r,Y_q)$ in which every pair corresponds to i insertions, e deletions and s substitutions. Since we shall consistently be referring to the strings X and Y, we refer to this set as $\Gamma_{i,e,s}$. Thus, using the notation of (1), and assuming (i,e,s) is feasible for the problem, W(i,e,s) has the expression

$$W(i,e,s) = \text{Min}_{(X'_r,\, Y'_q) \in \Gamma_{i,e,s}} \left[\sum_{1 \le j \le |X'_r|} \{ d(x'_{rj}, y'_{qj}) \} \right] \qquad (5)$$

We shall now state the recursively computable properties of the array W(.,.,.).

THEOREM III.

Let W(i,e,s) be the quantity defined as in (5) for any two strings X and Y. Then,

$$W(i,e,s) = \text{Min}\left[\{W(i-1, e,s) + d(\theta, y_{i+s})\}, \{W(i, e-1, s) + d(x_{e+s}, \theta)\}, \{W(i,e,s-1) + d(x_{e+s}, y_{i+s})\} \right]$$

for all feasible triples (i,e,s).

PROOF : The theorem is quite involved and proved in [24]. ***

The computation of the distance $D_\tau(X,Y)$ from the array W (i,e,s) only involves combining the appropriate elements of the array using τ, the set of the number of insertions permitted. This is proved in the following theorem.

THEOREM IV.

The quantity $D_\tau(X,Y)$ is related to the elements of the array W(i,e,s) as below.

$$D_\tau(X,Y) = \text{Min}_{i \in \tau}\left[W(i, N-M+i, M-i) \right]$$

PROOF. The theorem follows directly from Theorem I by substituting r = N and q=M.

***

IV. THE COMPUTATION OF $D_\tau(X,Y)$

To compute $D_\tau(X, Y)$ in [24] we made use of the fact that though the latter index itself does not seem to have any recursive properties, the index W (.,.,.), which is closely related to it has the interesting properties stated in Theorem III. Using these properties an algorithm was proposed which computed the array W(.,.,.) for all feasible values of the variables i,e and s. The latter algorithm required cubic time and space respectively. Subsequently, using the array W (i,e,s) as the input, a second algorithm was presented in [24] which computed $D_\tau(X,Y)$ by comparing the contributions of the

pertinent elements in W(.,.,.) as specified by Theorem IV.

In this paper we shall proceed in a completely different way. Rather than compute and store the entire array W(.,.,.) we shall compute only those of its elements which are necessary to evaluate $D_\tau(X,Y)$. To do this we note the following facts:

Fact A: Let T be the largest integer in τ. Then, to compute $D_\tau(X,Y)$, it is sufficient to evaluate the components of W(.,.,.) for values of i in the range $0 \leq i \leq T$.

Fact B: For a particular value of i, in order to compute W(i,e,s) for all permissible values of e and s, it is sufficient to store only the values of W (i-1, e, s) for all the corresponding permissible values of e and s. This is indeed true from Theorem III, since the computation of any one quantity requires **at most** three previously computed quantities, namely W(i-1, e,s), W(i, e-1,s) and W(i, e, s-1).

Consider a three dimensional trellis, in which the coordinates are i, e and s. The strategy which we propose involves successively computing the array W in planes parallel to the plane i = 0. Four arrays are maintained, namely,

(a) Wie : the plane in which s = 0.

(b) Wis : the plane in which e = 0.

(c) Wes0 : the plane parallel to i=0, maintained for the previous value of i, and,

(d) Wes1 : the plane parallel to i=1 maintained for the current value of i.

The algorithm merely computes these arrays in a systematic manner. Initially the weights associated with the individual axes are evaluated. The i-e and i-s planes are then computed and stored in the arrays Wie and Wis respectively . The trellis is then traced plane by plane - always retaining only the current plane parallel to the plane i=0. Observe that prior to updating Wes0, its pertinent component required in the computation of $D_\tau(X,Y)$ should be used to update the latter quantity.

Finally, observe that by virtue of Fact A, the planes parallel to i=0 need be traced only till i = T, where T is the largest element in the set τ. For the sake of completeness, we shall now present algorithmically the technique to compute $D_\tau(X,Y)$.

Algorithm ConstrainedDistance

Input : The strings $X = x_1x_2...x_N$, $Y = y_1y_2...y_M$, the set of elementary edit distances and the constraint set τ, whose largest element is $T > 0$. Let R=Min[M,N].

Output : The constrained edit distance $D_\tau(X,Y)$.

Method :

Wis (0,0) = Wie (0,0) = Wes0(0,0) = 0

For s = 1 **to** Min[M,N] **do** /* initialize s-axis */

Wis(0,s) = Wes0(0,s) = Wis (0,s-1) + d (x_s, y_s)

For e = 1 **to** N **do** /* initialize e-axis */

Wie (0,e) = Wes0(e,0) = Wie (0,e-1) + d(x_e, θ)

For i = 1 **to** T **do** /* initialize i-axis */

Wis(i,0) = Wie (i,0) = Wis (i-1,0) + d(θ, y_i)

For i = 1 **to** T **do** /* Compute Wie-plane */

For e = 1 **to** N-M+i **do**

Wie (i,e) = Min [Wie(i-1, e) + d (θ, y_{i+s}), Wie(i,e-1) + d (x_e,θ)]

For e = 1 **to** N **do** /* Compute Wes-plane parallel to i=0 */

For s = 1 **to** N-e **do**

Wes0(e,s) = Min [Wes0 (e-1, s) + d (x_{e+s},θ) , Wes0(e,s-1)+d(x_{e+s}, y_s)]

If 0 ∈ τ **then** $D_\tau(X,Y)$ = Wes0(N-M, M) /* Initialize D_τ (X,Y) */

Else $D_\tau(X,Y) = \infty$

For i = 1 **to** T **do** /* Trace planes parallel to i=0 */

Begin

For e = 1 **to** N-M + i **do** /* Update line parallel to s-axis from Wis plane */

Wes1 (0,s) = Wis (i,s)

For s = 1 **to** Min [N, M-i] **do** /* Update line parallel to e-axis from Wie */

Wes1(e,o) = Wie (i,e) /* plane */

For e = 1 **to** N-M + i **do** /* Compute Wes1 using Theorem III */

For s = 1 **to** Min [N-e,M-i] **do**

Wes1(e,s) = Min [Wes1(e,s-1) + d(x_{e+s}, y_{i+s}),

Wes1(e-1,s)+d(x_{e+s},θ), Wes0(e,s)+d(θ,y_{i+s})]

If i ∈ τ **Then** D_τ (x,y) = Min [D_τ (X,Y) , Wes1(N-M+i, M-i)]

For e = 1 **to** N **do** /* Update Wes0 from Wes1 */

For s = 1 **to** N-e **do**

Wes0(e,s) = Wes1 (e,s)

End

END ALGORITHM ConstrainedDistance

Theorem V.

The worst case space complexity of Algorithm ConstrainedDistance is 0 (MN).

PROOF : The array Wie requires MN memory locations. Similarly, the array Wis requires MR memory locations, where, R = Min (M,N). Finally, the arrays Wes0 and Wes1 require NR memory locations respectivley. Hence the space requirement of the algorithm is:

$MN + MR + 2NR = O(MN)$.

***

Theorem VI

Let τ be the given constraint set, whose maximum value is T. Then if $N \geq M$, the worst case time complexity of Algorithm ConstrainedDistance is $O(N^2 + T^2N)$,

PROOF : Let us assume that $N \geq M$. Obviously, the worst case time complexity has an upper bound if the analysis was performed by assuming that both the strings were of length N.

The computational complexity of algorithms involving the comparison of two strings is conveniently given by the number of symbol comparisons required by the algorithm, [1,20,33]. In this case, an upper bound on the number of symbol comparisons required by Algorithm ConstrainedDistance can be evaluated by the number of times the distance matrix d(.,.) must be referred to. We list the counts below:

(1) To initialize the s-axis : N look-ups necessary.

(2) To initialize the e-axis : N look-ups necessary

(3) To initialize the i-axis : T look-ups necessary

(4) To compute the Wie plane : $2 . \sum_{1 \leq i \leq T} (i)$ look-ups necessary

(5) To compute the Wis plane : $2 . \sum_{1 \leq i \leq T} (N-i)$ look-ups necessary

(6) To compute the Wes0 plane : $2 . \sum_{1 \leq e \leq N} (N-e)$ look-ups necessary

(7) To compute the Wes1 plane in all : $3.\sum_{1 \le i \le T} \sum_{1 \le e \le i} (N-e)$ look-ups.

Hence, the total number of symbol comparisons necessary is:

$$\# \text{ (symbol comparisons)} \le 2N + T + T(T+1) + 2TN - T(T+1) + 2N^2 - N(N+1)$$

$$+ 3 \sum_{1 \le i \le T} (Ni - i(i+1)/2)$$

$$= 2N^2 + 2TN + 2N + 1/2\,(3TN + 3T^2N - 3T^2 - T^3)$$

$$\le 2N^2 + 2TN + 2N + 1/2\,(3TN + 3T^2N)$$

$$= 0(N^2 + T^2N).$$

Hence the theorem! ***

Corollary 1. If the set τ contains only one element T and $T << N$, then the time complexity of the algorithm is $0(N^2)$.

V. THE NOISY SUBSEQUENCE CORRECTION PROBLEM

Let us assume the characteristic of the noisy channel are known, and further, let L be the **expected** number of insertions introduced in the process of transmitting U. This figure can be estimated [3, 17] although the actual number of symbols inserted in any particular transmission is unknown. Since U can be any arbitrary subsequence of X^*, and since the words of the dictionary can be of completely different lengths, it is obviously meaningless to compare Y with every $X \in H$ using the GLD. Thus, before we compare Y with the individual words of the dictionary, we have to use the additional information obtainable from the noisy channel.

Since the number of insertions introduced in any transmission is unknown, it is reasonable to compare $X \in H$ and Y subject to the constraint that the number of insertions that **actually** took place is its best estimate. Of course, in the absence of any other information, the best estimate of the number of insertions that could have

taken place is indeed its expected value, which we have referred to as L. However, if L is not a feasible value for the number of insertions, then the closest feasible value is used to compare $X \in H$ and Y. This is the rationale for the algorithm presented below.

Algorithm RecognizeSubsequences

Input: (i) The finite dictionary H and L, the expected number of insertions that can take place per transmission.

(ii) Y a noisy version of a subsequence of an unknown $X^* \in H$.

Output: The estimate X^+ of X^*.

Method:

For every string $X \in H$ **do**

Begin

If L is a feasible value **Then**

T = L

Else

T = closest feasible integer to L.

Compute $D_{\{T\}}(X,Y)$ using Algorithm ConstrainedDistance

End

X^+ is the string minimizing $D_{\{T\}}(X,Y)$

END Algorithm RecognizeSubsequences

Remark : Observe that the constraint set τ that we have used could have been equivalently defined in terms of substitutions, deletions or insertions, as described in Section 2. However, since the string transmitted can be any arbitarary subsequence of X^*, we believe it is most appropriate to define it in terms of the number of insertions occuring in the transmission rather than in terms of any other type of operation.

VI. EXPERIMENTAL RESULTS

The technique reported in this paper has been rigoroursly tested for thousands of noisy subsequences with strings of various lengths. The results obtained are remarkable. A detailed report of the dictionary used, the noisy strings generated and the edit distances used in the experiments follows.

VI.1 The Dictionary Used

The dictionary used consisted of a hundred strings taken from the classical book on pattern recognition by Duda and Hart [8]. Each string was the first line of a section or sub-section of the book, starting from Section 1.1 and ending with Section 6.4.3. Further to mimic an UNIX nroff (or troff) file, all the Greek symbols were typed in as English strings. Subsequently, to make the problem more difficult, the spaces were eliminated, thus discarding the contextual information obtainable by using the blanks as delimiters. Finally, these strings were randomly truncated so that the length of the words in the dictionary was uniformly distributed in the interval [40, 80]. Thus, the first line of the Section 3.4.1 of [8] which reads as below:

"In this section we calculate the a posteriori density p(θ/X) and the desired probability"

yielded the following string in the dictionary H:

"inthissectionwecalculatetheaposterioridensitypthetaxandthedesiredpro".

A subset of the hundred words used in the dictionary is given in Table I.

VI.2 Generation of Noisy Strings and Results Obtained

Noisy strings used in the experiment were generated using the two strategies given below.

Noisy String Generation: Technique A

In this technique noisy strings were generated using the channel model described in [17]. The method is as follows: A string X^* was initially chosen from H. The number of insertions in Y was randomly obtained from a geometric distribution, and the positions of these insertions and the symbols inserted were assumed to be

equally likely. Subsequently, the individual symbols of U were either deleted or substituted by a predefined probability distribution. This distribution was symbol dependent and was related to the relative proximity of the individual letters of the alphabet on a typewriter keyboard. However, to make the problem meaningful, the probability of deleting a symbol $a \in A$ was made 0.35 times the probability of transmitting it correclty. This implied, that although the mean length of the string X^* was 60, the average number of errors per received string Y, after the substitution, insertion and deletion errors were included, was 26.547. In this particular case, the mean of the number of insertions per transmission was 2.

The individual edit distances were calculated as follows for all a, b, $\in$, A.

$d(a, b) = -\ln [Pr(a \rightarrow b) / Pr(a \rightarrow a)]$

$d(a,\theta) = -\ln [Pr(a \rightarrow \theta) / Pr(a \rightarrow a)]$

$d(\theta , a) = -K. \ln [Pr(a \text{ is inserted} \mid \text{insertion occured}) / Pr(a \rightarrow a)]$

The definition of the distances as above is consistent with the distance assignments made in the literature by authors who have used the GLD in string correction [10, 14, 16, 19]. However, in this case, we have added a multiplying constant in the definition of the edit distance associated with insertions so as to render the triangular law of inequalities valid [23].

The algorithm reported in this paper was tested for a thousand strings. Of these, 995 were correctly recognized, thus reprersenting an accuracy of 99.5%. The average number of errors for the wrongly classified strings was 37.6.

Noisy String Generation : Technique B

In the second techniques the noise generation was made even more **rigorous** than in the first. A string X^* was initially chosen. Subsequently, alternate contiguous substrings of X^* were deleted and retained. These substrings were of lengths that were Poissonly distributed and with parameter 6. The net resultant subsequence U had an average of 30.24 characters deleted from X^*. This string U was now further subjected to substitution, insertion and deletion errors using a technique identical to

the one described in Technique A. For the sake of completeness, a list of 40 of the 500 noisy strings generated is given in Table II. In the table, we have given the index of X^*, the string U, the noisy string Y and the number of errors introduced in the process of transforming X^* to Y. The average number of errors in this case is 44.72. In this case, out of the 500 noisy strings tested 477 were correctly recognized. This implies an accuracy of 95.4%. The average number of errors per wrongly estimated string is 46.696. The power of the scheme is obvious.

VI. CONCLUSIONS

In this paper we have considered the problem of recognizing strings by processing their noisy subsequences. The solution which we propose, which is the only known solution in the literature, has a quadratic time complexity and is remarkably accurate. Given a noisy subsequence Y of an unknown string $X^* \in H$, the technique we propose estimates X^* by computing the constrained edit distance between every $X \in H$ and Y. The procedure to compute the latter edit distance has been described in detail and a rigorous time and space analysis for the procedure has been performed.

Experimental results using strings of length between 40 and 80 and with a high percentage of noisy symbols, demonstrate the power of the strategy we have proposed.

We are currently investigating the use of the techniques described in this paper for the recognition of closed contours using the noisy information obtained from a subsequence of the string representation [19] of the entire contour.

ACKNOWLEDGEMENTS

I am indebted to David Ng for implementing the algorithm for me and for implementing the noise generation techniques. I am also grateful to Christine Marland who decoded my handwriting and typed the manuscript for me.

REFERENCES

1. Aho, A. V., Hirschberg D.S., Ullmann J. R., "Bounds on the Complexity of the Longest Common Sub-Sequence Problem", J-ACM, Vol. 23, pp.1-12, 1976.

2. Aho, A. V., and Peterson, T. G., "A Minimum Distance Error-Correcting Parser for Context-Free Languages", SIAM J. Comput. Vol. 1, pp.305-312, 1972.

3. Bahl, R. L., and Jelinek, F., "Decoding for Channels with Insertions, Deletions, and Substitutions with Application to Speech Recognition", IEEE Trans. on Information Theory, IT-21, pp. 404-411, 1975.

4. Berlekamp, E.R., "Algebraic Coding Theory", McGraw Hill, New York, 1968.

5. Blair, C.R., "A program for Correcting Spelling Errors", Inf. and Cont., pp. 60-67, 1960.

6. Damerau, F. J., "A Technique for Computer Detection and Correction of Spelling Errors", C-ACM, Vol. 7, pp.171-176, 1964.

7. Denning, D., "Cryptography and Data Security", Addison Wesley, 1982.

8. Duda, R. D. and Hart, P.E., "Pattern Classification and Scene Analysis", John Wiley, 1973.

9. Forney, G. D., "The Viterbi Algorithm", Proceedings of the IEEE, Vol. 61, 1973.

10. Fung, L. W. and Fu, K. S., "Stochastic Syntactic Decoding for Pattern Classification", IEEE Trans. Comp., C-24, pp. 662-667, 1975.

11. Hall, P.A.V., and Dowling G.R., "Approximate String Matching", Computing Surveys, Vol. 12, pp. 381-402, 1980.

12. Hirschberg, D.S., "A Linear Space Algorithm for Computing Maximal Common Subsequences", C-ACM, Vol. 18, pp. 341-343, 1975.

13. Hunt, J.W., and Szymanski, T.G., "A Fast Algorithm for Computing Longest Common Subsequences", C-ACM, Vol. 20, pp. 350-353, 1977.

14. Kashyap, R.L., "Syntactic Dec. Rules for Recog. of Spoken Words and Phrases Using a Stochastic Automaton", IEEE Trans. on Pat. Anal. and Mach. Intel., PAMI-1, pp.154-163, 1979.

15. Kashyap, R.L., and Oommen, B.J., "The Noisy Substring Matching Problem", IEEE Trans. on Software Eng., Vol. SE-9, pp. 365-370, 1983.

16. Kashyap, R.L., and Oommen, B.J., "An Effective Algorithm for String Correction using Generalized Edit Distances -I. Description of the Algorithm and its Optimality", Information Sciences, Vol. 23, No. 2, pp. 123-142, 1981.

17. Kashyap, R.L., and Oommen, B.J., "Probablistic Correction of Strings", Proc. of the IEEE Trans. on Pat. Recog. and Image Processing, pp. 28-33, 1982.

18. Levenshtein, A., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," Sov. Phy. Dokl., Vol. 10, pp. 707-710, 1966.

19. Lu, S.Y. and Fu, K.S., "A Sentence-to-Sentence Clustering Procedure for Pattern Analysis", Proc. IEEE Comp. Soc., 1st International Comp. Software and App. Conference, pp. 492-498, 1977.

20. Maier, D., "The Complexity of Some Problems on Subsequences and Supersequences," J-ACM, Vol. 25, pp. 322-336, 1978.

21. Masek, W.J. and Paterson, M. S., "A Faster Algorithm Computing String Edit Distances", Journal of Comp. and Sys. Sciences, Vol. 20, pp. 18-31, 1980.

22. Neuhoff, D. L., "The Viterbi Algorithm as an Aid in Text Recognition", IEEE Trans. Information Theory, IT-21, pp. 222-226, 1975.

23. Okuda, T., Tanaka, E., and Kasai, T., "A Method for Correction of Garbled Words Based on the Levenshtein Metric", IEEE Trans. Comp., C-25, pp. 172-177, 1976.

24. Oommen, B.J., "Constrained String Editing", To Appear in Information Sciences.

25. Peterson, W.W., and Weldon, E.J., "Error Correcting Codes", MIT Press, Second Edition, 1981.

26. Raviv, J., "Decision Making in Markov Chains Applied to the Problem of Pattern Recognition", IEEE Trans. Information Theory, pp. 536-551, 1967.

27. Sankoff, D., "Matching Sequences under Deletion/Insertion Constraints", Proc. of the Nat. Acad. Science, USA, Vol. 69, pp. 4-6, 1972.

28. Sankoff, D., and Kruskal, J.B., "Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison", Addison Wesley, 1983.

29. Shinghal, R., and Toussaint, G.T., "Experiments in Text Recognition with the Modified Viterbi Algorithm", IEEE Trans on Pat. Anal. and Mach. Intel., PAMI-1, pp. 184-192, 1979.

30. Srihari, S., "Computer Text Recognition and Error Correction", IEEE Computer Society Press, 1984.

31. Viterbi, A. J., "Error Bounds for Convolutional Codes and an Asymptotically Optimal Decoding Algorithm", IEEE Trans. on Information Theory, IT-13, pp.260-269, 1967.

32. Wagner, R. A., and Fisher, M.J., "The String to String Correction Problem", J-ACM, Vol. 21, pp. 168-173, 1976.

33. Wong, C. K. and Chandra, A.K., "Bounds for the String Editing Problem, J-ACM, Vol. 23, pp. 13-16, 1976.