PATTERN RECOGNITION OF STRINGS WITH SUBSTITUTIONS, INSERTIONS, DELETIONS AND GENERALIZED TRANSPOSITIONS[1]

B. J. Oommen and R. K. S. Loke
School of Computer Science
Carleton University
Ottawa ; CANADA : K1S 5B6

ABSTRACT

We study the problem of recognizing a string Y which is the noisy version of some unknown string X^* chosen from a finite dictionary, **H**. The traditional case which has been extensively studied in the literature is the one in which Y contains substitution, insertion and deletion (SID) errors. Although some work has been done to extend the traditional set of edit operations to include the straightforward transposition of adjacent characters[2] [14] the problem is unsolved when the transposed characters are themselves subsequently substituted, as is typical in cursive and typewritten script, in molecular biology and in noisy chain-coded boundaries. In this paper we present the first reported solution to the analytic problem of editing one string X to another, Y using these four edit operations. A scheme for obtaining the optimal edit operations has also been given. Both these solutions are optimal for the infinite alphabet case. Using these algorithms we present a syntactic pattern recognition scheme which corrects noisy text containing all these types of errors. The paper includes experimental results involving sub-dictionaries of the most common English words which demonstrate the superiority of our system over existing methods.

[1]Partially supported by the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper can be found in the *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, October 1995, pp. 1154-1159.

[2]There has been some recent work done [Oo93] to consider the squashing and expansion operations too, where in the squashing operation two (or more) contiguous characters of X can be transformed into a single character of Y, and in the expansion operation a single character in X may be expanded into two or more contiguous characters of Y.

I. INTRODUCTION

A common problem in text editing is that of finding the occurrence of a given string in a file. This is usually done to locate one's bearings in the file or to replace the occurrence of one string by another. With no loss of generality, the file can be considered as a sequence of words from a finite dictionary. One mishap that often occurs is that the string sought for is noisily represented, either due to mistyping or ignorance of spelling. We study the problem of recognizing the original string by processing its noisy version.

Apart from text editing, spelling correction has numerous applications in text and document retrieval using noisy keywords, word processing, designing effective spelling checkers and in image processing where the boundary of the image to be recognized is syntactically coded as a string [11]. Inexact sequence comparison is also used extensively in the comparison of biological macro-molecules where "noisy" version of proteins strings are compared with their exact forms typically stored in large protein databases. The literature on spelling correction is extensive; indeed, the reviews [2,20] list hundreds of publications that tackle the problem from various perspectives.

Damareau [See references in 2,20] was probably the first researcher to observe that most of the errors that were found in strings were either a single Substitution, Insertion and Deletion (SID) or reversal (transposition) error. Thus the question of computing the dissimilarity between strings was reduced to comparing them using these edit transformations. In most of the existing literature the transposition operation has been modelled as a sequence of a single insertion and deletion. 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 the SID 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 [13], 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.* [19] extended this concept by weighting the edit operations, and many other researchers among whom are Wagner and Fisher [23] 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 GLD is that it can be made a metric if the individual edit distances obey some constraints [19,21]. Wagner and Fischer [23] also proposed an efficient algorithm for computing this distance using dynamic programming. This algorithm has been proved to be the optimal algorithm for the infinite alphabet case. Various amazingly similar versions of the algorithm are available in the literature, a review of which can be found in [21]. Masek and Paterson [15] improved the algorithm for the finite alphabet case.

Related to these algorithms are the ones used to compute the Longest Common Subsequence (LCS) of two strings due to Hirschberg [3,4], Hunt and Szymanski [5] and others [21]. The complexity of the LCS problem has been studied by Aho *et. al.* [1]. String correction using the GLD as a criterion has been done for strings [2,20,21], substrings [9], dictionaries treated as generalized tries [7] and for grammars [21]. Besides these deterministic techniques, various probabilistic methods have been studied in the literature [2,20,10].

Although some work has been done to extend the traditional set of SID operations to include the transposition of adjacent characters [14,21] the problem is unsolved for "Generalized" Transposition (GT) errors. Indeed, as can be seen from the vast body of literature, typical string comparison problems are solved by assuming that there is no correlation between neighboring characters within a word or sentence. If one views the elements of the confusion matrices as probabilities this is equivalent to assuming that the transformation probabilities at each position in the string are statistically independent and possess first-order Markovian characteristics. This model is usually assumed for simplicity rather it having any statistical significance. Indeed, in molecular biological applications it is expected that neighboring characters are highly correlated but this correlation is ignored in the first approximation because of performance considerations.

The method of comparing strings using transpositions essentially includes a nearest-neighbour correlation term. However, as opposed to the work of [14,21] the concepts introduced here are intended to account for simultaneous nearest-neighbour transposition-substitution events[3]. Certainly, such event would result in an observed correlation among nearest-neighbour positions; however, there are many other factors that could conceivably contribute. Indeed, as already noted there is good evidence that the structural properties of biomolecular sequences result directly in neighbouring correlations.

In this paper we revisit the problem for this setting. The difference between the latter errors and those traditionally considered as "transposition errors" is the following. Currently, transposition errors merely imply errors caused when the order of adjacent symbols in a sequence are reversed. Such an error could cause the string "develop" to be mutated into "dveelop". As opposed to this, *Generalized* Transposition (GT) errors permit these transposed symbols to be subsequently substituted. Thus, if one was working on a QWERTY typewriter keyboard, this could cause the string "develop" to be mutated into "dbrelop" -- which would arise when the typist inherently "reversed" the two characters ("ev") due to the sequence in which the fingers touched the keyboard, but also accidentally shifted his/her hands to the right of the keyboard one key too far -- which happens all too often. Of course, it is clear that GT errors can be represented as a sequence of two substitutions ('e' $\varnothing$ 'b', and 'v' $\varnothing$ 'r'). However, we shall show that the recognition accuracies involved by representing them as GTs is *much* more than can be obtained by representing them as two substitutions. Furthermore, it will become clear that the additional computational burden is but marginal; the order of the two complexities is identical -- both being optimal and quadratic.

We formalize the problem as follows. We are given a string Y which is the noisy version of some unknown string X^* chosen from a finite dictionary, **H**. Apart from Y containing SID errors, it also contains transposed characters which are themselves subsequently substituted. The intention is to recognize X^* by processing Y. To achieve this we present the first reported solution to the analytic problem of editing one string X to another, Y using these four edit operations. A scheme for obtaining the optimal edit operations has also been given. Both

[3]The difference between the **types of operations** considered here and in the results of by Lowrance and Wagner [14,21] for traditional transpositions is explained below. The salient differences between our **algorithm** and theirs are explained in a subsequent section.

these solutions are optimal for the infinite alphabet case. Using these algorithms we present a syntactic PR scheme which corrects noisy text containing all these types of errors.

This "new" GT operation is not only applicable in the recognition of typewritten and cursive script, but also has vast potential application in processing of chain-coded images [11] and biological macro-molecules. To see the former, consider the representation of the a handwritten cursive "2". A study of various boundaries shows that the "hook" at the top of "2" varies with the writer --some "hooks" being more curved than others. A less curved "hook" can have a "0101" chain-coded representation, which is equivalent to "1010" when the symbols are transposed. As opposed to this, a more curved "hook" can have the code "6710", which is precisely edited from "0101" by two GTs, where the symbols of one of the transpositions has been subsequently substituted. Indeed, such scenarios are numerous in boundary representations. GT errors are also encountered in the study of biological macro-molecules [21] where the mutation (substitution) of transposed molecules occurs in the "next" generation after the proteins in any particular sequence are transposed.

I.1 Notation

A is a finite alphabet, and $\mathbf{A}^*$ is the set of strings over **A**. θ is the null symbol, where θ **A**, and is distinct from μ the empty string. Let $\hat{\mathbf{A}} = \mathbf{A} \approx \{\theta\}$. $\hat{\mathbf{A}}$ is referred to as the *Appended Alphabet*. A string X $\mathbf{A}^*$ of the form $X = x_1 \ldots x_N$, where each x_i **A**, and is said to be of length |X| = N. Its prefix of length i will be written as X_i, for 1 i N. Uppercase symbols represent strings, and lower case symbols, elements of the alphabet under consideration.

Let Z' be any element in $\hat{\mathbf{A}}^*$, the set of strings over $\hat{\mathbf{A}}$. The *Compression Operator* C is a mapping from $\hat{\mathbf{A}}^*$ to $\mathbf{A}^*$: 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.

We now define the costs associated with the individual edit operations. If $\mathbf{R}^+$ is the set of nonnegative real numbers, we define the elementary edit distances using four elementary functions $d_s(.,.)$, $d_i(.)$, $d_e(.,.)$, $d_t(.,.)$ defined as :

(i) $d_s(p,q)$ is a map from **A** X **A**∅ $\mathbf{R}^+$ and is called the Substitution Map. In particular, $d_s(a,b)$ is the distance associated with substituting b for a, a,b **A**. For all a **A**, $d_s(a,a)$ is generally assigned the value zero, although this is not mandatory.

(ii) $d_i(.)$ is a map from **A** ∅ $\mathbf{R}^+$ and is called the Insertion Map. The quantity $d_i(a)$ is the distance associated with inserting the symbol a **A**.

(iii) $d_e(.)$ is a map from **A** ∅ $\mathbf{R}^+$ and is called the Deletion or Erasure Map. The quantity $d_e(a)$ is the distance associated with deleting (or erasing) the symbol a **A**.

(iv) $d_t(.,.)$ is a map from $\mathbf{A}^2$ X $\mathbf{A}^2$∅ $\mathbf{R}^+$ called the Transposition Map. The quantity $d_t(ab,cd)$ is the distance associated with transposing the string "ab" into "cd". This can be thought of as a "serial" operation: "ab" is first transposed to "ba" and subsequently the individual characters are substituted.

I.2 The Set of Edit Possibilities : $\Gamma_{X,Y}$

For every pair (X,Y), X, Y $\mathbf{A}^*$, the finite set $\Gamma_{X,Y}$ is defined by means of the compression operator C, as a subset of $\hat{\mathbf{A}}^*$ X $\hat{\mathbf{A}}^*$ as :

$\Gamma_{X,Y} = \{ (X',Y') \mid (X',Y')$ $\hat{\mathbf{A}}^*$ X $\hat{\mathbf{A}}^*$, and each (X',Y') obeys

(i) C(X') = X, C(Y') = Y,

(ii) |X'| = |Y'|,

(iii) For all $1 \leq i \leq |X'|$, it is not the case that $x_i' = y_i' = \theta$ $\}$. (1)

By definition, if (X',Y') $\Gamma_{X,Y}$ then Max (|X|, |Y|) $\leq$ |X'| = |Y'| $\leq$ |X| + |Y|.

Each element in $\Gamma_{X,Y}$ corresponds to one way of editing X into Y, using the SID operations. The edit operations themselves are specified for all $1 \leq i \leq |X'|$ by (x_i', y_i'), which represents the transformation of x_i' to y_i'. The cases below consider the SID operations :

(i) If x_i' **A** and y_i' **A**, it represents the substitution of y_i' for x_i'.

(ii) If x_i' **A** and $y_i' = \theta$, it represents the deletion of x_i'.

(iii) If $x_i' = \theta$ and y_i' **A**, it represents the insertion of y_i'.

$\Gamma_{X,Y}$ is an exhaustive enumeration of the set of all the ways by which X can be edited to Y using the SID operations. However, on examining the individual elements of $\Gamma_{X,Y}$ it becomes clear that each pair contains more information than that. Indeed, *in each pair*, there is also information about the various ways by which X can be edited to Y even if the set of edit operations is extended so as to include GTs. Thus, when (X',Y') = (abθ, cde), apart from the operations described above, the pair also represents the GT of 'ab' to 'cd' and the insertion of 'e'.

Observe that the transformation of a symbol a **A** to itself is also considered as an operation in the arbitrary pair (X',Y') $\Gamma_{X,Y}$. Also note that the same set of edit operations can be represented by multiple elements in $\Gamma_{X,Y}$. This duplication serves as a powerful tool in the proofs of various analytic results [7,8,10,11,18].

Since the Edit Distance between X and Y is the minimum of the sum of the edit distances associated with operations required to change X to Y, this distance, D(X,Y), has the expression :

$$D(X,Y) = \min_{((X',Y') \; \Gamma_{X,Y})} \sum_{i=1}^{|J'|} [\text{ Distances Associated with the Operations in (X', Y')}], \quad (2)$$

where, (X',Y') represents J' possible edit operations.

II. THE RECURSIVE PROPERTIES OF THE EDIT DISTANCE

Let D(X,Y) be the distance associated with transforming X to Y with SID and GT operations. We shall describe how D(.,.) can be computed. To achieve this, we shall first derive the properties of D(X,Y) which can be derived recursively in terms of the corresponding quantities defined in terms of the prefixes of X and Y, (X_i and X_j respectively) with the assumption that D(μ, μ) is zero.

LEMMA 0a.

Let $X = X_i = x_1...x_i$ be the prefix of X and $Y = \mu$, the null string. Then, $D(X_i, \mu)$ obeys :

$$D(X_i, \mu) = D(X_{i-1}, \mu) + d_e(x_i). \qquad \rightarrow\rightarrow\rightarrow$$

LEMMA 0b.

Let $X = \mu$, and $Y_j = y_1...y_j$ be the prefix of Y. Then, $D(\mu, Y_j)$ obeys :

$$D(\mu, Y_j) = D(\mu, Y_{j-1}) + d_i(y_j). \qquad \rightarrow\rightarrow\rightarrow$$

LEMMA 0c.

Let $X = x_1$ and $Y = y_1$. Then, D(X, Y) obeys :

$$D(X, Y) = \textbf{Min} \left[D(\mu, Y) + d_e(x_1), \quad D(X, \mu) + d_i(y_1), \quad d_s(x_1, y_1) \right]. \qquad \rightarrow\rightarrow\rightarrow$$

We shall now state and prove the main result of our paper.

THEOREM I.

Let $X_i = x_1...x_i$ and $Y_j = y_1...y_j$ with i , j 2. Also, let $D(X_i, Y_j)$ be the edit distance associated with the transforming X_i to Y_j with the SID and GT operations. Then, the following is true :

$$D(X_i, Y_j) = \textbf{Min} \left[D(X_{i-1}, Y_j) + d_e(x_i), \quad D(X_i, Y_{j-1}) + d_i(y_j), \right.$$
$$\left. D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j), \quad D(X_{i-2}, Y_{j-2}) + d_t(x_{i-1}x_i, y_{j-1}y_j) \right].$$

Proof : The proof of the theorem is given in Appendix A. $\rightarrow\rightarrow\rightarrow$

III. THE COMPUTATION OF D(X,Y)

To compute D(X,Y) we make use of the recursive properties given above. The idea is essentially one of computing the distance $D(X_i, Y_j)$ between the prefixes of X and Y. The computation of the distances has to be done in a schematic manner, so that any quantity $D(X_i, Y_j)$ is computed before its value is required in any further computation. This can be actually done in a straightforward manner by tracing the underlying graph, commonly referred to as a *trellis* and maintaining an array Z(i,j) defined for all 0 i N and 0 j M when |X| = N and |Y| = M. The quantity Z(i,j) is nothing but $D(X_i, Y_j)$. We will discuss the properties of the our particular trellis subsequently.

The algorithm to compute Z(.,.) is given below.

ALGORITHM Distance_SID_GT

Input : The strings $X = x_1...x_N$ and $Y = y_1...y_M$, and the set of elementary edit distances defined using the five elementary functions $d_s(.,.)$, $d_i(.)$, $d_e(.)$, $d_t(.,.)$.

Output : The distance D(X, Y) associated with editing X to Y using the SID and GT operations.

Method :

$Z(0, 0) \blacklozenge 0$

For i $\blacklozenge$ 1 to N Do

  $Z(i, 0) \blacklozenge Z(i-1, 0) + d_e(x_i)$

For j $\blacklozenge$ 1 to M Do

  $Z(0, j) \blacklozenge Z(0, j-1) + d_i(y_j)$

For i $\blacklozenge$ 1 to N Do

  $Z(i, 1) \blacklozenge \text{Min} [Z(i-1, 1) + d_e(x_i), Z(i, 0) + d_i(y_1), Z(i-1, 0) + d_s(x_i,y_1)]$

For j $\blacklozenge$ 2 to M Do

  $Z(1, j) \blacklozenge \text{Min} [Z(1, j-1) + d_i(y_j), Z(0, j) + d_e(x_1), Z(0, j-1) + d_s(x_1,y_j)]$

For i $\blacklozenge$ 2 to N do

  For j $\blacklozenge$ 2 to M do

$$Z(i,j) \blacklozenge \text{Min} [Z(i-1, j) + d_e(x_i), \quad Z(i, j-1) + d_i(y_j), \quad Z(i-1, j-1) + d_s(x_i,y_j), \quad Z(i-2, j-2) + d_t(x_{i-1}x_i , y_{j-1}y_j)]$$

$D(X,Y) \blacklozenge Z(N, M)$

END ALGORITHM Distance_SID_GT

Remarks

1. The computational complexity of string comparison algorithms is conveniently given by the number of symbol comparisons required by the algorithm [1,6,21]. In this case, the number of symbol comparisons is quadratic. In the body of the main loop, we will need at most four additions and at most four minimizations. The lower bound result claimed in [6] naturally implies that our algorithm is optimal for the infinite alphabet case. This is because, first of all, we have not placed any restrictions on the edit costs. Also, the lower bound of [6] applies to the more restricted problem of finding a minimum cost alignment. Finally, when GTs have infinite costs, our underlying problem contains the traditional string alignment problem as a special case.

2. In 1975, Lowrance and Wagner [14] extended the traditional set of SID operations to include the operation of transposing two adjacent characters. We shall now discuss their results so as to illustrate the difference between [14] and our current results.

First of all, as opposed to the traditional set of SID operations, when straightforward transpositions are permitted the lines in the underlying traces are permitted to intersect [14]. In order for Lowrance and Wagner's algorithm to work these intersection lines have to be "tightly constrained". This, in turn, is achieved by constraining the *symbol dependent* costs for the edit operations (represented by W_s, W_t W_i and W_d for the cost of substitution, transposition, insertion and deletion respectively) by a rigid inequality. Indeed, for Lowrance and Wagner's algorithm to work the weights have to obey :

(i) $W_s < W_i + W_d$, and, (ii) $2W_t \quad W_i + W_d$.

The first of these is clearly the triangular inequality. The second can be seen to be a necessary condition to ensure sure that a trace can be partitioned into smaller sub-traces with at most one line crossing. In other words, this ensures that the prefix of a trace would not cross over to its suffix. Theorems III-V in [14] prove that the lines in a trace will either cross a single line (when a transposition occurs) or none at all (when a substitution occurs). Furthermore, this is true only when the transposed symbols before and after the transposition are identical. Under these conditions, if the indices (h_1 and h_2) of the transposed symbols are known, it can be shown that no symbol in X in the range (h_1, h_2) is exactly the same as the symbol at x_{h_1}. The same is true for the symbols in Y.

The reader should observe that the second constraint is necessary for Lowrance and Wagner's algorithm to work. If the inequality is not satisfied (for example $W_s > W_i + W_d > 2W_t$), the minimal cost trace may be a "daisy-chain" containing lines which touch every character of X and Y, and with lines intersecting more than one other line as seen in Figure I below.

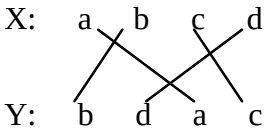

Figure 1: A "daisy-chain" trace obtained when the inequality required by Lowrance and Wagner is not satisfied.

Clearly, our present algorithm requires no such constraint and is therefore much more general. First of all it permits GTs -- the substitution of characters that have been physically displaced by straightforward transposition. Secondly, it permits the distances to be fairly arbitrary -- they can be chosen to reflect the confusion matrix of the garbling mechanism -- as is done in typical PR applications [7,17,21]. Last of all, (but if not the most important), is the relative simplicity of the present scheme -- it is but a straightforward generalization of the Wagner-Fischer algorithm.

3. A note about the *modus operandus* of the proof of Theorem I is not out of place. From a superficial perspective it is possible to consider our scheme to be a mere application of dynamic programming to extensions of a widely studied problem. The latter is not the case. There is a very fine point in which our proof differs from the proofs currently described in the literature. The fundamental difference is that in the current proof, whenever the set over which the minimization is achieved is grown, it is not merely a single optimization scenario which is encountered. Thus in Case 9 of the proof in the Appendix, there are two possible scenarios by which the minimization can be achieved. The **same four terms** appear in their different combinations in various cases encountered in the minimization process. This, augmented with the fact that we have not required the distances associated with the transposition operations to obey any "generalized triangular inequality", make our proof more interesting and different from the proofs of [14,22,23]. Rather, just as in [17] the concept is reminiscent of a

control system in which various outputs are computed in terms of the **same** state variables by using different "Output Functions".

III.1 Graphical Representation of the Algorithm and an Example

In the computation of various string similarity and dissimilarity measures, the underlying graph that has to be traversed is commonly called a *trellis*. This trellis is 2-dimensional in the case of the GLD [2, 8, 15, 19, 21, 23], the Length of the LCS [3,4,21] and the Length of the Shortest Common Supersequence [8]. Indeed, the same trellis can be traversed using various set operators to yield the Set of the LCSs and the Set of the Shortest Common Supersequences [8]. The trellis becomes 3-dimensional when one has to compute string probabilities [10], constrained edit distances [16] and correct noisy subsequences [17]. Although the trellis itself is 2-dimensional in the former examples, because the graphs are **cycle-free** they can be represented and traversed by merely maintaining single-dimensional structures [3]. In the cases of computing string probabilities [10], constrained edit distances [16] and correcting noisy subsequences [17], although the trellises are 3-dimensional, they can be represented and traversed using 2-dimensional arrays.

Even though the set of edit operations has been expanded the fundamental properties of the underlying trellis remains the same. In this case, the graph G = (V, E), where, V and E are the set of vertices and edges respectively described below :

$V = \{ <i, j> \mid 0 \quad i \quad N, \ 0 \quad i \quad M \quad \}$, and,

$E = \{ (<i, j>, <i+1,j>) \mid 0 \quad i \quad N-1, 0 \quad j \quad M \quad \} \approx \{ (<i, j>, <i, j+1>) \mid 0 \quad i \quad N, 0 \quad j \quad M-1 \quad \} \approx$

$\{ (<i, j>, <i+1, j+1>) \mid 0 \quad i \quad N-1, \ 0 \quad j \quad M-1 \quad \} \approx \{ (<i, j>, <i+2, j+2>) \mid 0 \quad i \quad N-2, \ 0 \quad j \quad M-2 \quad \}$.

The graph essentially has arcs whenever a single edit operation can be applied. Indeed, the algorithm describes an efficient quadratic time scheme by which the trellis can be traversed.

For the sake of clarity a pictorial representation of the graph is given in Figure 2.

Example I.

Let X=ag and Y = bcf. Let us suppose we want to edit X to Y. We shall now follow through the computation of D(ag,bcf) using Algorithm Distance_SID_GT. To begin with, the weight associated with the origin is initialized to be the value zero. The i and j axes are first traversed:

$Z(1, 0) \blacklozenge d_e(a)$ $Z(2, 0) \blacklozenge d_e(a) + d_e(g)$

$Z(0, 1) \blacklozenge d_i(b)$ $Z(0, 2) \blacklozenge d_i(b) + d_i(c)$ $Z(0, 3) \blacklozenge d_i(b) + d_i(c) + d_i(f)$

The lines for i = 1 and j = 1 are then traversed:

$Z(1, 1) \blacklozenge$ **Min** $[Z(0, 1) + d_e(a), Z(1, 0) + d_i(b), Z(0, 0) + d_s(a,b)]$

$Z(1, 2) \blacklozenge$ **Min** $[Z(0, 2) + d_e(a), Z(1, 1) + d_i(c), Z(0, 1) + d_s(a,c)]$

$Z(1, 3) \blacklozenge$ **Min** $[Z(0, 3) + d_e(a), Z(1, 2) + d_i(f), Z(0, 2) + d_s(a,f)]$

$Z(2, 1) \blacklozenge$ **Min** $[Z(1, 1) + d_e(g), Z(2, 0) + d_i(b), Z(1, 0) + d_s(g,b)]$

The strict interior of the trellis for values when both i, j 2 are then traversed :

$Z(2, 2) \blacklozenge$ **Min** [$Z(1, 2) + d_e(g), Z(2, 1) + d_i(c), Z(1, 1) + d_s(g, c), Z(0, 0) + d_t(ag, bc)$]

$Z(2, 3) \blacklozenge$ **Min** [$Z(1, 3) + d_e(g), Z(2, 2) + d_i(f), Z(1, 2) + d_s(g, f), Z(0, 1) + d_t(ag, cf)$]

Indeed, the required edit distance D(ag, bcf) is Z(2, 3). $\rightarrow\rightarrow\rightarrow$

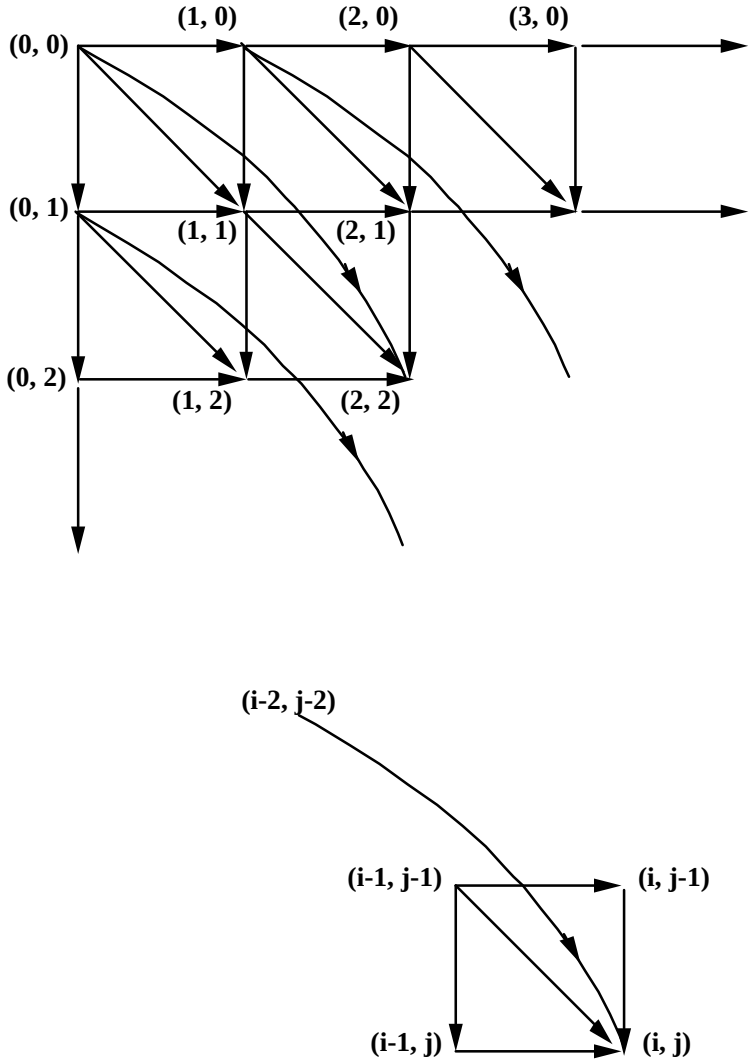

Figure 2 : The Trellis that has to be traversed in order to compute D(X,Y). Note that the only edges terminating at (i, j) are those starting at (i, j-1), (i-1, j), (i-1, j-1) and (i-2, j-2).

III.2 Computing the Best Edit Sequence

Just as in all the edit processes studied in the literature, the traversal of the trellis not only yields the information about the distance between the strings X and Y. By virtue of the way the trellis has been traversed the distances between the prefixes of the strings has also been maintained in the process of computation, and thus, the array Z contains information which can be used to compute the best edit sequence which yields the optimal edit distance. This is done by backtracking through the trellis from the array element (N,M) in the reverse direction of

the arrows so as to reach the origin, always remembering the path that was used to reach the node which is currently being visited. Thus the actual sequence of edit operations can be printed out in the reverse order.

The technique is well known in dynamic-programming problems. It has been used extensively for edit sequences [8, 15, 21, 23] and the LCS problem [3, 4, 5, 8]. Without further comment we now present Algorithm ProduceEditOperations, which has as its input the array Z(.,.). To simplify the code, we exclude the possibility of encountering negative values of i and j by rendering Z(.,.) infinite whenever any of the indices is negative. Obviously Algorithm ProduceEditOperations is performed in O(max(M,N)) time.

ALGORITHM ProduceEditOperations

Input : The strings $X = x_1 \ldots x_N$ and $Y = y_1 \ldots y_M$, the set of elementary edit distances defined as in Algorithm Generalized Distance and the array Z.

Output : The best edit sequence that can transform X to Y using the edit operations of substitution, insertion, deletion and transposition.

Method :

```
Define Z(i,j ) ♦      whenever i < 0 or j < 0.
i ♦  N
j ♦  M
While (i ≠  0 or j ≠  0 ) Do
  If (Z(i, j) = Z(i-1, j-1) + ds(xi, yj)) Then
       Print ("Substitute" xi "by" yj)
       i ♦  i - 1
       j ♦  j - 1
  Else
       If (Z(i, j) = Z(i, j-1) + di(yj)) Then
            Print ("Insert" yj)
            j ♦  j - 1
       Else
            If (Z(i, j) = Z(i-1, j) + de(xi)) Then
                 Print ("Delete" xi)
                 i ♦  i - 1
            Else
                 If (Z(i, j) = Z(i-2, j-2) + dt(xi-1xi, yj-1yj)) Then
                      Print ("Transpose" xi-1xi "into" yj-1yi)
                      i ♦  i - 2
                      j ♦  j - 2
                 EndIf
            EndIf
       EndIf
  EndIf
EndWhile
```

END ALGORITHM ProduceEditOperations

IV. EXPERIMENTAL RESULTS

To investigate the power of our new measure (and its computation) and to demonstrate the accuracy of our new scheme in the original PR problem various experiments were conducted. The results obtained were remarkable. The algorithm was compared with PR results obtained if

(i) only SID errors were assumed as in the case of the Wagner & Fischer [23] algorithm and,

(ii) SID and traditional transposition errors were assumed as in the case of the Lowrance and Wagner [14] algorithm.

The dictionary consisted of 342 words obtained as a subset of the 1023 most common English words [7,9,12] augmented with words used in computer literature. The length of all the words in the dictionary was greater than or equal to 7 and the average length of a word was approximately 8.3 characters. From these words two sets of 1026 noisy strings were generated using the method described in [12] (with the inclusion of GT errors). We shall refer to these sets as **SA** and **SB** respectively. The average percentage number of errors **per word** associated with these two sets was 51.56% and 67.89% respectively.

The conditional probability of inserting any character a A given that an insertion occurred was assigned the value 1/26; and the probability of deletion was set to be 1/20. The table of probabilities for substitution (typically called the confusion matrix) was based on the proximity of the character keys on a standard QWERTY keyboard and is given in Table I. The statistics associated with the sets **SA** and **SB** are given below in Table II. Notice that the percentage error was intentionally made to be large relative to the average length of the words so as to test the various algorithms for such error conditions. A subset of some of the words in SA is given Table III. Notice that some words are very similar even before garbling -- for example, "official" and "officials"; similarly "attention", "station" and "situation" are words whose noisy versions can themselves easily be mis-recognized.

	SA	**SB**
Number of insertions	1629 (1.588)	2747 (2.677)
Number of deletions	478 (0.466)	164 (0.160)
Number of substitutions	709 (0.691)	687 (0.670)
Number of transpositions	1565 (1.525)	2170 (2.115)
Total number of errors	4381 (4.270)	5768 (5.622)
Percentage error	51.56%	67.89%

Table II: Noise statistics of the sets **SA** and **SB**. The figures in brackets are the average number of errors per word

As is typical in PR problems using edit distances [17,21], the individual edit distances were calculated for all a,b,c,d A as follows :

$$d_s(a,b) = -\ln [\ Pr(a \varnothing b) / Pr(a \varnothing a)\ ]$$

$$d_e(a,\theta) = -\ln [\ Pr(a \varnothing \theta) / Pr(a \varnothing a)\ ]$$

$$d_i(\theta,a) = -K \cdot \ln [\ Pr(a \text{ is inserted} \mid \text{insertion occurred}) / Pr(a \varnothing a)\ ]$$

$$d_t(ab,cd) = 1 + d_s(a,d) + d_s(b,c).$$

In the above, the value of K was determined as the most conservative value (among all substitutions) which satisfied the triangular inequality $d_s(a,b) < d_i(b) + d_e(a)$. In our experiments K had the value 1.3.

The three algorithms, Wagner & Fischer (WF), Lowrance & Wagner (LW) and our algorithm (SID_GT), were tested with the sets of 1026 noisy words, **SA** and **SB**. The results obtained in terms of accuracy and approximate computation times for the two sets are tabulated below in Tables IV and V. Note that our scheme far outperforms the traditional string correction algorithm (97.9 % instead of 77.2 %). It also outperforms the Lowrance and Wagner algorithm (97.9 % instead of 94.5 %). The reader should observe that in this case (as in all PR applications) it is much harder to increase the recognition accuracies at the higher end of the spectrum. Indeed, we believe that our algorithm is the best reported scheme to date when the errors encountered include SID and GTs.

Algorithm	Accuracy	Approximate computation time
Wagner & Fischer	79.92%	3 minutes 57 seconds
Lowrance & Wagner	94.83%	4 minutes 00 seconds
SID_GT	97.08%	4 minutes 30 seconds

Table IV : The results obtained from the set **SA**.

Algorithm	Accuracy	Approximate computation time
Wagner & Fischer	77.19%	3 minutes 57 seconds
Lowrance & Wagner	94.54%	4 minutes 00 seconds
SID_GT	97.86%	4 minutes 30 seconds

Table V : The results obtained from the set **SB**.

Our algorithm is marginally slower than Lowrance & Wagner's due to the following reasons. First of all, our edit distances (based on the inter-symbol confusion probabilities) are represented by a *matrix* for substitution weights, and by two linear arrays for insertions and deletions. As opposed to this, these are represented by four constant values for the Lowrance & Wagner's algorithm [14]. This inevitably increases the computational look-up time required by our algorithm. Another reason is that our algorithm, while looking for GTs searches for all possible transpositions of adjacent characters. As opposed to this, the transposition of adjacent characters in [14] is considered only for the case where the transposed characters remain unchanged.

V. CONCLUSIONS

In this paper we have studied the problem of recognizing a string Y which is the noisy version of some unknown string X^* chosen from a finite dictionary, **H**. We assume that the Y contains substitution, insertion and deletion (SID) errors and also generalized transposition (GT) errors in which the characters which are physically displaced by "straightforward transposition" are themselves subsequently substituted for. In this paper we present the first reported solution to the analytic problem of editing X to Y using these four edit operations. A scheme for obtaining the optimal edit operations has also been given. Both these solutions are optimal for the infinite alphabet case. Using these algorithms we present a syntactic PR strategy which corrects noisy text containing all these types of errors. The paper includes experimental results involving sub-dictionaries of the most common English words which demonstrate the superiority of our system over existing methods.

We are currently developing a character recognition scheme which uses these generalized edit operations and distances.

Acknowledgments : The authors are very grateful to Ms. Latika Khanna of the School of Computer Science at Carleton for her assistance in proofreading the manuscript. We are also very grateful to the referee who provided us with feedback which improved the quality of the paper.

REFERENCES

[1] A. V. Aho, D. S. Hirschberg, and J. D. Ullman, Bounds on the complexity of the longest common subsequence problem, *J. Assoc. Comput. Mach.*, 23:1-12 (1976).

[2] P. A. V. Hall and G. R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).

[3] D. S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. Assoc. Comput. Mach.*, 18:341-343 (1975).

[4] D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.*, 24:664-675 (1977).

[5] J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, 20:350-353 (1977).

[6] X. Huang, A lower bound for the edit distance problem under an arbitrary cost function, *Inf. Proc. Letters*, 27: 319-321 (1988).

[7] R. L. Kashyap and B. J. Oommen, An effective algorithm for string correction using generalized edit distances -I. Description of the algorithm and its optimality, *Inform. Sci.*, 23(2):123-142 (1981).

[8] R. L. Kashyap and B. J. Oommen, A common basis for similarity and dissimilarity measures involving two strings, *Internat. J. Comput. Math.*, 13:17-40 (1983).

[9] R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).

[10] R. L. Kashyap, and B. J. Oommen, String correction using probabilistic methods, *Pat. Recog. Letters*, 147-154 (1984).

[11] A. Marzal and E. Vidal, Computation of normalized edit distance and applications, *IEEE Trans. on Pat. Anal. and Mach. Intel.*, PAMI-15:926-932 (1993).

[12] B. J. Oommen and R. L. Kashyap, Symbolic Channel Modelling for Noisy Channels which Permit Arbitrary Noise Distributions, *Proc. of the 1993 Int. Symp. on Comp. and Inform. Sci.*, Istanbul, Turkey, November 1993, pp. 492-499.

[13] A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).

[14] R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.*, 22:177-183 (1975).

[15] W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).

[16] B. J. Oommen, Constrained string editing, *Inform. Sci.*, 40: 267-284 (1987).

[17] B. J. Oommen, Recognition of Noisy Subsequences Using Constrained Edit Distances, *IEEE Trans. on Pattern Anal. and Mach. Intel.*, PAMI-9:676-685 (1987).

[18] B. J. Oommen, "String Alignment With Substitution, Insertion, Deletion, Squashing and Expansion Operations", *Information Sciences*, Vol. 77: 89-107 (1995).

[19] T. Okuda, E. Tanaka, and T. Kasai, A method of correction of garbled words based on the Levenshtein metric, *IEEE Trans. Comput.*, C-25:172-177 (1976).

[20] J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Comm. Assoc. Comput. Mach.*, 23:676-687 (1980).

[21] D. Sankoff and J. B. Kruskal, *Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison*, Addison-Wesley (1983).

[22] E. Ukkonen, Algorithms for approximate string matching, *Inf. and Cont.*, 64: 100-118 (1985).

[23] R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).

Original words	Noisy words	WF	LW	OURS
according	ocrding	holding	according	according
account	awocnt	wouldnt	account	account
addition	vkdtion	addition	addition	addition
address	awrdss	address	address	address
advance	awqvce	advance	advance	advance
advantage	qvntahe	advantage	another	advantage
altogether	akvotbterdhoanuxker	average	another	altogether
anything	vtyhinh	nothing	nothing	anything
attention	ntntion	station	station	station
building	vlidinh	holding	holding	building
certain	cwtrin	certain	certain	certain
certainly	trainly	training	certainly	certainly
changed	cwnaed	engaged	changed	changed
character	raactqr	greater	character	character
company	cmapny	company	company	company
computers	cpmsuptejsr	chapter	computers	computers
condition	cpnsihtijno	condition	position	condition
conditions	cpnjkdtinohs	conditions	conditions	conditions
correcting	cprjnrctnihg	correcting	correcting	correcting
defense	dfnesw	defines	defines	defense
defines	dfniez	defines	defines	defines
electric	epejtcrihc	electric	electric	electric
electronic	epejtcroinhc	electric	electronic	electronic
evening	eeinnv	feeling	evening	evening
experts	epretz	express	experts	experts
express	epersz	experts	experts	express
expressed	ezpserssjde	experts	expressed	expressed
foreign	friegn	friends	foreign	foreign
friends	finedz	defines	friends	friends
greater	getaet	greater	greatest	greater
greatest	gtejtaesht	greatest	greatest	greatest
increase	incjnrashe	increase	increase	increase
increased	incserasjde	increase	increase	increased
largely	lregly	largely	largely	largely
largest	lregsy	already	largely	largest
learned	lanrex	largest	learned	learned
natural	ntruap	neutral	natural	natural
neutral	nurtap	neutral	neutral	neutral
official	ovfjfiiahl	official	official	official
officials	ovfsckiajsl	official	official	officials
permitted	pwrsimttjde	written	written	permitted
situation	sutsautijno	station	station	station
station	saiton	station	station	station
victory	vcotry	country	victory	victory
weather	wahtet	weather	weather	weather
whatever	wyajntvehr	whatever	weather	whatever
whether	wehtet	weather	weather	whether

Table III : A subset of the original words, their noisy versions and the words identified by the three algorithms.

APPENDIX

Proof of Theorem I.

Let $X_i = x_1...x_i$ and $Y_j = y_1...y_j$ and $D(X_i, Y_j)$ be the edit distance associated with the transforming X_i to Y_j with the edit operations of SID and GT. Then, we are to prove that :

$$D(X_i, Y_j) = \textbf{Min}\ [\ D(X_{i-1}, Y_j) + d_e(x_i),\quad D(X_i, Y_{j-1}) + d_i(y_j),\quad D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j),\quad D(X_{i-2}, Y_{j-2}) + d_t(x_{i-1}x_i, y_{j-1}y_j)\ ]. \qquad (A.1)$$

The proof of the theorem is by induction on the lengths of the strings X_i and Y_j.

The basis step of the proof involves the proofs of the Lemmas 0a-0c. These, of course, should be proved in the interest of mathematical rigor, but they can be proved by straightforward enumeration and their proofs are simple extensions of the results already found in the literature. Hence they are omitted here in the interest of brevity. We merely proceed to the inductive step.

Let Γ_{X_i,Y_j} be the set of all ways by which X_i can be edited into Y_j defined as in (A.1) for X_i and Y_j. Consider the distance $D(X_i, Y_j)$ which has the expression :

$$D(X_i, Y_j) = \min_{((X_i^{'},Y_j^{'})\ \Gamma_{X,Y})} \sum_{i=1}^{|J'|} [\ \text{Distances Associated with Operations in } (X_i^{'}, Y_j^{'})\ ], \quad (A.2)$$

where, $(X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}$ represents J' possible edit operations. Throughout this proof[4], we shall assume that the arbitrary element $(X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}$ is of length L and is of the form given as :

$X_i = x_{i1}^{'}x_{i2}...x_{iL}^{'}$, and, $Y_j = y_{j1}^{'}y_{j2}^{'}...y_{jL}^{'}$.

We partition the set Γ_{X_i,Y_j} into nine mutually exclusive and exhaustive subsets:

$$\Gamma^1_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = \theta,\ x_{iL}^{'} = \theta,\ y_{jL-1}^{'} = y_{j-1},\ y_{jL}^{'} = y_j\ \}$$

$$\Gamma^2_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = \theta,\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = y_j,\ y_{jL}^{'} = \theta\ \}$$

$$\Gamma^3_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = \theta,\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = y_{j-1},\ y_{jL}^{'} = y_j\ \}$$

$$\Gamma^4_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_i,\ x_{iL}^{'} = \theta,\ y_{jL-1}^{'} = \theta,\ y_{jL}^{'} = y_j\ \}$$

$$\Gamma^5_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_i,\ x_{iL}^{'} = \theta,\ y_{jL-1}^{'} = y_{j-1},\ y_{jL}^{'} = y_j\ \}$$

$$\Gamma^6_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_{i-1},\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = \theta,\ y_{jL}^{'} = \theta\ \}$$

$$\Gamma^7_{X_i,Y_j} = \{\ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'})\ \ \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_{i-1},\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = \theta,\ y_{jL}^{'} = y_j\ \}$$

[4]This notation is not religiously correct. Indeed, the length of the arbitrary element should be $L(X_i^{'}, Y_j^{'})$. But this will make an already tedious notation even more cumbersome. We request the reader to permit us this breach in notation with the understanding that he remembers that L is dependent on the element itself.

$$\Gamma^{8}_{X_i,Y_j} = \{ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'}) \quad \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_{i-1},\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = y_j,\ y_{jL}^{'} = \theta \}$$

$$\Gamma^{9}_{X_i,Y_j} = \{ (X_i^{'}, Y_j^{'}) \mid (X_i^{'}, Y_j^{'}) \quad \Gamma_{X_i,Y_j}, \text{ with } x_{iL-1}^{'} = x_{i-1},\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = y_{j-1},\ y_{jL}^{'} = y_j \}$$

By their definitions, we see that the above nine sets are mutually exclusive. Further, since the corresponding elements of $(X_i^{'}, Y_j^{'})$ cannot be θ simultaneously, it is clear that every pair in Γ_{X_i,Y_j} must be in one of the above sets. Hence these nine sets partition Γ_{X_i,Y_j}. Rewriting (A.2) we obtain :

$$D(X_i, Y_j) = \min_{1 \le k \le 9} \min_{((X_i^{'},Y_j^{'}) \in \Gamma^{k}_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_i^{'}, Y_j^{'})], \tag{A.3}$$

where, $(X_i^{'}, Y_j^{'})$ represents J' possible edit operations.

We shall now consider each of the nine terms in (A.3) individually.

Case 1.

Consider the first term in (A.3). In every pair in $\Gamma^{1}_{X_i,Y_j}$ we know that the last two elements of each string in the pair are :

$$x_{iL-1}^{'} = \theta,\ x_{iL}^{'} = \theta,\ y_{jL-1}^{'} = y_{j-1},\ y_{jL}^{'} = y_j.$$

Hence,

$$\min_{((X_i^{'},Y_j^{'}) \in \Gamma^{1}_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_i^{'}, Y_j^{'})]$$

$$= \min_{((X_i^{'},Y_j^{'}) \in \Gamma^{1}_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_{iL-1}^{'}, Y_{jL-1}^{'})] + d_i(y_{jL}^{'}). \tag{A.4}$$

For every element in $\Gamma^{1}_{X_i,Y_j}$ there is a unique element in $\Gamma_{X_i,Y_{j-1}}$ and vice versa, where $\Gamma_{X_i,Y_{j-1}}$ is the set of all ways by which X_i can be edited into Y_{j-1} defined as in (A.2) for X_i and Y_{j-1}. This unique element is obtained by merely reducing the length of the strings $X_i^{'}$ and $Y_j^{'}$ by unity. By the inductive hypothesis the first term in (A.4) is exactly $D(X_i, Y_{j-1})$. Since $y_{jL}^{'} = y_j$, (A.4) simplifies to :

$$D(X_i, Y_{j-1}) + d_i(y_j). \tag{A.5}$$

Case 2.

Consider the second term in (A.3). In every pair in $\Gamma^{2}_{X_i,Y_j}$ we know that the last two elements of each string in the pair are :

$$x_{iL-1}^{'} = \theta,\ x_{iL}^{'} = x_i,\ y_{jL-1}^{'} = y_j,\ y_{jL}^{'} = \theta.$$

Hence,

$$\min_{((X_i',Y_j')\ \Gamma^2_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_i', Y_j')]$$

$$= \min_{((X_i',Y_j')\ \Gamma^2_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_{iL-1}', Y_{jL-1}')] + d_e(x_{iL}'). \qquad (A.6)$$

For every element in $\Gamma^2_{X_i,Y_j}$ there is a unique element in Γ_{X_{i-1},Y_j} and vice versa, where Γ_{X_{i-1},Y_j} is the set of all ways by which X_{i-1} can be edited into Y_j defined as in (A.2) for X_{i-1} and Y_j, and this unique element is again obtained by merely reducing the length of the strings X_i' and Y_j' by unity. Again, by the inductive hypothesis, the first term in (A.6) is $D(X_{i-1}, Y_j)$. Since $x_{iL}' = x_i$, (A.6) simplifies to :

$$D(X_{i-1}, Y_j) + d_e(x_i). \qquad (A.7)$$

Case 3.

Consider the third term in (A.3). From its definition, we know that in every pair in $\Gamma^3_{X_i,Y_j}$ the last two elements of each string in the pair are :

$$x_{iL-1}' = \theta,\ x_{iL}' = x_i,\ y_{jL-1}' = y_{j-1},\ \ y_{jL}' = y_j.$$

Hence we are to compute :

$$\min_{((X_i',Y_j')\ \Gamma^3_{X_i,Y_j})} \sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_i', Y_j')] \qquad (A.8)$$

$$\min_{((X_i',Y_j')\ \Gamma^3_{X_i,Y_j})} \Big[\sum_{i=1}^{|J'|} [\text{Distances Associated with Operations in } (X_{iL-2}', Y_{jL-2}')] + d_i(y_{jL-1}') + d_s(x_{iL}', y_{jL}') \Big]. \qquad (A.9)$$

Since the inductive hypothesis is assumed true for the prefixes of X_i and Y_j the first and second terms in the minimization can be coalesced in the computation. In this case it can be seen that for every element in $\Gamma^3_{X_i,Y_j}$ there is a unique element in $\Gamma_{X_{i-1},Y_{j-1}}$ and vice versa, where $\Gamma_{X_{i-1},Y_{j-1}}$ is the set of all ways by which X_{i-1} can be edited into Y_{j-1}. This unique element is obtained by merely reducing the lengths of the strings X_i' and Y_j' by unity. Hence, the coalescing of the first two terms of (A.9) yields $D(X_{i-1}, Y_j)$. Since $x_{iL}' = x_i$, (A.9) simplifies to :

$$D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j). \qquad (A.10)$$

Similar arguments can be given for each of the remaining cases. The ideas are almost identical, because, in each case we consider the term to be minimized, analyze how the inductive hypothesis can be utilized, and compute the term obtained on utilizing the consequences of the latter. The additional terms that appear thereafter

are then written down based on the individual edit operations that are represented. In the interest of brevity, the details of the remaining cases are omitted and the final results in each of the cases is written down.

Case 4.

Consider the fourth term in (A.3). In every pair in $\Gamma^{4}_{X_i,Y_j}$ the last two symbols of each string in the pair are

$$x'_{iL-1} = x_i,\ x'_{iL} = \theta,\ y'_{jL-1} = \theta,\ y'_{jL} = y_j.$$

Thus we are to evaluate :

$$\min_{((X'_i,Y'_j)\ {}^{4}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X'_i, Y'_j)\right]$$

$$= D(X_i, Y_{j-1}) + d_i(y'_{jL}), \qquad = D(X_i, Y_{j-1}) + d_i(y_j). \tag{A.11}$$

Case 5.

Consider the fifth term in (A.3). In every pair in $\Gamma^{5}_{X_i,Y_j}$ we know that the last two elements of each string in the pair are

$$x'_{iL-1} = x_i,\ x'_{iL} = \theta,\ y'_{jL-1} = y_{j-1},\ y'_{jL} = y_j.$$

Thus we are to evaluate :

$$\min_{((X'_i,Y'_j)\ {}^{5}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X'_i, Y'_j)\right]$$

$$= D(X_i, Y_{j-1}) + d_i(y'_{jL}), \qquad = D(X_i, Y_{j-1}) + d_i(y_j). \tag{A.12}$$

Case 6.

Consider the sixth term in (A.3). In every pair in $\Gamma^{6}_{X_i,Y_j}$ we know that the last two elements of each string in the pair are

$$x'_{iL-1} = x_{i-1},\ x'_{iL} = x_i,\ y'_{jL-1} = \theta,\ y'_{jL} = \theta.$$

Thus we are to evaluate :

$$\min_{((X'_i,Y'_j)\ {}^{6}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X'_i, Y'_j)\right]$$

Arguing as in the case of the first term we obtain that this leads to the the quantity :

$$D(X_{i-1}, Y_j) + d_e(x'_{iL}) = D(X_{i-1}, Y_j) + d_e(x_i). \tag{A.13}$$

Case 7.

Consider the seventh term in (A.3). In every pair in $\Gamma^{7}_{X_i,Y_j}$ we know that the last two elements of each string in the pair are

$$x'_{iL-1} = x_{i-1},\ x'_{iL} = x_i,\ y'_{jL-1} = \theta,\ y'_{jL} = y_j.$$

Thus we are to evaluate :

$$\min_{((X_i',Y_j')\ {}^{7}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X_i', Y_j')\right]$$

Arguing as in the case of the first term we obtain that this leads to the the quantity :

$$D(X_{i-1}, Y_{j-1}) + d_s(x_{iL}', y_{jL}') = D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j). \qquad (A.14)$$

Case 8.

Consider the eighth term in (3). In every pair in ${}^{8}\Gamma_{X_i,Y_j}$ we know that the last two elements of each string in the pair are

$$x_{iL-1}' = x_{i-1},\ x_{iL}' = x_i,\ y_{jL-1}' = y_j,\ y_{jL}' = \theta.$$

Thus we are to evaluate :

$$\min_{((X_i',Y_j')\ {}^{8}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X_i', Y_j')\right]$$

Arguing as in the case of the first term we obtain that this leads to the the quantity :

$$D(X_{i-1}, Y_j) + d_e(x_{iL}') = D(X_{i-1}, Y_j) + d_e(x_i) \qquad (A.15)$$

Case 9.

Finally, consider the ninth term in (A.3). In every pair in ${}^{9}\Gamma_{X_i,Y_j}$ we know that the last two elements of each string in the pair are

$$x_{iL-1}' = x_{i-1},\ x_{iL}' = x_i,\ y_{jL-1}' = y_{j-1},\ y_{jL}' = y_j.$$

Thus we are to evaluate :

$$\min_{((X_i',Y_j')\ {}^{9}\Gamma_{X_i,Y_j})} \sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X_i', Y_j')\right]$$

In this case the inductive hypothesis leads to two distinct possibilities for the minimization to be achieved because the growing of (X_{iL-2}', Y_{jL-2}') to (X_i', Y_j') represents two unique sequences of edit operations given as Case 9.1 and 9.2 respectively.

Case 9.1 In this case the minimum is obtained by arguing as in Case 1 to yield :

$$D(X_{i-1}, Y_{j-1}) + d_s(x_i, y_j). \qquad (A.16)$$

Case 9.2 In this case the minimum is obtained as :

$$\min_{((X_i',Y_j')\ {}^{9}\Gamma_{X_i,Y_j})} \left[\sum_{i=1}^{|J'|} \left[\text{Distances Associated with Operations in } (X_{iL-2}', Y_{jL-2}')\right] + d_t(x_{iL-1}' x_{iL}, y_{jL-1}' y_{jL}') \right]. \qquad (A.17)$$

For every element in $\Gamma^{9}_{X_i,Y_j}$ there is a unique element in $\Gamma_{X_{i-2},Y_{j-2}}$ and vice versa, where $\Gamma_{X_{i-2},Y_{j-2}}$ is the set of all ways by which X_{i-2} can be edited into Y_{j-2}. This unique element is obtained by reducing the length of the strings $X_i^{'}$ and $Y_j^{'}$ by two respectively. But by the inductive hypothesis the latter is exactly $D(X_{i-2}, Y_{j-2})$. Since $x_{iL-1} = x_{i-1}$, $x_{iL}^{'} = x_i$, and $y_{jL-1}^{'} = y_{j-1}$, $y_{jL}^{'} = y_j$, (A.17) yields :

$D(X_{i-2}, Y_{j-2}) + d_t(x_{i-1}x_i, y_{j-1}y_j)$. (A.18)

Combining (A.5), (A.7), (A.10), (A.11-A.18) the theorem is proved. $\rightarrow\rightarrow\rightarrow$

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t
a	867	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1	20	1
b	1	861	1	1	1	1	14	14	1	1	1	1	1	20	1	1	1	1	1	1
c	1	1	861	14	1	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1
d	1	1	14	835	14	20	1	1	1	1	1	1	1	1	1	1	1	10	20	1
e	1	1	1	14	857	5	1	1	1	1	1	1	1	1	1	1	1	20	14	1
f	1	1	14	20	10	824	20	1	1	1	1	1	1	1	1	1	1	20	1	10
g	1	14	1	1	1	20	835	20	1	1	1	1	1	1	1	1	1	5	1	14
h	1	14	1	1	1	1	20	835	1	20	1	1	1	14	1	1	1	1	1	5
i	1	1	1	1	1	1	1	1	861	10	14	5	1	1	20	1	1	1	1	1
j	1	1	1	1	1	1	1	20	10	835	20	1	14	14	1	1	1	1	1	1
k	1	1	1	1	1	1	1	1	14	20	848	20	14	1	10	1	1	1	1	1
l	1	1	1	1	1	1	1	1	5	1	20	876	1	1	14	14	1	1	1	1
m	1	1	1	1	1	1	1	5	1	14	14	1	876	20	1	1	1	1	1	1
n	1	20	1	1	1	1	5	14	1	14	1	1	20	857	1	1	1	1	1	1
o	1	1	1	1	1	1	1	1	20	1	14	14	1	1	861	20	1	1	1	1
p	1	1	1	1	1	1	1	1	1	1	1	14	1	1	20	893	1	1	1	1
q	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	892	1	5	1
r	1	1	1	14	17	14	5	1	1	1	1	1	1	1	1	1	1	863	1	17
s	17	1	1	17	10	1	1	1	1	1	1	1	1	1	1	1	5	1	841	1
t	1	1	1	1	1	14	14	10	1	1	1	1	1	1	1	1	1	17	1	858
u	1	1	1	1	1	1	1	14	17	14	10	1	1	1	1	1	1	1	1	1
v	1	17	17	5	1	14	14	1	1	1	1	1	1	1	1	1	1	1	1	1
w	10	1	1	5	17	1	1	1	1	1	1	1	1	1	1	1	17	1	14	1
x	5	1	17	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1
y	1	1	1	1	1	1	14	14	1	5	1	1	1	1	1	1	1	1	1	17
z	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	1

Table I: The "confusion matrix" with the probabilities of substituting a character with another character. The figures in the table are to be multiplied by a factor of 10^{-3}.