PATTERN RECOGNITION OF STRINGS CONTAINING TRADITIONAL AND GENERALIZED TRANSPOSITION ERRORS[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] [LW75] 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 subdictionaries of the most common English words which demonstrate the superiority of our system over existing methods.

1. 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 [MV93]. Inexact sequence comparison is also used extensively in the comparison of biological macromolecules 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 [HD80,Pe80] list hundreds of publications that tackle the problem from various perspectives.

Damareau [See references in Oo87] 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 [Le66], 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.* [OTK76] extended this concept by weighting the edit operations, and many other researchers among whom are Wagner and Fisher [WF74] 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 [OTK76,SK83]. Wagner and Fischer [WF74] 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 the algorithm are

[1]Partially supported by the Natural Sciences and Engineering Research Council of Canada.

2There 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.

available in the literature, a review of which can be found in [SK83]. Masek and Paterson [MP80] 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 (Hirschberg [Hi77], Hunt and Szymanski [HS77] and others [SK83]). The complexity of the LCS problem has been studied by Aho *et. al.* [AHU76]. String correction using the GLD as a criterion has been done for strings [Pe80,SK83], substrings [KO83], dictionaries treated as generalized tries [KO83] and for grammars [SK83]. Besides these deterministic techniques, various probabilistic methods have been studied in the literature [Pe80,KO84].

Although some work has been done to extend the traditional set of SID operations to include the transposition of adjacent characters [LW75,SK83] the problem is unsolved for "Generalized" Transposition (GT) errors. 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 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 optimal and quadratic.

The only reported result for traditional transpositions is the the one proposed by Lowrance and Wagner [LW75,SK83]. The difference between our algorithm and the scheme presented by Lowrance and Wagner for traditional transpositions is given in the unabridged paper [OL94].

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 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 [MV93] and biological macromolecules. To see the latter, 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 macromolecules [SK83] where the mutation (substitution) of transposed molecules occurs in the "next" generation after the protein sequences are transposed.

1.1 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 $\tilde{\mathbf{A}} = \mathbf{A} \approx \{\theta\}$. $\tilde{\mathbf{A}}$ 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 $\tilde{\mathbf{A}}^*$, the set of strings over $\tilde{\mathbf{A}}$. The *Compression Operator* C is a mapping from $\tilde{\mathbf{A}}^*$ 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 $Z' = f\theta o\theta r$, $C(Z') = for$.

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 $\mathbf{A} \times \mathbf{A} \varnothing \mathbf{R}^+$ and is called the Substitution Map. In particular, $d_s(a,b)$ is the distance associated with substituting b for a, a,b $\mathbf{A}$. For all a $\mathbf{A}$, $d_s(a,a)$ is generally assigned the value zero, although this is not mandatory.

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

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

(iv) $d_t(.,.)$ is a map from $\mathbf{A}^2 \times \mathbf{A}^2 \varnothing \mathbf{R}^+$ 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.

1.2 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 $\tilde{\mathbf{A}}^* \times \tilde{\mathbf{A}}^*$ as :

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

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

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

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

(1)

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

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 $1 \le i \le |X'|$ by (x_i', y_i'), which represents the transformation of x_i' to y_i'. The cases below consider the SID operations :

(i) If $x_i' \in \mathbf{A}$ and $y_i' \in \mathbf{A}$, it represents the substitution of y_i' for x_i'.

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

(iii) If $x_i' = \theta$ and $y_i' \in \mathbf{A}$, 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 grown 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 $\mathbf{A}$ to itself is also considered as an operation in the arbitrary pair $(X',Y') \in \Gamma_{X,Y}$. 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 [KO81,KO83,KO84,Oo87,Oo93].

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 :

$$D(X,Y) = \min_{((X',Y') \in \Gamma_{X,Y})} \sum_{i=1}^{|J'|} \left[\xi(X',Y') \right],$$

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

$\xi(X',Y')$ = distances associated with the operations in (X', Y').

2. 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 $D(\mu, \mu)$ is zero.

LEMMA 0a.

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

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

$\rightarrow\rightarrow\rightarrow$

LEMMA 0b.

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

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

$\rightarrow\rightarrow\rightarrow$

LEMMA 0c.

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

$$D(X, Y) = \textbf{Min} \left[\begin{array}{l} D(\mu, Y) + d_e(x_1), \\ D(X, \mu) + d_i(y_1), \\ d_s(x_1, y_1) \end{array} \right]$$

$\rightarrow\rightarrow\rightarrow$

We shall now state 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 edit operations of substitution, insertion, deletion and generalized transposition. Then, the following is true :

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

The proof, which differs from proofs traditionally used in the literature can be found in the unabridged paper [OL94].

$\rightarrow\rightarrow\rightarrow$

3. 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 ds(.,.), di(.), de(.), dt(.,.).

Output: The distance D(X, Y) associated with transforming X to Y using the four edit operations of substitution, insertion, deletion and transposition

Method :

Z(0, 0) ♦ 0
For i ♦ 1 to N Do
  Z(i, 0) ♦ Z(i-1, 0) + de(xi)
For j ♦ 1 to M Do
  Z(0, j) ♦ Z(0, j-1) + di(yj)

For i ♦ 1 to N Do

  Z(i, 1) ♦ 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 ♦ 2 to M Do
  Z(1, j) ♦ 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 ♦ 2 to N do
  For j ♦ 2 to M do
  Z(i,j) ♦ Min [Z(i-1, j) + $d_e(x_i)$,
    Z(i, j-1) + $d_i(y_j)$,
    Z(i-1, j-1) + $d_s(x_i,y_j)$,
    Z(i-2, j-2) + $d_t(x_{i-1}x_i , y_{j-1}y_j)$]

D(X,Y) ♦ 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 [AHU76,WC76]. In this case, the number of symbol comparisons is quadratic. In the interior of the main loop, we will need at most four additions and the computation of the minimum of a fixed (at most four) quantities.

2. The lower bound result claimed in [Hu88] 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 [Hu88] 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.

3. The difference between our algorithm and the scheme presented by Lowrance and Wagner for traditional transpositions [LW75] is also given in the unabridged paper [OL94].

4. As mentioned earlier, the underlying graph that has to be traversed is called a *trellis*. This trellis is two dimensional in this case. Even though the set of edit operations has been expanded the

fundamental properties of the underlying graph traversed remains the same. Here, the graph G is :

G = (V, E), where, V and E are :

$V = \{ \langle i, j \rangle \mid 0 \quad i \quad N, \ 0 \quad i \quad M \quad \}$, and,

$E = \{ (\langle i, j \rangle, \langle i+1, j \rangle) \mid 0 \ i \ N-1, \ 0 \ j \ M \quad \}$

$\approx \{ (\langle i, j \rangle, \langle i, j+1 \rangle) \mid 0 \ i \ N, \ 0 \ j \ M-1 \quad \}$

$\approx \{ (\langle i, j \rangle, \langle i+1, j+1 \rangle) \mid 0 \ i \ N-1, \ 0 \ j \ M-1 \quad \}$

$\approx \{ (\langle i, j \rangle, \langle i+2, j+2 \rangle) \mid 0 \ i \ N-2, \ 0 \ j \ M-2 \quad \}.$

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. The pictorial representation of the graph and an example of the traversal is given in [OL94].

5. Just as in all the edit processes studied in the literature, the trellis can also be used to compute the best edit sequence to yields the optimal edit distance. This is done by backtracking through the trellis from the array element (N,M) in the reverse direction so as to reach the origin. The details of this are found in the unabridged paper [OL94].

4. 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 which could have been obtained if

(i) only SID errors were assumed as in the case of the Wagner & Fischer [WF74] algorithm and,

(ii) SID and traditional transposition errors were assumed as in the case of the Lowrance and Wagner [LW75] algorithm.

The dictionary consisted of 342 words obtained as a subset of the 1023 most common English words [KO81,KO83,KO93] 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 words were generated using the method described in [KO93] (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. A subset of some of the words in SA is given Table III.

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 individual inter-symbol edit distances were computed using negative logarithms of the symbol confusion accuracies as explained in [SK83,KO87]. The details of the confusion matrices and the distance assignments used are given in the unabridged paper [OL94]. The results obtained in terms of accuracy and approximate computation times for the two sets are tabulated below. 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 I : 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 II : 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 a fixed number of the constant values for the Lowrance & Wagner's algorithm [LW75]. 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 [LW75] is considered only for the case where the transposed characters remain unchanged.

5. 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 transposition errors . Although some work has been done to extend the traditional set of edit operations to include the straightforward transposition of adjacent characters [LW75], 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 subdictionaries of the most common English words which demonstrate the superiority of our system over existing methods.

REFERENCES

[AHU76] 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).

[HD80] P. A. V. Hall and G. R. Dowling, Approximate string matching, *Comput. Surveys*, 12:381-402 (1980).

[Hi77] D. S. Hirschberg, Algorithms for longest common subsequence problem, *J. Assoc. Comput. Mach.*, 24:664-675 (1977).

[Hu88] X. Huang, A lower bound for the edit distance problem under an arbitrary cost function, *Inf. Proc. Letters*, 27: 319-321 (1988).

[HS77] J. W. Hunt and T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Comm. Assoc. Comput. Mach.*, 20:350-353 (1977).

[KO81] 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).

[KO83] R. L. Kashyap and B. J. Oommen, The noisy substring matching problem, *IEEE Trans. Software Engg.*, SE-9:365-370 (1983).

[KO84] R. L. Kashyap, and B. J. Oommen, String correction using probabilistic methods, *Pat. Recog. Letters*, 147-154 (1984).

[MV93] 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).

[KO93] 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.

[Le66] A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Phys. Dokl.*, 10:707-710 (1966).

[LW75] R. Lowrance and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.*, 22:177-183 (1975).

[MP80] W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.*, 20:18-31 (1980).

[Oo87] B. J. Oommen, Recognition of noisy subsequences using constrained edit distances, *IEEE Trans. on Pat. Anal. and Mach. Intel.*, PAMI-9:676-685 (1987).

[Oo93] B. J. Oommen, "String alignment with substitution, insertion, deletion, squashing and expansion operations". To appear in *Inf. Sci.*.

[OL94] B. J. Oommen and R. K. S. Loke, Pattern recognition of strings with substitutions, insertions, deletions and generalized transpositions. Unabridged Paper. Will be soon available as a Carleton University technical report.

[OTK76] 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).

[Pe80] J. L. Peterson, Computer programs for detecting and correcting spelling errors, *Comm. Assoc. Comput. Mach.*, 23:676-687 (1980).

[SK83] D. Sankoff and J. B. Kruskal, Time Warps,String Edits and Macromolecules: The Theory and practice of Sequence Comparison, Addison-Wesley (1983).

[Uk85] E. Ukkonen, Algorithms for approximate string matching, *Inf. and Cont.*, 64: 100-118 (1985).

[WF74] R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.*, 21:168-173 (1974).

[WC76] C. K. Wong and A. K. Chandra, Bounds for the string editing problem, *J. Assoc. Comput. Mach.*, 23:13-16 (1976).

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.

Original Words	Noisy words	Algorithm WF	Algorithm LW	Algorithm SID–GT
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
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
express	epersz	experts	experts	express
friends	finedz	defines	friends	friends

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