The Noisy Substring Matching Problem

R.L. Kashyap* and B.J. Oommen**

SCS-TR-16
January 1983

* Department of Electrical Engineering, Purdue University, W. Lafayette, IN, 47907, USA.

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

Accepted for publication by the IEEE Transactions on Software Engineering. Details of the issue of publication are not known.

THE NOISY SUBSTRING MATCHING PROBLEM[†]

R. L. Kashyap and B. J. Oommen

ABSTRACT

Let T(U) be the set of words in the dictionary H which contains U as a substring. The problem considered here is the estimation of the set T(U) when U is not known, but Y, a noisy version of U is available. The suggested set estimate $S^*(Y)$ of T(U) is a proper subset of H such that its every element contains at least one substring which resembles Y most according to the Levenshtein metric. The proposed algorithm for the computation of $S^*(Y)$ requires cubic time. The algorithm uses the recursively computable dissimilarity measure $D^k(X,Y)$, termed as the k-th distance between two strings X and Y which is a dissimilarity measure between Y and a certain subset of the set of contiguous susbtrings of X. Another estimate of T(U), namely $S^M(Y)$ is also suggested. The accuracy of $S^M(Y)$ is only slightly less than that of $S^*(Y)$, but the computation time of $S^M(Y)$ is substantially less than that of $S^*(Y)$. Experimental results involving 1,900 noisy substrings and dictionaries which are subsets of 1,023 most common English words [11] indicate that the accuracy of the estimate $S^*(Y)$ is around 99 percent and that of $S^M(Y)$ is about 98 percent.

Index Terms: Error correction in strings, noisy substring matching, string dissimilarity in terms of the dissimilarity of their substrings, string set estimation, text editing, Levenshtein metric.

[†] Partially supported by the National Science Foundation under grant ECS-80-09041.

I. INTRODUCTION

A common problem in text editing is that of finding the occurence of a given substring in a file. This is usually done to locate one's bearings in the file or to replace the occurence 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 occurence of a certain substring, U, is sought for, the exact substring matching problem aims to obtain the *set* of all the words in the dictionary which contains U as a contiguous substring. Many algorithms [1,2] 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 occurence of the substring sought for [3].

Let T(U) be the *set* of words in the dictionary which contains U. In this paper, we consider the problem of estimating the *set* T(U) when U is not known, but Y, the noisy version of U is available. Substitution, insertion and deletion errors transform U into Y.

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 a contiguous substring of the keyword. In such a case the program that solves the noisy substring matching problem can be used to process a noisy (incorrectly spelled) version of a *fragment* of a keyword, and locate a record characterized by the entire keyword.

Ideally, the set estimate S(Y) should include the set T(U). In that case, given additional information in the form of (noisy) prefixes or suffixes of U, we can identify the correct word in the dictionary, by searching the relatively small set S(Y). Hence we measure the quality of the estimate S(Y) by the frequency of S(Y) containing T(U). Our goal is to obtain a set estimate S(Y) which has a high value of that frequency, but is not a very large subset of the dictionary. The solution S(Y) which equals the entire dictionary is trivial and useless.

The proposed estimate $S^*(Y)$ is computed by comparing the noisy string Y with all the contiguous substrings of the words in the dictionary using an appropriate measure of dissimilarity between two strings. Many measures can be used for this purpose such as the Generalized Levenshtein Distance (GLD) between them, the length of their

Shortest Common Supersequence, and a monotonically decreasing function of the length of their Longest Common Subsequence [4-7]. Throughout this paper the Generalized Levenshtein Distance (GLD) D(V,Y) is used as the measure to quantify the measure the dissimilarity between the strings V and Y. The results obtained here can easily be extended for any of the other dissimilarity measures referred to above [6].

Let H^* be the set of all the substrings of the words in the dictionary. Let R be a subset of H^* whose members are most similar to the given string Y according to the GLD. The required set estimate $S^*(Y)$ is the subset of the dictionary, each member containing at least one member of R.

To compute the set $S^*(Y)$, we define a dissimilarity measure $D^k(X,Y)$ between the strings X and Y as the minimum value of the GLD between Y and the individual elements of a certain subset of the contiguous substrings of X. The measure $D^k(X,Y)$ has the property that it can be recursively computed. The solution to the noisy substring matching problem is exactly the set of words in the dictionary which minimizes $D^{|X|}(X,Y)$, where X is any arbitary word in the dictionary. The measure $D^{|X|}(X,Y)$, can be computed in time proportional to $|Y|.|X|^2$. Hence the set estimate $S^*(Y)$ can be computed in cubic time.

In Section II we formulate the problem explicitly, and present $S^*(Y)$, the proposed solution. Its computation is discussed in Section III. In this section, we also suggest an approximation of $S^*(Y)$ as an alternative solution to the problem. This approximation, $S^M(Y)$, is almost as accurate as $S^*(Y)$ but is more easily computable. We conclude the paper with some experimental results obtained using dictionaries which are subsets of the 1023 most common English words [11].

I.1 Notation

Upper case symbols from the latter part of the English alphabet such as U,V, ...,Z, will always refer to strings. Lower case symbols, subscripted or otherwise, are used to represent letters of the alphabet under consideration. Upper and lower case letters of the English alphabet from I to N will be used as integers. We reserve the symbols A, B, H, Q, R, S and T to represent sets of strings. Otherwise, the symbols used in the paper are self-explanatory.

II. FORMULATION OF THE PROBLEM AND PROPOSED SOLUTION

Let A be any finite alphabet, and H a finite dictionary which is a subset of A^*, the set of strings over A. Let H^* be the set of all the non-null contiguous substrings of the words in H. Let U be any non-null string in A^*. We seek the words in H which contain U. The solution to the exact substring matching problem is the *set* T(U) defined by (2.1).

$$T(U) = \{X \mid X \in H,\ U \text{ is a substring of } X\}, \qquad \text{if } U \in H^*$$

$= \phi$, the null set, otherwise, (2.1)

If Y is a noisy version of U, the aim of the noisy substring matching problem is to obtain a non-trivial estimate of the *set* T(U) by processing only Y.

Before giving the solution we need the concept of the generalized Levenshtein distance (GLD) D(Z,Y) between two strings Z and Y of arbitrary lengths [4,5,9]. The GLD measures the dissimilarity between Z and Y in terms of the edit operations on the individual symbols of Y needed to transform it into Z. The edit operations are substitution, deletion and insertion. Let d(b,c) be the weight associated with replacing the letter 'b' in Y by the letter 'c'. Similarly $d(b,\lambda)$, $d(\lambda,b)$ are the weights associated with the erasure of 'b' in Y and insertion of 'b' into Y respectively.

$$d(c,c) = 0,\ 0 < d(b,c) < \infty,\ \forall\, c \neq b;\ 0 < d(\lambda,b),\ d(b,\lambda) < \infty,\ \forall\, b$$

Then the GLD D(Z,Y) is the minimum of the sum of the weights associated with the various edit operations on Y to transform it into Z. Typically d(c/b) is chosen to reflect the frequency of the noisy mechanism to convert 'b' into 'c' [5,10]. The simplest choice of the weights is: $d(b,c) = 1$ only if $c \neq b$, $d(\lambda,b) = d(b,\lambda) = 1$. This choice will be termed as the zero-one Levenshtein distance.

The solution to the problem posed earlier is obtained by comparing Y with all the elements of H^*, the set of all the contiguous substrings of H. A subset of the latter, R(Y), can be extracted which is the set of contiguous substrings most similar to Y according to the generalized Levenshtein distance. The estimate $S^*(Y)$ of T(U) is obtained as a union of the words in H which have at least one of the substrings most resembling Y. Explicitly,

$$R(Y) = \{ Z \mid Z \epsilon H^*,\ D(Z,Y) \leq D(Z',Y)\ \forall Z' \epsilon H^* \} \tag{2.2}$$

$$S^*(Y) = \bigcup_{V \epsilon R(Y)} T(V) \tag{2.3}$$

The example below will help clarify these sets.

Example I

Let H = { construction, attention, attending, opinion } and Y=sion. The dissimilarity measure is the Levenshtein metric with zero-one weights. Then,

R(Y) = { tion, nion }

S(Y) = { construction, attention, opinion }

***

Theorem I

Let Y be any noisy version of a string $U \epsilon H^*$. The set estimate of T(U) given by $S^*(Y)$ has the following properties:

(i) If Y=U, $S^*(Y) = T(U)$.

(ii) T(U) is a subset of $S^*(Y)$ if U or any one of its contiguous substrings is in R(Y).

***

The proof of the theorem follows directly from the properties of the GLD between two strings [4-8], and the definition of $S^*(Y)$.

Remarks

(1) From (2.2) it can be seen that if Y=U or $Y \epsilon H^*$, $S^*(Y)$, the solution to the noisy substring matching problem, is identically equal to the solution to the exact substring matching problem.

(2) In some applications, it may be desirable to have $S^*(Y)$ identical to the set T(U) as frequently as possible. This is equivalent to maximizing the frequency of obtaining R(Y) as the *singleton* set {U}. The latter can be achieved by defining the GLD in terms of intersymbol edit distances which are *real* non-negative numbers, and which are functions of the probabilities of the mechanism which noisily transforms U into Y [4-6, 10].

(3) As mentioned earlier, the quality of the estimate S(Y) can be measured by the frequency of T(U) being a subset of S(Y), given that Y was generated from U. An analytical expression for this frequency is very difficult to obtain, even after making appropriate assumptions regarding the noisy mechanism which generates Y from U. The experimental results of section (iv) indicate that for $S^*(Y)$, this frequency is relatively high.

III COMPUTATION OF $S^*(Y)$ AND ITS APPROXIMATION

The computation of $S^*(Y)$ from its definition is far too cumbersome, since the set H^* is usually *many orders* larger than the set H. We propose a computational scheme which computes $S^*(Y)$ by sequentially comparing Y with the *individual elements of H.* We first introduce the concept of the k-th distance between two strings, $D^k(X,Y)$. This quantity is defined in terms of the GLD between Y and V, where V is an element of a subset of the contiguous substrings of X. The k-th distance between X and Y can be recursively computed, and further if $|X|=N$, the N-th distance is exactly the minimum value of GLD between Y and all the contiguous substrings of X. By computing $D^{|X|}(X,Y)$ for every $X \epsilon H$, the set $S^*(Y)$ can be obtained.

III.1 The k-th Distance $D^k(X,Y)$

Let $X=x_1...x_N$ be any string of length N, where each $x_i \epsilon A$. We define $Q^k(X)$ as the set of all the prefixes of every suffix of X which is of length greater than or equal to N-k+ 1. The k-th distance $D^k(X,Y)$ is defined as the minimum of the GLD between V and Y for all $V \epsilon Q^k(X)$.

$$Q^k(X) = \bigcup_{i=1}^{k} \bigcup_{j=i}^{|X|} \{ X_{i,j} \} \tag{3.1}$$

where, $X_{i,j} = x_i x_{i+1}...x_j$, $1 \leq i \leq j \leq |X|$.

$$D^k(X,Y) = \min_{V \epsilon Q^k(X)} \left[D(V,Y) \right] \tag{3.2}$$

The following example illustrates these quantities.

Example II

Let X=nion and Y=son.

$Q^1(X)$= {n, ni, nio, nion}, $Q^2(X) = Q^1(X) \cup$ {i, io, ion}

$Q^4(X) = Q^3(X)$ = {n, ni, nio, nion, i, io, ion, o, on}.

The intersymbol edit distances in the GLD are only zero or one.

$D^1(X,Y) = 2$ and $D^i(X,Y) = 1$ for i=2,3,4.

***

We now demonstrate that $D^k(X,Y)$ is recursively computable.

Theorem II

Let $D^k(X,Y)$ be the k-th distance between X and Y, defined as in (3.2). Then $D^k(X,Y)$ can be evaluated using $D^{k-1}(X,Y)$ and the distances obtained in the process of computing $D(X_{k,N},Y)$ where $|X|=N$.

Proof. Let

$$B_k(X) = \{V \mid V \text{ is a prefix of } X_{k,N} \text{ ; } |X| = N \}$$

By definition,

$$Q^k(X) = Q^{k-1}(X) \cup B_k(X)$$

Hence,

$$D^k(X,Y) = \text{Min} \left[D^{k-1}(X,Y), \min_{V \epsilon B_k(X)} \{ D(V,Y) \} \right]$$

The computation of $D(X_{k,N},Y)$ by the Wagner-Fischer algorithm [7] yields all the distances $D(V,Y)$, $V \epsilon B_k(X)$.

***

We now present an algorithm to compute $D^k(X,Y)$, which is essentially an application of the above theorem. The heart of the algorithm is the Wagner-Fischer Algorithm [7]. We have utilized the fact that in the computation of any GLD, D(Z,W), the latter algorithm automatically computes the GLD between the individual prefixes of both Z and W.

Algorithm D^k

Input : $X=x_1x_2\cdots x_N$, $Y=y_1y_2\cdots y_M$ and any k, $1 \leq k \leq N$. The intersymbol edit distances are also prespecified for all $a,b \in A$ as below:

$d(a,b)$: the distance associated with substituting 'a' with 'b'.

$d(a,\lambda)$: the distance associated with deleting 'a'.

$d(\lambda,b)$: the distance associated with inserting 'b'.

Output : The quantity $D^k(X,Y)$ defined by (3.2).

Method :

$D^k(X,Y) = \infty$
for q = N-k+ 1 to N do
  V = $X_{N-q+1,N}$
  P(0,0)=0
  for i=1 to q do
    P(i,0) = P(i-1,0) + $d(v_i , \lambda)$
  end
  for j=1 to M do
    P(0,j) = P(0,j-1) + $d(\lambda , y_j)$
  end
  for i=1 to q do
    for j=1 to M do
      r1 = P(i-1,j-1) + $d(v_i , y_j)$
      r2 = P(i,j-1) + $d(\lambda , y_j)$
      r3 = P(i-1,j) + $d(v_i , \lambda)$
      P(i,j) = Min [r1, r2, r3]
    end
    $D^k(X,Y)$ = Min [$D^k(X,Y)$, P(i,M)]
  end
end
END Algorithm D^k

Remarks

(1) The string V sequentially assumes the value of the suffixes of X. The array element P(i,j) is the GLD between the prefix of V of length i and the prefix of Y of length j. Thus, P(i,M) will be the GLD between $V_{1,i}$ and Y. As each P(i,M) is computed, the value of $D^k(X,Y)$ is updated.

(2) If $|V| = q$ and $|Y| = M$, the quantity D(V,Y) can be computed using exactly qM symbol comparisons. Hence, the number of symbol comparisons required to compute $D^k(X,Y)$ is :

$$\sum_{q=N-k+1}^{N} qM = M\left[\frac{N.(N+1)}{2} - \frac{(N-k).(N-k+1)}{2}\right]$$

$$= M\left[N.k - \frac{k.(k-1)}{2}\right] \qquad (3.3)$$

From (3.3) we see that the number of symbol comparisons required to compute $D^k(X,Y)$ is cubic in M, N and k, the terms in k being ***monotonically decreasing.***

(3) Since $Q^{|X|}(X)$ is exactly the set of all the contiguous substrings of X, the quantity $D^{|X|}(X,Y)$ is the minimum GLD between any substring of X, and Y. This quantity can be computed in $O(M|X|^2)$ time.

III.2 Procedure for Obtaining $S^*(Y)$

We now present an algorithm to compute $S^*(Y)$ by redefining it in terms of the distances $D^{|X|}(X,Y)$, $X \epsilon H$.

Theorem III

$S^*(Y)$ defined by (2.3) has the following expression:

$$S^*(Y) = \{X \mid X \epsilon H,\ D^{|X|}(X,Y) \leq D^{|X'|}(X',Y)\ \forall X' \epsilon H\} \qquad (3.4)$$

Proof.

From (3.1) we see that $Q^{|X|}(X)$ is the set of all the contiguous substrings of X. Hence,

$$\bigcup_{X \epsilon H} Q^{|X|}(X) = H^* \qquad (3.5)$$

By definition,

$$D^{|X|}(X,Y) = \min_{V} D(V,Y),\ V \text{ is a substring of } X.$$

Minimizing both sides over X, $X \epsilon H$, and using (3.5), we get,

$$\min_{X \epsilon H} D^{|X|}(X,Y) = \min_{V \epsilon H^*} D(V,Y) \qquad (3.6)$$

Thus, if $V \epsilon H^*$ minimizes D(V,Y) and X contains the substring V, then X will

definitely minimize $D^{|X|}(X,Y)$. Conversely, if any $X \epsilon H$ minimizes $D^{|X|}(X,Y)$, then X will contain at least one substring V, which minimizes D(V,Y). Hence the definition of $S^*(Y)$ given by (3.4) is equivalent to the one given by (2.2).

***

Using this definition, $S^*(Y)$ can be computed as below.

Algorithm S^*

Input : The dictionary H and the noisy string Y.

Output : The set $S^*(Y)$ defined by (2.2).

Method :

$S^*(Y) = \phi$

for every $X \epsilon H$

compute $D^{|X|}(X,Y)$ using Algorithm D^k

end

$A = \underset{X \epsilon H}{\text{Min}} \left[D^{|X|}(X,Y)\right]$

for every $X \epsilon H$

if $D^{|X|}(X,Y) = A$

$S^*(Y) = S^*(Y) \cup \{X\}$

end

END Algorithm S^*

Since any one $D^{|X|}(X,Y)$ can be computed in time proportional to $|X|^2|Y|$, the set $S^*(Y)$ can be computed in cubic time. We now propose an approximation of $S^*(Y)$ which is more easily computable.

III.3 An Approximation to $S^*(Y)$

By definition, Y is a substring of an unknown string U, containing substitution, insertion and deletion errors. Since U can be any element of H^*, $S^*(Y)$ was defined in terms of the GLD between Y and any arbitary element of H^*. To compute $S^*(Y)$, the measure $D^{|X|}(X,Y)$ was computed for every $X \epsilon H$.

An approximation to $S^*(Y)$ can be obtained by terminating the recursive computation of $D^k(X,Y)$ at some i, $1 \leq i \leq |X|$. This would be equivalent to comparing Y with only those substrings of X which are in the set $Q^i(X)$. If the string Y is of length M, and if $M \leq |X|$, then we can be sure that every suffix of X of length greater than or equal to M will be in the set $Q^{|X|-M+1}(X)$. Further, the latter set will contain *all the prefixes* of every one of these suffixes. Hence a good approximation to $D^{|X|}(X,Y)$ is the quantity $D^{|X|-M+1}(X,Y)$, if $M \leq |X|$.

To generalize this for all M, we approximate $D^{|X|}(X,Y)$ by the quantity $D^K(X,Y)$, where K is given by (3.7).

$$K = \text{Max}\left[|X|-M+1\ ,\ 1\right] \tag{3.7}$$

The estimate $S^M(Y)$ which approximates $S^*(Y)$ is obtained by comparing Y with only those prefixes of X of length greater than or equal to K, defined above. We define $S^M(Y)$ as the set of all the words in H which minimizes $D^K(X,Y)$. Explicitly,

$$S^M(Y) = \{X \mid X \in H; D^K(X,Y) \leq D^K(X',Y),\ \forall X' \in H, \text{ where } K = \max[\ |X|-M+1\ ,1\ ]\}$$

An algorithm similar to Algorithm S^* can be given to compute this set.

In practice, it is observed that the frequency of $S^M(Y)$ containing T(U) is only slightly less than the frequency of $S^*(Y)$ containing T(U). However, it is definitely less expensive to compute. Consider the computation of $D^K(X,Y)$, where K is given by (3.7) for any X. Using Algorithm D^k, this can be computed using $\tau_{X,M}$ symbol comparisons, where,

$$\tau_{X,M} = M\left[\ |A|.K - \frac{K.(K-1)}{2}\ \right] \quad \text{if } K > 1$$

$$= M|X| \qquad \text{if } K = 1$$

The latter expression simplifies to (3.8) using (3.7).

$$\tau_{X,M} = \frac{M}{2}\left[\ |X|(|X|+1) - M(M-1)\right] \qquad \text{if } M < |X|$$

$$= M|X| \qquad \text{if } M \geq |X|, \tag{3.8}$$

Just as $D^{|X|}(X,Y)$ requires cubic computation time, we see from (3.8) that $D^K(X,Y)$ also requires cubic time. However, due to the negative quadratic terms in M in $\tau_{X,M}$ the value of the latter *decreases* as M increases. The fractional decrease in computation is the ratio of M(M-1) to $|X|(|X|+1)$ which is of the order of $(M/|X|)^2$. Further, in the limit, the expression for $\tau_{X,M}$ is quadratic. This renders $S^M(Y)$ to be, in the limit, one order computationally less expensive than $S^*(Y)$.

IV EXPERIMENTAL RESULTS AND DISCUSSION

The noisy substring matching problem was studied in five experiments involving 1,900 noisy strings and two dictionaries which are subsets of the 1,023 most common English words [11]. In the first two experiments the 292 most common English words of length greater than or equal to seven, constituted the dictionary. In the last three experiments, the dictionary, was the 166 most common words of length greater than or equal to eight.

Noisy strings were generated as follows. A substring U was randomly chosen from a string X ϵ 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. The noisy strings used in the study contained at least one error, but not more than two. The average number of errors per noisy string was approximately 1.4. Variations in the experiments were achieved by varying the minimum length of U.

For purposes of standardization, the elementary edit distances used in the Levenshtein metric were zero or one. The performance index used to evaluate the estimates was the frequency of the estimates $S^*(Y)$ and $S^M(Y)$ containing T(U). Details of the experiments are given in Tables IV.1 and IV.2. From the tables, we see that the estimate $S^*(Y)$ is very accurate. $S^*(Y)$ was least accurate in Experiment III, in which, it contained T(U) in 490 out of the 500 cases studied. This corresponds to an accuracy of 98.0%. The average size of the set $S^*(Y)$ in this experiment was 2.39. From Table IV.1, we also see that the estimate $S^M(Y)$ is almost as accurate as $S^*(Y)$. In the same experiment, $S^M(Y)$ contained T(U) in 482 out of the 500 cases studied corresponding to an accuracy of 96.4%. The average size of $S^M(Y)$ in this experiment was 2.35.

To compare the computations required to compute $S^*(Y)$ and $S^M(Y)$, the number of symbol comparisons required in the first two experiments, was evaluated for various values of M, the length of Y. The results are tabulated in Table IV.3 and graphically presented in Figure IV.1. From both these, we see that the number of symbol comparisons required to compute $S^*(Y)$ is much greater than the number of symbol comparisons required to compute $S^M(Y)$. For example, if M=9, $S^*(Y)$ and $S^M(Y)$ required 100,017 and 27,891 symbol comparisons respectively. The decrease in the computation required for $S^M(Y)$ with increasing values of M is also obvious from Figure IV.1. In our judgement, the marginal loss in accuracy of $S^M(Y)$ is more than compensated by its computational advantage.

V CONCLUSIONS

We considered the problem of estimating the set T(U), the subset of words in the dictionary H which contains U as a substring, using only Y, a noisy version of U. The suggested set estimate $S^*(Y)$ which needs cubic time for computation has relatively high accuracy as verified by experiments. Another estimate $S^M(Y)$ was also proposed which is only slightly less accurate than $S^*(Y)$ but requires substantially less computation. The estimate, $S^*(Y)$ is the proper subset of H, every element of which contains a substring which resembles Y the most according to the Generalized Levenshtein metric. The algorithms for the computation of $S^*(Y)$ or $S^M(Y)$ are easily extendible to other measures of

dissimilarity between strings such as the Length of their Shortest Common Supersequence, and the Length of their Longest Common Subsequence.

REFERENCES

1. Knuth, D. E., Morris Jr., J. H., and Pratt, V. R., "Fast Pattern Matching In Strings", SIAM Journal of Comp., Vol. 6, No. 2, 1977, pp. 323-350.
2. Boyer, R. S., and Moore, J. S., "A Fast String Searching Algorithm", C-ACM, Vol. 20, No. 10, 1977, pp. 762-772.
3. The UNIX text editor.
4. Kashyap, R. L., and Oommen, B. J., "An Effective Algorithm for String Correction Using a Generalized Edit Distance - I. Description of the Algorithm and its Optimality", Information Sciences, Vol. 23, No. 2, 1981, pp.123-142.
5. Kashyap, R. L., and Oommen, B. J., "An Effective Algorithm for String Correction Using a Generalized Edit Distance - II. Computational Complexity of the Algorithm and Some Applications", Information Sciences, Vol. 23, No. 3, 1981, pp.201-217.
6. Kashyap, R. L., and Oommen, B. J., "A Unifying Theory for Order Preserving Properties Involving Two Strings", Proceeding of the Princeton Conf. on Information Sciences and Systems, 1980, pp.193-198.
7. Wagner, R. A., and Fischer, M. J., "The String to String Correction Problem", J-ACM, Vol. 21, 1974, pp. 168-173.
8. Hirschberg, D.S., "Algorithms for the Longest Common Subsequence Problem", J-ACM, Vol. 24, 1977, pp.664-675.
9. Okuda, T., Tanaka, E., and Kasai, T., "A Method of Correction of Garbled Words Based on the Levenshtein Metric," IEEE Trans. Comp., Vol. C-25, Feb. 1976, pp. 172-177.
10. Kashyap, R. L., "Syntactic Dec. Rules for the Recognition of Spoken Words and Phrases Using a Stochastic Automaton", IEEE Trans. on Pat. Anal. and Mach. Intel., Vol. 1, 1979, pp.154-163.
11. Dewey, G., "Relative Frequency of English Speech Sounds", Harvard Univ. Press, 1923.

Exp.	Nstr.	Dict.	Min.|U|	Av. Noise	Estimate	Acc.	Av. Size
I	200	H1	5	1.38	$S^*(Y)$	0.990	3.38
I	200	H1	5	1.38	$S^M(Y)$	0.970	2.89
II	200	H1	6	1.36	$S^*(Y)$	0.980	1.89
II	200	H1	6	1.36	$S^M(Y)$	0.970	1.81
III	500	H2	5	1.36	$S^*(Y)$	0.980	2.39
III	500	H2	5	1.36	$S^M(Y)$	0.964	2.35
IV	500	H2	6	1.40	$S^*(Y)$	0.992	1.48
IV	500	H2	6	1.40	$S^M(Y)$	0.990	1.47
V	500	H2	7	1.43	$S^*(Y)$	0.994	1.30
V	500	H2	7	1.43	$S^M(Y)$	0.992	1.23

Table IV.1 : Results of the five experiments. The notation used is :

Nstr : No. of noisy strings used in the experiment.

H1 : Dictionary consisting of 292 words of length ≥ 7.

H2 : Dictionary consisting of 166 words of length ≥ 8.

Acc. : Frequency of the estimate S^* or S^M containing T(U).

Av. Noise : Average no. of errors in a single string Y.

Av. Size : Average size of the estimate set.

U	Y	An element of $S^*(Y)$	Size of $S^*(Y)$
dditio	ditxo	addition	3
administ	ndmibist	administration	1
advanta	advuantpa	advantage	1
altoget	altkget	altogether	1
nything	nthing	anything	3
ently	etly	apparently	5
artill	vrtfill	artillery	1
lable	lablr	available	1
ginnin	ginin	beginning	2
ombardm	imbardmr	bombardment	1
uildi	uslsi	building	7
siness	sinejss	business	1
paign	paiyb	campaign	1
ittee	ittzee	committee	1
communi	vomuni	community	1
conditio	donditio	condition	2
idenc	idenx	confidence	2
nside	nsde	consider	7
erable	ezrable	considerable	1
idere	ivdere	considered	2
ntinue	sntinue	continue	1
ontract	ohtragct	contract	1
onden	onlcen	correspendent	1
clared	ckared	declared	1
defende	defepde	defender	1
ivere	ivee	delivered	4
evelop	eelop	developed	1
enclos	enclod	enclosed	1
nforced	nfkrcec	enforced	1
uipment	uuipment	equipment	1
blishe	bpishe	established	1
verythi	veeythi	everything	1
verywher	veryahef	everywhere	1
utive	ucive	executive	1
istence	iztence	existence	1
xistin	xistiny	existing	1
expect	expecti	expected	1
press	eess	expressed	7
llowed	lliwec	followed	1
enera	eeray	generally	9

Table IV.2 : Some results obtained in Experiment III.
In all the cases $S^*(Y)$ contains T(U).

M	τ^*	τ^M
5	55,565	40,965
6	66,678	40,398
7	77,791	34,867
8	88,904	30,552
9	100,017	27,891
10	111,130	27,480
11	122,243	28,138
12	133,356	29,376
13	144,469	31,200
14	155,582	33,236

Table IV.3 : Comparison of the number of symbol comparisons required to compute $S^*(Y)$ with the number of symbol comparisons required to compute $S^M(Y)$. The notation is :

M : Length of the string Y. τ^* : No. of symbol comparisons required to compute $S^*(Y)$.

τ^M : No. of symbol comparisons required to compute $S^M(Y)$.

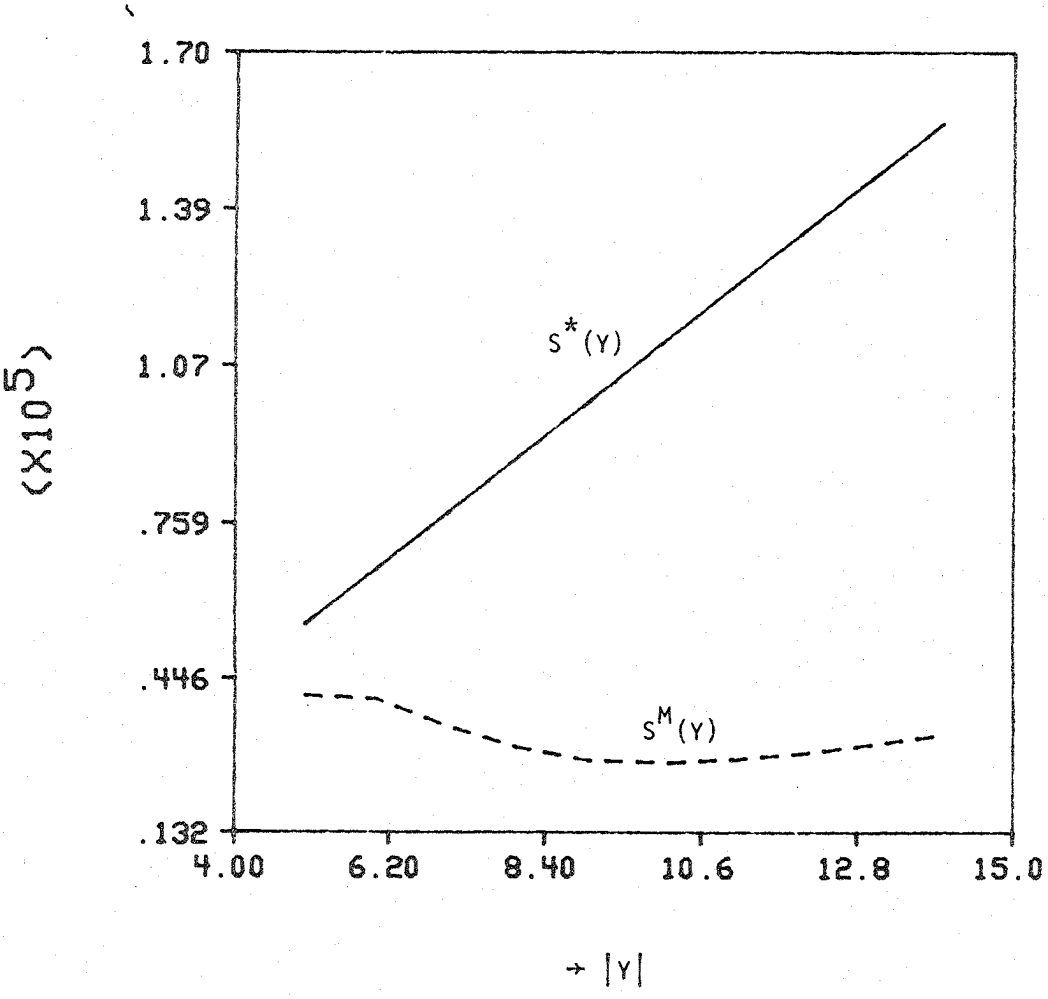

Figure IV.1: Graphs of the number of symbol comparisons required to compute $S^*(Y)$ and $S^M(Y)$ versus $|Y| = M$ in experiments I and II.

CARLETON UNIVERSITY
School of Computer Science

Bibliography of SCS Reports

SCS-TR-1 THE DESIGN OF CP-6 PASCAL
Jim des Rivieres and Wilf R. LaLonde, June 1982.

SCS-TR-2 SINGLE PRODUCTION ELIMINATION IN LR(1) PARSERS:A SYNTHESIS
Wilf R. LaLonde, June 1982.

SCS-TR-3 A FLEXIBLE COMPILER STRUCTURE THAT ALLOWS DYNAMIC PHASE ORDERING
Wilf R. LaLonde and Jim des Rivieres, June 1982.

SCS-TR-4 A PRACTICAL LONGEST COMMON SUBSEQUENCE ALGORITHM FOR TEXT COLLATION
Jim des Rivieres, June 1982.

SCS-TR-5 A SCHOOL BUS ROUTING AND SCHEDULING PROBLEM
Wolfgang Lindenberg, Frantisek Fiala, July 1982.

SCS-TR-6 ROUTING WITHOUT ROUTING TABLES
Nicola Santoro, Ramez Khatib, July 1982.

SCS-TR-7 CONCURRENCY CONTROL IN LARGE COMPUTER NETWORKS
Nicola Santoro, Hasan Hural, July 1982.

SCS-TR-8 ORDER STATISTICS ON DISTRIBUTED SETS
Nicola Santoro, Jeffrey B. Sidney, July 1982.

SCS-TR-9 OLIGARCHICAL CONTROL OF DISTRIBUTED PROCESSING SYSTEMS
Moshe Krieger, Nicola Santoro, August 1982.

SCS-TR-10 COMMUNICATION BANDS FOR SELECTION IN A DISTRIBUTED SET
Nicola Santoro, Jeffrey B. Sidney, September 1982.

SCS-TR-11 A SIMPLE TECHNIQUE FOR CONVERTING FROM A PASCAL SHOP TO C SHOP
Wilf R. LaLonde, John R. Pugh, November 1982.

SCS-TR-12 EFFICIENT ABSTRACT IMPLEMENTATIONS FOR RELATIONAL DATA STRUCTURES
Nicola Santoro, December 1982.

SCS-TR-13 ON THE MESSAGE COMPLEXITY OF DISTRIBUTED PROBLEMS
Nicola Santoro, December 1982.

SCS-TR-14 A COMMON BASIS FOR SIMILARITY MEASURES INVOLVING TWO STRINGS
R.L. Kashyap and B.J. Oommen, January 1983.

SCS-TR-15 SIMILARITY MEASURES FOR SETS OF STRINGS
R.L. Kashyap and B.J. Oommen, January 1983.

SCS-TR-16 THE NOISY SUBSTRING MATCHING PROBLEM
R.L. Kashyap and B.J. Oommen, January 1983.