INFORMATION SCIENCES **23**, 201–217 (1981)

An Effective Algorithm for String Correction Using Generalized Edit Distances— II. Computational Complexity of the Algorithm and Some Applications*

R. L. KASHYAP
and
B. J. OOMMEN

School of Electrical Engineering, Purdue University, West Lafayette, Indiana 47907

Communicated by John M. Richardson

ABSTRACT

This paper deals with the problem of estimating an unknown transmitted string X_s belonging to a finite dictionary H from its observable noisy version Y. In the first part of this paper [18] we have developed an algorithm, referred to as Algorithm I, to find the string $X^+ \in H$ which minimizes the generalized Levenshtein distance $D(X/Y)$. In this part of the paper we study the computational complexity of Algorithm I, and illustrate quantitatively the advantage Algorithm I has over the standard technique and other algorithms. Its superiority has been shown for various dictionaries, including the one consisting of the 1021 most common English words of length greater than unity [23]. A comparison between Algorithm I and other algorithms used to correct misspelled words of a regular language is also made here. Some applications of Algorithm I are also discussed.

I. INTRODUCTION

This paper discusses the computational complexity of the algorithm, called Algorithm I, presented in the companion paper [18]. The algorithm processes a string Y, which is a noisy version of a transmitted string X_s, an element of a finite dictionary H. It yields as its output the string $X^+ \in H$, defined as the best estimate of X_s, which minimizes the generalized Levenshtein distance $D(X/Y)$ between X and Y.

To study the complexity of Algorithm I, we shall make use of the tree structure of the finite state machine (FSM) that accepts the dictionary H. We shall first derive an upper bound for the number of symbol comparisons

*Partially supported by National Science Foundation under Grant No. NSF 78-18271.

52 Vanderbilt Ave., New York, NY 10017 0020-0255/81/03201-17$02.50

required by Algorithm I in terms of the parameters of this tree. We shall then proceed to obtain an approximate, but more useful, expression for the number of symbol comparisons required by Algorithm I. Using these results, a quantitative comparison is made between Algorithm I and the standard technique (ST), which computes X^+ after individually computing $D(X/Y)$ for every $X \in H$. This comparison has been made for many dictionaries, including the one consisting of the 1021 most common English words of length greater than unity [23]. From the results presented here, we claim that Algorithm I is in general less complex than the ST and other algorithms.

Throughout this paper, a knowledge of the rest of the terminology and the notation of the companion paper [18] is assumed.

II. COMPUTATIONAL COMPLEXITY

To evaluate the number of symbol comparisons Algorithm I needs, we will have to take into account *not only* the lengths of the words in the dictionary and the length of the incoming string, but also the advantage obtained by the use of the information contained in the prefixes of the words. To do this we shall make use of the tree structure of the FSM that accepts H, described in Sec. III of our companion paper [18]. Let V be the set of all the prefixes of the words in H. Any study of the complexity of Algorithm I will have to progressively count the number of elements of V (or equivalently, the number of nodes of the tree) to which $Y^{(K)}$ can be edited with a finite pseudodistance. The technique by which this counting is done is explained in the next subsection.

II.1. UPPER BOUND FOR THE NUMBER OF SYMBOL COMPARISONS REQUIRED

Let H be a dictionary consisting of J words, and let N_m be the length of the longest word in H. Let M be the length of the noisy string Y. Let q_i be the number of elements of V of length i. Since μ is the only prefix of length zero, we define q_0 to be equal to unity. Further, since N_m is the length of the longest words in H, q_{N_m+1} is identically equal to zero.

For the sake of convenience, we define a_i as the ratio of q_i to q_{i-1} for $i \geq 1$:

$$a_i = \frac{q_i}{q_{i-1}}, \qquad i \geq 1.$$

By definition, a_0 is unity, and a_{N_m+1} is zero. Consequently, $q_i = a_0 a_1 \ldots a_i$ for all i. a_i can be interpreted as the average number of branches from the nodes of the tree which have a depth of $i-1$.

To simplify the notation, we introduce a constant T, defined as

$$\begin{aligned} T &= \left\lceil \frac{N_m}{2} \right\rceil \quad \text{if} \quad M \geqslant \frac{N_m}{2}, \\ &= M \qquad \text{otherwise,} \end{aligned} \tag{2.1}$$

where $\lceil b \rceil$ is the smallest integer greater than b. Using the above constants we shall first obtain an upper bound for the size of the set $R^{(i)}$.

LEMMA I. *Let q_i be the number of elements of V which are of length i. Then an upper bound for the number of elements in $R^{(i)}$ is*

$$\begin{aligned} \#(R^{(i)}) &\leqslant \sum_{j=0}^{2i} q_j \qquad \text{if} \quad i < T, \\ &\leqslant \sum_{i=0}^{N_m} q_j \qquad \text{otherwise.} \end{aligned}$$

Proof. Suppose we edit $Y^{(i)}$ into some string Z. If the pseudodistance $D_1(Z/Y^{(i)})$ has to be finite, then by virtue of Assumption ii of [18], $|Z|$ must be less than or equal to $2i$. Hence, the set $R^{(i)}$ is always a subset of the smaller of the two sets $H^{(2i)}$ and V. When $i < T$, the smaller of the two sets is always $H^{(2i)}$, and when $i \geqslant T$, every string in V can be in $R^{(i)}$. The Lemma is proved by combining this fact with the fact that

$$\#(H^{(i)}) = \sum_{j=0}^{i} q_j. \qquad \blacksquare$$

We shall now derive an upper bound for the number of symbol comparisons required by Algorithm I.

THEOREM I. *Let q_i be the number of elements of V which are of length i. Further, let q_{N_m} be zero. Then an upper bound for the number of symbol comparisons required by Algorithm I is given by δ_{A1} as*

$$\delta_{A1} = \sum_{i=1}^{T} \sum_{j=0}^{2(i-1)} (q_{j+1} + q_{j+2}) + (M - T) \sum_{j=0}^{N_m - 1} (q_{j+1} + q_{j+2}). \tag{2.2}$$

Proof. Consider the tree which represents the FSM described above.

(1) Between any node α and its child α_d there is only one extra symbol, say a_{df}, which is the last symbol of the node α_d. Thus for any node $\alpha \in R^{(i-1)}$, we need only one symbol comparison, namely that of a_{df} with y_i, to determine whether its child α_d belongs to $R^{(i)}$ or not.

(2) Between any node α and its grandchild α_g there are two extra symbols, say a_{gf} and a_{df}, the last and the last-but-one symbols of α_g respectively. By virtue of Theorem II of [18], for every node $\alpha \in R^{(i-1)}$, we need only *one* symbol comparison, namely that of a_{gf} with y_i, to determine whether α_g belongs to $R^{(i)}$ or not.

(3) Consider the ith stage of Algorithm I, when the symbol y_i has to be processed. Let us assume that all the q_j nodes at the jth level of the tree are in $R^{(i-1)}$. These q_j nodes together have exactly q_{j+1} children nodes, and q_{j+2} grandchildren nodes. Using the arguments of (1) and (2) above, we will need one symbol comparison for each child and grandchild node respectively to determine whether they are in $R^{(i)}$ or not. Hence in the process of iterating from stage $i-1$ to stage i the total number of symbol comparisons required by all the q_j nodes in $R^{(i-1)}$ is given by

$$q_{j+1}+q_{j+2}.$$

Clearly this will be true for all $0 \leqslant j \leqslant N_m - 1$, since $q_{N_m+1}=0$.

(4) Combining the results of (1)–(3) above, and using the expression for the upper bound of the size of $R^{(i)}$ given by Lemma I, we obtain that for a *given* $i \leqslant T$, the total number of symbol comparisons required to iterate from stage $i-1$ to stage i has an upper bound of

$$\sum_{j=0}^{2(i-1)} (q_{j+1}+q_{j+2}). \tag{2.3}$$

Consequently, the total number of symbol comparisons required to iterate till stage T is bounded by

$$\sum_{i=1}^{T} \sum_{j=0}^{2(i-1)} (q_{j+1}+q_{j+2}). \tag{2.4}$$

In a similar way we argue that the total number of symbol comparisons required to iterate from stage T till the end of the algorithm is bounded by

$$(M-T) \sum_{j=0}^{N_m-1} (q_{j+1}+q_{j+2}). \tag{2.5}$$

If $M<|N_m/2|$, obviously, only the terms contributed by (2.4) need to be included in δ_{A1}. In such a case, the terms contributed by (2.5) will be automatically excluded, since T will then have the value M. Combining (2.4) and (2.5), we obtain the expression given in (2.2), and the theorem is proved. ■

REMARK. The above expression can be equivalently written in terms of the a_j's defined earlier. In the case of a dictionary in which each a_j is equal to an integer a, where $a>2$, δ_{A1} has the form

$$\delta_{A1}=\sum_{i=1}^{T}\sum_{j=0}^{2(i-1)}(a^{j+1}+a^{j+2})+(M-T)\sum_{i=0}^{N_m-2}(a^{i+1}+a^{i+2})+(M-T)J.$$

If all the words in H are of the same length, this reduces to

$$\delta_{A1}=\frac{a^2}{(a-1)^2}\cdot a^{2T}+(M-T)J\cdot\frac{2a}{a-1}-\frac{a^2}{(a-1)^2}-\frac{Ma(a+1)}{(a-1)}. \quad (2.6)$$

II.2. APPROXIMATE NUMBER OF SYMBOL COMPARISONS REQUIRED

Simulations of Algorithm I indicate that the upper bound for the size of $R^{(i)}$ given by Lemma I is far too conservative. Thus, we believe that the upper bound δ_{A1} is also far too conservative. We will derive a "better" estimate of the number of symbol comparisons required by Algorithm I, by obtaining an estimate of $R^{(i)}$ which gives a better reflection of its true size than that given by Lemma I.

Suppose that with every $b\in A$, we associate a set A_b which is a subset of A. Let $x\in A_b$ iff $d(x/b)<\infty$. A_b is the set of letters in A which could have given rise to the received symbol b. The set A_b is always a set much smaller than A [12, 21]. Hence $R^{(i)}$ will not contain all the elements of V which are of length less than or equal to i. Some typical values of the sizes of $R^{(i)}$ and $H^{(i)}$ obtained in the actual simulation of Algorithm I for a dictionary of 50 words used in the context of computers is given in Table 1.

TABLE 1

i	Approx. Size $R^{(i)}$	Size of $H^{(i)}$
1	17	17
2	33	52
3	51	97
4	74	138
5	100	167

Table 1 seems to indicate that a more reasonable assumption regarding the set $R^{(i)}$ is that $R^{(i)}$ has the same number of elements as $H^{(i)}$. Using this assumption, and arguments similar to the ones used to get the upper bound δ_{A1}, we can derive an expression for θ_{A1}, the approximate number of symbol comparisons required by Algorithm I. For a dictionary which has $a_j = a$, and when $M = N$, θ_{A1} will have the following expression:

$$\theta_{A1} = \sum_{i=1}^{N-1} \sum_{j=0}^{i-1} a^j(a+a^2) + \sum_{j=0}^{N-2} a^j(a+a^2) + J. \tag{2.7}$$

We shall later compare Algorithm I and the ST using (2.7).

III. COMPARISON OF ALGORITHM I WITH OTHER EXISTING ALGORITHMS

III.1. COMPARISON WITH THE ST

The standard technique (ST) of obtaining the string X^+ which minimizes the distance $D(X/Y)$ is one of individually computing the distances $D(X/Y)$ for every $X \in H$, and then by observation deciding on X^+. Wagner and Fisher [21] have given an algorithm using which $D(X/Y)$ can be computed for a given pair X and Y involving at least $|X||Y|$ symbol comparisons.

An asymptotically faster algorithm has been suggested in [5], but due to the amount of preprocessing required for it, the latter algorithm is preferable only when $|X|$ and $|Y|$ are very large. In our comparison of Algorithm I with the ST, we shall restrict ourselves to the algorithm of Wagner and Fisher [21], since it has some desirable optimal properties [22].

Let H, the finite dictionary, consist of J words, and let N_i be the length of the ith word in H. Then θ_{ST}, the minimum number of symbol comparisons required by the ST, has the expression

$$\theta_{ST} = M \sum_{i=1}^{J} N_i,$$

where M is the length of the noisy string Y.

Comparing the expressions for θ_{ST} and δ_{A1}, we observe that they are dependent on different sets of parameters of H. Whereas θ_{ST} involves the lengths of the words in H, δ_{A1} involves the parameters q_i, $i \leqslant N_m$, defined earlier. Thus a quantitative comparison between θ_{ST} and δ_{A1} can be made only for specific dictionaries, in which all the parameters required for the comparison are known.

We shall employ the following strategy to compare θ_{ST} and δ_{A1}. In Case i, we first consider the dictionary consisting of the 1021 most common English words of length greater than unity [23] and show the superiority of Algorithm I over the ST for this dictionary. In Cases ii and iii we consider some generalized dictionaries specified by the parameters of the FSM accepting H. Concerning the lengths of the words in H, we observe that whereas δ_{A1} requires the knowledge of the length of only the longest word in H, θ_{ST} requires the knowledge of the lengths of all the words in H. Specifying the lengths of all the words in H will increase the number of parameters in this comparison, and will make the comparison more cumbersome. To simplify the issue, we make the comparison in the latter two cases by assuming that all the words in H are of the same length N.

Case i. Comparison with the ST for a Practical Dictionary

We shall first compare Algorithm I with the ST for a practical dictionary, taking into consideration both the varying lengths of the words in H, and the statistical properties of the prefixes of the words in H. The dictionary which we have used is the set of the 1021 most common English words of length greater than unity [23]. The lengths of the words varied from 2 to 14, the average length being 5.57.

In Table 2 we give some statistics regarding the dictionary and of its prefixes. From the table we observe that though the sum of the lengths of all the strings is 5684, the total number of states in the FSM is only 2942.

TABLE 2

i	No. of words	q_i [a]
1	0	24
2	27	157
3	91	524
4	239	677
5	205	561
6	167	409
7	126	272
8	76	156
9	51	88
10	20	39
11	9	19
12	6	10
13	2	4
14	2	2

[a]Notation: q_i is the number of elements of V of length i.

Using arguments analogous to those used in Theorem I, we have computed δ_{A1}, the upper bound of the number of symbol comparisons required by Algorithm I. We have also computed θ_{ST}, the number of symbol comparisons required by the ST. δ_{A1} and θ_{ST} have been evaluated for values of M, the length of the incoming string, ranging from 2 to 14. Since we have assumed that the channel obeys Assumption ii of [18], if the noisy string is of length M, the length of the transmitted words has to be less than or equal to $2M$. Hence, to be fair, when we computed θ_{ST}, we considered only the words of H which were of length less than or equal to $2M$.

In Table 3 we have tabulated the results obtained. The superiority of Algorithm I over the ST is obvious. For example, if the length of the incoming string is 4, θ_{ST} is greater than δ_{A1} by a factor of 1.614. In this case, the number of symbol comparisons required by the ST is at least 19,200; the upper bound on the number of symbol comparisons required by Algorithm I is only 11,895. The approximate number of symbol comparisons in this case is only 6,407, indicating that Algorithm I has an approximate advantage over the ST of a factor of about 3.00. The average value of θ_{A1} is 28,180, and the average value of θ_{ST} is 43,843, which exceeds θ_{A1} by approximately 56%.

TABLE 3

M	θ_{ST}	δ_{A1}	$\frac{\theta_{ST}}{\delta_{A1}}$	θ_{A1}	$\frac{\theta_{ST}}{\theta_{A1}}$
2	2,566	2,244	1.15	1,043	2.46
3	9,930	6,515	1.52	3,106	3.20
4	19,200	11,895	1.61	6,407	3.00
5	27,295	17,646	1.55	10,678	2.56
6	33,780	23,484	1.44	15,630	2.16
7	39,788	29,342	1.36	21,010	1.89
8	45,472	35,202	1.29	26,634	1.71
9	51,156	41,062	1.25	32,385	1.58
10	56,840	46,922	1.21	38,194	1.49
11	62,524	52,782	1.19	44,032	1.42
12	68,208	58,642	1.16	49,884	1.37
13	73,892	64,502	1.14	55,742	1.33
14	79,576	70,362	1.13	61,602	1.29

The superiority of Algorithm I over the ST for this dictionary is much smaller than what we can obtain for other dictionaries. In this case, whereas a_1 has the value 24, and a_2 has the value of about 6.5, all the other a_i's are of the order of unity. This large value of a_1 in comparison with the rest of the a_i's dampens the advantage gained by Algorithm I. A much greater advantage can be expected when all the a_i's are of the same order of magnitude. The following two cases illustrate this.

Case ii. When $M=N\leqslant 4$ and the Parameters a_i Vary Independently

When $N=2$,

$$\delta_{A1}=2a_1+3a_1a_2$$

and

$$\theta_{ST}=4a_1a_2.$$

It is seen that $\theta_{ST}>\delta_{A1}$ for all a_1 when $a_2>2$. When $a_2=2$, $\theta_{ST}=\delta_{A1}$, and hence, independently of a_1, for all $a_2>2$, Algorithm I is computationally less complex than the ST.

When $N=3$,

$$\delta_{A1}=3a_1+5a_1a_2+4a_1a_2a_3$$

and

$$\theta_{ST}=9a_1a_2a_3,$$

so

$$\theta_{ST}>\delta_{A1} \quad \text{for} \quad a_3>\frac{3}{5a_2}+1. \tag{3.1}$$

From (3.1) we observe that $\theta_{ST}>\delta_{A1}$ for all dictionaries for which $a_2=1$. Further, when $a_2>1$, the subset of dictionaries for which the ST can be superior to Algorithm I is a small subset of the set of dictionaries that can be formed with words of length 3 having $a_2>1$. This is because, if a_3 is plotted as a function of a_2 so as to satisfy (3.1), it is seen that area in which $a_3<3/5a_2+1$ is much smaller than the area in which $a_3>3/5a_2+1$.

When $N=4$,

$$\delta_{A1}=4a_1+7a_1a_2+6a_1a_2a_3+5a_1a_2a_3a_4$$

and

$$\theta_{ST}=16a_1a_2a_3a_4. \tag{3.2}$$

The inequality $\theta_{ST}>\delta_{A1}$ has three independent parameters a_2, a_3, and a_4. But

we can analyze it for various values of a_2. For example, if $a_2 = 1$,

$$\theta_{ST} > \delta_{A1} \quad \text{when} \quad a_4 > \frac{1}{a_3} + \frac{6}{11}.$$

In this case, the area in the a_3-a_4 space for which $\theta_{ST} > \delta_{A1}$ is much larger than the area for which $\theta_{ST} < \delta_{A1}$. Further, this area increases with a_2. Thus the subset of dictionaries for which the ST can be superior to Algorithm I is a very small subset of the set of dictionaries of words of length four.

Case iii. Dictionaries in Which $N > 4$ and All the a_i's Are Equal.

When $N > 4$, a comparison such as the above is more complex, due to the number of independent parameters. We shall simplify the comparison by using the expression (2.6). Consider the case when N is even. Since $J = a^N$,

$$\delta_{A1} = \frac{a^2 J}{(a-1)^2} + (2M-N)J\frac{a}{a-1} - \frac{a^2}{(a-1)^2} - \frac{Ma(a+1)}{a-1}, \tag{3.3}$$

$$\theta_{ST} = JMN.$$

From the above we observe that though θ_{ST} has a product term involving M and N, δ_{A1} has only terms linear in M and N. Hence, δ_{A1} will be one order of magnitude smaller than θ_{ST}.

In particular, $\delta_{A1} \leq \theta_{ST}$ for all $N > 2$ and even. This follows from the fact that since $a/(a-1) > 2$ for $a > 2$, the sufficient condition for $\delta_{A1} < \theta_{ST}$ reduces to

$$(M+2)(N-4) > -4,$$

which is clearly satisfied for all positive integers $M, N, N > 2$.

To appreciate the advantage of Algorithm I for more general values of J, M, and N, the equation (2.6) has been evaluated for various values of a, M, and N, $N > 4$. For any given a and N, the length M of the noisy string Y was varied from $N/2$ to $3N/2$. In each case the values of θ_{ST} and δ_{A1} and the ratio θ_{ST}/δ_{A1} were computed. The average values of θ_{ST}, δ_{A1}, and θ_{ST}/δ_{A3} are tabulated in Table 4. We have simultaneously tabulated the values of θ_{A1} from (2.7) for various values of the parameters and given the ratio of θ_{ST} to θ_{A1}. In the last column, we present the actual ratio of θ_{ST} to θ_{A1}, for $M = N$. The following are

TABLE 4[a]

J	a	N	$\mathrm{Av}(\delta_{A1})$	$\mathrm{Av}(\theta_{ST})$	$\mathrm{Av}\left(\frac{\theta_{ST}}{\delta_{A1}}\right)$	θ_{A1}	$\frac{\theta_{ST}}{\theta_{A1}}$
32	2	5	353	720	2.85	278	2.88
64	2	6	984	2,304	2.76	592	3.89
128	2	7	2,005	5,824	4.15	1,226	5.12
243	3	5	1,976	5,468	4.80	1,419	4.28
256	2	8	5,068	16,384	3.95	2,500	6.55
512	2	9	10,185	39,168	5.69	5,054	8.21
729	3	6	8,163	26,244	4.00	4,329	6.06
1,024	4	5	7,250	23,040	6.64	4,964	5.16
1,024	2	10	24,512	102,400	5.27	10,168	10.07

[a]Notation: Av( ) = average of the quantity in parentheses for $N/2 \leqslant M \leqslant 3N/2$.

some of the results observed from the computations:

(1) For all values of M and N, the ratio of θ_{ST} to δ_{A1} increases with J.

(2) For all values of J and N, the ratio θ_{ST}/δ_{A1} decreases with M.

(3) For all N and for all $N/2 \leqslant M \leqslant 3N/2$, independent of the value of J, $\theta_{ST} \gg \delta_{A1}$. For example, when H consisted of 256 words each of length 8, for $M=4$, the number of comparisons required by the ST was 8192, whereas 996 comparisons were required by Algorithm I. The ratio θ_{ST}/δ_{A1} for this J, N and M was 8.225. It must be noted that the average value of θ_{ST}/δ_{A1} is not the same as the ratio of the average values of θ_{ST} and δ_{A1}. The index $\mathrm{Av}(\theta_{ST}/\delta_{A1})$ has been computed by evaluating θ_{ST}/δ_{A1} for various values of M, $N/2 \leqslant M \leqslant 3N/2$ and averaging these quantities over M. In this case the average values of θ_{ST} and δ_{A1} were 16384 and 5068 respectively. The average value of θ_{ST}/δ_{A1} indicated an average advantage of Algorithm I over the ST of a factor 3.945.

(4) From the last two columns of the table, we observe that the approximate number of symbol comparisons required by Algorithm I given by θ_{A1} is much less than the upper bound given by δ_{A1}. Consequently, the advantage of Algorithm I over the ST is much greater than the sixth column seems to indicate. For example, in the case when H consists of 256 words, each of length eight and $a=2$, the number of approximate number of comparisons required by Algorithm I is only 2500, compared to the 16,384 comparisons required by the ST. In this case the gain in computation is approximately 6.5.

III.3. COMPARISON WITH ALGORITHMS OTHER THAN THE ST

Wagner [20] has given an algorithm, called Algorithm A_W, to correct errors in regular languages, which utilizes the fact that the recognizer for a regular language is a FSM. We shall compare Algorithm I and Algorithm A_W, making

use of the assumption that the channel does not delete P consecutive symbols of X_s in transmission (which assumption has been validated in [18]).

Since Algorithm I makes the maximum use of the information contained in the prefixes of the words in H, its complexity is not merely a function of the length of the noisy string, but is also an explicit function of the quantities q_i, defined in Sec. II. To render the comparison between the two algorithms meaningful, we shall compare Algorithm I and Algorithm A_W for identical dictionaries and for identical noisy strings.

Since the dictionary H is finite, and since the processing of the input string is symbol by symbol, the FSM used by A_W to accept H will be identical to the one defined by us. Both Algorithm A_W and Algorithm I perform the distance computation in M stages, where M is the length of the noisy string Y. The number of computations done per stage in either case is the function of the total number of computations done per state of the FSM and the number of states for which the computation must be done.

At the Kth stage of the computation, let $Q^{(K)}$ be the set of states for which the edit distance must be computed by Algorithm A_W. The corresponding set for Algorithm I is $R^{(K)}$. In the case of A_W, even for the processing of y_1 (the first incoming symbol), the distance $D(\alpha/Y^{(1)})$ will be computed for every $\alpha \in V$. This is because Algorithm A_W has no ability to exploit the property of the channel given by Assumption ii of [18]. However, as shown in Lemma I, the set $R^{(K)}$ is always a subset of $H^{(2K)}$, which in turn is always a subset of V. Thus for every stage of the computation, Algorithm A_W computes $D(\alpha/Y^{(K)})$ for a superset of $R^{(K)}$. This fact itself makes Algorithm I computationally less complex than Algorithm A_W.

Further, in the execution of Algorithm I, suppose some state $\alpha \in V$ is an element of $R^{(K)}$. To determine $D_1(\alpha/Y^{(K)})$, the only distances that need to be considered are $D_1(\beta/Y^{(K-1)})$ where β is an element of a set, say $T_{A1}(\alpha)$, defined by

$$T_{A1}(\alpha) = \{\beta \mid \beta \text{ is the } j\text{th order left derivative of } \alpha;\ 0 \leq j \leq P\}.$$

The set $T_{A1}(\alpha)$ will contain at the most $P+1$ terms. Hence, for every $\alpha \in R^{(K)}$, to compute $D_1(\alpha/Y^{(K)})$, only a maximum of $P+1$ computations need be done.

However, if the algorithm A_W is at an identical stage in the processing of Y, for every $\alpha \in Q^{(K)}$, the computation of $D(\alpha/Y^{(K)})$ will involve the distances $D(\beta/Y^{(K-1)})$ for every $\beta \in T_W(\alpha)$, where

$$T_W(\alpha) = \{\beta \mid \beta \text{ is } \textit{any} \text{ left derivative of } \alpha\}.$$

Clearly $T_W(\alpha)$ is a superset of $T_{A1}(\alpha)$ and will contain $|\alpha|+1$ terms. Thus to

compute every $D(\alpha/Y^{(K)})$ Algorithm I will require L additions and minimizations less than Algorithm A_W, where

$$L=\max[|\alpha|-P,0].$$

Combining the above two facts, we conclude that the number of computations required by Algorithm A_W will exceed the number of computations required by Algorithm I, the excess being proportional to:

$$\sum_{K=0}^{M}\sum_{\alpha\in Q^{(K)}}\#(T_W(\alpha))-\sum_{K=0}^{M}\sum_{\alpha\in R^{(K)}}\#(T_{A1}(\alpha)).$$

Algorithm I further optimizes on the number of computations per state per recursion by working in the pseudodistance measure $D_1(\cdot/\cdot)$ and then in the final stage computing the distance $D(\cdot/\cdot)$.

To sum up, Algorithm I is superior to Algorithm A_W because: (1) it takes the maximum advantage of the fact that H is a *finite* dictionary, and thus its recognizer is a *cycle-free* FSM, (2) it optimizes on the fact that the channel obeys Assumption ii of [18], and (3) it does the recursion using the pseudodistance measure, which reduces the number of terms in any one stage of the distance computation to merely three.

IV. POTENTIAL APPLICATIONS OF ALGORITHM I TO CORRECT COMPUTER PROGRAMS

Algorithm I can be readily applied to problems where erroneous strings are to be corrected. One of the problems which it has been used to solve is that of searching a list when the input to the algorithm is an approximate version of one element in the list. It can thus be used to retrieve data from data bases and libraries.

Another area where an algorithm which corrects misspelled strings will be of use is in the correction of computer programs. Considerable literature is available on this topic. Morgan [11] noted that most mispunched errors were a single substitution, deletion, insertion, or interchange of adjacent symbols. Litecky and Davis [8] have indicated that a large percentage of the total errors in COBOL were due to misspelling and errors due to punctuation. Out of 1777 errors studied, at least 574 were due to misspelling, punctuations added or missed out, and other keypunch errors.

As other authors have pointed out, an algorithm such as this can be included to automatically correct errors encountered when the source program is being

compiled. Only when an erroneous string is detected need the error-correcting algorithm be called. Scowen [15] claimed that misspelled identifiers in FORTRAN can be recognized by the system by an initial declaration of all the variables that are encountered in the program. If similarly the names of the functions and subroutines are also included, a finite dictionary H can be created, and using it the algorithms presented here can be used to correct errors in spelling, not only in legal FORTRAN phrases but also in misspelled identifiers, function subprograms, and subroutine names. By including the punctuation symbols too in H [such as (,), ., etc.], misspelled logical operators (.NOT., .OR., .GE., .AND., etc.) and other syntax errors can also be corrected.

Fifty words used in the programming context were used to test Algorithm I. For the purpose of study the confusion matrix was assumed and the sets A_b were subjectively created. The elements of A_b were some of the symbols of A nearest to b on the typewriter keyboard. The details of these sets are given in Table 5.

Using the confusion matrix and the techniques described by Kashyap [4], elementary edit distances related to the probabilities of the individual errors were obtained. The distance associated with deleting a symbol was made equal to the distance associated with inserting one (for lack of any better information). 100 erroneous strings were then fed in as inputs to the algorithm. The average number of errors per string was two, and the maximum was three. In some erroneous strings, all three types of errors (substitution, insertion, and deletion errors) were present. In others, errors of the same type were repeated. But in all the cases, the estimated word X^+ was identical to X_s. Some of these results are tabulated in Table 6.

The error-correcting capability of the distance measure used, which uses *intersymbol* elementary edit distances, can be clearly seen by studying the correction of the misspelled string 'eecode'. 'eecode' can either represent a transmitted 'decode' or a transmitted 'encode'. But since the probability that a typewritten 'n' is mistyped as an 'e' is much smaller than the probability of a 'd'

TABLE 5

Details Regarding the Sets A and A_b for $b \in A$

$A = \{a, b, c, d, e, f, g, i, l, m, n, o, p, r, s, t, u, w, 0\}$	
$A_a = \{a, s, w\}$	$A_b = \{b, c, g, n\}$
$A_c = \{c, d\}$	$A_d = \{c, d, e, f, r, s\}$
$A_e = \{d, e, r, s, w\}$	$A_f = \{c, d, f, g, r, t\}$
$A_g = \{b, f, g, o, t\}$	$A_i = \{i, l, u, 0, o\}$
$A_l = \{i, l, m, o, p\}$	$A_m = \{m, n\}$
$A_n = \{b, m, n\}$	$A_o = \{i, l, o, p, 0\}$
$A_p = \{l, p, o, 0\}$	$A_r = \{e, f, r, t\}$
$A_s = \{a, d, e, s, w\}$	$A_t = \{g, r, t, u\}$
$A_u = \{i, u\}$	$A_w = \{a, e, s, w\}$
$A_0 = \{o, p, 0\}$	

TABLE 6

Erroneous input string	Recognized word	Erroneous input string	Recognized word
ane	and	mpliscit	implicit
ssaign	assign	innut	input
acll	call	inetgar	integer
cmmin	common	loical	logical
cntonud	continue	miann	main
dat	data	nsmelst	namelist
febbog	debug	nott	not
eecode	decode	ogf	off
drffine	define	ot	or
edlete	delete	ooitput	output
demenson	dimension	pattse	pause
dp	do	persision	precision
dobuble	double	pint	print
dinp	dump	prigraam	program
necod	encode	pout	put
ned	end	treed	read
emndflle	endfile	reel	real
er	err	reurnn	return
flase	false	reiwinnd	rewind
illes	file	srop	stop
found	find	bubroutine	subroutine
fromat	format	twwpe	tape
unctioon	function	tarcce	trace
gotd	goto	triu	true
iif	if	weitr	write

being mistyped as an 'e', it is more likely that 'eecode' represents 'decode' than 'encode'. However, we can edit 'encode' to either 'decode' or 'encode' by one substitution. Neither Levenshtein nor weighted Levenshtein distances will be capable of deciding which is the more likely estimate. But since $d(d/e)<d(n/e)$, the word estimated is 'decode', which is the more likely estimate of X_s.

V. CONCLUSIONS

In this paper we have dealt with the problem of estimating a given transmitted string X_s from its noisy version Y. A distance measure $D(X/Y)$ was defined in [18] between $X \in H$ (the finite dictionary) and Y. Using it, X^+, the best estimate of X_s, is defined as that string X which minimizes $D(X/Y)$.

In [18], we have presented an algorithm (Algorithm I) which yielded X^+ as its output without computing every $D(X_i/Y)$ individually. Here, by studying Algorithm I using the tree structure of the FSM that accepts H, we have shown

that it is, in general, computationally less complex than the ST, which obtains

discussed.

REFERENCES

1. A. V. Aho, D. S. Hirschberg, and J. R. Ullmann, Bounds on the complexity of the longest common sub-sequence problem," *J. Assoc. Comput. Mach.* 23(1):1–12 (Jan. 1976).
2. A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, 1972.
3. L. W. Fung and K. S. Fu, Stochastic syntactic decoding for pattern classification, *IEEE Trans. Computers* C-24(6):662–667 (June 1975).
4. R. L. Kashyap, Syntactic decision rules for recognition of spoken words and phrases using a stochastic automaton, *IEEE Trans. Pat. Anal. and Mach. Intel.* PAMI-1(2):154–163 (Apr. 1979).
5. W. J. Masek and M. S. Paterson, A faster algorithm computing string edit distances, *J. Comput. System Sci.* 20:18–31 (1980).
6. D. E. Knuth, *The Art of Computer Programming*, Vol. 1, Addison-Wesley, Reading, Mass., 1973, pp. 473–479.
7. A. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Soviet Physics Dokl.* 10(8):707–710 (Feb. 1966).
8. C. R. Litecky and G. B. Davis, A study of errors, error proneness, and error diagnosis in COBOL, *Comm. ACM* 19(1):33–37 (Jan. 1976).
9. S. Y. Lu and K. S. Fu, A sentence-to-sentence clustering procedure for pattern analysis, in *Proceedings of the IEEE Computer Society 1st International Computer Software and Applications Conference*, 1977, pp. 492–498.
10. R. Lawrence and R. A. Wagner, An extension of the string to string correction problem, *J. Assoc. Comput. Mach.* 22:177–183 (1975).
11. H. L. Morgan, Spelling corrections in systems programs, *Comm. ACM*, 13(2):90–94 (1970).
12. D. L. Neuhoff, The Viterbi algorithm as an aid in text recognition, *IEEE Trans. Information Theory* IT-21(2):222–226 (1975).
13. T. Okuda, E. Tanaka, and T. Kasai, A method of correction of garbled words based on the Levenshtein metric, *IEEE Trans. Computers* C-25:172–177 (Feb. 1976).
14. E. M. Riseman and A. R. Hanson, A contextual post processing system for error correction using binary *n*-grams, *IEEE Trans. Computers* C-23:480–493 (May 1974).
15. R. S. Scowen, On detecting misspelt identifiers in FORTRAN, *Software—Practice and Experience* 7:536 (1977).
16. M. G. Thomason, Errors in regular languages, *IEEE Trans. Computers* 23:597–602 (1974).
17. R. L. Bahl and F. Jelinek, Decoding for channels with insertions, deletions, and substitutions with application to speech recognitions, *IEEE Trans. Information Theory* IT-21:404–411 (1975).
18. 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, *Information Sci.*

19. J. D. Ullman, A binary *N*-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words, *Comput. J.* 20:141–147 (1977).
20. R. A. Wagner, Order-*n* correction for regular languages, *Comm. ACM* 17:265–268 (1974).
21. R. A. Wagner and M. J. Fisher, The string to string correction problem, *J. Assoc. Comput. Mach.* 21:168–173 (1974).
22. C. K. Wong and A. K. Chandra, Bounds for the string editing problem," *J. Assoc. Comput. Mach.* 23:13–16 (Jan 1976).
23. G. Dewey, *Relative Frequency of English Speech Sounds*, Harvard U.P., 1923.

Received July 1980