Yugoslav Journal of Operations Research
7 (1997), Number 1, 97-107

APPLICATION OF THE CONSTRAINED EDIT DISTANCE ALGORITHM TO SEARCH PROCEDURES

Vladimir JOVIČIĆ, Zora KONJOVIĆ
Faculty of Engineering Novi Sad
Computer, Control & Measurements Institute
21000 Novi Sad, Trg Dositeja Obradović a 6
Yugoslavia

Abstract: Searching the sequential file to detect a given substring is a common problem appearing, among others, in text processors and database search systems. One possible approach is OOmmen's constrained edit distance algorithm. The paper contains a brief description of this algorithm and some results of a simulation experiment regarding accuracy and execution speed of the algorithm depending on the probability of the insertion, deletion and substitution errors. The paper also presents one possible practical application of the algorithm to search procedures. Application is based on reduction of the dictionary size according to the probability of editing errors and organization of the contents of the dictionary in a way to so as speed up the search process. Some comparative simulation results are presented illustrating direct and suggested practical applications of the constrained edit distance algorithm to search procedures.

Keywords: Noisy subsequence, constrained edit distance algorithm, search procedures.

1. INTRODUCTION

A common problem in text processing and database search procedures is detection of a given substring in a sequential file. A sequential file can be considered a sequence of words in a finite dictionary without loss in generality. Searching for the appearance of a particular string U by exact matching means determining all the words in the dictionary containing U as a continuous substring.

Unfortunately, very often one faces the problem that the given string U is a noisy one, damaged usually by editing or spelling errors. Let's denote by Y a noisy version of string U. The result of a search using exact matching is a set of words containing noisy string Y, or an empty set even if there exist some words in the dictionary containing the original string U as their continuous substring.

Damareau [1] noticed that most of the errors appearing in strings are actually caused by substitution, deletion, insertion or changing the position of the symbol. He reduced the string comparison problem to edit transformation-based comparisons. Since a position changing transformation can be expressed by a transformation sequence consisting of one deletion and one insertion, decreasing drastically the complexity of the comparison problem, most of the researchers in the field deal only with substitution, deletion and insertion transformations.

Usage of Levenshtein's metrics brought significant improvement in this field [3]. Levenshtein's distance between two given strings is the minimal number of edit transformations required to transform one string to another.

One of the papers based on Levenshtein's distance is [4]. In the paper Oommen considers the problem of noisy subsequences and formulates a correction algorithm based on Levenshtein's distance. The paper contains certain experimental results approving correctness of the algorithm. The same author suggests some improvements to this algorithm in [5, 6] that are of quadratic computational complexity.

In designing and applying string comparison algorithms two basic requests appear. The first one is the accuracy of the algorithm (robustness of the technique concerning the number of errors in the noisy damaged string) that should be the greatest possible. The second one, particularly in on-line processing, is the execution time that should be the shortest possible. When two requests are in contradiction they are resolved in practical applications by reasonable compromise.

The first part of the paper presents a brief description of the algorithm proposed in [5,6] and its direct application to search problems followed by some simulation results intended to illustrate its performances regarding accuracy and execution time efficiency. The second part of the paper presents an improved application of the same algorithm that increases time efficiency keeping accuracy at the same level.

2. DIRECT SEARCH FOR A NOISY SUBSTRING USING CONSTRAINED EDIT DISTANCE

2.1. Constrained edit distance algorithm

The problem solved by the constrained edit distance algorithm can be formulated in the following way. Let us select string X^* from limited known dictionary H. First, some of its symbols are randomly deleted giving a subsequence of string X^* that we shall denote by U. Then string U passes through the communication channel where some insertion, substitution and deletion errors occur giving the noisy version of string U denoted by Y. The receiver receives string Y. The aim of the algorithm is to recognize string X^* knowing string Y and dictionary H.

The result of the algorithm is a real number corresponding to Levenshtein's distance between two strings under constrained editing. Executing the algorithm supposes knowing some characteristics of the communication channel. These characteristics are the probability of symbol insertion, the probability of symbol deletion and the probability that some symbol is going to be replaced by some other symbol from the limited alphabet over which the string is constructed.

In the algorithm a three-dimensional matrix W under constrained editing is calculated using the dynamic programming method. From matrix W one can directly obtain the value of the constrained edit distance for these two strings. The computational complexity of the procedure directly calculating this matrix is cubic. Applying some constraints obtained from the statistical characteristics of the communication channel leads to a modification of the algorithm reducing computational complexity to a quadratic one without loss in accuracy [5,6].

Any modification of or damage to the string occurring over a single symbol can be considered an elementary edit operation. These operations are insertion, deletion and substitution of the symbol since the permutation of two symbols can be reduced to successive deletion and insertion or two substitutions. Elementary edit distance (d) is assigned to each elementary edit operation. Elementary edit distance is a function of two variables and according to the number of possible edit operations it appears in three forms

$d(x_i,y_j)$- Distance corresponding to replacing x_i by y_j, $x_i,y_j \in A$

$d(x_i,\Theta)$ - Distance corresponding to deleting x_i, $x_i \in A$

$d(\Theta,y_j)$ - Distance corresponding to inserting y_j, $y_j \in A$

The strings under comparison X and Y consist of symbols belonging to alphabet A, while Θ is an empty symbol that does not belong to A. Indices i and j represent the position of the symbol in the strings. The symbol Θ is used to equalize the lengths of the strings under comparison. All possible transformations of string X to string Y are given by limited set $\Gamma(X,Y)$. Its cardinality is

$$\left|\Gamma(X,Y)\right| = \sum_{k=Max[0,|Y|-|X|]}^{|Y|} \left[\frac{(|X|+k)!}{k!(|Y|-k)!(|X|-|Y|+k)!} \right] \tag{1}$$

where $|X|$ and $|Y|$ are the lengths of strings X and Y' respectively. It is easy to notice that the number of elements in this set depends only on the lengths of the strings under comparison. To each element of the set $\Gamma(X,Y)$ one can assign the value of the sum of the constrained edit distances corresponding to the edit operations required to transform string X to string Y.

Since the general Levenshtein distance $D(X,Y)$ between strings X and Y represents the minimum of the sum of the edit distances assigned to edit operations required to transform X to Y, it is given by the expression

$$D(X,Y)= \underset{(X,Y)\in\Gamma(X,Y)}{Min}\left[\sum_{i=1}^{|X'|}\left\{d\left(x_i',y_i'\right)\right]\right. \tag{2}$$

It is obvious that set $\Gamma(X,Y)$ contains a large number of elements. To decrease the necessary computation one introduces some constraints. These constraints can be classified in two categories. The first category covers the constraints caused by the impossibility of carrying out some edit transformation due to a difference in strings lengths. The second one contains the constraints imposed by the properties of the communication channel and they represent the maximal number of specific operations that can be performed on a string.

The final version of the algorithm based on constrained edit distances is obtained after observing some important properties of matrix W that enable the computation of four two-dimensional matrices instead of three-dimensional matrix [5,6].

2.2. Direct search procedure using the constrained edit distance algorithm

The application of this algorithm to search procedures reduces naturally to determining the string from the dictionary that has the minimal constrained edit distance from the input string. The simplest and most direct application is given by the following pseudocode.

Algorithm SearchForString

Input: (i) Limited dictionary H, L - expected number of insertions occurring in transfer.

(ii) Y - noisy version of unknown string X^* from dictionary H.

Output: String X^+ with minimal edit distance from string X^*.

Method:

For each string $X \in H$ do
Begin
If L is a feasible value Then
$T = L$
Else
T = closest feasible value L
ConstrainedDistance $(X,Y,d,\tau,D_{\{T\}}(X,Y))$
$X^+ = X$
End
End of Algorithm SearchForString

The detailed algorithm of the procedure ConstrainedDistance is given in [2].

Some simulations have been carried out using this program to indicate the performance of this algorithm. According to practical requirements tests are related to accuracy and execution time. The results of the first group of tests are expressed by the

bounds of the maximal number of errors per string allowing the algorithm to produce exact search results. The second group of tests relates to determining the dependency of the execution time on the probability of a particular class of errors. Tests have been carried out on a dictionary containing 100 strings representing words in the Serbian language and using a PC486 computer.

Simulation results showing the accuracy of the algorithm for given examples are summarized in Table 1.

Table 1. Accuracy of the algorithm

Test No	Insertion probability	Deletion probability	Substitution probability	Average number of errors/string	Number of "hits" in 500 tests	Accuracy (%)
1	0.03	0.5	0.1	44.7680	491	98.20
2	0.03	0.1	0.5	44.9280	500	100.00
3	0.20	0.3	0.2	50.1640	496	99.20
4	0.03	0.6	0.1	52.2300	292	58.40
5	0.03	0.1	0.6	52.3755	493	98.59
6	0.03	0.1	0.7	58.9360	396	79.20
7	0.03	0.3	0.5	59.4000	247	49.40
8	0.03	0.2	0.6	59.7900	310	62.00

The results indicate that accuracy does not depend directly on the average number of errors per string (see rows 4 and 5). From this experiment one could conclude that the algorithm is more sensitive to the deletion than to the substitution of symbols (see rows 4 and 5 as well as 7 and 8).

The general conclusion that could be derived from the above results is that the algorithm works correctly when the string contains not more than 50% damaged symbols.

Execution time is analyzed with respect to the influence of one of the following probabilities: deletion probability, insertion probability and substitution probability. Experiments were carried out so as to vary one of these probabilities keeping the other probabilities at minimal value 0.01. Relative execution times (with respect to execution time for $P_{insertion} = P_{substitution} = P_{deletion} = 0.01$) are presented in the following Tables.

Table 2. Dependence of execution time on deletion probability

$P_{deletion}$	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40
Relative execution time	1.00	0.95	0.92	0.89	0.87	0.84	0.82	0.79	0.76

Table 3. Dependence of execution time on substitution probability

$P_{substitution}$	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40
Relative execution time	1.00	0.95	0.92	0.89	0.87	0.84	0.82	0.79	0.76

Table 4. Dependence of execution time on insertion probability

$P_{insertion}$	0.01	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40
Relative execution time	1.00	0.95	0.92	0.89	0.87	0.84	0.82	0.79	0.76

3. IMPROVED SEARCH PROCEDURE BASED ON THE CONSTRAINED EDIT DISTANCE ALGORITHM AND ITS PERFORMANCES

3.1. Improved search procedure

The basic assumption for the improved application of the constrained edit distance algorithm is as follows.

The probability that a certain string from the dictionary is the original of the noisy string decreases with increasing difference in the length between the noisy string and the string from the dictionary. This assumption has an additional condition when applying the constrained edit distance algorithm that allows a comparison of the given string and a string from the dictionary only if the absolute value of this difference belongs to a fixed interval. The bounds of this interval can be simply determined according to deletion and insertion probabilities. The lower bound of the interval can be determined as the average length of the strings from the dictionary multiplied by the deletion probability and by a certain constant value. The upper bound of the interval can be determined as the average length of the strings from the dictionary multiplied by the insertion probability and also by a certain constant value. The multiplication constant can be determined experimentally for a particular dictionary to satisfy the compromise between accuracy and execution speed.

With these assumptions, it is possible to construct the search procedure so that the number of strings to be compared (to compute the constrained edit distance) with the noisy string is decreased, but this procedure still accesses all the strings in the dictionary.

If the strings in the dictionary are sorted by length in ascending order, it is possible to apply a binary search and find the shortest string in the dictionary satisfying the condition stated by the first assumption. Then, all the strings below a the upper bound are compared with the noisy string. The algorithm is given by the following pseudocode.

ALGORITHM FindString $(X[n], \text{aver\_length}, \text{const}, Y, p_{insertion}, p_{deletion}, p_{substitution}, (X_{orig})$

Input: $X[n]$ - Dictionary of n strings sorted by length in ascending order

aver_length - Average length of the strings in the dictionary

const - Experimentally determined constant

Y - Noisy substring
$p_{insertion}, p_{deletion}, p_{substitution}$ - Error probabilities

Output: X_{orig} - Original string from the dictionary

Constants: MaximumDistance - Maximal expected distance

Method: L = aver_length* $p_{insertion}$
upper = const*aver_length* $p_{deletion}$
lower = const*aver_length* $p_{insertion}$

For $i = 1$ to n
distance[i] = MaximumDistance
By binary search find q - the first index satisfying:
$length(X[q]) = (length(Y) - \text{upper})$
For $i = q$ until $length\ (X[i]) < (\text{length}(Y) + \text{lower}))$ do
StartForBlock
If $L \le length\ (Y) \wedge (L \ge max\,(0, length(Y) - length(X[i])))$
$T = L$
else
If $L >$ length(Y)
T = length(Y)
else
$T = \max(0, \text{length}(Y) - \text{length}(X[i]))$
distance[i] = ConstrainedDistance ($X[i], Y, p_{insertion}, p_{deletion}, p_{substitution}, T$)
EndForBlock
$X_{orig} = X[i]$ satisfying distance[i] = $\underset{j \in [0,n]}{Min}$ distance [j]
End

END OF ALGORITHM FindString.

3.2. Performances of the improved search procedure

In order to apply the suggested search procedure it was necessary to determine the value of the constant const to satisfy the compromise between accuracy and execution speed.

In this paper we determined experimentally that this value should be within the interval [2,2.5]. For the experiment we used dictionaries containing 200 strings with a normal distribution. The average lengths of the strings in the dictionaries were

15 and 10 with standard deviation 1, 2, 3 and 4. The results of the experiment are shown in following tables.

Table 6. Simulation results for $P_{insertion} = P_{substitution} = P_{deletion} = 0.10$, Average string length = 15

Deviation	4		3		2		1	
Const	Correct	Strings accessed	Correct	Strings accessed	Correct	Strings accessed	Correct	Strings accessed
Original	88.60%	200	93.50%	200	92.7%	200	88.8%	200
1.00	52.20%	25.89	71.20%	69.54	70.20%	91.50	43.10%	69.80
1.50	68.60%	50.46	73.00%	69.59	71.50%	92.51	65.80%	122.89
2.00	78.00%	74.05	83.90%	100.03	82.60%	124.39	78.80%	162.83
2.50	87.40%	94.54	90.80%	123.15	90.50%	152.08	85.50%	179.04
3.00	85.30%	94.95	93.30%	144.97	92.70%	173.44	86.50%	181.84
3.50	90.80%	112.73	93.10%	143.06	93.90%	172.24	87.50%	193.30
4.00	89.90%	131.90	94.40%	162.02	94.40%	183.37	89.50%	197.68
4.50	89.70%	147.24	93.00%	174.76	94.40%	190.64	87.70%	199.08
5.00	88.70%	147.22	93.10%	182.31	93.40%	195.31	88.70%	199.44
5.50	89.60%	158.91	93.50%	184.16	93.90%	192.88	89.70%	199.90

Table 7. Simulation results for $P_{insertion} = P_{substitution} = P_{deletion} = 0.15$, Average string length = 15

Deviation	4		3		2		1	
Const	Correct	Strings accessed	Correct	Strings accessed	rrect	Strings accessed	Correct	Strings accessed
Original	86.80%	200	86.50%	200	88.00%	200	88.5%	200
1.00	50.50%	45.73	47.40%	57.68	49.50%	78.49	41.80%	99.01
1.50	70.20%	67.41	69.20%	84.81	70.30%	110.76	68.90%	142.60
2.00	81.40%	89.67	82.30%	105.67	86.10%	159.24	80.60%	168.44
2.50	85.30%	108.06	87.50%	147.65	88.20%	174.47	88.10%	195.38
3.00	87.40%	140.57	87.80%	160.84	88.50%	187.13	89.10%	198.65
3.50	86.90%	151.71	86.70%	174.02	88.90%	193.47	89.20%	199.78
4.00	86.00%	163.64	85.70%	182.14	87.80%	196.60	89.00%	199.96
4.50	87.10%	172.31	87.80%	188.74	88.30%	198.77	86.90%	200.00
5.00	87.40%	178.76	87.30%	192.74	88.00%	199.74	87.60%	200.00
5.50	85.10%	184.62	86.90%	195.27	87.20%	199.77	88.20%	200.00

Table 8. Simulation results for $P_{insertion} = P_{substitution} = P_{deletion} = 0.10$, Average string length = 10

Deviation	4		3		2		1	
Const	Correct	Strings accessed	Correct	Strings accessed	Correct %	Strings accessed	Correct %	Strings accessed
Original	88.00%	200	87.40%	200	86.60%	200	86.00%	200
1.00	44.90%	25.71	62.70%	36.54	61.40%	54.17	61.30%	80.33
1.50	68.60%	49.68	62.90%	35.77	66.00%	55.78	63.50%	79.86
2.00	80.40%	73.70	79.60%	68.99	80.40%	100.05	79.30%	139.94
2.50	88.40%	95.47	80.60%	67.73	79.30%	97.42	77.20%	135.20
3.00	86.50%	94.54	85.60%	98.90	84.30%	139.73	83.20%	174.80
3.50	88.30%	116.90	83.70%	99.72	84.60%	138.15	85.10%	174.66
4.00	89.10%	132.81	87.50%	126.75	85.90%	164.75	87.90%	189.93
4.50	88.80%	147.45	88.20%	125.86	86.70%	163.94	89.00%	190.44
5.00	89.20%	147.00	86.80%	144.85	85.90%	180.15	88.60%	195.56
5.50	87.40%	156.36	87.80%	144.41	87.90%	180.92	87.70%	197.25

Table 9. Simulation results for $P_{insertion} = P_{substitution} = P_{deletion} = 0.15$, Average string length = 10

Deviation	4		3		2		1	
Const	Correct	Strings accessed	Correct	Strings accessed	Correct	Strings accessed	Correct	Strings accessed
Original	75.80%	200	77.60%	200	76.00%	200	76.30%	200
1.00	40.80%	23.41	41.90%	36.27	41.00%	44.71	38.90%	72.93
1.50	55.60%	47.84	61.70%	73.62	57.60%	82.35	55.70%	123.97
2.00	69.00%	68.89	71.40%	104.23	71.00%	117.45	74.00%	167.05
2.50	73.30%	88.68	75.20%	127.68	76.50%	144.19	77.80%	184.52
3.00	73.00%	89.80	77.50%	127.45	75.70%	143.34	75.30%	183.41
3.50	72.80%	106.07	77.80%	150.76	79.40%	164.66	78.10%	193.92
4.00	73.80%	124.32	78.50%	165.95	77.70%	178.58	75.60%	197.76
4.50	74.60%	126.33	77.40%	177.47	74.90%	187.97	76.20%	199.67
5.00	73.30%	139.52	77.60%	179.72	76.90%	187.58	75.70%	199.64
5.50	74.30%	155.54	75.50%	186.84	75.60%	194.33	78.30%	199.88

Since the main goal of the suggested application was to improve the time efficiency of the search procedure based on constrained edit distance, some tests have been carried out to illustrate the level of improvement in time efficiency keeping the same level of accuracy.

To the indicate possible application of the search procedure, tests were carried out on a dictionary containing 100 strings (Dictionary 1) and a dictionary containing 366 strings (Dictionary 2). Both dictionaries were constructed with no particular attention paid to their statistical characteristics. For this experiment the value of the constant was set at 2.5. The results are summarized in the following Tables.

Table 10. Dictionary 1 (100 strings, average string length: 71.2)

	Pins = 0.03 Psub = 0.03 Pdel = 0.03		Pins = 0.03 Psub = 0.03 Pdel = 0.05		Pins = 0.03 Psub = 0.05 Pdel = 0.03		Pins = 0.05 Psub = 0.03 Pdel = 0.03	
	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)
Direct	100	89.6	100	98.7	100	91.1	100	75.8
Improved	53.73	95.3	56.94	90.0	48.6	85.9	64.29	80.4

Table 11. Dictionary 2 (366 strings, average string length: 45.6)

	Pins = 0.03 Psub = 0.03 Pdel = 0.03		Pins = 0.03 Psub = 0.03 Pdel = 0.05		Pins = 0.03 Psub = 0.05 Pdel = 0.03		Pins = 0.05 Psub = 0.03 Pdel = 0.03	
	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)	Strings accessed	Accuracy (%)
Direct	366	69.5	366	78	366	77.4	366	68.9
Improved	34.0	90.5	46	84.9	31.7	85.3	45.5	82.3

From the presented results one can see that the suggested implementation shows a significant advantage when applied to larger dictionaries, while application to smaller dictionaries keeps performances close to direct application.

4. CONCLUSION

This paper presents one possible practical application of the constrained edit based algorithm to search procedures. Some experiments were carried out to analyze algorithm properties that are of particular importance in practical applications: accuracy and time efficiency.

From the simulation results we conclude that the algorithm is characterized by very good properties regarding accuracy - the algorithm produces correct results in 99% of the cases if the string contains up to 50% damaged symbols.

The main drawback of the algorithm is its computational complexity and quite large memory requirements for large dictionaries.

Based on the performed analysis we suggest one possible practical implementation of the algorithm in search procedures. The main idea of the application is to reduce the size of the dictionary assuming that lengths of strings reflect their similarity and to improve the search using that assumption and proper organization of the dictionary.

An important problem that still exists is determination of the multiplication constant used in the suggested application. In the paper this constant is determined by simulation. Since both accuracy and efficiency strongly depend on this constant, one should employ some more exact approaches in determining its value .

REFERENCES

[1] Damareau, F., J., "A technique for computer detection and correction of spelling errors", *Commun. ACM,* 7 (1964) 171-176.

[2] Jovičić, V., "Constrained edit distance algorithm and its application in Library Information Systems", Faculty of Technical Sciences, Novi Sad, 1994 (in Serbian).

[3] Levenshtein, A., "Binary codes capable of correcting deletions, insertions and reversals", *Sov. Phy. Dokl.,* 10 (1966) 707-710.

[4] Oommen, B., J., "Constrained string editing", *Inform. Sci,* 40 (1986) 267-284.

[5] Oommen, B., J., " Recognition of noisy subsequences using constrained edit distances ", *IEEE Trans. Pattern Anal. Mach. Intell.,* 9 (1987) 676-685.

[6] Oommen, B., J., "Correction to ' Recognition of noisy subsequences using constrained edit distances'", *IEEE Trans. Pattern Anal. Mach. Intell.,* 10 (1988) 983-984.