A Fast Algorithm for Finding the Nearest Neighbor of a Word in a Dictionary

Horst Bunke

IAM-93-025

November 1993

A Fast Algorithm for Finding the Nearest Neighbor of a Word in a Dictionary

Horst Bunke

Abstract

In this paper a new algorithm for string edit distance computation is proposed. It is based on the classical approach [11]. However, while in [11] the two strings to be compared may be given online, our algorithm assumes that one of the two strings to be compared is a dictionary entry that is known a priori. This dictionary word is converted, in an off-line phase to be carried out beforehand, into a special type of deterministic finite state automaton. Now, given an input string corresponding to a word with possible OCR errors and the automaton derived from the dictionary word, the computation of the edit distance between the two strings corresponds to a traversal of the states of the automaton. This procedure needs time which is only linear in the length of the OCR word. It is independent of the length of the dictionary word. Given not only one but N different dictionary words, their corresponding automata can be combined into a single deterministic finite state automaton. Thus the computation of the edit distance between the input word and each dictionary entry, and the determination of the nearest neighbor in the dictionary need time that is only linear in the length of the OCR word. Particularly, the time complexity is independent of the size of the dictionary.

Categories and Subject Descriptors: I.7.0 [Text Processing]: General; I.7.1 [Text Processing]: Text Editing; H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing - *dictionaries*

General Terms: algorithms, computational complexity

Additional Key Words and Phrases: string edit distance, finite state automaton, nearest neighbor search, dictionary lookup.

1 Introduction

The field of optical character recognition (OCR) has progressed very rapidly over the last thirty years [1-4]. But today even with the most sophisticated OCR methodology, a recognition rate of 100% cannot be reached. In order to improve the overall performance of an automatic reading device, nevertheless, the application of post-processing techniques using contextual information is considered very useful. A recent survey of contextual postprocessing methods has been given in [5]. For an earlier collection of papers addressing the same problem domain see [6].

There are different categories of contextual postprocessing methods. One class of methods is based on n-gram statistics [7,8]. Such methods rely on transition probabilities between consecutive letters of a word. As an advantage of n-gram based methods, these transition probabilities can be determined off-line, for example, from a dictionary of legal words. Consequently, the run time needed online for error detection or correction is independent of the size of the dictionary. On the other hand, n-gram statistics represent only part of the information present in a dictionary and may thus not lead to a performance as good as that of methods that use a full dictionary.

Another category of postprocessing methods is based on dictionary search [9,10]. If a word output by an OCR device is present in the dictionary it is assumed that all its characters have been correctly recognized. If it is not contained, then the most similar dictionary entries are retrieved and considered as candidates. Dictionary based methods are generally considered computationally expensive. Therefore, efficient dictionary organization by means of trie or hash data structures is essential. Also, assumptions about the type and the number of errors are usually made in order to make the problem computationally tractable.

A fundamental concept in dictionary based postprocessing of OCR output is string edit distance, also known as generalized Levenshtein distance. This concept is based on a set of elementary edit operations, for example, the insertion, substitution, and deletion of a single symbol in a word. In order to model the fact that certain errors may be more likely than others, costs of elementary edit operations can be defined. The string edit, or generalized Levenshtein distance of two strings of symbols corresponds to the minimum cost sequence of elementary edit operations that make the two strings identical. The classical reference for edit distance computation of two words is the Wagner & Fischer algorithm [11]. This algorithm is based on dynamic programming. It has a time complexity of $O(n \cdot m)$ where n and m are the lengths of the two strings under comparison. Improving the time complexity of the Wagner & Fischer algorithm has been a major subject of research for many years. Two algorithm with a better assymptotical time complexity have been published [12,13]. A general discussion of string edit distance including various applications is contained in [14]. For a resent survey see [15]. Stochastic versions of string matching and their applications to the correction of distorted words have been described in [16,17]. One of the apparent problems in the application of string

edit distance to the correction of OCR-output is the high computational complexity. This problem is particularly serious if the underlying dictionary is large. Then some preselection technique is indispensible in order to quickly determine a small set of potential candidate words from the dictionary [18].

In this paper a new algorithm for string edit distance computation is proposed. It is based on the classical approach [11]. However, while in [11] the two strings to be compared may be given online, our algorithm assumes that one of the two strings to be compared is a dictionary entry that is known a priori. This dictionary word is converted, in an off-line phase to be carried out beforehand, into a special type of deterministic finite state automaton. Now, given an input string corresponding to a word with possible OCR errors and the automaton derived from the dictionary word, the computation of the edit distance between the two strings corresponds to a traversal of the states of the automaton. This procedure needs time which is only linear in the length of the OCR word. It is independent of the length of the dictionary word. Given not only one but N different dictionary words, their corresponding automata can be combined into a single deterministic finite state automaton. Thus the computation of the edit distance between the input word and each dictionary entry, and the determination of the nearest neighbor in the dictionary need time that is only linear in the length of the OCR word. Particularly, the time complexity is independent of the size of the dictionary.

In this paper we describe the new algorithm and prove its correctness. Also, we analyze its computational complexity. Potential applications are beyond the scope of this paper. However, the practical use of the algorithm in the context of a system for reading check forms [19] is currently under investigation. The rest of this paper is organized in the following way. Section 2 introduces the basic terminology and briefly reviews the Wagner & Fischer algorithm. Then, some fundamental definitions and basic properties will be given in section 3. The new algorithm for edit distance computation between an input string and one dictionary word will be presented in section 4. Next, section 5 shows the extension to a dictionary of any size. This includes the determination of the nearest neighbor of a given word in a dictionary. Finally, section 6 provides a summary and further remarks.

2 Preliminaries

We consider strings $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ over a finite alphabet V. The empty string is denoted by ϵ. An edit operation is a pair (a, b), also denoted by $a \rightarrow b$, where $a, b \in V \cup \{\epsilon\}$, $(a, b) \neq (\epsilon, \epsilon)$, $a \neq b$. String B results from string A through the edit operation $a \rightarrow b$ if $A = xay$ and $B = xby$ for some strings x and y over V. We call $a \rightarrow b$ a substitution if $a \neq \epsilon$ and $b \neq \epsilon$, a deletion if $b = \epsilon$, and an insertion if $a = \epsilon$. A sequence E of edit operations will be called an edit sequence. Let $E = e_1, e_2, \ldots, e_k$ be an edit sequence. We say that B is derivable from A via E if there is a sequence $D_0, D_1, \ldots, D_k$ of strings such that $A = D_0$, $B = D_k$ and D_i

results from D_{i-1} through e_i, $i = 1, \ldots, k$.

A cost function c is a function assigning a non-negative real number $c(a \to b)$ to each edit operation $a \to b$. The cost of a sequence $E = e_1, e_2, \ldots, e_k$ of edit operations is defined by

$$c(E) = \sum_{i=1}^{k} c(e_i).$$

Now, the edit distance of strings A and B is defined by

$$d(A, B) = min\{c(E) | B \text{ is derivable from } A \text{ via } E\}.$$

An algorithm for computing the string edit distance $d(A, B)$ was given by Wagner & Fischer [11]. This algorithm plays a fundamental role for the method described in this paper. It will be called *basic algorithm* in the rest of this paper. Next, we give a brief review of this algorithm.

In the basic string edit distance computation algorithm, two strings $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ are considered. An edit matrix $D(i, j)$ of dimension $(n + 1) \times (m + 1)$ with indices running from 0 to n, and 0 to m is constructed. The first row and column are simply computed by

$$D(0, 0) = 0,\ D(0, j) = D(0, j - 1) + c(\epsilon \to b_j),\ D(i, 0) = D(i - 1, 0) + c(a_i \to \epsilon).$$

For the interior elements, the following recursion formula is used

$$D(i, j) = min\{D(i-1, j-1)+c(a_i \to b_j), D(i-1, j)+c(a_i \to \epsilon), D(i, j-1)+c(\epsilon \to b_j)\}. \quad (1)$$

It was shown that $d(A, B) = D(n, m)$ [11]. The basic algorithm uses $O(n \cdot m)$ time and space.

Throughout sections 3, 4, and 5 of this paper, we will restrict our considerations to the following cost function

$$c(a \to b) = c(a \to \epsilon) = c(\epsilon \to a) = 1 \quad \text{for any} \quad a, b \in V. \quad (2)$$

An extension to more general cost functions will be discussed in section 6.

3 Basic Properties and Definitions

Throughout this section, we consider strings $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ together with the edit matrix $D(i, j)$ of the basic algorithm. First, we show that the absolute difference of two consecutive elements in a row or column of the edit matrix is never greater than one.

Lemma 3.1 *Let $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ be two strings and $D(i, j)$ the corresponding edit matrix of the basic algorithm. Then*

a) $|D(i, j) - D(i, j + 1)| \leq 1$ for any $i = 0, 1, \ldots, n$; $j = 0, 1, \ldots, m - 1$.

b) $|D(i, j) - D(i + 1, j)| \leq 1$ for any $i = 0, 1, \ldots, n - 1$; $j = 0, 1, \ldots, m$.

Proof: The proof is by induction. Properties a) and b) are trivially true for the initial row and column of the edit matrix. Lets assume that the properties hold true for any row $t = 0, 1, \ldots, i$ and any column $t = 0, 1, \ldots, j$. Now consider $D(i+1, j+1)$. For cost function (2) nine different cases can occur:
1. $D(i+1,j) = D(i,j) = D(i,j+1)$
2. $D(i+1,j) = D(i,j) = D(i,j+1) - 1$
3. $D(i+1,j) = D(i,j) = D(i,j+1) + 1$
4. $D(i+1,j) - 1 = D(i,j) = D(i,j+1)$
5. $D(i+1,j) - 1 = D(i,j) = D(i,j+1) - 1$
6. $D(i+1,j) - 1 = D(i,j) = D(i,j+1) + 1$
7. $D(i+1,j) + 1 = D(i,j) = D(i,j+1)$
8. $D(i+1,j) + 1 = D(i,j) = D(i,j+1) - 1$
9. $D(i+1,j) + 1 = D(i,j) = D(i,j+1) + 1$

For each of the cases it is easy to show that (1) implies that the absolute differences $|D(i+1,j+1) - D(i,j+1)|$ and $|D(i+1,j+1) - D(i+1,j)|$ are never greater than one. □

Definition 3.2 *Let* $C(j) = (D(0,j), D(1,j), \ldots, D(n,j))_t$ *and* $C(k) = (D(0,k), D(1,k), \ldots, D(n,k))_t$ *be two columns of the edit matrix of the basic algorithm,* $0 \leq j, k \leq m$. *The columns* $C(j)$ *and* $C(k)$ *are* similar *if* $D(i,j) = D(i,k) + l$ *for some integer* l *and* $i = 0, 1, \ldots, n$. □

As an example, consider the edit matrix in Fig. 1, where the columns 3 and 4 are similar.

		0	1	0	0
	0	1	2	3	4
0	1	0	1	2	3
1	2	1	0	1	2
1	3	2	1	1	2

Fig. 1: The edit matrix for $A = 011$ and $B = 0100$ under cost function (2).

Lemma 3.3 *Let* $C(j)$ *and* $C(k)$ *be two similar columns in the edit matrix of the basic algorithm;* $0 \leq j, k \leq m-1$. *If* $b_{j+1} = b_{k+1}$, *then the columns* $C(j+1)$ *and* $C(k+1)$ *are also similar.*

Proof: The proof is by induction. Under cost function (2) we know that $D(0, j+1) = D(0,j) + 1$ and $D(0, k+1) = D(0,k) + 1$. As $D(0,j) + l = D(0,k)$ for some integer l we conclude that $D(0, j+1) + l = D(0, k+1)$. Hence, lets assume that $D(t,j) + l = D(t,k)$ for $t = 0, 1, \ldots, i$. From the basic algorithm it follows that $D(i+1, j+1) = \min\{D(i,j), D(i-1,j), D(i,j-1)\}$ if $a_{i+1} = b_{j+1}$ and

$D(i+1,j+1) = \min\{D(i,j)+c(a_{i+1} \to b_{j+1}), D(i-1,j), D(i,j-1)\}$ if $a_{i+1} \neq b_{j+1}$.
Similary,
$D(i+1,k+1) = \min\{D(i,k), D(i-1,k), D(i,k-1)\}$ if $a_{i+1} = b_{k+1}$ and
$D(i+1,k+1) = \min\{D(i,k)+c(a_{i+1} \to b_{k+1}), D(i-1,k), D(i,k-1)\}$ if $a_{i+1} \neq b_{k+1}$.
As $b_{k+1} = b_{j+1}$, $D(i,j)+l = D(i,k)$, $D(i-1,j)+l = D(i-1,k)$, $D(i,j-1)+l = D(i,k-1)$, and $c(a_{i+1} \to b_{j+1}) = c(a_{i+1}, b_{k+1})$, we conclude in either case $a_{i+1} = b_{j+1}$ or $a_{i+1} \neq b_{j+1}$ that $D(i+1,k+1) = D(i+1,j+1)+l$. □

Definition 3.4 *Let $C(j)$ be a column of the edit matrix of the basic algorithm. Its* type *$T(C(j))$ is an n-tupel, $T(C(j)) = (t_1, t_2, \ldots, t_n)$ where $t_i = D(i,j) - D(i-1,j)$; $i = 1, \ldots, n$. $C(j)$ is called an* instance *of $T(C(j))$.* □

As an example, in Fig. 1 we have $T(C(0)) = (1,1,1)$, $T(C(1)) = (-1,1,1)$, $T(C(2)) = (-1,-1,1)$, and $T(C(3)) = T(C(4)) = (-1,-1,0)$. The column $(0,1,2,3)$ is an instance of $(1,1,1)$, and both $(3,2,1,1)$ and $(4,3,2,2)$ are instances of $(-1,-1,0)$.

As we know from Lemma 3.1 that the difference between two successive elements in a column is not greater than one, we can conclude that the t_i's in Definition 3.4 can take on only values from the set {1,-1,0} for any edit matrix. From this observation, the following Corollary follows.

Corollary 3.5 *Let $A = a_1 a_2 \ldots a_n$ and $B = b_1 b_2 \ldots b_m$ be two strings and $D(i,j)$ the corresponding edit matrix of the basic algorithm. Then there are at most 3^n different types of columns in $D(i,j)$.* □

The next Corollary is an immediate consequence of Definitions 3.2 and 3.4.

Corollary 3.6 *If two columns $C(j)$ and $C(k)$ are similar, then $T(C(j)) = T(C(k))$ and vice versa.* □

Let us consider an edit matrix with columns $C(j)$ and $C(j+1)$, $j = 0, 1, \ldots, m-1$. In the rest of this paper we will use the notation $C(j+1) = S(C(j), b_{j+1})$ to indicate that $C(j+1)$ is the successor column of $C(j)$ under symbol b_{j+1}. Similarly, we write $C(j) = S(C(j), \epsilon)$ and $C(j+l) = S(C(j), b_{j+1} \ldots b_{j+l})$. With this notation, Lemma 3.3 can be reformulated in the following way.

Corollary 3.7 *Let $C(j)$, $C(j+1)$, $C(k)$, $C(k+1)$ be columns in an edit matrix with $S(C(j), b) = C(j+1)$, $S(C(k), b) = C(k+1)$ and $T(C(j)) = T(C(k))$ for some $b \in V$. Then $T(C(j+1)) = T(C(k+1))$.* □

4 A Class of Finite State Automata for String Edit Distance Computation

The problem considered in this section is the construction of a finite state automaton that computes the string edit distance $d(A, B)$ between a prototype string A, which

is given a priori, and an input string B. Let V be the underlying finite alphabet of symbols.

Definition 4.1 *Let $A = a_1 a_2 \ldots a_n$ be a string over V. A* string edit distance computation finite state automaton *for A, or SED-automaton for short, is a 3-tuple*

$\Delta(A) = (Q, I, \delta)$ *where*

- Q is the finite set of states
- $I \in Q$ is the initial state
- $\delta : Q \times V \to Q$ is the state transition function.

The components Q, I, and δ are defined in the following way:

1. *Q is the set of 3^n different types of columns in an edit matrix with $n+1$ rows.*
2. *$I = (1, 1, \ldots, 1)$ (n-times)*
3. *For any $q, q' \in Q$ and $b \in V$ we set $\delta(q, b) = q'$ if there exists a column $C(j)$ in an edit matrix with n rows such that $T(C(j)) = q$, $S(C(j), b) = C(j+1)$, and $T(C(j+1)) = q'$.* □

Lemma 4.2 *Let $\Delta(A)$ be the SED-automaton for a string $A = a_1 a_2 \ldots a_n$. The state transition function $\delta(q, b)$ is uniquely determined for any $q \in Q$ and $b \in V$.*

Proof: The proof is by contradiction. Assume there exist $\delta_1(q, b)$ and $\delta_2(q, b)$ such that $\delta_1(q, b) \neq \delta_2(q, b)$. Then there exist different pairs of columns, $C(j)$, $C(j+1)$, and $C(j')$, $C(j'+1)$ with $S(C(j), b) = C(j+1)$, $S(C(j'), b) = C(j'+1)$ such that $T(C(j)) = T(C(j')) = q$ and $T(C(j+1)) \neq T(C(j'+1))$. Clearly, this is contradictory to Corollary 3.7. □

In the procedural construction of an SED-automaton for a given string A, we have to compute a number of columns of an $(n+1)$-row edit matrix using the basic algorithm. We start with a row consisting of $n+1$ elements that is an instance of the initial state I. An example of such an instance is $(0, 1, \ldots, n)_t$. Then we compute, by application of the basic algorithm, the successor column of $(0, 1, \ldots, n)_t$ under each symbol of V and determine its type. This yields the immediate successor states of the initial state I. For each of these successor states we take again an instance and compute the successor column under each symbol of V by the basic algorithm. This gives us the successors of the successors of the initial state. The procedure is repeated as long as we can reach new states, i.e., as long as new types of columns are generated by the basic algorithm. The number of states of the SED-automaton is at most 3^n according to Corollary 3.5. Therefore, the construction procedure is guaranteed to terminate. If there are states in the SED-automaton that cannot be reached from the initial state, they may be deleted. The instance that is chosen for

a given column type, i.e., state of the SED-automaton, is completely arbitrary as all possible instances are similar to each other. Hence, the successor state under a certain symbol is always uniquely determinied (see Lemma 4.2).

Example 4.3 Let $V = \{0, 1\}$ and $A = 001$. Then $\Delta(A)$ is shown in Fig. 2. □

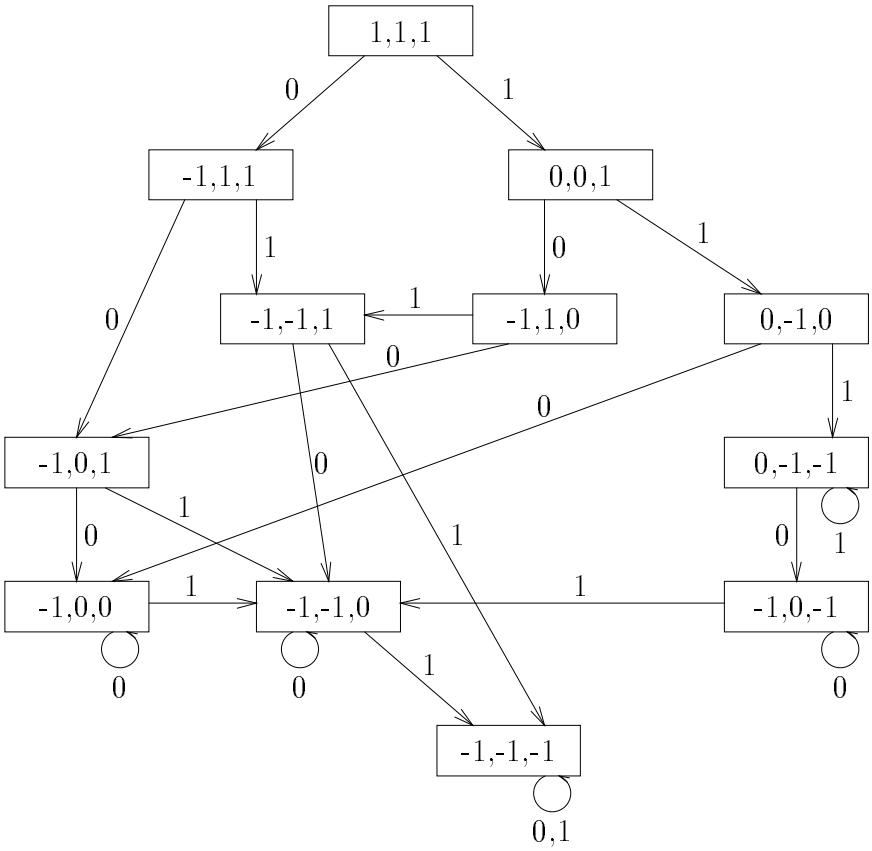

Fig. 2: The SED-automaton for $A = 001$ under cost function (2).

Lemma 4.4 *Let $\Delta(A)$ be the SED-automaton of a string $A = a_1 a_2 \ldots a_n$. Then for any string $B = b_1 b_2 \ldots b_m$ the type of the column $S(0, b_1 b_2 \ldots b_m)$ in the edit matrix $D(i, j)$, $i = 0, 1, \ldots, n$; $j = 0, 1, \ldots, m$, is identical with the state $\delta(I, b_1 b_2 \ldots b_m)$ in $\Delta(A)$, i.e., $T(S(0, b_1 b_2 \ldots b_m)) = \delta(I, b_1 b_2 \ldots b_m)$.*

In this Lemma, the function $\delta : Q \times V \to Q$ has been extended to $\delta : Q \times V^* \to Q$ in the standard fashion; see, for example, chapter 2 of [20].

Proof of Lemma 4.4: The proof is by induction on the length of B. For $B = \epsilon$ we have $\delta(I, \epsilon) = I = (1, 1, \ldots, 1)$. Clearly, $S(0, \epsilon) = C(0)$ and $T(C(0)) =$

$(1, 1, \ldots, 1)$.
Now assume that $T(S(0, b_1 b_2 \ldots b_i)) = \delta(I, b_1 b_2 \ldots b_i)$. For symbol b_{i+1} we first observe that $\delta(I, b_1 b_2 \ldots b_{i+1}) = \delta(\delta(I, b_1 b_2 \ldots b_i), b_{i+1})$. Similarly, $S(0, b_1 b_2 \ldots b_{i+1}) = S(S(0, b_1 b_2 \ldots b_i), b_{i+1})$. As $T(S(0, b_1 b_2 \ldots b_i)) = \delta(I, b_1 b_2 \ldots b_i)$ it is guaranteed by the construction rule for δ, given in Def. 4.1, and the uniqueness property of Lemma 4.2 that $T(S(0, b_1 b_2 \ldots b_i), b_{i+1}) = \delta(\delta(I, b_1 b_2 \ldots b_i), b_{i+1})$. □

Theorem 4.5 *Given a string $B = b_1 b_2 \ldots b_m$ and an SED-automaton $\Delta(A)$ for some string $A = a_1 a_2 \ldots a_n$, the string edit distance $d(A, B)$ is given by*

$$d(A, B) = m + \sum_{i=1}^{n} t_i$$

$$\textit{where } (t_1, t_2, \ldots, t_n) = \delta(I, B).$$ □

Example 4.6 Given $A = 001$ and the corresponding SED-automaton $\Delta(A)$ shown in Fig. 2, we get, for example $d(A, \epsilon) = 0 + 3 = 3$; $d(A, 0) = 1 + 1 = 2$; $d(A, 1) = 1 + 1 = 2$; $d(A, 00) = 2 + 0 = 2$; $d(A, 01) = 2 - 1 = 1$; $d(A, 10) = 2 + 0 = 2$; $d(A, 11) = 2 - 1 = 1$; $d(A, 000) = 3 - 1 = 2$; $d(A, 001) = 3 - 2 = 1$; $d(A, 0^n) = n - 1$ for $n \geq 3$; $d(A, 0010^n) = n + 1$ for $n \geq 1$; a.s.o. □

Proof of Theorem 4.5: Let $s = (s_1, s_2, \ldots, s_n)$ be the type of the last column, $C(m)$, in the edit matrix for the computation of $d(A, B)$. It follows immediately from Definition 3.4 that

$$D(n, m) = D(0, m) + \sum_{i=1}^{n} s_i.$$

From Lemma 4.4 it follows that $\delta(I, B) = T(C(m))$. Therefore $s_i = t_i$ for $i = 1, \ldots, n$. As $d(A, B) = D(n, m)$ and $D(0, m) = m$, we conclude that

$$d(A, B) = m + \textstyle\sum_{i=1}^{n} t_i.$$ □

If the method implied by Theorem 4.5 is to be used for matching strings A and B, both of which are given online, then its time complexity is exponential in the length of the shorter of the two strings as the number of states in the SED-automaton may be, in the worst case, exponential in the length of the corresponding string. However, if string A is a prototype that is given a priori, for example, a fixed entry in a dictionary, then the construction of the corresponding SED-automaton can be done offline. If we neglect the computational of this off-line step then the computation of the string edit distance $d(A, B)$ is linear in the length of B for any string B. That is, the time complexity is no longer dependent on A. The algorithm implied by Theorem 4.5 needs only a small number of constant operations for each symbol of B and can be expected very fast. Notice, however, that the SED-automaton for string A needs an amount of space which is exponential in the length of A.

5 Nearest-Neighbor Search Based on String Edit Distance

In this section we show how the SED-automata introduced in section 4 can be used for nearest-neighbor (NN) classification of strings. In NN classification we are given a finite set $A_1, A_2, \ldots, A_N$ of prototype strings, for example, words in a dictionary, and an unknown string B. The problem to be solved is the determination of the prototype A_i with smallest string edit distance to B. That is, we want to find A_i such that $d(A_i, B) = min\{d(A_j, B) | j = 1, \ldots, N\}$.

Definition 5.1 *a) Let $q = (t_1, t_2, \ldots, t_k)$ and $q' = (t'_1, t'_2, \ldots, t'_l)$ be two states of any two SED-automata. State q is* less than *or* equal *to state q', $q \leq q'$, if and only if $\sum_{i=1}^{k} t_i \leq \sum_{i=1}^{l} t'_i$.*

b) Let $Q = \{q_1, q_2, \ldots, q_r\}$ be a set of states. State $q \in Q$ is the minimum *of all states in Q, $q = min\{q_1, q_2, \ldots, q_r\}$, if and only if $q \leq q_i$, $i = 1, \ldots, r$.*

Example 5.2 The SED-automaton $\Delta(A)$ for string $A = 001$ is shown in Fig. 2. Another SED-automaton $\Delta(A')$ for string $A' = 10$ is shown in Fig. 3. Let $t = (-1, 0, -1)$ in Fig. 2 and $t' = (-1, 1)$ in Fig. 3. Then we have $t \leq t'$. The minimum of the union of all states in Fig. 2 and Fig. 3 is $(-1, -1, -1)$. □

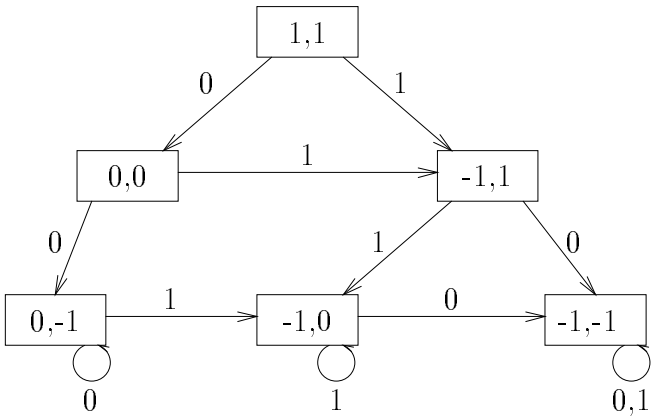

Fig. 3: The SED-automaton for $A = 10$ under cost function (2).

Definition 5.3 *Let A_i be a prototype string and $\Delta(A_i) = (Q_i, I_i, \delta_i)$ its corresponding SED-automaton, $i = 1, \ldots, N$. The NN-SED-automaton for $A_1, A_2, \ldots, A_N$ is a 4-tuple*
$\Delta(A_1, A_2, \ldots, A_N) = (Q, I, \delta, \mu)$ where

- $Q = \{[q_1, q_2, \ldots, q_N] | q_i \in Q_i, i = 1, \ldots, N\}$
- $I = [I_1, I_2, \ldots, I_N] \in Q$
- $\delta([q_1, q_2, \ldots, q_N], b) = [q'_1, q'_2, \ldots, q'_N]$ *if and only if* $\delta_i(q_i, b) = q'_i$, $i = 1, \ldots, N$.
- $\mu : Q \to \bigcup_{i=1}^{N} Q_i$ *is a function such that* $\mu([q_1, q_2, \ldots, q_N]) = min\{q_1, q_2, \ldots, q_N\}$.

Example 5.4 The NN-SED-automaton for the two strings $A = 001$ and $A' = 10$ constructed from Fig. 2 and Fig. 3 is shown in Fig. 4. The μ-function is implicitly given by printing the minimum state from either $\Delta(A)$ or $\Delta(A')$ in Fig. 5 in boldface. If the minimum is not unique, both states are printed in boldface. □

Theorem 5.5 *Let $A_1, A_2, \ldots, A_N$ be prototype strings, $\Delta(A_1, A_2, \ldots, A_N)$ their corresponding NN-SED-automaton, and $B = b_1 b_2 \ldots b_m$ any string. A_i is the nearest neighbor of B, i.e., $d(A_i, B) = min\{d(A_j, B) | j = 1, \ldots, N\}$ if and only if $\delta(I, B) = [q_1, q_2, \ldots, q_N]$ and $\mu([q_1, q_2, \ldots, q_N]) = q_i$.* □

A possible implementation of this Theorem results in the following procedure. Given $\Delta(A_1, A_2, \ldots, A_N)$ and $B = b_1 b_2 \ldots b_m$, we start at the initial state and traverse the states of the NN-SED-automaton according to the input B. After the input is exhausted, the state $\delta(I, B)$ is reached. This state is an N-tuple $[q_1, q_2, \ldots, q_N]$ where $q_1 \in Q_1$, $q_2 \in Q_2$, a.s.o. If the minimum of $\{q_1, q_2, \ldots, q_N\}$ is q_i then we conclude that A_i is the nearest neighbor of B.

Example 5.6 Consider the NN-SED-automaton for $A = 011$ and $A' = 10$ in Fig. 4 and string $B = 110$. Obviously, $\delta(I, B) = [(-1, 0, 0), (-1, -1)]$. The minimum of $(-1, 0, 0)$ and $(-1, -1)$ is $(-1, -1)$. As $(-1, -1)$ is a state of $\Delta(A')$, we conclude that $d(A', B) \leq d(A, B)$. In fact, $d(A', B) = 1$ and $d(A, B) = 2$. □

Proof of Theorem 5.5: From Def. 5.3 it follows that $\delta_i(I_i, B) = q_i$ if and only if $\delta(I, B) = [q_1, q_2, \ldots, q_N]$. Let $q_i = (t_{1i}, t_{2i}, \ldots, t_{n_i i})$. If $q_i = \mu([q_1, q_2, \ldots, q_N])$ then $q_i = min\{q_1, q_2, \ldots, q_N\}$. This means that $\sum_{l=1}^{n_i} t_{li} \leq \sum_{l=1}^{n_j} t_{lj}$. As $d(A_i, B) = m + \sum_{i=1}^{n_i} t_{li}$ we conclude that $d(A_i, B) \leq d(A_j, B)$ for $j = 1, \ldots, N$. Conversely, if $d(A_i, B) \leq d(A_j, B)$ for $j = 1, \ldots, N$, then $\sum_{l=1}^{n_i} t_{li}$ must be minimal among all $\sum_{j=1}^{n_j} t_{lj}$. This implies that $\mu([q_1, q_2, \ldots, q_N]) = q_i$. □

Given strings $A_1, A_2, \ldots, A_N$ and their corresponding SED-automata $\Delta(A_1)$, $\Delta(A_2)$, ..., $\Delta(A_N)$ the construction of $\Delta(A_1, A_2, \ldots, A_N)$ takes $O(\prod_{i=1}^{N} |Q_i|)$ time and space. If we consider, similar to section 4, the construction of the NN-SED-automaton an offline operation and neglect its computational complexity, then the nearest neighbor search for an unknown string B and a dictionary with words $A_1, A_2, \ldots, A_N$ takes only linear time in the length of B. That is, the time complexity is completely independent of the number and the lengths of the words in the dictionary. However, the number of the states of the NN-SED-automaton is exponential, i.e. the space complexity of the method is $O(3 \cdot exp(\sum_{i=1}^{N} |A_i|))$.

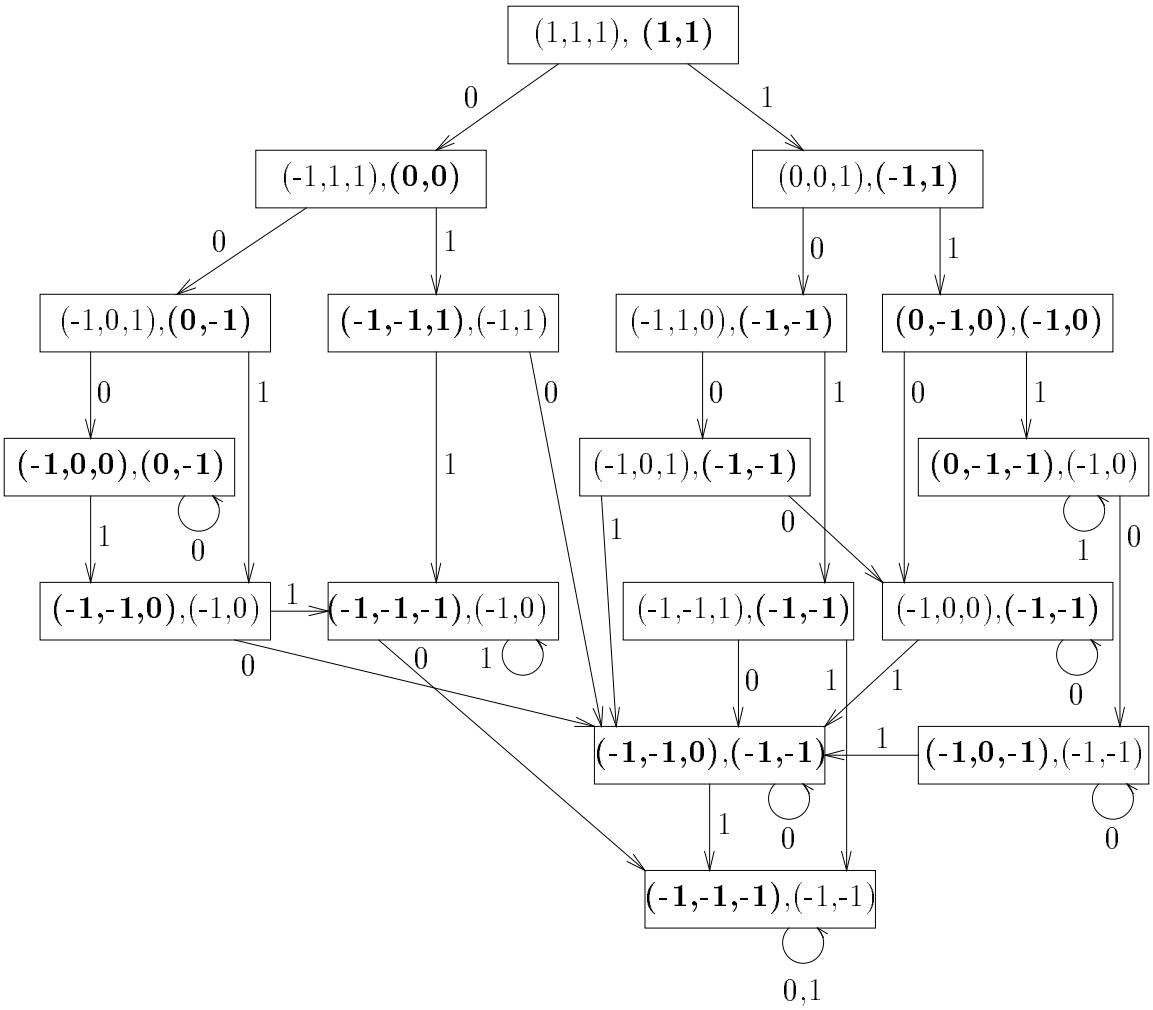

Fig. 4: The NN-SED-automaton $\Delta(A, A')$ for $A = 011$ and $A' = 10$; see also Fig. 2 and 3.

The nearest neighbor classification procedure resulting from Theorem 5.5 can be modified in a number of ways. If we are interested not only in *the* nearest neighbor but in the $k > 1$ nearest neighbors then we can simply generalize the μ-function such that $\mu([q_1, q_2, \ldots, q_N])$ gives the k smallest elements in $\{q_1, q_2, \ldots, q_N\}$. If ties are to be reported in either 1-NN or k-NN classification, then the μ-function can be modified accordingly. Notice that the classification method not only gives the nearest neighbor but also the actual string distance between the nearest dictionary word and the input.

6 Summary and Further Remarks

In this paper we have proposed a new algorithm for string edit distance computation. The new algorithm is very fast. It needs time that is only linear in the length of one of the two strings to be matched, provided that the other string has undergone some preprocessing in an off-line phase. This is a great improvement over other known algorithms for string edit distance computation [11,12,13]. The algorithm can be extended to matching a word against a dictionary of any size. In this case the time complexity is independent of the length of the dictionary words, and the number of entries in the dictionary. A potential limitation of the method, however, is its space complexity, which is exponential in the length of the dictionary words.

Throughout this paper we have considered only the cost function (2). However, it has been shown in [12] that under the condition of sparseness of the cost function there is always only a finite number of possible differences between two successive elements in a row or column of an edit matrix. Consequently, all results of the present paper can be easily extended to this case. The only major difference is the fact that the SED-automaton constructed for a prototype $A = a_1 a_2 \ldots a_n$ will have $O(C^n)$ states, where C is the number of possible differences between two consecutive matrix elements. Sparseness of the cost function means that there exists some constant r such that the cost of any edit operation is some integral multiple of r. Cost functions involving only integer or rational numbers are always sparse, but for real numbers sparseness is no longer guaranteed.

It is known that for the cost function

$$c(a \to b) = 2, \quad c(a \to \epsilon) = c(\epsilon \to a) = 1, \quad a \neq b$$

the edit distance of two strings is identical with the longest common subsequence [11]. A case analysis similar to the one provided in Lemma 3.1 reveals that under this cost function the difference between two successive elements in the edit matrix is either 1 or -1. This reduces the space complexity of any corresponding SED-automaton from $O(3^n)$ to $O(2^n)$.

Acknowledgment

The author wants to thank Mr.P.Graber for his help in preparing the manuscript and providing an implementation of the method described in this paper.

References

[1] Pavlidis, T. and Mori, S.: Optical Character Recognition, Special Issue of Proceedings of the IEEE, Vol. 80, No. 7, July 1992, 1027-1209

[2] O'Gorman, L. and Kasturi, R.: Document Image Analysis Systems, Special Issue of IEEE Computer, Vol. 25, No. 7, July 1992

[3] Kasturi, R. and O'Gorman, L.: Document Image Analysis Techniques, Special Issue of Machine Vision and Application, Vol. 5, No. 3, Springer Verlag, Summer 1992

[4] Baird, H., Bunke, H. and Yamamoto, K.(eds.): Structured Document Image Analysis, Springer Verlag, 1992

[5] Elliman, D.G. and Lancaster, I.T.: A Review of Segmentation and Contextual Analysis Techniques for Text Recognition, Pattern Recognition, Vol. 23, No. 3/4, 1990, 337-346

[6] Srihari, S.N.(ed.): Computer Text Recognition and Error Correction, Tutorial, IEEE Computer Society Press, Silver Spring, MD, 1985

[7] Riseman, E.M. and Hanson, A.R.: A Contextual Postprocessing System for Error Correction Using Binary n-Grams, IEEE Trans. on Computers, Vol. C-23, May 1974, 480-493

[8] Hull, J.J. and Srihari, S.N.: Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms, IEEE Trans. PAMI, Vol. PAMI-4, Sept 1982, 520-530

[9] Downtown, A.C. and Tregido, R.W.S.: The Use of a Trie Structured Dictionary as a Contextual Aid to Recognition of Handwritten British Postal Addresses, Proc. 1st ICDAR, Saint-Malo, France, 1991, 542-550

[10] Leroux, M., Salome, J.C. and Badard, J.: Recognition of Cursive Words in a Small Lexicon, Proc. 1st ICDAR, Saint-Malo, France, 1991, 774-782

[11] Wagner, R.A. and Fischer, M.J.: The String-to-String Correction Problem, Journal of the ACM, Vol. 21, No. 1, 1974, 168-173

[12] Masek, W.J. and Paterson, M.S.: A Faster Algorithm for Comparing String-Edit Distances, Journal of Computer and System Sciences, Vol. 20, No. 1, 1980, 18-31

[13] Ukkonen, E.: Algorithms for Approximate String Matching, Inform. and Control, Vol. 64, 1985, 100-118

[14] Hall, P.A.V. and Dowling, G.R.: Approximate String Matching, ACM Comp. Surveys, Vol. 12, No. 4, 1980, 381-401

[15] Bunke, H.: Recent Advances in String Matching, in Bunke, H.(ed.): Advances in Structural and Syntactic Pattern Recognition, World Scientific Publ. Co., Singapore, 1993, to appear

[16] Kashyap, R.L. and Oommen, B.J.: Spelling Correction Using Probabilistic Methods, Pattern Recognition Letters 2, 1984, 147-154

[17] Bozinovic, R. and Srihari, S.N.: A String Correction Algorithm for Cursive Script Recognition, IEEE Trans. PAMI, Vol. PAMI-4, 1982, 655-663

[18] Takahashi, H. Itoh, N., Amano, T. and Yamashita, A.: A Spelling Correction Method and its Application to an OCR System, Pattern Recognition, Vol. 23, No. 3/4, 1990, 363-377

[19] Ha Minh, T. and Bunke. H.: Very Fast Recognition of Giro Check Form, Proc. SPIE Conf. 1906 on Character Recognition Technologies, San Jose, CA, 1993, to appear

[20] Hopcroft, J.E. and Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, Addison Wesley Publ. Co., 1979