Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Discrete Algorithms 2 (2004) 303–312

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

A dynamic edit distance table

Sung-Ryul Kim [a,*,1], Kunsoo Park [b,2]

[a] *Division of Internet & Media and Multidisciplinary Aerospace System Design Team, Konkuk University, Seoul 143-701, South Korea*

[b] *School of Computer Science and Engineering, Seoul National University, Seoul 151-742, South Korea*

Abstract

In this paper we consider the incremental/decremental version of the edit distance problem: given a solution to the edit distance between two strings A and B, find a solution to the edit distance between A and B' where $B' = aB$ (incremental) or $bB' = B$ (decremental). As a solution for the edit distance between A and B, we define the difference representation of the D-table, which leads to a simple and intuitive algorithm for the incremental/decremental edit distance problem.

Keywords: String matching; Edit distance; Incremental/decremental edit distance

1. Introduction

Given two strings $A[1..m]$ and $B[1..n]$ over an alphabet Σ, the *edit distance* between A and B is the minimum number of *edit operations* needed to convert A to B. The edit distance problem is to find the edit distance between A and B. Most common edit operations are the following:

1. *change*: replace one character of A by another single character of B;
2. *deletion*: delete one character from A;
3. *insertion*: insert one character into B.

* Corresponding author.
E-mail addresses: kimsr@konkuk.ac.kr (S.-R. Kim), kpark@theory.snu.ac.kr (K. Park).

[1] Supported by Korea Reasearch Foundation Grant KRF-2002-003-D00304.

[2] Work supported by Brain Korea 21 Project, the IMT2000 Project AB02, and the MOST grant M6-0203-00-0039.

1570-8667/$ – see front matter
doi:10.1016/S1570-8667(03)00082-0

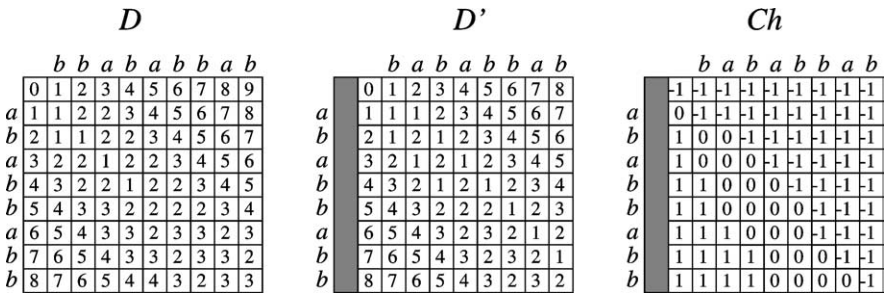

Fig. 1. An example *Ch*-table.

A well-known method for solving the edit distance problem in $O(mn)$ time uses the *D-table* [1,11]. Let $D(i, j)$, $0 \leqslant i \leqslant m$ and $0 \leqslant j \leqslant n$, be the edit distance between $A[1..i]$ and $B[1..j]$. Initially, $D(i, 0) = i$ for $0 \leqslant i \leqslant m$ and $D(0, j) = j$ for $0 \leqslant j \leqslant n$. An entry $D(i, j)$, $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$, of the D-table is determined by the three entries $D(i-1, j-1)$, $D(i-1, j)$, and $D(i, j-1)$. The recurrence for the D-table is as follows: for all $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$,

$$D(i, j) = \min\bigl\{D(i-1, j-1) + \delta_{ij}, D(i-1, j) + 1, D(i, j-1) + 1\bigr\} \tag{1}$$

where $\delta_{ij} = 0$ if $A[i] = B[j]$; $\delta_{ij} = 1$, otherwise.

In this paper we consider the following incremental (respectively decremental) version of the edit distance problem: given a solution for the edit distance between A and B, compute a solution for the edit distance between A and aB (respectively B' where $B = bB'$), where a (respectively b) is a symbol in Σ. By a *solution* we mean some encoding of the D-table computed between A and B. Since essentially the same techniques can be used to solve both incremental and decremental versions of the edit distance problem, we will consider only the decremental version.

Consider the symmetric problems where B is changed at the end: given a solution for the edit distance between A and B, compute a solution for the edit distance between A and Ba (respectively B' where $B = B'b$). Given the D-table between A and B, these problems are easily solved in $O(m)$ time by computing just one more column (respectively removing one column) at the end of the D-table. However, it is impossible to solve the problems considered in this paper in less than $\Theta(mn)$ time *when we are given the D-table as the solution* because the D-table between A and B and the D-table between A and aB (respectively B' where $B = B'b$) can be different in $\Theta(mn)$ places. (See the example in Fig. 1.) For an algorithm for incremental/decremental version to be useful, it needs to compute the new solution in time much less than $\Theta(mn)$, since $\Theta(mn)$ is the time required to compute the new solution from scratch.

As a solution for the edit distance between A and B, we define the difference representation of the D-table (*DR*-table for short). Each entry $DR(i, j)$ in the *DR*-table between A and B has two fields defined as follows: for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$,

1. $DR(i, j).U = D(i, j) - D(i-1, j)$,
2. $DR(i, j).L = D(i, j) - D(i, j-1)$.

A third field $DR(i, j).UL$, which is defined to be $D(i, j) - D(i-1, j-1)$, will be used later, but it need not be stored in $DR(i, j)$ because it can be computed as $DR(i, j).U + DR(i-1, j).L$. Because the possible values that each of $DR(i, j).U$ and $DR(i, j).L$ can have are -1, 0, and 1 [9], we need only four bits to store an entry in the DR-table. It is easy to see that the D-table can be converted to the DR-table in $O(mn)$ time, and vice versa. We can also compute one row (respectively column) of the D-table from the DR-table in $O(n)$ (respectively $O(m)$) time. For example, if we want to compute row i, we can set the value of $D(i, 0)$ from the definition and then sequentially compute each entry $D(i, j)$, $1 \leqslant j \leqslant n$, as $D(i, j-1) + DR(i, j).L$.

The incremental/decremental version of the edit distance problem was first considered by Landau et al. [3]. They used the C-table [2,4,5,8,10] (represented with linked lists) as a solution for the edit distance between A and B. Given a threshold k on the edit distance, their algorithm runs in $O(k)$ time. (If the threshold k is not given, it runs in $O(m+n)$ time.) Also, Schmidt [6] gave an algorithm based on weighted grid graphs that works in $O(m+n)$ time for the same problem. However, the results in [3,6] are quite complicated.

In this paper we present an $O(m+n)$-time algorithm for the incremental/decremental edit distance problem. Our result is much simpler and more intuitive than those of Landau et al. [3] and Schmidt [6]. A key tool in our algorithm is the *change table* between the two D-tables before and after an increment/decrement. The change table is not actually constructed in our algorithm, but it is central in understanding our algorithm.

The incremental/decremental edit distance problem finds a variety of applications, which include the longest prefix match problem, the approximate overlap problem, the cyclic string comparison problem, and the text screen update problem [3,6]. Another application can be found in computing an approximate period of a string [7]. To verify whether a string p is an approximate period of another string x, one needs to find the edit distance between p and every substring of x [7]. A naive method that computes a D-table of size $O(|p|^2)$ for each position of x will take $O(|p|^2|x|)$ time, but our algorithm as well as those in [3,6] reduces the time complexity to $O(|p| \cdot |x|)$.

This paper is organized as follows. In Section 2, we describe the important properties of the change table. In Section 3, we present our algorithm for the incremental/decremental edit distance problem.

2. Preliminary properties

Let Σ be a finite *alphabet* of *symbols*. A *string* over Σ is a finite sequence of symbols in Σ. The length of a string A is denoted by $|A|$. The ith symbol in A is denoted by $A[i]$ and the substring consisting of the ith through the jth symbols of A is denoted by $A[i..j]$.

Let A and B be strings of lengths m and n, respectively, over Σ, and let $B' = B[2..n]$. Let D be the D-table between A and B and let D' be the D-table between A and B'. Also let DR be the DR-table between A and B and let DR' be the DR-table between A and B'. In this section, we prove the key properties between D and D' that enables us to compute efficiently DR' from DR.

One key tool in understanding our algorithm is the *change table* (*Ch*-table for short) from D to D'. Later, when we compute DR' from DR, the first column of DR is discarded

and each entry $DR(i, j+1)$, $0 \leqslant i \leqslant m$ and $0 \leqslant j < n$, will be converted to $DR'(i, j)$. Thus, each entry in the *Ch*-table *Ch* from D to D' is defined as follows:

$$Ch(i, j) = D'(i, j) - D(i, j+1).$$

The *Ch*-table is not actually constructed in our algorithm because the initialization of the *Ch*-table will require $\Theta(mn)$ time. It will be used only for the description of the algorithm. See Fig. 1 for an example *Ch*-table.

Fig. 1 suggests a property of the *Ch*-table: the entries of value -1 (respectively 1) appear contiguously in the upper-right (respectively lower-left) part of the *Ch*-table in a *staircase-shaped* region. This property is formally proved in the following series of lemmas.

Lemma 1. *In the Ch-table Ch, the following properties hold.*

1. $Ch(0, j) = -1$ *for all* $0 \leqslant j < n$.
2. $Ch(i, 0) = 0$ *for all* $1 \leqslant i < k$, *where* k *is the smallest index in* A *such that* $A[k] = B[1]$.
3. $Ch(i, 0) = 1$ *for all* $k \leqslant i \leqslant m$.

Proof. Immediate from the definition of the D-table. □

Lemma 2. *For* $1 \leqslant i \leqslant m$ *and* $1 \leqslant j < n$, *the possible values of* $Ch(i, j)$ *are in the range* $\min\{Ch(i-1, j-1), Ch(i-1, j), Ch(i, j-1)\}..\max\{Ch(i-1, j-1), Ch(i-1, j), Ch(i, j-1)\}$.

Proof. Recall that $Ch(i, j)$ is defined to be $D'(i, j) - D(i, j+1)$. By recurrence (1), $D(i, j+1)$ is

$$\min\big\{D(i-1, j) + \delta_{i,j+1}, D(i-1, j+1) + 1, D(i, j) + 1\big\}. \tag{2}$$

Also, $D'(i, j)$ is $\min\{D'(i-1, j-1) + \delta'_{ij}, D'(i-1, j) + 1, D'(i, j-1) + 1\}$ where $\delta'_{ij} = 0$ if $A[i] = B'[j]$; $\delta'_{ij} = 1$, otherwise. Because $B'[j]$ is the same symbol as $B[j+1]$, $\delta'_{ij} = \delta_{i,j+1}$. Hence, $D'(i, j)$ can be rewritten as

$$\min\left\{\begin{array}{c} D(i-1, j) + Ch(i-1, j-1) + \delta_{i,j+1}, \\ D(i-1, j+1) + Ch(i-1, j) + 1, \\ D(i, j) + Ch(i, j-1) + 1 \end{array}\right\}. \tag{3}$$

Note that the only differences between (2) and (3) are additional terms $Ch(i-1, j-1)$, $Ch(i-1, j)$, and $Ch(i, j-1)$ in (3). Assume without loss of generality that the second argument is minimum in (2). If the second argument is minimum in (3), the lemma holds because $Ch(i, j) = Ch(i-1, j)$. Otherwise, assume without loss of generality that the third argument is minimum in (3). Then $Ch(i, j) = D(i, j) + Ch(i, j-1) + 1 - (D(i-1, j+1) + 1) \geqslant Ch(i, j-1)$ because the second argument is minimum in (2). Also, $Ch(i, j) \leqslant Ch(i-1, j)$ because the third argument is minimum in (3). □

Corollary 1. *The possible values of* $Ch(i, j)$ *are* $-1, 0,$ *and* 1.

Proof. It follows from Lemmas 1 and 2. □

Lemma 3. *For each* $0 \leqslant i \leqslant m$, *let* $f(i)$ *be the smallest integer* j *such that* $Ch(i, j) = -1$. ($f(i) = n$ *if* $Ch(i, j') \neq -1$ *for* $0 \leqslant j' < n$.) *Then,* $Ch(i, j') = -1$ *for all* $f(i) \leqslant j' < n$. *Furthermore,* $f(i) \geqslant f(i-1)$ *for* $1 \leqslant i \leqslant m$.

Proof. We use induction on i. When $i = 0$, $f(i) = 0$ and the lemma holds by Lemma 1. Assume inductively that the lemma holds for $i = k$. That is, $Ch(k, j') \neq -1$ for $0 \leqslant j' < f(k)$ and $Ch(k, j') = -1$ for $f(k) \leqslant j < n$.

Let $Ch(k+1, l)$ be the first entry in row $k+1$ that is -1. For $Ch(k+1, l)$ to be -1, at least one of $Ch(k, l-1)$ and $Ch(k, l)$ must be -1 by Lemma 2. Thus, we have shown that $l = f(k+1) \geqslant f(k)$. It is easy to see that $Ch(k+1, l') = -1$ for $f(k+1) < l' < n$ by the inductive assumption, the condition that $f(k+1) \geqslant f(k)$, and Lemma 2. □

The following lemma is symmetric to Lemma 3 and it can be similarly proved.

Lemma 4. *For each* $0 \leqslant j < n$, *let* $g(j)$ *be the smallest integer* i *such that* $Ch(i, j) = 1$. ($g(j) = m+1$ *if* $Ch(i', j) \neq 1$ *for* $0 \leqslant i' \leqslant m$.) *Then,* $Ch(i', j) = 1$ *for all* $g(j) \leqslant i' \leqslant m$. *Furthermore,* $g(j) \geqslant g(j-1)$ *for* $1 \leqslant j < n$.

We say that an entry $Ch(i, j)$ is *affected* if the values of $Ch(i-1, j-1)$, $Ch(i-1, j)$, and $Ch(i, j-1)$ are not the same. We also say that $DR'(i, j)$ is affected if $Ch(i, j)$ is affected.

Lemma 5. *If* $DR'(i, j)$ *is not affected, then* $DR'(i, j)$ *equals* $DR(i, j+1)$.

Proof. If $DR'(i, j)$ is not affected, then the value of $Ch(i, j)$ is the same as the common value of $Ch(i-1, j-1)$, $Ch(i-1, j)$, and $Ch(i, j-1)$ by Lemma 2. Then $DR'(i, j).U = D'(i, j) - D'(i-1, j) = D(i, j+1) + Ch(i, j) - (D(i-1, j+1) + Ch(i-1, j)) = DR(i, j+1).U$. Similarly, $DR'(i, j).L = DR(i, j+1).L$. □

We say that an entry $Ch(i, j)$ is a (-1)-*boundary* (respectively 1-*boundary*) *entry* if $Ch(i, j)$ is of value -1 (respectively 1) and at least one of $Ch(i, j-1)$, $Ch(i+1, j)$, and $Ch(i+1, j-1)$ (respectively $Ch(i, j+1)$, $Ch(i-1, j)$, and $Ch(i-1, j+1)$) is not of value -1 (respectively 1).

By Lemma 5 we can conclude that in computing DR' from DR, only the affected entries need be changed. See Fig. 1 again. Because the entries whose values are -1 (or 1) appear contiguously in the *Ch*-table, the affected entries are either (-1)- or 1-boundary entries themselves or appear adjacent to (-1)- or 1-boundary entries. The key idea of our algorithm is to scan the (-1)- and 1-boundary entries starting from the upper-left corner of the *DR*-table when we compute the affected entries. Lemmas 3 and 4 imply that the number of (-1)- and 1-boundary entries in the *DR*-table is $O(m+n)$.

3. Boundary scan algorithm

In this section we show how to compute DR' from DR. First, we describe how we scan the boundary entries starting from the upper-left corner of the DR'-table within the proposed time complexity. Then, we will mention the modifications to the boundary-scan algorithm which leads to an algorithm that converts DR to DR'.

For simplicity we will use the Ch-table in the description of our algorithm. However, the Ch-table is not explicitly constructed but accessed through the one-dimensional tables $f()$ and $g()$. The details will be given later.

Lemma 6. $Ch(i, j) = \min\{-DR(i, j+1).UL + Ch(i-1, j-1) + \delta_{i,j+1}, -DR(i, j+1).U + Ch(i-1, j) + 1, -DR(i, j+1).L + Ch(i, j-1) + 1\}$ (*i.e.,* $Ch(i-1, j-1)$, $Ch(i-1, j)$, $Ch(i, j-1)$, *and* $DR(i, j+1)$ *are needed to compute* $Ch(i, j)$).

Proof. Recall that $Ch(i, j) = D'(i, j) - D(i, j+1)$. Substituting recurrence (1.1) for $D'(i, j)$ and distributing $D(i, j+1)$ into the min function, we have $Ch(i, j) = \min\{\ldots, D'(i-1, j) - D(i, j+1) + 1, \ldots\}$ (only the second argument is shown). Substituting $D(i-1, j+1) + Ch(i-1, j)$ for $D'(i-1, j)$, the second argument becomes $D(i-1, j+1) - D(i, j+1) + Ch(i-1, j) + 1 = -DR(i, j+1).U + Ch(i-1, j) + 1$. The lemma follows from similar calculations for the first and the third arguments. □

Algorithm 1 is the boundary-scan algorithm. In the algorithm, the pair (i_{-1}, j_{-1}) (respectively (i_1, j_1)) indicates that $Ch(i_{-1}, j_{-1})$ (respectively $Ch(i_1, j_1)$) is the current (-1)-boundary (respectively 1-boundary) entry that is being scanned. By Lemma 1, their initial values correspond to boundary entries in the Ch-table. The following property holds for $Ch(i_{-1}, j_{-1})$ and $Ch(i_1, j_1)$ by Lemmas 3 and 4. See Fig. 2 for an illustration.

Property 1.

(1) $Ch(i, j) \neq -1$ *if* $i > i_{-1}$ *and* $j < j_{-1}$.
(2) $Ch(i, j) \neq 1$ *if* $i < i_1$ *and* $j > j_1$.

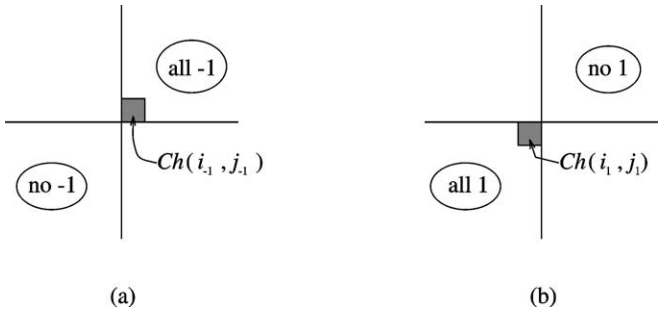

Fig. 2. Boundary entry conditions.

```
Algorithm 1
  Let k be the smallest index in A such that A[k] = B[1].
  (i_{-1}, j_{-1}) ← (0, 1); (i_1, j_1) ← (k, 0); f(0) ← 0; g(0) ← k
  finished_{-1} ← false
  finished_1 ← false
  while not finished_{-1} or not finished_1 do
    if i_{-1} < i_1 − 1 then {case 1}
      Compute Ch(i_{-1} + 1, j_{-1}). {See Fig. 4.}
      if Ch(i_{-1} + 1, j_{-1}) = −1 then
        i_{-1} ← i_{-1} + 1; f(i_{-1}) ← j_{-1}
      else
        j_{-1} ← j_{-1} + 1
      fi
    else if j_1 < j_{-1} − 1 then {case 2}
      Symmetric to case 1.
    else{case 3, i_1 = i_{-1} + 1 and j_1 = j_{-1} − 1}
      Compute Ch(i_{-1} + 1, j_{-1}). {See Fig. 5.}
      if Ch(i_{-1} + 1, j_{-1}) = −1 then
        i_{-1} ← i_{-1} + 1; i_1 ← i_1 + 1; f(i_{-1}) ← j_{-1}
      else if Ch(i_{-1} + 1, j_{-1}) = 1 then
        j_{-1} ← j_{-1} + 1; j_1 ← j_1 + 1; g(j_1) ← i_1
      else
        j_{-1} ← j_{-1} + 1; i_1 ← i_1 + 1
      fi
    fi
    if i_{-1} = m or j_{-1} = n then finished_{-1} ← true fi
    if i_1 = m + 1 or j_1 = n − 1 then finished_1 ← true fi
  od
```

Fig. 3. Algorithm 1.

In one iteration of the loop in Algorithm 1, one or both of the current boundary entries are moved to the next boundary entries. For example, the current (-1)-boundary entry is moved to the next (-1)-boundary entry which can be down or to the right of the current (-1)-boundary entry. We maintain the following invariants in each iteration of Algorithm 1.

Invariant 1.

(1) $i_{-1} < i_1$ *and* $j_{-1} > j_1$.
(2) *All values of* $f(0), \ldots, f(i_{-1})$ *are known.*
(3) *All values of* $g(0), \ldots, g(j_1)$ *are known.*

One iteration of Algorithm 1 has three cases. Case 1 applies when the current (-1)-boundary can be moved by one entry (down or to the right) without violating Invariant 1(1). Case 2 applies when the current 1-boundary can be moved by one entry (down or to the right) without violating Invariant 1(1). Case 3 applies when moving the (-1)-boundary entry down by one entry or moving the 1-boundary entry to the right by one entry will violate Invariant 1(1), and thus both boundary entries have to be moved simultaneously. What Algorithm 1 does in each case is described in Fig. 3.

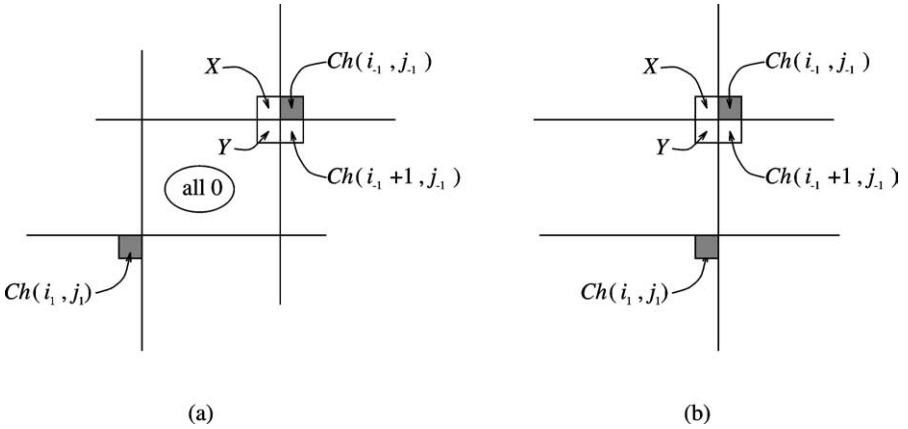

Fig. 4. Case 1.

What remains to show is the methods to obtain the values of the *Ch*-table entries that are used to compute a new *Ch*-table entry, e.g., $Ch(i_{-1}+1, j_{-1})$ in case 1. The two subcases for case 1 are depicted in Fig. 4. The first subcase is when $j_{-1} > j_1 + 1$. See Fig. 4(a). The unknown values of the *Ch*-table entries are X and Y. By Invariant 1(2) the value of $f(i_{-1})$ is known. If $f(i_{-1}) < j_{-1}$, then $X = -1$. Otherwise ($f(i_{-1}) = j_{-1}$), $X = 0$ because X is not 1 by Property 1(1). It is easy to see that $Y = 0$ because Y is inside the region in which there are no (-1)'s (by Property 1(1)) and no 1's (by Property 1(2)). The second subcase is when $j_{-1} = j_1 + 1$. See Fig. 4(b). We can compute the value of X as -1 if $f(i_{-1}) < j_{-1}$; 1 if $g(j_1) \leqslant i_{-1}$; 0, otherwise. We know that $Y \neq -1$ by Property 1(1). Thus, $Y = 1$ if $g(j_1) \leqslant i_{-1} + 1$; $Y = 0$, otherwise. Case 3 is depicted in Fig. 5. The value of X can be computed as we computed the value of X in the second subcase of case 1.

We now show that all affected *Ch*-table entries are computed by Algorithm 1. It is easy to see that each affected entry $Ch(i, j)$, $1 \leqslant i \leqslant m$ and $1 \leqslant j < n$, falls into one of the following types by Lemmas 3 and 4. For each of the types we can easily check which cases in our algorithm compute $Ch(i, j)$.

1. $Ch(i, j)$ is a (-1)-boundary entry such that $Ch(i, j-1) \neq -1$: $Ch(i, j)$ is computed by case 1 if $Ch(i, j-1) = 0$; by case 3, otherwise.
2. $Ch(i, j)$ is an 1-boundary entry such that $Ch(i-1, j) \neq 1$: $Ch(i, j)$ is computed by case 2 if $Ch(i-1, j) = 0$; by case 3, otherwise.
3. $Ch(i, j) = 0$ and either $Ch(i-1, j) = -1$ or $Ch(i, j-1) = 1$: $Ch(i, j)$ is computed by case 1 if $Ch(i, j-1) = 0$; by case 2 if $Ch(i-1, j) = 0$; by case 3, otherwise.

To compute DR' from DR, we first discard the first column from DR. Then, we run a modified version of Algorithm 1. The modifications to Algorithm 1 is to compute $DR'(i, j)$ whenever we compute the value of $Ch(i, j)$. Once $Ch(i, j)$ is computed using Lemma 6, the fields in $DR'(i, j)$ can be easily computed. That is, $DR'(i, j).L = DR(i, j+1).L + Ch(i, j) - Ch(i, j-1)$ and $DR'(i, j).U = DR(i, j+1).U + Ch(i, j) - Ch(i-1, j)$.

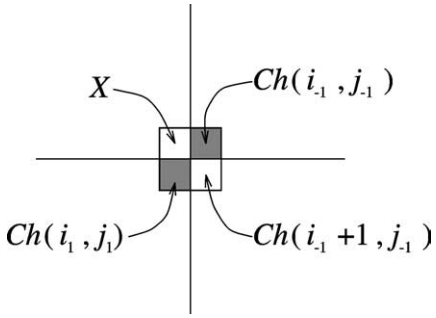

Fig. 5. Case 3.

We can easily check that one iteration of the loop takes only constant time and that it increases at least one of i_{-1}, j_{-1}, i_1, j_1 by one. Hence, the time complexity of our algorithm is $O(m+n)$.

Theorem 1. *Let A and B be two strings of lengths m and n, respectively, and $B' = B[2..n]$. Given the difference representation DR between A and B, the difference representation DR' between A and B' can be computed in $O(m+n)$ time.*

4. Conclusion

We have presented an $O(m+n)$-time algorithm for the incremental/decremental version of the edit distance problem. With slight modifications, our algorithm also applies to the Levenstein distance where only insertions and deletions are considered as possible edit operations. First, we have to modify the definition of δ_{ij} such that $\delta_{ij} = 0$ if $A[i] = B[j]$ and $\delta_{ij} = 2$, otherwise. It turns out that the possible values for *Ch*-table entries are 1 and -1 with the Levenstein distance. (Corollary 1 needs to be modified accordingly.) Also, in the first columns of *Ch*-tables for the edit distance and the Levenstein distance, the only differences are at the places where the *Ch*-table for the edit distance have zeros; they are all -1's in the *Ch*-table for the Levenstein distance. (Lemma 1 needs to be modified.) However, all other lemmas apply without changes. Thus, all we have to do is to change the initial value of (i_{-1}, j_{-1}) in Algorithm 1 to be $(k-1, 1)$.

References

[1] Z. Galil, R. Giancarlo, Data structures and algorithms for approximate string matching, J. Complexity 4 (1988) 33–72.
[2] Z. Galil, K. Park, An improved algorithm for approximate string matching, SIAM J. Comput. 19 (6) (1990) 989–999.
[3] G.M. Landau, E.W. Myers, J.P. Schmidt, Incremental string comparison, SIAM J. Comput. 27 (2) (1998) 557–582.
[4] G.M. Landau, U. Vishkin, Fast string matching with k differences, J. Comput. System Sci. 37 (1988) 63–78.

[5] G.M. Landau, U. Vishkin, Fast parallel and serial approximate string matching, J. Algorithms 10 (1989) 157–169.
[6] J.P. Schmidt, All highest scoring paths in weighted graphs and their applications to finding all approximate repeats in strings, SIAM J. Comput. 27 (4) (1998) 972–992.
[7] J.S. Sim, C.S. Iliopoulos, K. Park, W.F. Smyth, Approximate periods of strings, Theoret. Comput. Sci. 262 (1–2) (2001) 557–568.
[8] E. Ukkonen, Algorithms for approximate string matching, Inform. and Control 64 (1985) 100–118.
[9] E. Ukkonen, Finding approximate patterns in strings, J. Algorithms 6 (1985) 132–137.
[10] E. Ukkonen, D. Wood, Approximate string matching with suffix automata, Algorithmica 10 (1993) 353–364.
[11] R.A. Wagner, M.J. Fisher, The string-to-string correction problem, J. Assoc. Comput. Mach. 21 (1974) 168–173.