Edit Distance of Run-Length Encoded Strings

Ora Arbell*
Haifa University

Gad M. Landau†
Haifa University
and
Polytechnic University

Joseph S. B. Mitchell‡
University at Stony Brook

Abstract

Let X and Y be two run-length encoded strings, of encoded lengths k and l, respectively. We present a simple $O(|X|l + |Y|k)$ time algorithm that computes their edit distance.

1 Introduction

Two of the known methods to measure the similarity between two strings are "edit distance" and the "longest common subsequence." *Edit distance* [4] measures the minimum number of operations that are required to transform one string into the other one, when the permitted operations are substitution, deletion, and insertion. The goal is to find such a sequence of operations of minimum length. The *longest common subsequence* (LCS) measures the length of the longest identical subsequence of the two strings.

A string S is *run-length encoded* if it is described as an ordered sequence of pairs (σ, i), often denoted "σ^i," each consisting of an alphabet symbol, σ, and an integer, i. Each pair corresponds to a *run* in S, consisting of i consecutive occurrences of σ. For example, the string $aaaabbbbcccabbbbcc$ can be encoded as $a^4b^4c^3a^1b^4c^2$. Such a run-length encoded string can be significantly shorter than the expanded string representation. Indeed, run-length encoding serves as a popular image compression technique, since many classes of images (e.g., binary images in facsimile transmission or for use in optical character recognition) typically contain large patches of identically-valued pixels.

Let X and Y be two run-length encoded strings, of encoded lengths k and l, respectively. The LCS problem for strings of this kind has been considered previously. In particular, Bunke, and Csirik [2] presented an $O(|X|l + |Y|k)$ time algorithm, while Apostolico, Landau, and Skiena [1]

*Department of Computer Science, Haifa University, Haifa 31905, Israel; email: ora@cs.haifa.ac.il; partially supported by by the Israel Science Foundation grants 173/98 and 282/01.

†Department of Computer Science, Haifa University, Haifa 31905, Israel, phone: (972-4) 824-0103, FAX: (972-4) 824-9331; Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840; email: landau@poly.edu; partially supported by NSF grants CCR-9610238 and CCR-0104307, by NATO Science Programme grant PST.CLG.977017, by the Israel Science Foundation grants 173/98 and 282/01, by the FIRST Foundation of the Israel Academy of Science and Humanities, and by IBM Faculty Partnership Award.

‡Department of Applied Mathematics and Statistics, State University of New York, Stony Brook, NY 11794-3600, phone: (631) 632-8366, FAX: (631) 632-8490; email: jsbm@ams.sunysb.edu; partially supported by grants from the Binational Science Foundation, HRL Laboratories, the National Science Foundation (CCR-9732221, CCR-0098172), NASA Ames Research Center, Northrop-Grumman Corporation, Sandia National Labs, Seagull Technology, and Sun Microsystems.

described an $O(kl\log(kl))$ time algorithm. Mitchell [6] has obtained an $O((d+k+l)\log(d+k+l))$ time algorithm for a more general string matching problem in run-length encoded strings, where d is the number of matches of compressed characters. His algorithm is based on computing geometric shortest paths using special convex distance functions.

In this paper we present a simple $O(|X|l+|Y|k)$ time algorithm for computing the edit distance between two run-length encoded strings. This solves an open problem that was posed by Bunke and Csirik [2].

Since the original submission of this paper, two papers with related results have recently appeared. The first, by Mäkinen, Navarro, and Ukkonen [5] presents an algorithm similar to our own, which was independently and simultaneously discovered. The second, by Crochemore, Landau and Ziv-Ukelson [3], which achieves the same time complexity as the one reported in this paper, while extending the supported unit edit cost metrics to all similarity metrics that use an additive gap penalty. The algorithm is based on the total monotonicity properties of distance matrices.

2 Preliminaries

Let $X = x_1, x_2 \ldots x_n$ and $Y = y_1, y_2 \ldots y_m$ be two run-length encoded strings, with encoded lengths k and l, respectively. It is well known ([7]) that dynamic programming can be used to compute the edit distance between X and Y in time $\Theta(nm)$, as follows. The algorithm computes a matrix $A[0...n, 0...m]$, so that $A[i,j]$ $(0 \leq i \leq n, 0 \leq j \leq m)$ is the edit distance between $x_1x_2...x_i$ and $y_1y_2...y_j$. The computation begins with the values

$$\forall i,\ A[i,0] = i,$$

$$\forall j,\ A[0,j] = j,$$

and continues by computing the remaining matrix elements, as follows:

$$A[i,j] := \min\left\{A[i,j-1]+1, A[i-1,j]+1, A[i-1,j-1]+t_{i,j}\right\}, \tag{1}$$

where

$$t_{i,j} = \begin{cases} 0 & x_i = y_j, \\ 1 & x_i \neq y_j. \end{cases}$$

Theorem 1 (Ukkonen [8, 9]) *1.* $A[i,j] - A[i-1,j-1] \in \{0,1\} \quad 1 \leq i \leq n;\ 1 \leq j \leq m$

2. $A[i,j] - A[i-1,j], A[i,j] - A[i,j-1] \in \{-1,0,1\} \quad 1 \leq i \leq n;\ 1 \leq j \leq m$

In our algorithm, we divide the matrix A into submatrices, which we call "blocks." A *block* is a submatrix $A[i_{\mathsf{top}}...i_{\mathsf{bottom}}, j_{\mathsf{left}}...j_{\mathsf{right}}]$ consisting of two runs – one of X and one of Y – so that $x_{i_{\mathsf{top}}-1} \neq x_{i_{\mathsf{top}}} = x_{i_{\mathsf{top}}+1} = \ldots = x_{i_{\mathsf{bottom}}} \neq x_{i_{\mathsf{bottom}}+1}$ and $y_{j_{\mathsf{left}}-1} \neq y_{j_{\mathsf{left}}} = y_{j_{\mathsf{left}}+1} = \ldots = y_{j_{\mathsf{right}}} \neq y_{j_{\mathsf{right}}+1}$. Thus, by definition, the matrix $A[0...n, 0...m]$ is divided into exactly $k\ell$ blocks. The blocks are of two types: *black blocks*, corresponding to pairs of identical letters $x_{i_{\mathsf{top}}} = y_{j_{\mathsf{left}}}$, and *white blocks*, corresponding to pairs of distinct letters $x_{i_{\mathsf{top}}} \neq y_{j_{\mathsf{left}}}$. See Figure 1.

We also distinguish between *horizontal blocks*, for which $j_{\mathsf{right}} - j_{\mathsf{left}} \geq i_{\mathsf{bottom}} - i_{\mathsf{top}}$, and *vertical blocks*, for which $i_{\mathsf{bottom}} - i_{\mathsf{top}} \geq j_{\mathsf{right}} - j_{\mathsf{left}}$.

The *diagonal(d)* (defined for $d \in \{-n, -n+1, \ldots, m\}$) of A is the set of elements $A[i,j]$ such that $d = j - i$.

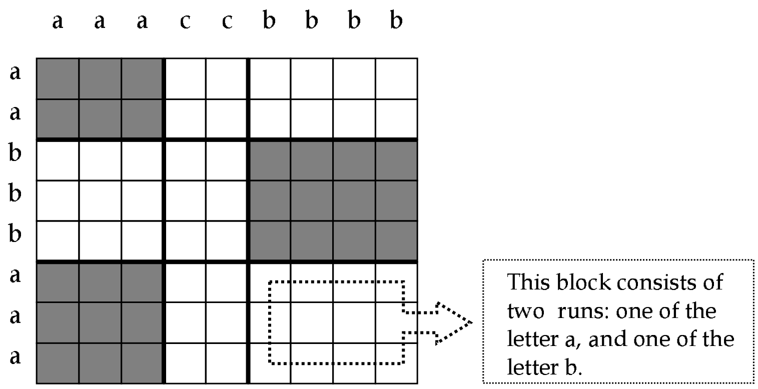


Figure 1: The matrix A is divided into 9 blocks. Three blocks (shaded black) correspond to pairs of identical letters, while the other 6 blocks (white) correspond to pairs of different letters.

3 The Algorithm

Our goal is to reduce significantly the number of elements in the matrix A that are evaluated when computing the edit distance (and corresponding sequence of edit operations) between X and Y. Our algorithm computes only the elements on the bottom-right boundary of each block. It will be shown that the elements interior to the blocks are not essential for the solution of the problem. The computation starts from the top-left block, continues rightward and downward, and ends at the bottom-right block. Section 3.1 presents an algorithm that computes the bottom-right elements of a black block; the computation of elements in a white block is described in Section 3.2.

To simplify the discussion from now on, we consider a block to include as its top row the bottom row of the block that lies immediately above it, and to include as its left column the rightmost column of the block immediately to its left. Thus, the upper-left boundary of the block serves as input to the computation for the block. We will often refer to this upper-left boundary (row and column) as the *frame* of the block.

3.1 Black blocks

Given a black block $A[i_{\mathsf{top}} \dots i_{\mathsf{bottom}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$ our goal is to compute the values of $A[i_{\mathsf{top}} \dots i_{\mathsf{bottom}}, j_{\mathsf{right}}]$ and $A[i_{\mathsf{bottom}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$. By Equation 1, an element of a black block is computed in the following way:

$$A[i,j] = min(A[i-1,j]+1, A[i,j-1]+1, A[i-1,j-1]). \quad (2)$$

Lemma 2 *For every element $A[i,j]$ of a black block, $A[i,j] = A[i-1,j-1]$.*

Proof: If $A[i,j] \neq A[i-1,j-1]$, then by Equation 2 it follows that $A[i,j] = A[i-1,j]+1$ or $A[i,j] = A[i,j-1]+1$. Without loss of generality, we assume that $A[i,j] = A[i-1,j]+1$. Likewise, according to Theorem 1, $A[i,j] = A[i-1,j-1]+1$, which means that $A[i-1,j-1] = A[i-1,j]$. But then, since $A[i,j]$ is located in a black block, we would have given it the value of $A[i-1,j-1]$, in contradiction to the assumption. ∎

Corollary 1 *Given the upper-left boundary of a black block, the value of each element $A[i,j]$ on its bottom-right is computed by copying the value of the element that is on the intersection between the upper-left boundary and the diagonal $d = j - i$.*

Time Complexity. Given the indices of the black block and (i,j), one can find in $O(1)$ time the location of the element on the upper-left boundary that has to be copied into $A[i,j]$. Hence, given a black block $A[i_{\mathsf{top}} \dots i_{\mathsf{bottom}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$, the values of $A[i_{\mathsf{top}} \dots i_{\mathsf{bottom}}, j_{\mathsf{right}}]$ and $A[i_{\mathsf{bottom}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$ are computed in $O((j_{\mathsf{right}} - j_{\mathsf{left}} + 1) + (i_{\mathsf{bottom}} - i_{\mathsf{top}} + 1))$ time.

3.2 White blocks

We show how to compute the values on the right side of the block; computing the elements on the bottom row is done similarly. Thus, given a white block $A[i_{\mathsf{top}} \dots i_{\mathsf{bottom}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$, our goal is to compute the values of $A[i_{\mathsf{top}}+1 \dots i_{\mathsf{bottom}}, j_{\mathsf{right}}]$. By Equation 1, an element of a white block is computed as follows:

$$A[i,j] = min(A[i-1,j]+1, A[i,j-1]+1, A[i-1,j-1]+1). \tag{3}$$

Given two elements, $A[a,b]$ and $A[p,q]$ with $a \leq p, b \leq q$, we define both the *distance* between them and $dis(A[a,b], A[p,q])$ to be the edit distance between $[x_a, \dots x_p], [y_b, \dots y_q]$. If $p < a$ or $q < b$ then $dis(A[a,b], A[p,q]) = \infty$.

Note that in a white block $dis(A[a,b], A[p,q]) = max\{p-a, q-b\}$. Following Equation 1, we get recursively that each element $A[i, j_{\mathsf{right}}]$, for $i_{\mathsf{top}} \leq i \leq i_{\mathsf{bottom}}$, gets its value from an element $A[p,q]$ (where $p \leq i$) on the upper-left frame of the block, and an addition of the distance of this element to $A[i, j_{\mathsf{right}}]$. Hence,

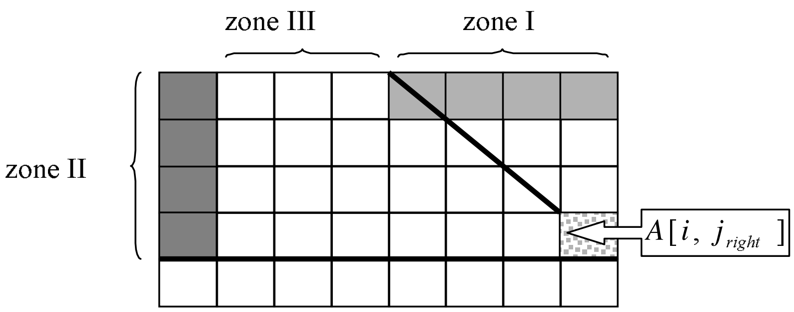
$$A[i, j_{\mathsf{right}}] = \min_{p,q}\{A[p,q] + dis(A[p,q], A[i, j_{\mathsf{right}}])\}, \tag{4}$$

where (p,q) ranges over all the indices with $p \leq i$ corresponding to the upper-left frame of the block.

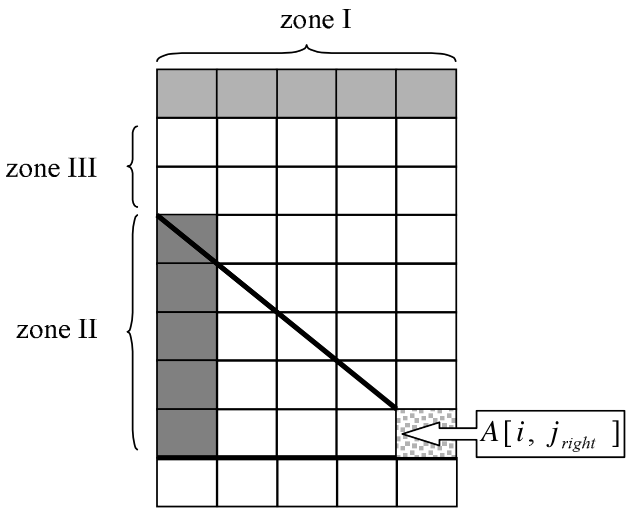
The diagonal $d = j_{\mathsf{right}} - i$ goes from $A[i, j_{\mathsf{right}}]$ up to the upper-left frame of the block. This diagonal crosses the upper-left frame either on the top through the element $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]$ (where $j_{\mathsf{diagonal}} = j_{\mathsf{right}} - (i - i_{\mathsf{top}})$), or on the left through the element $A[i_{\mathsf{diagonal}}, j_{\mathsf{left}}]$ (where $i_{\mathsf{diagonal}} = i - (j_{\mathsf{right}} - j_{\mathsf{left}})$). Given an element $A[i, j_{\mathsf{right}}]$ on the right frame, we divide the upper-left frame into three zones (refer to Figure 2):

Zone I - If $Diagonal(i - j_{\mathsf{right}})$ crosses the top frame, then Zone I consists of elements $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}} \dots j_{\mathsf{right}}]$; otherwise, if $Diagonal(i - j_{\mathsf{right}})$ crosses the left frame, then Zone I consists of elements $A[i_{\mathsf{top}}, j_{\mathsf{left}} \dots j_{\mathsf{right}}]$.

Zone II - If $Diagonal(i - j_{\mathsf{right}})$ crosses the top frame, then Zone II consists of elements $A[i_{\mathsf{top}} \dots i, j_{\mathsf{left}}]$; otherwise, if $Diagonal(i - j_{\mathsf{right}})$ crosses the left frame, then Zone II consists of elements $A[i_{\mathsf{diagonal}} \dots i, j_{\mathsf{left}}]$.



2a



2b

Figure 2: Figure 2a shows a horizontal block, where zone III is on the upper side of the frame. Figure 2b shows a vertical block, where zone III is on the left side of the frame.

Zone III - Zone III consists of all of the elements on the upper-left frame between Zone I and Zone II.

Lemma 3 *The distance between each element in Zone I and $A[i, j_{\mathsf{right}}]$ is $(i - i_{\mathsf{top}} + 1)$; the distance between each element in Zone II and $A[i, j_{\mathsf{right}}]$ is $(j_{\mathsf{right}} - j_{\mathsf{left}} + 1)$.*

Proof: By definition, the distance of element $A[i, j_{\mathsf{right}}]$ to any other element $A[i_{\mathsf{top}}, j]$ of Zone I is the edit distance between the strings $[y_j, \ldots y_{j_{\mathsf{right}}}]$ and $[x_{i_{\mathsf{top}}}, \ldots x_i]$. These two elements ($A[i, j_{\mathsf{right}}]$ and $A[i_{\mathsf{top}}, j]$) are placed in a white block; hence, there are no matches between them. In this case, the edit distance between the strings is always the length of the longer of them. In the case of Zone I, the length of the string $[x_{i_{\mathsf{top}}}, \ldots x_i]$ is always greater than or equal to the length of the string $[y_j, \ldots y_{j_{\mathsf{right}}}]$, for all j in Zone I. Thus, we clearly get that the distance is the length of $[x_{i_{\mathsf{top}}}, \ldots x_i]$, which is $(i - i_{\mathsf{top}} + 1)$. It is similarly true for Zone II, except that the longer string is now $[y_{j_{\mathsf{left}}} \cdots y_{j_{\mathsf{right}}}]$, and its length is $j_{\mathsf{right}} - j_{\mathsf{left}} + 1$. ∎

Before starting the calculation, notice that

Lemma 4 *Zone III is unnecessary.*

Proof: Consider a horizontal block. We are looking for an element that minimizes the sum of its value and its distance to $A[i, j_{\mathsf{right}}]$. Let $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]$ be the leftmost element in Zone I, and let $A[i_{\mathsf{top}}, c]$ be an element in Zone III. We will show that

$$A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}] + dis(A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}], A[i, j_{\mathsf{right}}]) \leq A[i_{\mathsf{top}}, c] + dis(A[i_{\mathsf{top}}, c], A[i, j_{\mathsf{right}}]). \tag{5}$$

It is easy to see that

$$dis(A[i_{\mathsf{top}}, c], A[i, j_{\mathsf{right}}]) = dis(A[i_{\mathsf{top}}, c], A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]) + dis(A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}], A[i, j_{\mathsf{right}}]),$$

and, from Theorem 1, we know that

$$A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}] \leq A[i_{\mathsf{top}}, c] + dis(A[i_{\mathsf{top}}, c], A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]).$$

Hence,

$$A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}] + dis(A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}], A[i, j_{\mathsf{right}}]) \leq$$
$$A[i_{\mathsf{top}}, c] + dis(A[i_{\mathsf{top}}, c], A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]) + dis(A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}], A[i, j_{\mathsf{right}}]),$$

and so

$$A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}] + dis(A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}], A[i, j_{\mathsf{right}}]) \leq A[i_{\mathsf{top}}, c] + dis(A[i_{\mathsf{top}}, c], A[i, j_{\mathsf{right}}]),$$

as Equation 5 claims. It follows immediately that the element $A[i_{\mathsf{top}}, c]$ will never minimize Equation 4. ∎

We have proved that an element $A[i, j_{\mathsf{right}}]$ gets its value either from an element in Zone I or from an element in Zone II. In the next two subsections we will find the two elements that produce the minimum value for $A[i, j_{\mathsf{right}}]$, one from Zone I and one from Zone II. Then, $A[i, j_{\mathsf{right}}]$ will choose the minimum of these two candidates.

3.2.1 Zone I

Our goal is to find, for each element $A[i, j_{\mathsf{right}}]$ on the right part of the frame, the element in Zone I that provides the minimum value for equation 4. As was shown in Lemma 3, the distance of all the elements $A[p, q]$ in Zone I to $A[i, j_{\mathsf{right}}]$ is equal to $(i - i_{\mathsf{top}} + 1)$; therefore, we have only to find the minimum value among the elements in Zone I. When considering the top element $A[i_{\mathsf{top}} + 1, j_{\mathsf{right}}]$, Zone I contains only two elements, so we start by finding the minimum value of them. Then we compute the minimum for $A[i_{\mathsf{top}} + 2, j_{\mathsf{right}}]$ to $A[i_{\mathsf{bottom}}, j_{\mathsf{right}}]$. The elements in Zone I that are considered for the computation of $A[i, j_{\mathsf{right}}]$ are either the same as the elements that are considered in the computation of $A[i - 1, j_{\mathsf{right}}]$ (when $j_{\mathsf{right}} - j_{\mathsf{left}} \leq i - i_{\mathsf{top}}$), or expanded with one element $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]$, the element on $diagonal(j_{\mathsf{right}} - i)$. Hence, the minimum value is either the one that was computed for $A[i-1, j_{\mathsf{right}}]$ or $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]$. The array MinZoneI will hold the minimum values of Zone I for all computed elements.

The following algorithm computes the minimum of Zone I:

Algorithm 1

for $i = i_{\mathsf{top}}$ to i_{bottom} do
  if $(i - i_{\mathsf{top}} \leq j_{\mathsf{right}} - j_{\mathsf{left}})$ then
    MinZoneI[i]:=min(MinZoneI[$i - 1$], $A[i_{\mathsf{top}}, j_{\mathsf{diagonal}}]$)
  else
    MinZoneI[i]:= MinZoneI[$i - 1$]

3.2.2 Zone II

Our goal is to find, for each element $A[i, j_{\mathsf{right}}]$ on the right part of the frame, the element $A[p, q]$ in Zone II that provides the minimum value for the sum

$$(A[p, q] + dis(A[p, q], A[i, j_{\mathsf{right}}])). \tag{6}$$

As was shown in Lemma 3, the distance of each element in Zone II to $A[i, j_{\mathsf{right}}]$ is equal to $(j_{\mathsf{right}} - j_{\mathsf{left}} + 1)$; therefore, it is sufficient to find the minimum value among the elements in Zone II. We start by finding the minimum value in Zone II for $A[i_{\mathsf{top}} + 1, j_{\mathsf{right}}]$, then for $A[i_{\mathsf{top}} + 2, j_{\mathsf{right}}]$, and then continue until we find the minimum value in Zone II for $A[i_{\mathsf{bottom}}, j_{\mathsf{right}}]$. In the beginning, the size of Zone II is two; it then grows to $(j_{\mathsf{right}} - j_{\mathsf{left}} + 1)$, and, after Zone II reaches its maximum size, in each iteration one element is deleted and one is added.

Observation 1 *Let s, w be two values in a given Zone II; without loss of generality, we assume that $s < w$. Then, following Theorem 1, all of the values between s and w must also appear in the zone.*

From this observation we get that for any Zone II, if the minimum value in the zone is a and the maximum value is b, then the values $a, a + 1, \ldots b - 1, b$ must appear in that zone.

In order to compute the minimum of the changing zone, we maintain a variable, Min, which keeps the minimum value of Zone II elements through the entire computation. The variable New keeps the value of the new element that is added to the zone ($A[i, j_{\mathsf{left}}]$), and Out is the one that is deleted from it ($A[i_{\mathsf{diagonal}} - 1, j_{\mathsf{left}}]$). In addition, we keep counters for each value in the zone by maintaining an array "counter" of size n. We assume that $m \leq n$; hence, the values in Zone II are at most n. The array MinZoneII holds the minimum value of Zone II elements for all computed elements.

The following algorithm computes the minimum value in Zone II:

Algorithm 2

for i=i_{top} to i_{bottom} do
  counter[New]:=counter[New]+1
  $Min := min(Min, New)$
  counter[Out]:=counter[Out]-1
  if counter[Min]=0 then[1] $Min = Min + 1$
  MinZoneII[i]:=Min

Time Complexity. Given a white block, the arrays MinZoneI and MinZoneII are computed in $O((j_{\text{right}} - j_{\text{left}} + 1) + (i_{\text{bottom}} - i_{\text{top}} + 1))$, using Algorithm 1 and Algorithm 2 respectively. Given a location (i, j_{right}) in a white block, one can find in $O(1)$ time the minimums of Zone I and Zone II for (i, j_{right}) from the arrays MinZoneI and MinZoneII. Hence, given a white block $A[i_{\text{top}} \dots i_{\text{bottom}}, j_{\text{left}} \dots j_{\text{right}}]$, the values of $A[i_{\text{top}} \dots i_{\text{bottom}}, j_{\text{right}}]$ and $A[i_{\text{bottom}}, j_{\text{left}} \dots j_{\text{right}}]$ are computed in $O((j_{\text{right}} - j_{\text{left}} + 1) + (i_{\text{bottom}} - i_{\text{top}} + 1))$ time.

Theorem 5 *The edit distance between two run-length encoded strings, X and Y, can be computed in $O(k \cdot m + l \cdot n)$ time.*

Proof: We have shown that the work for each block (black or white) is linear in the size of its bottom-right boundary. Hence, the total time complexity is linear in the total size of the boundaries of all blocks, and that is exactly $k \cdot m + l \cdot n$. ∎

3.3 Example

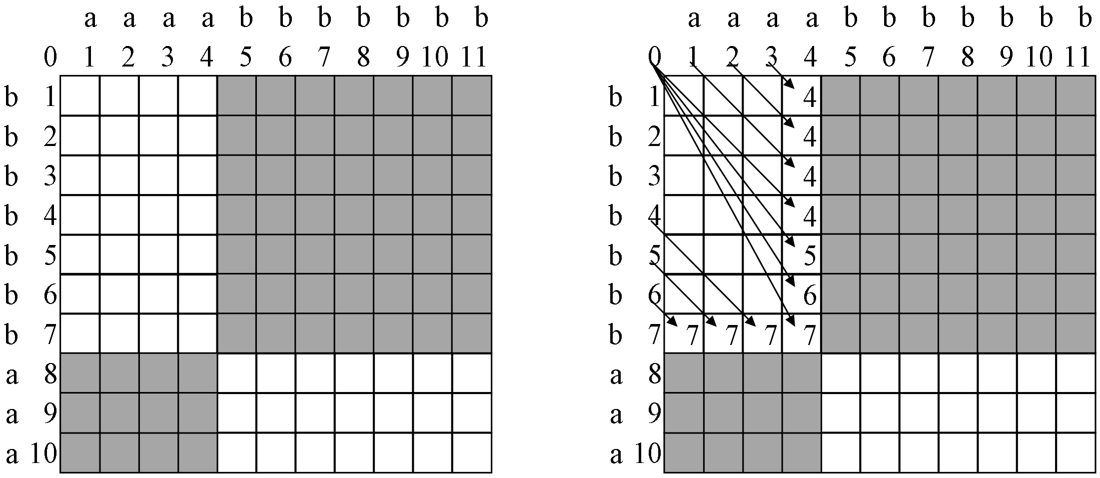
We now give an example to show how the algorithm works. We let $X = aaaabbbbbbb$ and $Y = bbbbbbbaaa$. The initial matrix is shown in Figure 3.

The first block to be calculated is the upper-left one, corresponding to the runs *aaaa* and *bbbbbbb*. This block is white and is vertical. The first element to be considered is $A[1, 4]$. Zone I now has only two values: 3 and 4. We choose 3. Zone II also has two values: 0 and 1. We choose 0. We compare $3+1$ with $0+4$ and get $A[1, 4] = 4$. Similarly, we get $A[2, 4] = A[3, 4] = A[4, 4] = 4$. We now consider $A[5, 4]$. It is the first element in which Zone II is all inside the block; thus, in each step down, it loses one element from its upper side, and gains a new one from its bottom side. For $A[5, 4]$ we already have the minimum value of Zone I, since it already contains the whole upper side of the block. Thus, it does not change anymore. The minimum value of Zone I remains 0 for the rest of this block. We have left to compute the minimum value of Zone II. The minimum value for $A[4, 4]$ was 0. After we move to $A[5, 4]$, 0 got out from Zone II so the new minimum will be 1. Finally, the comparison between the values received from the two zones gives $A[5, 4]$ the value 5. In exactly the same way we give $A[6, 4]$ the value 6, and $A[7, 4]$ gets the value 7.

To calculate the bottom part of the block, we make the same considerations, only Zone I and Zone II replace each other in their positions. The result is shown in Figure 3.

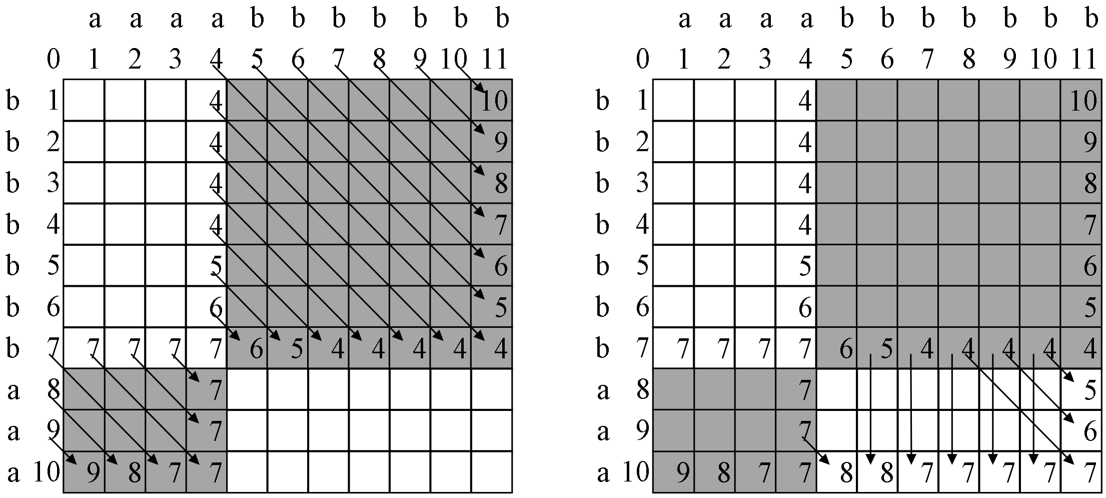
The next two blocks are black. For each element on the bottom-right part of the frame, we find the element on the upper-left part that is on the same diagonal and copy its value. The result is shown in Figure 4. The last block is white, and its calculation is similar to the first one. The final calculated matrix is shown in Figure 4.

[1] According to Observation 1, the new minimum is $Min + 1$, since it exists in Zone II.



a b

Figure 3: Part 3a shows the initial matrix corresponding to strings X and Y. Part 3b shows the matrix after calculating the first white block.



c d

Figure 4: Part 4c shows the matrix after calculating the two black blocks. Part 4d shows the calculated matrix.

Acknowledgement
We would to thank the anonymous referees for their helpful suggestions.

References

[1] Apostolico, A., G.M. Landau and S. Skiena, Matching for Run Length Encoded Strings, *Journal of Complexity*, **15**, 1, 4–16 (1999).

[2] Bunke, H. and J. Csirik, An Improved Algorithm for Computing the Edit Distance of Run Length Coded Strings, *Information Processing Letters*, **54**, 93–96 (1995).

[3] Crochemore, M., G.M. Landau and M. Ziv-Ukelson, A Sub-Quadratic Sequence Alignment Algorithm for Unrestricted Cost Matrices, *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 679-688 (2002).

[4] Levenshtein, V.I., Binary Codes Capable of Correcting, Deletions, Insertions and Reversals, *Soviet Phys. Dokl*, **10**, 707–710 (1966).

[5] Mäkinen, V., G. Navarro and E. Ukkonen, Approximate Matching of Run-Length Compressed Strings, *Proc. 12th Combinatorial Pattern Matching Symposium*, Lecture Notes in Computer Science **2089**, Springer-Verlag, 31–49 (2001).

[6] Mitchell, J., A Geometric Shortest Path Problem, with Application to Computing a Longest Common Subsequence in Run-Length Encoded Strings, Technical Report, Dept. of Applied Mathematics, SUNY Stony Brook, 1997.

[7] Sankoff D. and J.B. Kruskal (editors), *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, MA, 1983.

[8] Ukkonen, E., Algorithms for Approximate String Matching, *Information and Control*, **64**, 100–118 (1985).

[9] Ukkonen, E., On Approximate String Matching, *J. of Algorithms*, **6**, 132–137 (1985).