Edit distance of run length encoded strings

Ora Arbel, Gad M. Landau, Joseph S.B Mitchel

Dynamic programming algorithms course - 2008

Presented by Aviad Pinkovezky
Introduction – Edit Distance

• Edit distance is a known method to measure the similarity between two given strings.
• Edit distance is the minimum number of operations required to transform one string into the other.
• Permitted operations are: Substitution, Deletion and Insertion.
• A simple example - the edit distance between “abc” and “dc” is 2:
  ◦ “abc” → “bc” (deletion)
  ◦ “bc” → “dc” (substitution)
• Run time – $O(m*n)$
Introduction – Run Length Encoded Strings

- A string is run-length encoded if it can be described as an ordered sequence of pairs \((\sigma, i)\) when \(\sigma\) is a character and \(i\) is an integer.
- The meaning of \((\sigma, i)\): the character \(\sigma\) appears \(i\) consecutive times.
- Example: the string “aaaabbbbcxccabbbbc” will be encoded as: “a^4b^4c^3a^1b^4c^2”.
- Motivation: Image compression (Faxes), Multimedia Compression, Bioinformatics, etc.
The O(m*n) algorithm is based on the following calculation:

\[ A[i,j] = \min\{ A[i,j-1]+1, A[i-1,j]+1, A[i-1,j-1] + t_{i,j} \} \]

Initial values are: \( A[i,0] = i, A[j,0] = j \)

Insert: \( \rightarrow \)
Delete: \( \downarrow \)

- if \( x_i = y_j \) then \( t_{i,j} = 0 \): no operation
- if \( x_i \neq y_j \) then \( t_{i,j} = 1 \): Substitute
Blocks:

- **Black Block**
- **Vertical Block**
- **White Block**

Horizontal Block

Diagonal
The algorithm’s main idea:

- We want to reduce the number of matrix elements that are evaluated.
- We will show that it’s enough to calculate only the elements on the bottom-right border of each block.
- Order of computation - start from the top-left block, continue rightward and downward.
- To simplify discussion - a block’s top row is the bottom row of the block above it, a block’s left column is the rightmost column of the block immediately to its left.
The algorithm’s main idea (cont.):

We’ll obtain time complexity of $O(k \cdot m + l \cdot n)$:

\[ \begin{align*}
    y_1 & \quad y_2 & \quad y_3 & \quad y_4 & \quad y_5 & \quad y_6 & \quad y_7 & \quad y_8 & \quad y_9 = m \\
    x_1 & & & & & & & & & \\
    x_2 & & & & & & & & & \\
    x_3 & & & & & & & & & \\
    x_4 & & & & & & & & & \\
    x_5 & & & & & & & & & \\
    x_6 & & & & & & & & & \\
    x_7 & & & & & & & & & \\
    x_8 & & & & & & & & & \\
\end{align*} \]

\[ \begin{align*}
    l &= 3 \\
    k &= 3
\end{align*} \]
Theorem 1 (Ukkonen et al):

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

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

Examples:

\[
\begin{array}{c|c|c|c|c}
3 & 3 & 3 & 3 & 3 \\
3 & 4 & 2 & 3 & 4 \\
\end{array}
\]
Black blocks - Lemma 2:

For every element $A[i,j]$ of a black block,


**Explanation:** Remember that by taking the diagonal path in a black block we take no edit operation. If $A[i-1,j-1] = X$, then in the “best” case, due to theorem 1, we get $A[i-1,j] = X - 1$, (and / or $A[i,j-1] = X - 1$). Either way, $A[i,j]$ will get $X$, so by not choosing the diagonal path we didn’t “earn” anything. □

**Corollary:** Given the upper-left boundary of a black block, the values of each element \( A[i,j] \) on the bottom-right boundary is computed by copying the value of the element that is on the intersection between the upper-left boundary and the diagonal which starts at \( A[i,j] \).
Black blocks (cont.):

For every element $A[i,j]$ of a black block, 

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 $A[i_{\text{top}} \ldots i_{\text{bottom}}, j_{\text{left}} \ldots j_{\text{right}}]$, the values of the right-bottom boundary can be calculated in $O((j_{\text{right}} - j_{\text{left}}) + (i_{\text{bottom}} - i_{\text{right}}))$ time.
White blocks:

We will show how to calculate the elements on the right side of the block, computing the elements on the bottom row is done similarly.

**Definition:** $\text{dis}(A[a,b], A[p,q])$ is the edit distance between $[x_a, \ldots, x_p]$ and $[y_b, \ldots, y_q]$. If $p < a$ or $q < b$ then it’s $\infty$. 
White blocks:

- Note that within white blocks $\text{dis}(A[a,b], A[p,q]) = \max(p-a+1, q-b+1)$, that’s because we’re assured there are no similarities between the two words, so for each letter we need exactly one operation.

- If we’re to assume that $A[a,b]$ is in the top-left boundary, which is in a black block, then we get $\text{dis}(A[a,b], A[p,q]) = \max(p-a, q-b)$, that’s because $x_a = y_b$ and one operation is saved.
White blocks (cont.):

We know that in a white block:

\[ A[i,j] = \min\{ A[i,j-1]+1, A[i-1,j]+1, A[i-1,j-1]+1 \} \]

So we can get recursively that each element \( A[i,j_{\text{right}}] \) gets its value from an element \( A[p,q] \) which is on the top-left boundary, plus the distance between the two elements. Hence we get:

\[ A[i,j_{\text{right}}] = \min\{ A[p,q] + \text{dis}(A[p,q], A[i,j_{\text{right}}]) \} \]

(note that \((p,q)\) ranges over all the indices with \( p \leq i \) corresponding to the upper left boundary of the block)
White blocks - zones:
White blocks – zones (cont.):

Zone 1

Zone 2

Zone 3

$A[i,j_{\text{right}}]$
White blocks - zones - lemma 3:

The distance between each element in Zone 1 and $A[i,j_{\text{right}}]$ is $(i-i_{\text{top}}+1)$;

The distance between each element in Zone 2 and $A[i,j_{\text{right}}]$ is $(j_{\text{right}}-j_{\text{left}}+1)$.

**Proof:** Because all of our elements are in the white block, it holds for zone 1 that: $\text{dis}(A[i_{\text{top}},j], A[i,j_{\text{right}}]) = \max\{(j_{\text{right}}-j+1), (i-i_{\text{top}}+1)\}$, but for zone 1 we can easily see that $(i-i_{\text{top}}+1) \geq (j_{\text{right}}-j+1)$.

For zone 2 we get: $\text{dis}(A[i_1,j_{\text{left}}], A[i_2,j_{\text{right}}]) = \max\{(j_{\text{right}}-j_{\text{left}}+1), (i_2-i_1+1)\}$, but for zone 2 we can easily see that $(j_{\text{right}}-j_{\text{left}}+1) \geq (i_2-i_1+1)$. □
White blocks - zones - lemma 4:

Zone 3 is unnecessary.

**Intuition:** The editing distance between $A[i,j_{right}]$ and the element in zone 3 will increase, while the value of the element in zone 3 can decrease at the same ratio, at best.

**Proof:** We’ll look at a horizontal block. Let $A[i_{top},j_{diagonal}]$ be the leftmost element in Zone 1. Let $A[i_{top},c]$ be an element in Zone 3. We will show that:

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

It’s easy to see that:

(*) $\text{dis}(A[i_{top},c], A[i,j_{right}]) = \text{dis}(A[i_{top},c], A[i_{top},j_{diagonal}]) + \text{dis}(A[i_{top},j_{diagonal}], A[i,j_{right}])$
And from Theorem 1 (Because the difference between each two neighbor elements is either 0, 1 or -1) we know that:

\[ c - j_{\text{diagonal}} \leq A[i_{\text{top}}, j_{\text{diagonal}}] - A[i_{\text{top}}, c] \leq j_{\text{diagonal}} - c = \text{dis}(A[i_{\text{top}}, c], A[i_{\text{top}}, j_{\text{diagonal}}]) \]

So now we get:

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

Hence, by adding \(-\,\text{dis}(A[i_{\text{top}}, j_{\text{diagonal}}], A[i, j_{\text{right}}])\), we get:

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

Now, because of (*) we get:

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

So we proved that using \(A[i_{\text{top}}, c]\) will never minimize \(A[i, j_{\text{right}}]\). \(\Box\)
The algorithm - Zone 1:

• Reminder 1: 
  \[ A[i,j_{right}] = \min\{A[p,q] + \text{dis}(A[p,q], A[i,j_{right}])\} \]

• For each \( A[i,j_{right}] \) we want to find the elements from Zone 1 and Zone 2 that produce the above minimal values for \( A[i,j_{right}] \). We’ll choose the value for \( A[i,j_{right}] \) by the minimum of those two.

• We will start with finding the minimum value that can be obtained by using Zone 1.

• Reminder 2: Let \( A[p,q] \) be an element in Zone 1, then:
  \[ \text{dis}(A[p,q], A[i,j_{right}]) = (i - i_{top} + 1) \]

• Therefore, for each \( A[i,j_{right}] \) we need to find the minimum value among the elements in Zone 1.
The algorithm - Zone 1:

- When considering the top element on the rightmost boundary - $A[i_{top}+1, j_{right}]$, Zone 1 contains only two elements.
- Whenever we move down one element on the rightmost boundary, one element is added to Zone 1, so the minimum value for the new $A[i, j_{right}]$ is min(new element in Zone 1 which is $A[i_{top}, j_{diagonal}]$, minimum for $A[i-1, j_{right}]$).
The algorithm - Zone 1 (cont.):

- The following algorithm computes the minimum of Zone 1:
  
  \[
  \text{for } i = i_{\text{top}} \text{ to } i_{\text{bottom}} \text{ do}
  \]
  
  \[
  \text{if } (i - i_{\text{top}} \leq j_{\text{right}} - j_{\text{left}}) \text{ then} \quad \text{// we can add another element to zone 1}
  \]
  
  \[
  \text{MinZoneI}[i] = \min(\text{MinZoneI}[i-1], A[i_{\text{top}}, j_{\text{diagonal}}])
  \]
  
  \[
  \text{else} \quad \text{// means zone 1 consist already all of the top row}
  \]
  
  \[
  \text{MinZoneI}[i] = \text{MinZoneI}[i-1]
  \]
  
- The same algorithm will calculate Zone 1 which is related to the bottom most boundary - with the for loop running from \(j_{\text{left}}\) to \(j_{\text{right}}\).
  
- **Time Complexity:** \(O((i_{\text{bottom}} - i_{\text{top}}) + (j_{\text{right}} - j_{\text{left}}) + 1)\)
The algorithm - Zone 2:

- Reminder: Let $A[p,q]$ be an element in Zone 2, then:
  \[
  \text{dis}(A[p,q], A[i,j_{\text{right}}]) = (j_{\text{right}} - j_{\text{left}} + 1)
  \]

- Therefore, for each $A[i,j_{\text{right}}]$ we need to find the minimum value among the elements in Zone 2.

- Unlike Zone 1, when zone 2 reaches its maximal size, when we’re moving downward $A[i,j_{\text{right}}]$, one element is removed from zone 2 and a new element is added, as you can see in the following illustrations:
The algorithm - Zone 2 (cont.):

- We maintain a variable, \textit{Min}, which keeps the minimum of Zone 2 elements through the computation (for each \(A[i,j_{\text{right}}]\) we recalculate it).
- We maintain a variable, \textit{New}, that keeps the value of the new element that is added to the zone.
- We maintain a variable, \textit{Old}, that keeps the value of the element that is removed from the zone.
- We maintain a \textit{counter} array, which counts the appearances of each value in the zone (\(\text{counter}[i] = j\) when \(i\) appears \(j\) times in Zone 2).

\[\begin{array}{|c|c|c|c|c|c|}
\hline
3 & 3 & 4 & 5 & \\
\hline
\end{array}\]

\text{Counter:} 
\begin{array}{cccccc}
0 & 0 & 0 & 2 & 1 & 1 & 0
\end{array}
The algorithm - Zone 2 (cont.):

- **Observation:** Let $s, w$ be two values in a given Zone 2, 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.

- A simple running example:
The algorithm - Zone 2 (cont.):

- The following algorithm computes the minimum of Zone 2:

  for $i = i_{\text{top}}$ to $i_{\text{bottom}}$ do 
  
  $counter[\text{New}] = counter[\text{New}] + 1$
  
  $Min = \min(\text{Min}, \text{New})$
  
  $counter[\text{Out}] = counter[\text{Out}] - 1$
  
  if $counter[\text{Min}] = 0$ then $Min = Min + 1$ // according to the observation, 
  // the new minimum is $Min+1$, since it exists in Zone 2
  
  $Min_{\text{ZoneII}}[i] = Min$

- The same algorithm will calculate Zone 2 which is related to the bottom 
  most boundary - with the for loop running from $j_{\text{left}}$ to $j_{\text{right}}$.

- **Time Complexity:** $O((i_{\text{bottom}} - i_{\text{top}}) + (j_{\text{right}} - j_{\text{left}}) + 1)$
The algorithm - Time complexity:

- Computing zones 1 and 2 takes $O((i_{bottom} - i_{top}) + (j_{right} - j_{left}))$ time.
- After we have their values we still need to calculate the values of the bottom - right boundary, which takes another $O((i_{bottom} - i_{top}) + (j_{right} - j_{left}))$ time. That is also the total time complexity of calculating each white block.

**Theorem 5:**
The edit distance between two run-length encoded strings, $X$ and $Y$, can be computed in $O(k*m + l*n)$ time. ($X=x_1...x_n$ with encoded length $k$, $Y=y_1...y_m$ with encoded length $l$).

**Proof:**
The work on each block, black or white, is linear in the size of the block’s bottom - right boundary. Hence, the total time complexity is linear in the total size of the boundaries of all blocks. We have $k$ rows of blocks, each row’s length is $m$, so we have $k*m$ operations to compute all bottom boundaries. The same goes with computing right boundaries - $l$ blocks columns, each column’s length is $n$, total operations number is $l*n$. We get total $O(k*m + l*n)$. 
The algorithm - Time complexity (cont.):

Run time complexity is: $O(k \times m + l \times n)$
Example:
Example (cont.):
Example (cont.):
Example (cont.):

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>b</td>
<td>1</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>3</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>4</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>b</td>
<td>6</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>b</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>a</td>
<td>8</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>a</td>
<td>9</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>a</td>
<td>10</td>
<td></td>
<td></td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Example (cont.):
Example - backtracking:

Add $a$

Replace 3 $b$’s with $a$’s

4 nops

Replace 3 $a$’s with $b$’s