Edit Distance/Alignment Algorithm

March 24, 2018

What follows is a loose translation of Crochemore, Hancart, Lecroq, Algorithmique du Texte, *Vuibert, 2001, pp. 224-238. I abbreviated a bit, by omitting some theoretical results and proofs: the resulting document focuses on the programming aspects, while still giving a good overview of the equivalence between the Edit Distance problem and the Alignment problem. NPR*

7.1 Comparing strings

This section introduces the notions of distance between strings, edit operations, alignment and alignment graph.

Distance and edit operations

What is of interest here is the notion of similarity between two strings x and y of respective lengths m and n, or its counterpart, i.e. the distance between two strings.

(. . .)

The **Hamming distance** provides a simple (but not always relevant) way to compare two strings. For two strings that have the same length, it is defined as the number of positions that have different corresponding letters. By contrast, this section focuses on the distances defined in terms of the operations needed to transform string x into string y. Three kinds of elementary operations are to be considered, which we call **edit operations**:

- **substitution** of a letter of x at a given position with a letter of y
- **deletion** of a letter of x at a given position
- **insertion** of a letter of y at a given position

A positive integer associated to each operation represents its cost. Given a, $b \in A$, we use the following notation:

- $Sub(a, b)$ is the cost of substituting letter a with letter b;
- $Del(a)$ is the cost of deleting letter a;
- $Ins(b)$ is the cost of inserting letter b.

We assume that the costs do not depend on the positions where these operations happen. Provided with these elementary costs, we state:

$$EditDist(x, y) = min\{\text{cost of } \sigma : \sigma \in S_{x,y}\}$$

Operation	Resulting string	Cost
substitute A with A	A C G A	0
substitute C with T	A T G A	1
substitute G with G	A T G A	0
insert C	A T G C A	1
insert T	A T G C T A	1
substitute A with A	A T G C T A	0

Figure 1: An example, that illustrates the concept of edit distance. The table above shows a sequence of elementary operations that transform string ACGA into string ATGCTA. If, for all letters a, $b \in A$, we have costs $\mathrm{Sub}(a, a) = 0, \mathrm{Sub}(a, b) = 1$ when $a \neq b$, and $\mathrm{Del}(a) = \mathrm{Ins}(a) = 1$, the total cost of the sequence of edit operations is $0 + 1 + 0 + 1 + 1 + 0 = 3$. It is easy to verify that we can not improve on this cost. Alternatively, the edit distance between the two strings, i.e. $\mathrm{EditDist}(\texttt{ACGA}, \texttt{ATGCTA})$, is equal to 3.

where $S_{x,y}$ is the set of all sequences of elementary operations that transform x into y, and the cost of an element[1] $\sigma \in S_{x,y}$ is the sum of the costs of the edit operations that constitute sequence σ. The procedure $\mathrm{EditDist}(x, y)$ is a distance function from A^* to $\mathbb{Z}$. We call it the **edit distance** or **alignment distance**. Figure 1 illustrates the notions we just introduced.

The Hamming distance mentioned above is a special case of the edit distance, in which we consider only the substitution operation. It amounts to stating $\mathrm{Del}(a) = \mathrm{Ins}(a) = +\infty$, for each alphabet letter, keeping in mind that the two strings must have the same length.

The problem of computing $\mathrm{EditDist}(x, y)$ consists of finding a sequence of edit operations that transform x into y, while minimizing the total cost of the operations involved. In general, computing the distance between x and y amounts to maximizing a specific measure of similarity between these two strings. Any solution is not necessarily unique and can be expressed as a sequence of elementary substitution, deletion, and insertion operations.

Alignments

An **alignment** between two strings $x, y \in A^*$, with respective lengths m and n, is a way to visualize their similarity, as shown in Figure 2, where z, an alignment of x and y, is a sequence of pairs $(\bar{x}_i, \bar{y}_i)$. If we exclude the pair (ϵ, ϵ) from the set of possible pairs, $\bar{x}_i$ is either a letter of x or the empty string (ϵ), and $\bar{y}_i$ is either a letter of y or the empty string (ϵ).

An **aligned pair** (a, b) with $a, b \in A$ denotes the substitution of letter a with letter b. An aligned pair (a, ϵ) with $a \in A$ denotes the deletion of letter a. Finally, an aligned pair (ϵ, b) denotes the insertion of letter b. We often replace the ϵ symbol with the "-" symbol, also called a **gap**.

The cost of an aligned pair is defined as follows:

$$\begin{aligned}\mathrm{cost}(a, b) &= \mathrm{Sub}(a, b)\\ \mathrm{cost}(a, \epsilon) &= \mathrm{Del}(a)\\ \mathrm{cost}(\epsilon, b) &= \mathrm{Ins}(b)\end{aligned}$$

for $a, b \in A$. The cost of an alignment is the sum of the costs associated with every aligned pair.

The number of alignments between two strings is exponential. The following lemma gives an lower bound for the total number of alignments:

Lemma 7.1.1 *Let x and $y \in A$ be two strings of respective lengths m and n, with $m \leq n$. There are $\binom{2n+1}{m}$ alignments between x and y that do not contain consecutive deletions of letters*

[1](Translator's note) Just to restate the obvious, an element of $S_{x,y}$ is a sequence of operations.

Operation	Aligned pair	Cost
substitute `A` with `A`	`(A,A)`	0
substitute `C` with `T`	`(C,T)`	1
substitute `G` with `G`	`(G,G)`	0
insert `C`	`(-,C)`	1
insert `T`	`(-,T)`	1
substitute `A` with `A`	`(A,A)`	0

Figure 2: Continuation of the example shown in Figure 1. The aligned pairs are represented in the table above. The corresponding alignment is denoted `(A, A)(C, T)(G, G)(-, C)(-, T)(A, A)`, or also $\begin{pmatrix} \mathtt{A} & \mathtt{C} & \mathtt{G} & \mathtt{-} & \mathtt{-} & \mathtt{A} \\ \mathtt{A} & \mathtt{T} & \mathtt{G} & \mathtt{C} & \mathtt{T} & \mathtt{A} \end{pmatrix}$. This alignment is optimal, for its cost $0 + 1 + 0 + 1 + 1 + 0 = 3$ is the edit distance between the two strings.

of x.

Proof Every alignment of that particular kind is uniquely characterized by the location of the substitutions over the n positions of y, as well as by the location of the deletions between y's letters: there are exactly $n + 1$ locations for the deletions (including the ability to perform a deletion before $y[0]$ and after $y[n-1]$. The alignment is therefore characterized par the choice of substitutions or deletions at the $2n + 1$ possible locations, which results in the given lower bound.

Alignment graph

An alignment can be expressed as a graph. The **alignment graph** $G(x, y)$ of two strings x and $y \in A^*$ of respective lengths m and n is shown in Figure 3.

The set of vertices is the Cartesian product $V = \{-1, 0, 1, \cdots, m-1\} \times \{-1, 0, 1, \cdots, n-1$Âă$\}$[2]

Every edge label is defined by the following labeling function:

$$\begin{aligned} label((i-1, j-1), (i, j)) &= (x[i], y[j]) \\ label((i-1, j), (i, j)) &= (x[i], \epsilon), \\ label((i, j-1), (i, j)) &= (\epsilon, y[j]). \end{aligned}$$

Any path that starts at vertex $(-1, -1)$ and ends at vertex $(m-1, n-1)$ is labeled with an alignment of x and y. Alternatively, by setting $(-1, -1)$ as the start state, and $(m-1, n-1)$ as the accept state, the alignment graph can be interpreted as an automaton, that recognizes all alignments of x and y. The cost of an edge e is the cost of its label, i.e. $cost(label(e))$.

Computing an optimal alignment or $\mathrm{EditDist}(x, y)$ comes down to searching the graph $G(x, y)$ for a path of minimal cost that starts at $(-1, -1)$ and ends at $(m-1, n-1)$. There is a one-to-one correspondence between the paths of minimal cost from $(-1, -1)$ to $(m-1, n-1)$ and the optimal alignments of x and y. Since G is an acyclic graph, we can find a path of minimal cost by considering each vertex only once, in their topological order, i.e. by processing vertex rows from the top down, and every row from left to right. Alternatively, we could consider vertex columns from left to right, and every column from the top down. This computation can be performed by a dynamic programming technique, as shown in the next section.

[2](Translator's note) Informally: all coordinates (x_i, y_j), with x_i and y_j being the positions in x and y, respectively.

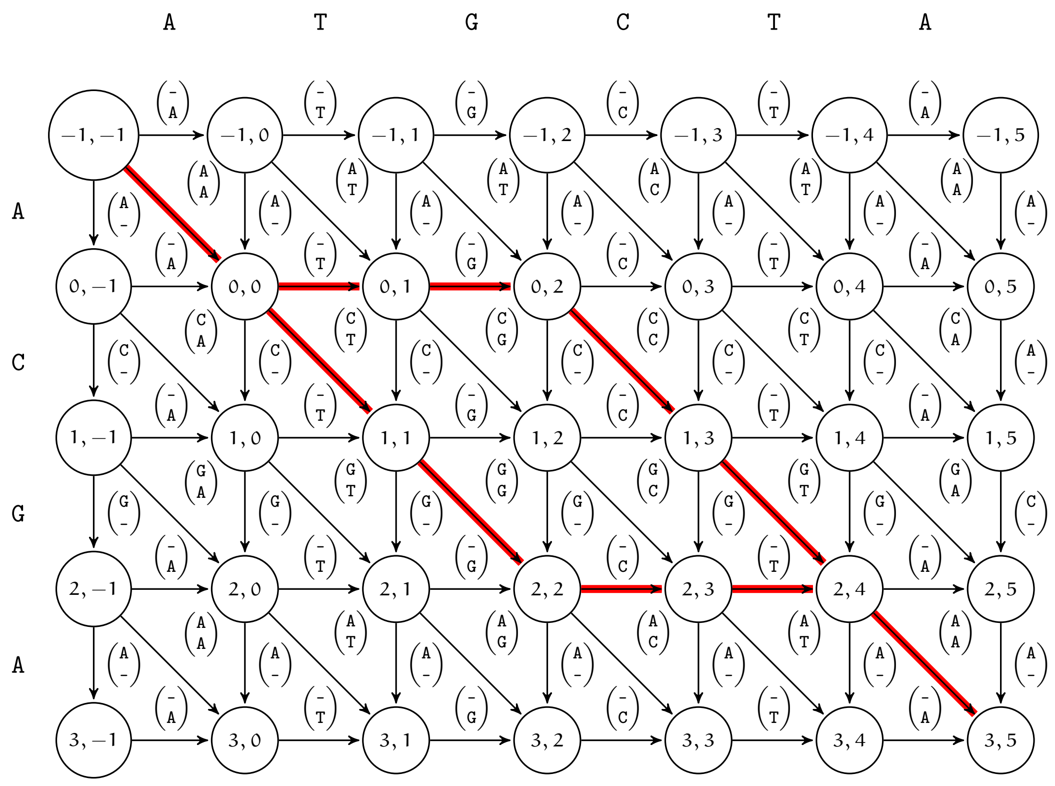


Figure 3: Alignment graph G(ACGA, ATGCTA), that does not represent the costs. Any path from vertex (-1,-1) to vertex (3,5) is an alignment of ACGA and ATGCTA. Highlighted paths represent optimal alignments.

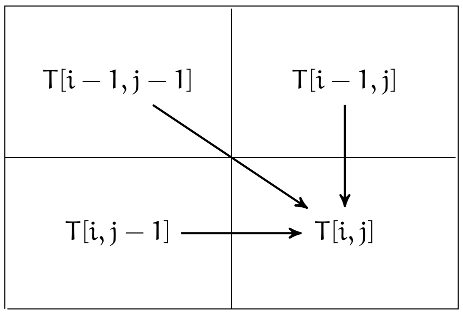


Figure 4: The value $T[i,j]$ depends only on the values in the three adjacent cells: $T[i-1,j], T[i-1,j-1]$, and $T[i,j-1]$ (when $i,j \geq 0$).

7.2 Optimal alignment

This section discusses the method to be followed in order to compute an optimal alignment between two strings. We use a straightforward dynamic programming technique, which stores the results of intermediary computations, thus avoiding redundant computations. Finding an alignment between two strings x and y is based on the edit distance computation. We explain how to carry out this computation. Then we describe how to find the associated optimal alignments.

Computing the edit distance

With two strings x and $y \in A^*$ of respective lengths m and n, let T be a $(m+1) \times (n+1)$ table defined by

$$T[i,j] = dist(x[0..i], y[0..j])$$

where $dist(x_i, y_j)$ is the minimal cost of a sequence of operations (insertion, deletion, substitution) that transforms string prefix x_i into y_j, for $i = -1, 0, \cdots, m-1$ and $j = -1, 0, \cdots, n-1$. Alternatively: $T[i,j]$ is the minimal cost of a path from $(1,1)$ to (i,j) in the alignment graph G.

In order to compute $T[i,j]$, we use the following recurrence:

Lemma 2 For $i = 0, 1, \cdots, m-1$ and $j = -1, 0, \cdots, n-1$, we have:

$$\begin{cases} T[-1,-1] = 0 \\ T[i,-1] = T[i-1,-1] + Del(x[i]) \\ T[-1,j] = T[-1,j-1] + Ins(y[i]) \\ T[i,j] = min \begin{cases} T[i-1,j-1] + Sub(x[i], y[j]) \\ T[i-1,j] + Del(x[i]) \\ T[i,j-1] + Ins(y[j]) \end{cases} \end{cases}$$

The value at position (i,j) in T , with $i,j \geq 0$, depends only on values at $(i-1,j-1), (i-1,j)$, and $(i,j-1)$ (see Figure 4). Figure 5 illustrates the computation.

The EDIT-DISTANCE algorithm, whose pseudo-code follows, computes the edit distance using table T. It populates T with the minimum edit costs for all prefixes of x and y and returns the value obtained in $T[m-1,n-1] = dist(x,y)$.

```
EDIT-DISTANCE($x, m, y, n$)
1  $T[-1,-1] = 0$
2  for $i = 0 \to m-1$
3      $T[i,-1] = T[i-1,-1] + Del(x[i])$
4  for $j = 0 \to n-1$
5      $T[-1,j] = T[-1,j-1] + Ins(y[i])$
6      for $i = 0 \to m-1$
7          $T[i,j] = min \begin{cases} T[i-1,j-1] + Sub(x[i], y[j]) \\ T[i-1,j] + Del(x[i]) \\ T[i,j-1] + Ins(y[j]) \end{cases}$
8  return $T[m-1,n-1]$
```

Reconstructing an optimal alignment

The EDIT-DISTANCE algorithm returns only the cost of transforming x into y. In order to obtain a sequence of operations that transforms x into y (or the corresponding alignment), we can start from position $[m-1, n-1]$ in the edit distance table T and go back up to position $[-1,-1]$. From a given position $[i,j]$, we visit, among adjacent positions $[i-1,j-1]$, $[i-1,j]$, and $[i,j-1]$, one of those positions whose cost was the basis for cost $T[i,j]$.

In the alignment graph $G(x,y)$, the **active edges** are the edges that are considered during the computation of an optimal alignment. Using the example of Figure 1 and Figure 2, the EDIT-DISTANCE algorithm computes the table T shown in Figure 6. The associated alignment graph is given in Figure 3; Figure 7 shows the active edges subgraph, which is derived from T. Formally, we say that an edge $((i',j'),(i,j))$ with label (a,b) is active when

$$\begin{cases} T[i,j] = T[i',j'] + Sub(a,b) \text{ if } i-i' = j-j' = 1, \\ T[i,j] = T[i',j'] + Del(a,b) \text{ if } i-i' = 1 \text{ and } j = j', \\ T[i,j] = T[i',j'] + Ins(a,b) \text{ if } i = i' \text{ and } j-j' = 1 \end{cases}$$

with $i, i' \in \{-1, 0, \dots, m-1\}, j, j' \in \{-1, 0, \dots, n-1\}, a, b \in A$.

(...)

The RECONSTRUCT-SOLUTION algorithm walks "upstream" along the active edges of the alignment graph, until it reaches vertex $(-1,-1)$. [3]

[3](Translator's note) The graph is implicitly defined in T.

(a)

Sub	A	C	D	E	G	K	L	P	Q	R	W	Y
A	0	3	3	3	3	3	3	3	3	3	3	3
C	3	0	3	3	3	3	3	3	3	3	3	3
D	3	3	0	3	3	3	3	3	3	3	3	3
E	3	3	3	0	3	3	3	3	3	3	3	3
G	3	3	3	3	0	3	3	3	3	3	3	3
K	3	3	3	3	3	0	3	3	3	3	3	3
L	3	3	3	3	3	3	0	3	3	3	3	3
P	3	3	3	3	3	3	3	0	3	3	3	3
Q	3	3	3	3	3	3	3	3	0	3	3	3
R	3	3	3	3	3	3	3	3	3	0	3	3
W	3	3	3	3	3	3	3	3	3	3	0	3
Y	3	3	3	3	3	3	3	3	3	3	3	0

	Del	Ins
A	1	1
C	1	1
D	1	1
E	1	1
G	1	1
K	1	1
L	1	1
P	1	1
Q	1	1
R	1	1
W	1	1
Y	1	1

(b)

T	j	−1	0	1	2	3	4	5	6	7	8	9	10	11
i		y[j]	E	R	D	A	W	C	Q	P	G	K	W	Y
−1	x[i]	$\boxed{0}$	1	2	3	4	5	6	7	8	9	10	11	12
0	E	1	$\boxed{0}$	$\boxed{1}$	$\boxed{2}$	3	4	5	6	7	8	9	10	11
1	A	2	1	2	3	$\boxed{2}$	3	4	5	6	7	8	9	10
2	W	3	2	3	4	3	$\boxed{2}$	3	4	5	6	7	8	9
3	A	4	3	4	5	4	$\boxed{3}$	4	5	6	7	8	9	10
4	C	5	4	5	6	5	4	$\boxed{3}$	4	5	6	7	8	9
5	Q	6	5	6	7	6	5	4	$\boxed{3}$	$\boxed{4}$	5	6	7	8
6	G	7	6	7	8	7	6	5	4	5	$\boxed{4}$	5	6	7
7	K	8	7	8	9	8	7	6	5	6	5	$\boxed{4}$	$\boxed{5}$	$\boxed{6}$
8	L	9	8	9	10	9	8	7	6	5	4	$\boxed{5}$	$\boxed{6}$	$\boxed{7}$

(c)

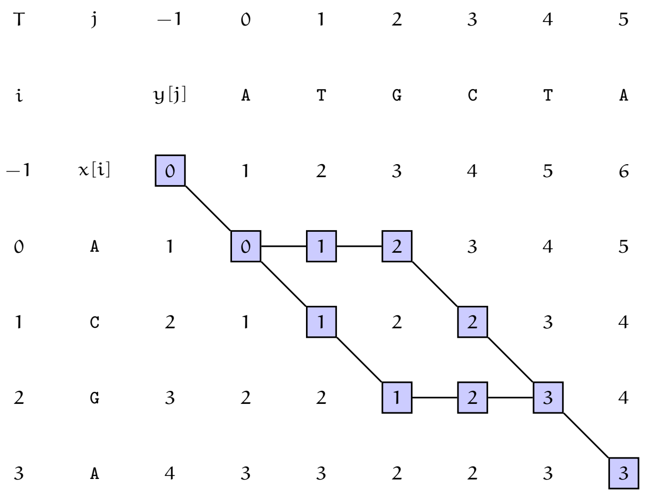
$$\begin{pmatrix} \mathtt{E} & \mathtt{-} & \mathtt{-} & \mathtt{A} & \mathtt{W} & \mathtt{A} & \mathtt{C} & \mathtt{Q} & \mathtt{-} & \mathtt{G} & \mathtt{K} & \mathtt{-} & \mathtt{-} & \mathtt{L} \\ \mathtt{E} & \mathtt{R} & \mathtt{D} & \mathtt{A} & \mathtt{W} & \mathtt{-} & \mathtt{C} & \mathtt{Q} & \mathtt{P} & \mathtt{G} & \mathtt{K} & \mathtt{W} & \mathtt{Y} & \mathtt{-} \end{pmatrix}$$

$$\begin{pmatrix} \mathtt{E} & \mathtt{-} & \mathtt{-} & \mathtt{A} & \mathtt{W} & \mathtt{A} & \mathtt{C} & \mathtt{Q} & \mathtt{-} & \mathtt{G} & \mathtt{K} & \mathtt{-} & \mathtt{L} & \mathtt{-} \\ \mathtt{E} & \mathtt{R} & \mathtt{D} & \mathtt{A} & \mathtt{W} & \mathtt{-} & \mathtt{C} & \mathtt{Q} & \mathtt{P} & \mathtt{G} & \mathtt{K} & \mathtt{W} & \mathtt{-} & \mathtt{Y} \end{pmatrix}$$

$$\begin{pmatrix} \mathtt{E} & \mathtt{-} & \mathtt{-} & \mathtt{A} & \mathtt{W} & \mathtt{A} & \mathtt{C} & \mathtt{Q} & \mathtt{-} & \mathtt{G} & \mathtt{K} & \mathtt{L} & \mathtt{-} & \mathtt{-} \\ \mathtt{E} & \mathtt{R} & \mathtt{D} & \mathtt{A} & \mathtt{W} & \mathtt{-} & \mathtt{C} & \mathtt{Q} & \mathtt{P} & \mathtt{G} & \mathtt{K} & \mathtt{-} & \mathtt{W} & \mathtt{Y} \end{pmatrix}$$

Figure 5: Computing the edit distance between strings EAWACQGKL and ERDAWCQPGKWY, with the corresponding alignments. **(a)** Alignment matrix: costs of edit operations, with $\mathrm{Sub}(a, b) = 3$ for $a \neq b$ et $\mathrm{Del}(a) = \mathrm{Ins}(a) = 1$. **(b)** Table T, as computed by the EDIT-DISTANCE algorithm. We obtain $\mathrm{EditDist}(\mathtt{EAWACQGKL}, \mathtt{ERDAWCQPGKWY}) = T[8, 11] = 7$. The table also features the paths of minimal cost between positions $[-1, -1]$ and $[8, 11]$. **(c)** The three associated optimal alignments.

(a)



T	j	−1	0	1	2	3	4	5
i		y[j]	A	T	G	C	T	A
−1	x[i]	$\boxed{0}$	1	2	3	4	5	6
0	A	1	$\boxed{0}$	$\boxed{1}$	$\boxed{2}$	3	4	5
1	C	2	1	$\boxed{1}$	2	$\boxed{2}$	3	4
2	G	3	2	2	$\boxed{1}$	$\boxed{2}$	$\boxed{3}$	4
3	A	4	3	3	2	2	3	$\boxed{3}$

(b)

$$\begin{pmatrix} \mathtt{A} & \mathtt{-} & \mathtt{-} & \mathtt{C} & \mathtt{G} & \mathtt{A} \\ \mathtt{A} & \mathtt{T} & \mathtt{G} & \mathtt{C} & \mathtt{T} & \mathtt{A} \end{pmatrix} \qquad \begin{pmatrix} \mathtt{A} & \mathtt{C} & \mathtt{G} & \mathtt{-} & \mathtt{-} & \mathtt{A} \\ \mathtt{A} & \mathtt{T} & \mathtt{G} & \mathtt{C} & \mathtt{T} & \mathtt{A} \end{pmatrix}$$

Figure 6: Table T, as obtained by the EDIT-DISTANCE algorithm. The edit distance EditDist(ACGA, ATGCTA) = T[3, 5] = 3. Paths of minimal cost transformations between positions [−1, −1] and [3, 5] are also represented.

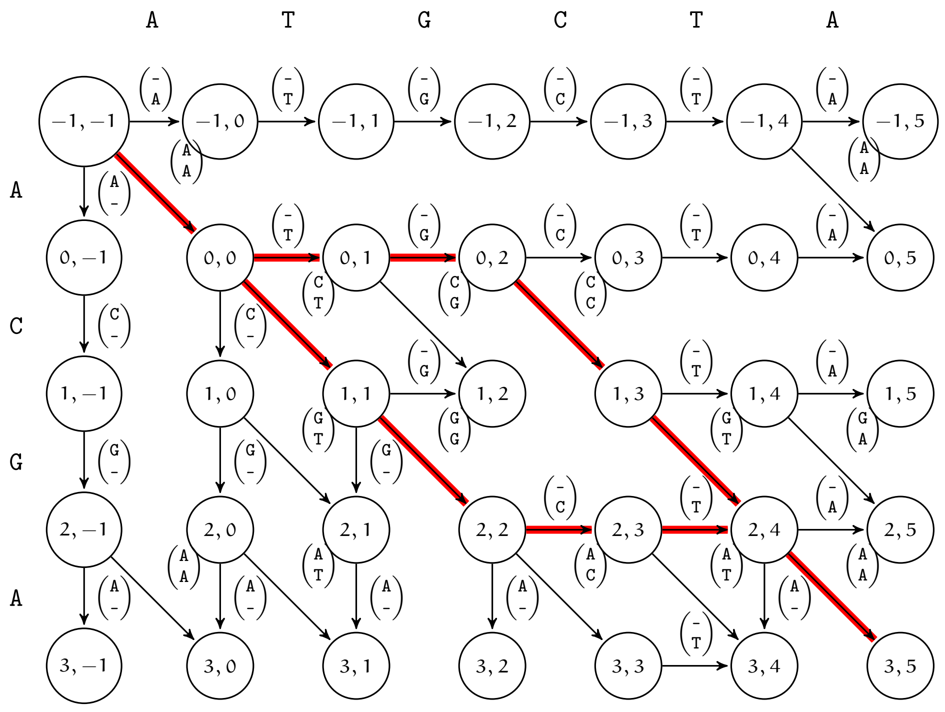


Figure 7: Active edges in the alignment graph of Figure 3. Paths highlighted in red connect vertices (−1, −1) et (3, 5); they correspond to an minimal edit distance between the two strings. Edges represent transformations needed to obtain the output string.

```
RECONSTRUCT-SOLUTION($x, m, y, n$)
$z \leftarrow (\epsilon, \epsilon)$
$(i, j) \leftarrow (m-1, n-1)$
while $i \neq -1$ and $j \neq -1$
    if $T[i, j] = T[i-1, j-1] + Sub[x[i], y[j]]$
    // Prepend $(x[i], y[j])$ to the sequence of alignments
        $z \leftarrow (x[i], y[j]) \cdot z$ // '$\cdot$' is the concatenation operator
        $(i, j) \leftarrow (i-1, j-1)$
    elseif $T[i, j] = T[i-1, j] + Del[x[i]$
    // Prepend $(x[i], \epsilon)$ to the sequence of alignments
        $z \leftarrow (x[i], \epsilon) \cdot z$
        $i \leftarrow i - 1$
    else // Prepend $(\epsilon, y[j])$ to the sequence of alignments
        $z \leftarrow (\epsilon, y[j]) \cdot z$
        $j \leftarrow j - 1$
while $i \neq -1$
    $z \leftarrow (x[i], \epsilon) \cdot z$
    $i \leftarrow i - 1$
while $j \neq -1$
    $z \leftarrow (\epsilon, y[j]) \cdot z$
    $j \leftarrow j - 1$
return $z$
```