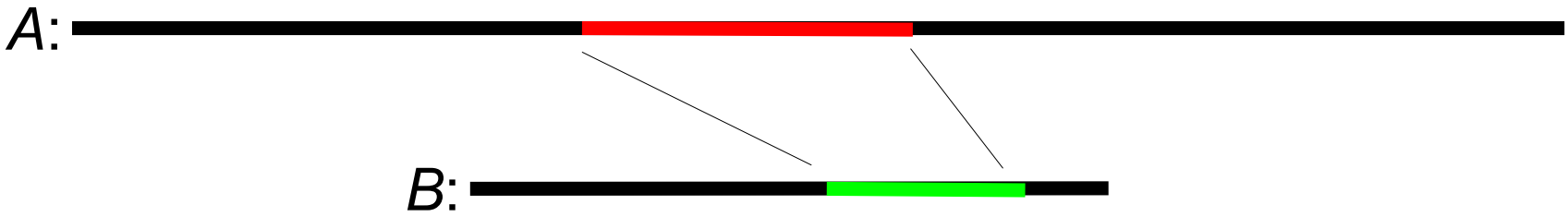
Local alignment

Local alignment

Objective:

Given two sequences *A* and *B*, a score matrix and a gap cost, find the pair of segments of *A* and *B* which has maximum alignment cost, i.e. maximum similarity.



Motivation: "... one must be alert to regions of similarity even when they occur embedded in an overall background of dissimilarity." (Doolittle)

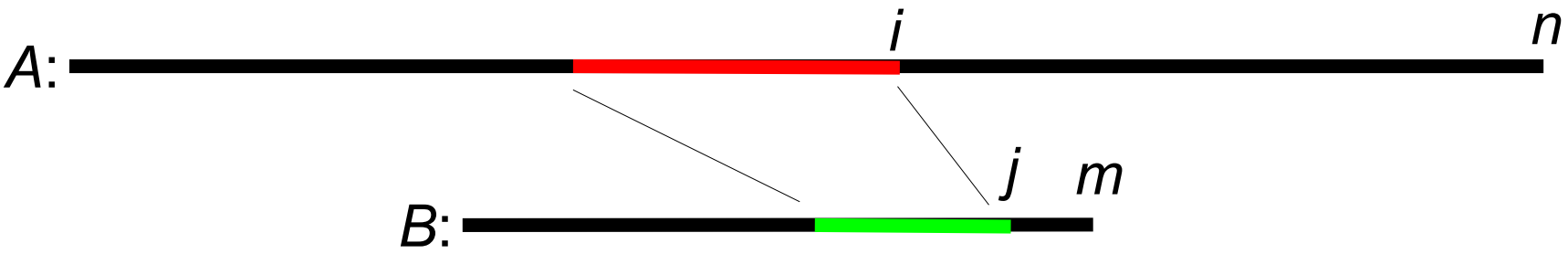
Seems more difficult than global alignment.

We have to align all pairs of segments?

Finding an optimal local alignment

Objective:

Given two sequences *A* and *B*, a score matrix and a gap cost, find the pair of segments of *A* and *B* which has maximum alignment cost, i.e. maximum similarity.



Definition: Let LocalCost(*i,j*) be the optimal cost of an alignment of segments ending at position *i* (of *A)* and *j* (of *B)*.

Observations: (1) LocalCost(0,0) = 0 (the cost of aligning two empty strings), and (2) the opt. score of an local alignment is:

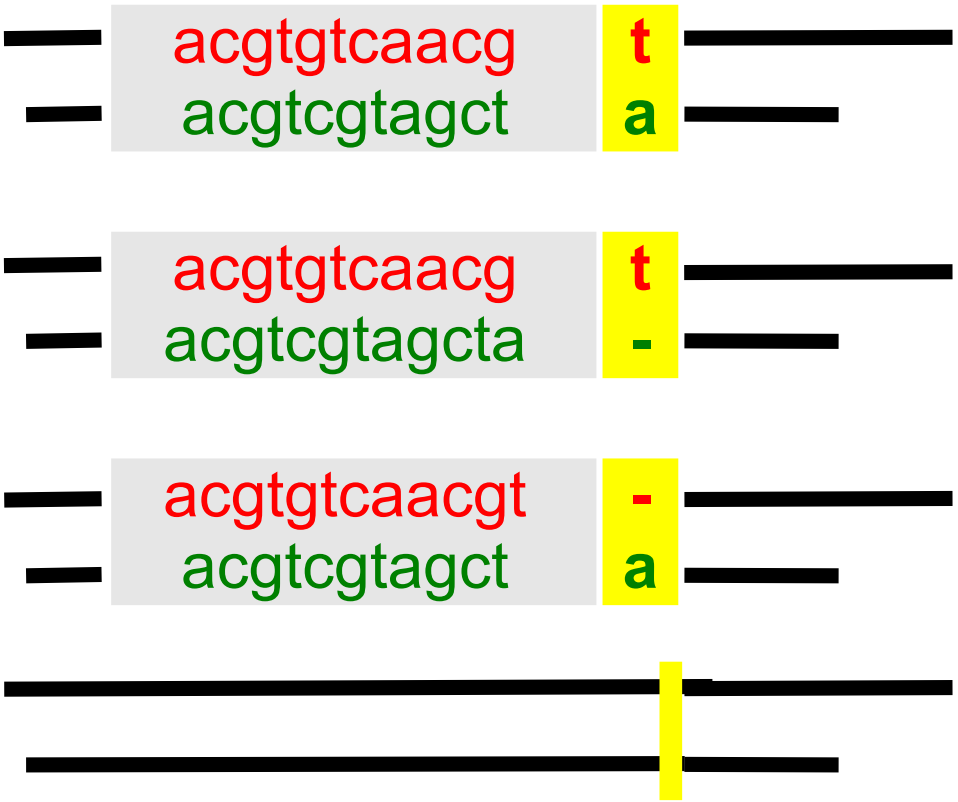
max LocalCost(*i*, *j*) for *i*=0,...,*n* and *j*=0,...,*m*

Computing LocalCost(i,j)

Observation: The maximum cost of an alignment of segments ending at position *i* (of *A)* and *j* (of *B)* is either:

(1) the cost of extending the best alignment of previous segments, or

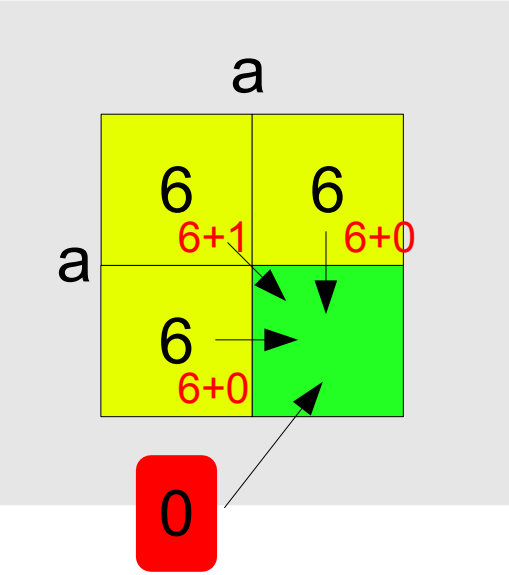
(2) the cost of two new/empty segments ...



Computing LocalCost(i,j)

Local alignment

$$\text{LocalCost}(i, j) = \max \begin{cases} \text{LocalCost}(i-1, j-1) + \text{subcost}(A[i], B[j]) \\ \text{LocalCost}(i-1, j) + \text{gapcost} \\ \text{LocalCost}(i, j-1) + \text{gapcost} \\ 0 \end{cases}$$



Fill out table (e.g. row by row), return maximum entry ....

Global alignment

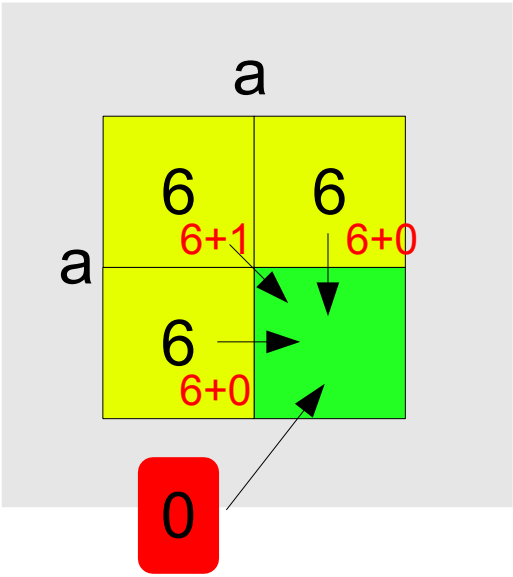
$$\text{Cost}(i, j) = \max \begin{cases} \text{Cost}(i-1, j-1) + \text{subcost}(A[i], B[j]) \\ \text{Cost}(i-1, j) + \text{gapcost} \\ \text{Cost}(i, j-1) + \text{gapcost} \\ 0 \text{ if } i=0 \text{ and } j=0 \end{cases}$$

Fill out table (e.g. row by row), return entry (n,m)....

Computing LocalCost(i,j)

Local alignment

$$\text{LocalCost}(i, j) = \max \begin{cases} \text{LocalCost}(i-1, j-1) + \text{subcost}(A[i], B[j]) \\ \text{LocalCost}(i-1, j) + \text{gapcost} \\ \text{LocalCost}(i, j-1) + \text{gapcost} \\ 0 \end{cases}$$



Fill out table (e.g. row by row), return maximum entry ....

Note: If the score function is such that the cost of an alignment cannot be negative, then an local alignment is the same as a global alignment

ment

$$\text{Cost}(i, j) = \max \begin{cases} \text{Cost}(i-1, j-1) + \text{subcost}(A[i], B[j]) \\ \text{Cost}(i-1, j) + \text{gapcost} \\ \text{Cost}(i, j-1) + \text{gapcost} \\ 0 \text{ if } i=0 \text{ and } j=0 \end{cases}$$

Score matrix:

	a	c	g	t
a	1	0	0	0
c	0	1	0	0
g	0	0	1	0
t	0	0	0	1

gapcost: 0

Fill out table (e.g. row by row), return entry (*n*,*m*)....

Finding an optimal local alignment

Local alignment

$$\text{LocalCost}(i, j) = \max \begin{cases} \text{LocalCost}(i-1, j-1) + \text{subcost}(A[i\,], B[j\,]) \\ \text{LocalCost}(i-1, j) + \text{gapcost} \\ \text{LocalCost}(i, j-1) + \text{gapcost} \\ 0 \end{cases}$$

Fill out table (e.g. row by row), return maximum entry ....

Backtracking

Let (i, j) = arg max LocalCost(i, j) for i=0,...,n and j=0,...,m

Backtrack in the dynamic programming matrix from entry (i, j) until you reach entry (h, k) where LocalCost(h, k)=0 (you will find such a (h, k) since LocalCost(0,0)=0). By construction we then have:

LocalCost(i, j) is the cost of an optimal alignment of $A[h+1 .. i]$ and $B[k+1 .. j]$ and the backtracking gives such an alignment.

Finding an optimal local alignment

Local alignment

$$\text{LocalCost}(i, j) = \max \begin{cases} \text{LocalCost}(i-1, j-1) + \text{subcost}(A[i\,], B[j\,]) \\ \text{LocalCost}(i-1, j) + \text{gapcost} \\ \text{LocalCost}(i, j-1) + \text{gapcost} \\ 0 \end{cases}$$

Fill out table (e.g. row by row), return maximum entry ....

Backtracking

Time and space is O(nm)

Let (i, j) = arg max LocalCost(i, j) for i=0,...,n and j=0,...,m

Backtrack in the dynamic programming matrix from entry (i, j) until you reach entry (h, k) where LocalCost(h, k)=0 (you will find such a (h, k) since LocalCost(0,0)=0). By construction we then have:

LocalCost(i, j) is the cost of an optimal alignment of $A[h+1 .. i]$ and $B[k+1 .. j]$ and the backtracking gives such an alignment.