Variations on global alignment

Longest common subsequence, end-space free alignment, approximate occurence of P in T

Longest common subsequence

Difference between substring and subsequence (we've already mentioned this on the previous lecture):

String: anavolimilovana

Substring: voli (this is also a subsequence)

Subsequence: aaaa

Not subsequence: lili (this is not relative order of characters)

Longest common subsequence

A subsequence of S is a sequence of characters from S in the same relative order as in S, but not necessarily consecutive.

Subsequence doesn’t have to be contiguous, substring has. Subsequence of S has same **relative order** of characters.

Longest common subsequence

Let's formalize:

Def: In a string S a subsequence is defined as a subset of characters of S arranged in their original relative order. More formally, a subsequence of string S of length n is specified by the list of indices $i_1 < i_2 < i_3 < ... < i_k$ for some $k \leqslant n$. The subsequence is specified by this list of indices in the string $S(i_1)S(i_2)...S(i_k)$.

Longest common subsequence

Longest common subsequence problem: for strings S1 and S2 find longest common subsequence.

AT-C-TGAT

-TGCAT-A-

Longest common subsequence

Solution: With a scoring scheme that scores a one for each match and a zero for each mismatch or space, the **matched characters in alignment** of maximum value form a longest common subsequence.

Also, consecutive gaps (insertion-then-deletion or deletion-then-insertion) are preferred to mismatch.

Note: we can (again) invert scoring matrix (and look for minimum function value) until we come to local alignment, but then we really need to stop.

Longest common subsequence

Consecutive gaps (insertion-then-deletion or deletion-then-insertion) are preferred to mismatch.

1. We do this either in algorithm, where if S1(i)==S2(j) we check minimum for all the edit operations, otherwise we can only for insertion/deletion.

   ...

   if $x_i = y_j$:

   D(i,j)=max(D(i-1,j), D(i, j-1), D(i-1, j-1)+1)

   else:

   D(i,j)=max(D(i-1,j), D(i, j-1))

2. By adapting the scoring matrix, by penalizing mismatch the most:

	A	C	G	T	-
A	1	-1	-1	-1	0
C	-1	1	-1	-1	0
G	-1	-1	1	-1	0
T	-1	-1	-1	1	0
-	0	0	0	0	

$s(a, b)$

Longest common subsequence

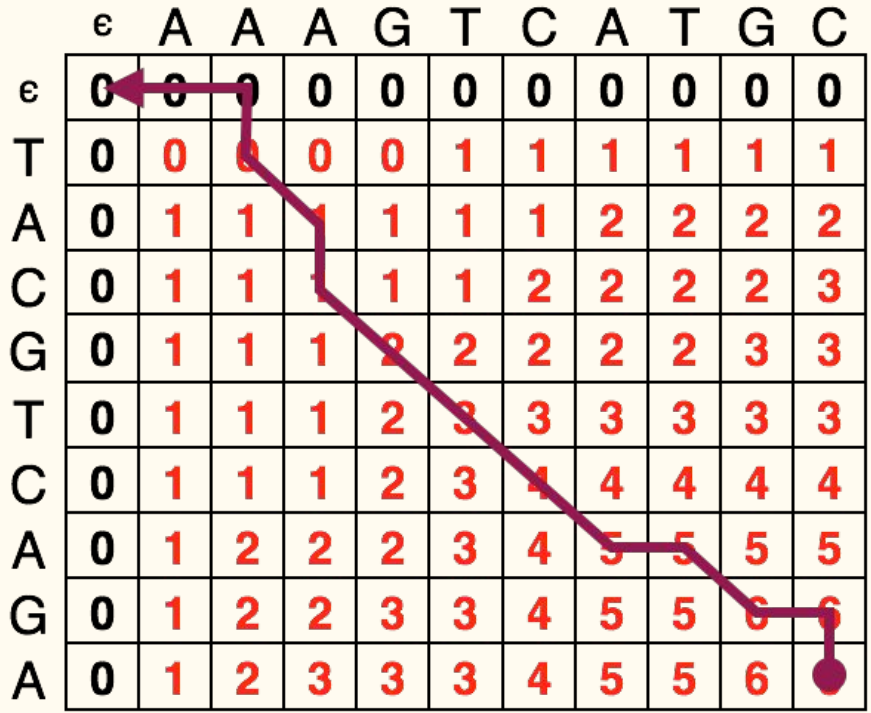
	ϵ	A	A	A	G	T	C	A	T	G	C
ϵ	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	1	1	1	1	1	1
A	0	1	1	1	1	1	1	2	2	2	2
C	0	1	1	1	1	1	2	2	2	2	3
G	0	1	1	1	2	2	2	2	2	3	3
T	0	1	1	1	2	3	3	3	3	3	3
C	0	1	1	1	2	3	4	4	4	4	4
A	0	1	2	2	2	3	4	5	5	5	5
G	0	1	2	2	3	3	4	5	5	6	6
A	0	1	2	3	3	3	4	5	5	6	6

$s(a, b)$

	A	C	G	T	-
A	1	-1	-1	-1	0
C	-1	1	-1	-1	0
G	-1	-1	1	-1	0
T	-1	-1	-1	1	0
-	0	0	0	0	

6 is LCS length

Longest common subsequence



	ϵ	A	A	A	G	T	C	A	T	G	C
ϵ	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	1	1	1	1	1	1
A	0	1	1	1	1	1	1	2	2	2	2
C	0	1	1	1	1	1	2	2	2	2	3
G	0	1	1	1	2	2	2	2	2	3	3
T	0	1	1	1	2	3	3	3	3	3	3
C	0	1	1	1	2	3	4	4	4	4	4
A	0	1	2	2	2	3	4	5	5	5	5
G	0	1	2	2	3	3	4	5	5	6	6
A	0	1	2	3	3	3	4	5	5	6	7

Alignment is:

__TACGTCA_G_A

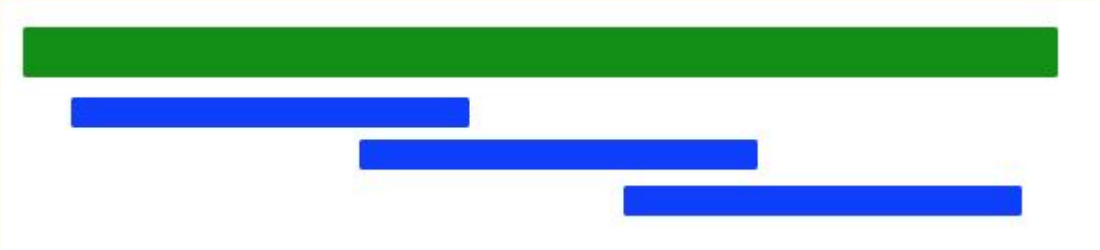
AA_A_GTCATGC_

Diagonal move gives LCS:

$AGTCAG$

End space-free variant

Often, we see have strings which come from the same parts of the genome, but which are not completely overlapping*



We try to reconstruct the genome and get overlap between those reads. What happens is that we get overlap on one end of the reads and empty spaces on the end of the reads.

*In actual genomic data processing, we usually want to remove reads which completely overlap (duplicates) because they can skew our statistics later, and contain redundant information

End space-free variant

Example alignment:

_ _ c a c _ d b d

l t c a b b d b _

What we want sometimes, it **not to** penalize empty spaces at the end of the strings because they reflect true relationship between the reads.

End space-free variant

We just change the base, conditions, everything else stays the same in recurrence:

$$V(0,j)=0$$

$$V(i,0)=0,\ \forall\ i, j$$

This will allow us not having to end the traceback in cell (0,0) (takes care of left side empty spaces).

End space-free variant

But what about right side end-free spaces?
Instead of starting traceback in lower right corner (n,m), we start from cell in the in row *n* or column *m* which has maximum value.

If we start from cell in row *n*, it means that all characters of string S1 (vertical column in the table) contributes to the alignment, but not all of S2 do (we have free end-spaces). If we start from cell in column *m* than it's the opposite.

Approximate occurrences of P in T

Problem: can we find out if there is string *similar* to P contained in string T.

Real life example: we've sequenced a water from the pond. We're interested if it contains a certain bacteria. We need to compare our sequences (reads) to bacteria reference. Of course, our reads to not contain entire bacteria sequence, so we're searching for similarity with substring of reference. And we are interested if degree of similarity is sufficiently high.

Approximate occurrences of P in T

What do we need to to:

- Do inexact matching of our sequence with reference.
- Allow end-space free variant but only for read (not reference).
- Compute similarity score for alignment of reads and part of reference where it aligned to and check if it's above certain threshold.
- (Do the traceback if we're interested how does P align to T).

Approximate occurrences of P in T

Def: Given a parameter $\boldsymbol{\sigma}$, a substring T' of T is said to be an *approximate occurence* of P in T if and only if the optimal alignment of P to T' has value at least $\boldsymbol{\sigma}$.

Solution: There is and approximate occurence of P in T ending at position *j* of T if and only if $V(n,j) \geq \boldsymbol{\sigma}$. Moreover, T[k..j] is an approximate occurrence of P in T if and only if $V(n, j) \geq \boldsymbol{\sigma}$ and there is a path of backpointers from cell (n,j) to cell (0,k).

Approximate occurrences of P in T

Set initial conditions*:

$$V(0,j)=0,\ \forall j$$

Then, we go through all columns of row n, and for every cell with value greater than $\boldsymbol{\sigma}$ we can say we have one occurence of P in T ending there.

***Note:** This way we allow space-end free variant for read. If we set initial conditions as stated, that means T string is along the horizontal axis.

Approximate occurrences of P in T

Backtrack: if we need to retrieve every alignment, we need to do traceback starting from every (n,j), where $V(n,j) \geq \boldsymbol{\sigma}$ to row 0.

When we do traceback from certain position, if there are multiple paths, we break ties by choosing a vertical pointer over a diagonal one and diagonal over horizontal one.

Approximate occurrences of P in T

In case where we're interested in **best** occurence, we start from maximum value in row n.

		ε	A	A	C	C	C	T	A	T	G	T	C	A	T	G	C	C	T	T	G	G	A
	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	T	1	1	1	1	1	1	0	1	0	1	1	1	1	0	1	1	1	0	0	1	1	1
	A	2	1	1	2	2	2	1	0	1	2	2	2	1	1	2	2	2	1	1	2	2	2
	C	3	2	2	1	2	2	2	1	1	2	2	1	2	2	2	1	2	2	2	2	2	2
	G	4	3	3	2	2	3	3	2	2	1	2	2	2	3	2	2	2	3	3	2	2	3
P	T	5	4	4	3	3	3	3	3	2	2	1	2	3	2	3	3	3	2	3	3	3	3
	C	6	5	5	4	3	3	4	4	3	3	2	1	2	3	3	3	3	3	3	4	4	4
	A	7	6	5	5	4	4	4	4	4	4	3	2	1	2	3	4	4	4	4	4	5	4
	G	8	7	6	6	5	5	5	5	5	4	4	3	2	2	2	3	4	5	5	4	4	5
	C	9	8	7	6	6	5	6	6	6	5	5	4	3	3	3	2	3	4	5	5	5	5

***Note:** In this case we've used edit-distance alike scoring schema, so we're looking for minimum values in matrix.

Variations on global alignment

Since all of these algorithms are variation on existing global alignment, time/space complexity is same as for global alignment - O(nm).

Local alignment

Smith-Waterman alignment

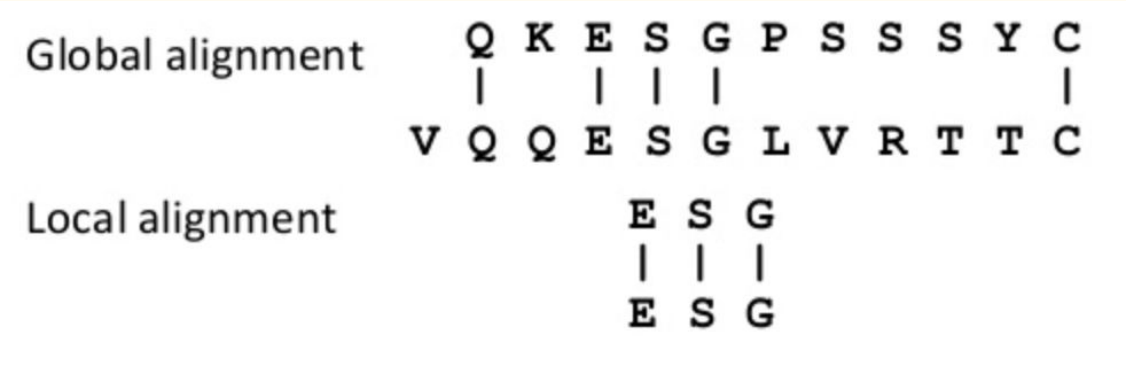
Local alignment

Sometimes, we want to detect only areas of high similarity, although two strings (genomes) might not be similar in their entirety.

- Comparing protein domains across species for detection of conserved characteristics
- Tracing evolutionary characteristics of genes
- We are interested in high *homology* regions

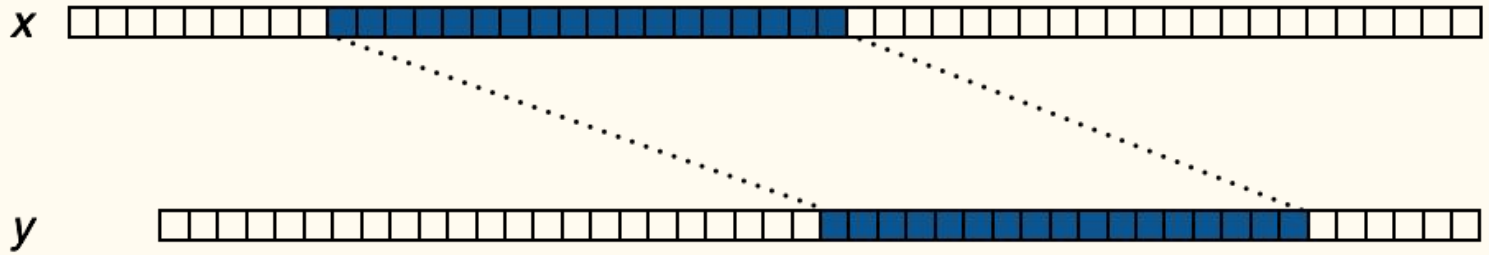
Local vs global alignment

- Global alignment does end-to-end string comparison.
- Local alignment: returns/emphasizes only areas of high similarity



Local alignment

In simple words: Given strings x and y, what is the optimal global alignment value of a *substring* of x to a *substring* of y. This is *local alignment*.



Assume global alignment scoring where: (a) similarities get > 0, (b) dissimilarities get < 0, (c) alignment of ϵ to any string has score 0.

Somehow we must weigh *all possible pairs* of substrings.

What is bound for # substring pairs, assuming $| x | = n$, $| y | = m$? $O(m^2n^2)$

Local alignment

Problem definition: Given two strings S1 and S2 find substrings α and β of S1 and S2, respectively, whose similarity (optimal global alignment value) is maximum over all pairs of substrings S1 and S2.

We define local alignment as **maximizing** objective function* (unlike edit distance which minimizes the objective function).

***Note:** in case of local alignment, this is strict requirement. While we could have implemented global alignment with both maximisation and minimisation of objective function (with proper scoring matrices applied for both of these cases) this is not possible for local alignment. **Local alignment must be implemented as maximization of objective function!**

Local alignment - scoring schema

- Matches contribute positively and mismatches and gaps contribute negatively. This way, we're more likely to find regions of high similarity.

$$s(a, b)$$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Local alignment - recurrence relation

Base conditions:

$$v(0,i)=0$$

$$v(0,j)=0,\ \forall i, j$$

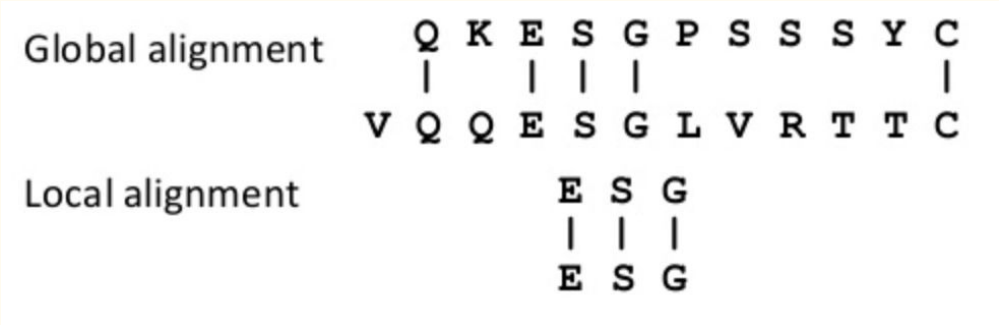
And recurrence relation:

$$V(i,j)=\max[0, v(i-1, j)+s(S1(i), \text{‘\_’}), v(i, j-1)+s(\text{‘\_’}, S2(j)), v(i-1, j-1)+s(S1(i), S2(j))]$$

Local alignment

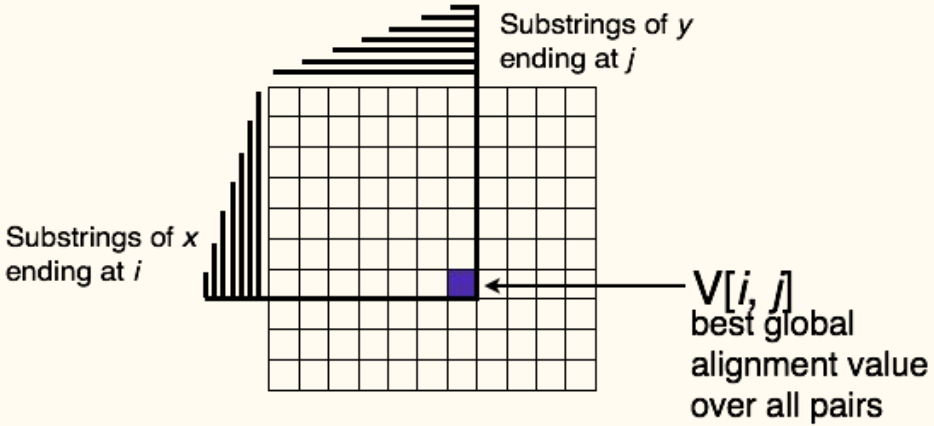
Differences compared to global alignment:

- Zero in recurrence relation: By adding zero we can “restart” the counter (score) - ignore any number of initial characters and emphasize regions of high similarity.
  This way when we’re doing traceback we don’t have to go all the way to the first row/column, and can ignore any number of initial characters.
- We also add zeros as base conditions: we can choose empty suffix.



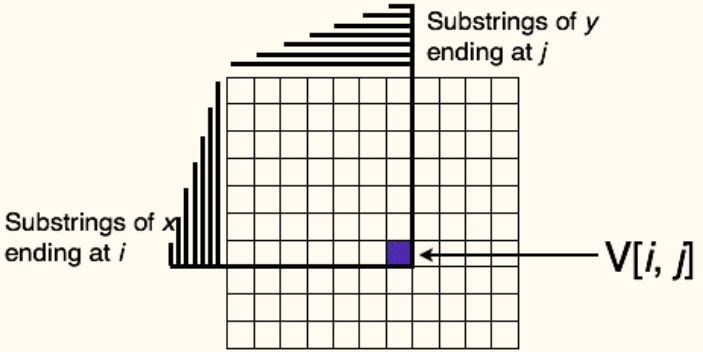
Local alignment - traceback

Let V[i, j] be the optimal global alignment value of a substring of x ending at i and a substring of y ending at j. The substrings may be empty.



The maximum V[i, j] over all i, j is the optimal score we're looking for. Do the traceback to any cell(i,j) which has value 0 to retrieve local alignment.

Local alignment (Smith-Waterman)



	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Let $V[0, j] = 0$, and let $V[i, 0] = 0$

Otherwise, let $V[i, j] = \max \begin{cases} V[i-1, j] + s(x[i-1], -) \\ V[i, j-1] + s(-, y[j-1]) \\ V[i-1, j-1] + s(x[i-1], y[j-1]) \\ 0 \end{cases}$

$s(a, b)$ assigns a score to a particular match, gap, or replacement

Local alignment: example

Why we initialize matrix with zeros: we can choose empty suffix.

Also, based on scoring schema, 0 is best score in recurrence relation for s('_', $S_{1/2}$(i))

		Y													
		ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
	ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0														
	G	0														
	T	0														
	A	0														
	T	0														
	G	0														
X	C	0														
	T	0														
	G	0														
	G	0														
	C	0														
	G	0														
	C	0														
	T	0														
	A	0														

$s(a, b)$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Local alignment: example

$$V[i,j] = \max \begin{cases} V[i-1,j] + s(x[i-1], -) \\ V[i,j-1] + s(-, y[j-1]) \\ V[i-1,j-1] + s(x[i-1], y[j-1]) \\ 0 \end{cases}$$

	ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
A	0	0	4	0	?										
T	0														
G	0														
C	0														
T	0														
G	0														
G	0														
C	0														
G	0														
C	0														
T	0														
A	0														

$s(a, b)$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

Local alignment: example

	ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
C	0	0	0	0	0	0	2	10	4	0	4	0	0	0	0
T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
G	0	0	0	0	0	0	2	0	2	8	4	4	0	0	0
C	0	0	0	0	0	0	0	4	0	2	10	4	0	0	0
G	0	0	0	0	0	0	2	0	6	2	4	12	6	0	0
C	0	0	0	0	0	0	0	4	0	2	4	6	8	2	0
T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

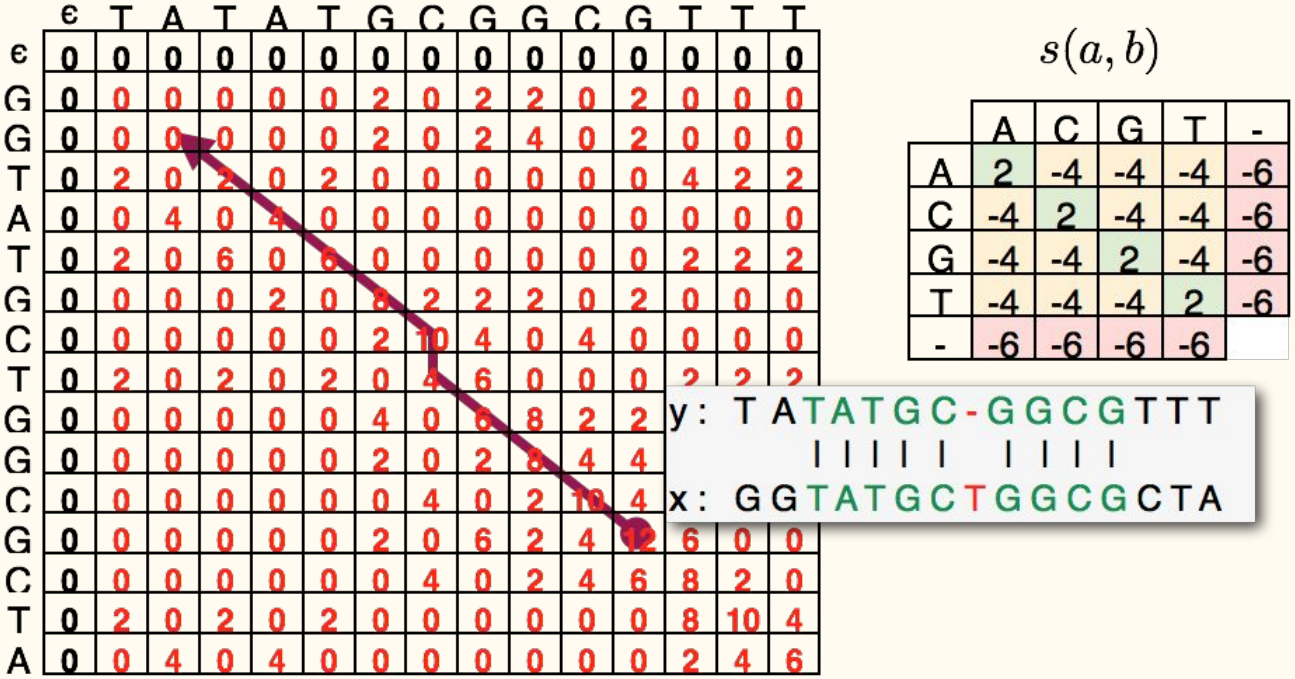
$s(a, b)$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

0's in essence allow peaks of similarity to rise above "background" of 0s.

Local alignment: example

Backtrace: (a) start from *maximal* cell in the matrix, (b) stop backtrace when we reach a cell with score = 0.



ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
C	0	0	0	0	0	2	0	4	0	4	0	0	0	0	0
T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
G	0	0	0	0	0	0	2	0	2	6	4	4	0	0	0
C	0	0	0	0	0	0	0	4	0	2	10	4	2	0	0
G	0	0	0	0	0	0	2	0	6	2	4	12	6	0	0
C	0	0	0	0	0	0	0	4	0	2	4	6	8	2	0
T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

$s(a, b)$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

```
y: T ATATGC-GGCGTTT
     |||||  |||| |
x: GGTATGCTGGCGCTA
```

Local alignment: example

What if we didn't have a positive "bonus" for matches?
All cells would = 0

	ϵ	T	A	T	A	T	G	C	G	G	C	G	T	T	T
ϵ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
C	0	0	0	0	0	0	2	10	4	0	4	0	0	0	0
T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
G	0	0	0	0	0	0	2	0	2	8	4	4	0	0	0
C	0	0	0	0	0	0	0	4	0	2	10	4	0	0	0
G	0	0	0	0	0	0	2	0	6	2	4	12	6	0	0
C	0	0	0	0	0	0	0	4	0	2	4	6	8	2	0
T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

$s(a, b)$

	A	C	G	T	-
A	2	-4	-4	-4	-6
C	-4	2	-4	-4	-6
G	-4	-4	2	-4	-6
T	-4	-4	-4	2	-6
-	-6	-6	-6	-6	

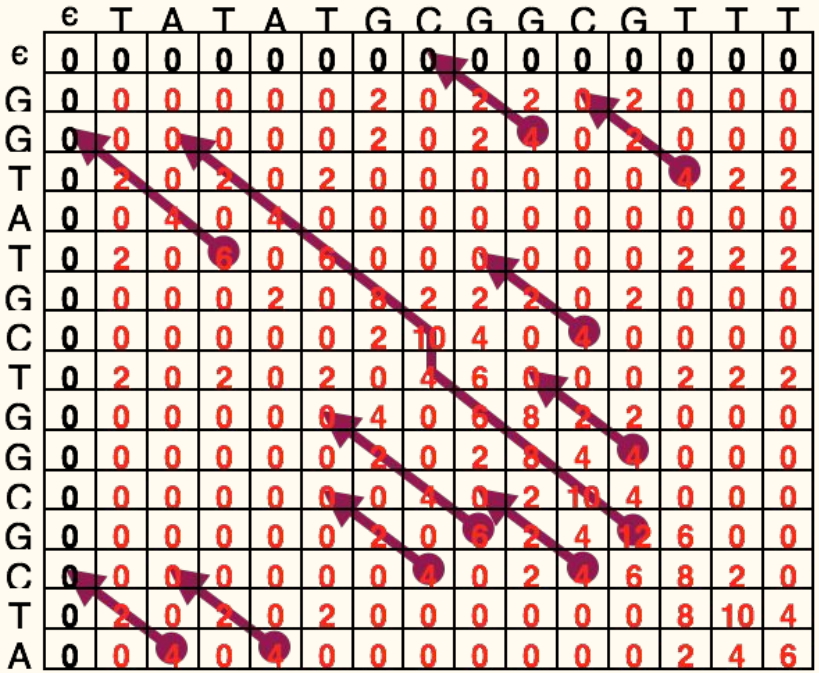
What if we didn't have negative "penalties" for edits?

In the recurrence relation, we would never use 0, and we would essentially have global alignment.

$$\max \begin{cases} V[i-1,j] + s(x[i-1], -) \\ V[i,j-1] + s(-, y[j-1]) \\ V[i-1,j-1] + s(x[i-1], y[j-1]) \\ 0 \end{cases}$$

Local alignment: more local alignments

We might be interested in the *best* local alignment, or in many *good-enough* local alignments.



Local alignment: time analysis

Filling out the table - O(nm) time and space.

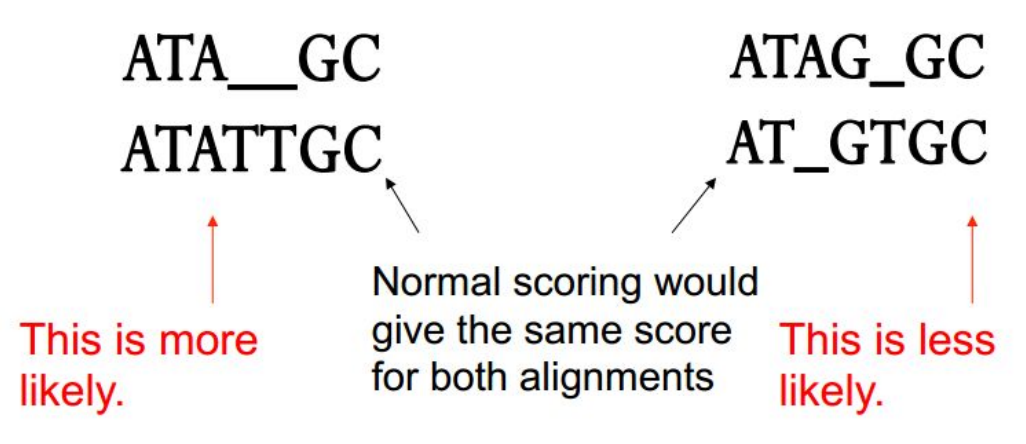
Gaps

Filling the blanks

Gaps

Def: A gap is any maximal, consecutive run of spaces in a single string of a given alignment.

In nature, a series of k indels often comes as a single event rather than a series of k single nucleotide events:



Gaps

Often is the case that instead of instead of single insertion/deletion, larger string is deleted (inserted).

TTACAGAGAGT

TTAC____AGT

Although gap is several characters long, it is usually the result of single mutational event. Therefore, our current model where we penalize each space independently might not be most adequate in this case.

Gap penalty models

Current model (up until now):

For a gap length of x, penalty is -σx (σ is penalty for insertion/deletion).

What we usually do for gap is have separate penalties for gap initiation and penalties for gap extension (each additional space in the gap).

Gap penalty models: constant

Constant: we can penalize each gap with the same weight W_g (for gap opening) independent of gap length. This means that we don't penalize gap extension(empty spaces: ($s(x,\_)=s(\_,x)=0$), just gap opening.

Then value of alignment (which we're trying to maximize) would be:

$$\sum_{i=1}^{l} s(S'_1(i), S'_2(j)) - kW_g,$$

Where k is the number of gaps.

Gap penalty models: affine

Affine: We score independently (but both with constant weight) *gap initiation* W_g and *gap extension* W_s (cost of extending the gap by one space).

Weight of the gap is then:

$$W_g + qW_s,$$

where q is the number of spaces.

We're then trying to maximize the following alignment problem:

$$\sum_{i=1}^{l} s(S'_1(i), S'_2(j)) - W_g(\#gaps) - W_s(\#spaces)$$

Penalty weight example - FASTA algorithm: $W_g=10$, $W_s=2$. Constant gap model is affine with $W_s=0$.

Gap penalty models: affine

$$V(i,j)=\max[E(i,j), F(i,j), G(i,j)]$$

$G(i,j)= V(i-1,j-1)+\boldsymbol{\delta}(i,j)$
$E(i,j) = \max[V(i,j-1)-W_g, E(i,j-1)]-W_s$ (opening or extending gap in first string)
$F(i,j) = \max[V(i-1,j)-W_{g,} F(i-1,j)]-W_s$ (opening or extending gap in second string)

What's different now is that when we have empty space (indel) we check what is the situation - are we extending or opening the gap.

Initial conditions are:

$$V(i,0)=-W_g -iW_s$$

$$V(0,j)=-W_g -jW_s$$

Gap penalty models: affine - complexity

Time complexity - As before O(nm), as we only compute four matrices instead of one.

Space complexity: there's a need to save four matrices (for F, G, H and V respectively) during the computation. Hence, O(nm) space is needed, for the trivial implementation.

Gap penalty models: convex and arbitrary

Convex model: Each additional space in a gap contributes less to the gap weight than the preceding space. This is a *convex* function (negative second derivative). One of the examples would be $W_g + \log_e q$, where q is the length of the gap.

Arbitrary: Gap weight is an arbitrary function $w(q)$ of it's length q.

Gap models: time complexity

Constant:	$O(nm)$
Affine:	$O(nm)$
Convex:	$O(mn \log(m+n))$
Arbitrary:	$O(nm^2+n^2m)$