

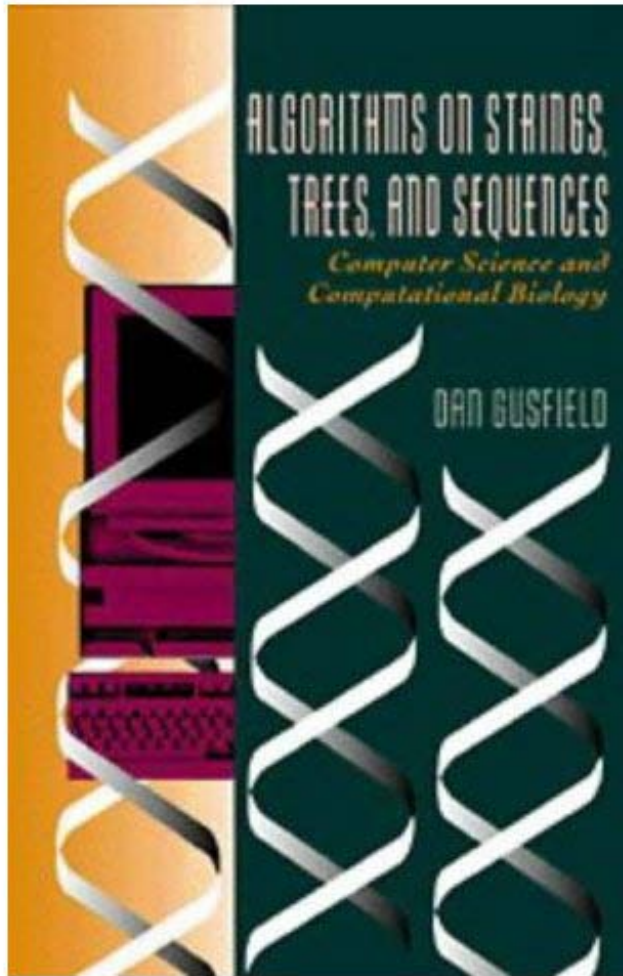
# Lecture 2

# Pairwise Alignment

Some slides from Serafim Batzoglou, Stanford



# Main source



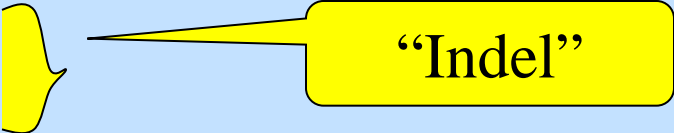
# Sequence Alignment

- Learn functionality or structure without performing experiments,
- Use known structure /function information

- The problem: Comparing two sequences while allowing certain mismatches between them.
- Motivation:
  - Comparing DNA seqs and proteins from databases,
    - Comparing two or more sequences for similarity
    - Searching databases for related sequences and subsequences
    - Finding informative elements in protein and DNA sequences
  - Building block in Fragment Assembly: Reconstructing long DNA sequences from overlapping subsequences
  - Determining physical and genetic maps from probe data,
  - Exploring frequently occurring patterns of nucleotides



# Similarity and Difference

- **Resemblance** of DNA sequences of different organisms explained by common ancestral origin
  - **Differences** are explained by mutation:
    - Insertion
    - Deletion
    - Substitution
- A yellow bracket groups the terms 'Insertion' and 'Deletion' from the list above. A yellow callout box with a black border and a pointer pointing to the bracket contains the text "Indel".
- **Distance** between two sequences is the minimal (weighted) sum of mutations transforming one into the other.
  - **Similarity** of two sequences is the maximum (weighted) sum of resemblances between them.



# Nomenclature

- **Computer Science:**
  - String, word
  - Substring (contiguous)
  - Subsequence
  - Exact matching
  - inexact matching
- **Biology:**
  - Sequence
  - Subsequence
  - .
  - N/a
  - N/a
  - Alignment

non contiguous segment of a sequence

We shall use the **biology** nomenclature



# Simplest model: Edit Distance

The **edit distance** between two sequences is the min no. of edit operations (insertions, deletions and substitutions) needed to transform one sequence into the other.

ACCTGA and AGCTTA

ACCTGA

AGCCTGA

AGCTTGA

AGCTTA

A\_CCTGA

AGCTT\_A

3 operations

space

ACCTGA

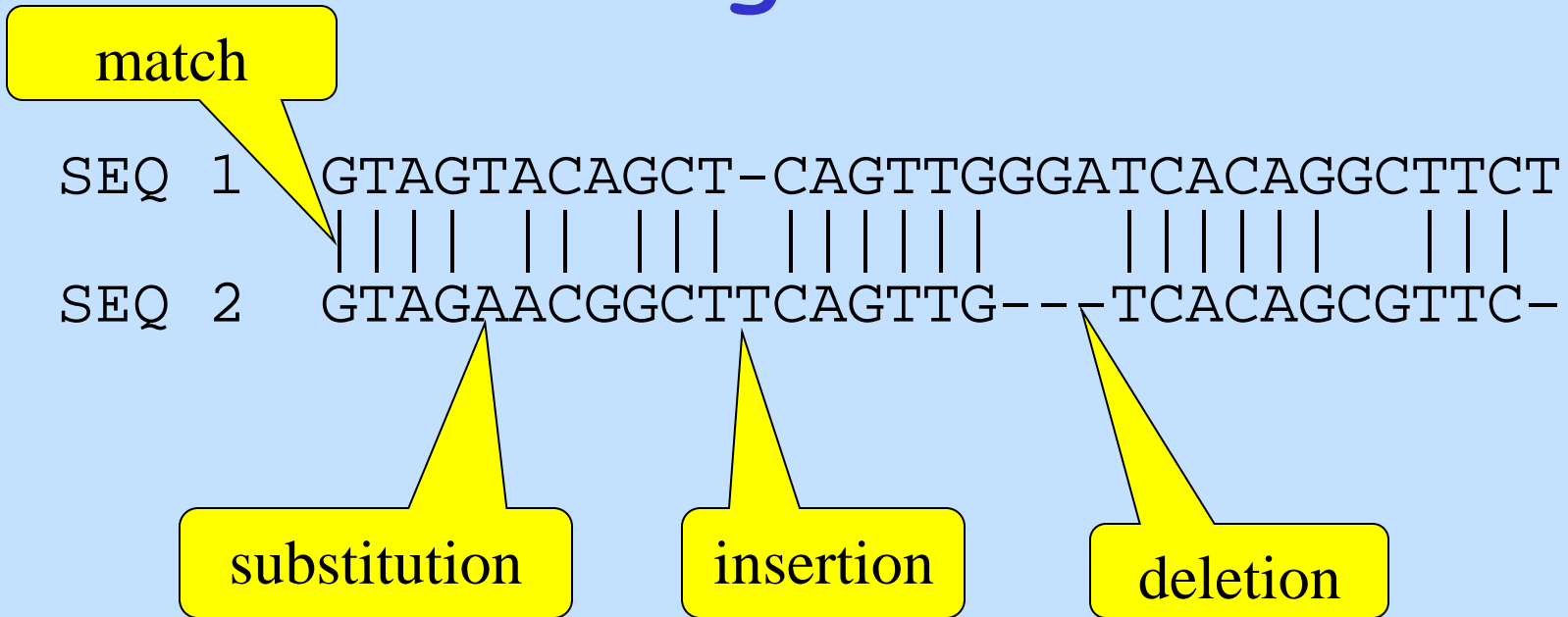
AGCTTA

2 operations

dist=2<sup>6</sup>



# Alignment



- 24 matches,
- Subs: TA, AG, GC, CG
- Indels -T, G-, G-, A-, T-
- Distance I : match 0, subs 1, indel 2 → dist = 14



# Alignment

```
SEQ 1  GTAGTACAGCT-CAGTTGGGATCACAGGCTTCT
        |||||  ||  |||  |||||  |||||  |||||  |||
SEQ 2  GTAGAACGGCTTCAGTTG---TCACAGCGTTC-
```

- 24 matches, Subs: TA, AG, GC, CG, Indels -T, G-, G-, A-, T-
- Distance II: match 0,  $d(A,T)=d(G,C)=1$ ,  $d(A,G)=1.5$  indel 2  
→ dist=14.5
- Similarity I: match 1, subs 0, indel -1.5 → similarity =16.5
- 
- **General setup: substitution matrix  $S(i,j)$ , indel  $S(i,-)$**

Usually symmetric, Alignment - to - not allowed. 8



# Models for Alignment

Problem: Global Alignment

Input: Two sequences  $S$ ,  $T$  of roughly the same length

Goal: Find a best alignment between them

Problem: Local Alignment

Input: Two sequences  $S$ ,  $T$

Goal: Find a subsequence of  $S$  and a subsequence of  $T$  with best alignment (most similar subsequences).



# Models for Alignment (2)

Problem: Ends free alignment (End space-free alignment)

Input: Two sequences  $S, T$  (possibly of different length)

Question: Find a best alignment between subsequences of  $S$  and  $T$  when at least one of these subsequences is a prefix of the original sequence and one (not necessarily the other) is a suffix.

gap: max contiguous run of spaces in a single sequence within a given alignment

Problem: Alignment with Gap Penalty

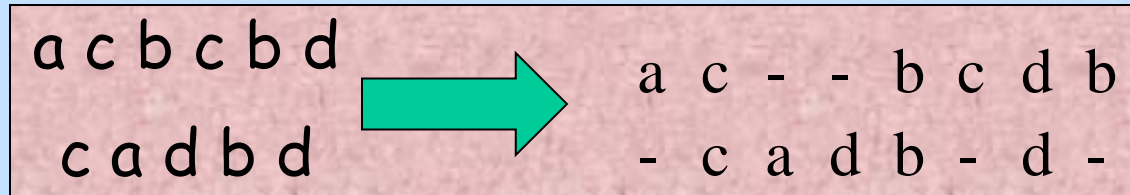
Input: Two sequences  $S, T$

Question: Find a best alignment between them, given a gap penalty function.

measures the cost of a gap as a (nonlinear) function of its length



# Global Alignment



## Global Alignment Problem:

Input: Two sequences  $S=s_1\dots s_n$ ,  $T=t_1\dots t_m$  ( $n\sim m$ )

Goal: Find an optimal alignment

Optimal - max similarity or min distance, according to a given quality measure.



# Global alignment

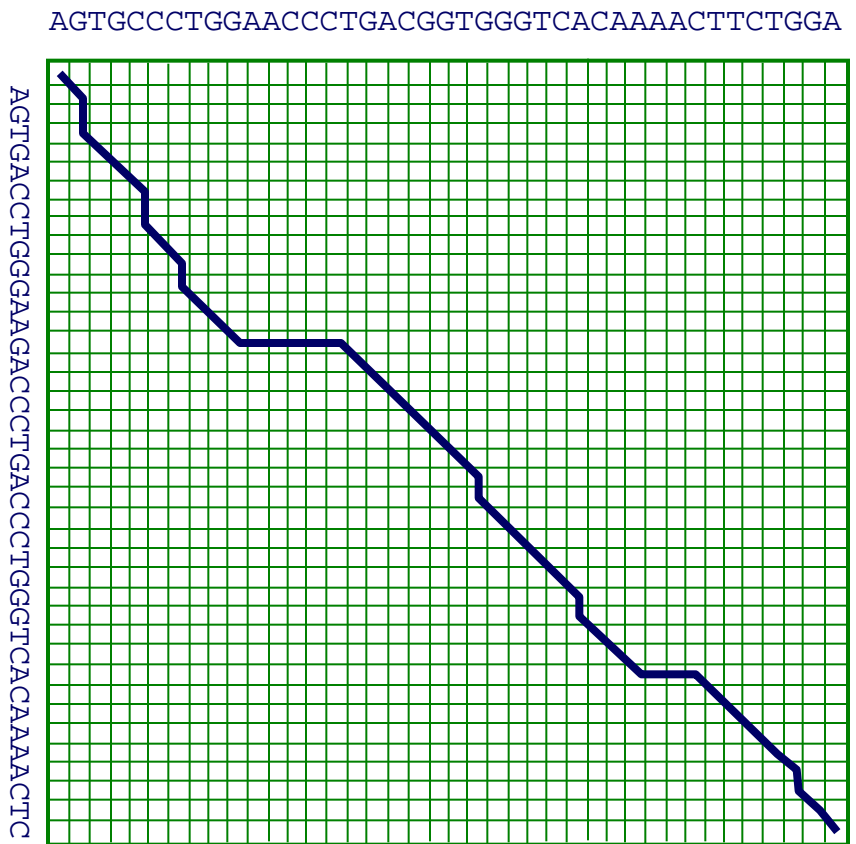
## Human hexosaminidase A vs Mouse hexosaminidase A

```
Identities = 486/753 (65%), Positives = 570/753 (76%)
 1 SE-RGDQR-AMTSSRLWFSLLLAAAFAGRATALWPPQNFQTSQRYVLYPNNFQFQYDVSSAAQPGCSVLDEAFQRYRD
   : || | ||: || |||||:| |||||: || | :||:|||||:| |||| | | |||||:|:|
 1 AAGRGAGRWAMAGCRLWVSLLLAAALACLATALWPPQYIQTYHRRYTLYPNNFQFRYHVSSAAQAGCVVLDEAFRRYRN
 79 LLFGSGSWPRPYLTGKRHTLEKNVLVSVVTPGCMQLPTLESVENYTLTINDDQCLLLSETVWGALRGLETFSQLVWKS
   ||||| ||||| :||:|:| ||||| |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 81 LLFGSGSWPRPSFSNKQQTGKNILVSVVTAECNEFPNLESVENYTLTINDDQCLLASETVWGALRGLETFSQLVWKS
 159 EGTFFINKTEIEDFPRFPHRGLLLDTSRHYLPLSSILDTLDMAYNKLVNFHWHLVDDPSFPYESFTFPELMRKGSYNPV
   |||||:|:| |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 161 EGTFFINKTKIKDFPRFPHRGVLLDTSRHYLPLSSILDTLDMAYNKFVNFHWHLVDDSSFPYESFTFPELTRKGSFNPV
 239 THIYTAQDVKEVIEYARLRGIRVLAEFDTPGHTLSWGGPIGLLTPCYSGSEPSGTFGPVNP SLNNTYEFMSTFFLEVSS
   |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 241 THIYTAQDVKEVIEYARLRGIRVLAEFDTPGHTLSWGGAPGLLTPCYSGSHLSGTFGPVNP SLNSTYDFMSTLFLEISS
 319 VFPDFYLHLGGDEVDFTCWKSNIQDFMKKKGFGEQFKQLESFYIQTLLDIVSSYGGYVWQEVFDNKVKIQPDTIIQ
   |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 321 VFPDFYLHLGGDEVDFTCWKSNIQAQFMKKKGF-TDFKQLESFYIQTLLDIVSDYDKGYVWQEVFDNKVKVRPDTIIQ
 399 VWREDIPVNYMKELELVTKAGFRALLSAPWYLNRIISYGPDWKDFYVVEPLAFEGTPEQKALVIGGEACHWGEYVDNTNLV
   |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 400 VWREEMPVEYMLEMQDITRAGFRALLSAPWYLNRVKYGPDWKDMYKVEPLAFHGTPEQKALVIGGEACHWGEYVDSTNLV
 479 PRLWPRAGAVAERLWSNKLTSDLTFAYERLSHFRCCELLRRGVQAQPLNVGFCEQEFEQT*APGTEEGAGCR*MVVEPGFHF
   |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||: |||||
 480 PRLWPRAGAVAERLWSSNLTINIDFAFKRLSHFRCELVRRGIQAQPIISVGCCEQEFEQT*A--T--SA--E----HPG--
 559 CILARGRSPLPSCPLPACPCAWRERGRRCWRSHSISKNVAFFYFNKHGLPVFKKKS VNGVRVRAQPGWSQCLPLRSFKLRAG
   | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
 548 -----G-----C----CP-----L-SQ-LR--*A-----P---RR-V--LALR-E---Q-VP--G-Q--G
 639 NETYSLCAVLPLCL*AMSLPSHS*PYSRHL*SSACSLHFCIISP RRWYMEKDVGAWRCSGQWGLQTQPGHKRASPPCIL
   : : : | | | : | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
 574 -*SPT-----A-SRPGES--T--P---CP--C--APVT--TEKEAGA---GT--GV--Q---*R-----
```





# How do we compute the best alignment?

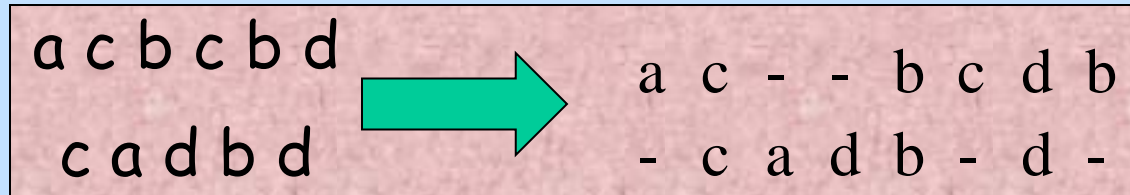


Too many possible alignments:

$$\gg 2^n$$

(exercise)

# Global Alignment



## Global Alignment Problem:

Input: Two sequences  $S=s_1\dots s_n$ ,  $T=t_1\dots t_m$  ( $n\sim m$ )

Goal: Find an optimal alignment

## Notation:

- $\sigma(a,b)$ : the **score** (weight) of the alignment of character a with character b.
- $V(i,j)$ : the **optimal score** of the alignment of  $S'=s_1\dots s_i$  and  $T'=t_1\dots t_j$  ( $0 \leq i \leq n$ ,  $0 \leq j \leq m$ )



$V(i,j)$  := optimal score of the alignment of  $S'=s_1\dots s_i$  and  $T'=t_1\dots t_j$  ( $0 \leq i \leq n$ ,  $0 \leq j \leq m$ )

Lemma:  $V(i,j)$  has the following properties:

• Base conditions:

-  $V(i,0) = \sum_{k=0..i} \sigma(s_k, -)$

-  $V(0,j) = \sum_{k=0..j} \sigma(-, t_k)$

• Recurrence relation:

$$\forall 1 \leq i \leq n, 1 \leq j \leq m: V(i,j) = \max \begin{cases} V(i-1,j-1) + \sigma(s_i, t_j) \\ V(i-1,j) + \sigma(s_i, -) \\ V(i,j-1) + \sigma(-, t_j) \end{cases}$$

Alignment with 0 elements  $\equiv$  spacing

$S'=s_1\dots s_{i-1}$  with  $T'=t_1\dots t_{j-1}$   
 $s_i$  with  $t_j$ .

$S'=s_1\dots s_i$  with  $T'=t_1\dots t_{j-1}$   
 and '-' with  $t_j$ .



# Optimal Alignment - Tabular Computation

- Use dynamic programming to compute  $V(i,j)$  for all possible  $i,j$  values:

```
for i=1 to n do
begin
  For j=1 to m do
  begin
    Calculate  $V(i,j)$  using
     $V(i-1,j-1)$ ,  $V(i,j-1)$ ,  $V(i-1,j)$ 
  end
end
```

		T					
		j	0	1	2	3	4
i			c	a	d	b	d
0		0	-1	-2	-3	-4	-5
1	a	-1	-1	1			
2	c	-2					
3	b	-3					
4	c	-4					
5	d	-5					
6	b	-6					

↑ S Costs: match 2, mismatch/indel -1

Snapshot of computing the table



# Optimal Alignment - Tabular Computation

- Add back pointer(s) from cell  $(i,j)$  to father cell(s) realising  $V(i,j)$ .
- Trace back the pointers from  $(m,n)$  to  $(0,0)$
- Needleman-Wunsch, '70

		← T						
		j	0	1	2	3	4	5
i								
	0		0	-1	-2	-3	-4	-5
	1	a	-1	-1	1	0	-1	-2
	2	c	-2	1	0	0	-1	-2
	3	b	-3	0	0	-1	2	1
	4	c	-4	-1	-1	-1	1	1
	5	d	-5	-2	-2	1	0	3
6	b	-6	-3	-3	0	3	2	

↑ S





# Example

x = AGTA  
y = ATA

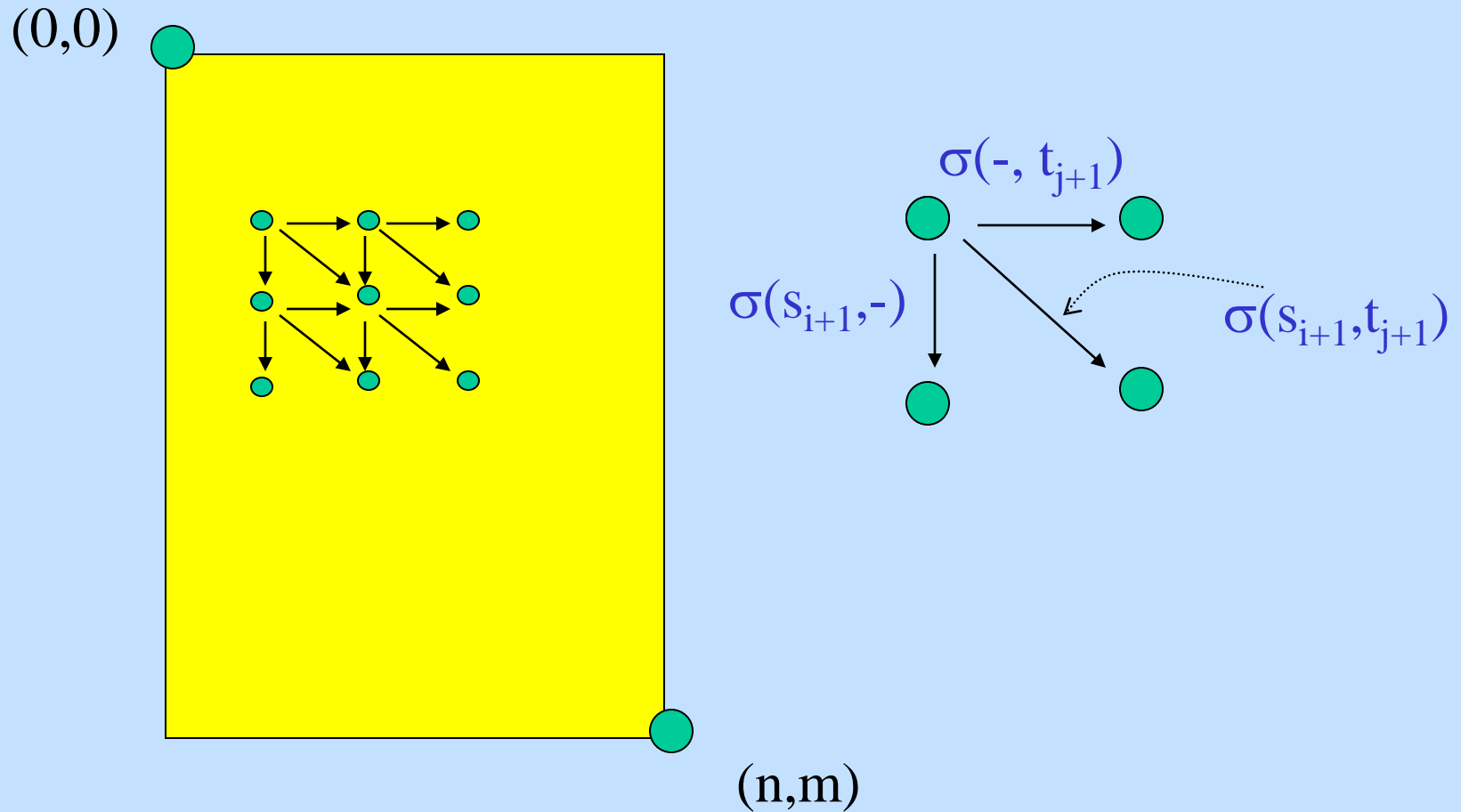
m = 1  
s = -1  
d = -1

F(i,j)	i = 0	1	2	3	4	
j = 0	' '	' '	A	G	T	A
1	A					
2	T					
3	A					

$$V(1, 1) = \max\{V(0,0) + s(A, A), V(0, 1) - d, V(1, 0) - d\} = \max\{0 + 1, -1 - 1, -1 - 1\} = 1$$

AGTA  
A - TA

# Alignment Graph



# Alignment Graph

**Definition:** The **alignment graph** of sequences  $S=s_1\dots s_n$  and  $T=t_1\dots t_m$ , is a directed graph  $G=(V,E)$  on  $(n+1)\times(m+1)$  nodes, each labeled with a distinct pair  $(i,j)$  ( $0\leq i\leq n, 0\leq j\leq m$ ), with the following weighted edges:

- $((i,j), (i+1,j))$  with weight  $\sigma(s_{i+1}, -)$
- $((i,j), (i,j+1))$  with weight  $\sigma(-, t_{j+1})$
- $((i,j), (i+1,j+1))$  with weight  $\sigma(s_{i+1}, t_{j+1})$

**Note:** a path from node  $(0,0)$  to node  $(n,m)$  corresponds to an alignment and its total weight is the alignment score.

**Goal:** find an optimal path from node  $(0,0)$  to node  $(n,m)$



# Complexity

- Time:  $O(mn)$  (proportional to  $|E|$ )
- Space to find opt alignment:  $O(mn)$  (proportional to  $|V|$ )
- Space is often the bottleneck!
- Space to find opt alignment **value only**:  $O(\min(m,n))$  (need to maintain only two rows/columns)
- Can we improve space complexity for finding opt alignment?



# Reducing Space Complexity

$V^*(i,j)$  = opt alignment value of  $s_i \dots s_n$  and  $t_j \dots t_m$

Lemma: 
$$V(n, m) = \max_{0 \leq k \leq m} \left\{ V\left(\frac{n}{2}, k\right) + V^*\left(\frac{n}{2} + 1, k + 1\right) \right\}$$

Pf:  $\max\{\dots\} \leq V(n, m)$ :

- $\forall$  position  $k'$  in  $T$ ,  $\exists$  alignment of  $S$  and  $T$  consisting of:
  - an opt alignment of  $s_1 \dots s_{n/2}$  and  $t_1 \dots t_{k'}$  and
  - a disjoint opt alignment of  $s_{n/2 + 1} \dots s_n$  and  $t_{k'+1} \dots t_m$ .

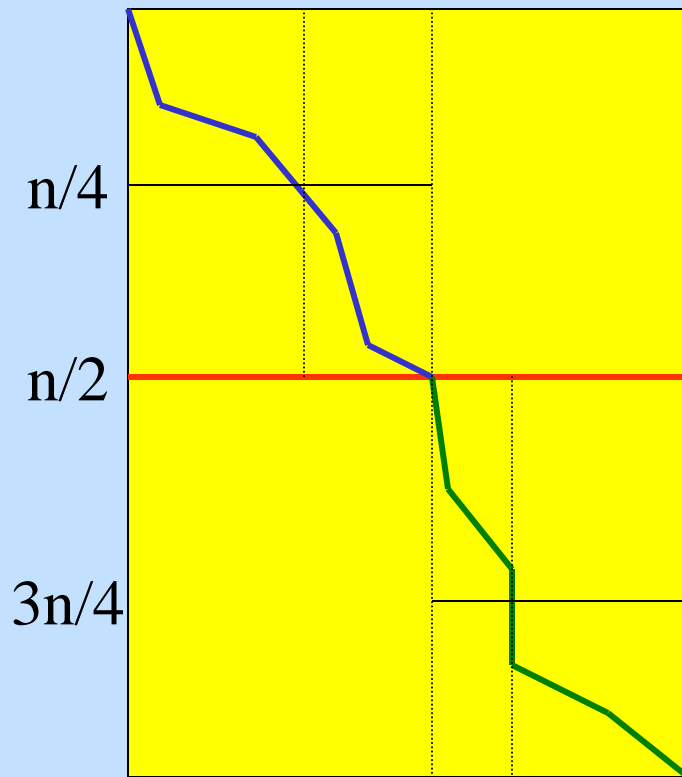


# Proof (contd)

- $\max\{\dots\} \geq V(n,m)$ :
  - For an optimal alignment of  $S$  and  $T$ , let  $k'$  be the rightmost position in  $T$  that is aligned with a character at or before position  $n/2$  in  $S$ . Then the optimal alignment of  $S$  and  $T$  consists of:
    - an alignment of  $s_1 \dots s_{n/2}$  and  $t_1 \dots t_{k'}$  and
    - a disjoint alignment of  $s_{n/2+1} \dots s_n$  and  $t_{k'+1} \dots t_m$ .



# 'Divide & Conquer' Alg (Hirschberg '75)



- Compute opt cost of all paths from start, to any point at **centerline**
- Compute opt cost of back paths from end to any pt at centerline
- Pick centerline pt with opt sum of the two costs
- Continue recursively on the subproblems



# Hirschberg Alg in more detail

$k^*$  - position  $k$  maximizing  
 $V(n/2, k) + V^*(n/2+1, k+1)$

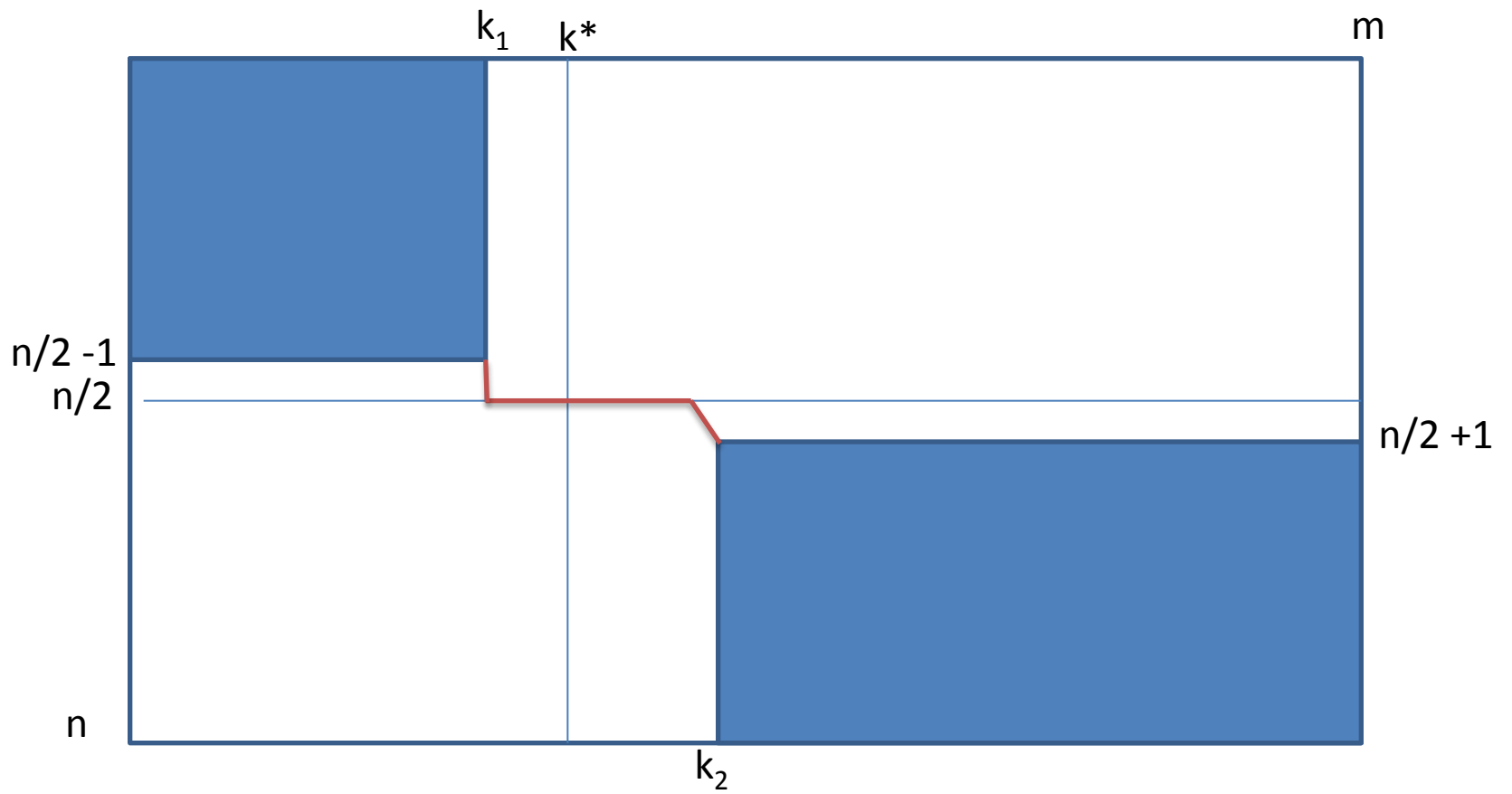
Proved:  $\exists$  opt path  $L$  through  $(n/2, k^*)$

Def:  $L_{n/2}$  - subpath of  $L$  that

- starts with the last node in  $L$  in row  $n/2-1$  and
- ends with the first node in  $L$  in row  $n/2+1$



$L_{n/2}$



Lemma:  $k^*$  can be found in  $O(mn)$  time and  $O(m)$  space.  $L_{n/2}$  can be found and stored in same bounds

Run DP up to including row  $n/2$ , getting values  $V(n/2, i)$  for all  $i$  and back pointers for row  $n/2$   $O(mn)$  time,  $O(m)$  space

Run DP backwards up to including row  $n/2$ , getting values  $V^*(n/2+1, i)$  for all  $i$  and forward pointers for row  $n/2$   $O(m)$  time,  $O(m)$  space

Compute  $V(n/2, i) + V^*(n/2+1, i+1)$  for each  $i$ , get maximizing index  $k^*$   $O(m)$  time,  $O(m)$  space

Use back pointers to compute subpath from  $(n/2, k^*)$  to first node in row  $n/2-1$   $O(m)$  time,  $O(m)$  space

Use forward pointers to compute subpath from  $(n/2, k^*)$  to first node in row  $n/2+1$   $O(m)$  time,  $O(m)$  space incl storage



# Full Alg and Analysis

- Assume time to fill a  $p$  by  $q$  DP matrix :  $cpq$
- → time to compute rows  $V(n/2, ..)$ ,  $V^*(n+1/2, ..)$   $cmn$
- → time  $cmn$ , space  $O(m)$  to find  $k^*$ ,  $k_1$ ,  $k_2$ ,  $L_{n/2}$
- *Recursively* solve top subproblem of size  $\leq nk^*/2$ , bottom subproblem of size  $\leq n(m-k^*)/2$
- Time for top level  $cmn$ , 2<sup>nd</sup> level  $cmn/2$
- Time for all  $i$ -th level computations  $cmn/2^{i-1}$  (each subproblem has  $n/2^i$  rows, the cols of all subprobs are distinct)
- Total time:  $\sum_{i=1 \text{ to } \log n} cmn/2^{i-1} \leq 2cmn$
- Total space:  $O(m)$



# Dan Hirschberg

Daniel S. Hirschberg is a full professor in Computer Science at University of California, Irvine. His research interests are in the theory of design and analysis of algorithms.



Hirschberg, D. S. (1975). "A linear space algorithm for computing maximal common subsequences". *Communications of the ACM* **18** (6): 341–343



# Local Alignment

**Definition:** Given sequences  $S$ ,  $T$ , find subsequences  $\alpha$  of  $S$  and  $\beta$  of  $T$ , of maximum similarity ( i.e., with optimal global alignment between  $\alpha$  &  $\beta$ ).

## **Motivation:**

- ignore stretches of non-coding DNA
- protein domains (functional subunits)

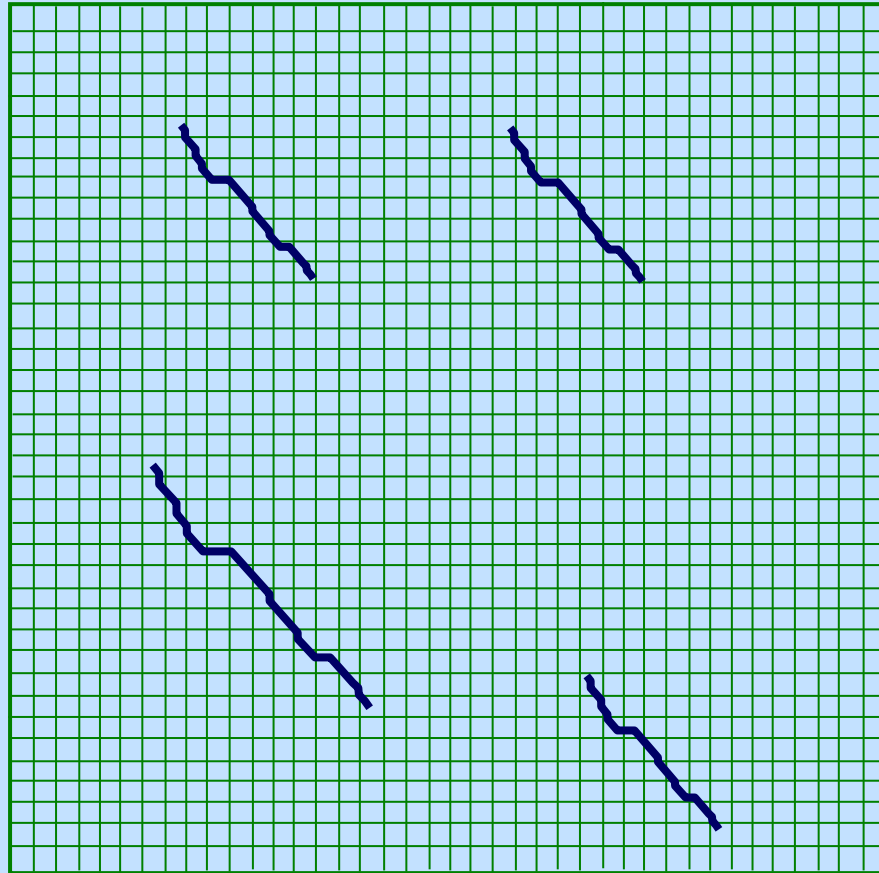
## **Example:**

- $S=abcxdex$ ,  $T=xxxcde$ ,
- Similarity score:  $2$  per match,  
 $-1$  for subs/indel,
- $\alpha=cxde$  and  $\beta=c-de$  have optimal alignment score.

	a	b	c	x	d	e	x
x	x	x	c	-	d	e	d



# Local alignments in the alignment graph



# Computing Local Alignment

- Definition: Let  $S'=s_1\dots s_i$  and  $T'=t_1\dots t_j$

The **local suffix alignment** problem for  $S', T'$ : find a (possibly empty) suffix  $\alpha$  of  $S'=s_1\dots s_i$  and a (possibly empty) suffix  $\beta$  of  $T'=t_1\dots t_j$  such that the value of their alignment is maximum over all values of alignments of suffixes of  $S'$  and  $T'$ .

- $V(i,j)$ : the value of optimal local suffix alignment for a given pair  $i, j$  of indices.
- Assume the weights satisfy:

$$\sigma(x,y) = \begin{cases} \geq 0 & \text{if } x,y \text{ match} \\ \leq 0 & \text{o/w (mismatch or indel)} \end{cases}$$



# Computing Local Alignment (2)

## A scheme of the algorithm:

- Find maximum similarity between suffixes of  $S'=s_1\dots s_i$  and  $T'=t_1\dots t_j$
- Discard the prefixes  $S'=s_1\dots s_i$ , and  $T'=t_1\dots t_j$  whose similarity is  $\leq 0$  (and therefore decrease the overall similarity)
- Find the indices  $i^*$ ,  $j^*$  of  $S$  and  $T$  respectively after which the similarity only decreases.

## Algorithm - Recursive Definition

Base Condition:

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

Recursion Step:  $\forall i>0, j>0$

$$V(i,j) = \max \begin{cases} \hat{0}, \\ V(i-1, j-1) + \sigma(s_i, t_j), \\ V(i, j-1) + \sigma(-, t_j), \\ V(i-1, j) + \sigma(s_i, -) \end{cases}$$

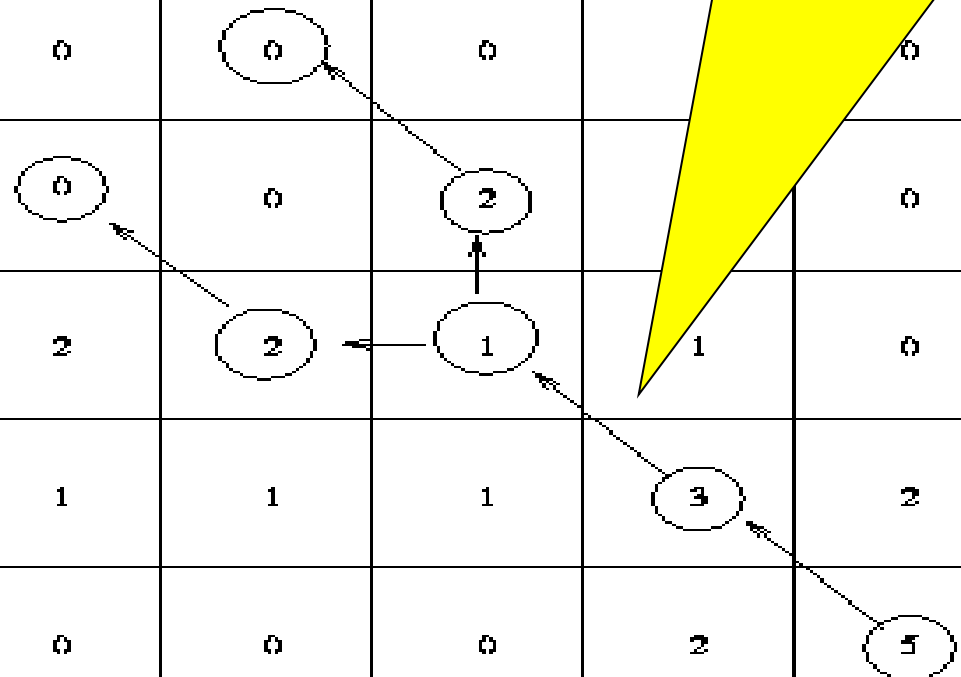
Compute  $i^*, j^*$

$$\text{s.t.} \quad V(i^*, j^*) = \max_{1 \leq i \leq n, 1 \leq j \leq m} V(i,j)$$



- As usual the pointers are created while filling the values in the table,
- The alignments are found by tracking the pointers from cell  $(i^*, j^*)$  until reaching an entry  $(i', j')$  that has value 0.

j \ i	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	2	0	0	0
4	0	2	2	2	2	1	0	0
5	0	1	1	1	1	1	3	2
6	0	0	0	0	0	0	2	5
7	0	2	2	2	2	1	1	4



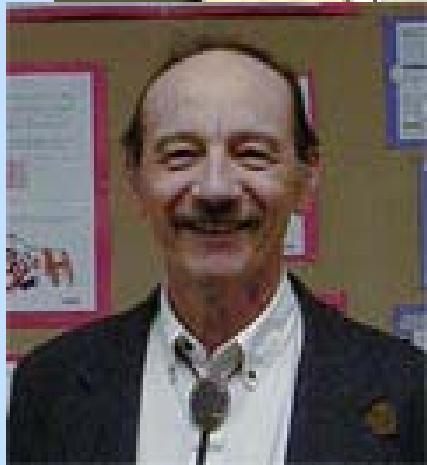
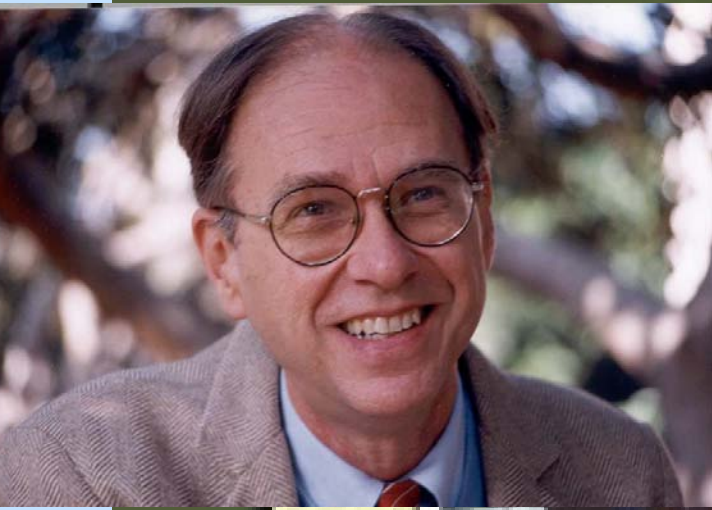
# Computing Local Alignment (3)

## Complexity:

- Time  $O(nm)$
- Space  $O(n+m)$ 
  - The optimum value and the ends of subsequences  $\alpha$  and  $\beta$  can be found in linear space
- Finding the starting point of the two subsequences can be done by reverse dynamic programming using linear space, using Hirschberg's method (exercise).
- Smith-Waterman 81



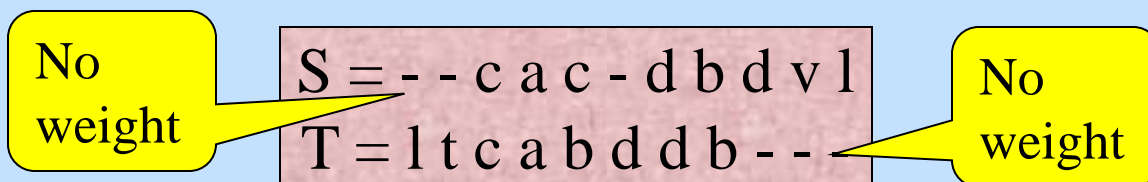
# Smith and Waterman



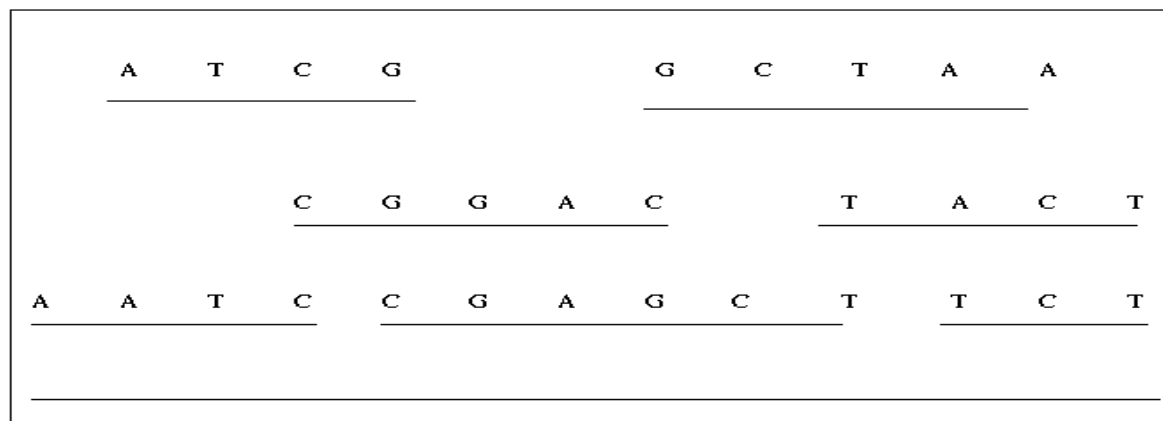
# End-Space Free Alignment

- Suppose spaces at the beginning and the end of the alignment contribute zero weight.

• Example:



- Motivation: "shotgun sequence assembly" - finding the original sequence given many of its subsequences (possibly overlapping).



Sequence assembly



# End-Space Free Alignment (2)

- solution: similar to global alignment alg:

- Base conditions:

$$V(i,0) = 0$$

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

- Recurrence relation:

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \sigma(s_i,t_j) \\ V(i-1,j) + \sigma(s_i,-) \\ V(i,j-1) + \sigma(-,t_j) \end{cases}$$

- Search for  $i^*$  and  $j^*$  such that

$$V(n, i^*) = \max_i \{ V(n, i) \}$$

$$V(j^*, m) = \max_j \{ V(j, m) \}$$

- $V(S, T) = \max \{ V(n, i^*), V(j^*, m) \}$

Instead of:  $\sum_{k=0..i} \sigma(s_k, -)$   
and  $\sum_{k=0..j} \sigma(-, t_k)$

The same as in global alignment

- Time complexity:  $O(nm)$ 
  - computing the matrix:  $O(nm)$ ,
  - finding  $i^*$  and  $j^*$ :  $O(n+m)$ .
- Space complexity:  $O(n+m)$ 
  - computing the matrix:  $O(n+m)$ ,
  - computing  $i^*$  and  $j^*$  requires the last row and column to be saved:  $O(n+m)$

Instead of  $V(n,m)$



# Gap Penalties

- Observation: spaces tend to occur in batches.
- Idea: when scoring an alignment, use the no. of contiguous gaps and not the no. of spaces
- Definitions:
  - A *gap* is any maximal run of consecutive spaces in a single sequence of a given alignment.
  - The *length* of a gap is the number of spaces in it.
  - The number of gaps in the alignment is denoted by *#gaps*.
- Example:
  - 4 gaps, 8 spaces, 7 matches, 0 mismatches.

```
S= attc--ga-tggacc  
T= a--cgtgatt---cc
```



# Gap Penalty Model

## ■ Motivation:

- Insertion or deletions of entire subsequence in a single mutation.
- Gaps of varying sizes in mutations.
- Contiguous gaps are anticipated when aligning cDNA and DNA.

- Gap penalty models: weight of gap is different from simply the sum of indel weights.

## • Gap Penalties Types:

- Constant / Affine / Convex / Arbitrary



# Constant and Affine Gap Weight Models

Notation:  $S', T'$  - the aligned sequences including gaps.

## Constant Gap Penalty Model:

- Each individual space is free,
- Constant weight  $W_g$  for each gap, independent of its length (**gap opening cost**)

Goal: Find an alignment that maximizes:

$$\sum \sigma(s'_i, t'_i) + W_g \times \#gaps$$

## Affine Gap Penalty Model:

- Additionally to  $W_g$ , each space has cost  $W_s$ . (**gap extension cost**)

Goal: Find an alignment that maximizes:

$$\sum \sigma(s'_i, t'_i) + W_g \times \#gaps + W_s \times \#spaces$$



# Alignment with Affine Gap Penalty

## Three Types of Alignments:

- ①

S.....i
T.....j

  - $G(i,j)$  is max value of any alignment of type 1, where  $s_i$  and  $t_j$  match
- ②

S.....i-----
T.....j

  - $E(i,j)$  is max value of any alignment of type 2, where  $t_j$  matches a space
- ③

S.....i
T.....j-----

  - $F(i,j)$  is max value of any alignment of type 3, where  $s_i$  matches a space



# Alignment with Affine Gap Penalty (2)

## Base Conditions:

$$V(i, 0) = F(i, 0) = W_g + iW_s$$

$$V(0, j) = E(0, j) = W_g + jW_s$$

## Recursive Computation:

$$V(i, j) = \max\{ E(i, j), F(i, j), G(i, j) \}$$

where:

- $G(i, j) = V(i-1, j-1) + \sigma(s_i, t_j)$
- $E(i, j) = \max\{ E(i, j-1) + W_s, G(i, j-1) + W_g + W_s, F(i, j-1) + W_g + W_s \}$
- $F(i, j) = \max\{ F(i-1, j) + W_s, G(i-1, j) + W_g + W_s, E(i-1, j) + W_g + W_s \}$

G(i,j)	S.....i T.....j
--------	--------------------

E(i,j)	S.....i----- T.....j
--------	-------------------------

F(i,j)	S.....i T.....j-----
--------	-------------------------

- Time complexity  $O(nm)$  - compute 4 matrices instead of one.
- Space complexity  $O(nm)$  - saving 4 matrices (trivial implementation).

# Other Gap Penalty Models:

## Convex Gap Penalty Model:

- Each additional space in a gap contributes less to the gap weight.
- Biological motivation.
- Example:  $W_g + \log(q)$ , where  $q$  is the length of the gap.
- solvable in  $O(nm \log m)$  time

## Arbitrary Gap Penalty Model:

- Most general gap weight.
- Weight of a gap is an arbitrary function of its length  $w(q)$ .
- solvable in  $O(nm^2 + n^2m)$  time.



# Longest Common Non-contiguous Subsequence

sequence

## Definitions:

nc subsequence

- A non-contiguous (nc) subsequence of the sequence  $S = s_1 s_2 \dots s_n$  is  $s_{i_1} s_{i_2} s_{i_3} \dots s_{i_k}$  where  $i_1 < i_2 < i_3 < \dots < i_k$ .
- A common nc subsequence of the sequences  $S, T$  is a nc subsequence of both in  $S$  and  $T$ .
- Goal: find the longest nc subsequence of  $S$  and  $T$
- Note different cs and bio language here!

abcdefghijklmno

bklmp

sequences

abcdefghig kacdlminop

acdi

Common nc subsequence



# Solution

- Using Optimal Alignment:

The Scoring:

$$\sigma(x, -) = \sigma(-, y) = 0$$

$$\sigma(x, y) = \begin{cases} 1 & x=y \\ 0 & x \neq y \end{cases}$$

- Directly:

$$V(i, 0) = V(0, i) = 0$$

$$V(i, j) = \max \begin{cases} V(i-1, j-1) + \sigma(s_i, t_j) \\ V(i-1, j) \\ V(i, j-1) \end{cases}$$

