

# Sequence alignment (continued)

# Global Alignment (review)

- Input: Two strings, labeled  $s$  and  $t$ 
  - $|s|$  is  $n$
  - $|t|$  is  $m$
- Output: Two strings  $s_A$  and  $t_A$  such that:
  - $s_A$  and  $t_A$  are of equal length  $L$
  - Characters must be in same order, with “-” spacers as needed
  - If  $s_A[i] = \text{'-'}$ , then  $t_A[i] \neq \text{'-'}$
  - If  $t_A[i] = \text{'-'}$ , then  $s_A[i] \neq \text{'-'}$

# Optimal alignment

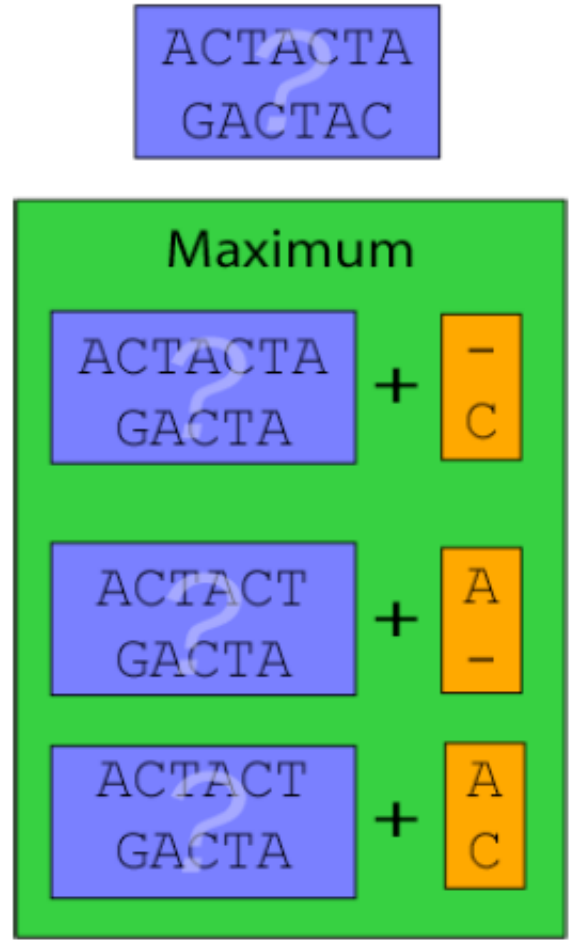
- We want to find an alignment that optimizes an evaluation function  $E(s_A, t_A)$
- The “brute force” method is a hard problem (NP-hard) requiring exponential time.
- The key to saving time is making valid assumptions on the scoring function.

# Types of scores

- Reasons:
  - Sequencing error(s)
  - Evolutionary change
- Three parameters
  - Gap (indel?)
  - Mismatch (misread base?)
  - Match (no change)
- We will only consider constant gap penalties for now.

We can solve this based on looking at three smaller problems

$$T[i,j] = \max \begin{cases} T[i-1,j-1] + \text{score}(s[i],t[j]) \\ T[i-1,j] + g \\ T[i,j-1] + g \end{cases}$$



	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5								
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

$$A[i, 0] = ig \quad A[0, j] = jg, \quad g = -5$$

	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5						
A	-10								
T	-15								
T	-20								
C	-25								
A	-30								
C	-35								

+10 for match, -2 for mismatch, -5 for space (rowwise)

# Time and space requirements

- $O(mn)$  space
  - 2D array of size  $m * n$
- $O(mn)$  time
  - Constant amount of work per cell in the dynamic programming table

# Practice

$$T[i, 0] = ig \quad T[0, j] = jg, \quad g = -3$$

$$T[i, j] = \max \begin{cases} T[i-1, j-1] + \text{score}(s[i], t[j]) \\ T[i-1, j] + g \\ T[i, j-1] + g \end{cases}$$

Match = 5

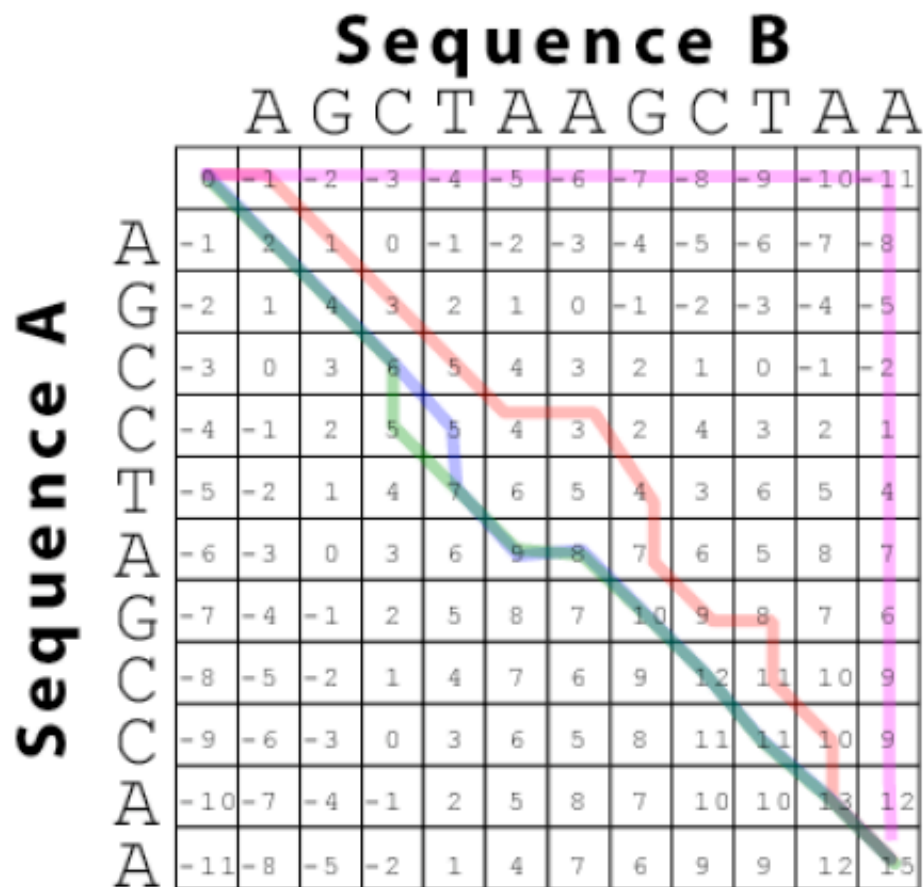
Mismatch = -2

S = AGCATTA

T = ACATTTAG

Find the score of  
the best alignment

# Example traceback paths



-AGCC-TAG-CCAA  
AGCTAAG-CT-A-A

AGCCTA-GCCAA  
AGCT-AAGCTAA

AGCCTA-GCCAA  
AGC-TAAGCTAA

-----AGCCTAGCCAA  
AGCTAAGCTAA-----

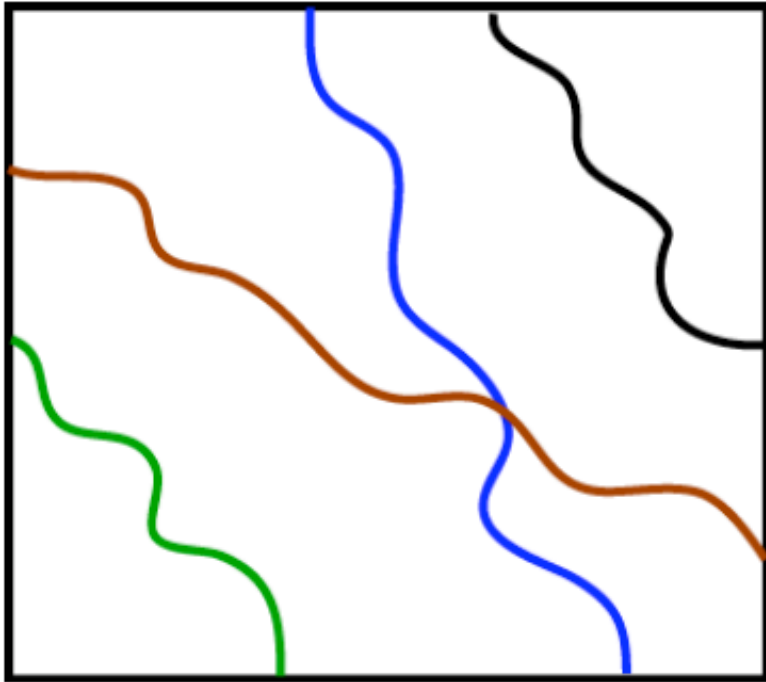
	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5	0	-5	-2	-7
T	-20	-5	10*	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

Traceback yields both optimal alignments in this example

# End-gap free alignment

- We often don't want to penalize gaps at the start or end of the alignment, especially when comparing short and long sequences
- Same as global alignment, except:
  - Initialize with zeros (free gaps at start)
  - Locate max in the last row/column (free gaps at end)
- Also called *semiglobal alignment*

# Example paths



# Practice

$$T[i, 0] = 0 \quad T[0, j] = 0$$

$$T[i, j] = \max \begin{cases} T[i-1, j-1] + \text{score}(s[i], t[j]) \\ T[i-1, j] + g \\ T[i, j-1] + g \end{cases}$$

Match = 5

Mismatch = -2

Gap = -3

S = AGCATTA

T = ACATTTAG

Find the score of  
the best semiglobal  
alignment

	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	0	0	0	0	0	0	0	0
C	0	10	5	10	5	10	5	0	10
A	0	5	8	5	8	5	20	15	10
T	0	0	15	10	5	6	15	18	13
T	0	-2	10	13	8	3	10	13	16
C	0	10	5	20	15	18	13	8	23
A	0	5	8	15	18	13	28	23	18
G	0	0	3	10	25	20	23	38	33

+10 for match, -2 for mismatch, -5 for space (rowwise)

# Local Alignment

$T[i, j]$  = Score of optimally aligning a suffix of  $s$  with a suffix of  $t$ .

$$T[i, j] = \max \begin{cases} T[i-1, j-1] + \text{score}(s[i], t[j]) \\ T[i-1, j] + g \\ T[i, j-1] + g \\ 0 \end{cases}$$

Initialize top row and leftmost column to zero.

# Practice

$$T[i, 0] = 0 \quad T[0, j] = 0$$

$$T[i, j] = \max \begin{cases} T[i-1, j-1] + \text{score}(s[i], t[j]) \\ T[i-1, j] + g \\ T[i, j-1] + g \\ 0 \end{cases}$$

Match = 5

Mismatch = -2

Gap = -3

S = AGCATTA

T = ACATTTAG

Find the score of  
the best local alignment

	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	0	0	0	0	0	0	0	0
C	0	1	0	1	0	1	0	0	1
A	0	0	0	0	0	0	2	0	0
T	0	0	1	0	0	0	0	1	0
T	0	0	1	0	0	0	0	0	0
C	0	1	0	2	0	1	0	0	1
A	0	0	0	0	1	0	2	0	0
C	0	1	0	1	0	2	0	1	1

+1 for a match, -1 for a mismatch, -5 for a gap

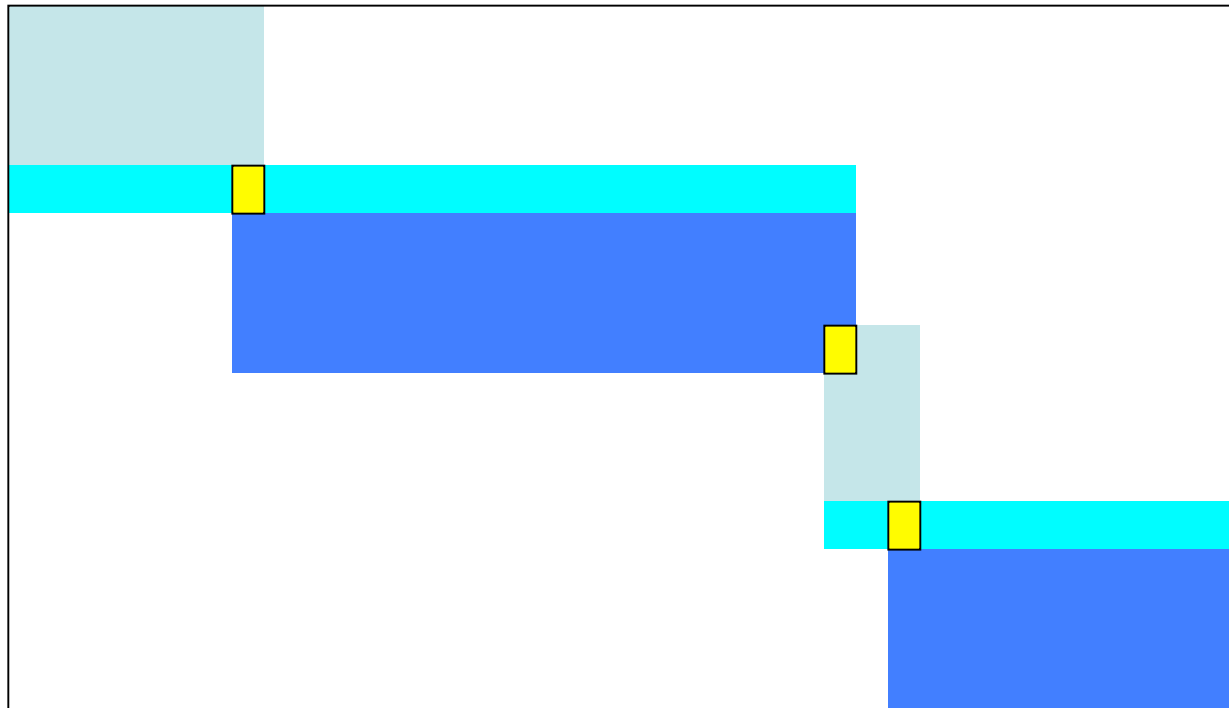
# Reducing space requirements

- $O(mn)$  tables are often the limiting factor in computing large alignments
- There is a linear space technique that only doubles the time required [Hirschberg77]

	$\lambda$	C	T	C	G	C	A	G	C
$\lambda$	0	0	0	0	0	0	0	0	0
C	0	10	5	10	5	10	5	0	10
A	0	5	8	5	8	5	20	15	10
T									
T									
C									
A									
G									

IDEA: We only need the *previous* row to calculate the *next*

# Linear-space Alignments



$$mn + \frac{1}{2} mn + \frac{1}{4} mn + \frac{1}{8} mn + \frac{1}{16} mn + \dots = 2 mn$$

# Alignment and new architectures

- [http://en.wikipedia.org/wiki/Smith-Waterman\\_algorithm](http://en.wikipedia.org/wiki/Smith-Waterman_algorithm)