

Pairwise Sequence Alignment

BMI/CS 576

www.biostat.wisc.edu/bmi576

Irene Ong

irene.ong@wisc.edu

Fall 2017

Overview

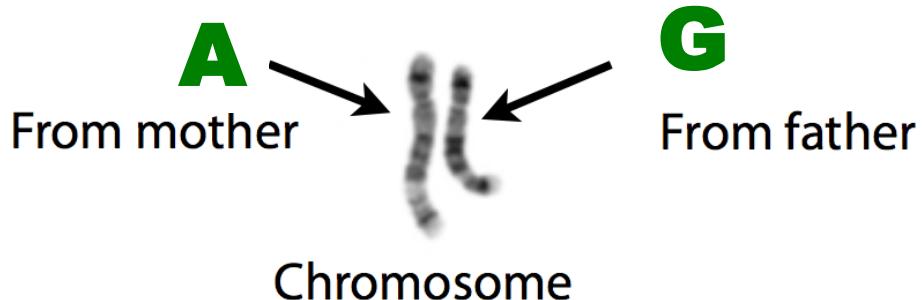
- What does it *mean* to *align* sequences?
- How do we cast sequence alignment as a *computational problem*?
- What *algorithms* exist for solving this computational problem?

Genotype and Phenotype

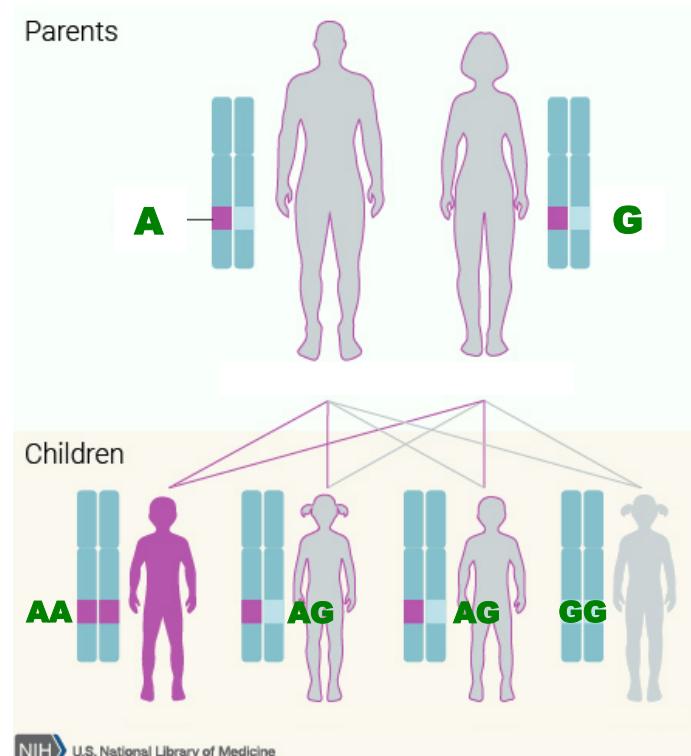
Genotype is all the inherited information

Mother **ACCTCT**A**TTCA**
Father **ACGTCT**G**TCCA**

A and G are alleles, the variable site is in a gene

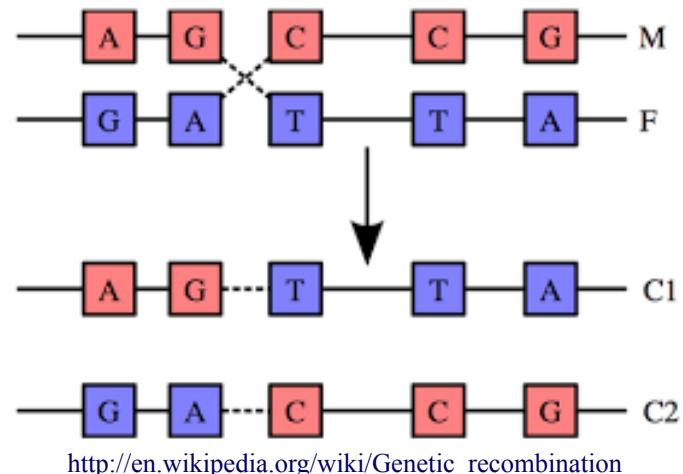


Phenotype is something we observe: eye color

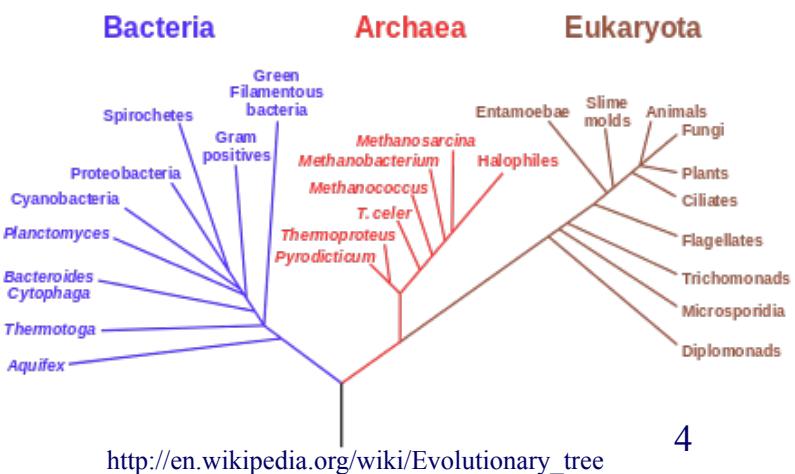


Evolution: select these genotypes

- Organisms reproduce, offspring inherit genotype
- Random mutation changes genotype and recombination shuffles chunks of genotypes in new combination
- Natural selection favors phenotypes that reproduce more
- Over time, produces diversity of life on earth. Incredibly, all organisms share a common ancestor

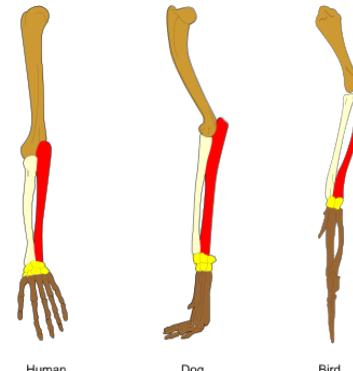


Phylogenetic Tree of Life

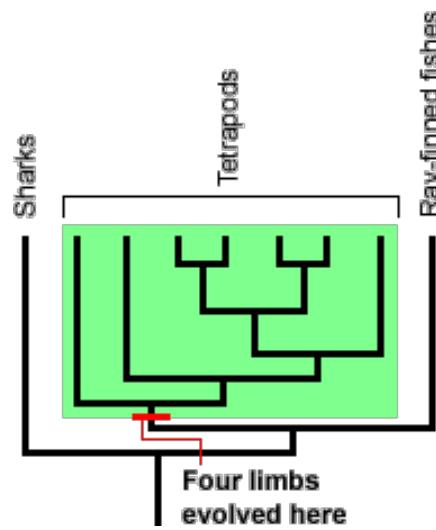


The Role of Homology

- *character*: some feature of an organism (could be molecular, structural, behavioral, etc.)
- *homology*: the relationship of two characters that have descended from a common ancestor



[http://en.wikipedia.org/wiki/Homology_\(biology\)](http://en.wikipedia.org/wiki/Homology_(biology))



http://evolution.berkeley.edu/evolibrary/article/evo_09 5

The Role of Homology

- homologous characters tend to be similar due to their common ancestry and evolutionary pressures
- thus we often infer homology from similarity
- similarly we can sometimes infer structure/function from sequence similarity

What is sequence alignment?

Pattern matching

suffix trees, locality-sensitive hashing,...

Database searching

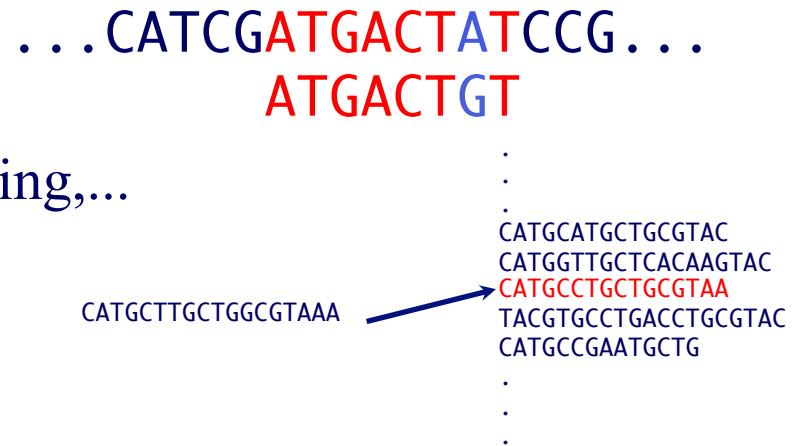
BLAST

Optimization problem

Needleman-Wunsch, Smith-Waterman,...

Statistical problem

Pair HMMs, TKF, Karlin-Altschul statistic...



$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{\int_{\theta'} P(D|\theta')}$$

DNA sequence edits

- Substitutions: A**C**GA → A**G**GA
- Insertions: ACGA → ACC**CGGAGA**
- Deletions: A**CGGAGA** → AGA
- Transpositions: A**CGGAGA** → AAG**CGG**A
- Inversions: AC**GGAGA** → ACT**CCGA**

Alignment scales

- For proteins and short DNA sequences (gene scale) we will generally only consider
 - Substitutions: cause *mismatches* in alignments
 - Insertions/Deletions: cause *gaps* in alignments
- For long DNA sequences (genome scale) we will consider additional events
 - Transposition
 - Inversion
- In this course we will focus on the case of short sequences

What is a pairwise alignment?

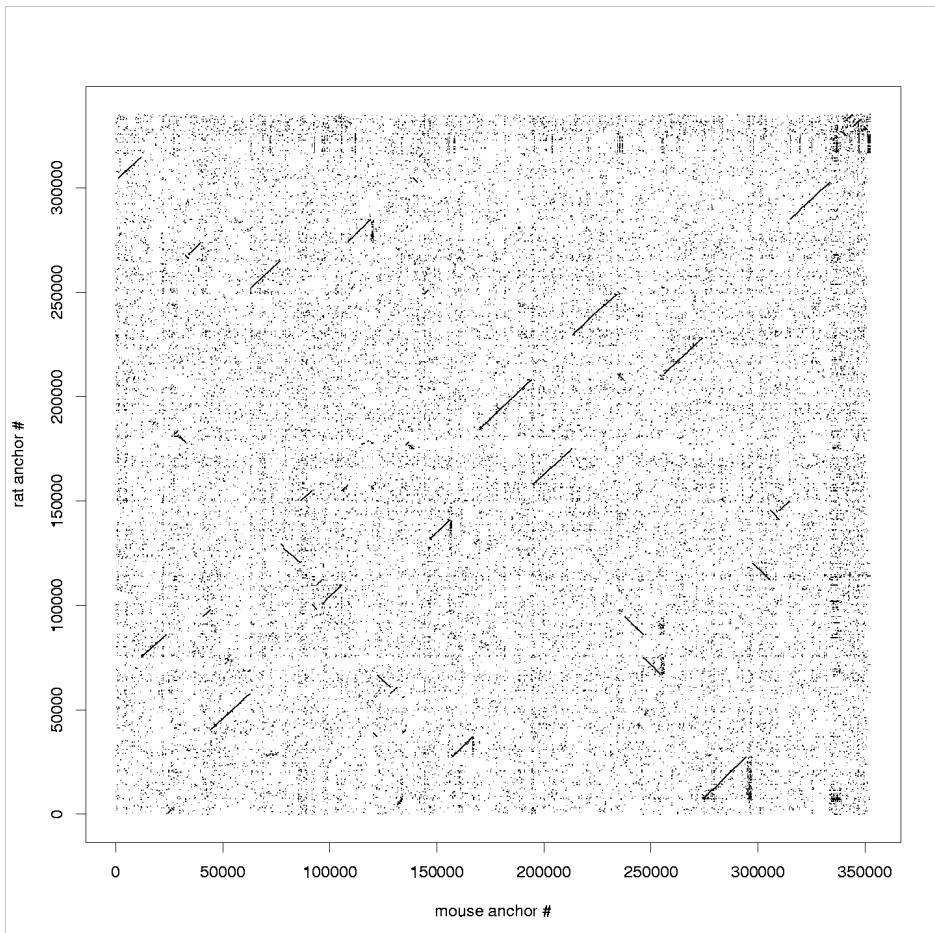
- We will focus on *evolutionary* alignment
- matching of *homologous* positions in two sequences
- positions with no homologous pair are matched with a *space* ‘-’
- A group of consecutive spaces is a *gap*

CA--GATTCTGAAT
CGCCGATT---AT

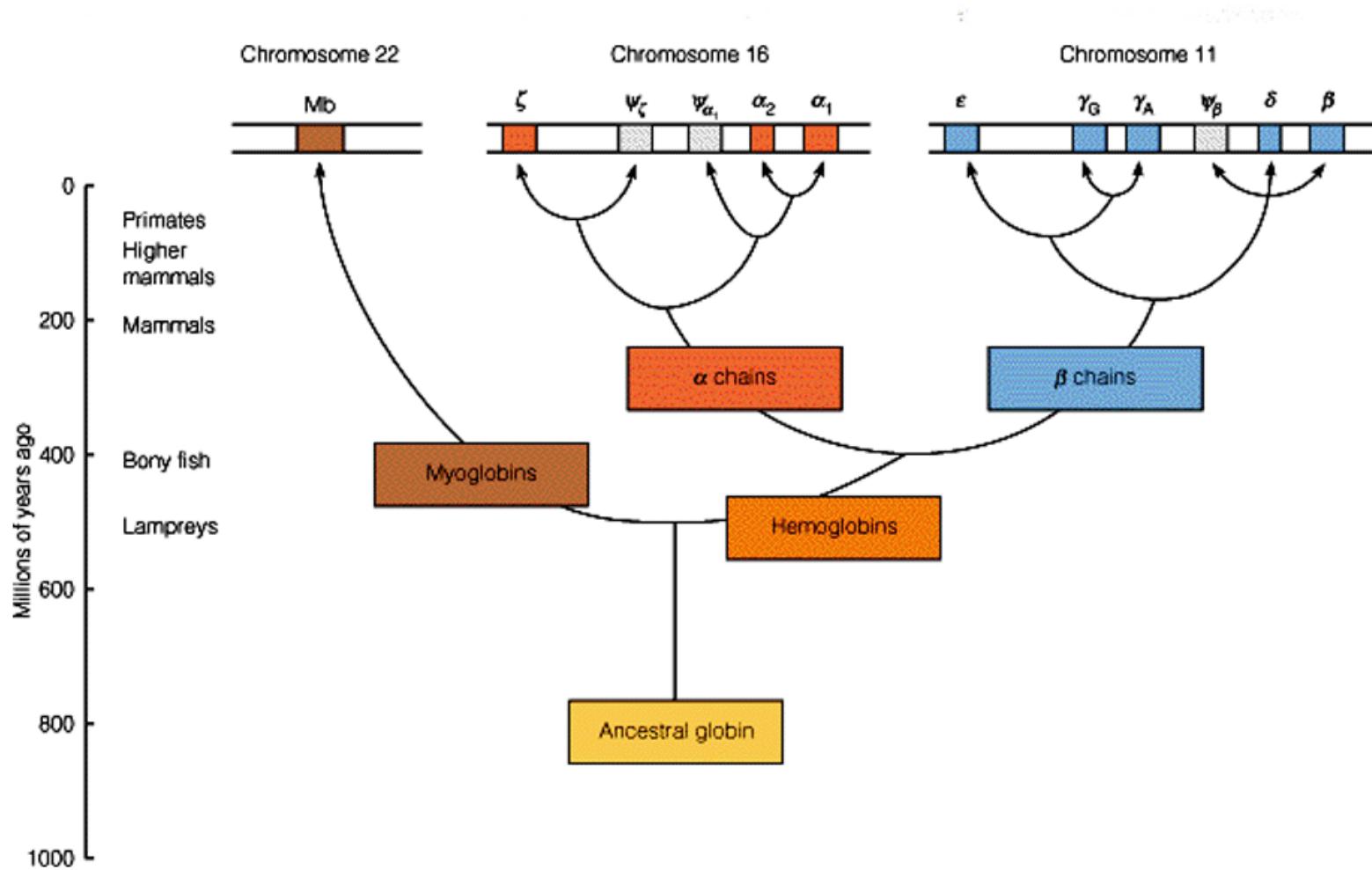
gap

Dot plots

- Not technically an “alignment”
- But gives picture of correspondence between pairs of sequences
- Dot represents similarity between segments of the two sequences

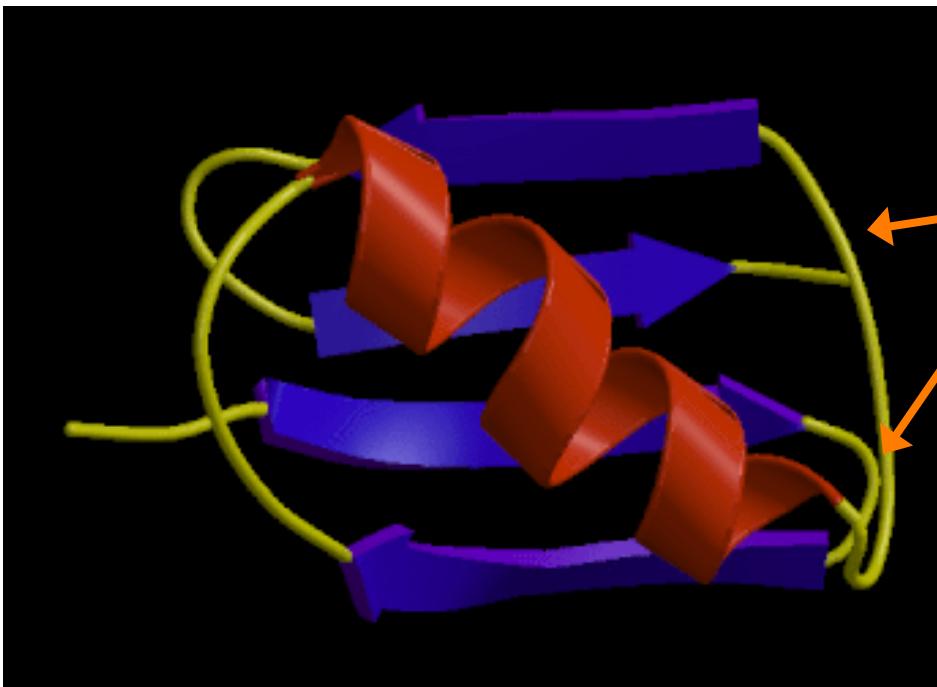


Homology Example: Evolution of the Globins



Insertions/Deletions and Protein Structure

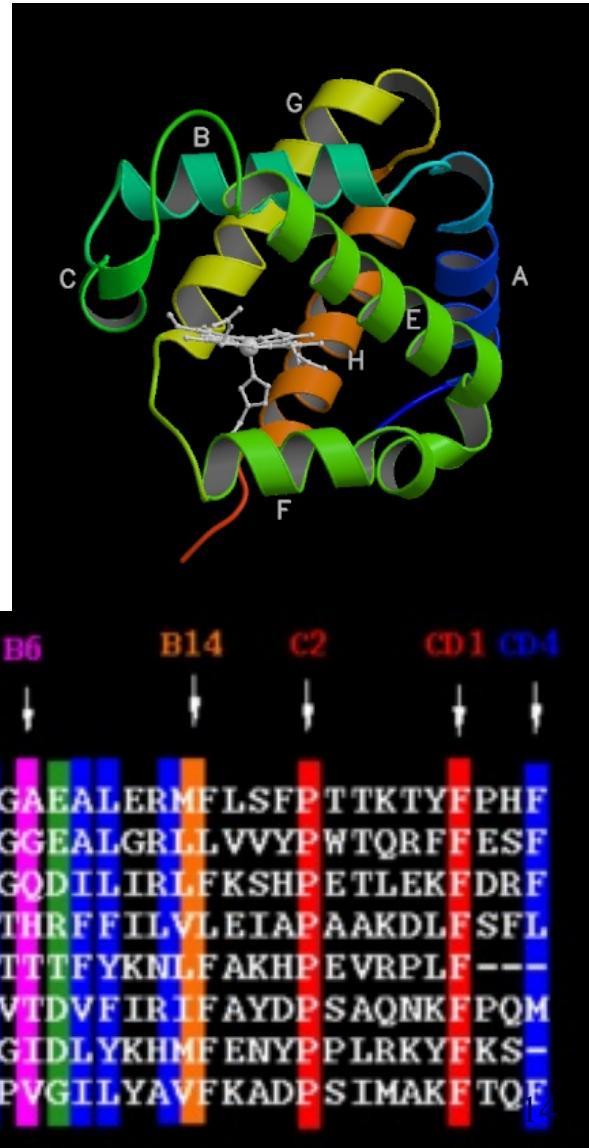
- Why is it that two “similar” sequences may have large insertions/deletions?
 - some insertions and deletions may not significantly affect the structure of a protein



loop structures: insertions/deletions here not so significant

Example Alignment: Globins

- figure at right shows prototypical structure of globins
- figure below shows part of alignment for 8 globins (-'s indicate gaps)



Issues in Sequence Alignment

- the sequences we're comparing probably differ in length
- there may be only a relatively small region in the sequences that matches
- we want to allow partial matches (i.e. some amino acid pairs are more substitutable than others)
- variable length regions may have been inserted/ deleted from the common ancestral sequence

Two main classes of pairwise alignment

- Global: All positions are aligned

CA--GAGTCGAAT

CGCCGA-TC-A--

- Local: A (contiguous) subset of positions are aligned

...GAGTC....

....GA-TC.

Pairwise Alignment: Task Definition

- Given
 - a pair of sequences (DNA or protein)
 - a method for scoring a candidate alignment
- Do
 - find an alignment for which the score is maximized

The Alignment Task

A	T	G	T	T	A	T	A
A	T	C	G	T	C	C	

Alignment Task (maximizing the number of points):

- Remove the 1st symbol from each sequence
 - 1 point if the symbols match, 0 points if they don't match
- Remove the 1st symbol from one of the sequences
 - 0 points

The Alignment Task

A T G T T A T A
A T C G T C C
+1

The Alignment Task

A	T	G	T	T	A	T	A
A	T	C	G	T	C	C	
+1+1							

The Alignment Task

A T - G T T A T A
A T C G T C C
+1+1

The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	C	C		
+1	+1		+1					

The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	C	C		
+1+1	+1+1							

The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	-	C	C	
+1+1	+1+1							

The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	-	C	C	
+1+1	+1+1							

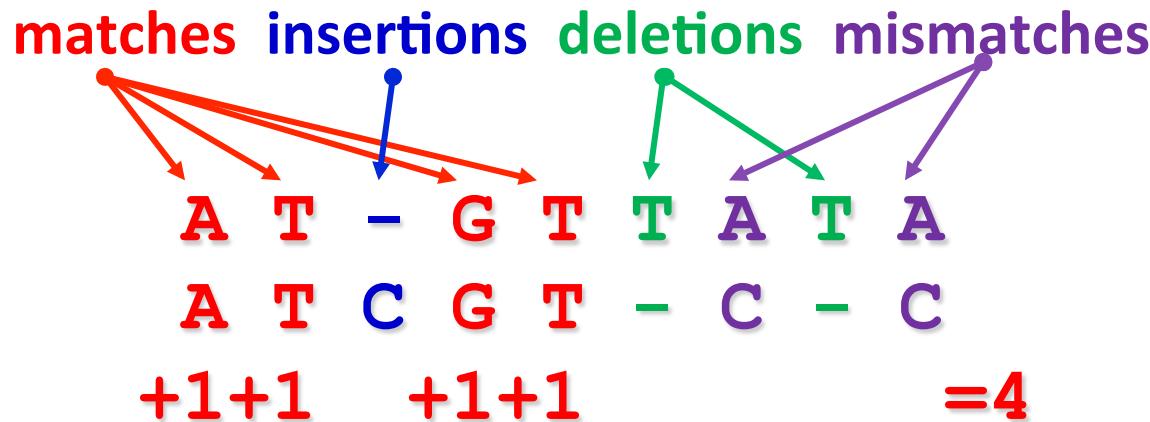
The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	-	C	-	C
+1+1	+1+1							

The Alignment Task

A	T	-	G	T	T	A	T	A
A	T	C	G	T	-	C	-	C
+1+1	+1+1						=4	

What Is the Sequence Alignment?



Alignment of two sequences is a two-row matrix:

1st row: symbols of the 1st sequence (in order) interspersed by “-”
2nd row: symbols of the 2nd sequence (in order) interspersed by “-”

Longest Common Subsequence

A	T	-	G	T	T	A	T	A
A	T	C	G	T	-	C	-	C

Matches in alignment of two sequences (**ATGT**) form their
Common Subsequence

Longest Common Subsequence Problem: Find a longest common subsequence of two strings.

- **Input:** Two strings.
- **Output:** A longest common subsequence of these strings.

Scoring An Alignment: What Is Needed?

- substitution matrix
 - $S(a,b)$ indicates score of aligning character a with character b
- gap penalty function
 - $w(k)$ indicates cost of a gap of length k

Linear Gap Penalty Function

- different gap penalty functions require somewhat different algorithms
- the simplest case is when a linear gap function is used

$$w(k) = s \times k$$

where s is a constant and $w(k)$ is cost of gap of length k

- we'll start by considering this case

Scoring an Alignment

- the score of an alignment is the sum of the scores for pairs of aligned characters plus the scores for gaps
- example: given the following alignment

VAHV---D--DMPNALSALSDLHAHKL

AIQLQVTGVVVTDATLKNLGSVHVSKG

- we would score it by

$$S(V,A) + S(A,I) + S(H,Q) + S(V,L) + 3s + S(D,G) + 2s \dots$$

The Space of Global Alignments

- some possible global alignments for **ELV** and **VIS**

ELV

VIS

-ELV

VIS-

--ELV

VIS--

ELV-

-VIS

E-LV

VIS-

ELV--

--VIS

EL-V

-VIS

Number of Possible Alignments

- given sequences of length m and n
- assume we don't count as distinct $\begin{matrix} \text{C-} \\ -\text{G} \end{matrix}$ and $\begin{matrix} -\text{C} \\ \text{G-} \end{matrix}$
- we can have as few as 0 and as many as $\min\{m, n\}$ aligned pairs
- therefore the number of possible alignments is given by

$$\sum_{k=0}^{\min\{m,n\}} \binom{n}{k} \binom{m}{k} = \binom{n+m}{n}$$

Number of Possible Alignments

- there are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

possible global alignments for 2 sequences of length n

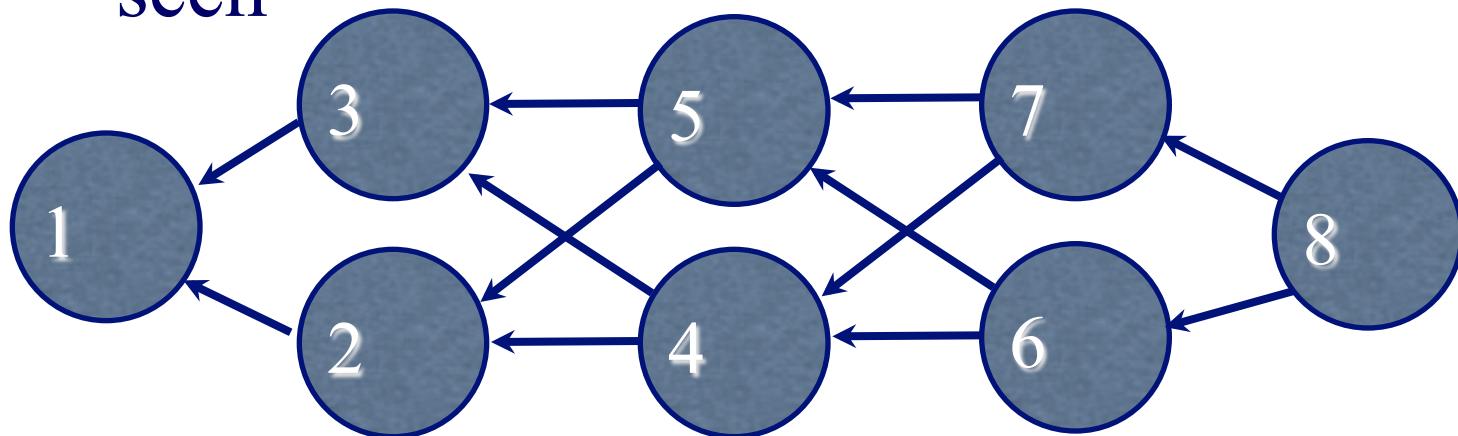
- e.g. two sequences of length 100 have $\approx 10^{59}$ possible alignments
- but we can use *dynamic programming* to find an optimal alignment efficiently

Dynamic Programming

- Algorithmic technique for optimization problems that have two properties:
 - *Optimal substructure*: Optimal solution can be computed from optimal solutions to subproblems
 - *Overlapping subproblems*: Subproblems overlap such that the total number of distinct subproblems to be solved is relatively small

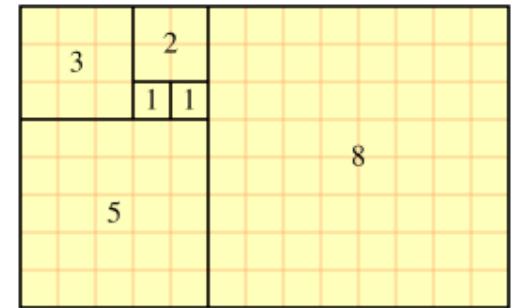
Dynamic Programming

- Break problem into overlapping subproblems
- use *memoization*: remember solutions to subproblems that we have already seen



Fibonacci example

- 1,1,2,3,5,8,13,21,...
- $fib(n) = fib(n - 2) + fib(n - 1)$
- Could implement as a simple recursive function
- However, complexity of simple recursive function is exponential in n

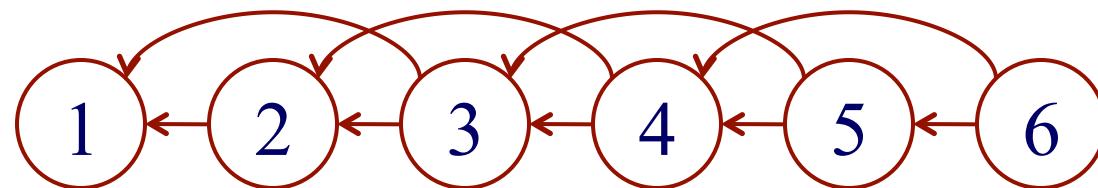


Fibonacci dynamic programming

- Two approaches
 1. *Memoization*: Store results from previous calls of function in a table (top down approach)
 2. Solve subproblems from smallest to largest, storing results in table (bottom up approach)
- Both require evaluating all $(n-1)$ subproblems only once: $O(n)$

Dynamic Programming Graphs

- Dynamic programming algorithms can be represented by a directed acyclic graph
 - Each subproblem is a vertex
 - Direct dependencies between subproblems are edges



graph for $\text{fib}(6)$

Why “Dynamic Programming”?

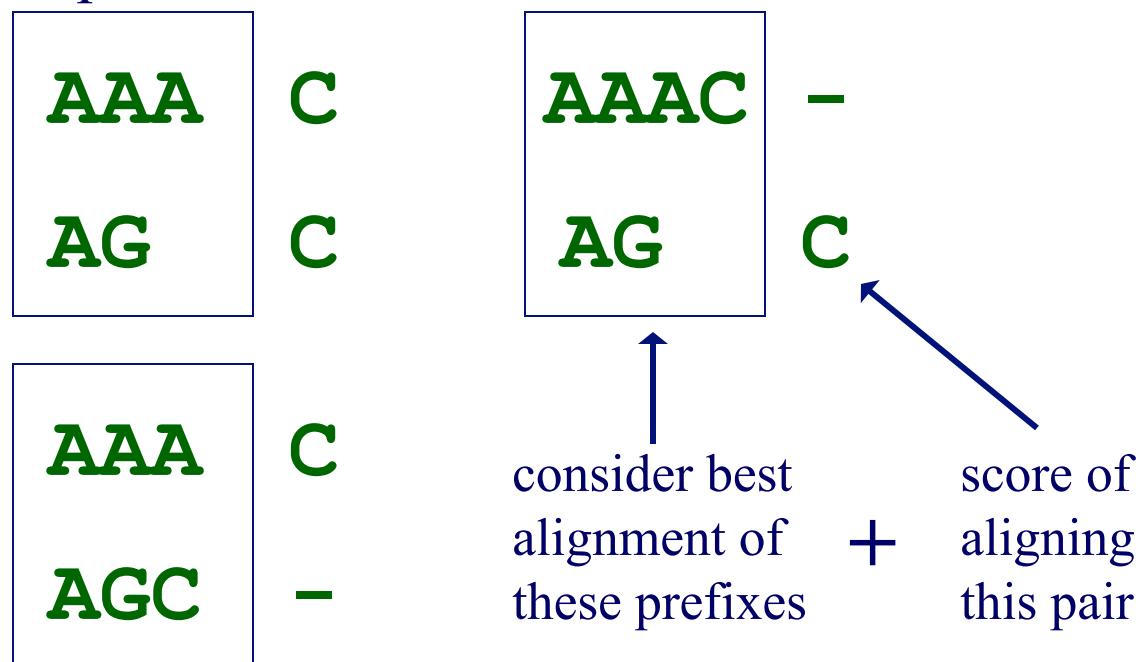
- Coined by Richard Bellman in 1950 while working at RAND
- Government officials were overseeing RAND, disliked research and mathematics
- “programming”: planning, decision making (optimization)
- “dynamic”: multistage, time varying
- “It was something not even a Congressman could object to. So I used it as an umbrella for my activities”

Pairwise Alignment Via Dynamic Programming

- first algorithm by Needleman & Wunsch,
Journal of Molecular Biology, 1970
- *dynamic programming algorithm*:
determine best alignment of two sequences
by determining best alignment of all
prefixes of the sequences

Dynamic Programming Idea

- consider last step in computing alignment of **AAAC** with **AGC**
- three possible options; in each we'll choose a different pairing for end of alignment, and add this to the best alignment of previous characters



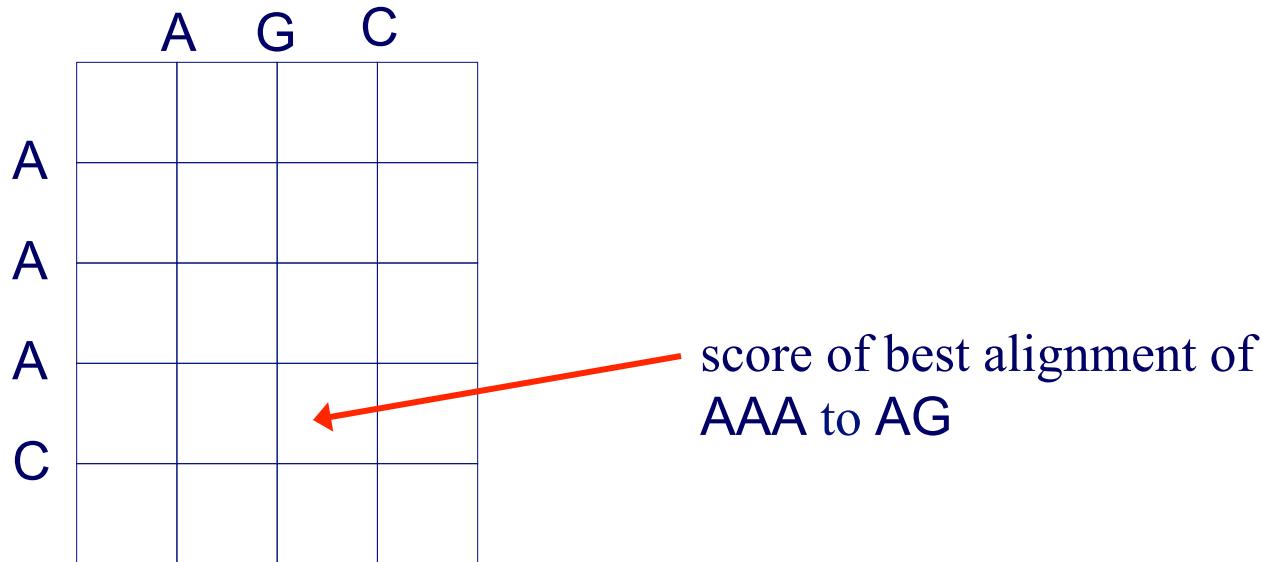
DP Algorithm for Global Alignment with Linear Gap Penalty

- Subproblem: $F(i, j)$ = score of best alignment of the length i prefix of x and the length j prefix of y .

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + S(x_i, y_j) \\ F(i - 1, j) + s \\ F(i, j - 1) + s \end{cases}$$

Dynamic Programming Implementation

- given an n -character sequence x , and an m -character sequence y
- construct an $(n+1) \times (m+1)$ matrix F
- $F(i, j) = \text{score of the best alignment of}$
 $x[1\dots i]$ with $y[1\dots j]$



Initializing Matrix: Global Alignment with Linear Gap Penalty

	A	G	C	
A	0 ← s ← $2s$ ← $3s$			
A				
A				
A				
C				

Diagram illustrating the initialization of a global alignment matrix with linear gap penalties. The columns are labeled A, G, C and the rows are labeled A, A, A, A, C. The diagonal elements are initialized to 0, s, 2s, 3s, and 4s respectively. Red arrows indicate the propagation of these values from left to right across the matrix.

DP Algorithm Sketch: Global Alignment

- initialize first row and column of matrix
- fill in rest of matrix from top to bottom, left to right
- for each $F(i, j)$, save pointer(s) to cell(s) that resulted in best score
- $F(m, n)$ holds the optimal alignment score; trace pointers back from $F(m, n)$ to $F(0, 0)$ to recover alignment

Global Alignment Example

- suppose we choose the following scoring scheme:

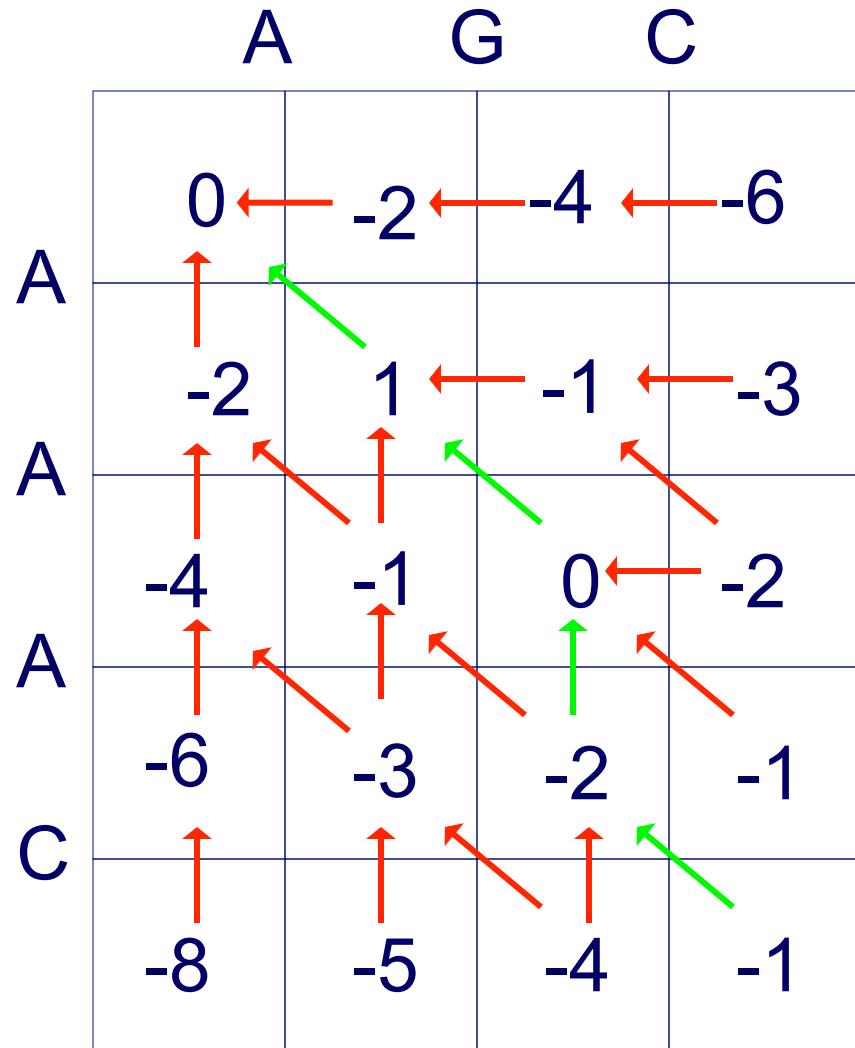
$$S(x_i, y_i) =$$

+1 when $x_i = y_i$

-1 when $x_i \neq y_i$

s (penalty for aligning with a space) = -2

Global Alignment Example



one optimal alignment

x: A A A C
y: A G - C

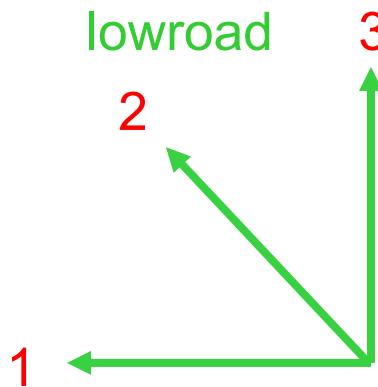
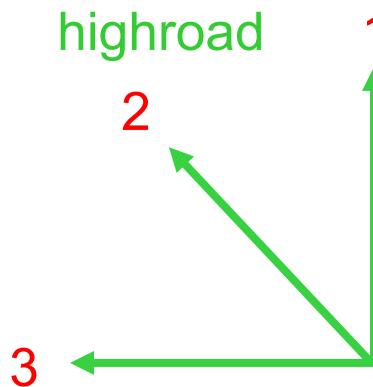
but there are three
optimal alignments
here (can you find
them?)

DP Comments

- works for either DNA or protein sequences, although the substitution matrices used differ
- finds an optimal alignment
- the exact algorithm (and computational complexity) depends on gap penalty function (we'll come back to this issue)

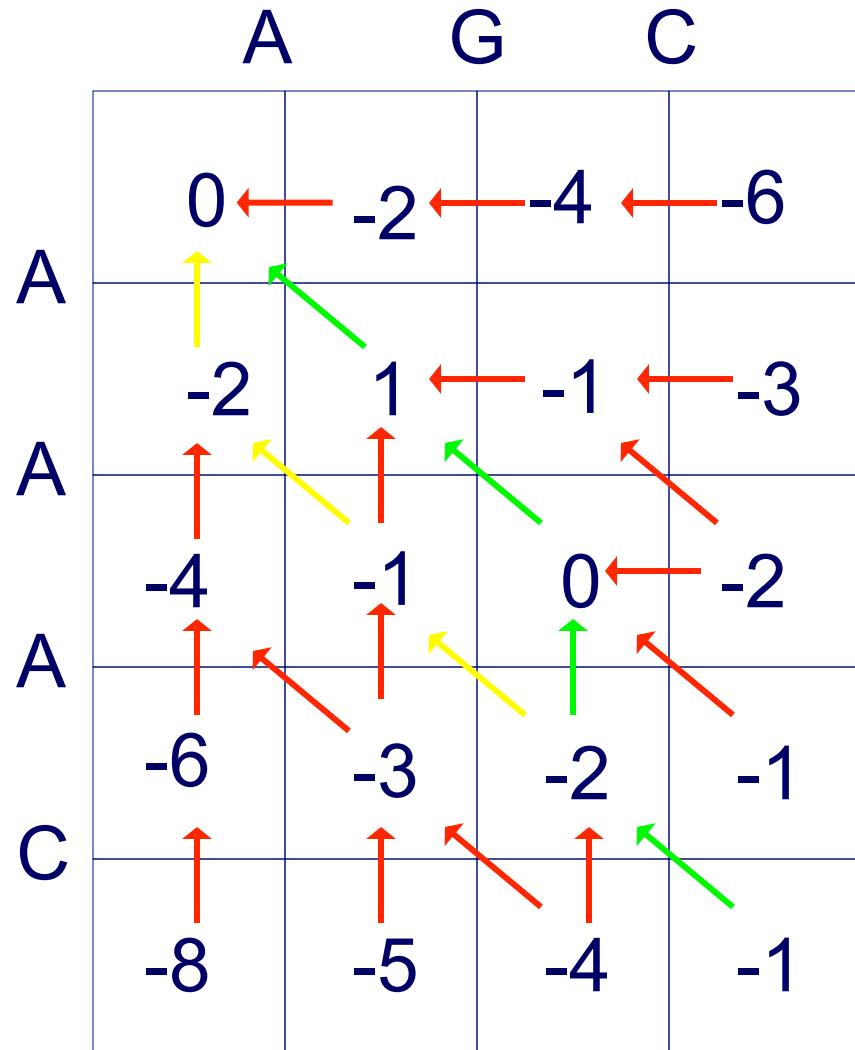
Equally Optimal Alignments

- many optimal alignments may exist for a given pair of sequences
- can use preference ordering over paths when doing traceback



- *highroad* and *lowroad* alignments show the two most different optimal alignments

Highroad & Lowroad Alignments



highroad alignment

x: A A A C
y: A G - C

lowroad alignment

x: A A A C
y: - A G C

Dynamic Programming Analysis

- recall, there are

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{2^{2n}}{\sqrt{\pi n}}$$

possible global alignments for 2 sequences of length n

- but the DP approach finds an optimal alignment efficiently

Computational Complexity

- initialization: $O(m)$, $O(n)$ where sequence lengths are m , n
- filling in rest of matrix: $O(mn)$
- traceback: $O(m + n)$
- hence, if sequences have nearly same length, the computational complexity is

$$O(n^2)$$

Local Alignment

- so far we have discussed *global alignment*, where we are looking for the best matching between sequences from one end to the other
- often, we will only want a *local alignment*, the best match between contiguous subsequences (substrings) of x and y

Local Alignment Motivation

- useful for comparing protein sequences that share a common *motif* (conserved pattern) or *domain* (independently folded unit) but differ elsewhere
- useful for comparing DNA sequences that share a similar *motif* but differ elsewhere
- useful for comparing protein sequences against *genomic DNA sequences* (long stretches of uncharacterized sequence)
- more sensitive when comparing highly diverged sequences

Local Alignment DP Algorithm

- original formulation: Smith & Waterman,
Journal of Molecular Biology, 1981
- interpretation of array values is somewhat different
 - $F(i, j)$ = score of the best alignment of a suffix of $x[1\dots i]$ and a suffix of $y[1\dots j]$

Local Alignment DP Algorithm

- the recurrence relation is slightly different than for global algorithm

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + S(x_i, y_j) \\ F(i-1, j) + s \\ F(i, j-1) + s \\ 0 \end{cases}$$


Local Alignment DP Algorithm

- initialization: first row and first column initialized with 0's
- traceback:
 - find maximum value of $F(i, j)$; can be anywhere in matrix
 - stop when we get to a cell with value 0

Local Alignment Example

A sequence logo showing the probability of each nucleotide (A, T, C, G) at each position. The x-axis represents positions 0, 1, 2, 3, 4, 5. The y-axis lists nucleotides A, T, C, G.

	A	A	G	A
T	0	0	0	0
T	0	0	0	0
A	0	0	0	0
A	0	1	1	0
A	0	1	2	0
G	0	0	0	3

Red arrows indicate transitions: from the G at position 3 to the G at position 4.

Match: +1

Mismatch: -1

Space: -2

x:	A	A	G
y:	A	A	G

More On Gap Penalty Functions

- a gap of length k is more probable than k gaps of length 1
 - a gap may be due to a single mutational event that inserted/deleted a stretch of characters
 - separated gaps are probably due to distinct mutational events
- a linear gap penalty function treats these cases the same
- it is more common to use gap penalty functions involving two terms
 - a penalty g associated with opening a gap
 - a smaller penalty s for extending the gap

Gap Penalty Functions

- linear

$$w(k) = sk$$

- affine

$$w(k) = \begin{cases} g + sk, & k \geq 1 \\ 0, & k = 0 \end{cases}$$

Dynamic Programming for the Affine Gap Penalty Case

- to do in $O(n^2)$ time, need 3 matrices instead of 1

$M(i, j)$ best score given that $x[i]$ is aligned to $y[j]$

$I_x(i, j)$ best score given that $x[i]$ is aligned to a gap

$I_y(i, j)$ best score given that $y[j]$ is aligned to a gap

Why Three Matrices Are Needed

- consider aligning the sequences **WFP** and **FW** using $g = -4$, $s = -1$ and the following values from the BLOSUM-62 substitution matrix:

$$\begin{array}{ll} S(F, W) = 1 & S(W, W) = 11 \\ S(F, F) = 6 & S(W, P) = -4 \\ S(F, P) = -4 & \end{array}$$

- the matrix shows the highest-scoring partial alignment for each pair of prefixes

	W	F	P	
F	0	-5	-6	-7
W	-5	1	1	-4
	-6	6	2	0

-WFP
FW--

optimal alignment

WF
FW

best alignment of these prefixes;
to get optimal alignment,
need to also remember

-WF
FW- ⁶⁴

Global Alignment DP for the Affine Gap Penalty Case

$$M(i, j) = \max \begin{cases} M(i - 1, j - 1) + S(x_i, y_j) \\ I_x(i - 1, j - 1) + S(x_i, y_j) \\ I_y(i - 1, j - 1) + S(x_i, y_j) \end{cases}$$

$$I_x(i, j) = \max \begin{cases} M(i - 1, j) + g + s \\ I_x(i - 1, j) + s \end{cases}$$

$$I_y(i, j) = \max \begin{cases} M(i, j - 1) + g + s \\ I_y(i, j - 1) + s \end{cases}$$

Global Alignment DP for the Affine Gap Penalty Case

- initialization

$$M(0,0) = 0$$

$$I_x(i,0) = g + s \times i$$

$$I_y(0,j) = g + s \times j$$

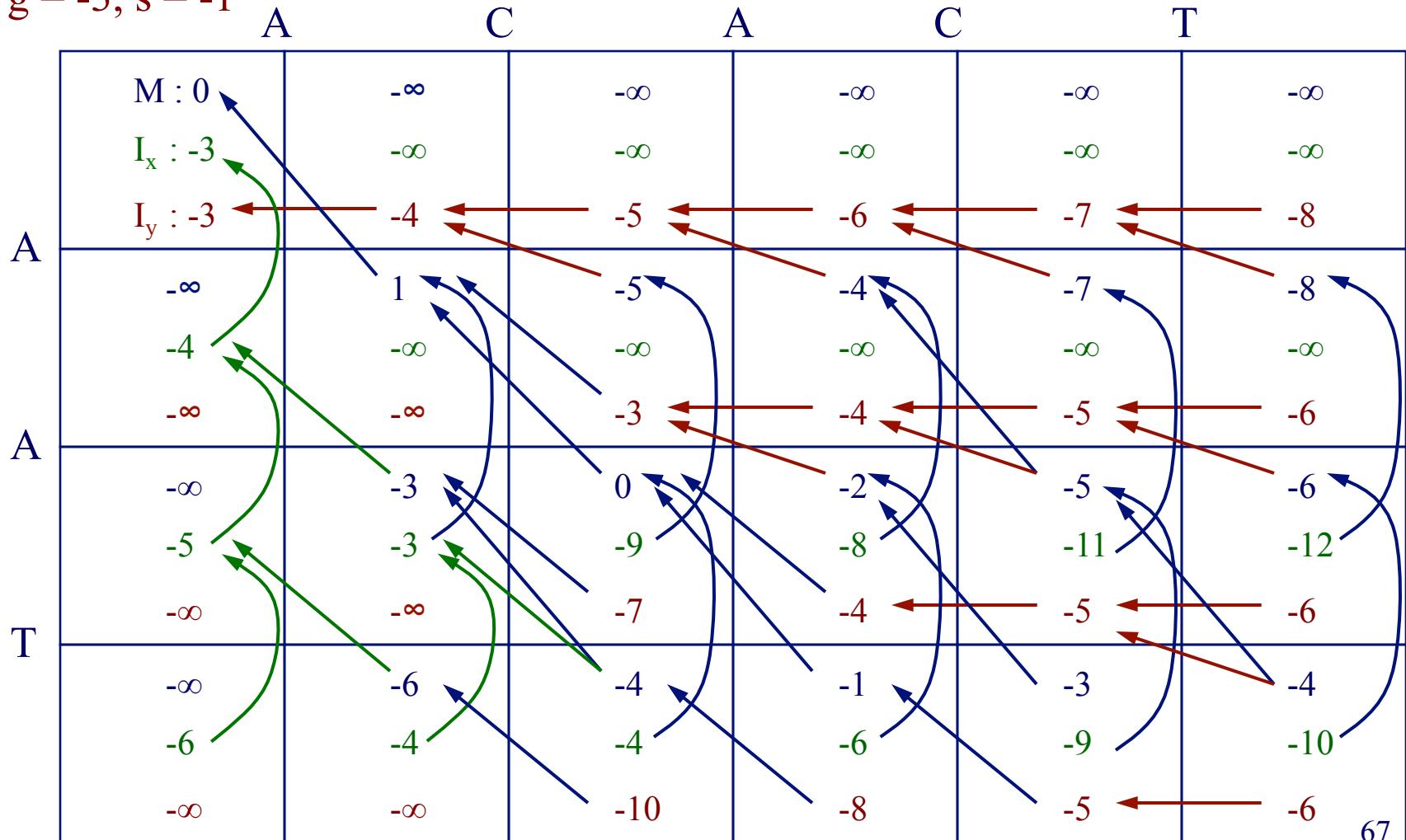
other cells in top row and leftmost column = $-\infty$

- traceback

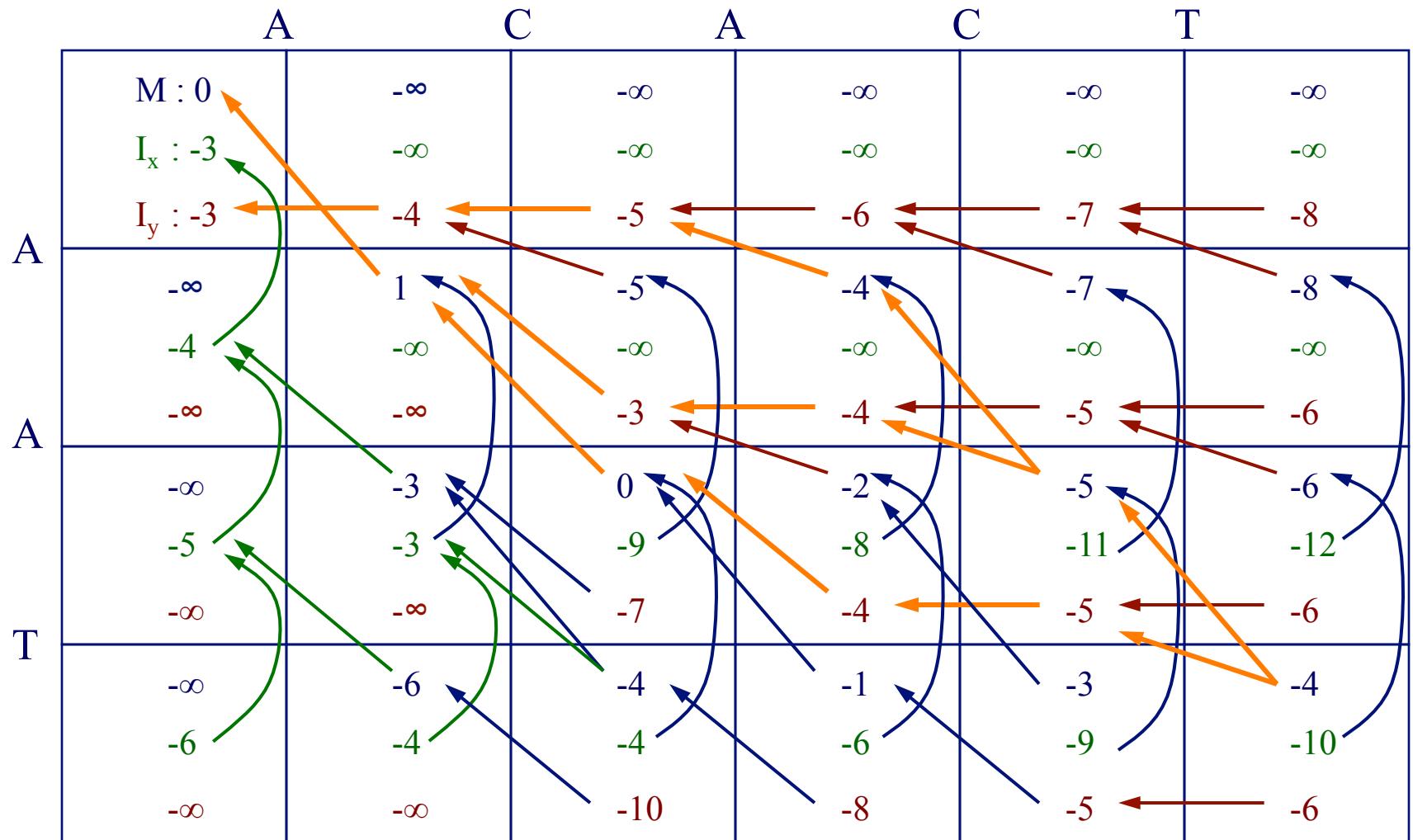
- start at largest of $M(m,n), I_x(m,n), I_y(m,n)$
- stop at any of $M(0,0), I_x(0,0), I_y(0,0)$
- note that pointers may traverse all three matrices

Global Alignment Example (Affine Gap Penalty)

$$g = -3, s = -1$$



Global Alignment Example (Continued)



three optimal alignments:

ACACT
AA--T

ACACT
A--AT

ACACT
--AAT⁶⁸

Local Alignment DP for the Affine Gap Penalty Case

$$M(i, j) = \max \begin{cases} M(i - 1, j - 1) + S(x_i, y_j) \\ I_x(i - 1, j - 1) + S(x_i, y_j) \\ I_y(i - 1, j - 1) + S(x_i, y_j) \\ 0 \end{cases}$$


$$I_x(i, j) = \max \begin{cases} M(i - 1, j) + g + s \\ I_x(i - 1, j) + s \end{cases}$$

$$I_y(i, j) = \max \begin{cases} M(i, j - 1) + g + s \\ I_y(i, j - 1) + s \end{cases}$$

Local Alignment DP for the Affine Gap Penalty Case

- initialization

$$M(0,0) = 0$$

$$M(i,0) = 0$$

$$M(0,j) = 0$$

cells in top row and leftmost column of $I_x, I_y = -\infty$

- traceback

- start at largest $M(i, j)$

- stop at $M(i, j) = 0$

Gap Penalty Functions

- linear:

$$w(k) = sk$$

- affine:

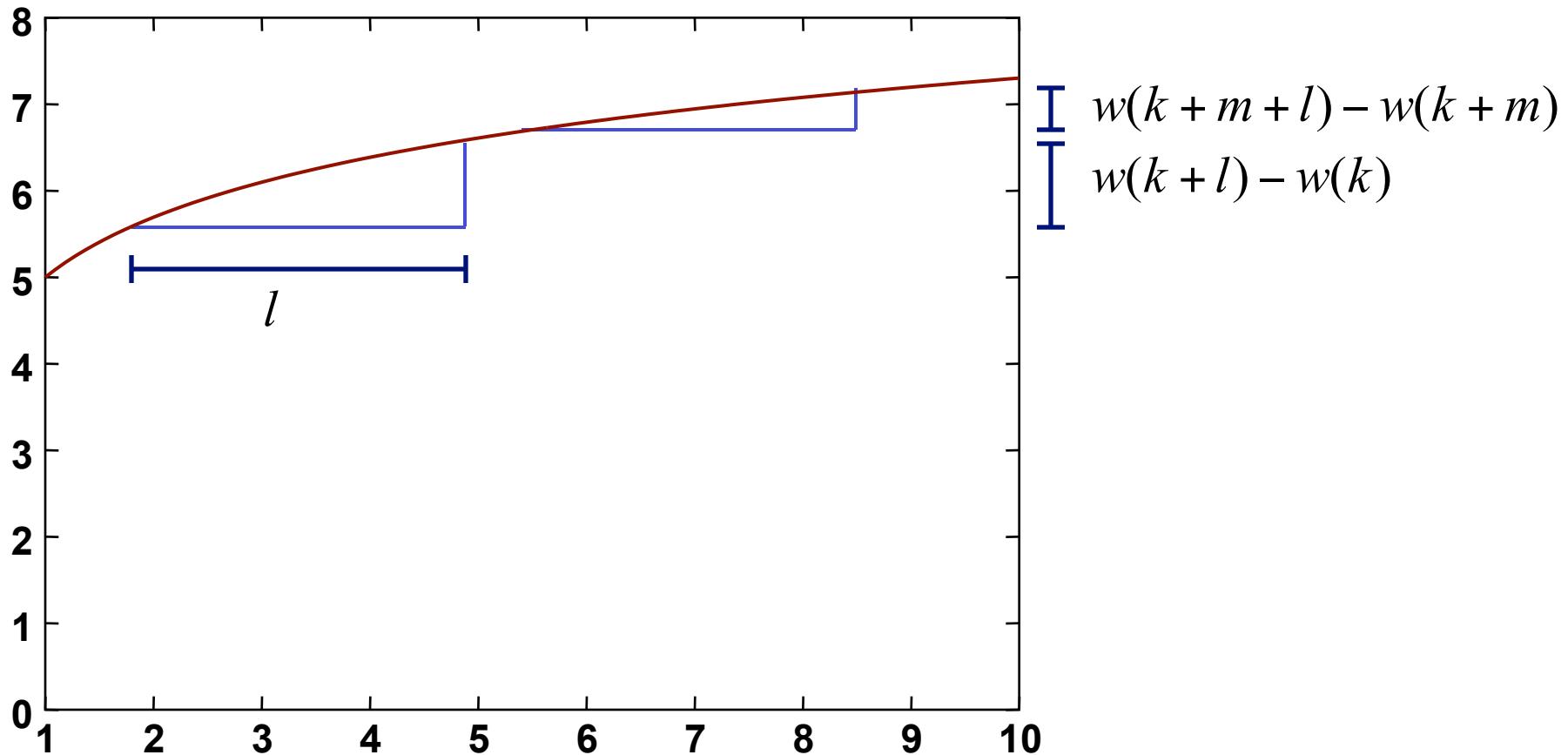
$$w(k) = \begin{cases} g + sk, & k \geq 1 \\ 0, & k = 0 \end{cases}$$

- concave: a function for which the following holds for all $k, l, m \geq 0$

$$w(k + m + l) - w(k + m) \leq w(k + l) - w(k)$$

e.g. $w(k) = g + s \times \log(k)$

Concave Gap Penalty Functions



$$w(k+m+l) - w(k+m) \leq w(k+l) - w(k)$$

Computational Complexity and Gap Penalty Functions

- linear: $O(n^2)$
- affine: $O(n^2)$
- concave $O(n^2)$
- general: $O(n^3)$

* assuming two sequences of length n

Alignment (Global) with General Gap Penalty Function

$$F(i, j) = \max \begin{cases} F(i - 1, j - 1) + S(x_i, y_j) \\ F(k, j) + \gamma(i - k) \\ F(i, k) + \gamma(j - k) \end{cases}$$

consider every previous element in the row

consider every previous element in the column

```
graph TD; A["F(i-1, j-1) + S(xi, yj)"] --> B["F(k, j) + γ(i-k)"]; B --> C["F(i, k) + γ(j-k)"]
```