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

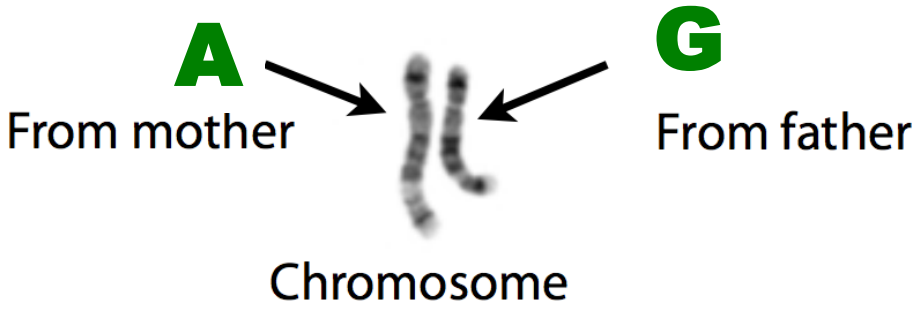
Genotype is all the inherited information

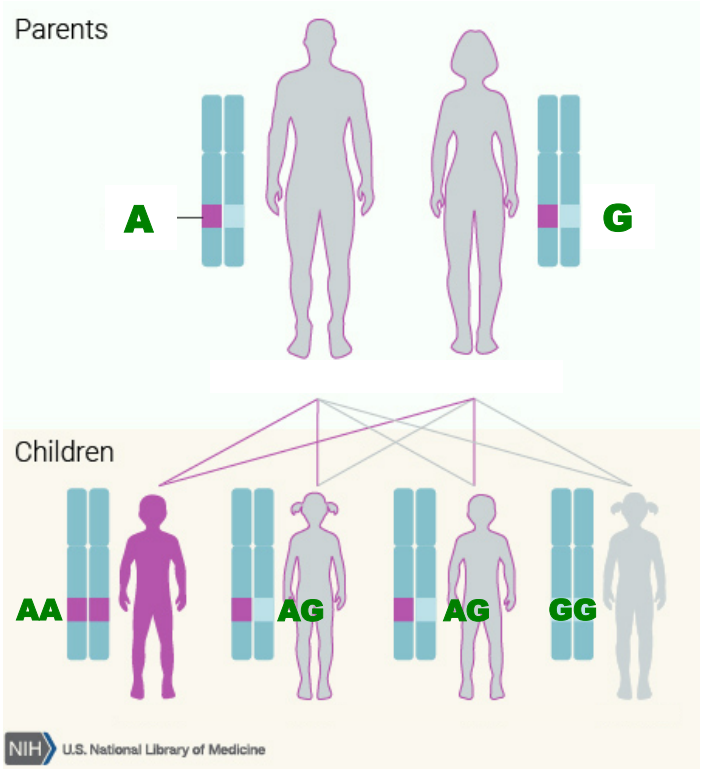
Phenotype is something we observe: eye color

Mother **ACCTCTATTCA**

Father **ACGTCTGTCCA**

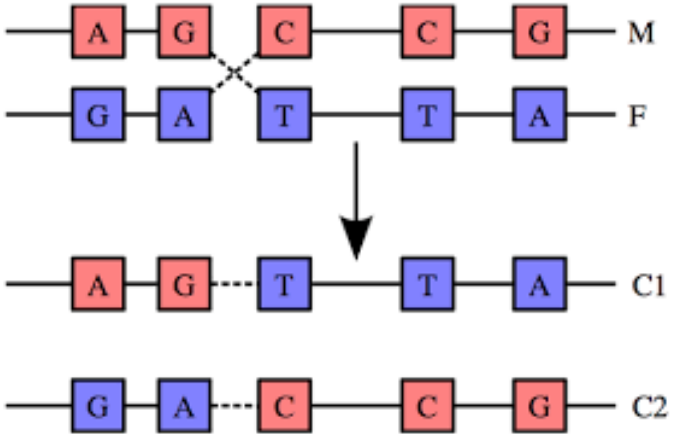
A and G are alleles, the variable site is in a gene



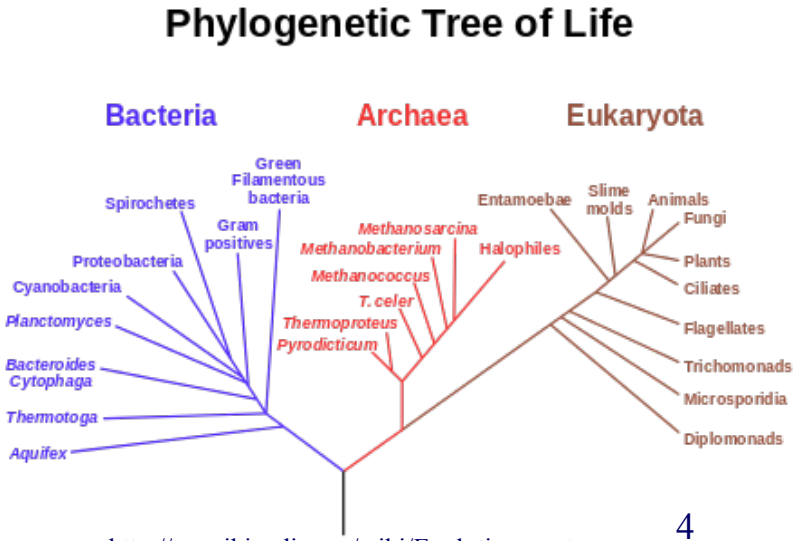


Evolution: select these genotypes

- 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



http://en.wikipedia.org/wiki/Genetic_recombination

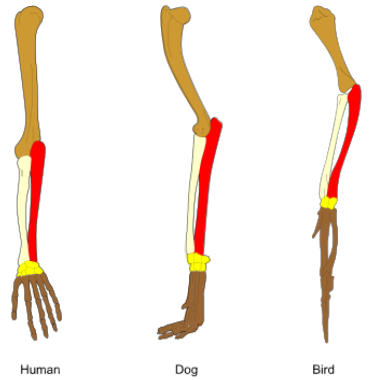


http://en.wikipedia.org/wiki/Evolutionary_tree

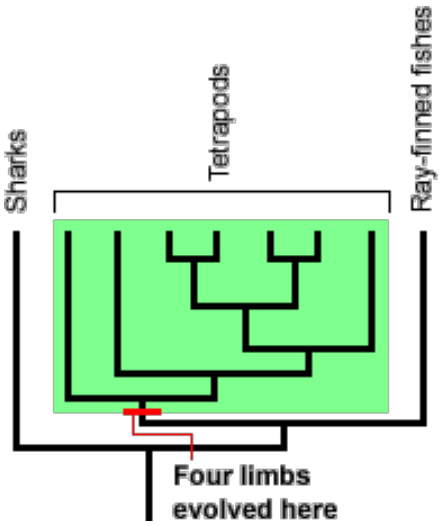
Slide courtesy of Ben Langmead

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

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

...CATCGATGACTATCCG...
ATGACTGT

suffix trees, locality-sensitive hashing,...

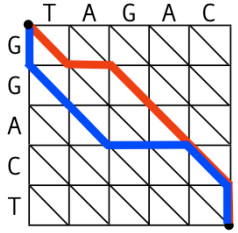
Database searching

CATGCTTGCTGGCGTAAA →

CATGCATGCTGCGTAC
CATGGTTGCTCACAAGTAC
CATGCCTGCTGCGTAA
TACGTGCCTGACCTGCGTAC
CATGCCGAATGCTG

BLAST

Optimization problem



Needleman-Wunsch, Smith-Waterman,...

Statistical problem

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

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

DNA sequence edits

- Substitutions: A**C**GA → A**G**GA
- Insertions: ACGA → AC**CGGA**GA
- Deletions: A**CGGA**GA → AGA
- Transpositions: A**CGG**AGA → AAG**CGG**A
- Inversions: AC**GGA**GA → AC**TCC**GA

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?

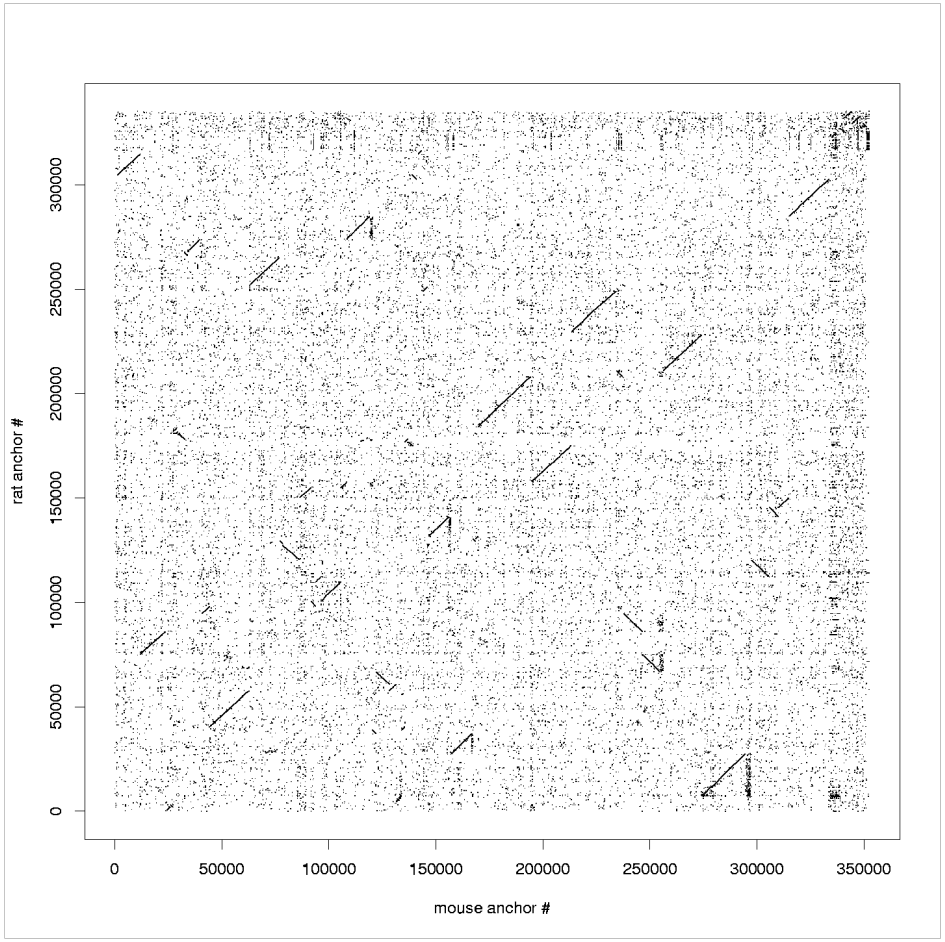
- We will focus on *evolutionary* alignment
- matching of *homologous* positions in two sequences
- positions with no homologous pair are matched with a *space* '–'
- A group of consecutive spaces is a *gap*

```
CA--GATTCGAAT
CGCCGATT---AT
```

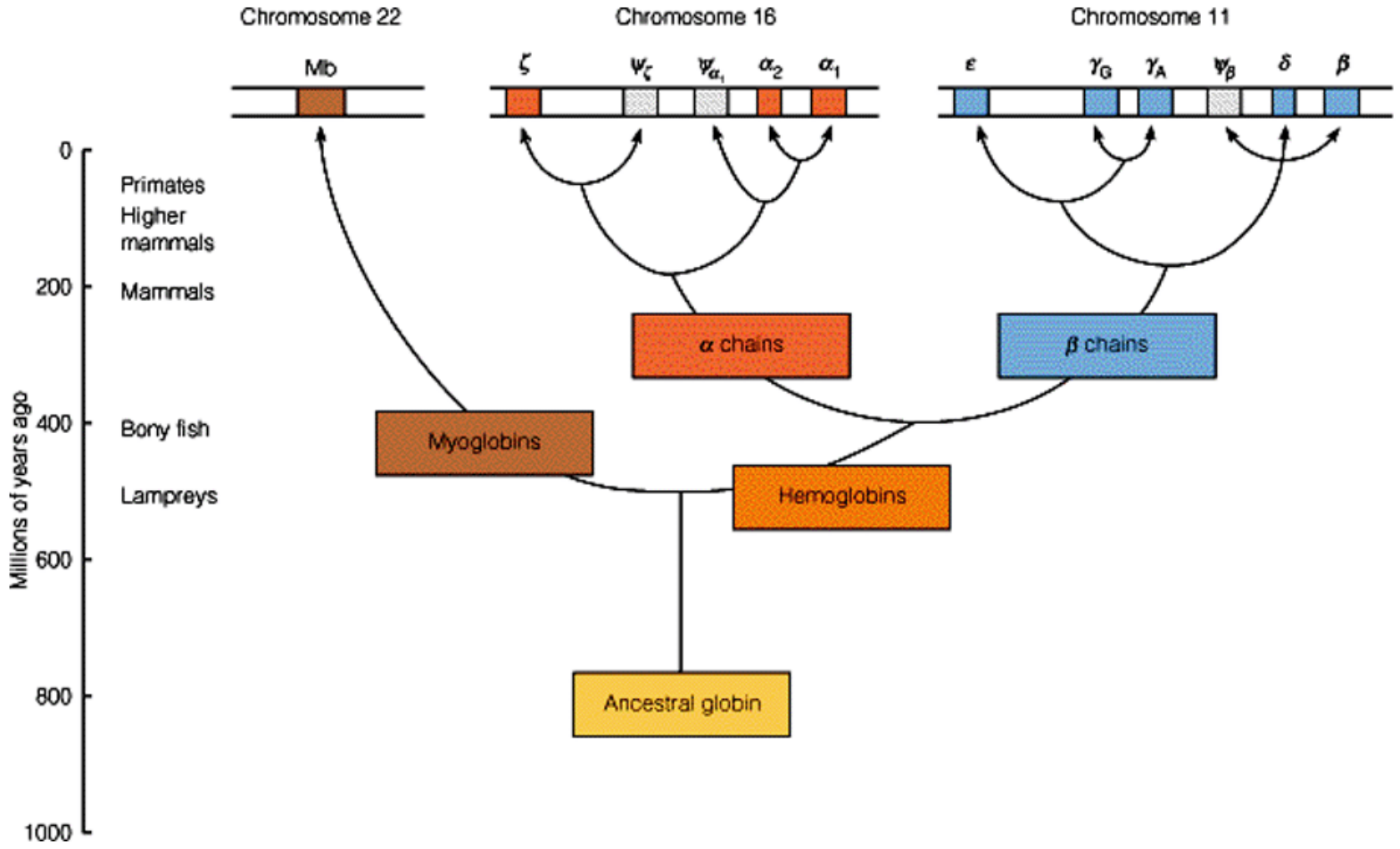
gap

Dot plots

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



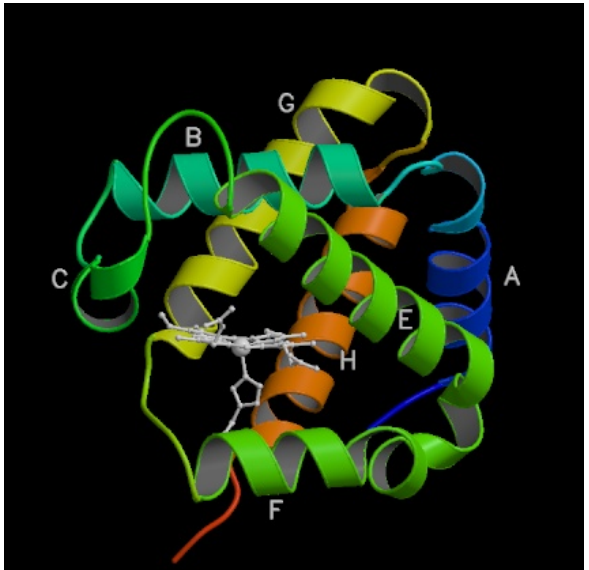
Homology Example: Evolution of the Globins



Example Alignment: Globins

- figure at right shows prototypical structure of globins

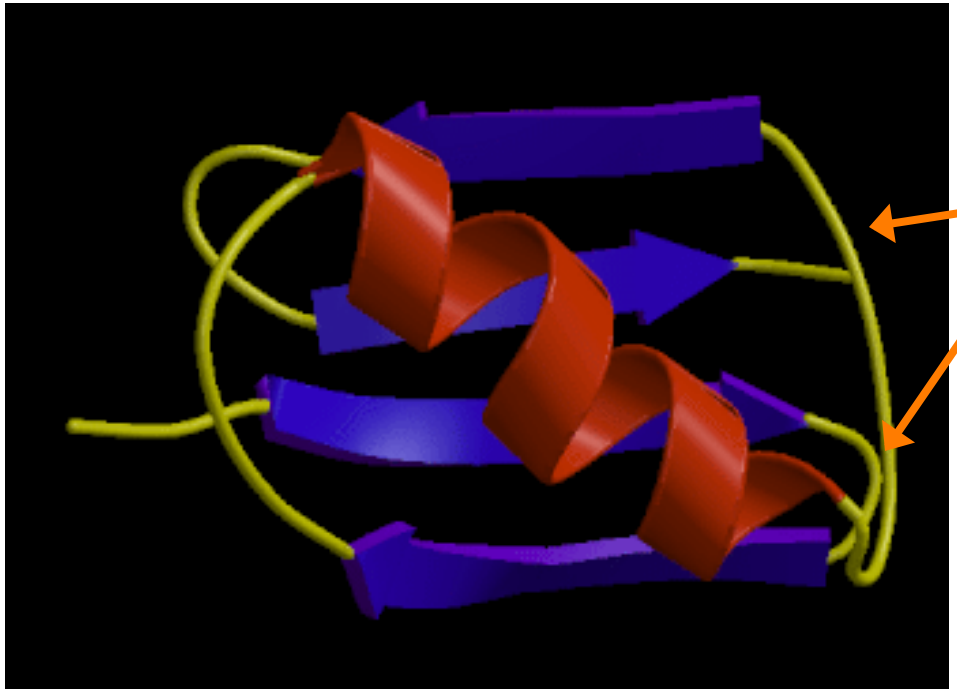
- figure below shows part of alignment for 8 globins (-'s indicate gaps)




```
                A0   A4   A8   A12       B1        B6        B14  C2        CD1 CD4
                ↓    ↓    ↓    ↓         ↓         ↓         ↓    ↓         ↓   ↓
Hb_a  ----------VLSPADKTNVKAAWGKVGA----HAGEYGAEALERMFLSFPTTKTYFPHF
Hb_b  ---------VHLTPEEKSAVTALWGKV------NVDEVGGEALGRLLVVYPWTQRFFESF
Mb_SW ----------VLSEGEWQLVLHVWAKVEA----DVAGHGQDILIRLFKSHPETLEKFDRF
LegHb ---------GALTESQAALVKSSWEEFNA----NIPKHTHRFFILVLEIAPAAKDLFSFL
BacHb --------LDQQTINIIKATVPVLKEHG------V-TITTTFYKNLFAKHPEVRPLF---
SeaHb GGTLAIQAQGDLTLAQKKIVRKTWHQLMR----NKTSFVTDVFIRIFAYDPSAQNKFPQM
AscHb -----------ANKTRELCMKSLEHAKVDTSNEARQDGIDLYKHMFENYPPLRKYFKS-
Eryt. ----------LSADQISTVQASFDKVKG---------DPVGILYAVFKADPSIMAKFTQF
```

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

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

Slides courtesy of Compeau and Pevzner

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
```

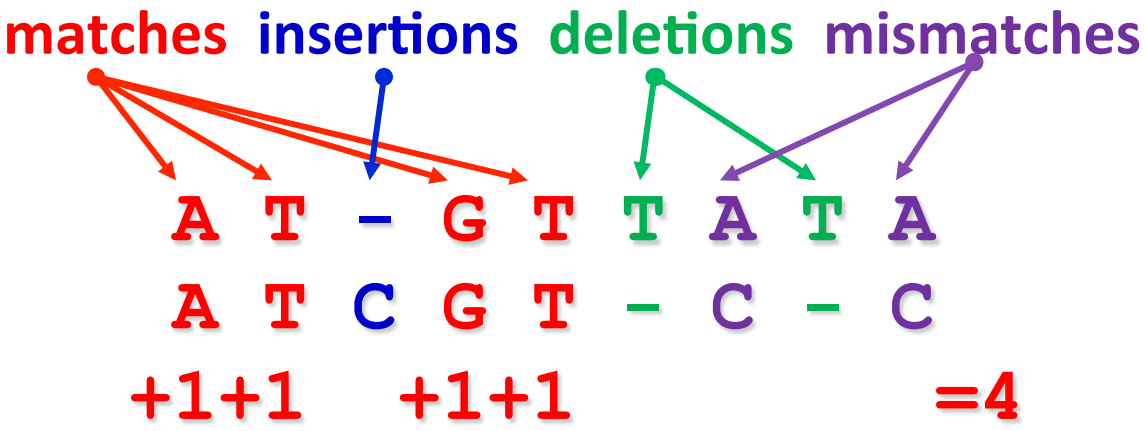
The Alignment Task

```
 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 "-"

Slides courtesy of Compeau and Pevzner

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.

Slides courtesy of Compeau and Pevzner

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) + 3*s* + S(D,G) + 2s …

The Space of Global Alignments

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

```
ELV     -ELV     --ELV     ELV-
VIS     VIS-     VIS--     -VIS

E-LV     ELV--     EL-V
VIS-     --VIS     -VIS
```

Number of Possible Alignments

- given sequences of length m and n
- assume we don't count as distinct

```
C-      and   -C
-G            G-
```

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

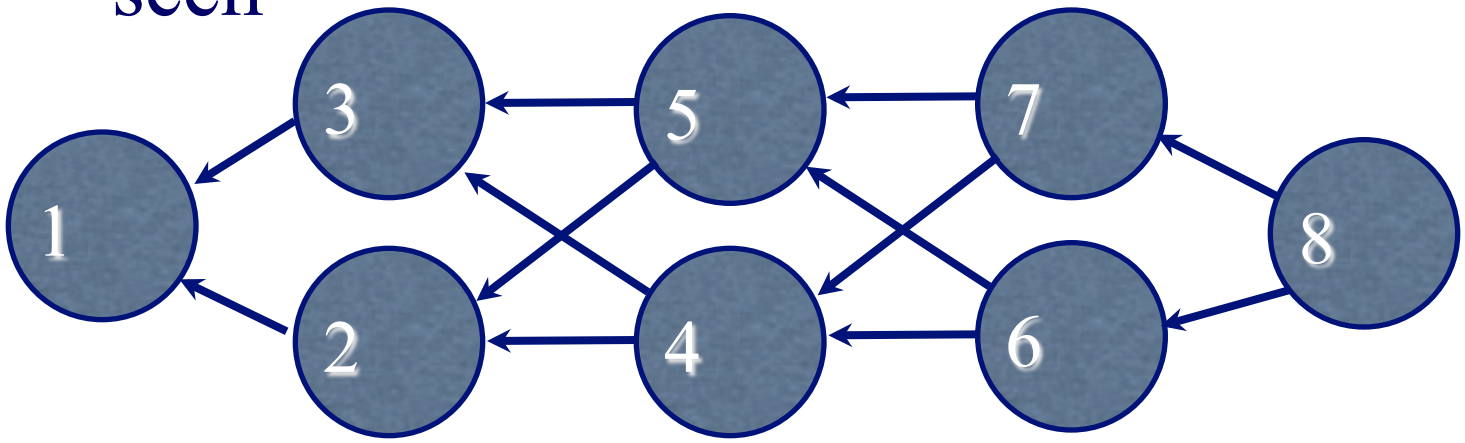
  possible global alignments for 2 sequences of length n
  - e.g. two sequences of length 100 have $\approx 10^{59}$ possible alignments
  - but we can use *dynamic programming* to find an optimal alignment efficiently

Dynamic Programming

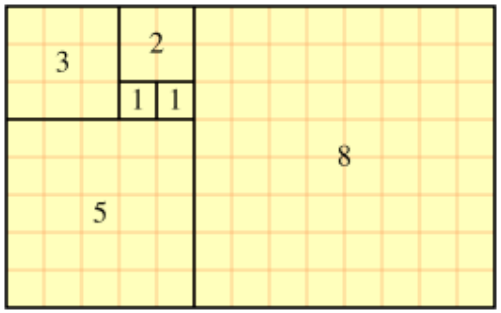
- Algorithmic technique for optimization problems that have two properties:
  - *Optimal substructure*: Optimal solution can be computed from optimal solutions to subproblems
  - *Overlapping subproblems*: Subproblems overlap such that the total number of distinct subproblems to be solved is relatively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

Dynamic Programming

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

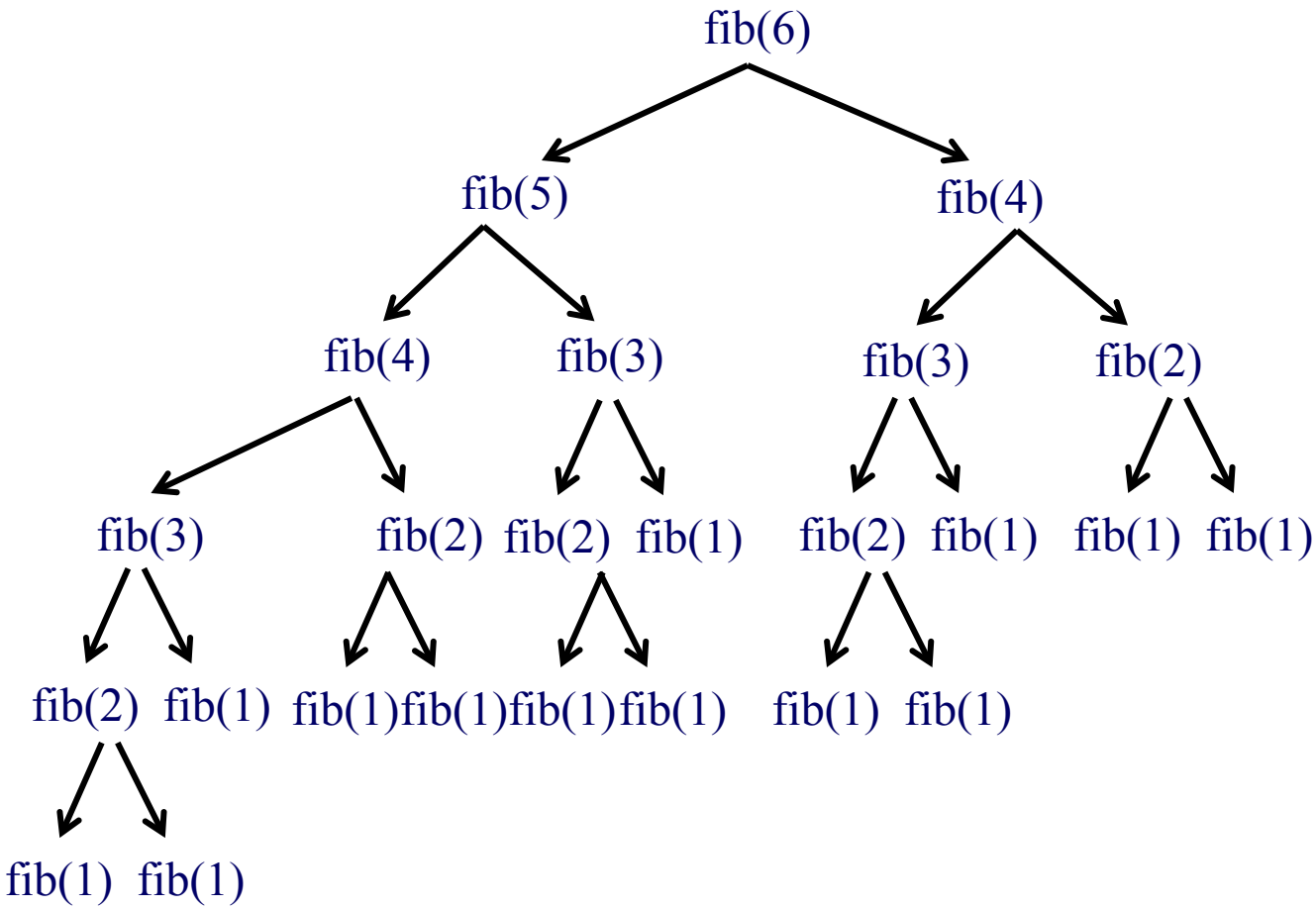


Fibonacci example



- 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 execution stack

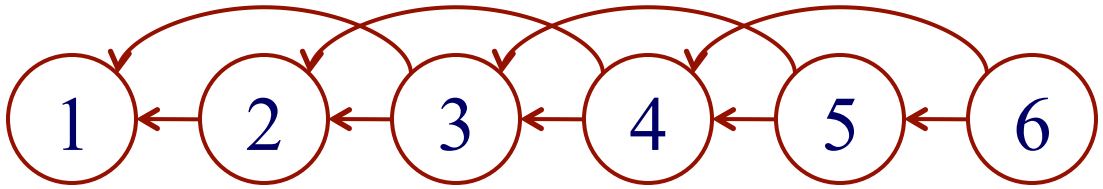


Fibonacci dynamic programming

- 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 fib(6)

Fibonacci DP

- Memoization (top-down approach)

```
fib_dict = {}
def fib(n):
    if n is fib_dict:
        return fib_dict[n]
    else:
        fib_dict[n] = n if n < 3 else fib(n-2) + fib(n-1)
    return fib_dict[n]
```

- Solve subproblems from smallest to largest (bottom-up approach)

```
def fib(n):
    a = [1,1,1]   // a[0]=a[1]=a[2]=1
    for i in range (3,n):
        a[i] = a [i-2] + a[i-1]
    return a[n]
```

Why “Dynamic Programming”?

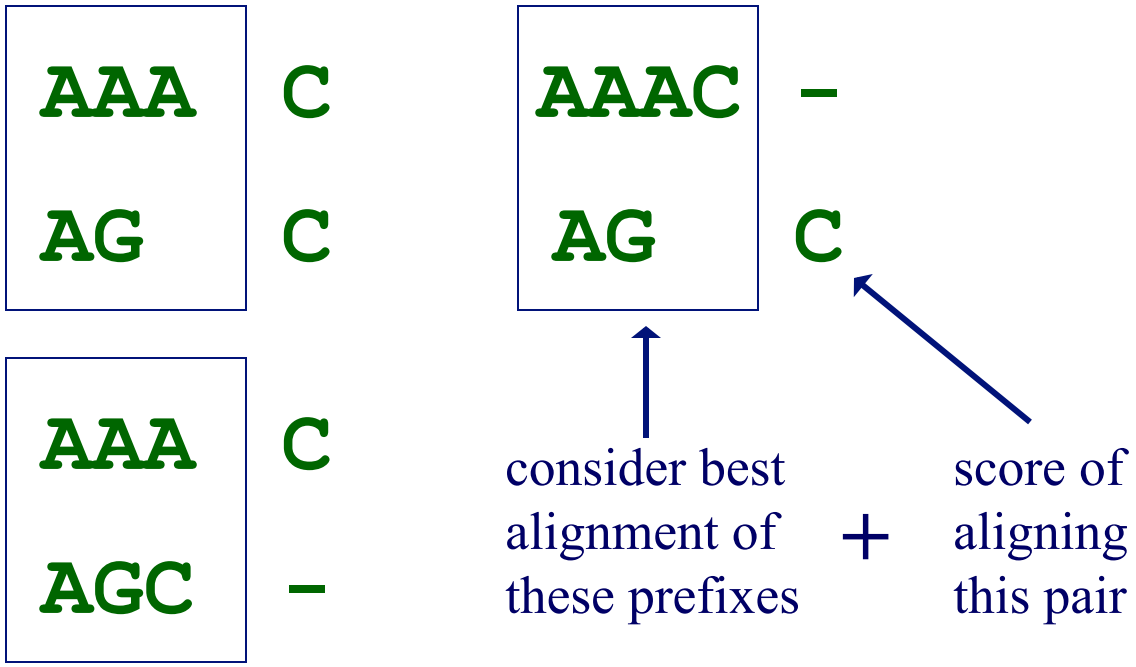
- Coined by Richard Bellman in 1950 while working at RAND
- Government officials were overseeing RAND, disliked research and mathematics
- “programming”: planning, decision making (optimization)
- “dynamic”: multistage, time varying
- “It was something not even a Congressman could object to. So I used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

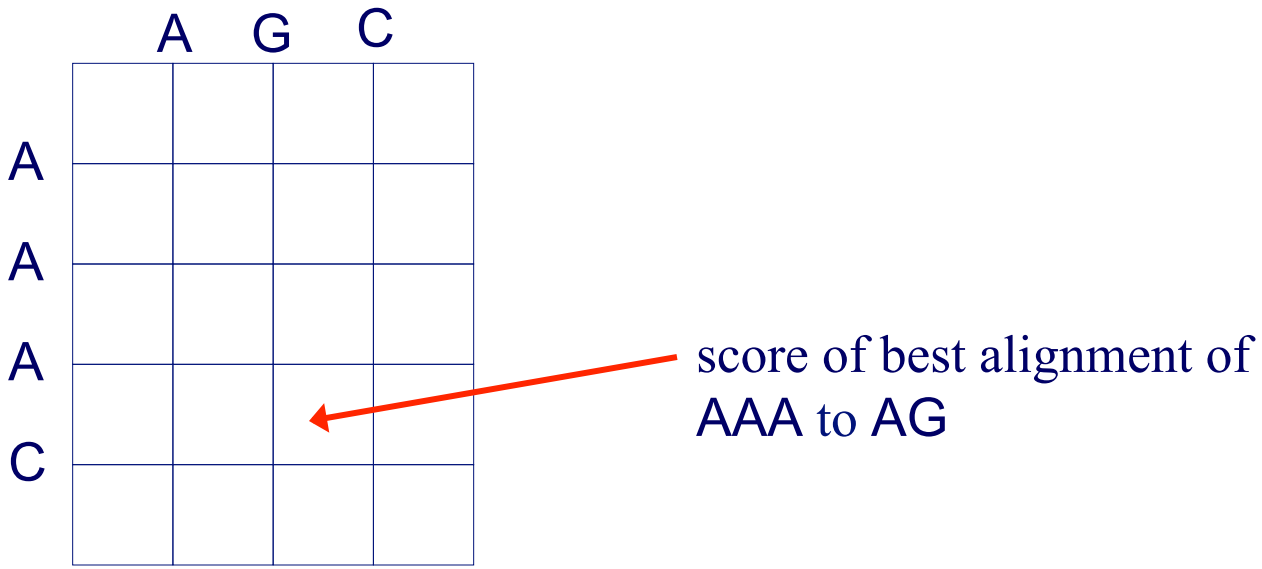
- Subproblem: *F(i,j)* = score of best alignment of the length *i* prefix of *x* and the length *j* prefix of *y*.

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + S(x_i, y_j) & \text{Match } x_i \text{ with } y_j \\ F(i-1,j) + s & \text{Insertion in x} \\ F(i,j-1) + s & \text{Insertion in y} \end{cases}$$

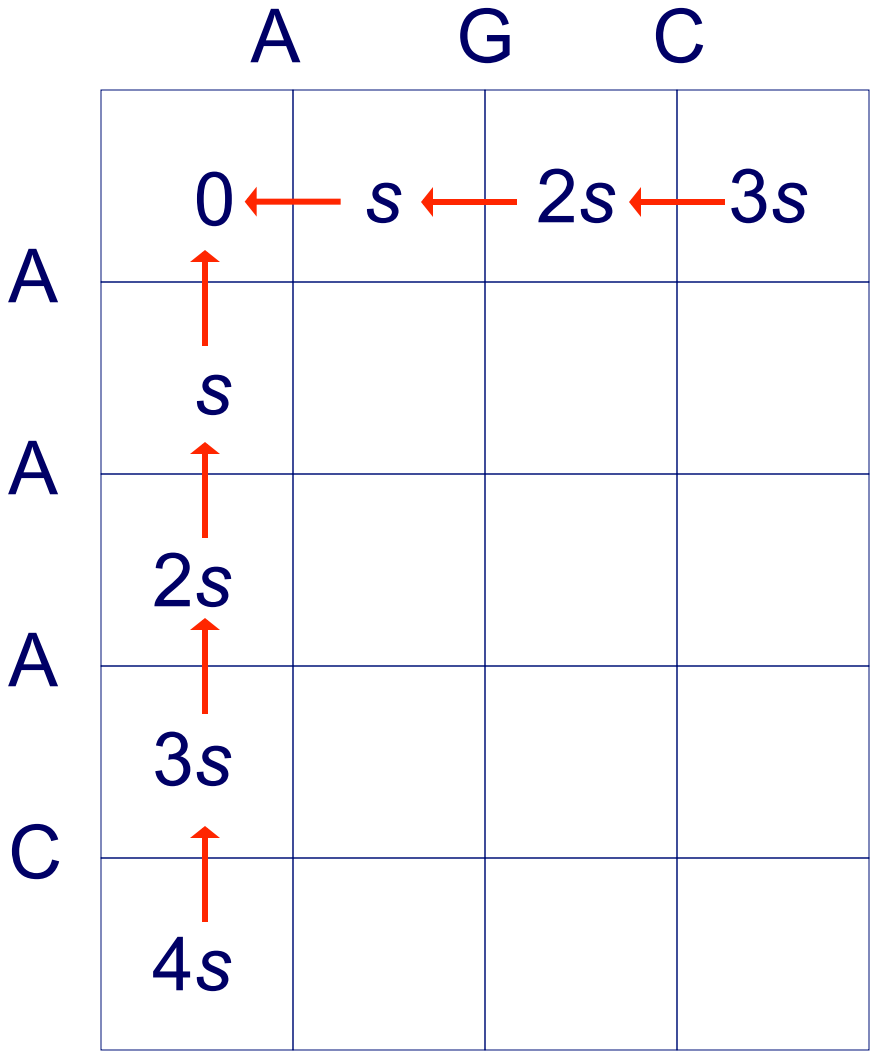
Dynamic Programming Implementation

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

$$x[1 \ldots i] \text{ with } y[1 \ldots j]$$



Initializing Matrix: Global Alignment with Linear Gap Penalty



s = gap penalty
k = length of gap

Initialize matrix:
F(0,0) = 0
F(i,0) = sk
F(0,j) = sk

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:

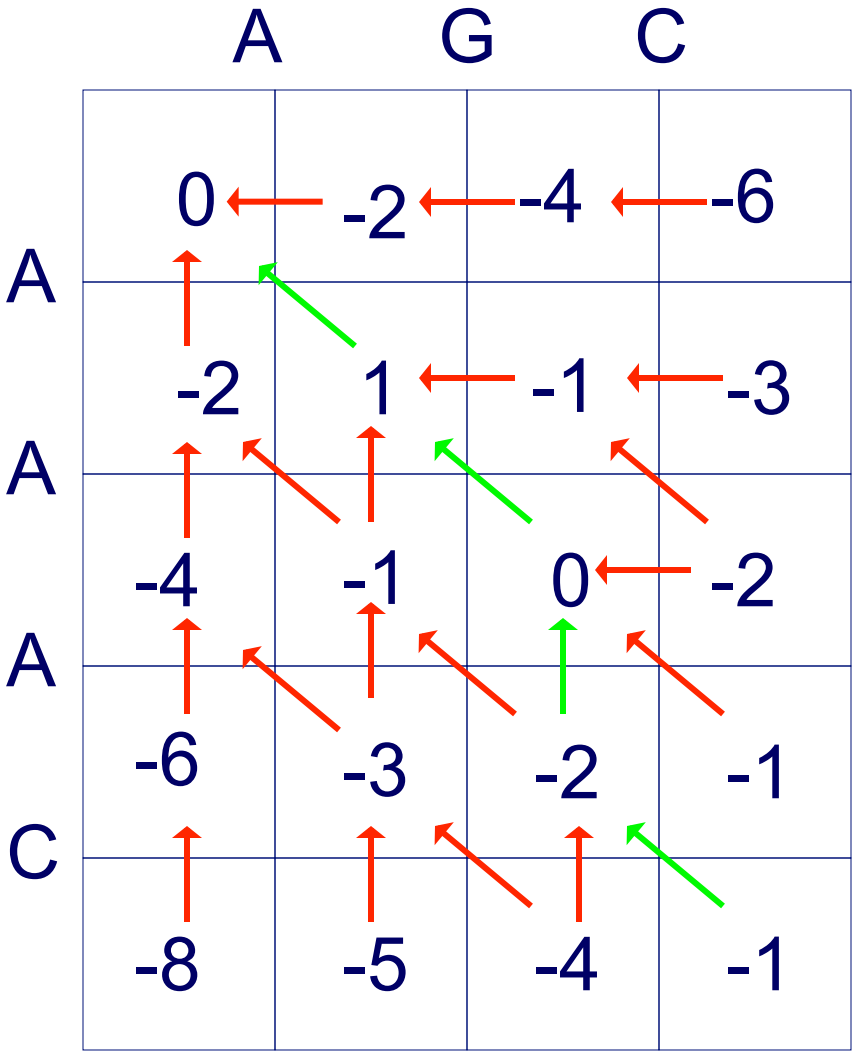
$S(x_i, y_i) =$

+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

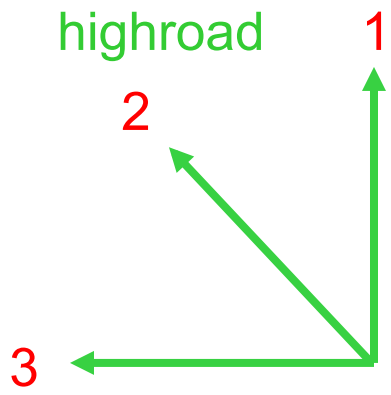
but there are three optimal alignments here (can you find them?)

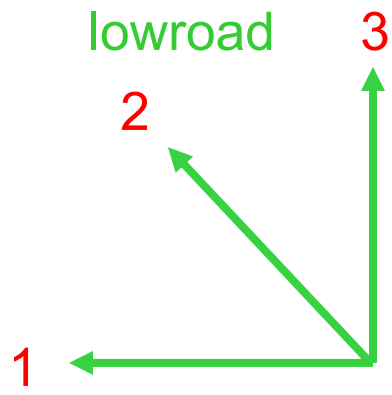
DP Comments

- 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

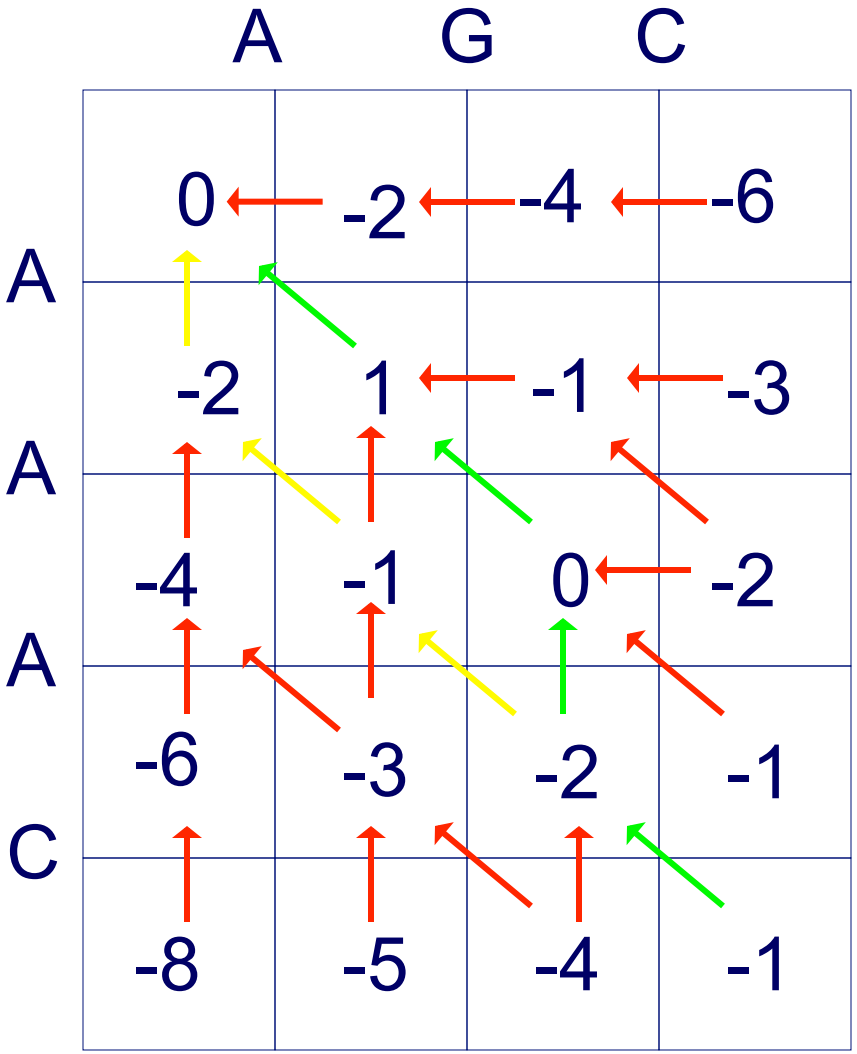
- many optimal alignments may exist for a given pair of sequences
- can use preference ordering over paths when doing traceback





- *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 $x[1 \ldots i]$ and a suffix of $y[1 \ldots j]$

Local Alignment DP Algorithm

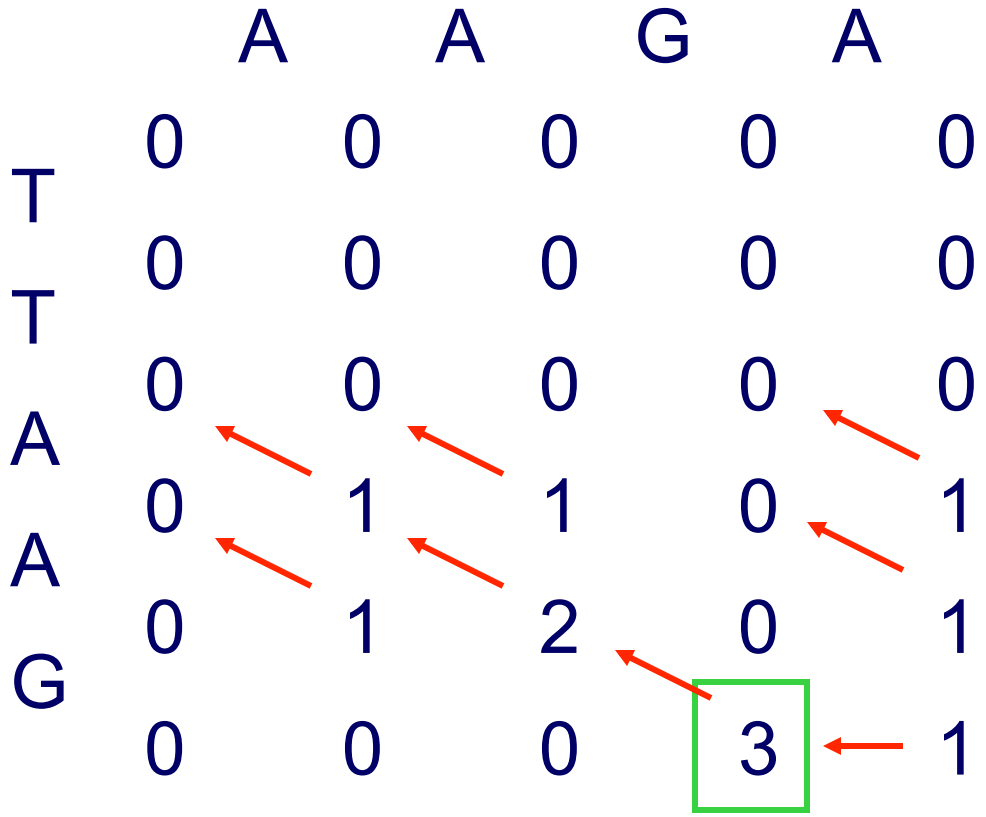
- the recurrence relation is slightly different than for global algorithm

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

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



x: A A G

y: A A G

Match: +1

Mismatch: -1

Space: -2

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

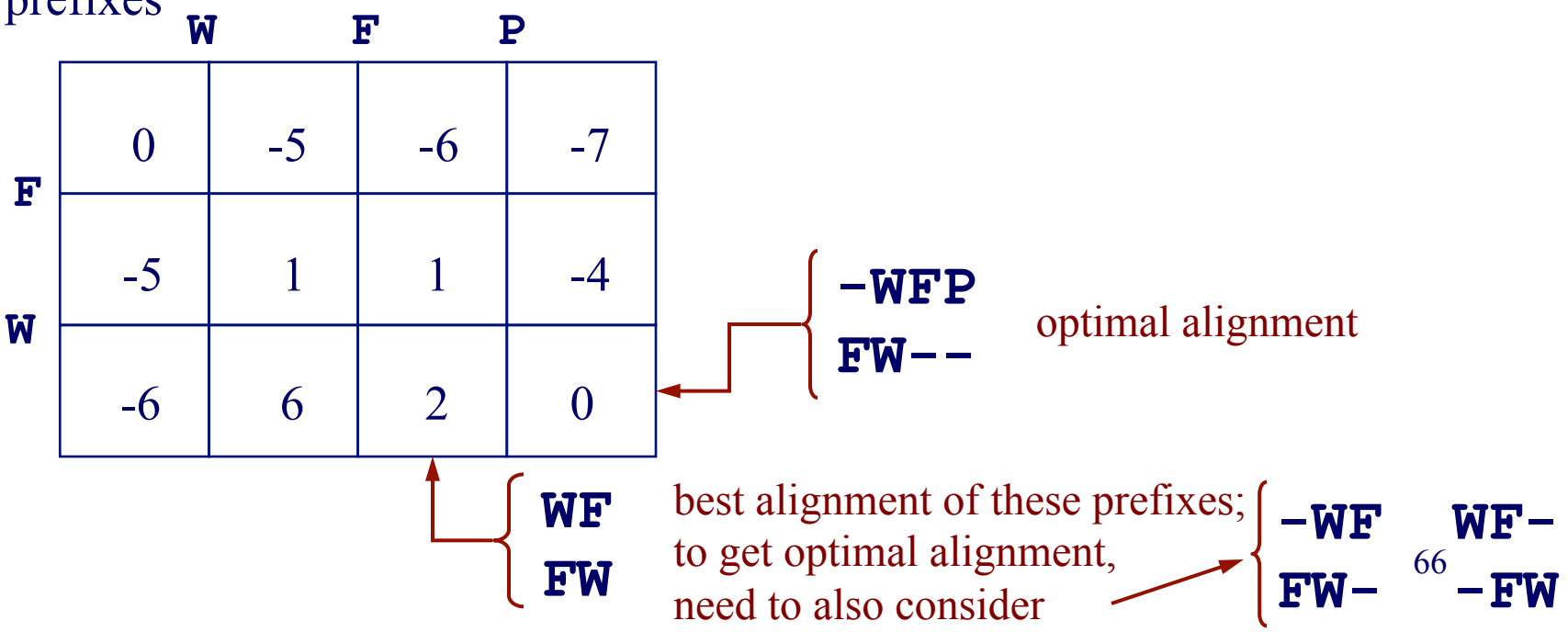
$I_y(i, j)$ best score given that $y[j]$ is aligned to a gap

Why Three Matrices Are Needed

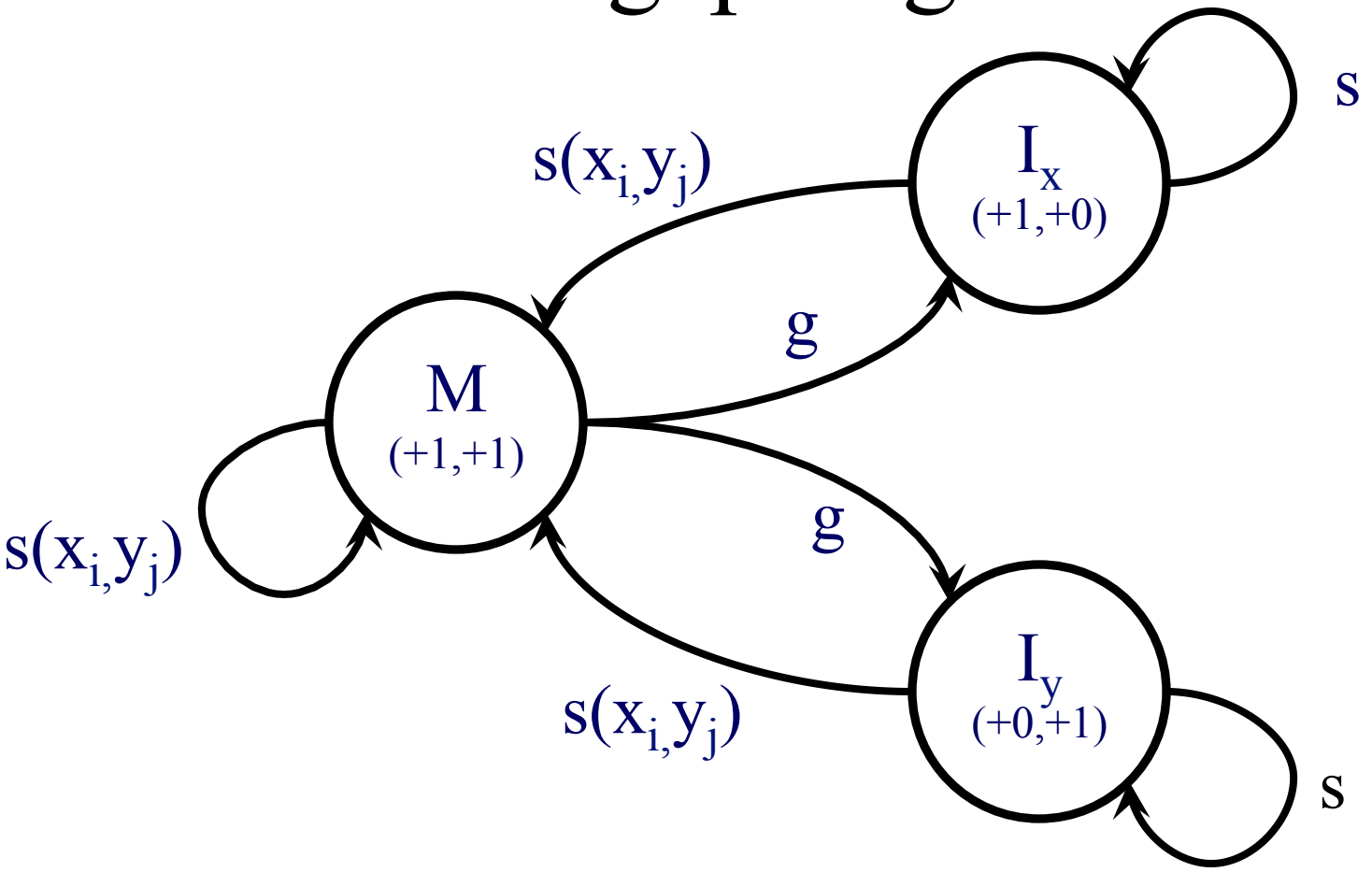
- consider aligning the sequences **WFP** and **FW** using g = -4, s = -1 and the following values from the BLOSUM-62 substitution matrix:

  S(**F**, **W**) = 1 S(**W**, **W**) = 11
  S(**F**, **F**) = 6 S(**W**, **P**) = -4
  S(**F**, **P**) = -4

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



Relationship of three states for affine gap alignment



Global Alignment DP for the Affine Gap Penalty Case

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

Match x_i with y_j

Insertion in x

Insertion in y

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

Open gap in x

Extend gap in x

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

Open gap in y

Extend gap in y

Global Alignment DP for the Affine Gap Penalty Case

- initialization

  $M(0,0) = 0$

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

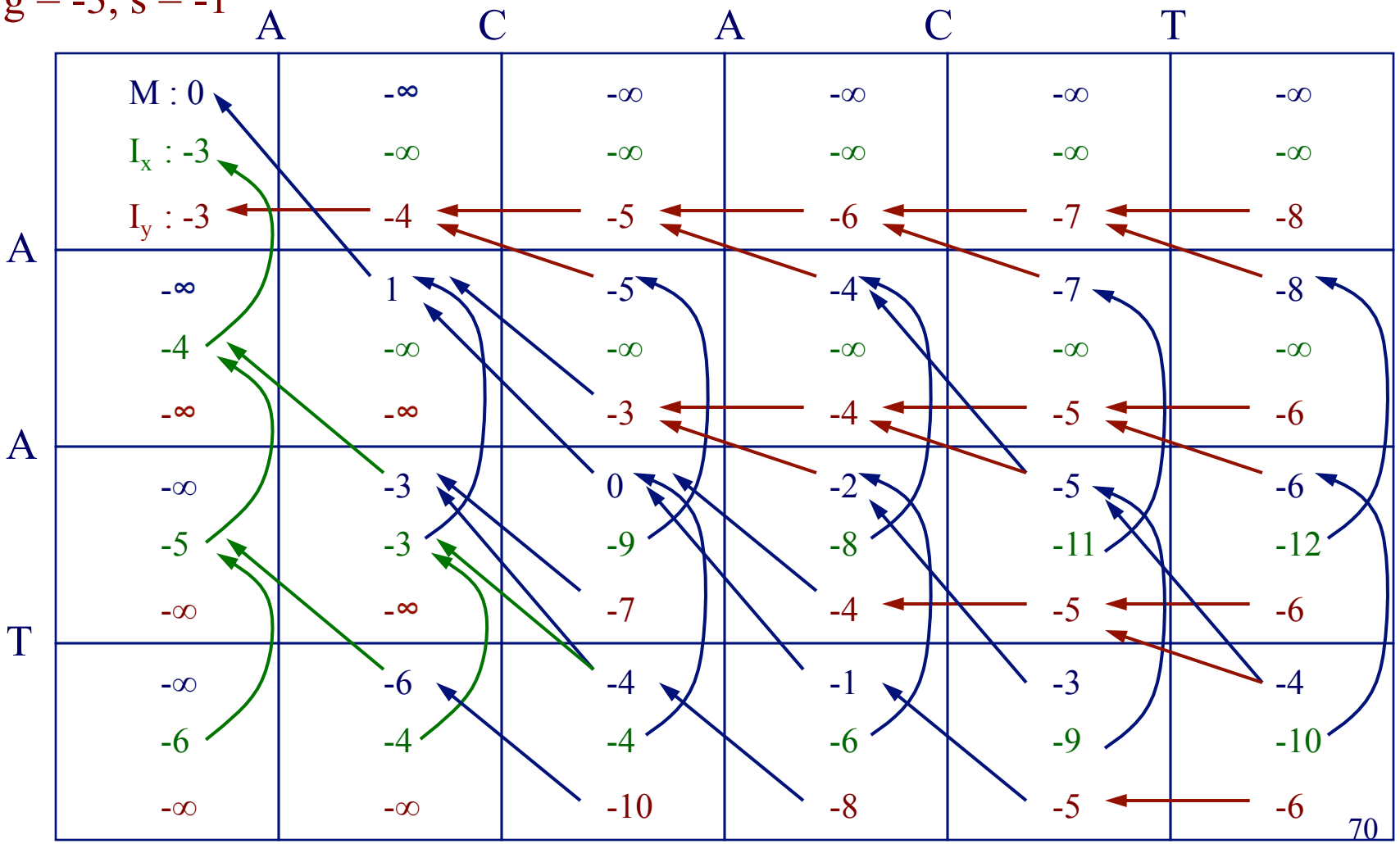
  $I_y(0,j) = g + s \times j$

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

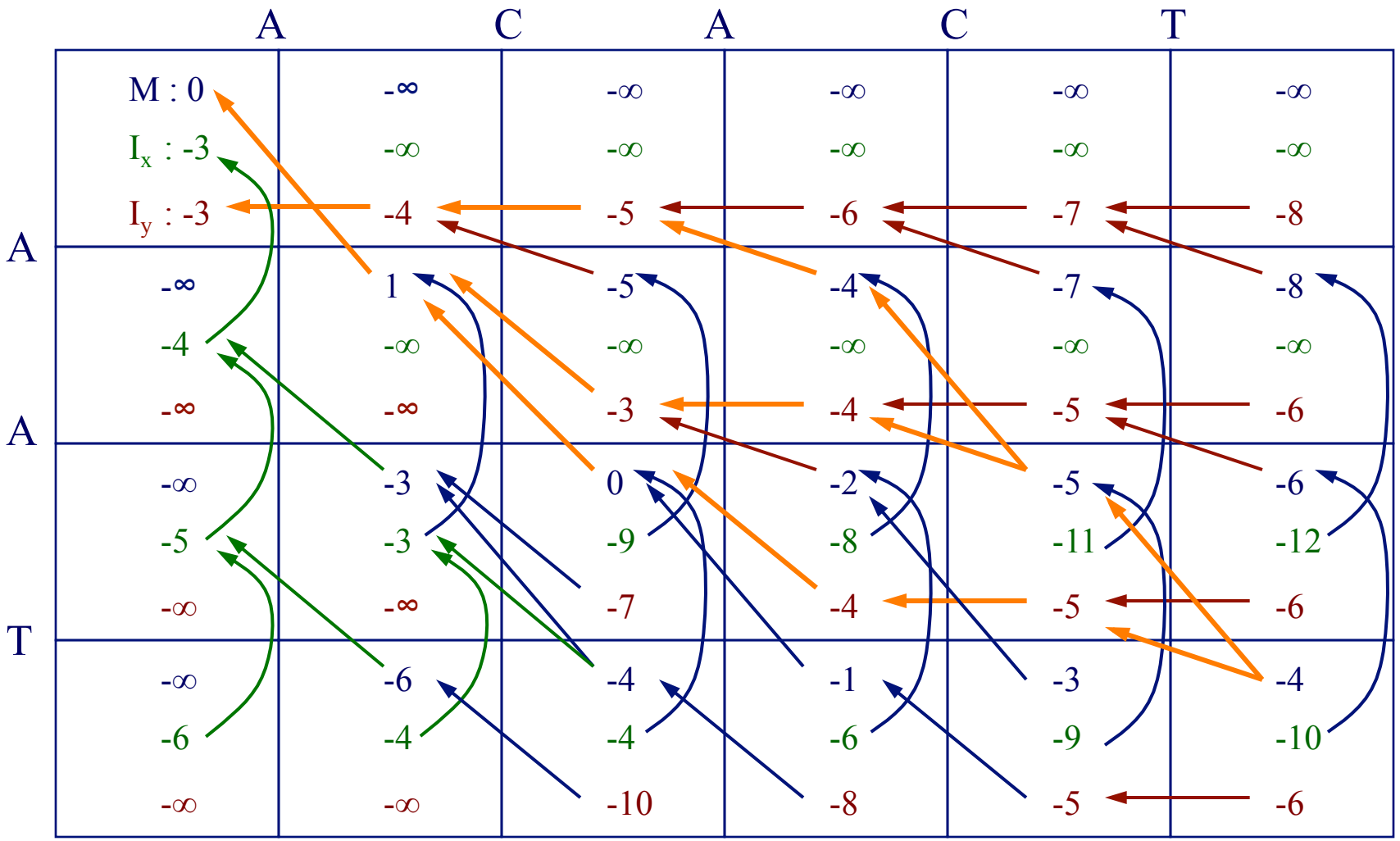
- traceback
  - start at largest of $M(m,n), I_x(m,n), I_y(m,n)$
  - stop at any of $M(0,0), I_x(0,0), I_y(0,0)$
  - note that pointers may traverse all three matrices

Global Alignment Example
(Affine Gap Penalty)

Match: +1
Mismatch: -1
g = -3, s = -1



Global Alignment Example (Continued)



three optimal alignments:

```
ACACT   ACACT   ACACT
AA--T   A--AT   --AAT
```

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: $w(k) = sk$

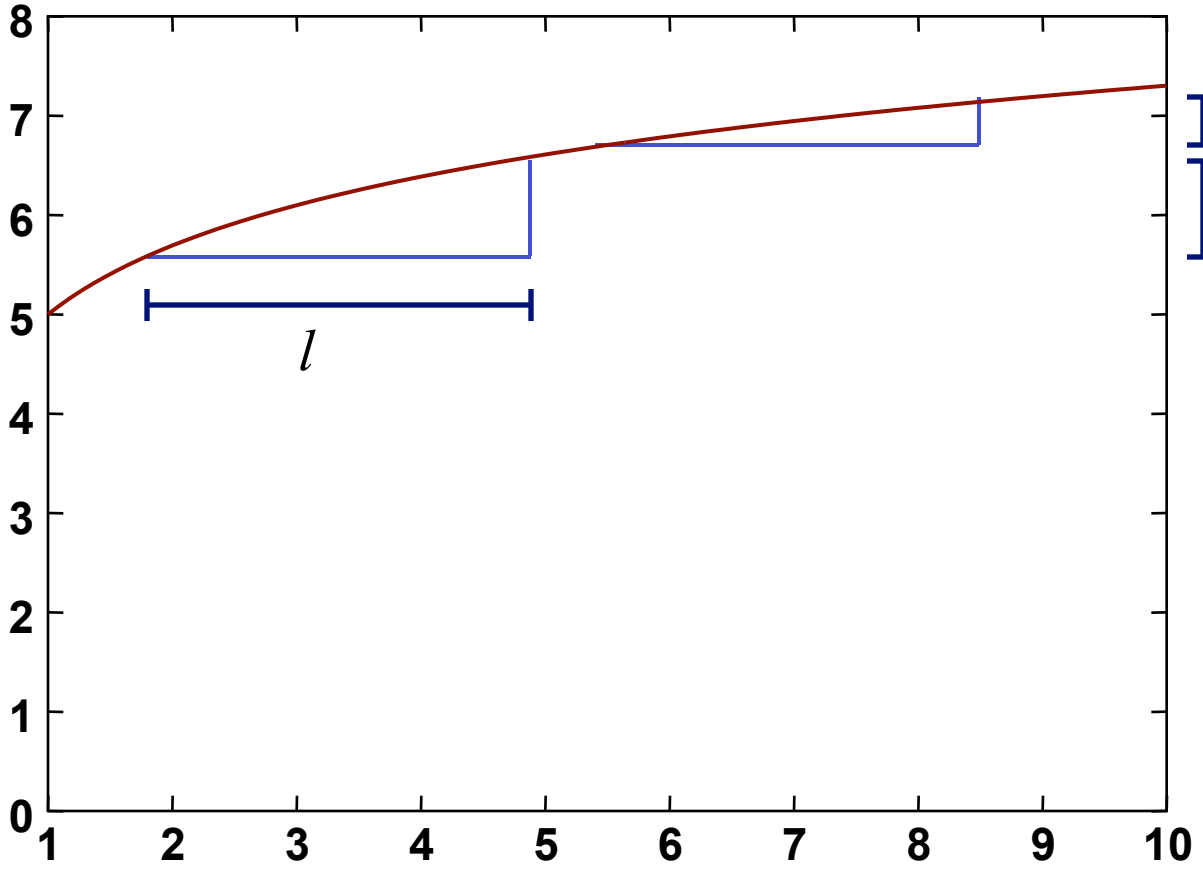
- affine: $$w(k) = \begin{cases} g + sk, & k \geq 1 \\ 0, & k = 0 \end{cases}$$

- concave: a function for which the following holds for all k, l, m ≥ 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



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

$w(k+l)-w(k)$

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

Computational Complexity and Gap Penalty Functions

- linear: w(k) = sk — $O(n^2)$
- affine: w(k) = g + sk, where s<g — $O(n^2)$
- concave: $\Delta W(k) \geq \Delta W(k+1)$ for all $k \geq 1$, where $\Delta W(k) = W(k+1) - W(k)$ — $O(n^2)$
- 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