Pairwise Sequence Alignment

BMI/CS 576
www.biostat.wisc.edu/bmi576/
Mark Craven
craven@biostat.wisc.edu

Pairwise alignment: task definition

Given

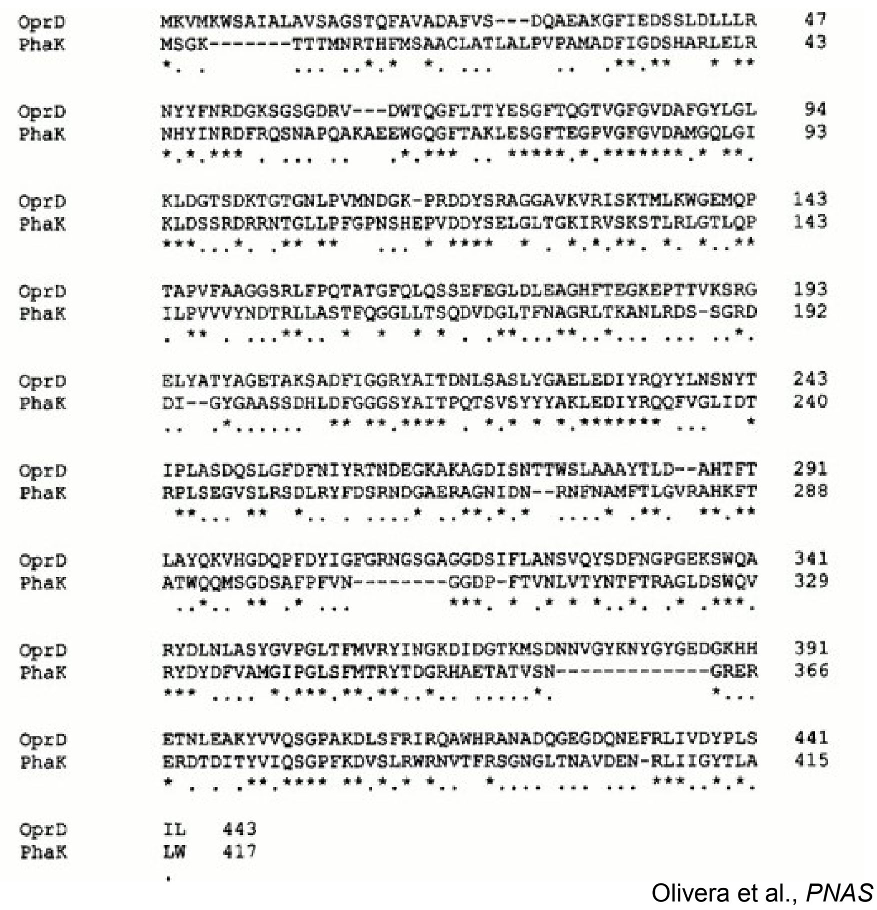
– a pair of sequences (DNA or protein)

– a method for scoring a candidate alignment

Do

– determine the correspondences between substrings in the sequences such that the similarity score is maximized

Protein alignment example



```
OprD      MKVMKWSAIALAVSAGSTQFAVADAFVS---DQAEAKGFIEDSSLDLLLR   47
PhaK      MSGK-------TTTMNRTHFMSAACLATLALPVPAMADFIGDSHARLELR   43
          *. .       ... ..*.*  *. ...      ..  .**.**   * **

OprD      NYYFNRDGKSGSGDRV---DWTQGFLTTYESGFTQGTVGFGVDAFGYLGL   94
PhaK      NHYINRDFRQSNAPQAKAEEWGQGFTAKLESGFTEGPVGFGVDAMGQLGI   93
          *.*.*** . ... ..   .*.*** .. *****.*.*******.* **.

OprD      KLDGTSDKTGTGNLPVMNDGK-PRDDYSRAGGAVKVRISKTMLKWGEMQP  143
PhaK      KLDSSRDRRNTGLLPFGPNSHEPVDDYSELGLTGKIRVSKSTLRLGTLQP  143
          ***...*. .** **   ... * ****  * . *.*.**. *. *..**

OprD      TAPVFAAGGSRLFPQTATGFQLQSSEFEGLDLEAGHFTEGKEPTTVKSRG  193
PhaK      ILPVVVYNDTRLLASTFQGGLLTSQDVDGLTFNAGRLTKANLRDS-SGRD  192
          . ** . ...**.. *  *  * * . .**...**..*... ... ..*.

OprD      ELYATYAGETAKSADFIGGRYAITDNLSASLYGAELEDIYRQYYLNSNYT  243
PhaK      DI--GYGAASSDHLDFGGGSYAITPQTSVSYYYAKLEDIYRQQFVGLIDT  240
          ..  .*......  ** **.**** . *.* * *.******* ...   *

OprD      IPLASDQSLGFDFNIYRTNDEGKAKAGDISNTTWSLAAAYTLD--AHTFT  291
PhaK      RPLSEGVSLRSDLRYFDSRNDGAERAGNIDN--RNFNAMFTLGVRAHKFT  288
           **... **  *.. . ....* ..**.*.*  ....* .**.  **.**

OprD      LAYQKVHGDQPFDYIGFGRNGSGAGGDSIFLANSVQYSDFNGPGEKSWQA  341
PhaK      ATWQQMSGDSAFPFVN--------GGDP-FTVNLVTYNTFTRAGLDSWQV  329
           ..*.. ** .* ...         ***. * .* * *..*. .* .***.

OprD      RYDLNLASYGVPGLTFMVRYINGKDIDGTKMSDNNVGYKNYGYGEDGKHH  391
PhaK      RYDYDFVAMGIPGLSFMTRYTDGRHAETATVSN-------------GRER  366
          ***  .... *.***.**.**..*.. .....*.             *...

OprD      ETNLEAKYVVQSGPAKDLSFRIRQAWHRANADQGEGDQNEFRLIVDYPLS  441
PhaK      ERDTDITYVIQSGPFKDVSLRWRNVTFRSGNGLTNAVDEN-RLIIGYTLA  415
          *  . . .**.**** **.*.* *..  *.... ... ... ***..*.*.

OprD      IL   443
PhaK      LW   417
          .
```

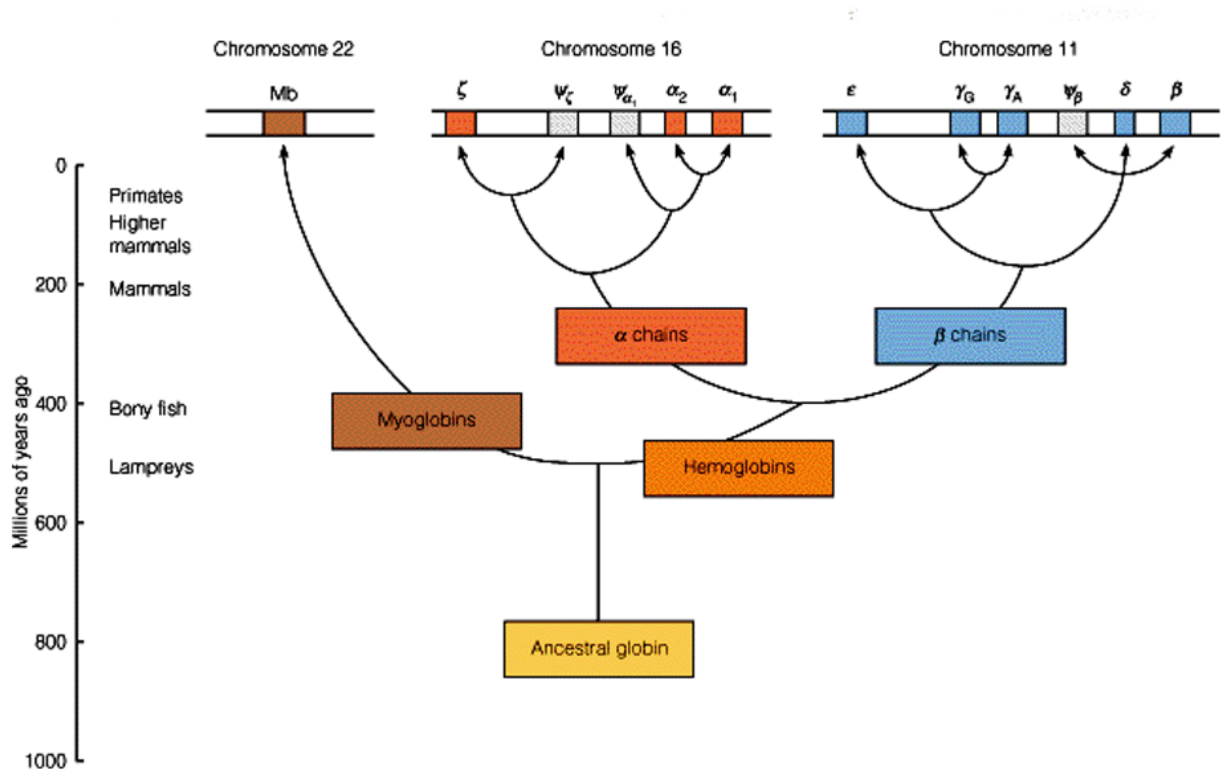
Alignment of the PhaK protein from Pseudomonas putida and OprD protein from Pseudomonas aeruginos

Olivera et al., *PNAS* 95:6419-6424, 1998

The role of homology in alignment

- *homology*: similarity due to descent from a common ancestor
- often we can infer homology from similarity
- thus we can sometimes infer structure/function from sequence similarity

Homology example: evolution of the globins



Homology

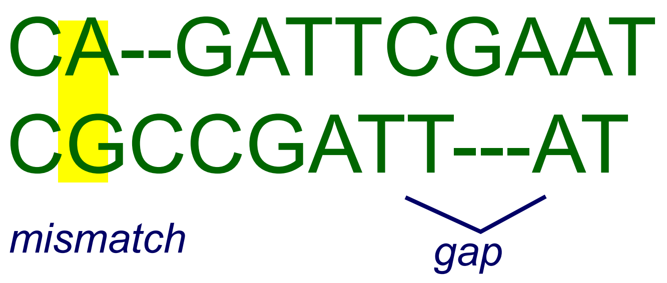
- homologous sequences can be divided into two groups
  - *orthologous sequences*: sequences that differ because they are found in different species (e.g. human α-globin and mouse α-globin)
  - *paralogous sequences*: sequences that differ because of a gene duplication event (e.g. human α-globin and human β-globin, various versions of both)

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

Mismatches and gaps

- substitutions in *homologous* sequences result in mismatches in an alignment
- insertions/deletions in *homologous* sequences result in mismatches in an alignment

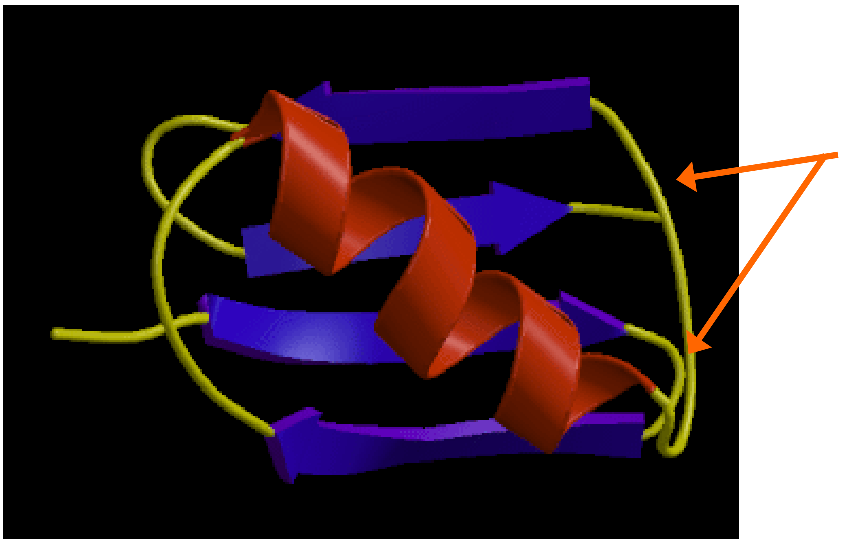


Alignment scales

- for short DNA sequences (gene scale) we will generally only consider
  - substitutions
  - insertions/deletions
- for longer DNA sequences (genome scale) we will consider additional events
  - transpositions
  - inversions
- in this course we will focus on the case of short sequences

Insertions/deletions and protein structure

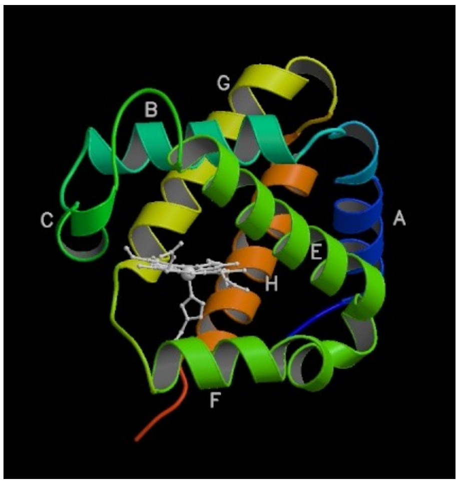
- Why is it that two “similar” sequences may have large insertions/deletions?
  - some insertions and deletions may not significantly affect the structure of a protein

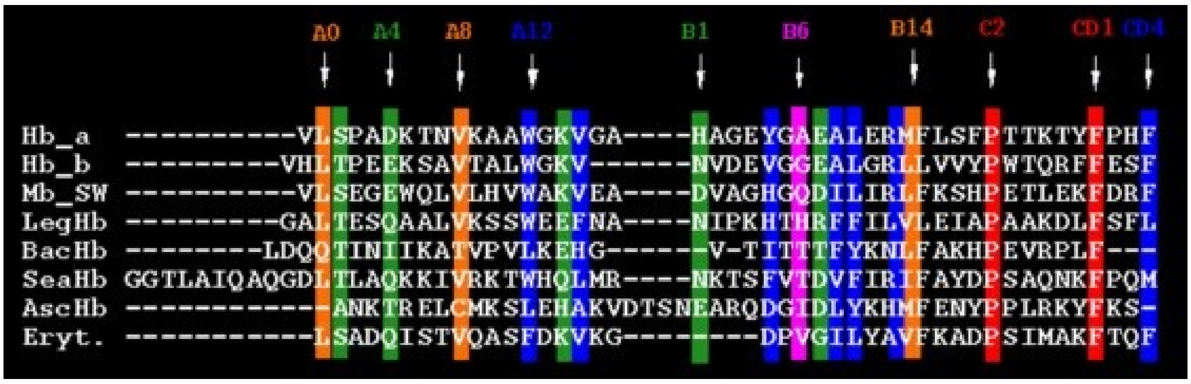


loop structures: insertions/deletions here not so significant

Example alignment: globins

- figure at right shows prototypical structure of globins
- figure below shows part of alignment for 8 globins





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
```

Issues in sequence alignment

- the sequences we’re comparing typically 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

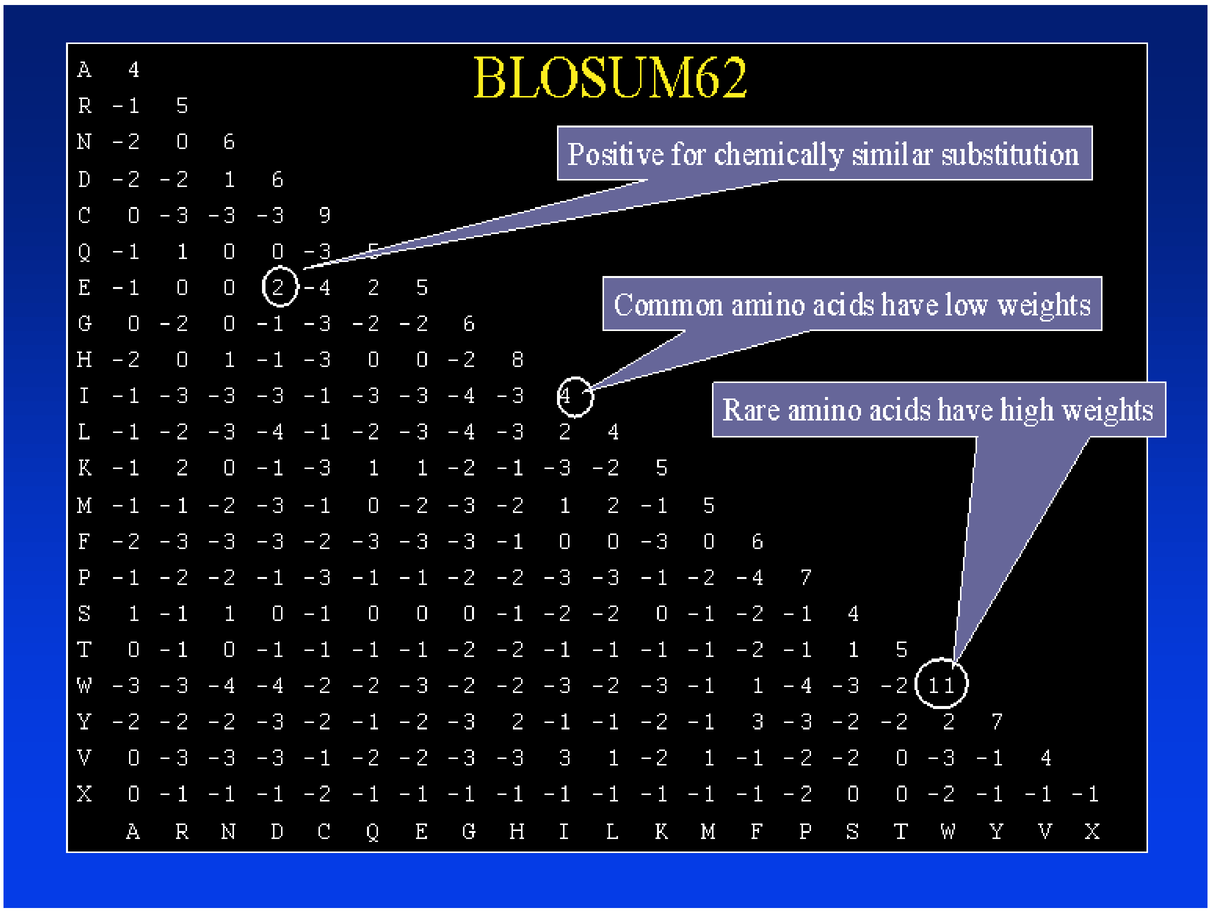
Types of alignment

- *global*: find best match of both sequences in their entirety
- *local*: find best subsequence match
- *semi-global*: find best match without penalizing gaps on the ends of the alignment

Scoring an alignment: what is needed?

- substitution matrix
  - *s(a,b)* indicates score of aligning character *a* with character *b*
- gap penalty function
  - *w(g)* indicates cost of a gap of length *g*

Blosum 62 substitution matrix


Positive for chemically similar substitution
Common amino acids have low weights
Rare amino acids have high weights

BLOSUM62

A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
X	0	-1	-1	-1	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-2	0	0	-2	-1	-1	-1
	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	X

Linear gap penalty function

- different gap penalty functions require somewhat different dynamic programming algorithms

- the simplest case is when a linear gap function is used

$$w(g) = -g \times d$$

where d is a constant

- 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) – 3d + s(D,G) – 2d
  …

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
```

- Can we find the highest scoring alignment by enumerating all possible alignments and picking the best?

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*

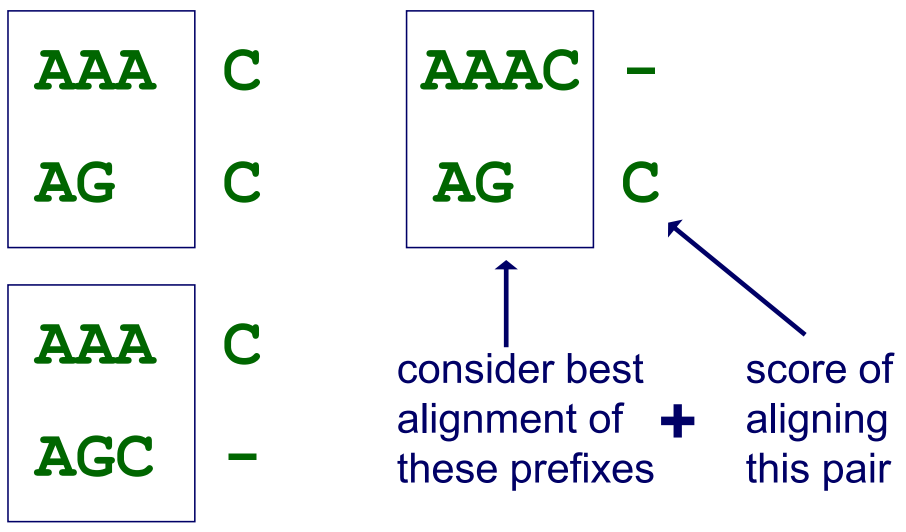
- e.g. two sequences of length 100 have $\approx 10^{77}$ possible alignments
- but we can use *dynamic programming* to find an optimal alignment efficiently

Pairwise alignment via dynamic programming

- first algorithm by Needleman & Wunsch, *Journal of Molecular Biology*, 1970
- *dynamic programming*: solve an instance of a problem by taking advantage of computed solutions for smaller subparts of the problem
- determine best alignment of two sequences by determining best alignment of all prefixes of the sequences

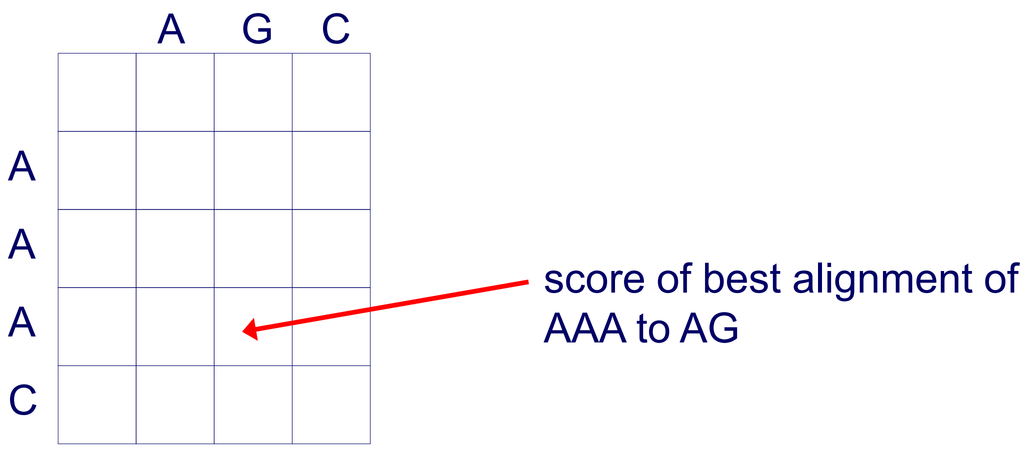
Dynamic programming idea

- 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 best alignment of previous characters


AAA C
AG C
AAAC –
AG C
AAA C
AGC –
consider best alignment of these prefixes
+
score of aligning this pair

Dynamic programming idea

- 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 $x[1\ldots i]$ with $y[1\ldots j]$



DP algorithm for global alignment with linear gap penalty

- one way to specify the DP is in terms of its recurrence relation:

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

Initializing matrix: global alignment with linear gap penalty

$$
\begin{array}{c|c|c|c|c|}
 & & A & G & C \\
\hline
 & 0 \leftarrow & -d \leftarrow & -2d \leftarrow & -3d \\
\hline
A & \uparrow\; -d & & & \\
\hline
A & \uparrow\; -2d & & & \\
\hline
A & \uparrow\; -3d & & & \\
\hline
C & \uparrow\; -4d & & & \\
\hline
\end{array}
$$

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

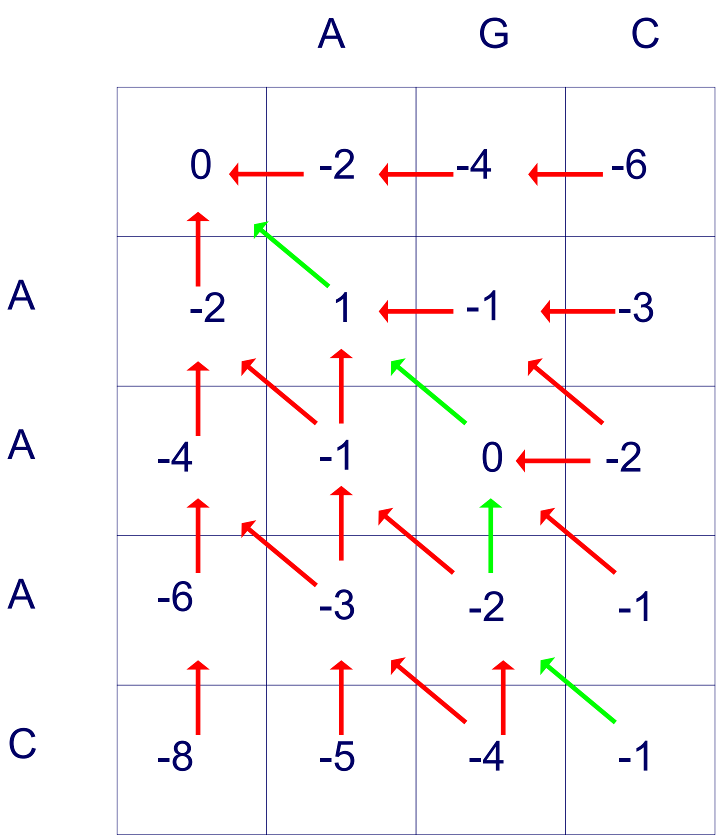
- suppose we choose the following scoring scheme:

$s(x_i, y_i) =$

+1	when	$x_i = y_i$
-1	when	$x_i \neq y_i$

d (penalty for aligning with a gap) = 2

Global alignment example



one optimal alignment

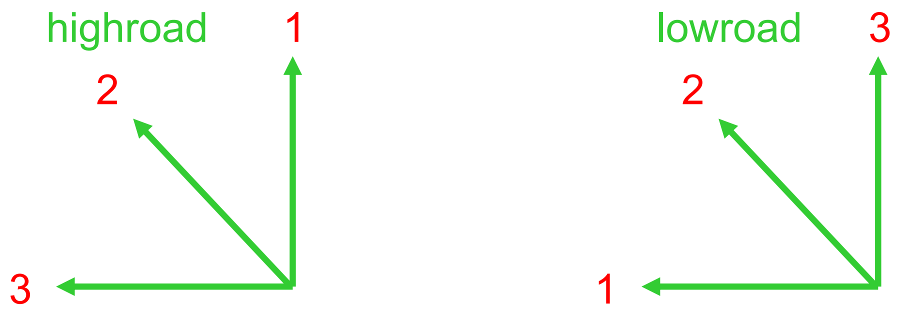
x:	A	A	A	C
y:	A	G	-	C

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)

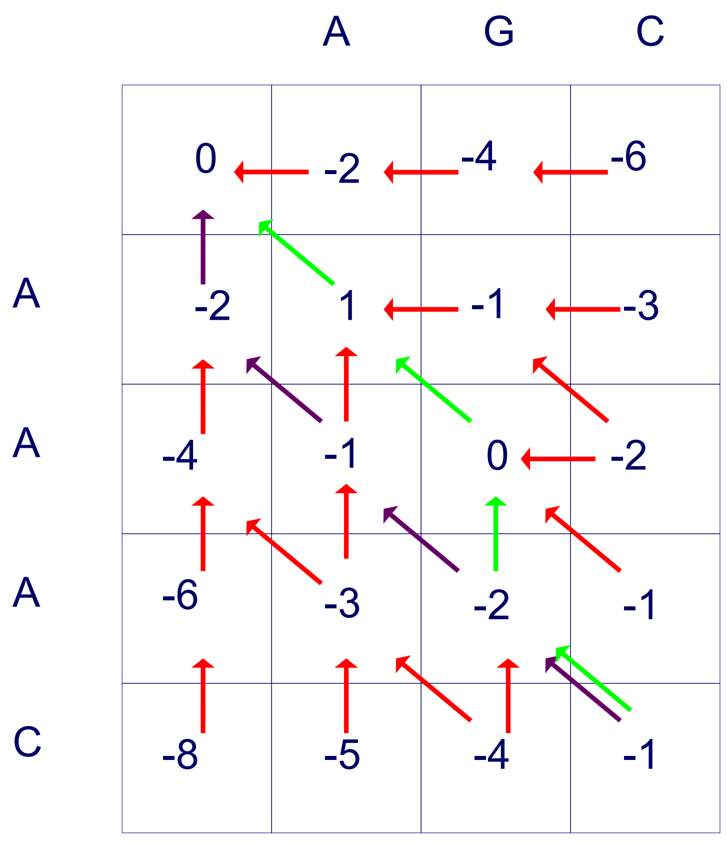
Equally optimal alignments

- 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

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 best match between sequences from one end to the other
- often we want a *local alignment*, the best match between <u>subsequences</u> of *x* and *y*

Example local alignment

- aligning my name against the sequence for dTDP-4-dehydrorhamnose reductase from the bacterium *opitutus terrae*

```
                                         MARKCRAVEN
…LSGAYHLAASGHTSWHGFASAIIDLMPLDARKCRAVEAIT…
```

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 <u>a suffix of</u> $x[1\ldots i]$ and <u>a suffix of</u> $y[1\ldots j]$

Local alignment DP algorithm

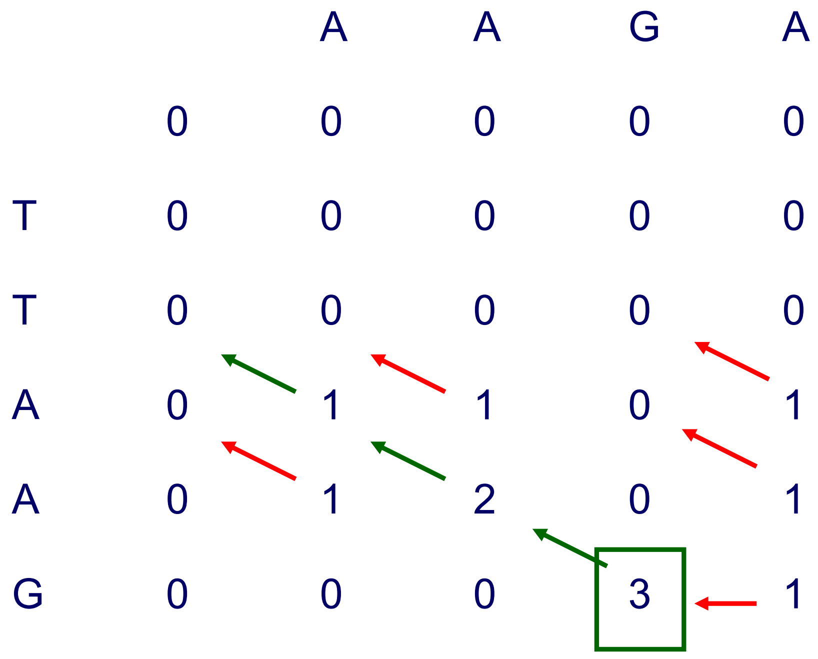
- the recurrence relation is slightly different than for global algorithm

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

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



$$\begin{array}{cccccc}
 & & A & A & G & A \\
 & 0 & 0 & 0 & 0 & 0 \\
T & 0 & 0 & 0 & 0 & 0 \\
T & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 1 & 1 & 0 & 1 \\
A & 0 & 1 & 2 & 0 & 1 \\
G & 0 & 0 & 0 & \boxed{3} & 1
\end{array}$$

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 *d* associated with <u>opening</u> a gap
  - a smaller penalty *e* for <u>extending</u> the gap

Gap penalty functions

linear

$$w(g) = -g \times d$$

affine

$$w(g) = \begin{cases} -d - (g-1)e, & g \geq 1 \\ 0, & g = 0 \end{cases}$$

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

Global alignment DP for the affine gap penalty case

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

$$I_x(i, j) = \max \begin{cases} M(i-1, j) - d \\ I_x(i-1, j) - e \end{cases}$$

$$I_y(i, j) = \max \begin{cases} M(i, j-1) - d \\ I_y(i, j-1) - e \end{cases}$$

Global alignment DP for the affine gap penalty case

- initialization

  $M(0,0) = 0$

  $I_x(i, 0) = -d - (i-1)e$ for $i > 0$

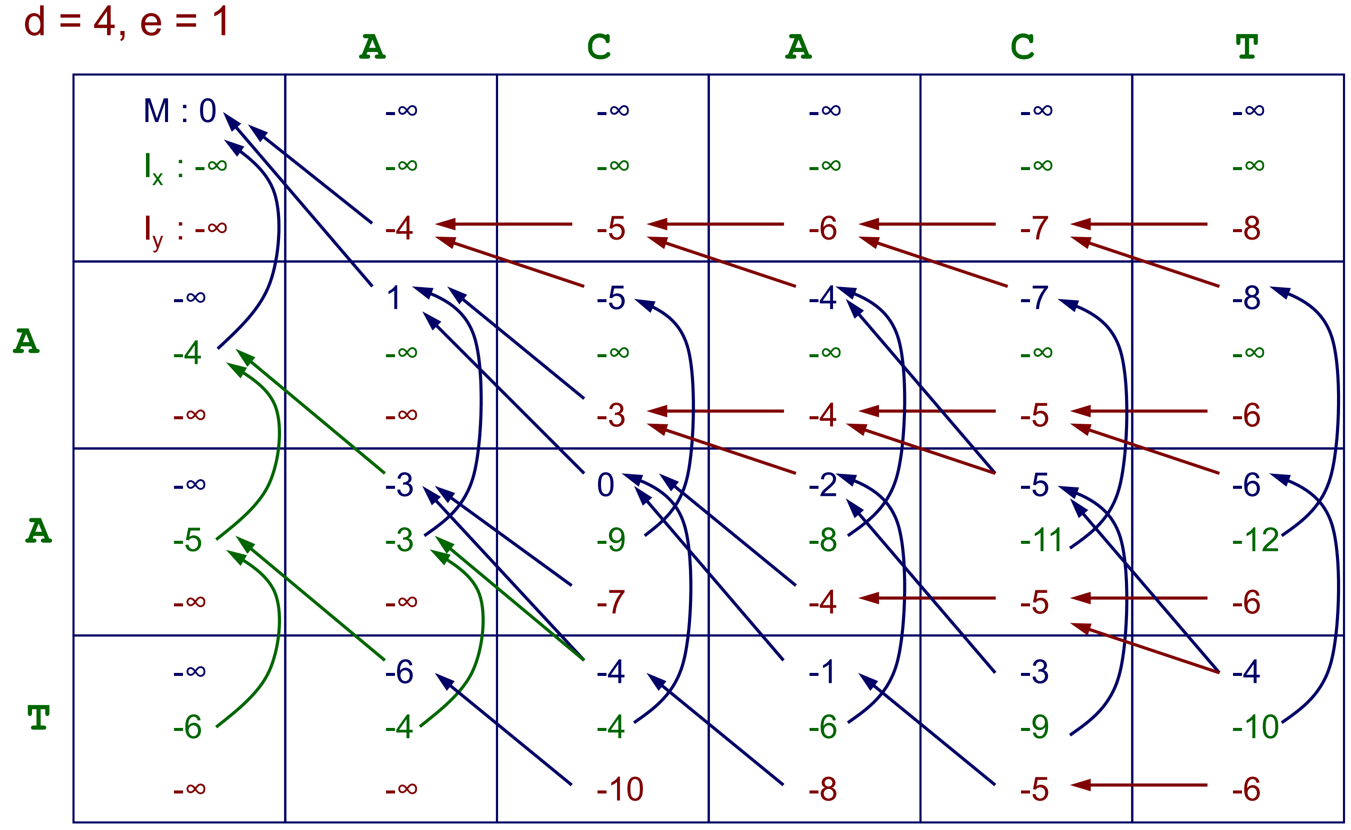
  $I_y(0, j) = -d - (j-1)e$ for $j > 0$

  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 $M(0,0)$
  - note that pointers may traverse all three matrices

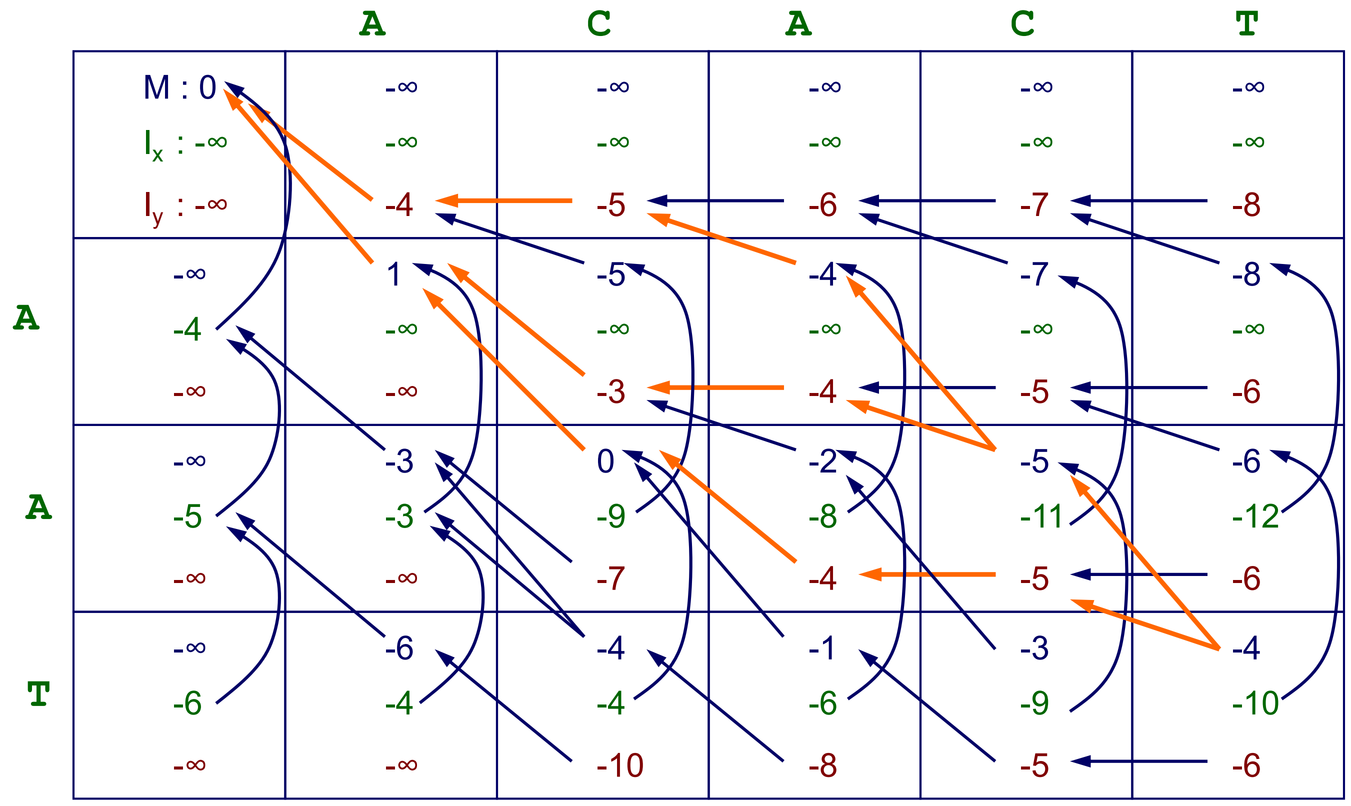
Global alignment example
(affine gap penalty)

d = 4, e = 1



	A	C	A	C	T
	M : 0 I_x : -∞ I_y : -∞	-∞ -∞ -4	-∞ -∞ -5	-∞ -∞ -6	-∞ -∞ -7	-∞ -∞ -8
A	-∞ -4 -∞	1 -∞ -∞	-5 -∞ -3	-4 -∞ -4	-7 -∞ -5	-8 -∞ -6
A	-∞ -5 -∞	-3 -3 -∞	0 -9 -7	-2 -8 -4	-5 -11 -5	-6 -12 -6
T	-∞ -6 -∞	-6 -4 -∞	-4 -4 -10	-1 -6 -8	-3 -9 -5	-4 -10 -6

Global alignment example (continued)



	A	C	A	C	T
	M : 0 I_x : -∞ I_y : -∞	-∞ -∞ -4	-∞ -∞ -5	-∞ -∞ -6	-∞ -∞ -7	-∞ -∞ -8
A	-∞ -4 -∞	1 -∞ -∞	-5 -∞ -3	-4 -∞ -4	-7 -∞ -5	-8 -∞ -6
A	-∞ -5 -∞	-3 -3 -∞	0 -9 -7	-2 -8 -4	-5 -11 -5	-6 -12 -6
T	-∞ -6 -∞	-6 -4 -∞	-4 -4 -10	-1 -6 -8	-3 -9 -5	-4 -10 -6

three optimal alignments:

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

Why three matrices are needed

- consider aligning the sequences **WFP** and **FW** using d = 5, e = 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

		W	**F**	**P**
	0	-5	-6	-7
F	-5	1	1	-4
W	-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-
```

Local alignment DP for the affine gap penalty case

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

$$I_x(i,j) = \max \begin{cases} M(i-1, j) - d \\ I_x(i-1, j) - e \end{cases}$$

$$I_y(i,j) = \max \begin{cases} M(i, j-1) - d \\ I_y(i, j-1) - e \end{cases}$$

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: $w(g) = -g \times d$

- affine:

$$w(g) = \begin{cases} -d - (g-1)e, & g \geq 1 \\ 0, & g = 0 \end{cases}$$

- convex: as gap length increases, magnitude of penalty for each additional character decreases

  e.g. $w(g) = -d - \log(g) \times e$

Computational complexity and gap penalty functions

linear:	$O(n^2)$
affine:	$O(n^2)$
convex:	$O(n^2 \log n)$
general:	$O(n^3)$

* assuming two sequences of length *n*

Alignment (global) with general gap penalty function

why the general case has time complexity $O(n^3)$

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

consider every previou
element in the column

k ranges over previous coordinates

consider every previous
element in the row

Pairwise alignment summary

- the number of possible alignments is exponential in the length of sequences being aligned
- dynamic programming can find optimal-scoring alignments in polynomial time
- the specifics of the DP depend on
  - local vs. global alignment
  - gap penalty function
- affine penalty functions are most commonly used