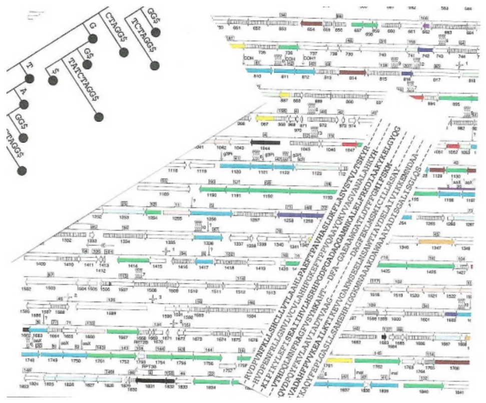
Lecture 2
Pairwise sequence alignment.

Principles Computational Biology

Teresa Przytycka, PhD



Assumptions:

- Biological sequences evolved by evolution.
- Micro scale changes: For short sequences (e.g. one domain proteins) we usually assume that evolution proceeds by:
  - Substitutions
  - Insertions/Deletions

Human MS**L**IC**S**IS**NEV**PE**H**P**CV**S**PVS** …

Protist MS**I**IC**T**IS**GQT**PE**E**P**VI**S-**KT** …

- Macro scale changes: For large sequences (e.g. whole genomes) we additionally allow,
  - Duplications
  - reversals
  - Protein segments known as domains are reused by different proteins (via various mechanisms)

Importance of sequence comparison

Discovering functional and evolutional relationships in biological sequences:

- Similar sequences $\rightarrow$ evolutionary relationship
- evolutionary relationship $\rightarrow$ related function
- Orthologs $\rightarrow$ same (almost same) function in different organisms.

"$\rightarrow$" should be read usually implies

Discovering sequence similarity by dot plots

Given are two sequence lengths *n* and *m* respectively. Do they share a similarity and if so in which region?

Dot-plot method: make *n* x *m* matrix with D and set D(i,j) = 1 if amino-acid (or nucleotide) position *i* in first sequence is the same (or similar as described later) as the amino-acid (nucleotide) at position j in the second sequence.

Print graphically the matrix printing dot for 1 and space for 0

Dot plot illustration

	T	T	A	C	T	C	A	A	T
A			●				●	●	
C				●		●			
T	●	●			●				●
C				●		●			
A			●				●	●	
T	●	●			●				●
T	●				●				●
A			●				●	●	
C				●		●			

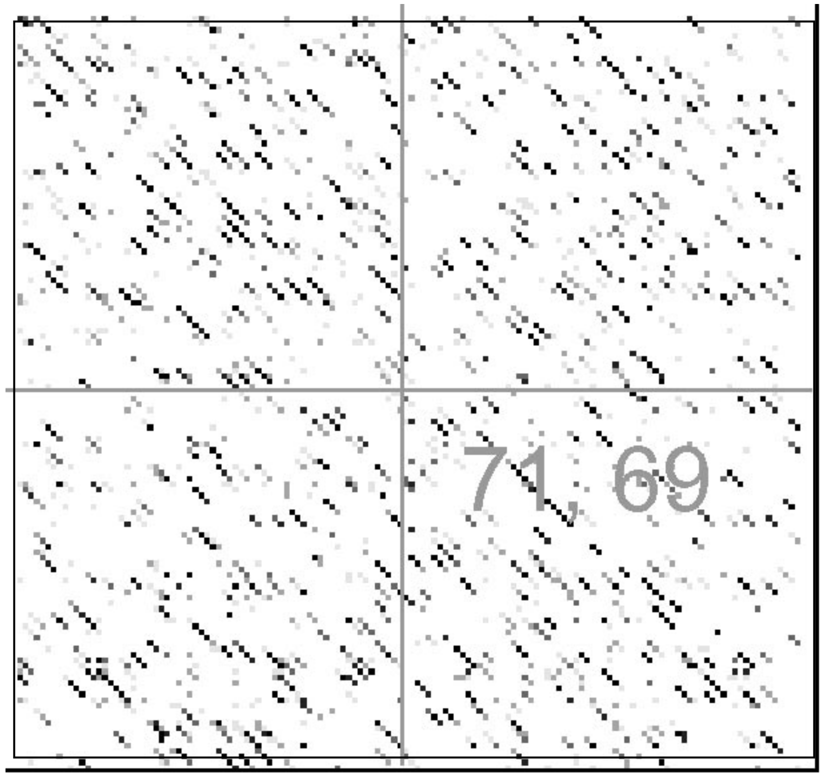
Diagonals from top left to bottom right correspond to regions that are identical in both sequences

The diagonals in the perpendicular direction correspond to reverse matches

Deletion?
or

Mutation?

An example of a dot plot where the relation between sequences in not obvious



(In an obvious case we would see a long diagonal line)

Figure drawn with Dotter : www.cgb.ki.se/cgb/sonnhammer/Dotter.html

Removing noise in dot plots

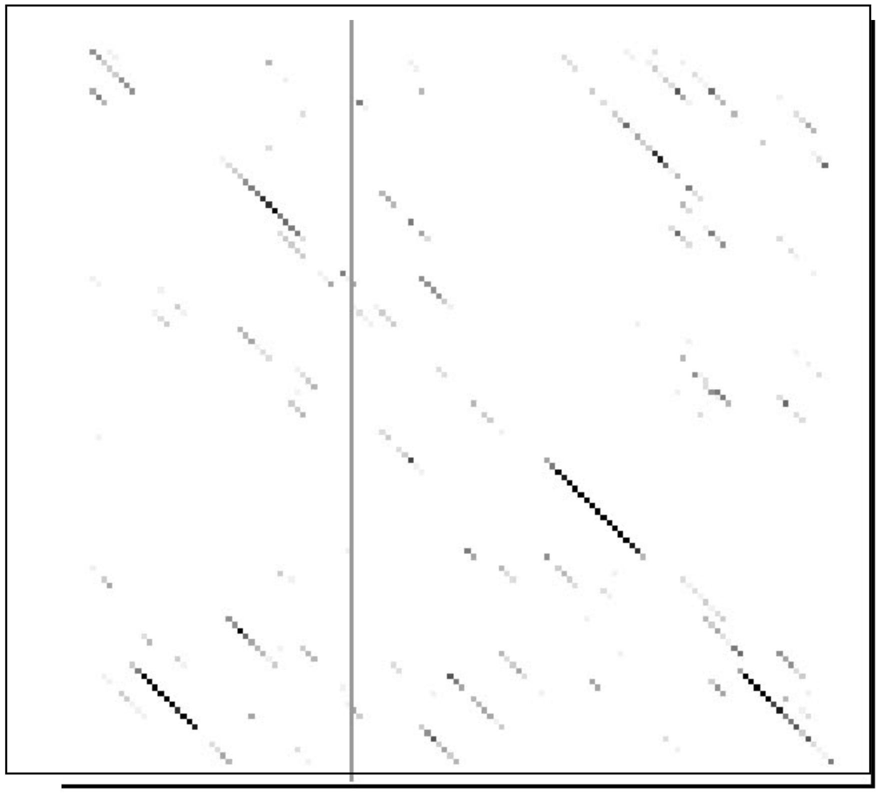
- Most of dots in a dot plot are by chance and introduce a lot of noise.
- Removing the noise: Put a dot ONLY if in addition to the similarity in the given position there is a similarity in the surrounding positions (we look at in a “window” of a size given as a parameter).

Dot plot with window 3

	T	T	A	C	T	C	A	A	T
A									
C				●		●			
T					●				
C				●		●			
A			●						
T		●							
T									
A									
C									

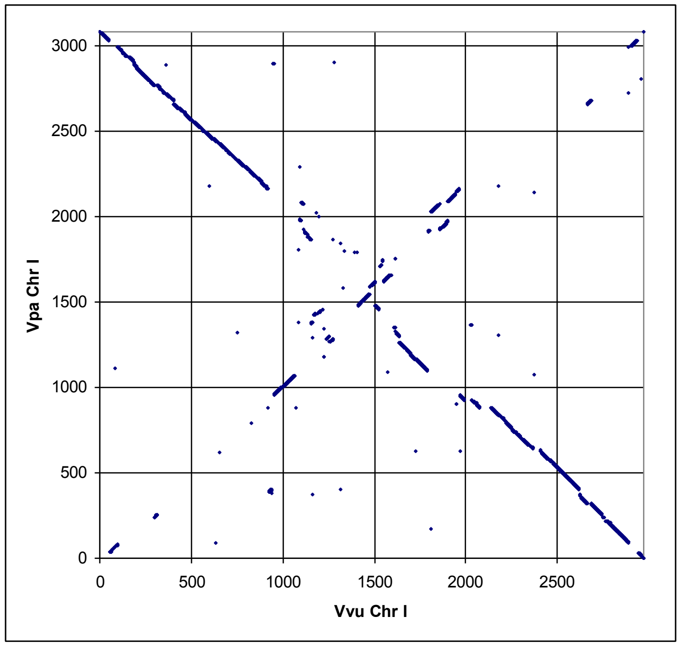
A dot is kept only if there ware a dots on both sides of it on the corresponding diagonal

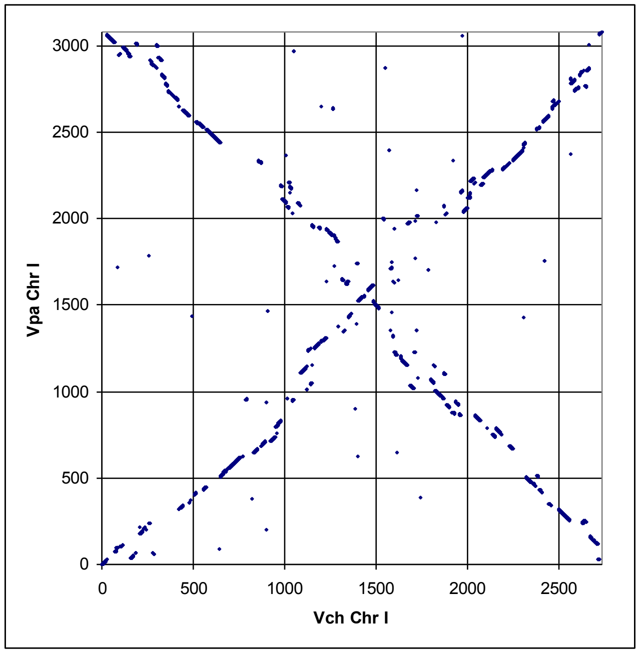
W = 10



EXAMPLE: Genomic dot plots

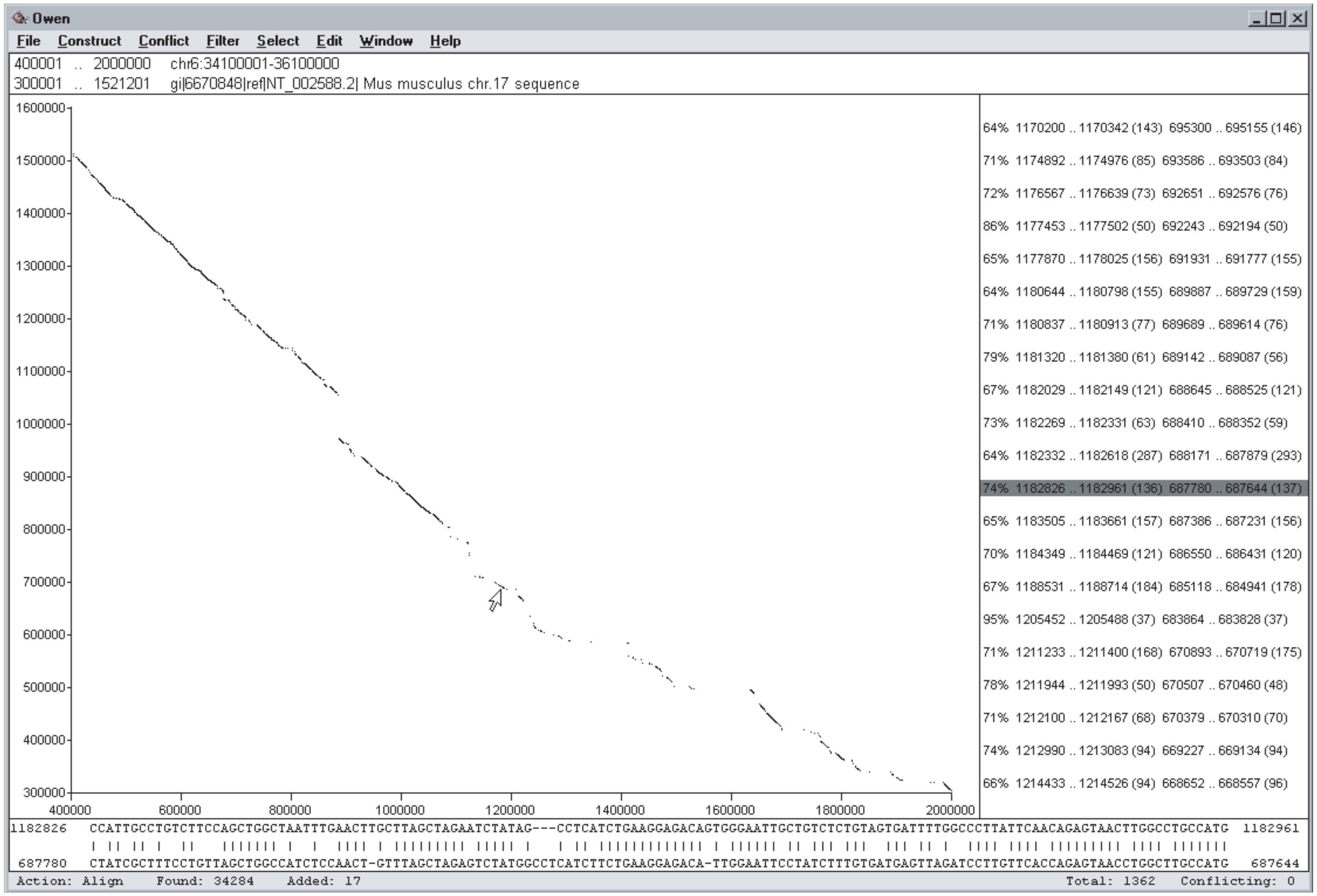
In these comparisons, each dot corresponds to a pair of orthologous genes The key feature of these plots is a distinct X-shaped pattern. This suggests that large chromosomal inversions reversed the genomic sequence symmetrically around the origin of replication; such symmetrical inversions appear to be a common feature of bacterial genome evolution.





OWEN: aligning long collinear regions of genomes

OWEN is an interactive tool for aligning two long DNA sequences that represents similarity between them by a chain of collinear local similarities. OWEN employs several methods for constructing and editing local similarities and for resolving conflicts between them.



```
400001 .. 2000000   chr6:34100001-36100000
300001 .. 1521201   gi|6670848|ref|NT_002588.2| Mus musculus chr.17 sequence
```

```
64%  1170200 .. 1170342 (143)  695300 .. 695155 (146)
71%  1174892 .. 1174976 (85)  693586 .. 693503 (84)
72%  1176567 .. 1176639 (73)  692651 .. 692576 (76)
86%  1177453 .. 1177502 (50)  692243 .. 692194 (50)
65%  1177870 .. 1178025 (156)  691931 .. 691777 (155)
64%  1180644 .. 1180798 (155)  689887 .. 689729 (159)
71%  1180837 .. 1180913 (77)  689689 .. 689614 (76)
79%  1181320 .. 1181380 (61)  689142 .. 689087 (56)
67%  1182029 .. 1182149 (121)  688645 .. 688525 (121)
73%  1182269 .. 1182331 (63)  688410 .. 688352 (59)
64%  1182332 .. 1182618 (287)  688171 .. 687879 (293)
74%  1182826 .. 1182961 (136)  687780 .. 687644 (137)
65%  1183505 .. 1183661 (157)  687386 .. 687231 (156)
70%  1184349 .. 1184469 (121)  686550 .. 686431 (120)
67%  1188531 .. 1188714 (184)  685118 .. 684941 (178)
95%  1205452 .. 1205488 (37)  683864 .. 683828 (37)
71%  1211233 .. 1211400 (168)  670893 .. 670719 (175)
78%  1211944 .. 1211993 (50)  670507 .. 670460 (48)
71%  1212100 .. 1212167 (68)  670379 .. 670310 (70)
74%  1212990 .. 1213083 (94)  669227 .. 669134 (94)
66%  1214433 .. 1214526 (94)  668652 .. 668557 (96)
```

```
1182826   CCATTGCCTGTCTTCCAGCTGGCTAATTTGAACTTGCTTAGCTAGAATCTATAG---CCTCATCTGAAGGAGACAGTGGGAATTGCTGTCTCTGTAGTGATTTTGGCCCTTATTCAACAGAGTAACTTGGCCTGCCATG  1182961
          | || || |  ||   |||||||| |  |  |||| | ||||||||| |||| |   | || ||||||||||||| | |||||| || || |||  ||| || |  ||||| |||| ||||||||| |||| |||||||
 687780   CTATCGCTTTCCTGTTAGCTGGCCATCTCCAACT-GTTTAGCTAGAGTCTATGGCCTCATCTTCTGAAGGAGACA-TTGGAATTCCTATCTTTGTGATGAGTTAGATCCTTGTTCACCAGAGTAACCTGGCTTGCCATG   687644
Action: Align    Found: 34284    Added: 17                                                                    Total: 1362    Conflicting: 0
```

Sequence alignment

- Write one sequence along the other so that to expose any similarity between the sequences. Each element of a sequence is either placed alongside of corresponding element in the other sequence or alongside a special "gap" character
- Example: TGKGI and AGKVGL can be aligned as

```
 TGK - GI
AGKVGL
```

- Is there a better alignment? How can we compare the "goodness" of two alignments.
- We need to have:
  - A way of scoring an alignment
  - A way of computing maximum score alignment.

Identity score

Let (x,y) be an aligned pair of elements of two sequences (at least one of x,y must not be a gap).

$$\mathrm{id}(x, y) = \begin{cases} 1 \text{ if } x = y \\ 0 \text{ if } x \neq y \end{cases}$$

Score of an alignment = sum of scores of aligned pairs

```
TGK - G
AGKVG
```

$$\frac{\begin{array}{l}\text{TGK - G}\\ \text{AGKVG}\end{array}}{0+1+1+0+1 = 3}$$

60 % identical

Gap penalties

Consider two pairs of alignments:

```
ATCG              AT – C G
ATTG      and     AT T  - G

ATC - -  T A   and   AT - C  - T A
ATT T T TA           AT T T T TA
```

They have the same identity score but alignment on the left is more likely to be correct

- The first problem is corrected by introducing "gap penalty".
- Second problem is corrected by introducing additional penalty for opening a gap.

Example

Score the above alignment using identity score; gap penalty = 1
Gap opening penalty = 2

```
ATCG
ATTG
```

1+1+0+1=3

```
AT – C G
AT T  - G
```

1+1-2-1-2-1+1=-3

```
ATC - -  T A
ATT T T TA
```

1+1+0-2-1-1+1+1=0

```
AT - C  - T A
AT T T T TA
```

1+1-2-1+0-2-1+1+1=-2

Problems with identity score

- In the two pairs of aligned sequence below there are mutations at the first and 6th position and insertion (or deletion) on the 4th position. However while V and A share significant biophysical similarity and we often see mutation between them, W and A do not often substitute one for the other.

```
  VGK – GI…     WGK – GI…
   AGKVGL…      AGKVGL
```

- What if I mutated to V and then back to I should this have the same score as when I was unchanged? If we will like to use the score to estimate evolutionary distances it would be wrong to consider them as identical.

Scoring Matrices

An amino-acid scoring matrix is a 20x20 table such that position indexed with amino-acids so that position X,Y in the table gives the score of aligning amino-acid X with amino-acid Y

Identity matrix – Exact matches receive one score and non-exact matches a different score (1 on the diagonal 0 everywhere else)
Mutation data matrix – a scoring matrix compiled based on observation of protein mutation rates: some mutations are observed more often then other (PAM, BLOSUM).
Not used:
Physical properties matrix – amino acids with similar biophysical properties receive high score.
Genetic code matrix – amino acids are scored based on similarities in the coding triple.

(scoring matrices will be discussed during next class)

Principles of Dynamic programming

- Need to figure out how to use solution to smaller problems for solving larger problem.
- We need to keep a reasonable bound on how many sub-problems we solve
- Make sure that each sub-problem is solved only once

Dynamic programming algorithm for computing the score of the best alignment

For a sequence $S = a_1, a_2, \ldots, a_n$ let $S_j = a_1, a_2, \ldots, a_j$

S,S' – two sequences

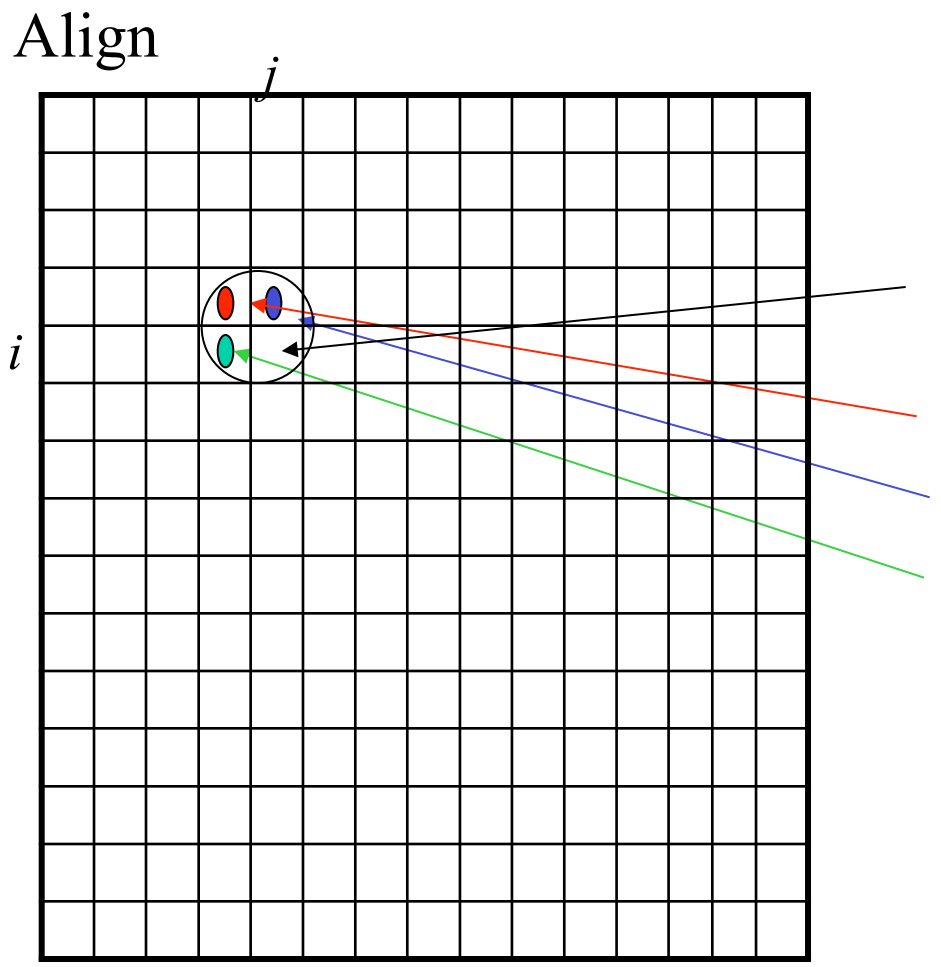
$\mathrm{Align}(S_i, S'_j)$ = the score of the highest scoring alignment between S^1_i, S^2_j

$S(a_i, a'_j)$= similarity score between amino acids a_i and a_j given by a scoring matrix like PAM, BLOSUM

g – gap penalty

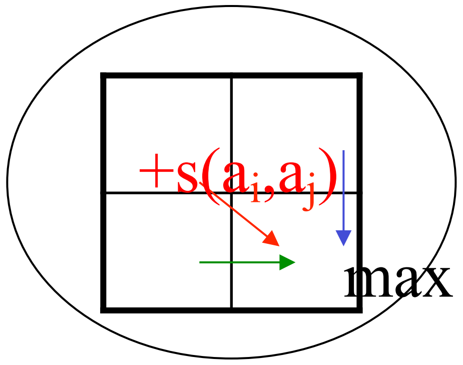
$$\mathrm{Align}(S_i, S'_j) = \max \begin{cases} \mathrm{Align}(S_{i-1}, S'_{j-1}) + S(a_i, a'_j) \\ \mathrm{Align}(S_i, S'_{j-1}) - g \\ \mathrm{Align}(S_{i-1}, S'_j) - g \end{cases}$$

Organizing the computation – dynamic programming table



Align(i,j) =

$$\text{Align}(S_i,S'_j) = \max \begin{cases} \text{Align}(S_{i-1},S'_{j-1}) + s(a_i, a'_j) \\ \text{Align}(S_{i-1},S'_j) - g \\ \text{Align}(S_i,S'_{j-1}) - g \end{cases}$$

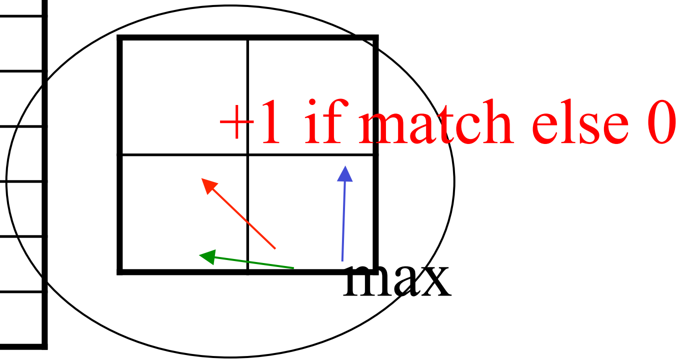


Example of DP computation with g = 0; match = 1; mismatch=0

Maximal Common Subsequence

initialization

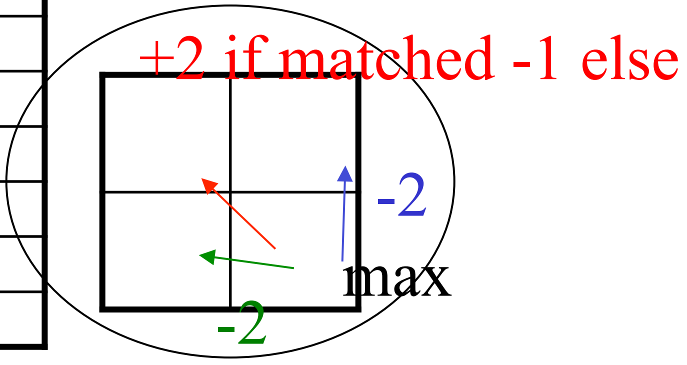
		A	T	T	G	C	G	C	G	C	A	T
	0	0	0	0	0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1	1	1	1	1	1
T	0	1	2	2	2	2	2	2	2			
G	0	1	2									
C	0	1										
T	0	1										
T	0	1										
A	0	1										
A	0	1										
C	0	1										
C	0	1										
A	0	1										



Example of DP computation with g = 2 match = 2; mismatch = -1

Initialization (penalty for starting with a gap)

		A	T	T	G	C	G	C	G	C	A	T
	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22
A	-2	↖2	←0	-2								
T	-4	↑0	↖4									
G	-6			↖6								
C	-8											
T	-10											
T	-12											
A	-14											
A	-16											
C	-18											
C	-20											
A	-22											



The iterative algorithm

```
m = |S|; n = |S’ |
for i ← 0 to m do A[i,0]←- i * g
for j ← 0 to n do A[0,j]← - j * g
for i ← 1 to m do
      for j ← 1 to n
          A[i,j]←max (
                    A[i-1,j] – g
                    A[i-1,j-1] + s(i,j)
                    A[i,j-1] – g
                    )
return(A[m,n])
```

Complexity of the algorithm

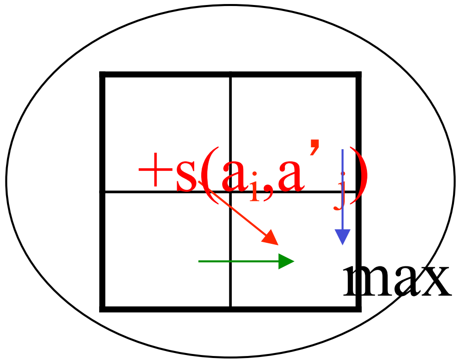
- Time O(nm); Space O(nm) where n, m the lengths of the two sequences.
- Space complexity can be reduced to O(n) by not storing the entries of dynamic programming table that are no longer needed for the computation (keep current row and the previous row only).

From computing the score to computing of the alignment

Desired output:

Sequence of substitutions/insertion/deletions leading to the optimal score.

ATTGCGTTATAT
AT- GCG- TATAT



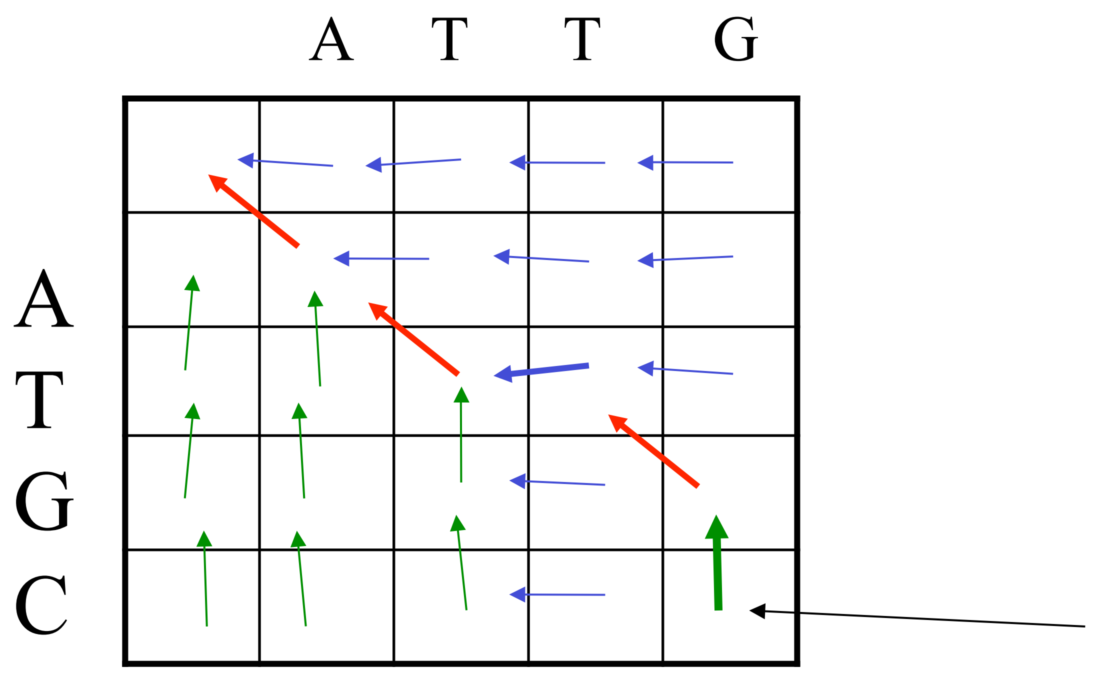
Red direction = mach
Blue direction = gap in horizontal sequence
Green direction = gap in vertical sequence

$a_1, a_2, \ldots\ a_j$
$a'_1, a'_2, \ldots\ a'_j$

$a_1, a_2, \ldots, a_j\ -$
$a'_1, a'_2, \ldots, a'_j$

$a_1, a_2, \ldots\ a_j$
$a'_1, a'_2, \ldots, a'_j\ -$

Recovering the path



A T T G -

A T - G C

If at some position several choices lead to the same max value, the path need not be unique.

Extra information not obligatory

Reducing space complexity in the global alignment

Recall: Computing the score in linear space is easy.

Leaving "trace" for finding optimal alignment is harder. Why?

Let $\mathrm{OPT}\begin{bmatrix} x \\ y \end{bmatrix}$ be an optimal alignment between sequence x and y

fix i, then there exist j such that $\mathrm{OPT}\begin{bmatrix} x \\ y \end{bmatrix}$ can be obtained as

$$\mathrm{OPT}\begin{bmatrix} x[1,\ldots i-1] \\ y[1,\ldots,j-1] \end{bmatrix} + \begin{matrix} x[i] \\ y[j] \end{matrix} + \mathrm{OPT}\begin{bmatrix} x[i+1,\ldots m] \\ y[j+1,\ldots,n] \end{bmatrix}$$

OR

$$\mathrm{OPT}\begin{bmatrix} x[1,\ldots i-1] \\ y[1,\ldots,j] \end{bmatrix} + \begin{matrix} x[i] \\ - \end{matrix} + \mathrm{OPT}\begin{bmatrix} x[i+1,\ldots m] \\ y[j+1,\ldots,n] \end{bmatrix}$$

Extra information – not obligatory

Computing which of the two cases holds and for what value of j:

1. Use dynamic programming for to compute the scores a[i,j] for fixed i=n/2 and all j. *O(nm/2)-time; linear space*
2. Do the same for the suffixes. *O(nm/2)-time; linear space*
3. Find out which of the two cases from the previous case applies and for which value of j.
4. Apply 1 & 2 recursively for the sequences to the left of (i,j) and to the right of (i,j) (figure from previous slide)

Ignoring initial and final gaps – semiglobal comparison

```
CAGCA - CTTGGATTCTCGG
- - - CAGCGTGG - - - - - - - - - -
```

No penalties for these gaps

Recall the initialization step for the dynamic programming table:

A[0,*i*] = *ig*; A[*j*,0]=*jg* – these are responsible for initial gaps.

set them to zero!

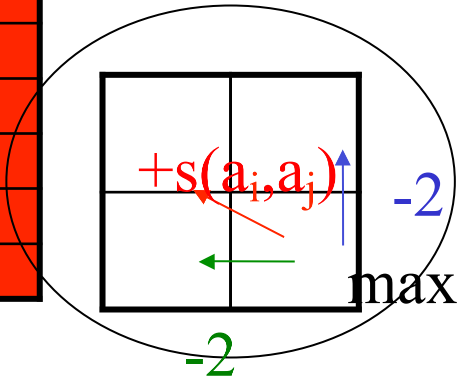
How to ignore final gaps?

Take the largest value in the last row /column and trace-back form there

Example of DP computation

ignoring flanking gaps by assigning 0 to initial gap penalties

		A	T	T	G	C	G	C	G	C	A	T
	0	0	0	0	0	0	0	0	0	0	0	0
A	0	1	-1									🟥
T	0	1	2									🟥
G	0											🟥
C	0											🟥
T	0											🟥
T	0											🟥
A	0											🟥
A	0											🟥
C	0											🟥
C	0											🟥
A	0	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥



To ignore final gap penalties choose the highest scoring entry in last column or last row and trace the path from there.

Trace back from the highest score in red row or column

Compressing the gaps

The two alignments below have the same score.
The second alignment is better.

```
ATTTTAGTAC          ATTTTAGTAC
ATT- - AGTAC        A-T-T -AGTAC
```

Solution: Have additional penalty for opening a gap

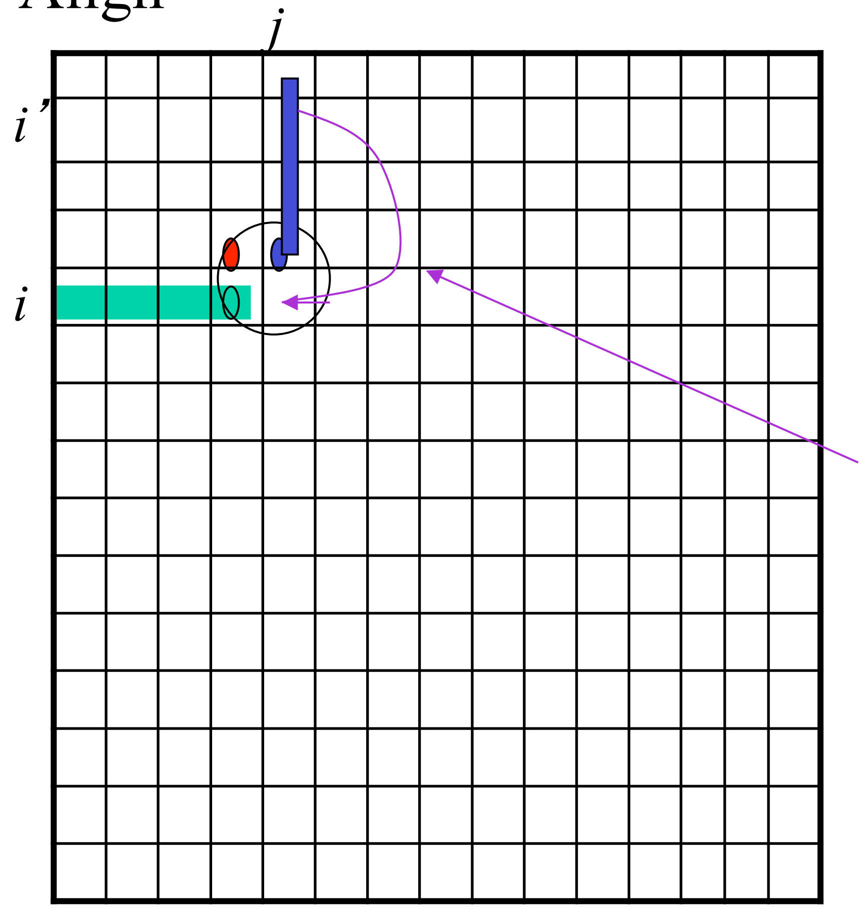
Affine gap penalty

$w(k) = h + gk$; h,g constants

Interpretation: const of starting a gap: h+g, extending gap: +g

Naïve extension of the previous algorithm

Align



Rather than checking for the best of Three values we have to check whole green row and blue column to consider all possible gap lengths.

That is find max over the following

max over i'

s(i' ,j) – opening gap -g(gap_length)

s(i-1,j-1)

max over j'

s(i,j') – opening gap -g(gap_length)

Complexity $O(n^3)$

General gap penalty

$$a[i,j] = \max \begin{cases} a[i-1,j-1] + s(i,j) \\ \max b[i,j-k] - w(k) \quad \text{for } 0 <= k <= i \\ \max b[i-k,j] - w(k) \quad \text{for } 0 <= k <= j \end{cases}$$

w(k) any gap penalty function (not necessarily afine)
k = size of a gap.

O(n^2) algorithm for afine gap penalty

Let $w(k) = h + gk$

S^1, S^2 compared sequences

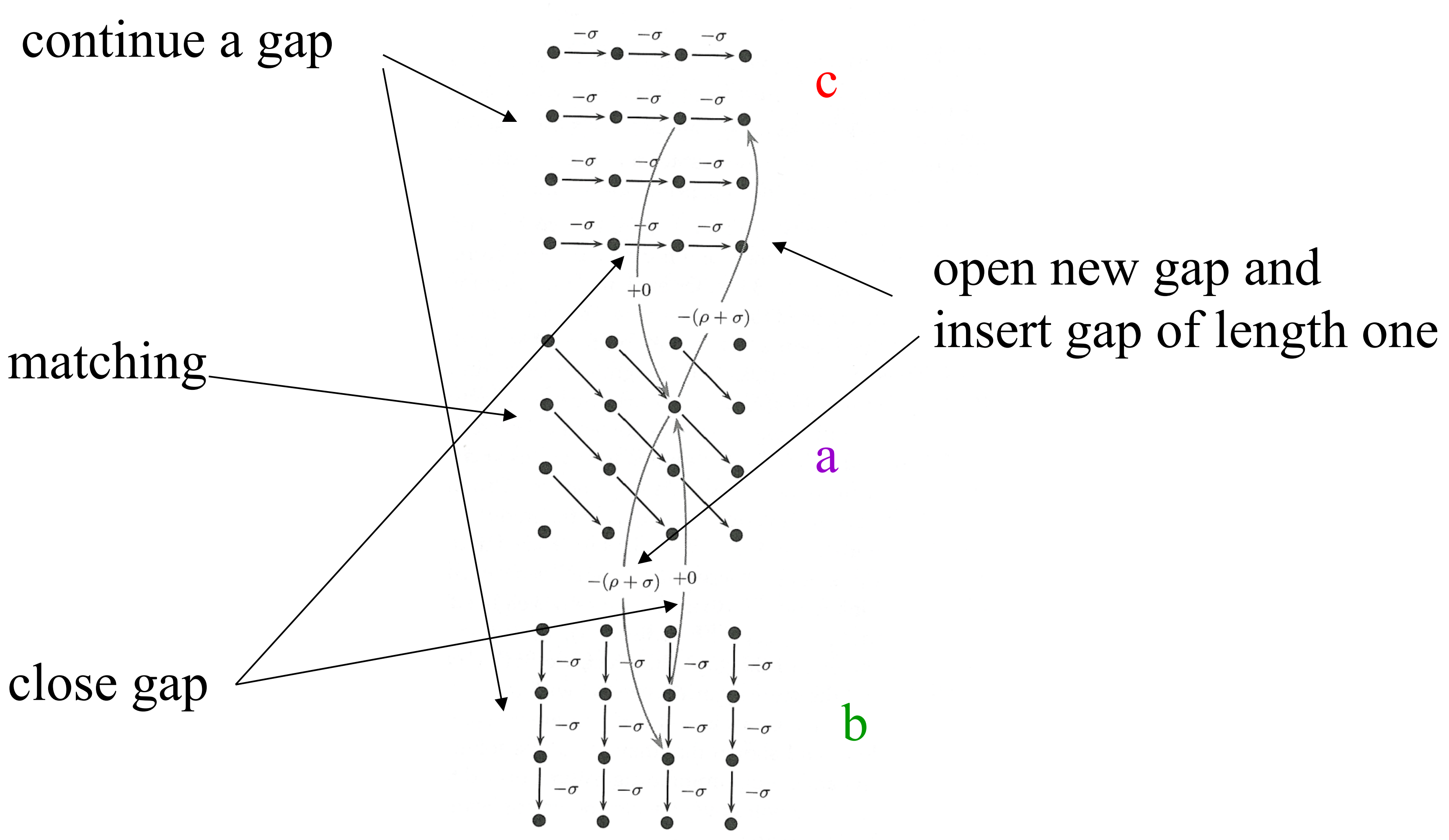
We will have 3 dynamic programming tables:

s | a[i,j] best possible alignment of S_i and S'_j

$\overset{\downarrow}{s}$ | b[i,j] best possible alignment of S_i and S'_j that ends with a gap in S

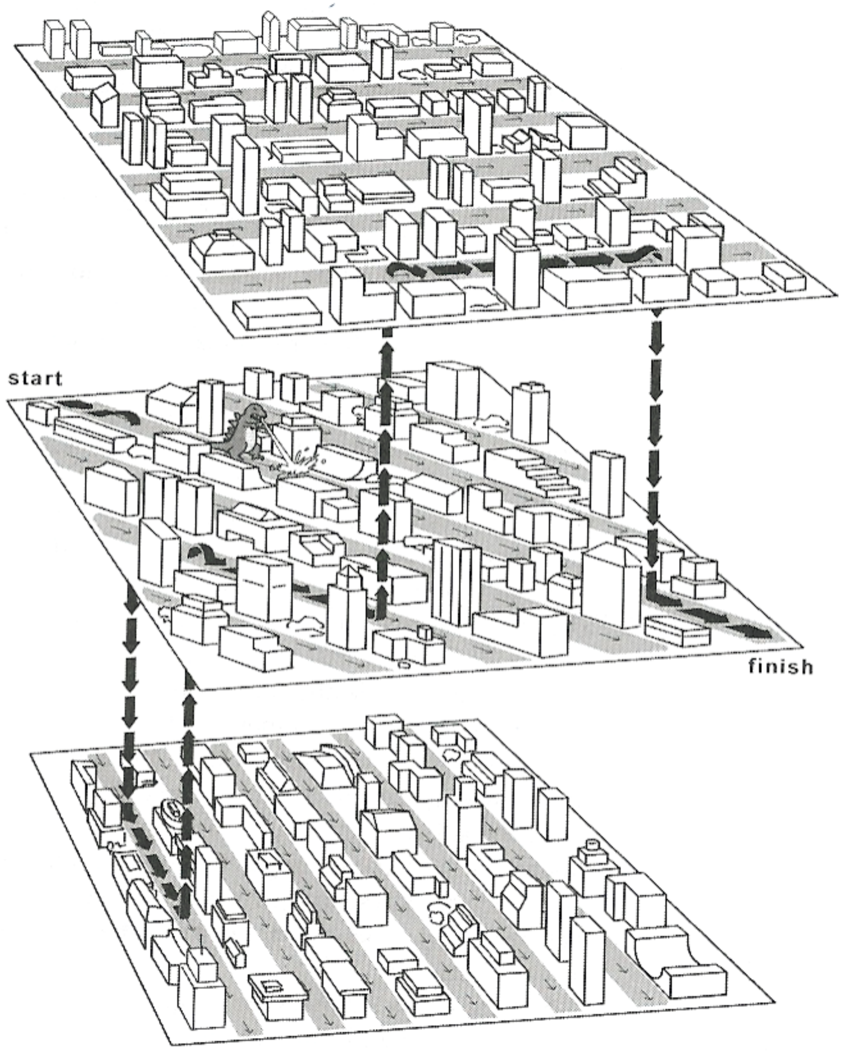
$\vec{s}$ | c[i,j] best possible alignment of S_i and S'_j that ends with a gap in S'

Text Pevzner's book notation



A three-level edit graph for alignment with affine gap pen

Jonson & Pevzner



Jonson & Pevzner

Initialization

Assume that we charge for initial and terminal gaps

-infinity is assigned where no alignment possible

a[0,0] = 0;
a[i,0] = - infinity (i<>0);
a[0,j] = -infinity (j<>0)

b[i,0] = - infinity
b[0,j] = - (h+gj)

c[i,0] = - (h+gi)
c[0,j] = - infinity

Affine gap penalty function - cont

$w(k) = h + gk$; h,g constants

Interpretation: const of starting a gap: h+g, extending gap: +g

Let a,b,c be as before. Now they can be completed as follows:

$$a[i,j] = \max \begin{cases} a[i-1,j-1] + s(i,j) \\ b[i,j] \\ c[i,j] \end{cases}$$

$$b[i,j] = \max \begin{cases} a[i,j-1] - (h+g) & \text{--- \textit{start a new gap in first seq}} \\ b[i,j-1] - g & \text{-- \textit{extend gap in second first by one}} \end{cases}$$

$$c[i,j] = \max \begin{cases} a[i-1,j] - (h+g) \\ c[i-1,j] - g \end{cases}$$

More sophisticated gap penalties

• gap penalty can be made to dependent non-linearly on length (e.g. as log function)

Let gap penalty be given by function w(k), where k-gap length.

- if w(k) is an arbitrary function – O(n^3) algorithm
- w(k) = log k (and other concave or convex functions) - O(n^2 log n) algorithm (non-trivial)

Comparing similar sequences

Similar sequences – optimal alignment has small number of gaps.

```
-GCGC-ATGGATTGAGCGA
TGCGCCATGGAT-GAGC-A
```

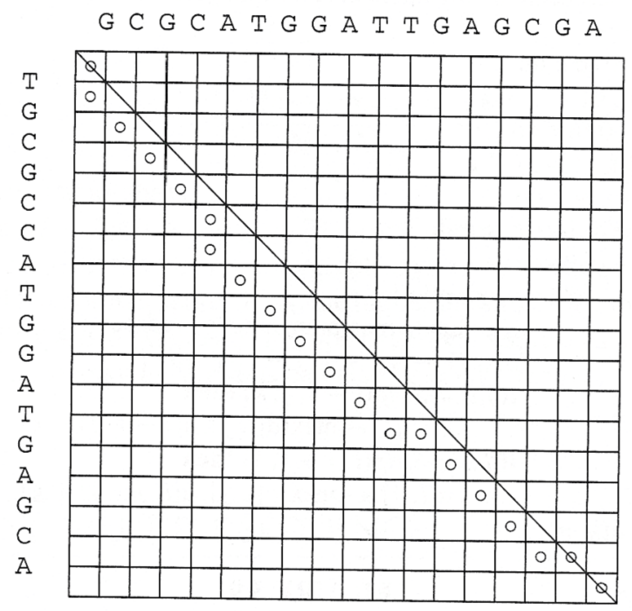

G C G C A T G G A T T G A G C G A
T G C G C C A T G G A T G A G C A

FIGURE 3.8

An optimal alignment and its corresponding path in the dynamic programming matrix. A line is drawn along the main diagonal.

The "alignment path' stays close to the diagonal

From book Setubal Meidanis "Introduction Comp. Mol. Biol"

Speeding up dynamic programming under assumption of small number of gaps

Idea: Use only strip of width 2k+1 along the diagonal. The rest of the array remains unused (and not initialized)

Modify the “max” expressions so that cell outside the strip are not considered.

Time complexity O(kn)
Space complexity – if you store only cells that are used – O(kn)

Identifying diagonals:

Number the diagonals as follows:
0 – main diagonal
+i ith diagonal above 0 diagonal
-i ith diagonal below 0 diagonal

Simple test to find the number of diagonal for element a(i,j):

$$j-i$$

k-band alignment

```
n = |S|= |S' |
for i ← 0 to k do A[i,0]←- i * g
for j ← 0 to k do A[0,j]← - j * g
for i ← 1 to n do
      for d ← -k to k
           j = i+d;
           if inside_strip(i,j,k) then:
           A[i,j]←max (
                              if inside_strip(i-1,j,k)  then A[i-1,j] – g else -infinity
                              A[i-1,j-1] + s(i,j)
                              if inside_strip(i-1,j,k) then A[i,j-1] – g else -infinity
                              )
return(A[m,n])
```

Where insid _strip(i,j,k) is a test if cell A[i.j] is inside the strip that is if |i-j|<=k

Local alignment

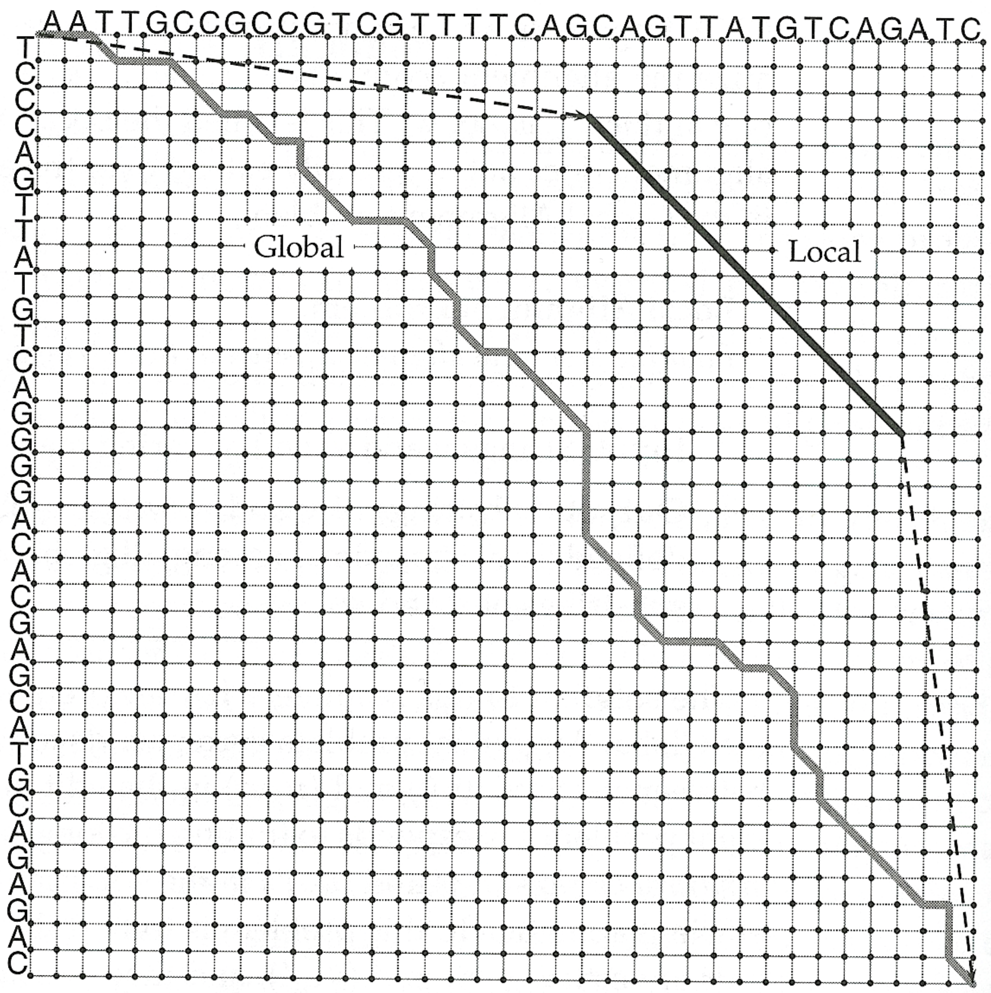
- The alignment techniques considered so far worked well for sequences which are similar over all their length
- This does not need to be the case: example gene from hox family have very short but   highly conserved subsequence – the so called hox domain.
- Considered so far global alignment methods (that is algorithm that try to find the best alignment over whole length can miss this local similarity region

Global

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |  || |  ||  | | | |||    || |  | |  | ||||   |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C
```

Local

```
                  tccCAGTTATGTCAGgggacacgagcatgcagaga
                     ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```


AATTGCCGCCGTCGTTTTCAGCAGTTATGTCAGATC
TCCCAGTTATGTCAGGGGACACGAGCATGCAGAGAC
Global
Local

Local alignment (Smith, Waterman)

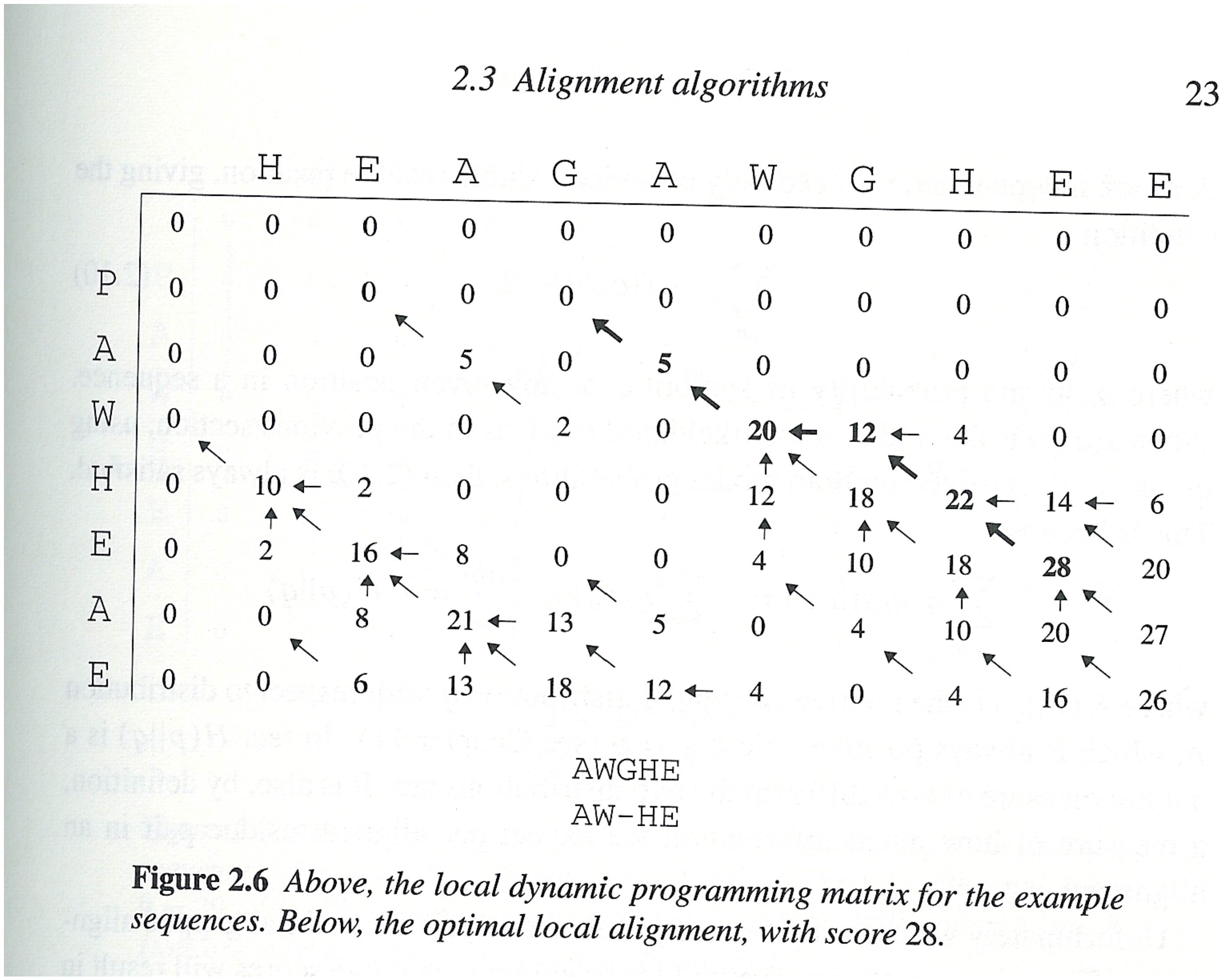
So far we have been dealing with global alignment.
Local alignment – alignment between substrings.
Main idea: If alignment becomes to bad – drop it.

Set p and g so that alignment of random strings gives negative score

$$a[i,j] = \max \begin{cases} a[i-1,j-1] + s(a_i, a_j) \\ a[i-1,j + g \\ a[i,j-1] + g \\ 0 \end{cases}$$

Finding the alignment: find the highest scoring cell and trace it back

Example

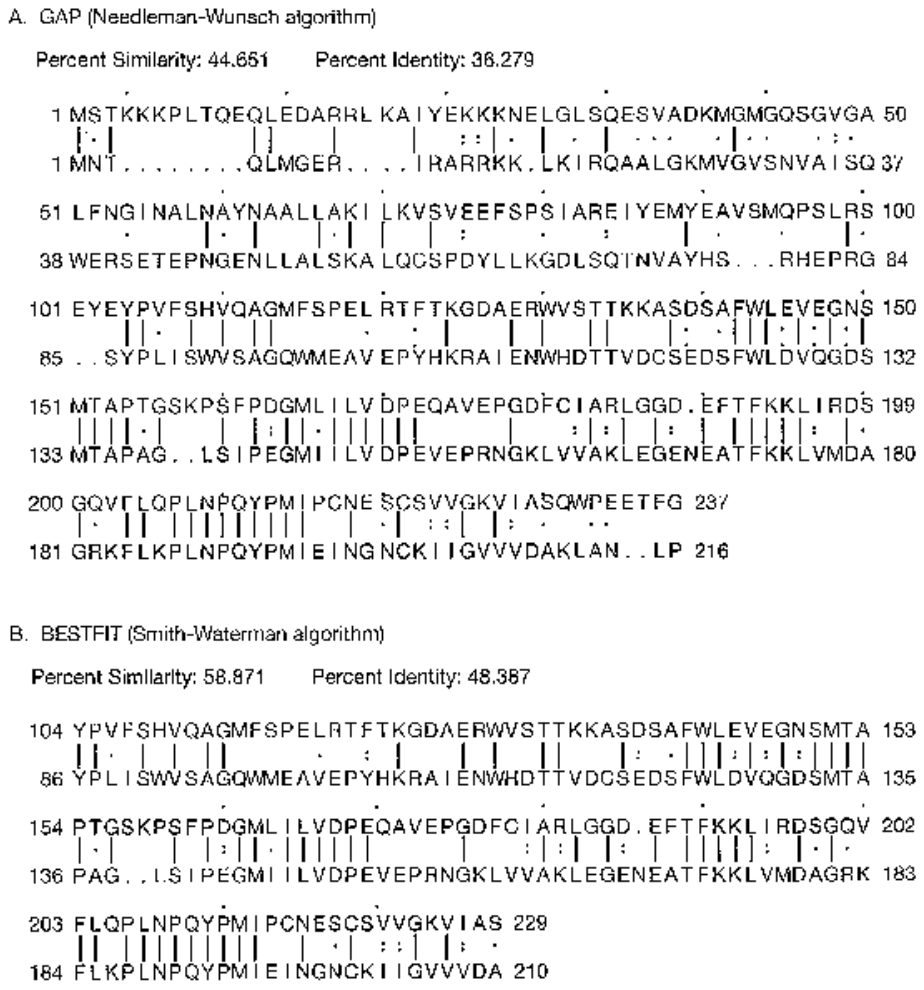


		H	E	A	G	A	W	G	H	E	E
	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	5	0	**5**	0	0	0	0	0
W	0	0	0	0	2	0	**20** ←	**12** ←	4	0	0
H	0	10 ←	2	0	0	0	12	18	**22** ←	14 ←	6
E	0	2	16 ←	8	0	0	4	10	18	**28**	20
A	0	0	8	21 ←	13	5	0	4	10	20	27
E	0	0	6	13	18	12 ←	4	0	4	16	26

```
AWGHE
AW-HE
```

Figure 2.6 *Above, the local dynamic programming matrix for the example sequences. Below, the optimal local alignment, with score 28.*

Global/local comparison



A. GAP (Needleman-Wunsch algorithm)

Percent Similarity: 44.651 Percent Identity: 36.279

```
   1 MSTKKKPLTQEQLEDARRLKAIYEKKKNELGLSQESVADKMGMGQSGVGA 50
     |·|        ||   |    |  ::|· | · | ···  ·|··  ::· 
   1 MNT........QLMGER....IRARRKK.LKIRQAALGKMVGVSNVAISQ 37

  51 LFNGINALNAYNAALLAKILKVSVEEFSPSIAREIYEMYEAVSMQPSLRS 100
        ·    |· |   |·| |  | :    ·   :    | ·   ·   |·
  38 WERSETEPNGENLLALSKALQCSPDYLLKGDLSQTNVAYHS...RHEPRG 84

 101 EYEYPVFSHVQAGMFSPELRTFTKGDAERWVSTTKKASDSAFWLEVEGNS 150
        ||· | | ||     ·  : |   | |  ||   |: ·|||:|:|:|
  85 ..SYPLISWVSAGQWMEAVEPYHKRAIENWHDTTVDCSEDSFWLDVQGDS 132

 151 MTAPTGSKPSFPDGMLILVDPEQAVEPGDFCIARLGGD.EFTFKKLIRDS 199
     ||||·|   | |:||·||||||     |   :|:| |: | |||||: |·
 133 MTAPAG..LSIPEGMIILVDPEVEPRNGKLVVAKLEGENEATFKKLVMDA 180

 200 GQVFLQPLNPQYPMIPCNESCSVVGKVIASQWPEETFG 237
     |· || |||||||||  | ·| ::| |: ·  ··    
 181 GRKFLKPLNPQYPMIEINGNCKIIGVVVDAKLAN..LP 216
```

B. BESTFIT (Smith-Waterman algorithm)

Percent Similarity: 58.871 Percent Identity: 48.387

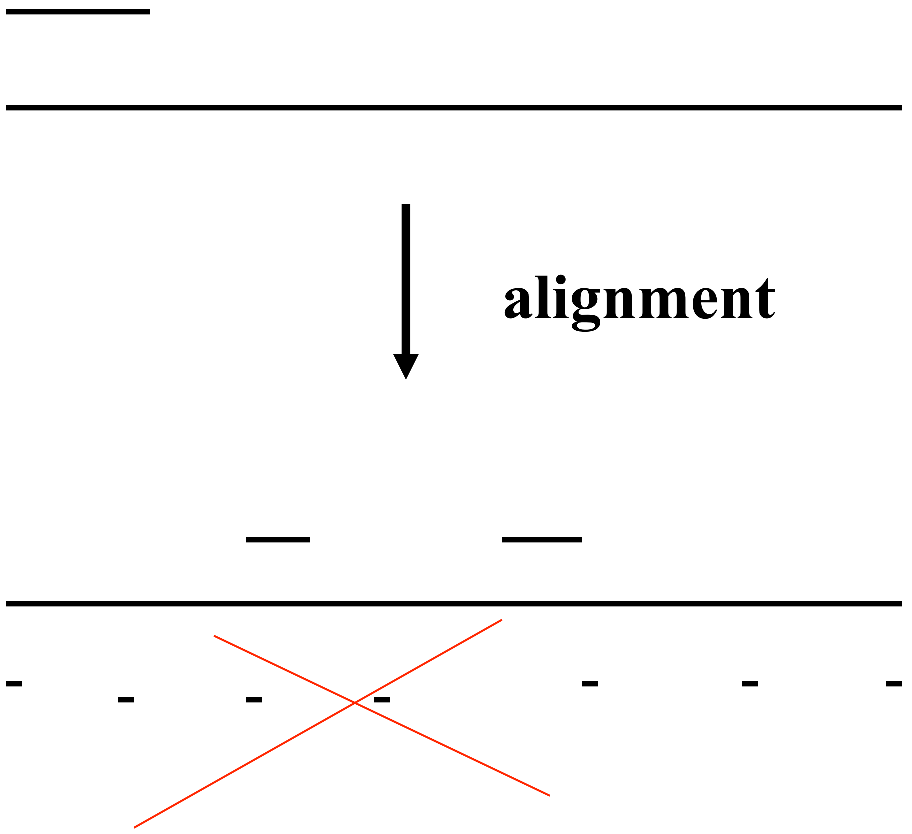
```
 104 YPVFSHVQAGMFSPELRTFTKGDAERWVSTTKKASDSAFWLEVEGNSMTA 153
     ||· | | ||     ·  : |   | |  ||   |: ·|||:|:|:||||
  86 YPLISWVSAGQWMEAVEPYHKRAIENWHDTTVDCSEDSFWLDVQGDSMTA 135

 154 PTGSKPSFPDGMLILVDPEQAVEPGDFCIARLGGD.EFTFKKLIRDSGQV 202
     |·|   | |:||·||||||     |   :|:| |: | ||||: |·|· 
 136 PAG..LSIPEGMIILVDPEVEPRNGKLVVAKLEGENEATFKKLVMDAGRK 183

 203 FLQPLNPQYPMIPCNESCSVVGKVIAS 229
     || |||||||||  | ·| ::| |: ·
 184 FLKPLNPQYPMIEINGNCKIIGVVVDA 210
```

Global alignment gives lower score.

Pairwise alignment: a combination of local and global alignments



•semiglobal

•global

Step 2 of FASTA

Locate best diagonal runs (gapless alignments) Give positive score for each hot spot

- Give negative score for each space between hot spots
- Find best scoring runs
- Score the alignments from the runs and find ones above a threshold. These are possible “sub-alignments”

Step 3 of FASTA

- Combine sub-alignments into one alignment.
- We need to solve a problem known as the chaining problem : find a collection of non-contradicting sub-alignments that maximize some scoring function.
- Problem reduces to a problem close to maximum common subsequence.