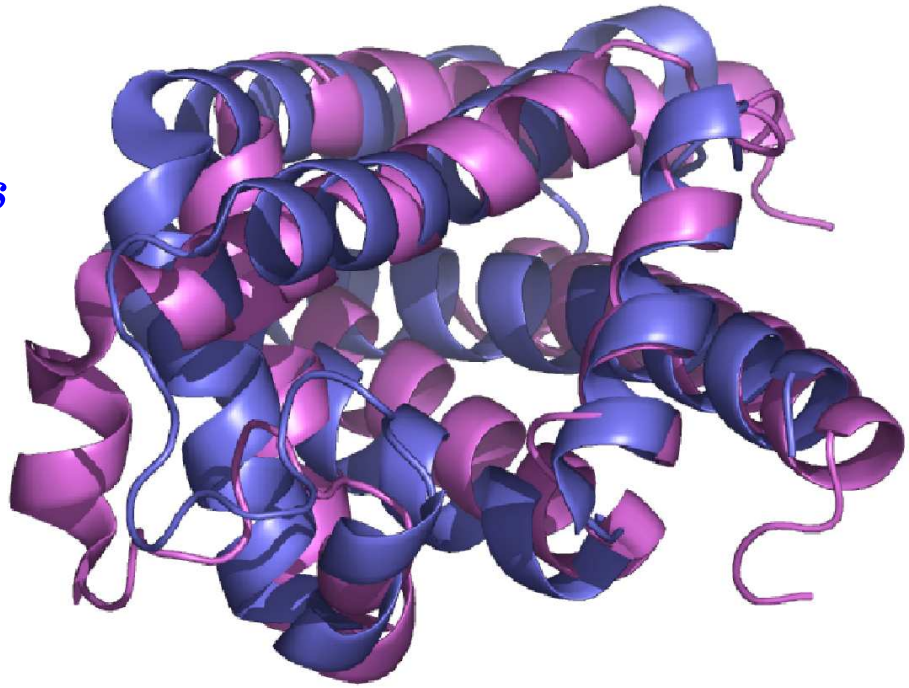
Pairwise Sequence Alignment

based on Ch. 2 from

Biological Sequence Analysis

by R. Durbin et al., 1998



[Sperm whale myoglobin (2lh7)

and Lupin leghaemoglobin (1mbd)]

Acknowledgements:

M.Sc. student Oana Răţoi
M.Sc. student Diana Popovici

Sequence alignments: two examples

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
            G+ +VK+HGKKV  A+++++AH+D++ +++++LS+LH  KL
HBB_HUMAN   GNPKVKAHGKKVLGAFSDGLAHLDNLKGTFATLSELHCDKL

HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHV---D--DMPNALSALSDLHAHKL
            ++ ++++H+ KV   + +A  ++           +L+ L+++H+ K
LGB2_LUPLU  NNPELQAHAGKVFKLVYEAAIQLQVTGVVVTDATLKNLGSVHVSKG
```

Above: human alpha globin vs. human beta globin: clear similarity

Below: human alpha globin vs. leghaemoglobin from yellow lupin: a plausible alignment

Informal definition: **Aligning two sequences (possibly extended with spaces/gaps '-') means putting their positions in a one-to-one correspondence; no gap-to-gap correspondence is allowed.**

In general — and so will be the case in this chapter — alignment techniques preserve the order of the residues within each sequence.

Plan

1. **Why** should one do sequence comparison?
2. **Scoring** systems used to rank alignments
3. **Dynamic programming alignment algorithms**
   - global alignment (Needleman-Wunsch)
   - local alignment (Smith-Waterman)
   - repeated matches alignment
   - overlap matches alignment
   - suboptimal alignment (Waterman-Eggert)
   - alignment with more complex models

◯ Statistical methods for
evaluating the significance of an alignment score

4. Optimised alignment algorithms
   - **Linear space** alignment with linear gaps
   - **Quadratic time** alignment with affine gaps
5. **Heuristic alignment algorithms:** FASTA, BLAST
6. **Discovery question:**
alignment algorithms with subquadratic time complexity?

1 Why should one do sequence comparison? First success stories

1984 (Russell Doolittle): similarity between the newly discovered ν-sis onco-gene in monkeys and a previously known gene PDGF (platelet derived growth factor) involved in controlled growth of the cell, suggested that the onco-gene does the right job at the wrong time.

1989: cystic fibrosis disease — an abnormally high content of sodium in the mucus in lungs:

Biologists suggested that this is due to a mutant gene (still unknown at that time, but narrowed down to a 1MB region) on chromosome 7.

Comparison with (then) known genes found similarity with a gene that encodes ATP, the adenosine triphosphate binding protein, which spans the cell membrane as an ion transport channel.

Cystic fibrosis was then proved to be caused by the deletion of a triplet in the CFTR (cystic fibrosis transport regulator) gene.

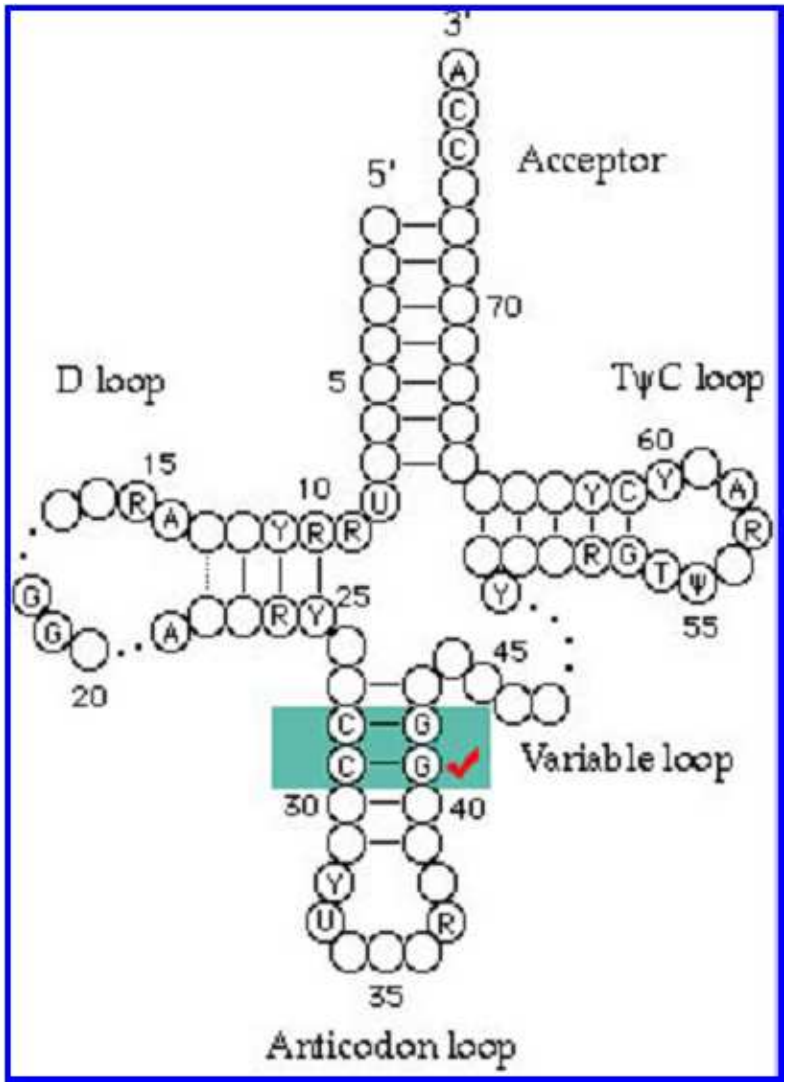
Note

Aligning RNA sequences, unlike DNA and proteins, requires taking into account long range dependencies between nucleotides.

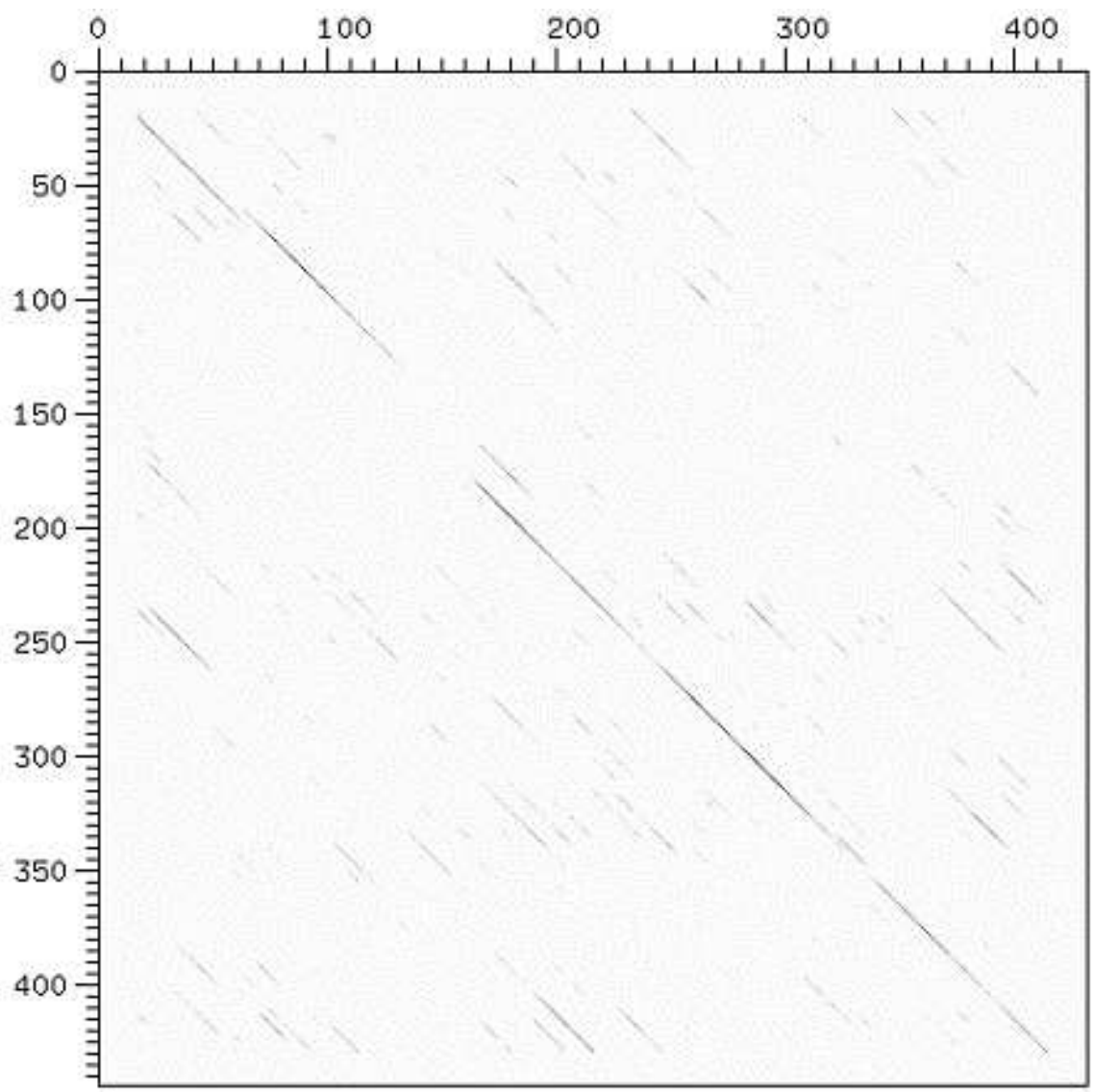
Ch. 10 from *Biological Sequence analysis*, Durbin et al., 1998, uses probabilistic context-free grammars (PCFGs) towards this aim.

The alignment algorithms presented here work mainly for DNAs and proteins.

For an alignment algorithm for RNAs based on dynamic programing, see for instance Blin & Touzet, "How to compare arc-annotated sequences: the alignment hierarchy", 2006.



A simple tool to visualise alignments: Dot Plots



A dot plot example

$y \setminus x$	H	E	A	G	A	W	G	H	E	E
P										
A			•		•					
W						•				
H	•							•		
E		•							•	•
A			•		•					
E		•							•	•

2 How to rank alignments: Scoring systems

Notations

- **Consider 2 sequences x and y of lengths n and m, respectively; x_i is the i-th symbol ("residue") in x and y_j is the j-th symbol in y.**
- **For DNA, the alphabet is $\{A, G, C, T\}$. For proteins, the alphabet consists of 20 amino acids.**

Defining alignment scores

The score of an alignment corresponds to the logarithm of the relative likelihood that the sequences are related, compared to being unrelated.

Two models:

- Random model (R): assume that each residue a occurs independently with a probability q_a
- Match model (M): assume that pairs of residues occur with a joint probability p_{ab}.

Basic mutational operations:

substitutions
insertions/deletions, referred as gaps or indels

Other mutational operations:

reversals, repetitions (not taken into account in this chapter)

Therefore, the score of an alignment will be computed as a sum of terms for each aligned pair of residues, plus terms for each gap.

How to derive substitution scores from a probabilistic model

The probability of aligning without gaps two sequences x and y having the same length:

in the random model: $P(x, y|R) = \prod_i q_{x_i} \prod_i q_{y_i}$

in the match model: $P(x, y|M) = \prod_i p_{x_i y_i}$

The *odds ratio:*

$$\frac{P(x, y|M)}{P(x, y|R)} = \frac{\prod_i p_{x_i y_i}}{\prod_i q_{x_i} \prod_j q_{y_i}} = \prod_i \frac{p_{x_i y_i}}{q_{x_i} q_{y_i}}$$

The *log-odds ratio:*

$$S = \sum_i s(x_i, y_i) \text{ where } s(a, b) = \log \frac{p_{ab}}{q_a q_b}$$

The scores $s(a, b)$ can be arranged in a matrix; the score matrix is also known as the substitution matrix.

Gap Penalties

Types of gap penalties:

linear score: $\gamma(g) = -gd$
affine score: $\gamma(g) = -d - (g-1)e$

where

$d > 0$ **is the *gap open* penalty,**
$e > 0$ **is the *gap extension* penalty, and**
$g \in N^*$ **is the length of the gap.**

Note:
Usually $e < d$, therefore long insertions and deletions will be penalised less than they would have been penalised by a linear gap cost.

Assuming that the length of a gap is independent of the residues it contains, it follows that

$$P(gap) = f(g) \prod_{i \ in \ gap} q_{x_i}$$

where q_a are the probabilities from the random model.

Therefore, the log-odds ratio for a gap is $\gamma(g) = log(f(g))$.

DNA substitution matrices

Simple:

	A	C	G	T
A	1	−1	−1	−1
C	−1	1	−1	−1
G	−1	−1	1	−1
T	−1	−1	−1	1

Used in genome alignments:

	A	C	G	T
A	91	−114	−31	−123
C	−114	100	−125	−31
G	−31	−125	100	−114
T	−123	−31	−114	91

Acknowledgement: this is a slide from the Sequence Analysis Master Course, Centre for Integrative Bioinformatics, Vrije Universiteit, Amsterdam

Protein substitution matrices:

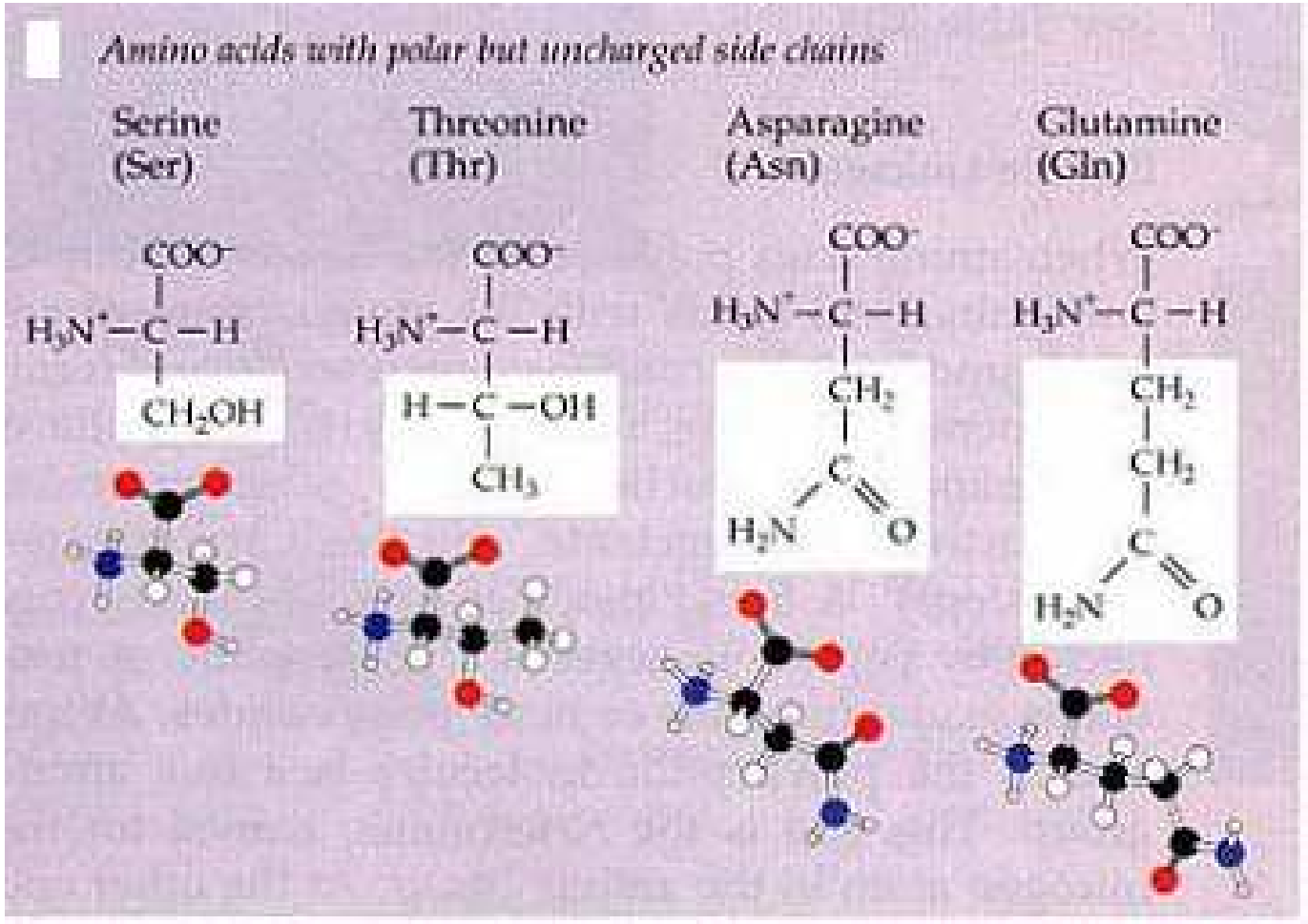
BLOSUM50

positive scores on diagonal (identities)
similar residues get higher (positive) scores
dissimilar residues get smaller (negative) scores

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5																			
R	−2	7																		
N	−1	−1	7																	
D	−2	−2	2	8																
C	−1	−4	−2	−4	13															
Q	−1	1	0	0	−3	7														
E	−1	0	0	2	−3	2	6													
G	0	−3	0	−1	−3	−2	−3	8												
H	−2	0	1	−1	−3	1	0	−2	10											
I	−1	−4	−3	−4	−2	−3	−4	−4	−4	5										
L	−2	−3	−4	−4	−2	−2	−3	−4	−3	2	5									
K	−1	3	0	−1	−3	2	1	−2	0	−3	−3	6								
M	−1	−2	−2	−4	−2	0	−2	−3	−1	2	3	−2	7							
F	−3	−3	−4	−5	−2	−4	−3	−4	−1	0	1	−4	0	8						
P	−1	−3	−2	−1	−4	−1	−1	−2	−2	−3	−4	−1	−3	−4	10					
S	1	−1	1	0	−1	0	−1	0	−1	−3	−3	0	−2	−3	−1	5				
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	2	5			
W	−3	−3	−4	−5	−5	−1	−3	−3	−3	−3	−2	−3	−1	1	−4	−4	−3	15		
Y	−2	−1	−2	−3	−3	−1	−2	−3	2	−1	−1	−2	0	4	−3	−2	−2	2	8	
V	0	−3	−3	−4	−1	−3	−3	−4	−4	4	1	−3	1	−1	−3	−2	0	−3	−1	5

Serine (S) and Threonine (T) have similar physico-chemical properties.
Aspartic acid (D) and Glutamic acid (E) have similar properties.

Therefore substitutions S/T and E/D occur reletively often;
they should result in scores that are only moderately lower than identities.



Estimation of the BLOSUM50 matrix

For each (multiple) alignment in the BLOCKS database, the sequences are grouped into clusters with at least 50% identical residues (for BLOSUM50)

Pairs of sequences are compared, and the observed pair frequencies are noted. (E.g., A aligned with A makes up 1.5% of all pairs, A aligned with C makes up 0.01% of all pairs, etc.)

Expected pair frequencies are computed from single amino acid frequencies. (E.g., $f_{A,C} = f_A \times f_C = 7\% \times 3\% = 0,21\%$)

For each amino acid pair, the substitution score is essentially computed as:

$$\textit{log}\ \frac{\text{Pair-freq (observed)}}{\text{Pair-freq (expected)}} \qquad s_{A,C} = \textit{log}\ \frac{0.01\%}{0.21\%} = -1.3$$

Acknowledgement: this slide, and the previous two slides are from

Anders Gorm Pedersen, Center for Biological Sequence Analysis, DTU (Technical University of Denmark)

Alignment algorithms: Which way to go?

(see Borodovsky & Ekisheva, pr. 2.5-2.7, pag. 29-38)

- The number of all the possible global alignments between two sequences of length n — satisfying the restriction: no gap in one seq. followed by gap in the other seq. — is exponential:

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

Note: To show this, use Stirling's formula: $x! \simeq \sqrt{2\pi} x^{x+\frac{1}{2}} e^{-x}$. (See *Biological Sequence Analysis*, Durbin et al., 1998, pag. 17, Ex. 2.2, 2.3, 2.4.)

Therefore, to find optimal alignments, it is not computationally feasible to enumerate all candidates.

- We can save the intermediate computations (corresponding to subsequence alignments) by using dynamic programming:
  - it reduces the number of computations
  - guarantees to find the best alignment.

- Heuristic methods have been also developed to perform the same search, they can be very fast, but they make additional assumptions and don't guarantee to find the best match for some sequence pairs.

3 Dynamic programming alignment algorithms

Important Remark

- The scoring scheme was introduced as a log-odds ratio, and better alignments will have a higher score. Therefore, the purpose is to maximize the score in order to find the optimal alignment.
- Sometimes scores are assigned by other means as *costs* or *edit distances*, when a minimum cost of an alignment is sought.
- Both approaches have been used in biological sequence comparison.
- Dynamic programming is applied to both cases: the difference is searching for a minimum or maximum score of the alignment.

The score matrix for the following examples

(from BLOSUM50)

	H	*E*	*A*	*G*	*A*	*W*	*G*	*H*	*E*	*E*
P	−2	−1	−1	−2	−1	−4	−2	−2	−1	−1
A	−2	−1	**5**	0	**5**	−3	0	−2	−1	−1
W	−3	−3	−3	−3	−3	**15**	−3	−3	−3	−3
H	**10**	0	−2	−2	−2	−3	−2	**10**	0	0
E	0	**6**	−1	−3	−1	−3	−3	0	**6**	**6**
A	−2	−1	**5**	0	**5**	−3	0	−2	−1	−1
E	0	**6**	−1	−3	−1	−3	−3	0	**6**	**6**

Note: **For all subsequent examples, the gap penalty $d = 8$ will be used.**

3.1 Global Alignment: Needleman–Wunsch Algorithm (1970)

Problem: **Find the optimal global alignment of two sequences $x = x_1 \dots x_n$ and $y = y_1 \dots y_m$, allowing gaps.**

Idea: **Use dynamic programming.**

Initialisation:

$F(0,0) = 0$, $F(i,0) = -id$, $F(0,j) = -jd$, **for** $i = 1, \dots, n$ **and** $j = 1, \dots, m$.

Recurrence:

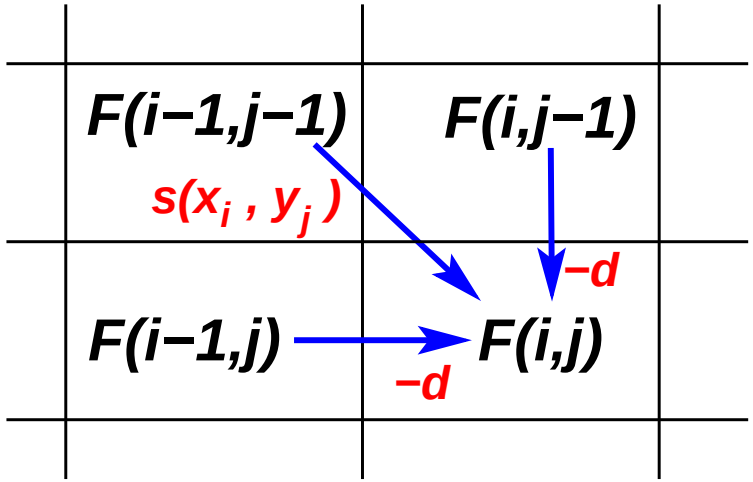
There are three ways in which the best score $F(i,j)$ of an alignment up to x_i, y_j could be obtained:

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

Fill in the matrix F from top left to bottom right.

Optimal score: $F(n,m)$.

Filling in the dynamic programming matrix



Important Note: (L. Ciortuz)

Unfortunately, throughout this chapter in *Biological Sequence Analysis*, Durbin et al., 1998,

the notation $F(i, j)$ is somehow misleading:
i is the column index, and j is the raw index.

Global Alignment: Example

Computing the dynamic programming matrix

$y \setminus x$		H		E		A		G		A		W		G		H		E		E
	0	$\leftarrow$	-8	$\leftarrow$	-16	$\leftarrow$	-24	$\leftarrow$	-32	$\leftarrow$	-40	$\leftarrow$	-48	$\leftarrow$	-56	$\leftarrow$	-64	$\leftarrow$	-72	$\leftarrow$	-80
	$\uparrow$	$\nwarrow$		$\nwarrow$		$\nwarrow$				$\nwarrow$								$\nwarrow$		$\nwarrow$	
P	-8		-2		-9	$\leftarrow$	-17	$\leftarrow$	-25	$\leftarrow$	-33	$\leftarrow$	-41	$\leftarrow$	-49	$\leftarrow$	-57	$\leftarrow$	-65	$\leftarrow$	-73
	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$		$\nwarrow$				$\nwarrow$											
A	-16		-10		-3		-4	$\leftarrow$	-12	$\leftarrow$	-20	$\leftarrow$	-28	$\leftarrow$	-36	$\leftarrow$	-44	$\leftarrow$	-52	$\leftarrow$	-60
	$\uparrow$		$\uparrow$		$\uparrow$	$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$									
W	-24		-18		-11		-6		-7	$\leftarrow$	-15		-5	$\leftarrow$	-13	$\leftarrow$	-21	$\leftarrow$	-29	$\leftarrow$	-37
	$\uparrow$	$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$		$\uparrow$	$\nwarrow$		$\nwarrow$						
H	-32		-14		-18		-13		-8		-9		-13		-7		-3	$\leftarrow$	-11	$\leftarrow$	-19
	$\uparrow$		$\uparrow$	$\nwarrow$				$\nwarrow$	$\uparrow$	$\nwarrow$		$\nwarrow$		$\uparrow$	$\nwarrow$		$\nwarrow$		$\nwarrow$		
E	-40		-22		-8	$\leftarrow$	-16		-16		-9		-12		-15		-7		3	$\leftarrow$	-5
	$\uparrow$		$\uparrow$		$\uparrow$	$\nwarrow$				$\nwarrow$		$\nwarrow$		$\nwarrow$			$\uparrow$		$\uparrow$	$\nwarrow$	
A	-48		-30		-16		-3	$\leftarrow$	-11		-11		-12		-12		-15		-5		2
	$\uparrow$		$\uparrow$	$\nwarrow$	$\uparrow$		$\uparrow$	$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$		$\nwarrow$	
E	-56		-38		-24		-11		-6		-12		-14		-15		-12		-9		1

Global Alignment: Example (cont'd)

Finding an optimal alignment

$y \setminus x$		H	E	A	G	A	W	G	H	E	E
	0	**−8**	**−16**	−24	−32	−40	−48	−56	−64	−72	−80
P	−8	−2	−9	**−17**	**−25**	−33	−41	−49	−57	−65	−73
A	−16	−10	−3	−4	−12	**−20**	−28	−36	−44	−52	−60
W	−24	−18	−11	−6	−7	−15	**−5**	**−13**	−21	−29	−37
H	−32	−14	−18	−13	−8	−9	−13	−7	**−3**	−11	−19
E	−40	−22	−8	−16	−16	−9	−12	−15	−7	**3**	−5
A	−48	−30	−16	−3	−11	−11	−12	−12	−15	**−5**	2
E	−56	−38	−24	−11	−6	−12	−14	−15	−12	−9	**1**

$$H\ E\ A\ G\ A\ W\ G\ H\ E - E$$
$$-\ -\ P - A\ W - H\ E\ A\ E$$

3.2 Local Alignment

Smith–Waterman Algorithm (1981; Gotoh, 1982)

Problem: **Find the best local alignment of two subsequences of x and respectively y, allowing gaps.**

Idea: **Modify the global alignment algorithm in such a way that it will end a local alignment whenever the score becomes negative.**

Initialisation:

$F(0,0) = 0$, $F(i,0) = F(0,j) = 0$, **for all** $i = 1, \ldots, n$ **and** $j = 1, \ldots, m$.

Recurrence:

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

Fill in the matrix F from top left to bottom right.

Optimal score: $\boldsymbol{max}_{i,j} F(i,j)$.

Note

For the local alignment algorithm to work,

- some $s(a, b)$ must be positive,
  otherwise the algorithm will not find any alignment at all;
- the expected score of a random alignment should be negative,
  otherwise

  long matches between unrelated sequences will have high scores, just based on their length, and
  a true local alignment would be masked by a longer but incorrect one.

  More concretely:
  In the case of ungapped alignments,
  assuming that subsequent positions in the alignment are independent,
  the above condition amounts to $\sum_{a,b} q_a \, q_b \, s(a, b) < 0$.
  (See further comments in [Durbin et al, 1998], p. 24.)

Local Alignment: Example

$y \setminus x$		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

$A\ W\ G\ H\ E$

$A\ W\ -\ H\ E$

3.3 Repeated Match (Local) Alignment

Problem: Find one or more non-overlapping sections of one sequence in the other sequence. Restrict the search to those alignments whose score is higher than a given threshold T.

Assumption: All match scores are positives.

Initialisation: $F(0,j) = 0$ **for** $j = 0, \ldots, m$

Recurrence:

$$F(i,0) = max \left\{ \begin{array}{l} F(i-1,0), \\ F(i-1,j) - T, \text{ for all } j = 1, \ldots, m \end{array} \right.$$

$$F(i,j) = max \left\{ \begin{array}{l} F(i,0), \\ F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{array} \right\} \text{ for } j = 1, \ldots, m$$

Fill in the matrix F from top left to bottom right (by columns).

Finally, add $F(n+1,0)$, and compute it using the first recurrence relation from above. The optimal score is $F(n+1,0)$.

Note: Increasing T may exclude matches; decreasing it may lead to finding weaker ones.

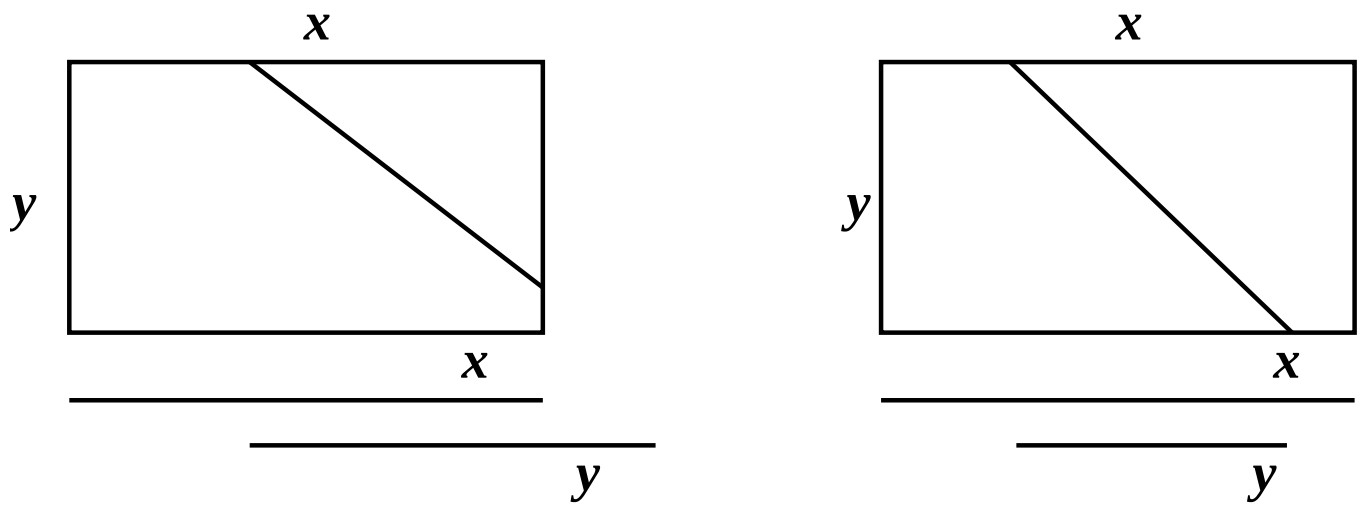
Repeated Match (Local) Alignment: Example

(threshold T = 20)

$y \setminus x$		H	E	A	G	A	W	G	H	E	E	
	0	0	0	0	**1**	1	1	1	1	3	**9** ←	**9**
P	0	0	0	0	**1**	1	1	1	1	3	9	
A	0	0	0	5	1	**6**	1	1	1	3	9	
W	**0**	0	0	0	2	1	**21** ←	**13** ←	5	3	9	
H	0	**10**	2	0	1	1	13	19	**23** ←	15	9	
E	0	2	**16** ←	8	1	1	5	11	19	**29**	21	
A	0	0	8	**21** ←	13	6	1	5	11	21	28	
E	0	0	6	13	18	12 ←	4	1	5	17	27	

$H\ E\ A\ G\ A\ W\ G\ H\ E\ E$
$H\ E\ A\ .\ A\ W\ -\ H\ E\ .$

3.4 Overlap Matches: Types of Configurations



This type of search is appropriate when a fragment of genomic DNA sequence is compared with another one, or with a larger chromosomal sequence.

Overlap Matches

Problem:

Find a maximum score alignment that starts on the left or top border of the matrix, and finishes on the right or bottom border.

Initialisation: $F(i,0) = F(0,j) = 0$ **for** $i = 0,\ldots,n$ **for** $j = 1,\ldots,m$

Recurrence: **the same as for global alignment**

Fill in the matrix F from top left to bottom right.

Optimal alignment score:

The maximum of the elements on the right or bottom border of the F matrix.

Traceback: **start at the maximum point and finish when the top or left border of the matrix is reached.**

Overlap Matches: Example

$y \setminus x$		H		E		A		G		A		W		G		H		E		E
	0		0		0		0		0		0		0		0		0		0	
		↖		↖		↖		↖		↖		↖		↖		↖		↖		
P	0		−2		−1		−1		**−2**		−1		−4		−2		−2		−1	
		↖		↖		↖		↖		↖		↖		↖		↖		↖		
A	0		−2		−3		4		−1		**3**		−4		−4		−4		−3	
		↖		↖		↑	↖		↖		↖								↖	
W	0		−3		−5		−4		1		−4		**18**	←	**10**	←	2	←	−6	
		↖					↖		↖				↑	↖		↖				
H	0		10	←	2	←	6		−6		−1		10		16		**20**	←	12	←
			↑	↖					↖				↑		↑	↖		↖		↖
E	0		2		16		8		0		−7		2		8		16		**26**	
		↖			↑	↖			↖					↖			↑		↑	↖
A	0		−2		8		21	←	13		5	←	−3		2		8		18	
		↖		↖		↑	↖		↖							↖		↖		↖
E	0		0		4		13		18		12	←	4	←	−4		4		14	

$$G\ A\ W\ G\ H\ E\ E$$
$$P\ A\ W\ -\ H\ E\ A$$

Repeat version of the overlap match algorithm

Use a treshold T to discard insignificant matches.

Initialisation: $F(0,j) = 0$ **for** $j = 0, \ldots, m$

Recurrence:

$$F(i,0) = max \left\{ \begin{array}{l} F(i-1,0), \\ F(i-1,m) - T, \end{array} \right.$$

$$F(i,j) = max \left\{ \begin{array}{l} F(i-1,j-1) + s(x_i, y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{array} \right\} \textbf{ for } j = 1, \ldots, m$$

The optimal alignment score:

$$F(n+1,0) = max \left\{ \begin{array}{l} F(n,0), \\ F(n,j) - T, \textbf{ for all } j = 1, \ldots, m \end{array} \right.$$

3.5 Suboptimal alignment

(from Durbin et al. 1998, Ch. 4.3, first section)

We are looking for a class of alignments with scores close to the optimal score, when it is suspected that more than one correct alignment might be present.

Sometimes minor variations at different places may provide an exponential number of such suboptimal alignments.

In particular, if there is a gap, then there is often the choice of where the gap should be placed.

For example, using BLOSUM50, d=12, e =2, the three alignments given below have very close global alignment scores: 3, 3, and respectively 6.

```
HBA_HUMAN   KVADALTNAVAHVD-----DMPNALSALSDLH
            KV   + +A  ++           +L+ L+++H
LGB2_LUPLU  KVFKLVYEAAIQLQVTGVVVTDATLKNLGSVH

HBA_HUMAN   KVADALTNAVAHVDDM-----PNALSALSDLH
            KV   + +A  ++           +L+ L+++H
LGB2_LUPLU  KVFKLVYEAAIQLQVTGVVVTDATLKNLGSVH

HBA_HUMAN   KVADALTNA-----VAHVDDMPNALSALSDLH
            KV   + +A     V  V     +L+ L+++H
LGB2_LUPLU  KVFKLVYEAAIQLQVTGVVVTDATLKNLGSVH
```

Finding distinct suboptimal local alignments

One possibility: use the *repeated matches* algorithm (Ch. 2). However, the best alignment may not even be present in the output!

Waterman – Eggert algorithm (1987)

Goal:

find the next best alignment that has no aligned residues in common with any previously determined alignment

Procedure:

after the top match is obtained, the standard DP matrix is recalculated, with an additional constraint during the recurrence step:

the cells corresponding to residue pairs contained in the best alignment are set to 0, preventing them from contributing to the next alignment

the second best alignment is obtained, after which the procedure can be repeated

Waterman – Eggert algorithm: Example

The standard local alignment matrix

$y \setminus x$		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

Waterman – Eggert algorithm: Example (cont'd)

$y \setminus x$		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		0		0		0	0	0	0
W	0	0		0		0		2		0		0		0	0	0	0
H	0	**10**	←	2		0		0		0		0		0	0	0	0
E	0	2		**16**	←	8		0		0		0		0	0	0	6
A	0	0		8		**21**	←	13		5		0		0	0	0	0
E	0	0		6		13		18		12	←	4		0	0	6	6

The best local match has been zeroed out so that the second best alignment can be obtained.

3.6 Dynamic programming with more complex gap models

Until now, the simplest gap model has been considered, the one that penalises additional gap steps as much as the first.

Given a general function $\gamma(g)$ to penalize gaps, the recurrence relations for all previous algorithms can be easily adjusted.

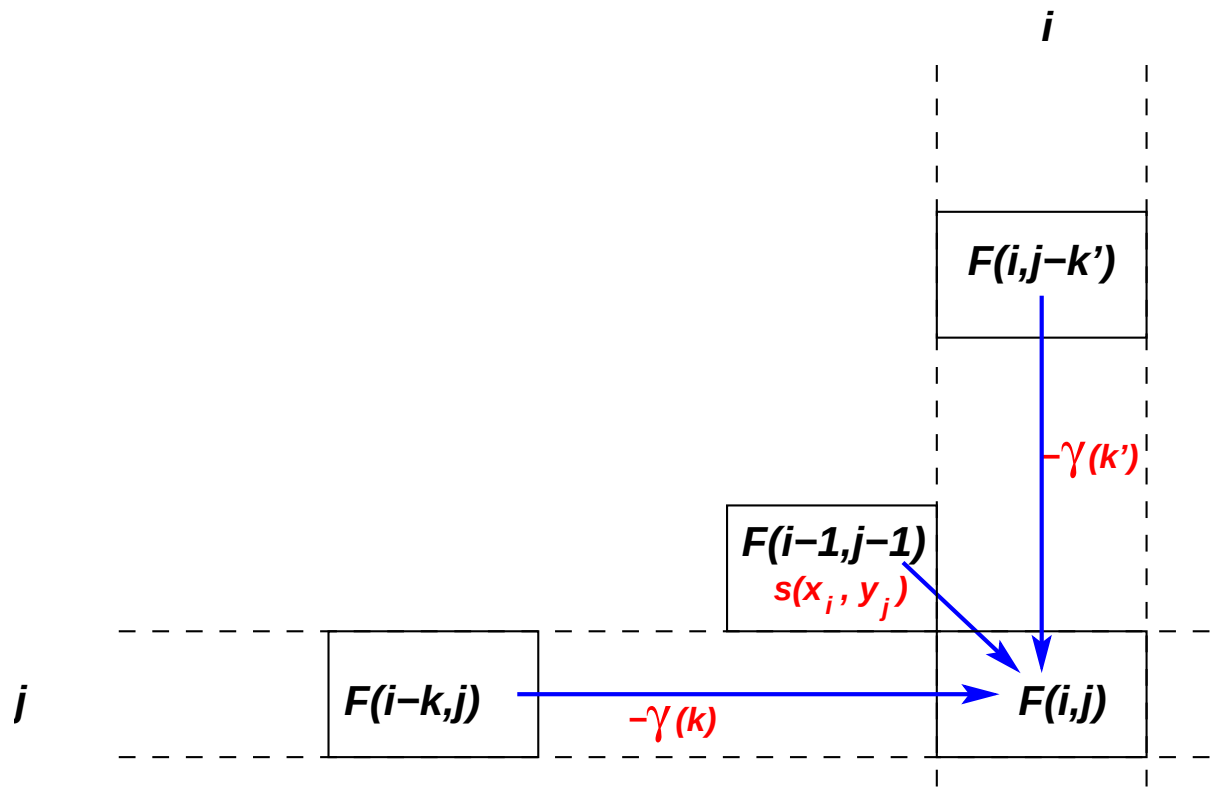
For instance, for global alignment:

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

Complexity: $O(n^3)$, because for each cell (i,j) there are $i+j+1$ possible precursors, not just three as before.

Affine gap penalty: $\gamma(g) = -d - (g-1)e$

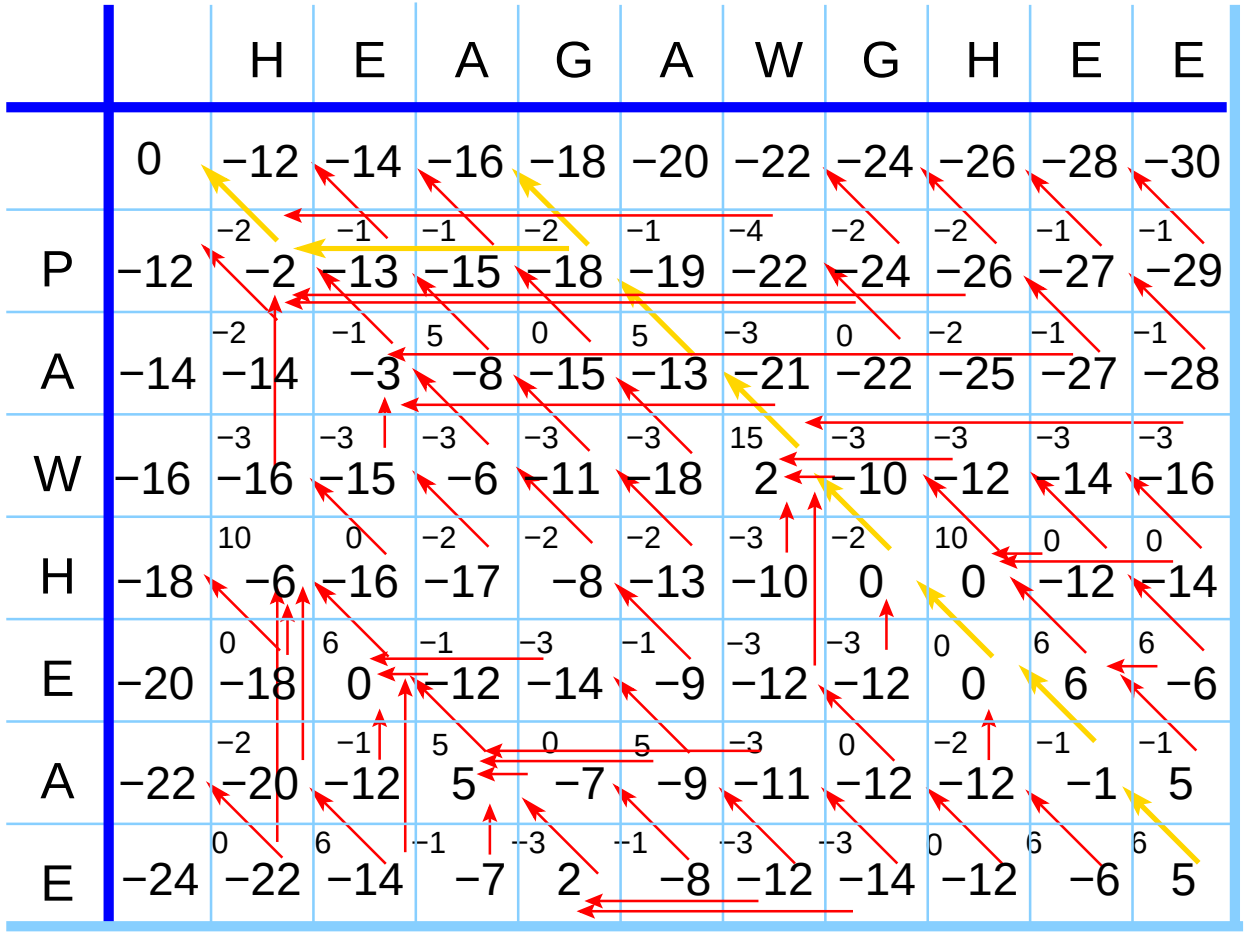
This diagram might help



Acknowledgement: this slide was adapted from Dr. Simon Colton, Imperial College of London

Global alignment with affine gap penalties:

Example ($d = 12$, $e = 2$) ...




```
HEAGAWGHEE
---PAWHEAE

HEAGAWGHEE
P---AWHEAE
```

5 Optimised alignment algorithms

5.1 Linear space alignment

(cf. [Durbin et al, 1998], building on ideas from [Hirschberg, 1975], [Myers & Miller, 1988], [Waterman, 1995])

The matrices $F(i, j)$ used so far have size nm.

We will show that there are techniques that give the optimal alignment in limited memory, $n + m$ instead of nm by doubling the time.

5.1.1:

If only the maximal score is needed:

Note that the recurrence relation for $F(i, j)$ is local, depending on entries only one row/column back, so the rest of the matrix can be ignored.

However, in this way, the path that provides the alignment will be lost.

5.1.2:

If the/an optimal alignment path π^* is also needed:

Assume that we search the optimal global alignment using linear gap scoring.

Idea for computing π^*:

Use *divide and conquer*

First step:

Let $u = \lfloor \frac{n}{2} \rfloor$.

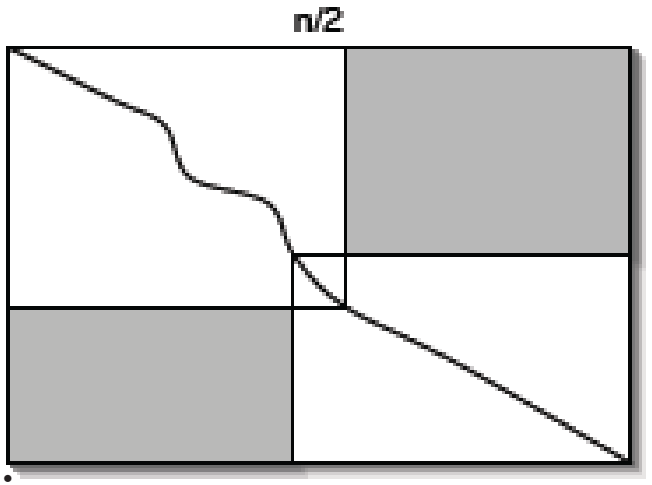
We will show (see the next slide) that we can find v such that the cell (u, v) is on the optimal alignment path.

The dynamic programming problem can be split into two parts: from $(0, 0)$ to (u, v) and from (u, v) to (n, m).

(LC Note: in certain cases, — i.e., if $v + 1 = w$, where $(u + 1, w)$ is on the optimal alignment path — from $(u + 1, v + 1)$ to (n, m).)

Next steps:

Fill in the whole alignment path by splitting it recursively.



Finding v, given u:

- **use the matrix c having n and m lines;**
- **for $i \geq u$ and $j = 0, \ldots, m$, when computing $F(i,j)$ compute also $c(i,j)$ using the following formula:**

  if $i = u$ then $c(i,j) = j$
  otherwise $c(i,j) = c(i',j')$, where (i',j') is the preceding cell to (i,j) from which $F(i,j)$ is derived.
- **every $c(i,j)$ satisfies the following property:**
  $(u, c(i,j))$ is on an optimal path from $(0,0)$ to (i,j)
- **the needed value is now stored in the bottom-right cell of the matrix: $v = c(n,m)$**

Note 1: For each back-trace path π that crosses the column u, the computation of the matrix c transmits into the rightmost column or the bottom row — by walking in the opposite direction to the arrows — the index of the (lowest) row where the path π crossed the column u and passed to the $u+1$ column. In particular, the cell $c(m,n)$ will receive this information for the optimal path $\pi^\star$.

Note 2: Only the right half of the matrix c needs to be computed. Moreover, since the calculus of $c(i,j)$ is local, we only need to maintain the previous row/column of the matrix c, like for $F(i,j)$.

Note 3: For a given u, the value of v can change at the subsequent recursion of the algorithm (see the previous slide). In order to retain the highest row where the path π crossed the column u, one would need another vector, d (of length n). By definition, $d(u)$ is the first computed value for v.

Note 4: It can be easily shown that the time needed for this version of the NW algorithm is the double of the time needed for the initial version.

Global alignment in linear space: Example

Computing the c matrix for the first step ($i = n = 10,\ j = m = 7,\ u = 5$)

The c values are written as subscripts to the $F()$ values. Note that the $F()$ values could be omitted; also, the arrows are drawn only to show how $c()$ values were computed.

$y \setminus x$	$_0$	H_1	E_2	A_3	G_4	A_5	W_6	G_7	H_8	E_9	E_{10}
$_0$	0 ←	-8 ←	-16 ←	-24 ←	-32 ←	-40_0 ←	-48_0 ←	-56_0 ←	-64_0 ←	-72_0 ←	-80_0
	↑ ↖	↖	↖		↖				↖	↖	
P_1	-8	-2	-9 ←	-17 ←	-25 ←	-33_1 ←	-41_1 ←	-49_1 ←	-57_1 ←	$-65_{0/1}$ ←	$-73_{0/1}$
	↑ ↖	↑ ↖	↖		↖						
A_2	-16	-10	-3	-4 ←	-12 ←	-20_2 ←	-28_2 ←	-36_2 ←	-44_2 ←	-52_2 ←	-60_2
	↑	↑	↑ ↖	↖	↖	↖	↖				
W_3	-24	-18	-11	-6	-7 ←	-15_3	-5_2 ←	-13_2 ←	-21_2 ←	-29_2 ←	-37_2
	↑ ↖	↖	↖	↖	↖	↖	↑ ↖	↖	↖		
H_4	-32	-14	-18	-13	-8	-9_4	-13_2	-7_2	-3_2 ←	-11_2 ←	-19_2
	↑	↑ ↖		↖	↑ ↖	↖	↖	↑ ↖	↖	↖	↖
E_5	-40	-22	-8 ←	-16	-16	-9_5	-12_4	-15_2	-7_2	3_2 ←	-5_2
	↑	↑	↑ ↖	↖	↖	↖	↖	↖	↑	↑ ↖	
A_6	-48	-30	-16	-3 ←	-11	-11_6	-12_5	-12_4	-15_2	-5_2	2_2
	↑	↑ ↖	↑	↑ ↖	↖	↖	↖	↖	↖	↖	↖
E_7	-56	-38	-24	-11	-6	-12_7	-14_6	-15_5	-12_4	-9_2	1_2

Global alignment in linear space:

Example (cont'd)

Therefore, after the first step we get $\pi^\star(5) = 2$.

Then we compute the solutions for the following two problems, yielding the values of $\pi^\star(2)$ and $\pi^\star(7)$:

		H	E	A	G	A
	0	$\leftarrow -8$	$\leftarrow -16$	$\leftarrow -24$	$\leftarrow -32$	$\leftarrow -40$
	$\uparrow$					
P	-8					
	$\uparrow$					
A	-16					

		W	G	H	E	E
	0	$\leftarrow -8$	$\leftarrow -16$	$\leftarrow -24$	$\leftarrow -32$	$\leftarrow -40$
	$\uparrow$					
W	-24					
	$\uparrow$					
H	-32					
	$\uparrow$					
E	-40					
	$\uparrow$					
A	-48					
	$\uparrow$					
E	-56					

And similarly the subsequent steps, thus computing the whole path $\pi^\star$.

5.2 Quadratic time alignment with affine gaps

Instead of a single value $F(i,j)$ for each pair (i,j), we can use multiple values:

- **$M(i,j)$, the best score up to (i,j) given that x_i is aligned to y_j**
- **$I_x(i,j)$, the best score given that x_i is aligned to a gap**
- **$I_y(i,j)$, the best score given that y_j is aligned to a gap.**

Recurrence relations for global alignment:

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

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

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

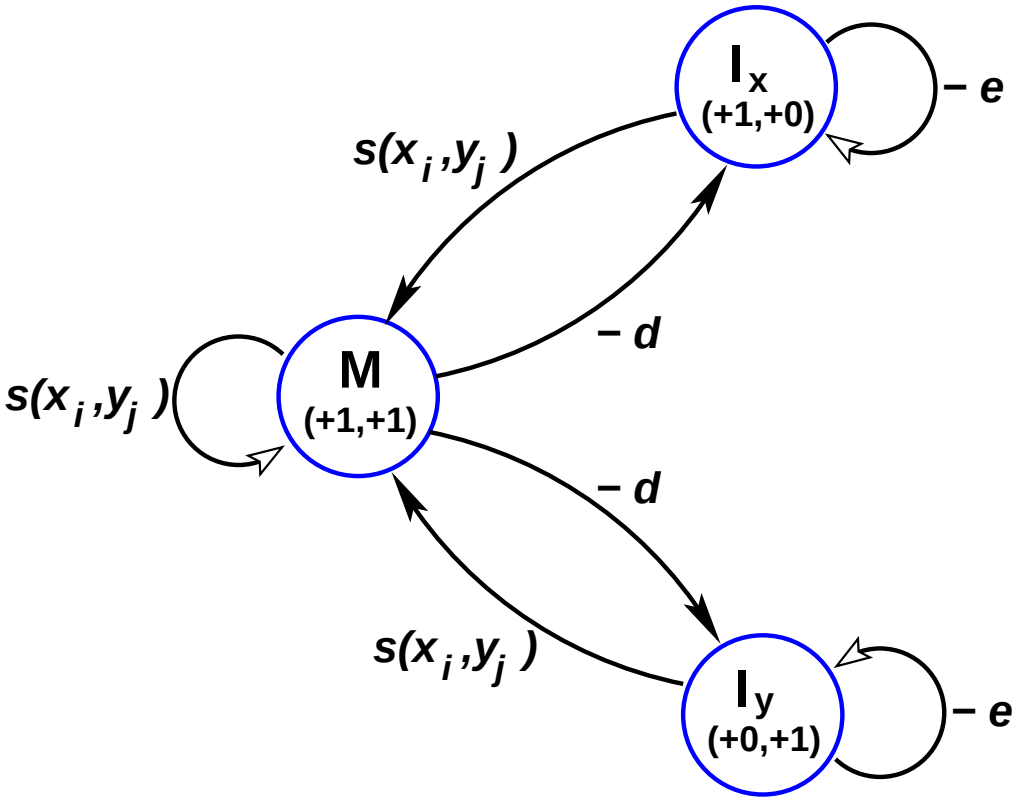
The link with the classical version of the Needleman-Wunsch algorithm:

$$F(i,j) = \boldsymbol{max} \{M(i,j),\ I_x(i,j),\ I_y(i,j)\}$$

LC Note: in the above formulas, it is understood that *max* is only taken among those (specified) variables which are defined.

Assumption for the previous recurrence relations:
A deletion will not be followed directly by an insertion.
(This is true if $-d-e$ is smaller than the lowest mismatch score.)

A finite state automaton that can describe the previous recurrence relations between M, I_x and I_y:

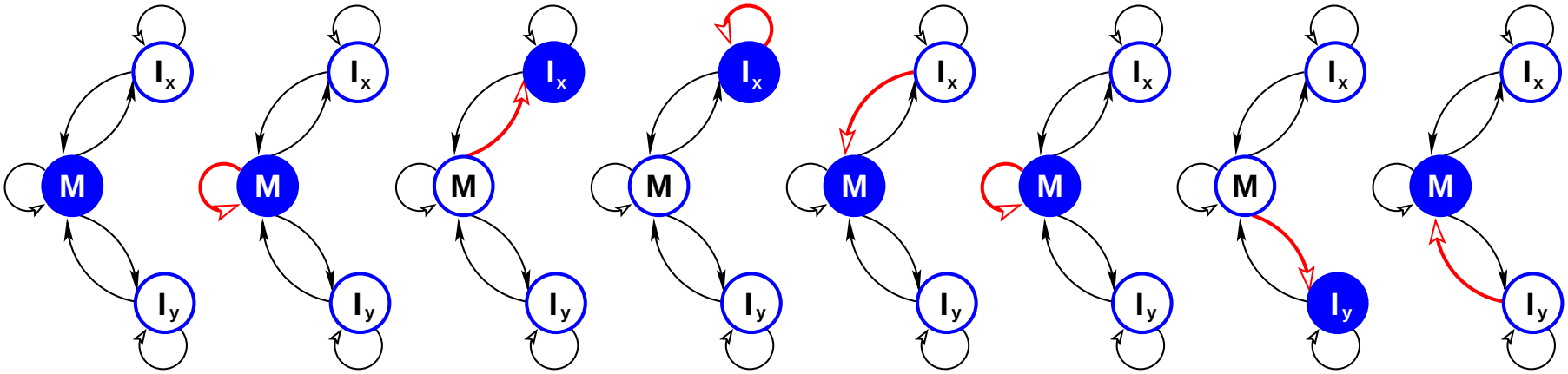


Each transition carries a score increment, and each state specifies a $\delta(i,j)$ pair.

An alignment corresponds to a path through the states.

An example of the state assignments for global alignment using the affine gap model

V	L	S	P	A	D	–	K
H	L	–	–	A	E	S	K



Two examples: global and respectively local alignments inspired from [Eddy, 2004] match: 5, mismatch:-2, $d = e =$ -6

$x \backslash y$	0	1 **T**	2 **G**	3 **C**
0	[0] $X:$ $Y:$ $M:0$	[-6] $X:$ $Y:-6$ $M:$	[-12] $X:$ $Y:-12$ $M:$	[-18] $X:$ $Y:-18$ $M:$
1 **T**	[-6] $X:-6$ $Y:$ $M:$	[5] $X:-12$ $Y:-12$ $M:5$	[-1] $X:-18$ $Y:-1$ $M:-8$	[-7] $X:-24$ $Y:-7$ $M:-14$
2 **T**	[-12] $X:-12$ $Y:$ $M:$	[-1] $X:-1$ $Y:-18$ $M:-1$	[3] $X:-14$ $Y:-7$ $M:3$	[-3] $X:-20$ $Y:-3$ $M:-3$
3 **C**	[-18] $X:-18$ $Y:$ $M:$	[-7] $X:-7$ $Y:-24$ $M:-14$	[-3] $X:-3$ $Y:-20$ $M:-3$	[8] $X:-9$ $Y:-9$ $M:8$

Note: Unfilled/undefined values for $i = 0$ or $j = 0$ must be taken equal to the defined ones.

Solution:

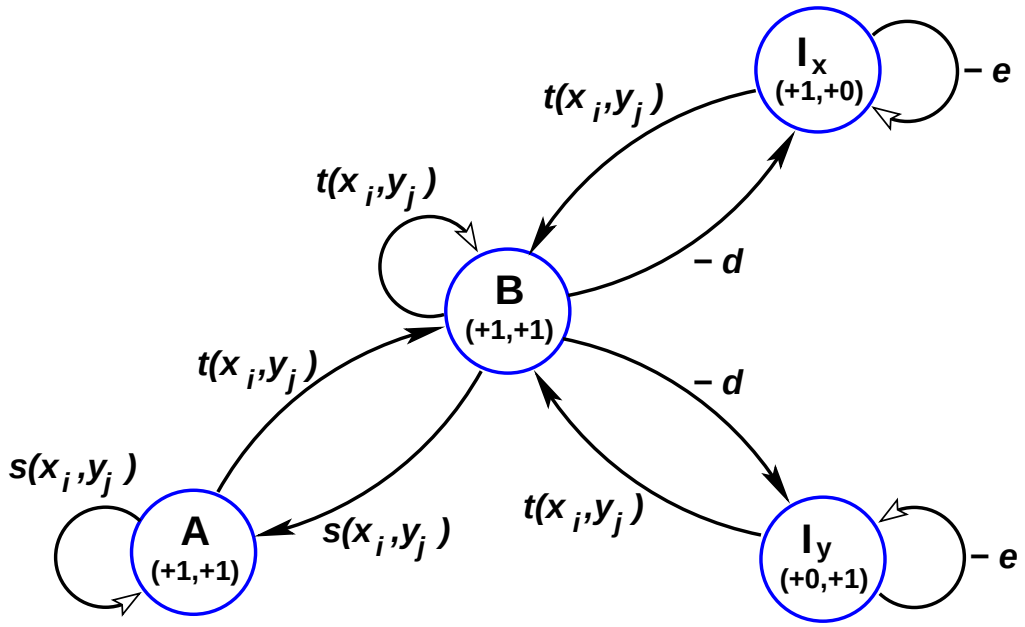
```
T G C
T C C
-----
5-2+5=8
```

The alignments corresponding to the two other components of the lower right case in the DP matrix:

```
T G C _          T G _ C
T C _ C          T C C _
-------          -------
5-2-6-6=-9       5-2-6-6=-9
```

$x \backslash y$	1 **T**	2 **G**	3 **C**	4 **T**	5 **C**	6 **G**	7 **T**	8 **A**
1 **T**	[5] $X:0$ $Y:0$ $M:5$	[0] $X:0$ $Y:0$ $M:0$	[0] $X:0$ $Y:0$ $M:0$	[5] $X:0$ $Y:0$ $M:5$	[0] $X:0$ $Y:0$ $M:0$	[0] $X:0$ $Y:0$ $M:0$	[5] $X:0$ $Y:0$ $M:5$	[0] $X:0$ $Y:0$ $M:0$
2 **T**	[5] $X:0$ $Y:0$ $M:5$	[3] $X:0$ $Y:0$ $M:3$	[0] $X:0$ $Y:0$ $M:0$	[5] $X:0$ $Y:0$ $M:5$	[3] $X:0$ $Y:0$ $M:3$	[0] $X:0$ $Y:0$ $M:0$	[5] $X:0$ $Y:0$ $M:5$	[3] $X:0$ $Y:0$ $M:3$
3 **C**	[0] $X:0$ $Y:0$ $M:0$	[3] $X:0$ $Y:0$ $M:3$	[8] $X:0$ $Y:0$ $M:8$	[2] $X:0$ $Y:2$ $M:0$	[10] $X:0$ $Y:0$ $M:10$	[4] $X:0$ $Y:4$ $M:1$	[0] $X:0$ $Y:0$ $M:0$	[3] $X:0$ $Y:0$ $M:3$
4 **A**	[0] $X:0$ $Y:0$ $M:0$	[0] $X:0$ $Y:0$ $M:0$	[2] $X:2$ $Y:0$ $M:1$	[6] $X:0$ $Y:0$ $M:6$	[4] $X:4$ $Y:0$ $M:0$	[8] $X:0$ $Y:0$ $M:8$	[2] $X:0$ $Y:2$ $M:2$	[5] $X:0$ $Y:0$ $M:5$
5 **T**	[5] $X:0$ $Y:0$ $M:5$	[0] $X:0$ $Y:0$ $M:0$	[0] $X:0$ $Y:0$ $M:0$	[7] $X:0$ $Y:0$ $M:7$	[4] $X:0$ $Y:1$ $M:4$	[2] $X:2$ $Y:0$ $M:2$	[13] $X:0$ $Y:0$ $M:13$	[7] $X:0$ $Y:7$ $M:0$
6 **A**	[0] $X:0$ $Y:0$ $M:0$	[3] $X:0$ $Y:0$ $M:3$	[0] $X:0$ $Y:0$ $M:0$	[1] $X:1$ $Y:0$ $M:0$	[5] $X:0$ $Y:0$ $M:5$	[2] $X:0$ $Y:0$ $M:2$	[7] $X:7$ $Y:0$ $M:0$	[18] $X:1$ $Y:1$ $M:18$

A more complex finite state automaton, with separate match states A and B for high (alignment) fidelity and respectively low fidelity regions.



Similarly, FS atomata can be built for alignments of transmembrane proteins with separate match states for intracellular, extracellular or transmembrane regions, or for other more complex scenarios.

5 Heuristic Alignment Algorithms

- The dynamic programming algorithms presented so far guarantee to find the optimal score according to the specified scoring scheme.
- Due to the rapid growth of biological databases, there is need for algorithms faster than straight dynamic programming.
- **Goal:** compute a small fraction of the cells in the dynamic programming matrix, but still look at all the highest scoring alignments.
- **Idea:** True match alignments are very likely to contain short stretches of identities or very high scoring matches. One can look for such stretches, use them as "seeds" for a good longer alignment.

FASTA

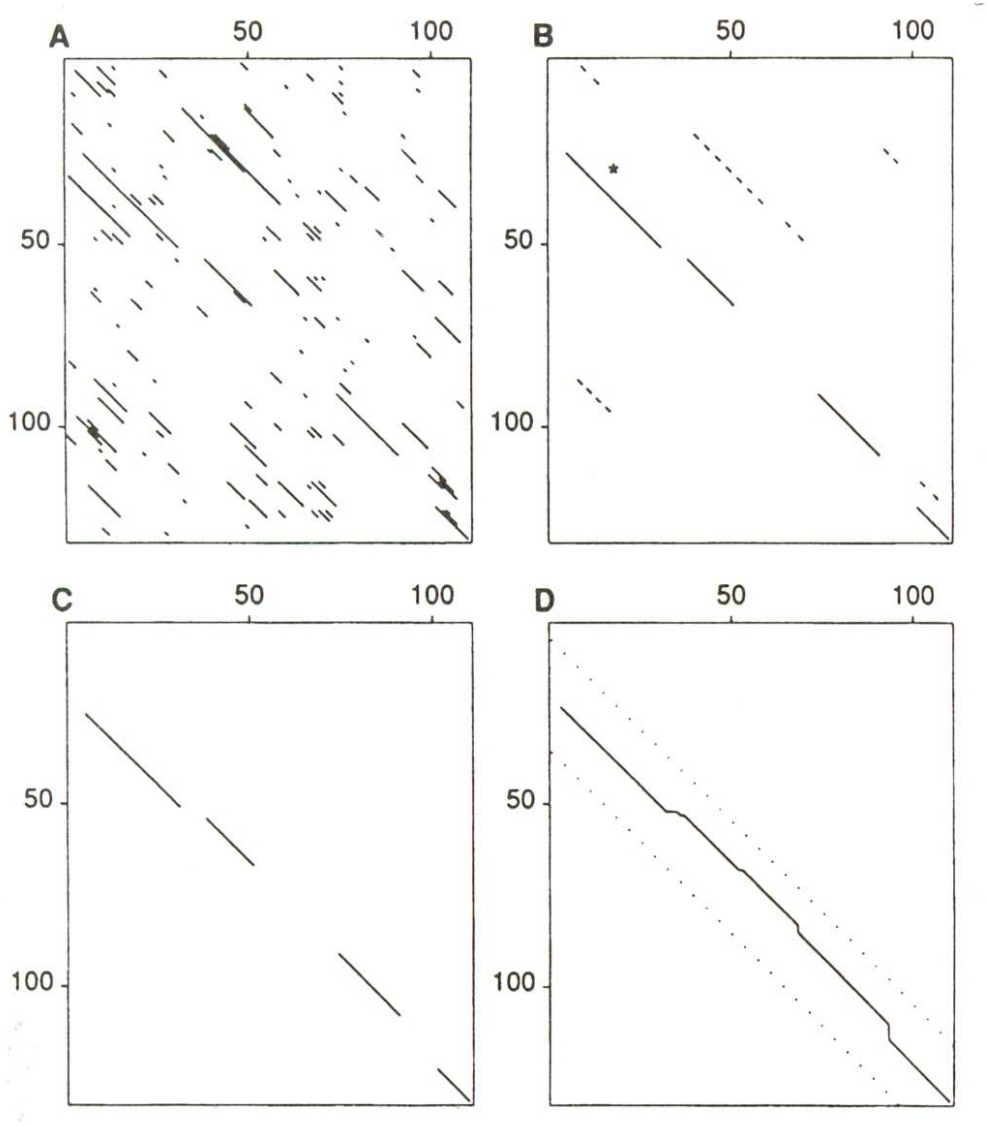
[Pearson and Lipman, 1988]

Goal: Find high scoring local alignments between two (DNA or protein) sequences.

Idea: Start from exact short word matches, go through maximal scoring ungapped extensions, to finally identify gapped alignments.

Steps:

1. Locate all identically matching words of length *ktup* between the two sequences. For proteins, *ktup* is 1 or 2, for DNA it may be 4 to 6.
2. Look for diagonals with many mutually supported word matches.
3. Test if the ungapped regions can be joined by a gapped region, allowing gap costs.
4. Realign the highest scoring candidates using a full dynamic programming algorithm restricted to a band around the candidate match.



Acknowledgement:
[**Pearson and Lipman, 1988**]

FASTA (cont'd)

Tradeoff between speed and sensitivity

through the choice of the parameter ***ktup***:

- higher values of ***ktup*** makes the search faster, but more likely to miss true significant matches;
- set $ktup = 1$ to achieve sensitivities close to those of full local dynamic programming.

BLAST — Basic Local Alignment Search Tool — at a glance

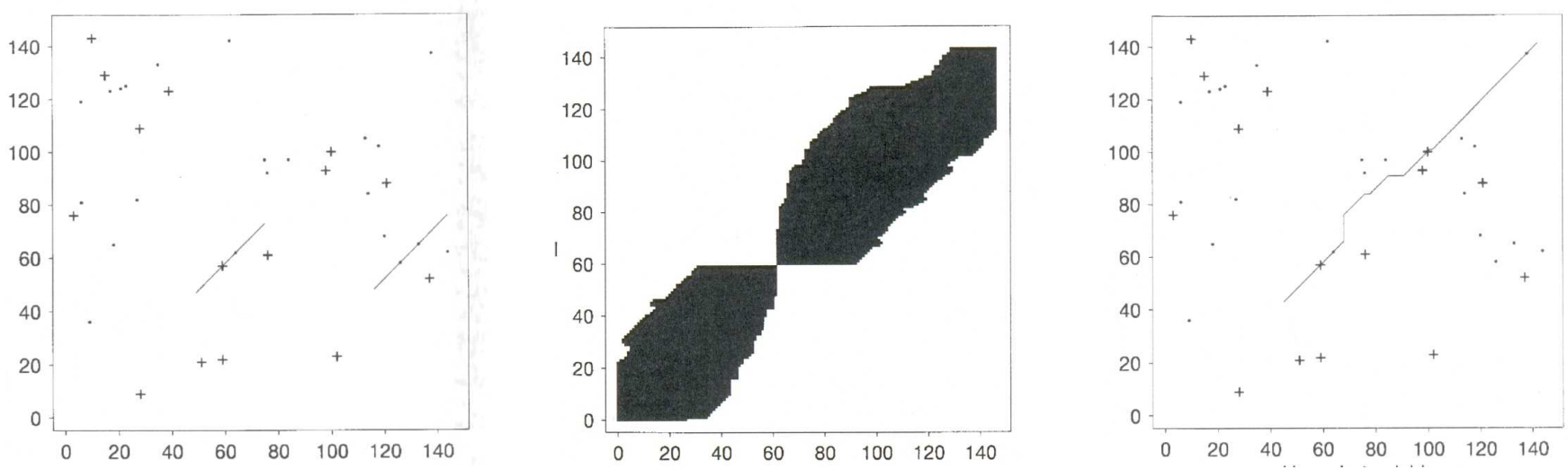
[Altschul *et al.*, 1990], [Altschul and Gish, 1996], [Altschul *et al.*, 1997]

Goal: Find high scoring local alignments between a "query" (DNA or protein) sequence and a "target" database.

Use fewer but better potential matches than FASTA.

Main steps:

1. make a list of all neighbourhood words of a fixed length k (by default 3 for proteins, and 11 for DNA), that would match the query sequence somewhere in the database with a score higher than a given threshold T;
2. use a look-up table to scan through the database and identify those sequences that contain exact matches of the neighbourhood words;
3. extend each match as a (firstly ungapped, secondly) gapped alignment in both directions, stopping at the maximum scoring extension (or, when the score falls bellow another given threshold X).
4. using the Altschul-Dembo-Karlin statistics (1991, 1993) compute the significance for the matches found in the database.




```
Leghemoglobin  43 FSFLKDSAGVVDSPKLGAHAEKVFGMVRDSAVQLRATGEVV--LDGKDGS------  90
                  F  L +   V+ +PK+ AH +KV          L + GE V  LD   G+
Beta globin    45 FGDLSNPGAVMGNPKVKAHGKKV----------LHSFGEGVHHLDNLKGTFAALSE  90

Leghemoglobin  91 IHIQKGVLDP-HFVVVKEALLKTIKEASGDKWSEELSAAWEVAYDGLATAI 140
                  +H  K  +DP +F ++   L+  +    G  ++ EL A+++    G+A A+
Beta globin    91 LHCDKLHVDPENFRLLGNVLVVVLARHFGKDFTPELQASYQKVVAGVANAL 141
```

Acknowledgement: [**Altschul et al., 1997**]

The BLAST Algorithm

- **Notes:**
  1. There are several implementations/variants of BLAST; here we limit our presentation to BLAST1 [Altschul *et al.*, 1990] and BLAST2 [Altschul *et al.*, 1997] papers.
  2. We will not discuss here PSI-BLAST (the profile-based version of BLAST).

- ***input:*** a query sequence Q and a database DB;

  Q and DB are considered here as being of the same type (either DNA or protein sequences); DB is already "indexed" upon k-mers, for k chosen as indicated below

- ***parameters:***

  s – the substitution score matrix;
  k – the dimension of "words" used for the indexation phase (*step1*);
  T – an alignment score threshold for selecting pairs of "neighbouring" words (*step2*);
  X (*step3*), and for BLAST2 also A (*step3*):
  two thresholds used for the extention and respectively the pairing of ungapped alignments ("high-scoring segment pairs" – HSPs);
  X_g for BLAST2 (*step4*) – a threshold used in the construction of gapped HSPs;
  d and e – the gap opening and extension penalties.
  E – a threshold to limit the number of delivered solutions (*step5*);

- ***output:***
  BLAST1: ungapped alignments (HSPs); BLAST2: gapped HSPs,
  together with their bit-score and statistical significance (e-value).

The BLAST *procedure*

***step0*:**

optionally mark "low complexity regions" in the query Q;

determine statistically (as a function of Q, ***DB*** and s) the threshold C – the maximum alignment length at which sequence similarities can show up just by chance;

[**LC: subsequently, the values of the parameters k and T are automatically selected by BLAST, again as a function of Q, *DB* and s.**]

***step1*:**

build an "index" of the query Q – a look-up table containing all k-mers in Q;

***step2*:**

for each k-mer α in Q, generate $G(\alpha)$, the set of all "neighbouring" words $u_\alpha \in \mathcal{A}^k$ such that $S(\alpha, u_\alpha) \geq T$, where $\mathcal{A}$ is the alphabet of Q and ***DB***;
in order to generate the neighbouring words, use Myer's sub-linear algorithm whose time complexity: $\mathcal{O}$(no. of generated neighbouring words);
for subsequent rapid retrieval, place all these neighbouring words in a tree – similar to suffix trees; see Aho-Corasick algorithm – which locates every occurrence of u_α in the ***DB***;

The BLAST *procedure* (cont'd)

***step3*:**

extend each pair (α, u_α) left- and right-wards until its score cannot increase anymore (or: falls more than X units below the score of the best short alignment already seen);

for BLAST2 use a "two-hit" strategy, compared to BLAST1's "one-hit": only those extended alignments ("high segment pairs", HSPs) which are found within a distance A of one another will be retained;

***step4*:**

for each HSP $(\bar{\alpha}, \bar{u}_\alpha)$ whose score exceeds the threshold C,

[BLAST2: apply a variant of Smith-Waterman algorithm, both left- and right-wards, starting from the middle of the highest-scoring k-mer inside the HSP, stopping the search when the computed score falls more than X_g below the best alignment found so far, and...]

compute the statistical significance (e-value) of the alignment produced (BLAST1: ungapped; BLAST2: gapped): $mn/2^S$, where m and n are the length of Q and respectively *DB*, and S is the alignment's bit-score;

***step5*:**

rank the results according to their bit-score or e-value.

BLAST flavours

BLASTN

Nucleotide query sequence
Nucleotide database

BLASTP

Protein query sequence
Protein database

BLASTX

Nucleotide query sequence
Protein database
Compares all six reading frames with the database

TBLASTN

Protein query sequence
Nucleotide database
On the fly six frame translation of database

TBLASTX

Nucleotide query sequence
Nucleotide database
Compares all reading frames of query with all reading frames of the database

FASTA vs BLAST

FASTA

Speed up searches by an order of magnitude compared to full Smith-Waterman

The statistical side of FASTA is still stronger than BLAST's

BLAST

Uses rapid word lookup methods to completely skip most of the database entries

Extremely fast:
two orders of magnitude faster than Smith-Waterman

Almost as sensitive as FASTA

Acknowledgement: this slide, and the next one are from Anders Gorm Pedersen
Center for Biological Sequence Analysis, DTU (Technical University of Denmark)

6 Discovery question

Are there

any non-heuristic alignment algorithms with subquadratic time complexity?

Short answer

First, study the case of a simpler problem, Longest Common Substring (LCS), which is a particular case for the problem of sequence (global) alignment (match 1, mismatch and indel 0).

Instead of computing score on characters, work on words of length t (t-mers). This is called *block alignment*. Consider $t = \log_2 n$.

Use look-up tables to precompute the result of aligning any pair of t-mers.

Generalise the global alignment algorithm so as to use given values for the left column and top line. Use this generalisation (with an off-set optimisation) for extending/generalising the look-up tables.

It works for LCS in $\mathcal{O}(n^2/log_2 n)$ time.

For details, see Jones & Pevzner, Ch. 7.

For the general sequence alignment problem...