Biological Sequence Analysis

Chapter 2. Pairwise alignment

Biointelligence Laboratory
School of Computer Sci. & Eng.
Seoul National University
Seoul 151-742, Korea

This slide file is available online at
http://bi.snu.ac.kr/

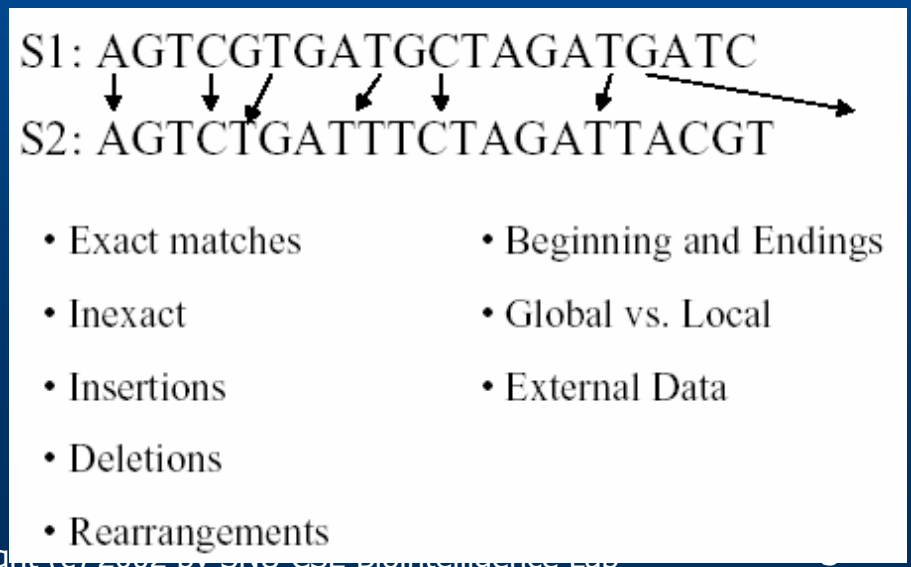
Outline

- Introduction
- Scoring model
- Alignment algorithms
- Dynamic Programming with complex models
- Heuristic Alignment algorithms
- Linear Space Alignments

- Score Significance
- Score parameter derivation from alignment data

2.1 Introduction

- Key issues
  - What sorts of alignments should be considered
  - Alignment ranking scoring system
  - Algorithm for optimal (or good) scoring alignments
  - Statistical methods for evaluating the significance of an alignment score
- Problem
  - Given two sequences from a specified alphabet
  - ➔ How do we find the best alignment ?



- Example – human alpha globin protein sequence

- Figure 2.1

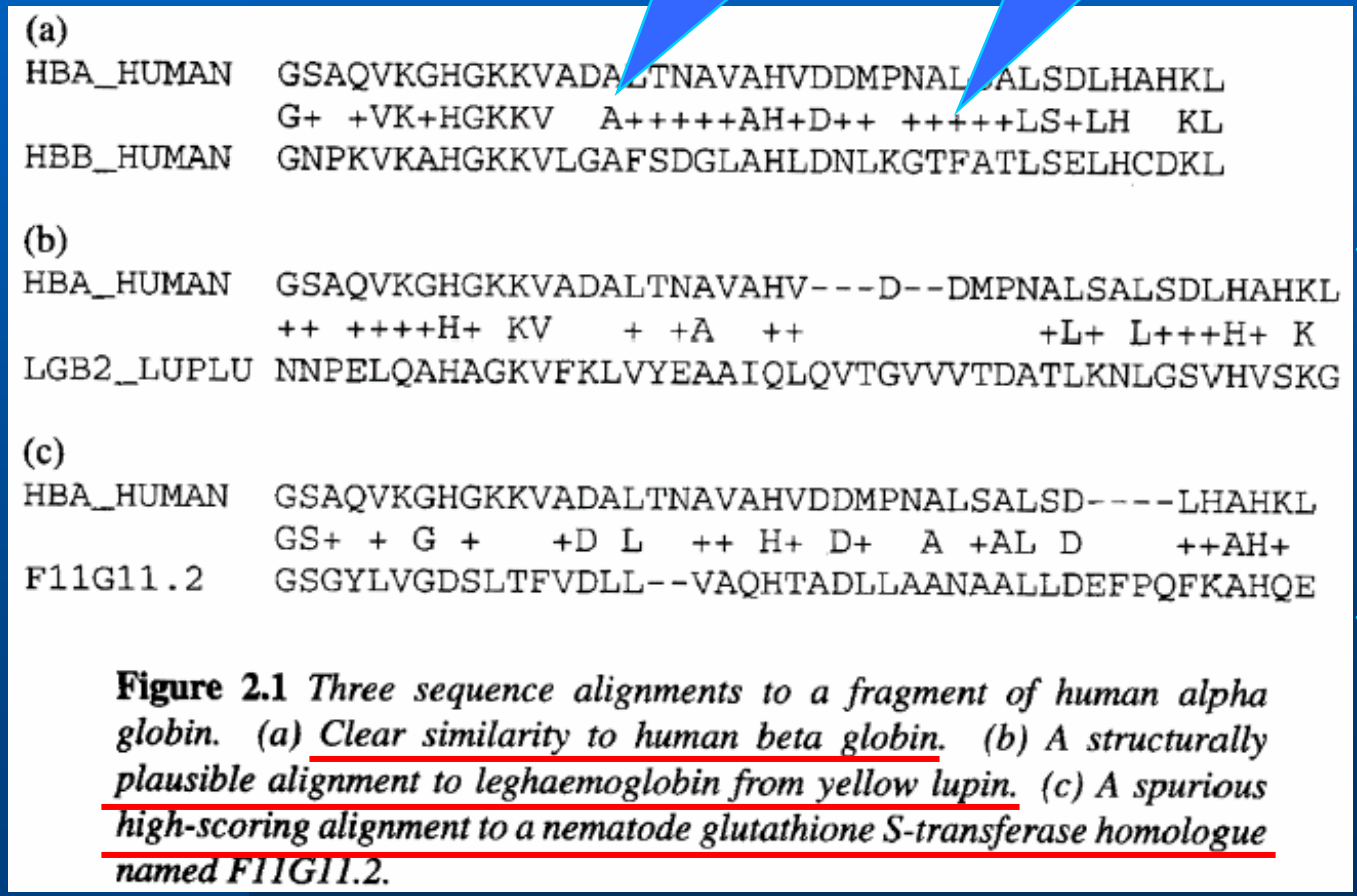


Figure 2.1 *Three sequence alignments to a fragment of human alpha globin. (a) Clear similarity to human beta globin. (b) A structurally plausible alignment to leghaemoglobin from yellow lupin. (c) A spurious high-scoring alignment to a nematode glutathione S-transferase homologue named F11G11.2.*

- Definitions (1/2)

alphabet = set of symbols

$\Sigma = \{A,C.G,T\}$

sequence = finite string of characters from an alphabet

ATGGCGT is a sequence from alphabet Σ

subsequence = sequence obtained by removing some characters froma sequence

AGGG is a subsequence of ATGGCGT

substring = subsequence of consecutive characters

TGG is a substring of ATGGCGT

prefix = substring containing first character of a string

ATG is a prefix of ATGGCGT

suffix = substring containing last character of a string

GT is a suffix of ATGGCGT

concatenation = result of appending two sequences

if u = ACGT, v = TTCAG then uv = ACGTTTCAG

- Definitions (2/2)

- Sequence comparison
  - Looking for evidence that they have diverged from a common ancestor by a process of *mutation* and *selection*.
- Basic mutation
  - Substitution: change residues in a sequence. (e.g. Insert / delete)
- *Gap*
  - Insertion: add some residues.
  - Deletion: remove some residues.
- Alignment score (+/- contribution to score terms)
  - A sum of terms for each aligned pair of residues + terms for each gap.
  - Positive(negative) score tem: from that (non-)conservative changes are expected to be observed more(less) frequently in real alignments than we expect by chance.

- Alignments

- Given two sequences: u = ATGGCT v = TGCTA
- Questions
  - Do they align?, What is the best alignment?
  - How do we score the alignment?
- ➔ We are allowed to inserts gaps ("-") in order to improve the alignment.
  - This makes better alignments but really increases the computations.
  - u' = ATGGCT
  - v' = - TG- CTA
- $|u'| = |v'|$ and there does not exist any position with gaps in u' and v'.

2.2 The scoring model

- "What's Important?"
- A good place to start:
  - score +1 for each match, -1 for each mismatch, -2 for a gap.
  - ➔ But, This doesn't tell the whole story. Some substitutions are allowed and are more likely.
    - E.g.) For a codon AAn it doesn't really matter what *n* is.
- Gap penalties
  - *Linear*: Each gap gets same penalty.
  - *Affine*: Add an additional penalty for gap length.
- Assumption
  - Mutations a
  independen
  - ➔ to reduc
  - For, structu

(a)

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

Score = 1-1-1-1 + … -1-1+1+1 = -5

- Substitution matrices (1/5)

- Define two sequences *x* and *y* with lengths of *n* and *m*.

x_i be the i-th symbol in x
y_i be the i-th symbol in y

- Two approaches:
  - *Random Model*: Assumes all elements are independent.
  - *Match Model*: Based on observed statistics and alignment frequencies.
- Let q_a be observed frequency of residue *a*.
- Let $p(a,b \mid M)$ be the probability of *a* aligned with *b* in the Match model :

$$p(a,b \mid M) = \frac{\text{number times } a \text{ aligned with } b}{\text{total number of aligned residues}}$$

- Substitution matrices (2/5)

- Random model: $p(x, y \mid R) = \prod_i q_{x_i} \prod_j q_{y_j}$

  Just the product

- Match model: $p(x, y \mid M) = \prod_i p_{x_i y_i}$

  With joint probability

- The ratio of these two likelihood is known as the *odd ratio*:

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

- In order to arrive at an additive scoring system ➔ take the logarithm of the ratio, known as the *log-odd* ratio:

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

  - ➔ log likelihood ratio of the residue pair (a,b) occurring as a aligned pair, as opposed to an unaligned pair.

- Substitution matrices (3/5)

- Equation $S = \sum_i s(x_i, y_i)$
  - Sum of individual scores $s(a,b)$ for each aligned pair of residues.
  - Score $s(a,b)$ can be arranged in a matrix.
  - $s(a_i, a_j)$:each component mean *i* th and *j* th amino acids.
  - ➔ Score Matrix:
    - Also known as a Substitution Matrix.
- For proteins, $s(a,b)$ elements can be arranged in a 20 x 20 matrix.
- An example of this is a *BLOSUM* 50 matrix.

- Substitution matrices (4/5)

- Example of substitution Matrix (BLOSUM 50)

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

Figure 2.2 *The* BLOSUM50 *substitution matrix. The log-odds values have been scaled and rounded to the nearest integer for purposes of computational efficiency. Entries on the main diagonal for identical residue pairs are highlighted in bold.*

- Substitution matrices (5/5)

- Calculation example
  - With BLOSUM 50 matrix

Self-study

(a)

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

What is the score of this alignment?

s(G,G) = 8 s(S,N) = 1 s(A,P) = -1 …

Score = 8+1-1+2+5+ … +2+5+10-1-1+6+5=130

- Gap Penalties

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

where

d = open gap penalty

e = gap extension penalty ($d > e$)

g = length of gap

  - $d > e$: allow long insertions and deletions to be penalized less than they would be by the linear gap cost.
    - ➔ desirable when gap of a few residues are expected almost as frequently as gaps of a single residue.
  - Assume that the prob. of a gap occurring at a particular site in a given sequence is the product of a function $f(g)$ of the length of the gap.
  - Combined probability of the set of inserted residues is
  - ➔ $P(\text{gap}) = f(g) \prod_{i \text{ in gap}} q_{x_i}$

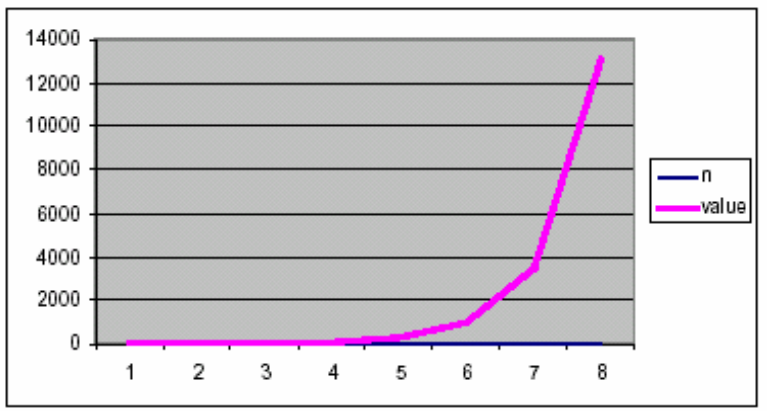
2.3 Alignment algorithms

- Scoring two sequences is very easy.
- Allowing for gaps makes it tough.
  - Or, if we consider local alignments between subsequences of two sequences
    - Where are the gaps?
    - How many?
- The computations goes as:

# of possible global alignments between two seq. of length n.

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

This is bad.



2.3 Alignment algorithms (2)

- Points
  - We only want to know the best alignment.
  - We do NOT want to perform all possible alignments in order to find the best alignment.
- *Dynamic programming* (also we have heuristic methods)
  - This will reduce the number of computations.
  - This is also a major component of this course.
    - Chapter 2 is base of other chapter – for DP.
  - Guaranteed to find the best alignment.
- Scores assigned by other means
  - *Costs*, *edit distances* : for this, we minimized the value.
  - ➔ both approaches have been used in the biological sequence comparison literature.

- Dynamic programming

- Making a New Matrix (by dynamic programming)
  - Given: *s* =HEAGAWGHEE and *b* =PAWHEAE.
  - What is optimal alignment?
  - We will build a new matrix *A*.

$$A(i,j)=\max\begin{cases} A(i-1,j)-2, & \text{align } a(i) \text{ with a gap} \\ A(i,j-1)-2, & \text{align } b(j) \text{ with a gap} \\ A(i-1,j-1)\pm 1, & \text{align } a(i) \text{ with } b(j) \end{cases}$$

	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**

With BLOSUM 50 Score Matrix

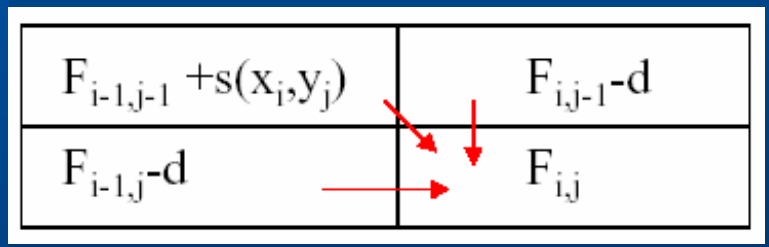
The positive values are in **bold** and we would like to use as many as these as possible. If the two sequences were exact then we would have a diagonal of positive numbers.

- Global alignment: Needleman & Wunsch

- Global Alignment algorithm
- Key idea
  - Build up an optimal alignment using solutions from smaller alignments.
- First step
  - Build an Alignment Matrix, F
  - $F_{i,j}$ is the score of the best alignment of $x_{1\cdots i}$ and $y_{1\cdots j}$.
  - To get $F_{i+1,j}$ we can use $F_{i,j}$ - recursive algorithm.

- Global alignment: Needleman & Wunsch

- Use 3 way method
  - Start $F_{0,0} = 0$
  - For an $F_{i,j}$ we can come from $F_{i-1,j-1}$, $F_{i,j-1}$, $F_{i-1,j}$.
    - If x_i aligns with y_j then $F_{i,j} = F_{i-1,j-1} + s(x_i, y_j)$
    - If x_i aligns with a gap then $F_{i,j} = F_{i-1,j} - d$
    - If y_j aligns with a gap then $F_{i,j} = F_{i,j-1} - d$



    - Pick the max one.

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

- Global alignment: Needleman & Wunsch

- Boundaries
  - The top row is not defined.
  - $F(i,0)$ is just an alignment with all gaps in *y* or *x*.
    - $F_{i,0} = -id,\ F_{0,j} = -jd$
- Traceback
  - $F_{n,m}$ is the final cell – the best score for an alignment.
  - It is also by definition the best score. We need to find the best path from *n*,*m* to 0,0.
  - "Traceback": Start with final cell go to $F_{i-1,j}$, $F_{i-1,j-1}$, $F_{i,j-1}$ whichever one it was that derived $F_{i,j}$.
  - There is only one path backwards.

$F(i, j-1)$ where $j = 0$

$F(i-1, j-1)$

- Global alignment: Needleman & Wunsch

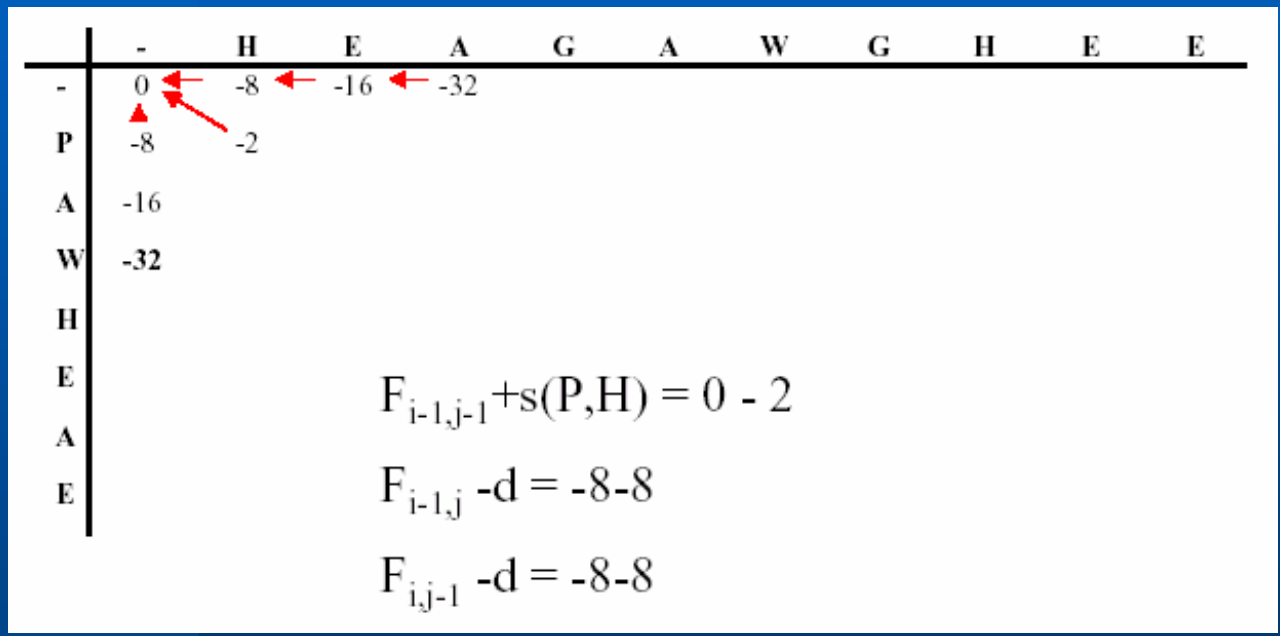
- Step 1

	-	H	E	A	G	A	W	G	H	E	E
-	0	-8	-16	-32							
P	-8										
A	-16										
W	**-32**										
H											
E											
A											
E											

  - Calculate alignment score for each match.

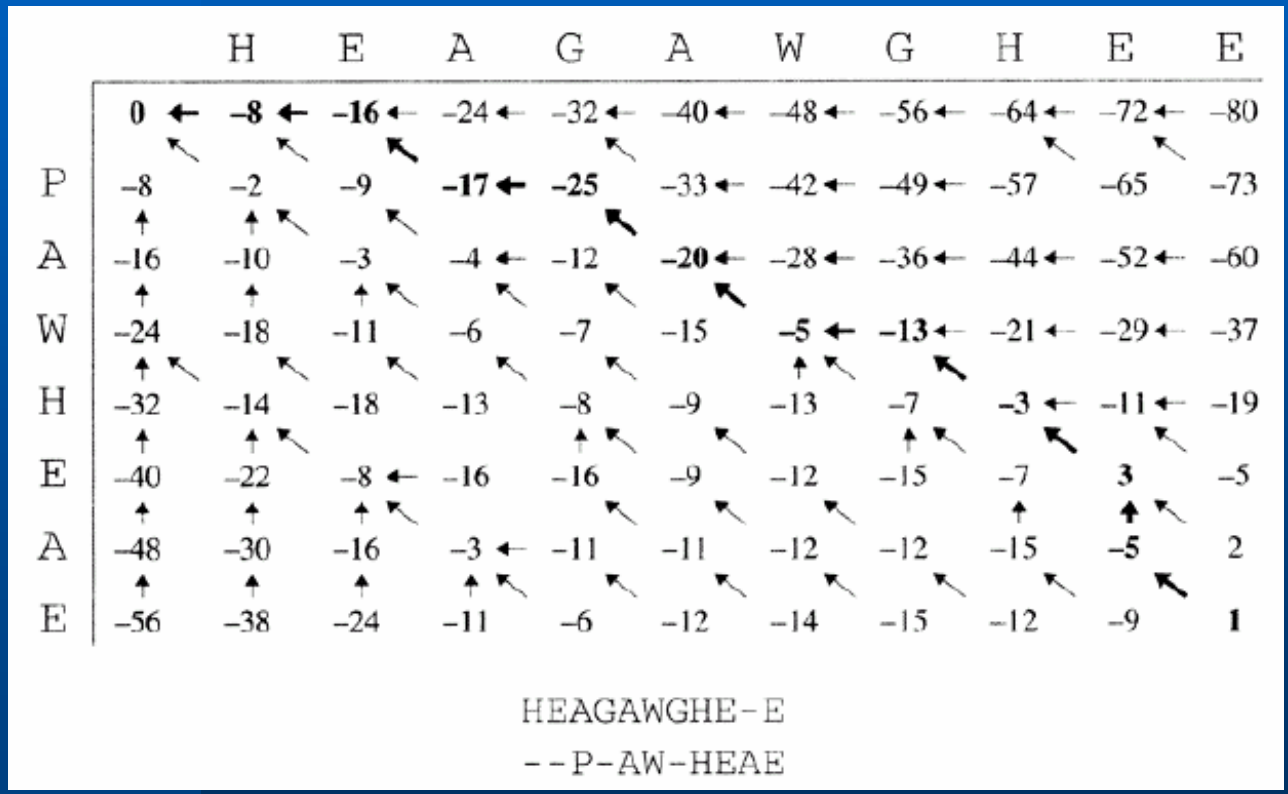
- Global alignment: Needleman & Wunsch

- Step 2



- Global alignment: Needleman & Wunsch

- Step 3



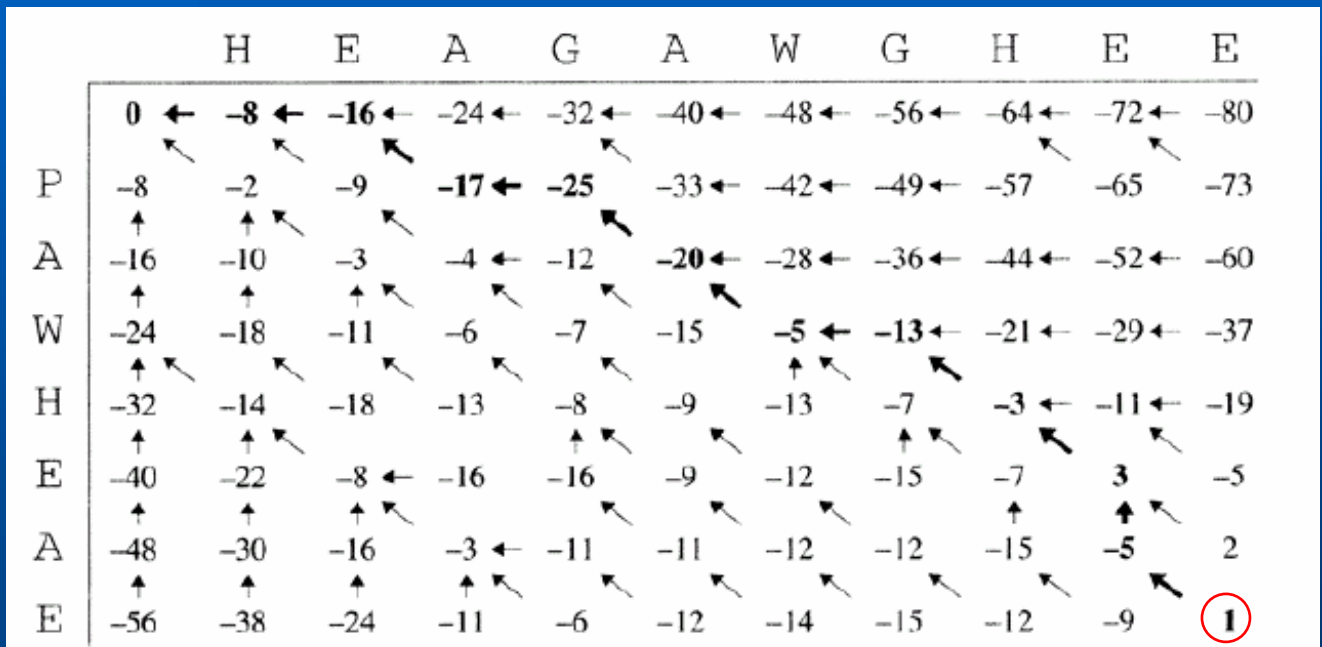
		H	E	A	G	A	W	G	H	E	E
	0	**−8**	**−16**	−24	−32	−40	−48	−56	−64	−72	−80
P	−8	−2	−9	**−17**	**−25**	−33	−42	−49	−57	−65	−73
A	−16	−10	−3	−4	−12	**−20**	−28	−36	−44	−52	−60
W	−24	−18	−11	−6	−7	−15	**−5**	**−13**	−21	−29	−37
H	−32	−14	−18	−13	−8	−9	−13	−7	**−3**	−11	−19
E	−40	−22	−8	−16	−16	−9	−12	−15	−7	**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**

HEAGAWGHE-E

--P-AW-HEAE

- Global alignment: Needleman & Wunsch

- Step 4



		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	–42	–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**

HEAGAWGHE-E

--P-AW-HEAE

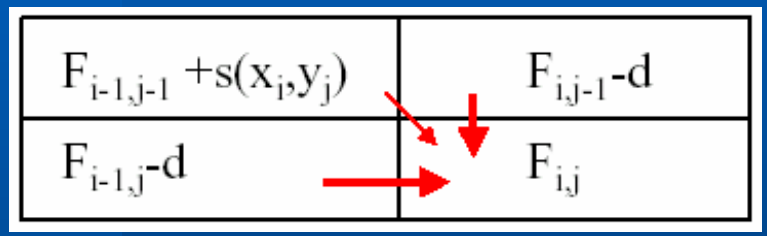
Start with last cell and follow the arrows back to home.
This is the best alignment.

- Global alignment: Needleman & Wunsch

- In the trace back process
  - Move back from the current cell (i,j) to the one of the cell (i-1,j-1), (i,j-1), (i-1,j).
    - From which $F(i, j)$ was derived.
  - Add a pair of symbols onto the from of the current alignment:
    - x_i, y_i : if the step was to $(i-1, j-1)$.
    - $x_i, -$: if the step was to $(i-1, j)$.
    - $-, y_i$: if the step was to $(i, j-1)$.
  - At the end ➔ we will arrive i=j=0. (start position)
    - See the figure of the previous page.

- Global alignment: Needleman & Wunsch

- Multiple Alignments
  - Sometimes we can get two traces providing the same value.



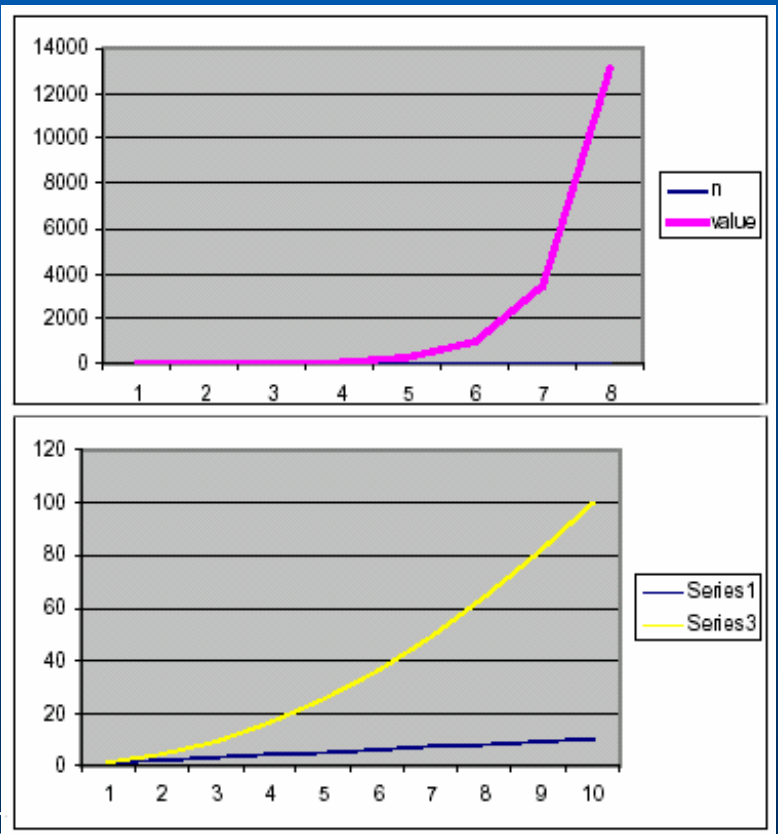
  - Which arrow do we use?
  - Doesn't really make any difference.
  - ➔ So, we have two possible paths. We keep both arrows and during traceback
  - An arbitrary choice will be okay.

- Global alignment: Needleman & Wunsch

- Store (n+1)(m+1) numbers
  - O(nm) time
  - O(nm) memory
- Complexity of search
  - Brute force method

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

  - Dynamic programming goes as O(mn).
  - Plotted here is O(nn).



- Local alignment: Smith & Waterman

- Smith-Waterman algorithm
  - Maybe one of the sequences is merely a sub-sequence of the other.
  - Looking or the best alignment between *subsequences* of *x* and *y*.
- Two differences (for global alignments)
  - In each cell in the table, extra possibility is added: case of F(i,j)=0
    - 0 corresponds to starting a new alignment.
    - Set the top row and left column with 0s, not $-id$ and $-jd$.

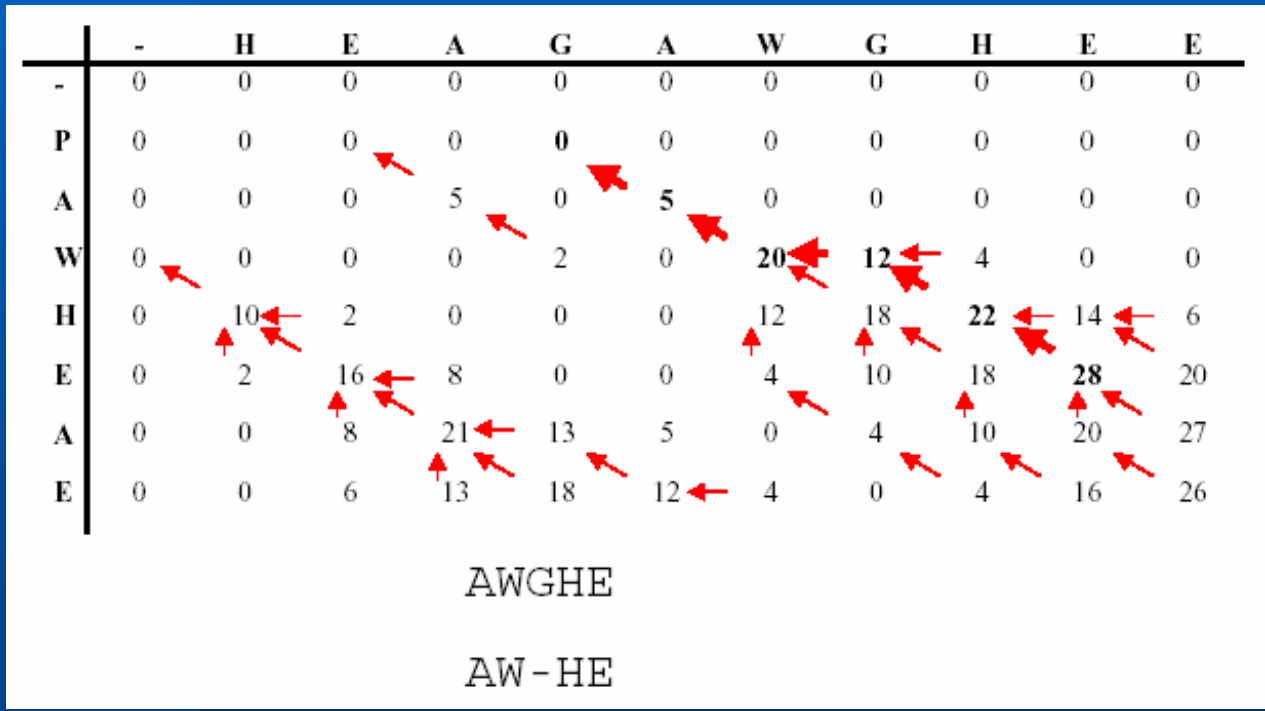
$$F_{i,j} = \max \begin{cases} 0 \\ F_{i-1,j-1} + s(a,b) \\ F_{i-1,j} - d \\ F_{i,j-1} - d \end{cases}$$

If all other options have value less than then F(i,j) take the value 0.

  - Alignment can end anywhere in the matrix.
    - Look for the highest value of F(i,j) over the whole matrix instead of taking the value in the bottom right corner $F_{i,j}$ for the best score.
  - Local alignment is a subset of the global alignment (not always).

- Local alignment: Smith & Waterman

- Trace



	-	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

- Local alignment: Smith & Waterman

- Requirements
  - The expected score for a random match must be negative
    - For the local alignment algorithm to work.
  - Otherwise,
    - Long matches between entirely unrelated sequences will have high scores
      - Just based on their length.
  - There must be some $s(a,b)$ greater than 0
    - Otherwise, the algorithm won't find any alignment at all.
- Very low speed.

- Local alignment: Repeated matches

- Repeated matches
  - One or both of the sequences are long
  - ➔ there are many different local alignments with a significant score.
    - E.g.) many copies of a repeated domain or motif in a protein.
  - We want to find all such matches.
- Algorithm for multiple match finding
  - Waterman & Eggert [1987] – See Chapter 4.
- Assume: only interested in matches scoring higher than some threshold *T*.
  - *T* is a threshold (in this example *T*=20).
  - Figure 2.7: there are two separate match regions.

- Local alignment: Repeated matches

- Use the matrix *F*.
- Use different recurrence.
- In the final alignment *x* will be partitioned
  - Regions that match parts of *y* in gapped alignment,
  - Regions that are unmatched.
- The score (positive, at least 0)
  - Standard gapped alignment score – threshold *T*.
- Start with $F(0,0) = 0$

(a)
$$F(i,0) = \max\begin{cases} F(i-1,0) \\ F(i-1,j) - T \end{cases}$$

$j = 1 \ldots m$

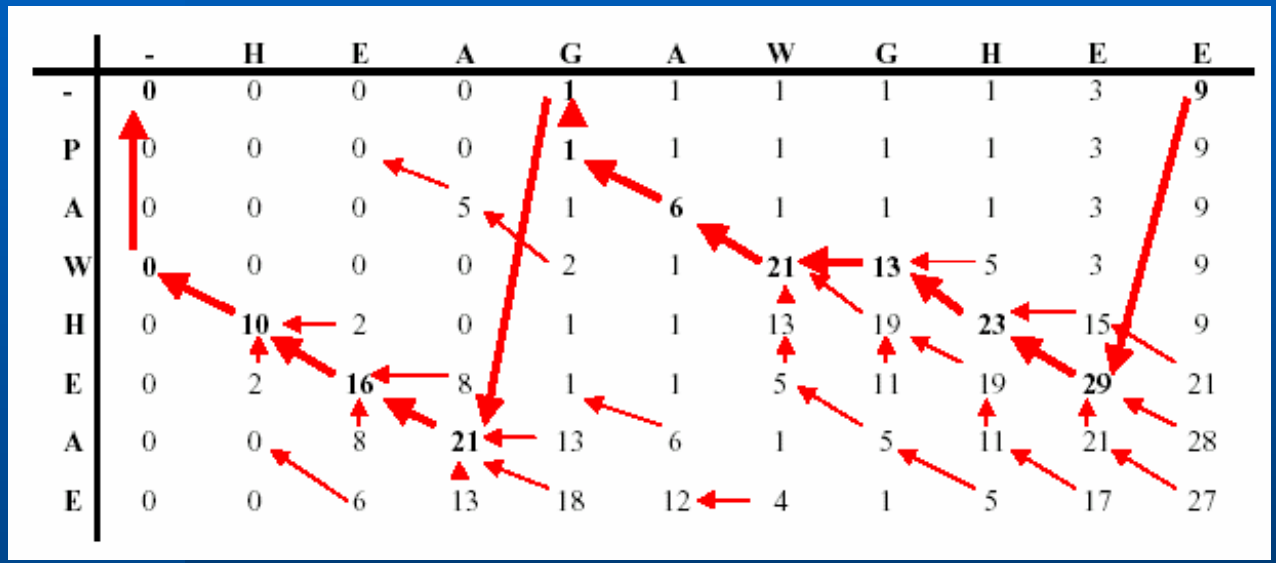
$$F(i,j) = \max\begin{cases} F(i,0) \\ F(i-1,j-1) + s(x_i, y_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \end{cases}$$
(b)

- Local alignment: Repeated matches

- Recurrence (a) handles (total score obtained by (a))
  - Unmatched regions.
  - End of matches.
    - Only allowing matches to end when they have score at least T.
- Recurrence (b) handles
  - Starts of matches and extensions.
- Individual match alignment
  - Can be obtained by tracking from cell (n+1,0) to (0,0).
  - Algorithm obtains all the local matches in one pass.
- Changing the value of T
  - Increasing: exclude matches.
  - Decreasing: split the matches as well as finding new weaker ones.

- Local alignment: Repeated matches

- Repeated Match Matrix



	-	H	E	A	G	A	W	G	H	E	E
-	**0**	0	0	0	**1**	1	1	1	1	3	**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

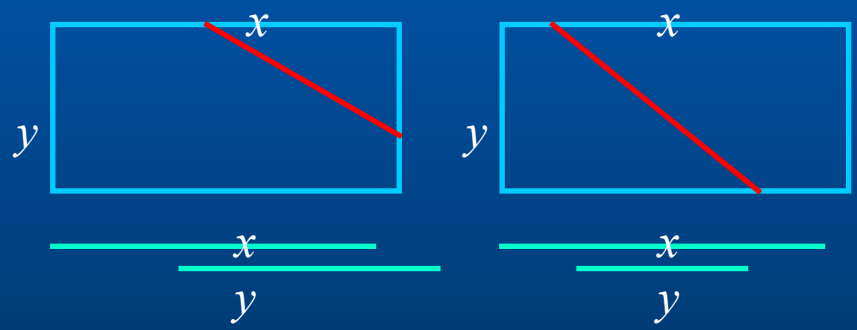
HEAGAWGHEE

HEA . AW - HE .

T =20
two separate match regions.
Dots : unmatched regions of *x*.

- Local alignment: Overlap matches

- One sequence contains the other, or that they overlap
  - Often occurs when comparing fragments of genomic DNA sequence to each other, or to larger chromosomal sequences.



  - We want
    - Match start on the top of left border of the matrix.
    - Match finish on the right or bottom border.
    - ➔ initial setting with $F(i,0) = F(0,j) = 0.$

$$\text{for } i = 1,\ldots,n \text{ and } j = 1,\ldots,m.$$

- Local alignment: Overlap matches

- Traceback
  - Starts from the maximum point in two border.
    - Right border (i, m), i=1,…,n.
    - Bottom border (n, j), j=1,…m.
  - Until the top or left edge is reached.
- Recurrences

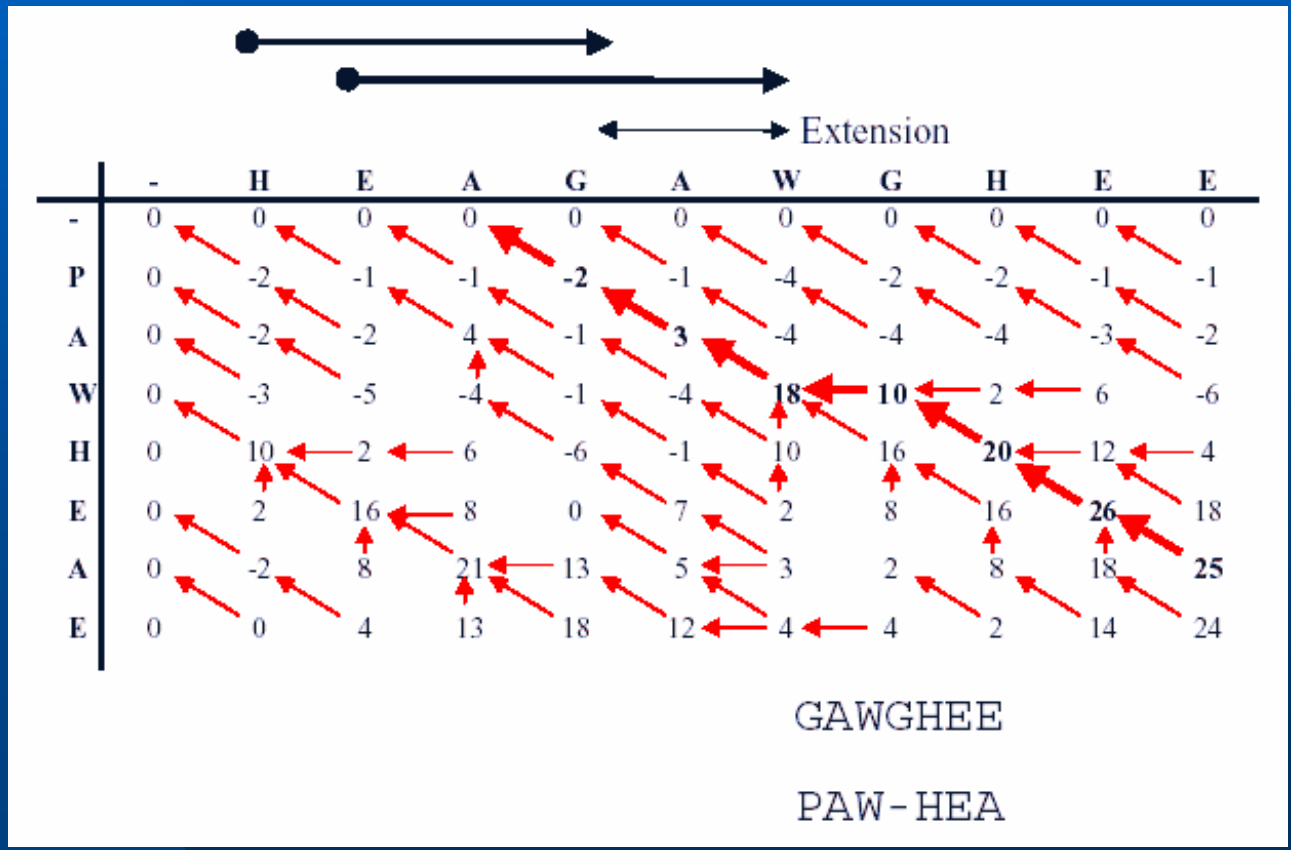
$$F(i,j) = \max \begin{cases} F(i-1,0) \\ F(i,m) - T \end{cases}$$

Look at complete matches $y_1,\ldots,y_m$

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

- Local alignment: Overlap matches

- Overlap matches



	-	H	E	A	G	A	W	G	H	E	E
-	0	0	0	0	0	0	0	0	0	0	0
P	0	-2	-1	-1	**-2**	-1	-4	-2	-2	-1	-1
A	0	-2	-2	4	-1	**3**	-4	-4	-4	-3	-2
W	0	-3	-5	-4	-1	-4	**18**	**10**	2	6	-6
H	0	10	2	6	-6	-1	10	16	**20**	12	4
E	0	2	16	8	0	7	2	8	16	**26**	18
A	0	-2	8	21	13	5	3	2	8	18	**25**
E	0	0	4	13	18	12	4	4	2	14	24

Start with max value on right border.

2.4 Dynamic programming with more complex models (1/4)

- Are all the gaps the same?
- Should two gaps at different locations be treated the same as two consecutive gaps?
- So far the gap penalty has been d=-8.
- To consider variations in gap scoring we go back to the affine method.

$$F_{i,j} = \max \begin{cases} F_{i-1,j-1} + s(x_i, y_j) \\ F_{i-1,j} + \gamma(i-k); \quad k = 0,\ldots,i-1 \\ F_{i,j-1} + \gamma(j-k) \quad k = 0,\ldots,j-1 \end{cases}$$

- Complexities
  - It require $O(n^3)$.
  - ← for each cell (i, j), have to look at $i+j+1$ potential precursors.

2.4 Dynamic programming with more complex models (2/4)

- Alignment with Affine Gap Structure
  - The standard is $\gamma(g) = -d - (g-1)e$
  - This now gets back to more complicated computations because we must keep track of the different values for the different gap lengths.
- Use 3-way method
  - M = Best score given x_i aligns with y_j
  - I_x = best score given x_i aligns with a gap.
  - I_y = best score given y_j aligns with a gap.
  - Instead of $F_{i,j}$ we now have M, Ix, Iy.

$$Ix_{i,j} = \max\begin{cases} M_{i,j-1} - d \\ Ix_{i-1,j} - e \end{cases}$$

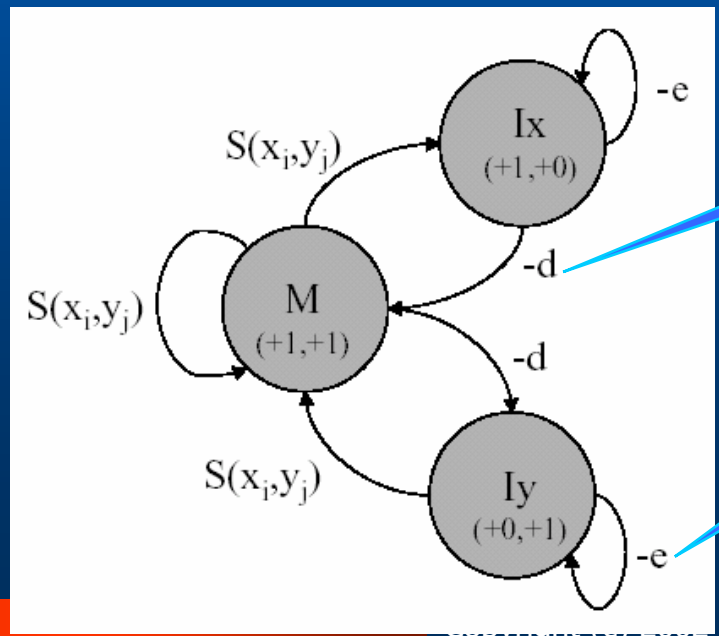
$$Iy_{i,j} = \max\begin{cases} M_{i-1,j} - d \\ Iy_{i,j-1} - e \end{cases}$$

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

2.4 Dynamic programming with more complex models (3/4)

- Assumption
  - For I_x and I_y a gap is not followed or preceded by an insertion.
  - New value for the element in the matrix is the maximum of these three scores.

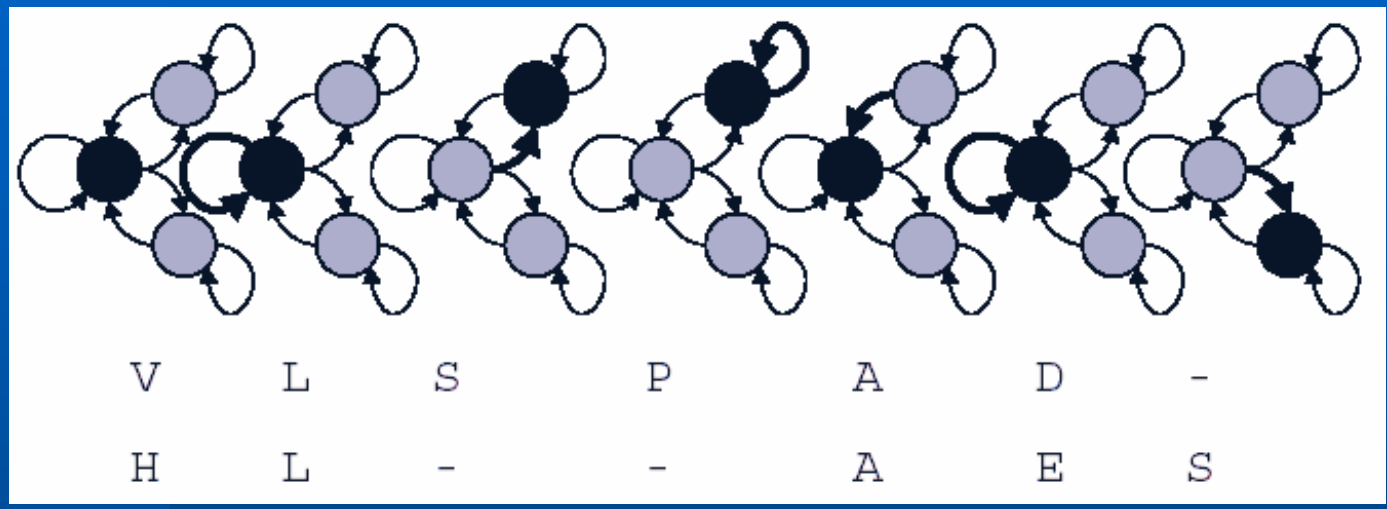
Finite State Automata: ***FSA***



Gap-open penalty

Gap-extension penalty

2.4 Dynamic programming with more complex models (4/4)



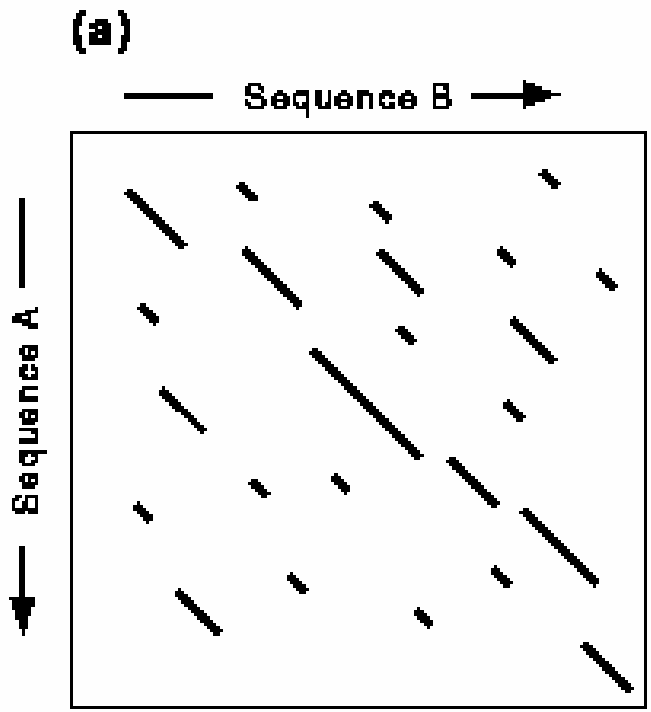
- Each transition: carry a score increment.
- Each state: specify a $\Delta(i, j)$ pair.
  - Used to determine the change in indices i and j when the state is entered.
- An alignment corresponds to a path through the states.

2.5 Heuristic alignment algo.

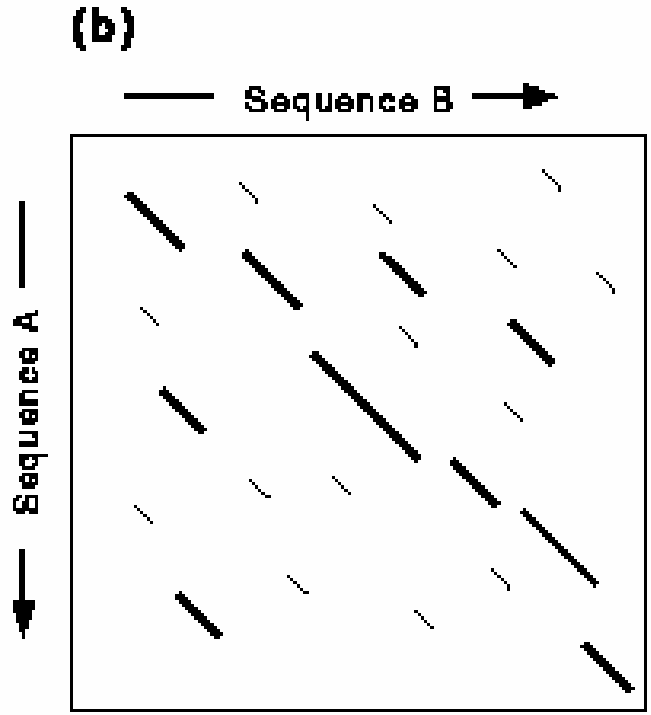
- Speed is an issue.
  - We need more faster algorithms than straight dynamic programming.
  - Non-exact match will be needed.
  - We use some heuristics.
- FASTA [Pearson & Limpman 1988]
  - Sequence similarity search program.
  - Search from sequence database with query.
  - Provide: Protein sequence search + Nucleic acid sequence search.
- BLAST [Altschul *et al*. 1990]
  - Sequence similarity search S/W.
  - Use GenBank database

- FASTA

- Algorithm



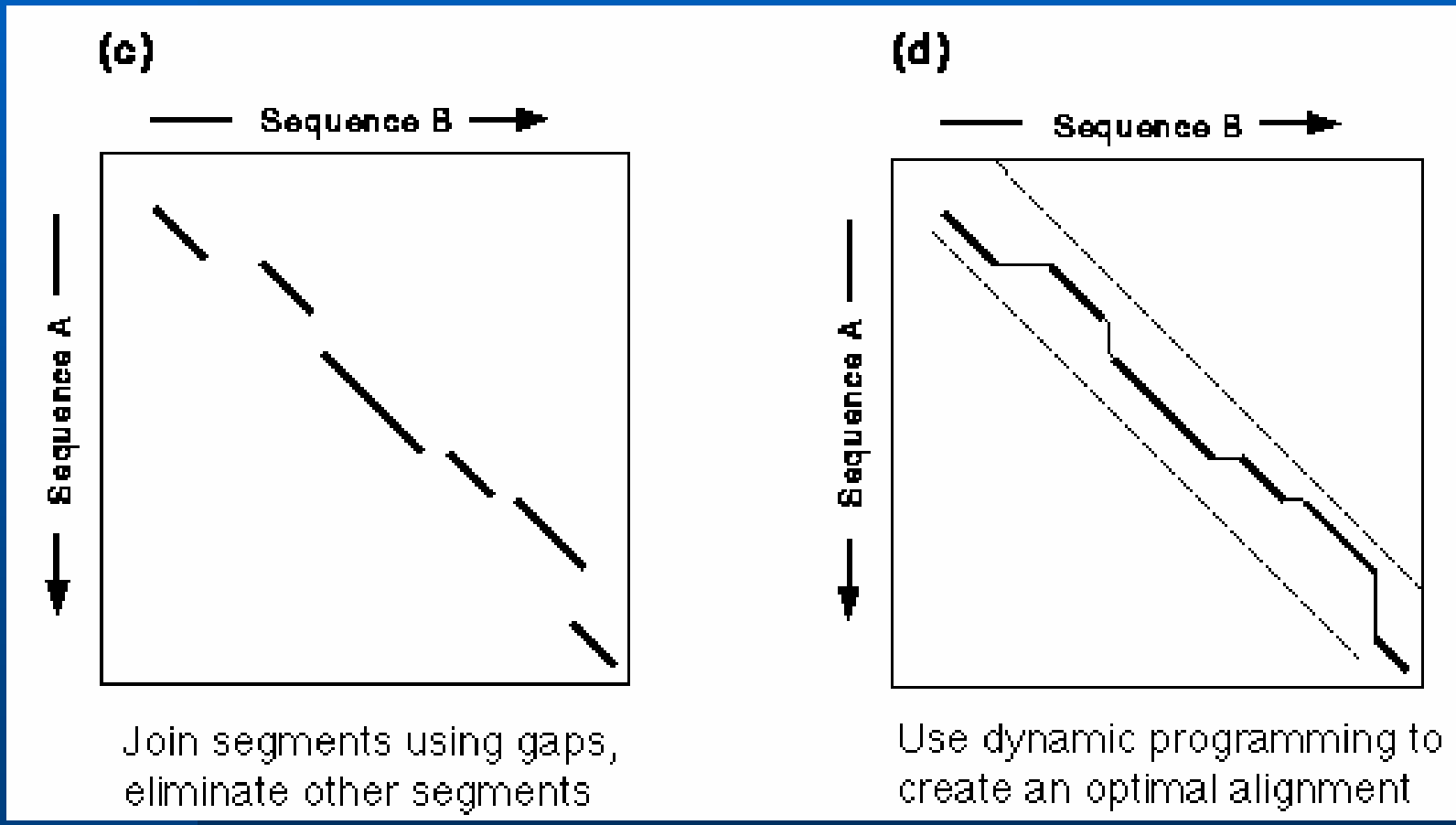
Find runs of identitical words



Re-score using PAM matrix

Keep top scoring segments

- FASTA

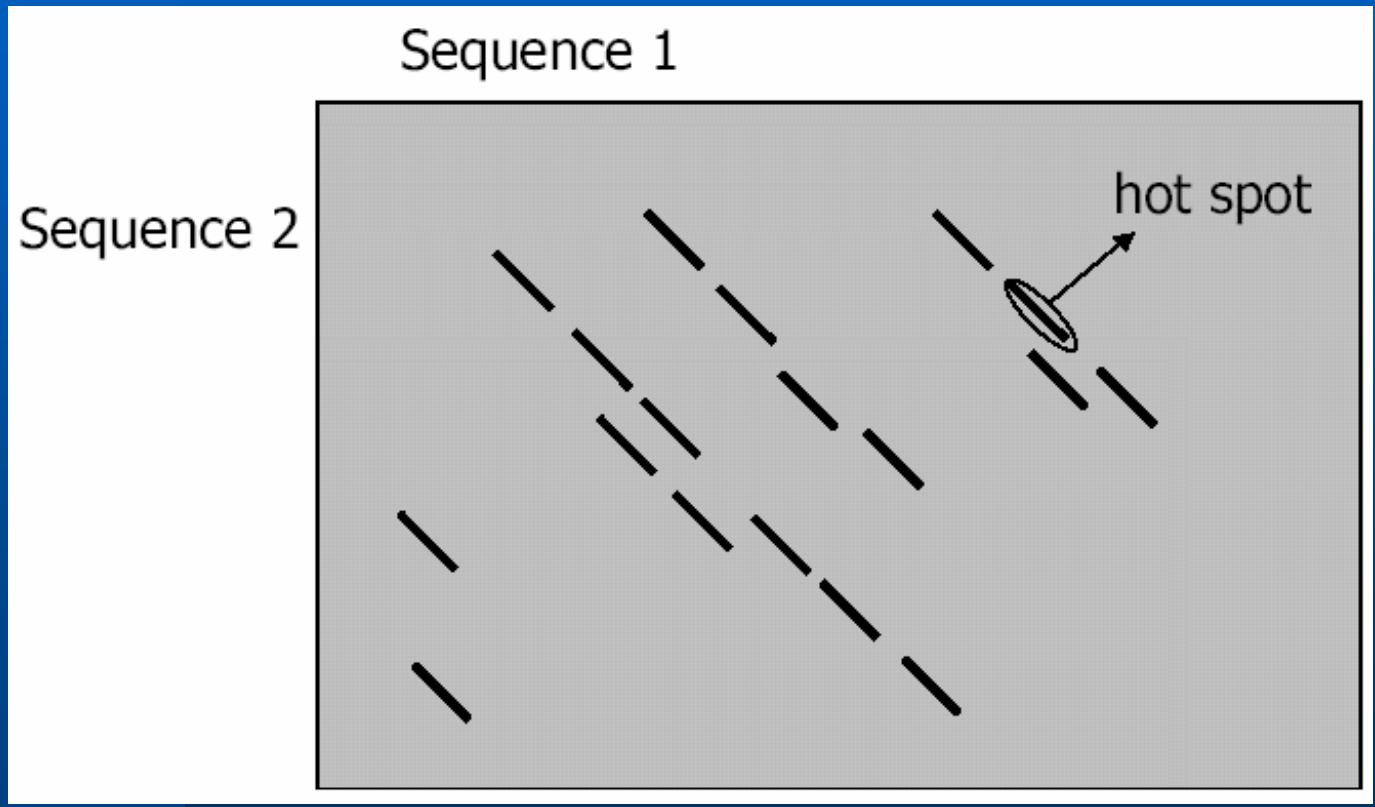


- FastA (detailed)

- Fast search algorithm for DNA or Protein sequence pair alignments.
  - Compare each sequence in DB with query sequence.
  - Consider continual match sequence with length *k*.
    - Called *ktup*.
    - Consider exact match first and use this as a base component.
- Step 1.
  - Search *hot spot* (match region).
    - Hot spot: - matched substring in two sequences with length *ktup*. (*ktup* setting: 4~6 for DNA, 1~2 for Protein)

- FastA (detailed)

- Step 1.

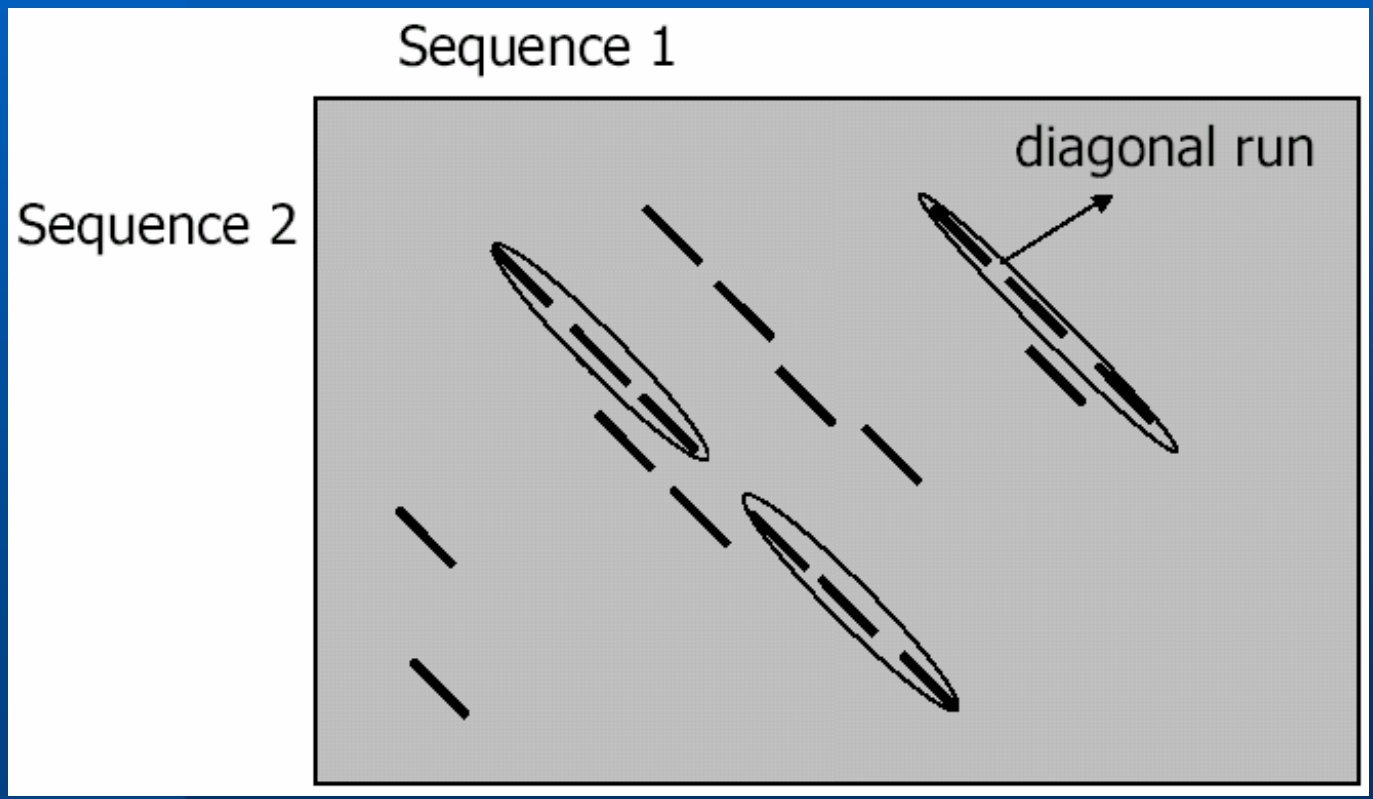


- FastA (detailed)

- Step 2.
  - Find 10 best diagonal run.
  - Diagonal run
    - Sequence of hot spot, which is located in near position in the same diagonal.
    - Score of diagonal run
      - hot spot score (positive) + score between hot spot (negative)
    - Diagonal run represent pair(sub-alignment) of aligned substring.

- FastA (detailed)

- Step 2.

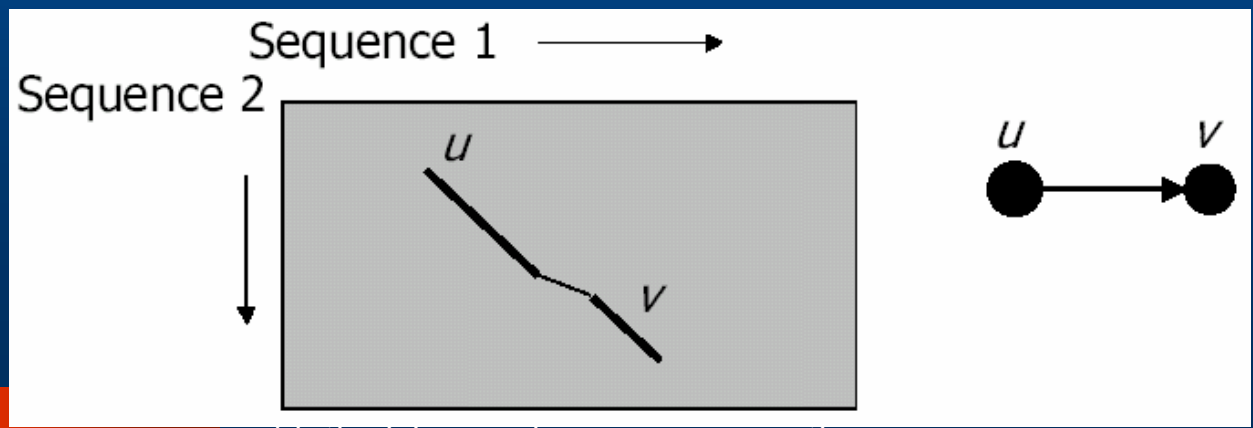


- FastA (detailed)

- Step 3.
  - Rescoring
    - Rescore the diagonal run with substitution matrix.
      - (BLAST does not do this.)
    - Find diagonal run with best score: $init_1$.
    - Cut off some diagonal run with some threshold score.
- Step 4.
  - Incorporate '*gap*'
    - Find local alignment by connecting remain diagonal run.
  - Connect diagonal run
    - Vertices: diagonal run with score more than cutoff.
    - Vertex weight: use the score of previous step.

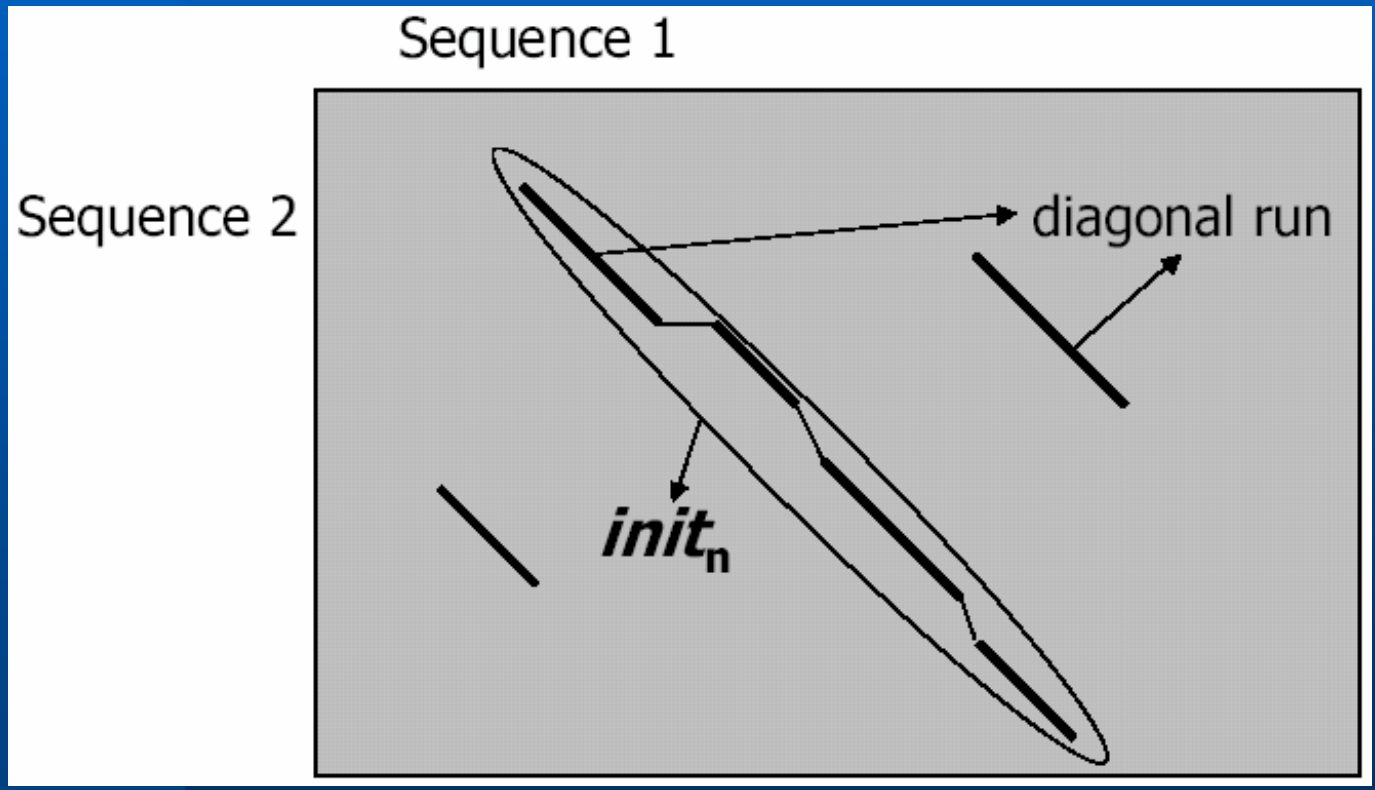
- FastA (detailed)

- Step 4. (contd)
  - Find sub-alignment u, v by diagonal run like
  - Connect two diagonal and assign negative weight.
    - Create directed edge from u to v.
  - Find maximum weight path
  - Connection of diagonal run in the path
    ➔ alignment of two sequences. (local)
  - It called $init_n$.



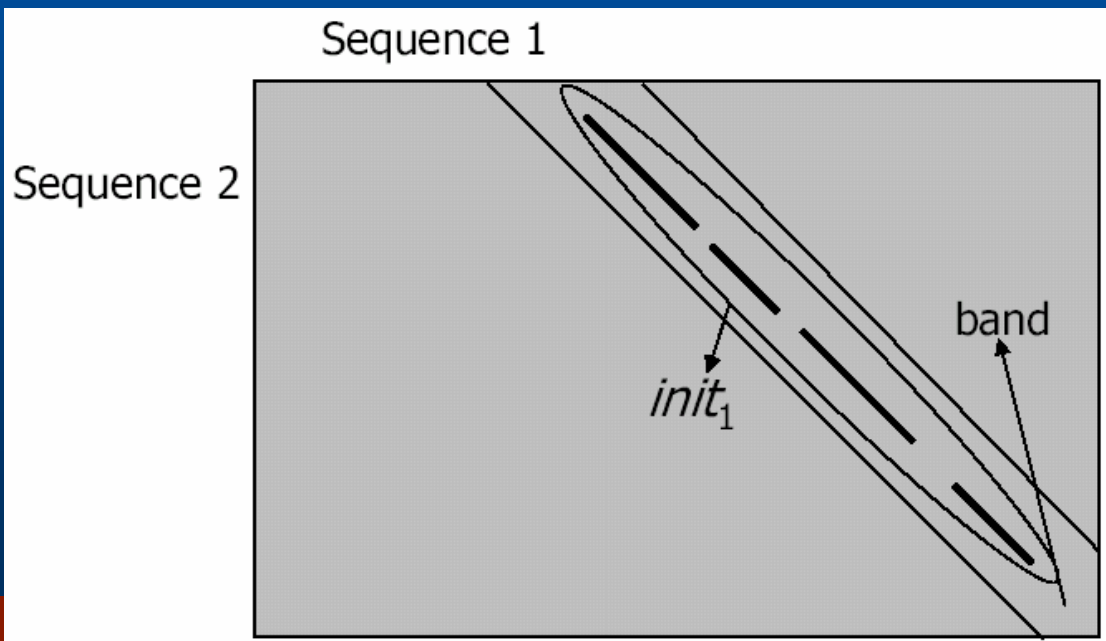
- FastA (detailed)

- Step 4. (contd)



- FastA (detailed)

- Step 5.
  - Alignment
    - Decide the band based on $init_1$.
    - Find alignment with a given band via Smith-Waterman algo.
    - This alignment called *opt*.



- FastA (detailed)

- Step 6.
  - Ranking
    - Sort DB sequences with $init_n$ score or *opt* score of query.
    - Perform full DP algorithm for high score sequences and find alignment with query.
- Step 7.
  - Random sequence simulation
    - Calculate a distribution of random sequence pair alignment.
    - ➔ for evaluating importance of alignment result.

- FastA (detailed) - output

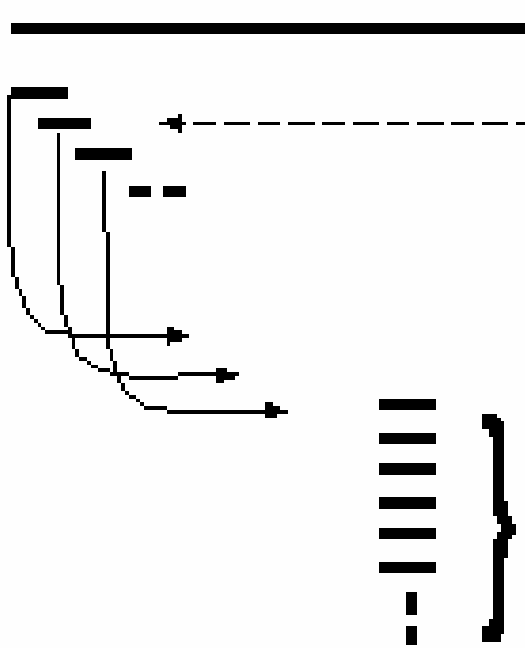
Figure : FastA output: The first lines contain general information about the search parameters. Score lines are made of nine rows: 1-3 details the name and the annotation of the hit, 4-9 are the FastA scores.

```
       opt      E()
< 20 82817     0:============================================================
  22     0     0:           one = represents 1381 library sequences
  24     0     0:
  26     0    11:*
  28     4   119:*
  30   224   725:*
  32   976  2804:= *
  34  3145  7603:===  *
  36  5406 15615:====       *
  38 11674 25805:=========         *
  40 33028 35996:========================  *
  42 58047 44001:====================================*==========
  44 70768 48537:====================================*===============
  46 61240 49436:====================================*=========
  48 42577 47329:==============================   *
  50 23779 43188:==================             *
  52 12532 37970:==========                   *
  54  9284 32433:=======                    *
  56 10961 27091:========                *
  58 16505 22241:============    *
```

- BLAST

- Algorithm (3 step)

(1) For the query, find the list of high scoring words of length w

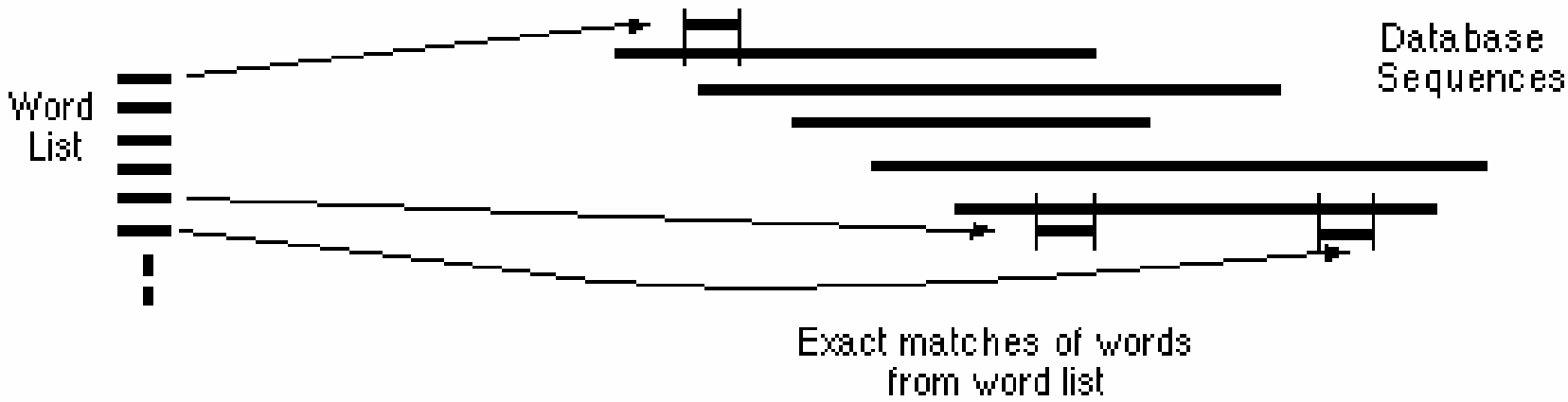


Query Sequence of length L

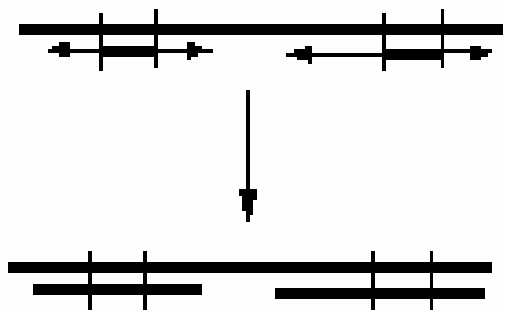
Maximum of L-w+1 words (typically w = 3 for proteins)

For each word from the query sequence find the list of words that will score at least T when scored using a pair-score matrix (e.g. PAM 250).

(2) Compare the word list to the database and identify exact matches



(3) For each word match, extend the alignment in both directions to find alignments that score greater than a threshold of value S



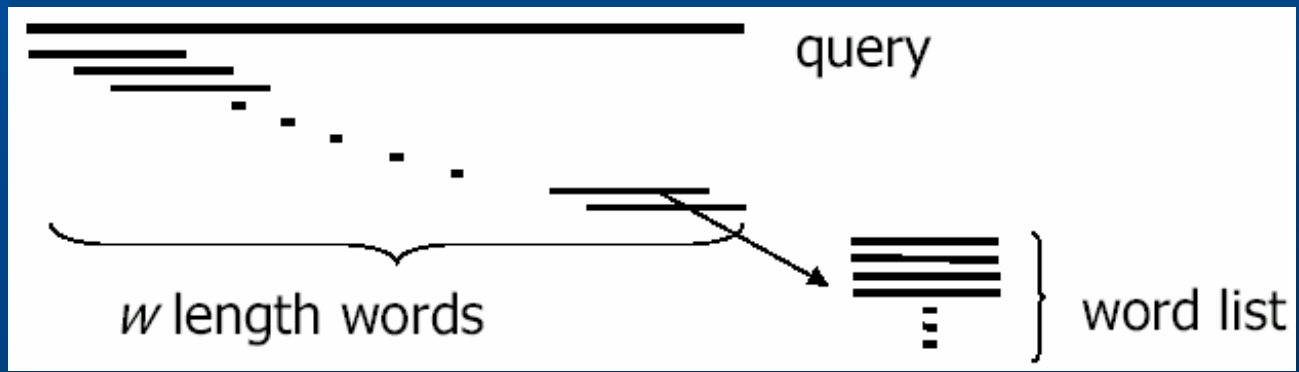
Maximal Segment Pairs (MSPs)

- BLAST (detailed)

- For fast search more than FastA.
- Feature
  - Consider substitution matrix for hot spot search.
    - Cf.) FastA use exact match.
  - Find all possible matching word list in preprocessing step.
  - Do not consider '*gap*'.
- Segment pair
  - Each substring pair aligned without gap for a two sequences.
- Maximal segment pair (MSP)
  - Segment pair with highest scores for two sequences.
- BLAST
  - Compare all DB sequences with query.
  - Find DB sequences, which have MSP with score more than cutoff.

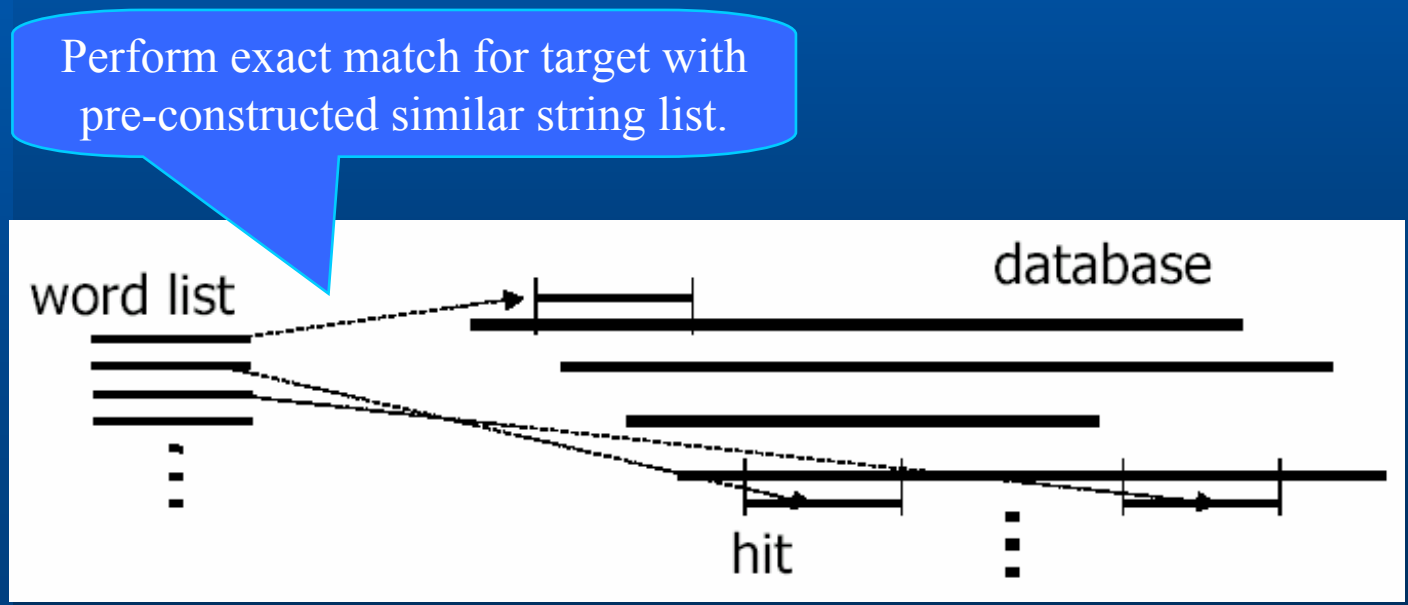
- BLAST (detailed)

- Step 1.
  - Preprocessing
    - Create wordlist for each string with length w, which has similarity more than t for substring of query with length w.
      - In the table or DFA.
    - w setting: 12 for DNA, 3~5 for Protein.



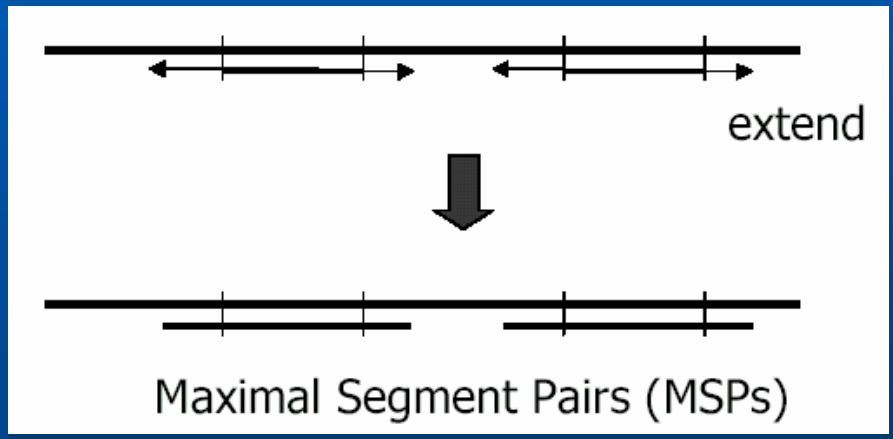
- BLAST (detailed)

- Step 2.
  - Find hit from database
    - Compare the word in the wordlist with database sequences and find the exact matching region (hit).



- BLAST (detailed)

- Step 3.
  - Find MSP by expanding each hit.
    - By dynamic programming.



Sequences producing High-scoring Segment Pairs:	High Score	Smallest Sum Probability P(N)	N
sp|P08506|DACC_ECOLI PENICILLIN-BINDING PROTEIN 6 PRECURS...	894	5.0e-120	1
sp|P38422|DACF_BACSU PENICILLIN-BINDING DACF PROTEIN PREC...	209	5.0e-47	3
sp|P28271|IREB_MOUSE IRON-RESPONSIVE ELEMENT BINDING PROT...	59	0.9996	1
sp|P31571|CAIA_ECOLI PROBABLE CARNITINE OPERON OXIDOREDUC...	48	0.9998	2

2.6 Linear space alignment

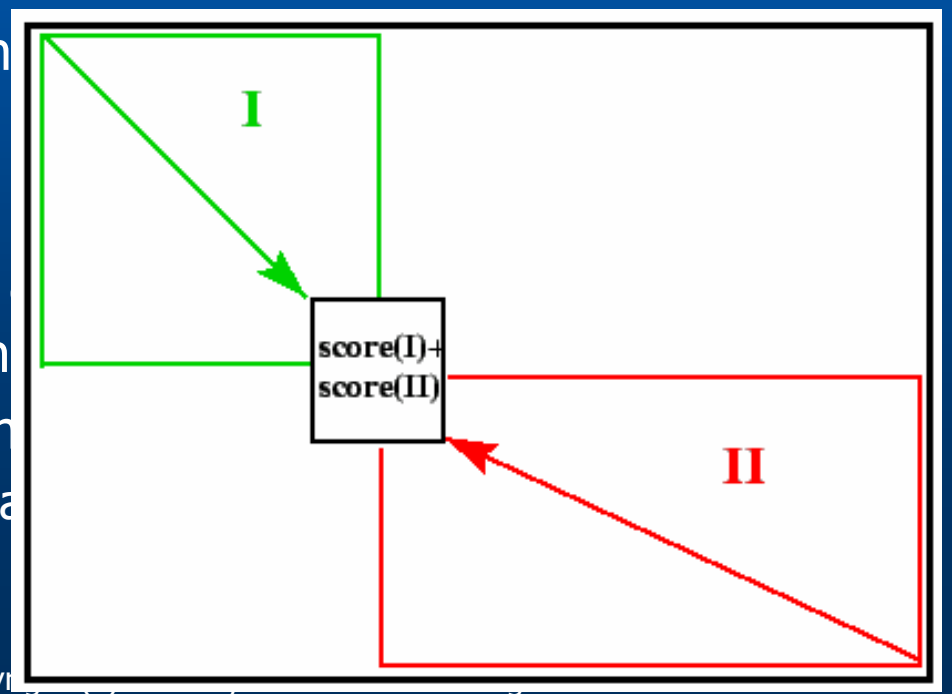
- Problem
  - Limited memory.
  - For two sequence with length *m* and *n*, we need over all size *mn*.
    - For two sequences ➔ will be okay.
    - For DNA sequences ➔ required memory for the full matrix can exceed a machine's physical capacity.
- *Linear space* method
  - Important basic technique in pairwise sequence DP.
  - Techniques that give the optimal alignment in limited memory, order *m*+*n* rather than *mn*, with no more than a doubling in time.

- Linear space alignment

- If only the maximal score needed
  - Recurrence relation for $F(i, j)$ is local.
  - We need only one row back entry.
  - Throw away rows of the matrix that are further than one back from the current point.
  - ➔ we lose the traceback poin
    - We can't find alignment.
- Suppose that
  - We can identify a v such that alignment (alignment cross th
  - ➔ can split the DP problem in
    - From top left (0,0) to (u, v), a
    - - Divide and conquer.

$$u = \left\lfloor \frac{n}{2} \right\rfloor$$

Integer part of $n/2$



- Linear space alignment

- Fill the whole alignment recursively
  - By successively halving each region.
  - For a short sequences,
    - $O(n^2)$ alignment and traceback method can be used.
- How do we find v?
  - For $i>v$, define c(i, j)
    - (u, c(i, j)) is on the optimal path from (1,1) to (i, j).
    - Update c(i, j) as we calculate $F(i, j)$.
    - If (i', j') is the preceding cell to (i, j) from which $F(i, j)$ is derived ➔ set c(i, j)=j' if $i=u$, else c(i, j)=c(i', j').
    - Only need to maintain the previous row of c(). Like $F()$
    - Can read out from the final cell of the matrix. (v=c(n, m))

- Linear space alignment

- Procedure for finding v has not been published.
- Widely known procedure first appeared in the CS literature [Hirschberg 1975]
  - Introduced to biology by Myers & Miller [1988].
- Third linear space approach [Waterman 1995].
  - See page 211.
- Review of linear space algorithm in pairwise alignment [Chao, Hardison & Miller 1994].

Further reading

- Review of dynamic programming method for biological sequence comparison
  - Pearson [1996], Pearson & Miller [1992]
- The sensitivity of dynamic programming methods
  - Pearson [1995], Shpaer *et al.* [1996]
- Probabilistic version of Smith-Waterman algorithm
  - Chapter 4 of this book.
- Fast 'bounded' dynamic programming algorithm
  - [Chao, Pearson & Miller 1992]
- Aligning protein query sequences to DNA target sequence
  - {Huang & Zhang 1996]
- Optimal, suboptimal and near-optimal problem
  - [Zuker 1991]