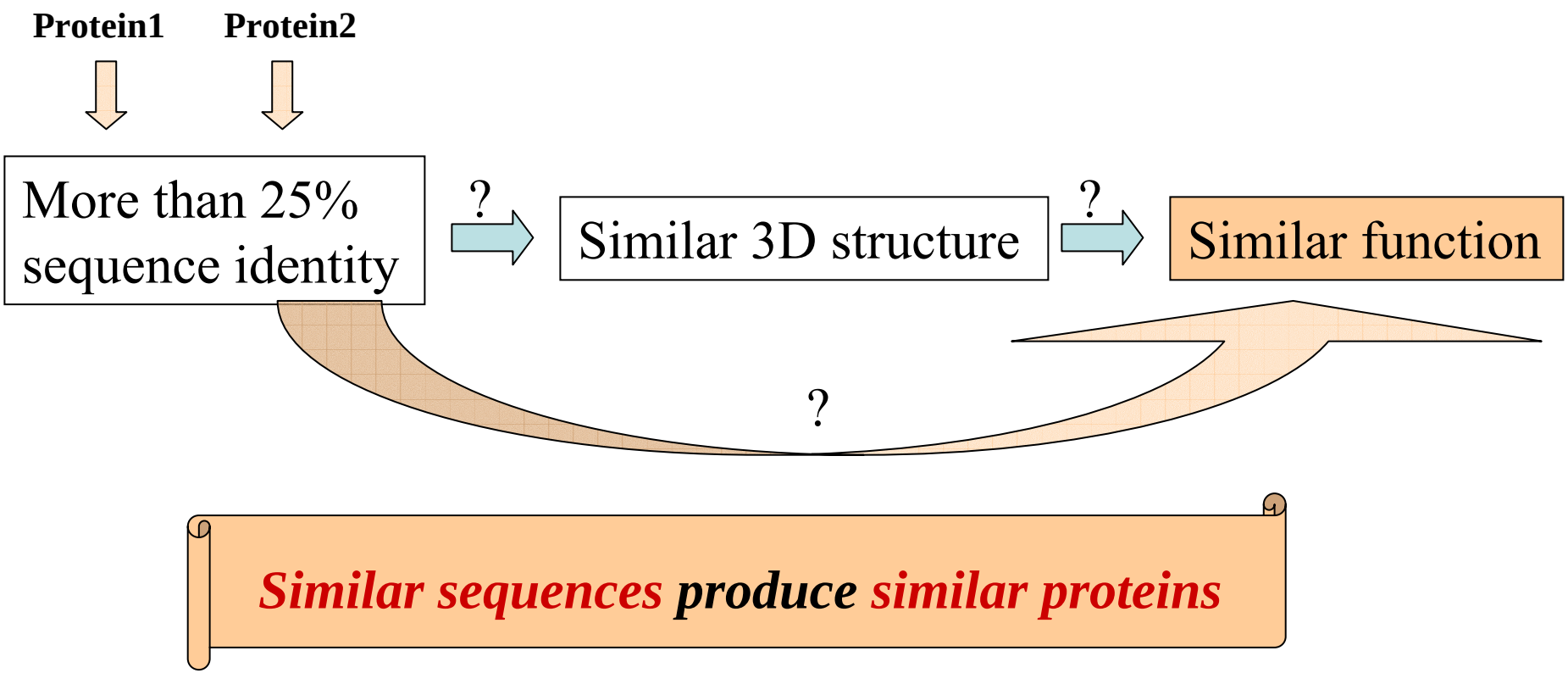
Seq.Align. ➡ Protein Function



Sequence Alignment

Motivation:

- Storing, retrieving and comparing DNA sequences in Databases.
- Comparing two or more sequences for similarities.
- Searching databases for related sequences and subsequences.
- Exploring frequently occurring patterns of nucleotides.
- Finding informative elements in protein and DNA sequences.
- Various experimental applications (reconstruction of DNA, etc.)

Exact Pattern Matching

- Given a pattern P of length m and a (longer) string T of length n, find all the occurrences of P in T.

- Naïve algorithm: O(m*n)
- Boyer-Moore, Knuth-Pratt-Morris:
  O(n+m)

Alignment - inexact matching

Substitution - replacing a sequence base by another.
Insertion - an insertion of a base (letter) or several bases to the sequence.
Deletion - deleting a base (or more) from the sequence.

(*Insertion* and *deletion* are the reverse of one another)

A possible alignment of the insulin proteins from sheep and zebrafish is

```
Fish:  MAVWLQAGALLVLLVV-SSVSTNPGTPQHLCGSHLVDALYLVCGPTGFFYNPK--R
Sheep: MALWTRLVPLLALLALWAPAPAHAFVNQHLCGSHLVEALYLVCGERGFFYTPKARR

Fish:  DVE-PLLGFLPPKSAQETEVADFAFKDHAELIRKRGIVEQCCHKPCSIFELQNYCN
Sheep: EVEGPQVGAL--ELAGGPG-AG-GL-EGPP-Q-KRGIVEQCCAGVCSLYQLENYCN
```

Seq. Align. Score

```
SEQ 1  GTAGTACAGCT-CAGTTGGGATCACAGGCTTCT
       ||||  ||  |||  ||||||   ||||||  |||
SEQ 2  GTAGAACGGCTTCAGTTG---TCACAGCGTTC-
```

Distance 1 - match 0, substitution 1, indel 2 $\Rightarrow$ distance = 14.

Distance 2 - match 0, d(A,T)=d(G,C)=1, d(A,G)=1.5 indel 2 $\Rightarrow$ distance = 14.5.

Similarity - match 1, substitution 0, indel -1.5 $\Rightarrow$ similarity = 16.5.

General setup - substitution matrix S(i,j), indel S(i,-) or S(-,j).

Commonly used matrices:
PAM250, BLOSUM64

Global Alignment

Global Alignment

INPUT: Two sequences S and T of roughly the same length.

QUESTION: What is the maximum similarity between them?
Find one of the best alignments.

The *IDEA*

s[1…n]
t[1…m]

To align s[1...i] **with** t[1…j] **we have three choices:**

* align s[1…i-1] with t[1…j-1] and ***match*** s[i] with t[j]

s[1…i-1] i
t[1…j-1] j

* align s[1…i] with t[1…j-1] and ***match*** a space with t[j]

s[1… i] -
t[1…j-1] j

* align s[1…i-1] with t[1…j] and ***match*** s[i] with a space

s[1…i-1] i
t[1… j] -

Recursive Relation

Define: scoring matrix m(a,b) $a,b \in \sum \cup \{-\}$

Define: H_{ij} the best score of alignment between s[1…i] and t[1…j]

for 1 <= i <= n, 1 <= j <= m

$$H_{ij} = \max \begin{cases} H_{i-1j-1} + m(s_i,t_j) \\ H_{ij-1} + m(-,t_j) \\ H_{i-1j} + m(s_i,-) \end{cases}$$

Optimal alignment score = H_{nm}

$H_{i0} = \sum_{0,k} m(s_k,-)$

$H_{0j} = \sum_{0,k} m(-,t_k)$

Needleman-Wunsch 1970

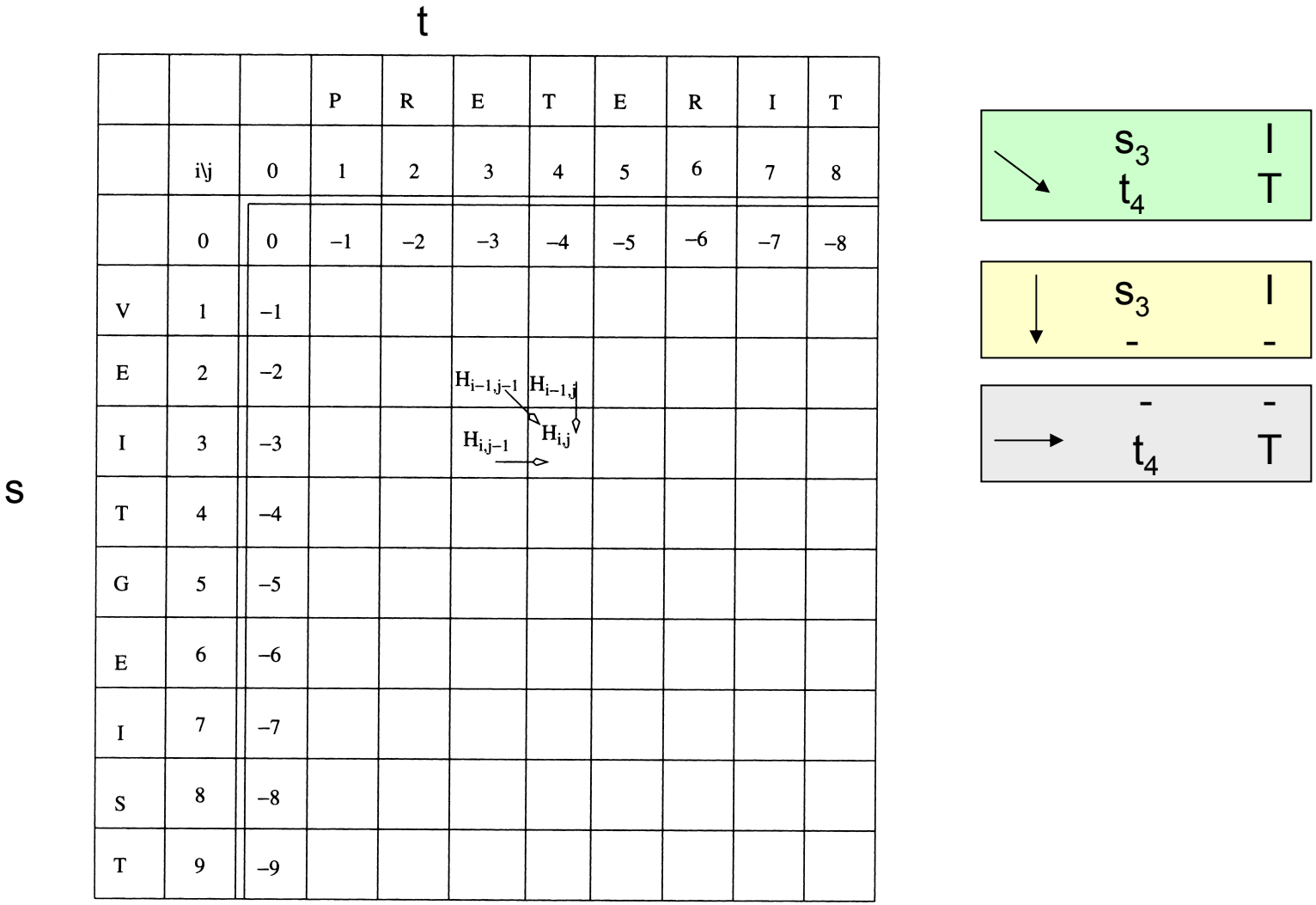


Figure 1.2 The dynamic programming matrix for the example sequences, and how the values of the cells are calculated. Row and column 0 are initialized for the score of a blank equal to -1. To calculate the value of $H_{i,j}$ one needs the values of $H_{i,j-1}$, $H_{i-1,j}$ and $H_{i-1,j-1}$.

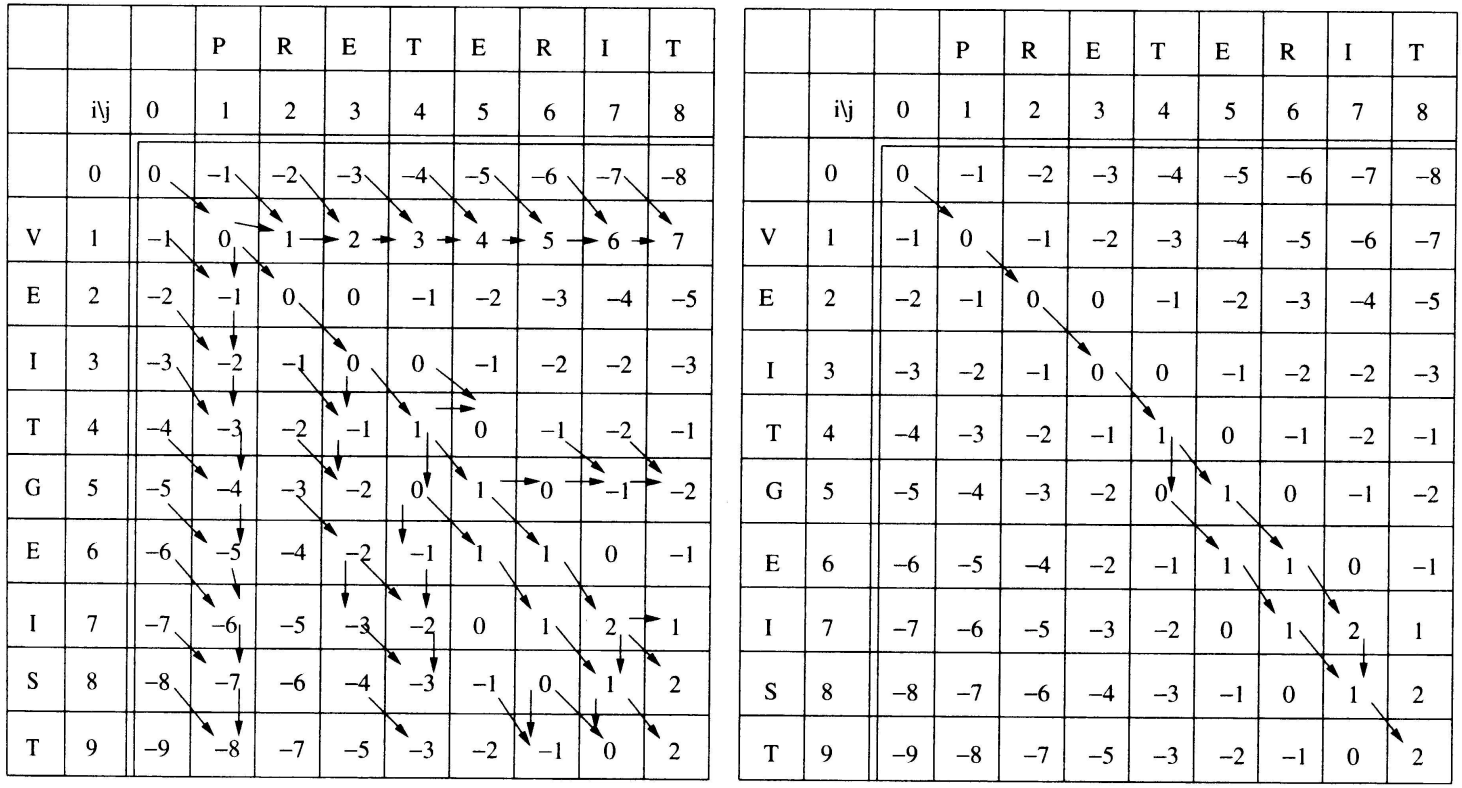


Figure 1.3 (a) The filled-in matrix. The arrows show which cells are used for finding the maximum score. Note that not all arrows are drawn. (b) The arrows showing the paths giving alignments with highest score.

Local Alignment

Local Alignment

INPUT: Two sequences S and T .

QUESTION: What is the maximum similarity between a subsequence of S and a subsequence of T ? Find most similar subsequences.

S = g g t c t g a g

T = a a a c g a

editing operations values:

match = 2 indel/substitution = -1

The best *local alignment* is:

α = ctga $(\in S)$

β = c-ga $(\in T)$

Recursive Relation

for $1 <= i <= n$, $1 <= j <= m$

$$H_{ij} = \max \begin{cases} H_{i-1j-1} + m(s_i, t_j) \\ H_{ij-1} + m(-, t_j) \\ H_{i-1j} + m(s_i, -) \\ 0 \end{cases}$$

$H_{i0} = 0$

$H_{0j} = 0$

Optimal alignment score $= \max_{ij} H_{ij}$

Smith-Waterman 1981

Penalties should be negative

Sequence Alignment

Complexity:

Time ***O(n*m)***

Space ***O(n*m)*** (exist algorithm with ***O(min(n,m))***)

Ends free alignment

Ends free alignment

INPUT: Two equences S and T (possibly of different length).

QUESTION: Find one of the best alignments between subsequences of S and T when at least one of these subsequences is a prefix of the original sequence and one (not necessarily the other) is a suffix.

Gap Alignment

Definition: A **gap** is the maximal contiguous run of spaces in a single sequence within a given alignment.The length of a gap is the number of indel operations on it. A gap penalty function is a function that measure the cost of a gap as a (nonlinear) function of its length.

Gap penalty

INPUT: Two sequences S and T (possibly of different length).

QUESTION: Find one of the best alignments between the two sequences using the gap penalty function.

Affine Gap: $W_{total} = W_g + qW_s$

W_g – weight to open the gap

W_s – weight to extend the gap

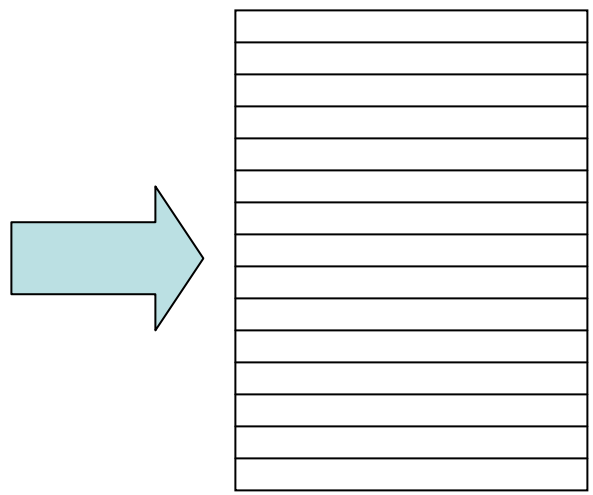
What Kind of Alignment to Use?

- The same protein from the different organisms.
- Two different proteins sharing the same function.
- Protein domain against a database of complete proteins.
- Protein against a database of small patterns (functional units)
- ...

Sequence Alignment vs. Database

- **Task:** Given a query sequence and millions of database records, find the optimal alignment between the query and a record

ACTTTTGGTGACTGTAC



Sequence Alignment vs. Database

- **Tool:** Given two sequences, there exists an algorithm to find the best alignment.

- **Naïve Solution:** Apply algorithm to each of the records, one by one

Sequence Alignment vs. Database

- **Problem:** An *exact algorithm* is too slow to run millions of times (even linear time algorithm will run slowly on a huge DB)

- **Solution:**
  - Run in parallel (expensive).
  - Use a fast (*heuristic*) method to discard irrelevant records. Then apply the *exact algorithm* to the remaining few.

Sequence Alignment vs. Database

General Strategy of Heuristic Algorithms:

-Homologous sequences are expected to contain un-gapped (at least) short segments (probably with substitutions, but without ins/dels)

-Preprocess DB into some fast access data structure of short segments.

FASTA Idea

- **Idea**: a good alignment probably matches some identical ‘words’ (*ktups*)
- **Example:**

Database record:

ACTTGTAGATACAAAATGTG

Aligned query sequence:

A-TTGTCG-TACAA-ATCTGT

Matching words of size 4

Dictionaries of Words

ACTTGTAGATAC Is translated to the dictionary:

ACTT,

CTTG,

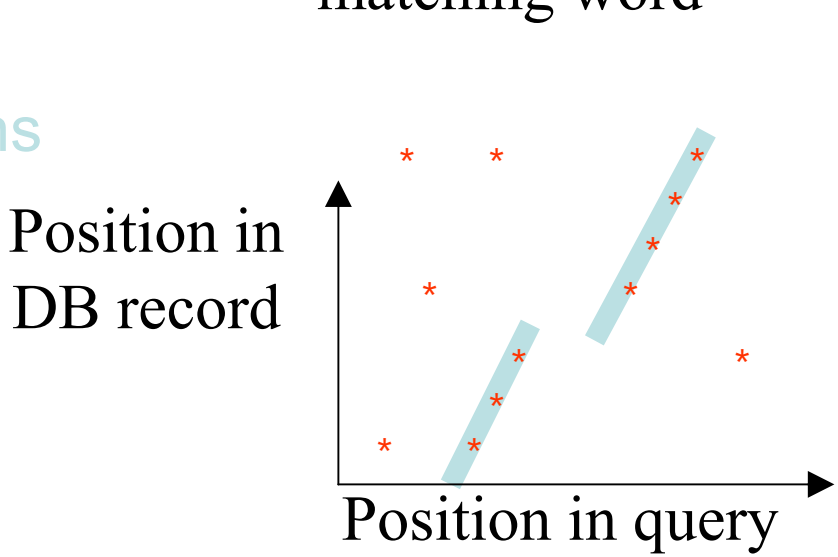
TTGT,

TGTA...

Dictionaries of well aligned sequences share words.

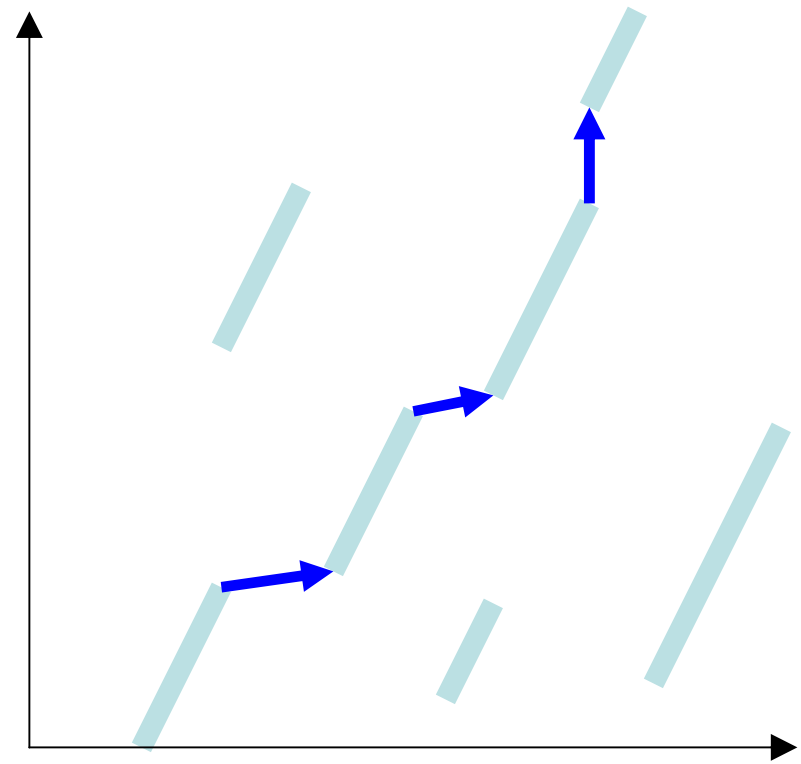
FASTA Stage I

- Prepare dictionary for db sequence (in advance)
- Upon query:
  - Prepare dictionary for query sequence
  - For each DB record:
    - Find matching words
    - Search for long diagonal runs of matching words
    - *Init-1 score*: longest run
    - Discard record if low score



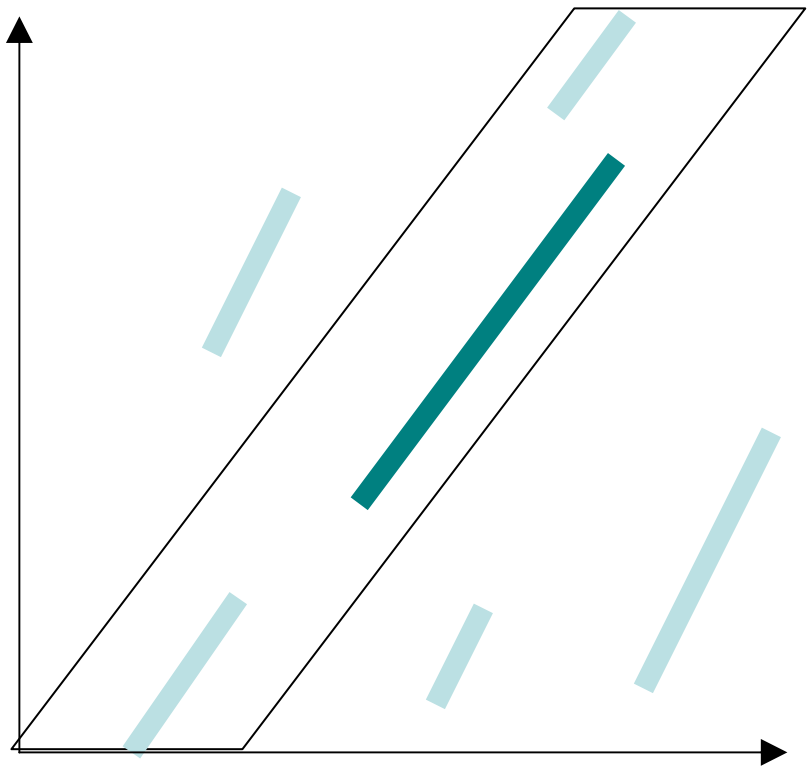
FASTA stage II

- Good alignment – path through many runs, with short connections
- Assign weights to runs(+) and connections(-)
- Find a path of max weight
- *Init-n score* – total path weight
- Discard record if low score



FASTA Stage III

- Improve *Init-1.* Apply an *exact algorithm* around *Init-1* diagonal within a given width band.
- *Init-1* → *Opt-score* – new weight
- Discard record if low score



FASTA final stage

- Apply an exact algorithm to surviving records, computing the final alignment score.

P-Value

The observed number of random records achieving E-value *E* or better (smaller) is distributed Poisson(*E*)

Prob(*r* such records) = $\frac{\exp(-E)E^{r}}{r!}$

Note: The model assumes an I.I.D. trial for each database record

BLAST (Basic Local Alignment Search Tool)
Approximate Matches

BLAST:

Words are allowed to contain inexact matching.

Example:

In the polypeptide sequence `IHAVEADREAM`

The 4-long word `HAVE` starting at position 2 may match

`HAVE`, `RAVE`, `HIVE`, `HALE`, …

Approximate Matches

For each word from DB generate similar words (according to the substitution matrix) and store them in a look-up table.

BLAST Stage I

- Find approximately matching word pairs
- Extend word pairs as much as possible, i.e., as long as the total weight increases
- Result: *High-scoring Segment Pairs (HSPs)*

```
THEFIRSTLINIHAVEADREAMESIRPATRICKREAD
      INVIEIAMDEADMEATTNAMHEWASNINETEEN
```

BLAST Stage II

- Try to connect HSPs by aligning the sequences in between them:

```
THEFIRSTLINIHAVEADREA____M_ESIRPATRICKREAD
      INVIEIAMDEADMEATTNAMHEW___ASNINETEEN
```

PAM250

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V	B	Z	X	*
A	2	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0	0	0	0	-8
R	-2	6	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2	-1	0	-1	-8
N	0	0	2	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2	2	1	0	-8
D	0	-1	2	4	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2	3	3	-1	-8
C	-2	-4	-4	-5	12	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2	-4	-5	-3	-8
Q	0	1	1	2	-5	4	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2	1	3	-1	-8
E	0	-1	1	3	-5	2	4	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2	3	3	-1	-8
G	1	-3	0	1	-3	-1	0	5	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1	0	0	-1	-8
H	-1	2	2	1	-3	3	1	-2	6	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2	1	2	-1	-8
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5	2	-2	2	1	-2	-1	0	-5	-1	4	-2	-2	-1	-8
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6	-3	4	2	-3	-3	-2	-2	-1	2	-3	-3	-1	-8
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5	0	-5	-1	0	0	-3	-4	-2	1	0	-1	-8
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6	0	-2	-2	-1	-4	-2	2	-2	-2	-1	-8
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9	-5	-3	-3	0	7	-1	-4	-5	-2	-8
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6	1	0	-6	-5	-1	-1	0	-1	-8
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2	1	-2	-3	-1	0	0	0	-8
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3	-5	-3	0	0	-1	0	-8
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17	0	-6	-5	-6	-4	-8
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	-2	-3	-4	-2	-8
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4	-2	-2	-1	-8
B	0	-1	2	3	-4	1	3	0	1	-2	-3	1	-2	-4	-1	0	0	-5	-3	-2	3	2	-1	-8
Z	0	0	1	3	-5	3	3	0	2	-2	-3	0	-2	-5	0	0	-1	-6	-4	-2	2	3	-1	-8
X	0	-1	0	-1	-3	-1	-1	-1	-1	-1	-1	-1	-1	-2	-1	0	0	-4	-2	-1	-1	-1	-1	-8
*	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	-8	1

M. Dayhoff Scoring Matrices
Point Accepted Mutations or **PAM matrices**

Proteins with 85% identity were used ->
the function is not significantly changed ->
the mutations are "**accepted"**

PAM units – the measure of the amount of evolutionary distance between two amino acid sequences.

One PAM unit – S_1 has converted (mutated) to S_2 with an average of one accepted point-mutation event per 100 amino acids.

1) Probability matrix

2) Scoring matrix

p_a ($=N_a/N$) – probability of occurrence of amino acid 'a' over a large, sufficiently varied, data set.

$\sum_a p_a = 1$

f_{ab} – the number of times the mutation a <-> b was observed to occur.

$f_a = \sum_{b!=a} f_{ab}$ - the total number of mutations in which a was involved

$f = \sum_a f_a$ - the total number of amino acid occurrences involved in mutations (twice the number of mutations).

Assumptions –

(a) 1 in 100 amino acids on average is changed.

(b) mutations are position independent.

(c) mutations are independent on its past.

The probability that a mutation contains 'a': $f_a / (f/2)$

The probability that a mutation originates from 'a': $0.5 * f_a / (f/2) = f_a / f$

$m_a = (f_a / f) * 1/(100 * p_a)$ relative mutability of amino acid 'a'.

It is the probability that the given amino acid will change in the evolutionary period of interest.

M^1 - 20x20 probability matrix

M^1_{ab} - the probability of amino acid 'a' changing into 'b' during one PAM unit.

$M^1_{aa} = 1 - m_a$ - the probablity of 'a' to remain unchanged.

$M^1_{ab} = Pr(a \rightarrow b) = Pr(a \rightarrow b \mid a \text{ changed}) \, Pr(a \text{ changed}) =$

$= (f_{ab} / f_a) m_a$

Easy to see:

$$\sum_b M^1_{ab} = 1 \quad = M^1_{aa} + \sum_{b!=a} (f_{ab}/f_a) m_a = 1 - m_a + m_a / f_a \sum_{b!=a} f_{ab} = 1$$

What is the probability that ‘a’ mutates into ‘b’ in two PAM units of evolution?

a->c->b or a->d-> …

$\sum_c M^1_{ac} M^1_{cb} = M^2_{ab}$ -> $M^2, M^3, M^4 \ldots M^{250} \ldots$

k->∞ M^k converges to a matrix with identical rows.

$M^k_{ac} = p_c$ - no matter what amino acid you start with, after a long period of evolution the resulting amino acid will be ‘c’ with probability p_c .

PAM-k matrix

$PAM\text{-}k_{ab} = M^k_{ab} / p_b$ - probability that a pair 'ab' is a mutation as opposed to being a random occurrence (*likelihood* or *odds* ratio).

If $PAM\text{-}k_{ab} > 1$ b replaces a more frequently than b just appears by chance.

$$M_{ab} / p_b = [(f_{ab}/f_a)m_a] / p_b = (f_{ab}/f_a)\, f_a / (f * 100 * p_a * p_b)$$
$$= f_{ab} / (f * 100 * p_a * p_b) = M_{ba} / p_a$$

The total alignment score is the product of $Pam\text{-}k_{ab}$.

To avoid accuracy problems: $Pam\text{-}k_{ab} = 10 \log M^k_{ab} / p_b$

-> The total alignment score is the sum of $Pam\text{-}k_{ab}$.

Multiple Sequence Alignment

- Mult-Seq-Align allows to detect similarities which cannot be detected with Pairwise-Seq-Align methods.
- Detection of family characteristics.

Three questions:

1. Scoring
2. Computation of Mult-Seq-Align.
3. Family representation.

Multiple Sequence Alignment

Definition A *multiple alignment* of strings $S_1, S_2, \ldots, S_k$ is a series of strings with spaces $S'_1, S'_2, \ldots, S'_k$ such that

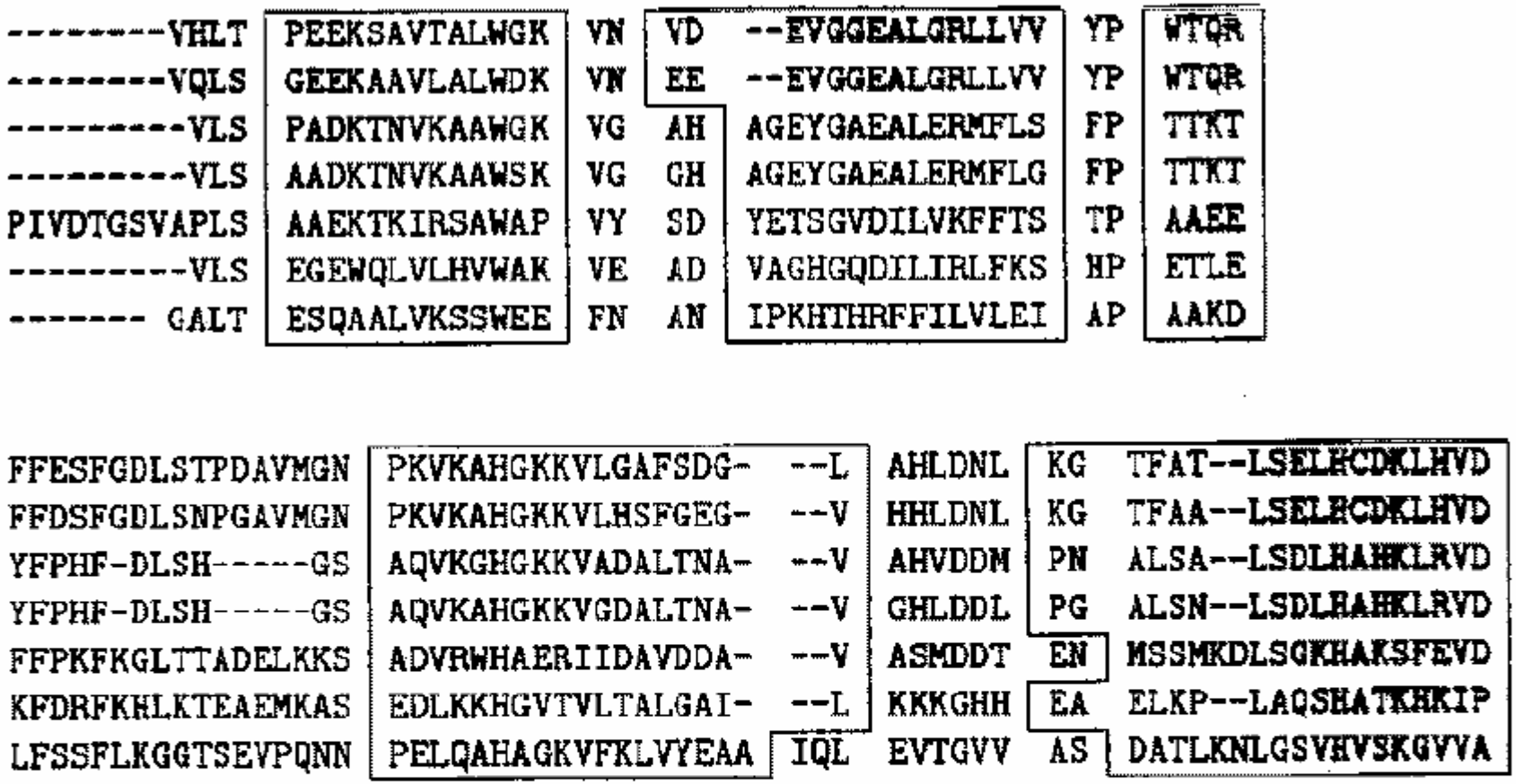
1. $|S'_1| = |S'_2| = \ldots = |S'_k|$.

2. S'_j is an extension of S_j, obtained by insertions of spaces.

For an example of a multiple alignment, see Figure 4.1.

```
AC..BCDB
.CADB.D.
ACA.BCD.
```

Figure 4.1: A multiple alignment of $ACBCBD$, $CADDB$ and $ACABCD$.



Scoring: **SP** (sum of pairs)

SP – the sum of pairwise scores of all pairs of symbols in the column.

(-,-) = 0

ρ_3(-,A,A) = (-,A)+(-,A)+(A,A)

SP Total Score = $\Sigma\ \rho_i$

Induced pairwise alignment

Induced pairwise alignment or
projection of a multiple alignment.

```
AC..BCDB
.CADB.D.
ACA.BCD.
```

$a(S_1, S_2)$
```
AC..BCDB
.CADB.D.
```

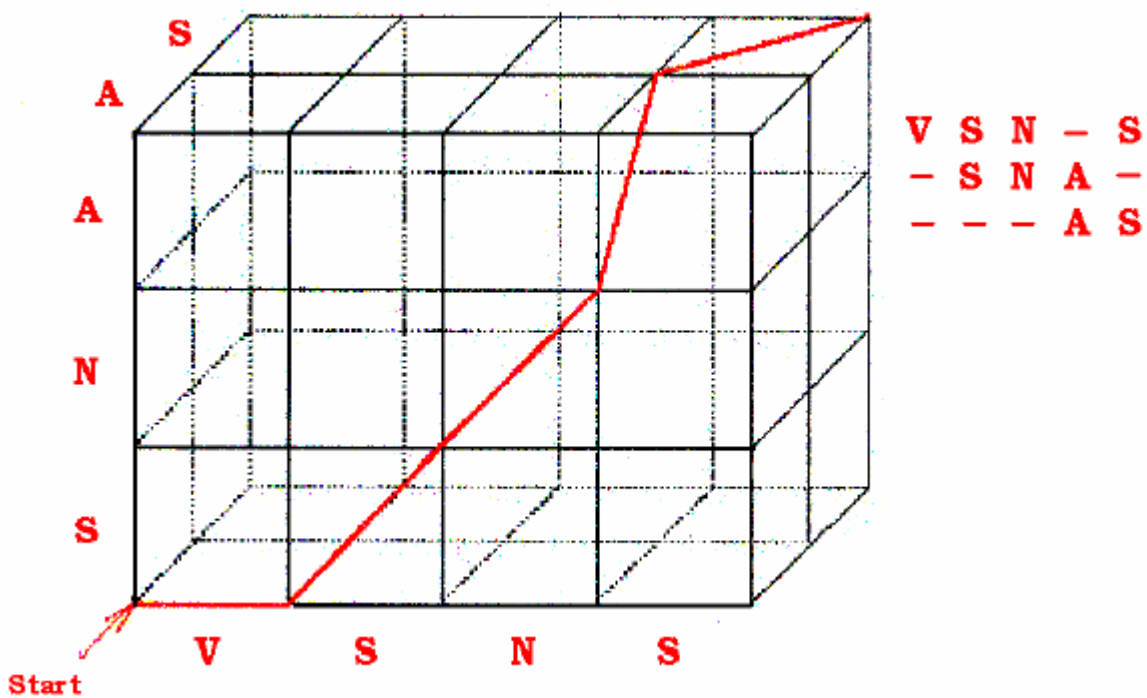
$a(S_2, S_3)$
```
.CADB.D.
ACA.BCD.
```

$a(S_1, S_3)$
```
AC..BCDB
ACA.BCD.
```

SP Total Score = $\Sigma_{i<j}$ score[$a(S_i, S_j)$]

(-,-) = 0

Dyn.Prog. Solution



We define

$$D(0, 0, \ldots, 0) = 0$$

And we calculate

$$D(j_1, j_2, \ldots, j_r) = min_{\epsilon \in \{0,1\}^n, \epsilon \neq 0}\{D(j_1 - \epsilon_1, j_2 - \epsilon_2, \ldots, j_r - \epsilon_r) + \rho(\epsilon_1 x_{j_1}, \ldots, \epsilon_r x_{j_r})\}$$

where ρ is the cost function, and

$$\epsilon = (\epsilon_1, \epsilon_2, ..., \epsilon_n) \in \{0, 1\}^n$$

Dynamic Programming Solution

- The best multiple alignment of r sequences is calculated using an r-dimensional hyper-cube
- The size of the hyper-cube is $O(\Pi n_i)$
- Time complexity $O(2^r n^r)$ * O(*computation of the ρ function*).

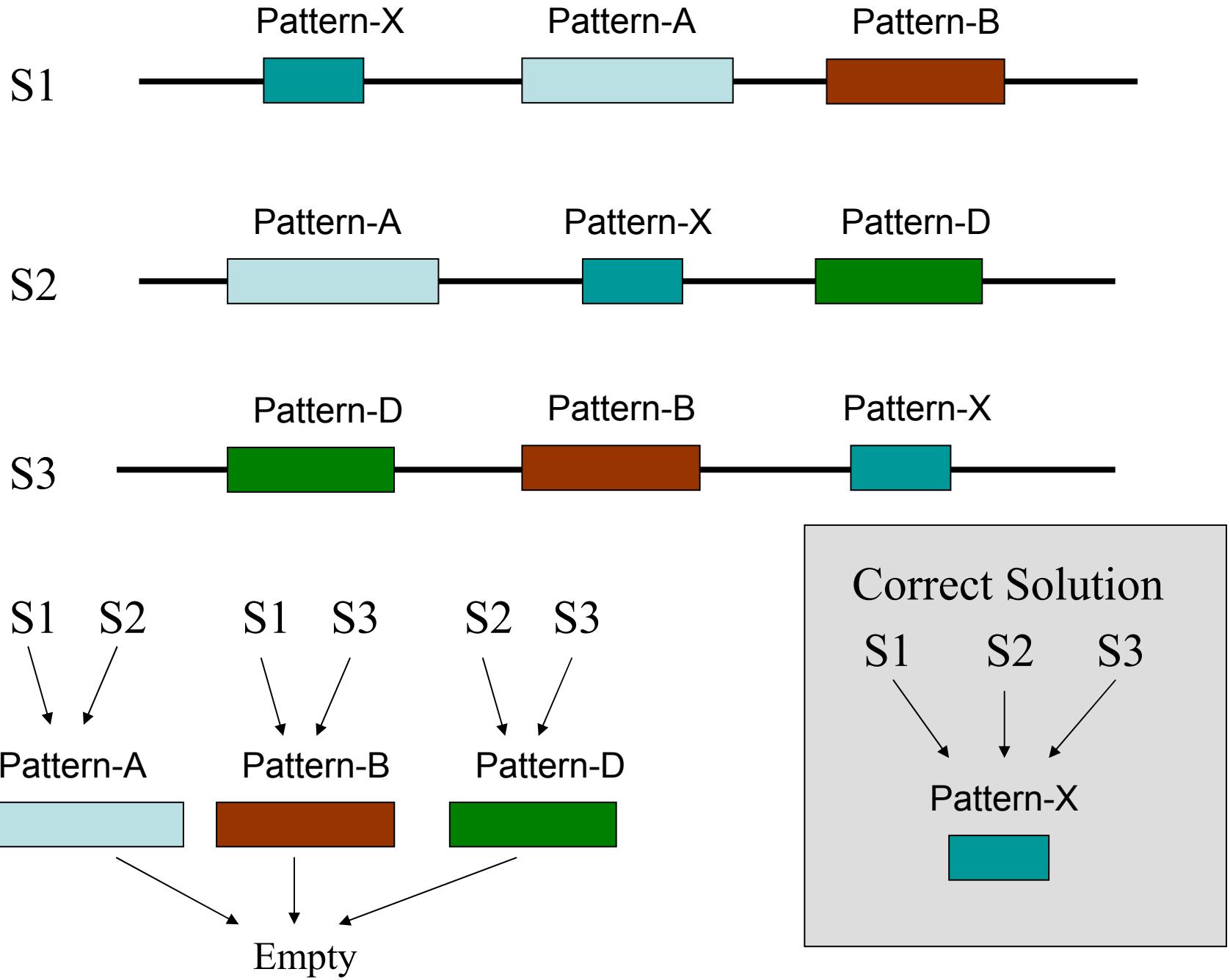
- Exact problem is NP-Hard (metrics: sum-of-pairs or evolutionary tree).

⇒ ***more efficient solution is needed***

Multiple Alignment from Pairwise Alignments ?

Problem:

- The best pairwise alignment does not necessary lead to the best multiple alignment.


Pattern-X
Pattern-A
Pattern-B
S1
Pattern-A
Pattern-X
Pattern-D
S2
Pattern-D
Pattern-B
Pattern-X
S3
S1 S2
S1 S3
S2 S3
Pattern-A
Pattern-B
Pattern-D
Empty
Correct Solution
S1 S2 S3
Pattern-X

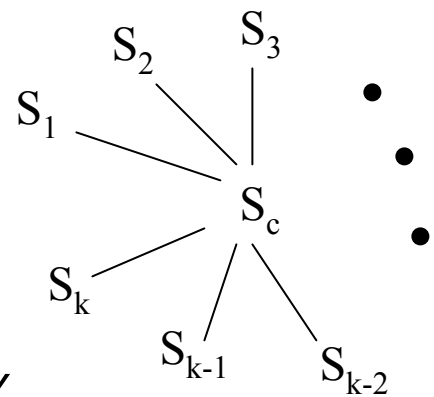
Center Star Alignment

(a) Scoring scheme – distance.

(b) Scoring scheme satisfies the triangle inequality: for any character a,b,c

$$dist(a,c) \leq dist(a,b) + dist(b,c)$$

(in practice not all scoring matrices satisfy the triangle inequality)

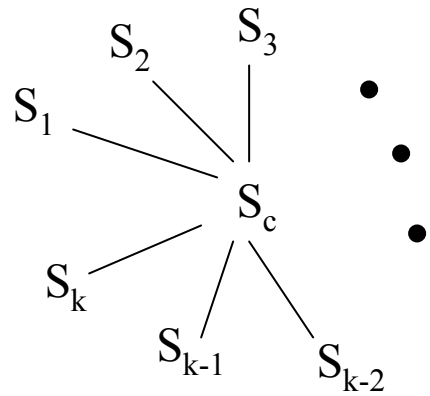


(c) $D(S_i, S_j)$ – score of the optimal pairwise alignment.

(d) $D(M) = \sum_{i<j} a_M(S_i, S_j)$ – score of the multiple alignment M.

(e) $a_M(S_i, S_j)$ – pairwise alignment/score induced by M.

The Center Star Algorithm:



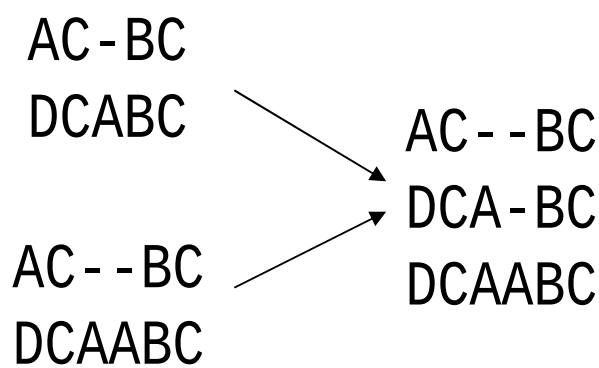
(a) Find S_c minimizing $\Sigma_{i \neq c} D(S_c, S_i)$.

(b) Iteratively construct the multiple alignment M_c:

1. $M_c = \{S_c\}$
2. Add the sequences in $S \setminus \{S_c\}$ to M_c one by one so that the induced alignment $a_{Mc}(S_c, S_i)$ of every newly added sequence S_i with S_c is optimal. Add spaces, when needed, to all pre-aligned sequences.

Running time:

$\binom{k}{2} * O(n^2)$.



$D(M_c)$ is at most twice the score of the $D(M_{opt})$

$D(M_c) / D(M_{opt}) \leq 2(k-1)/k \quad (< 2)$

Proof:

(a) $a(S_i, S_j) \geq D(S_i, S_j)$ (any induced align. is not better than optimal align.)
$a_{Mc}(S_c, S_j) = D(S_c, S_j)$

(b) $a_{Mc}(S_i, S_j) \leq a_{Mc}(S_i, S_c) + a_{Mc}(S_c, S_j) = D(S_i, S_c) + D(S_c, S_j)$
(follows from the triangle inequality)

(c) $2\,D(M_c) = \Sigma_{i=1..k} \Sigma_{j=1..k, j\neq i} a_{Mc}(S_i, S_j) \leq$

$\Sigma_{i=1..k} \Sigma_{j=1..k, j\neq i} (a_{Mc}(S_i, S_c) + a_{Mc}(S_c, S_j)) =$

$2(k-1) \Sigma_{j\neq c} a_{Mc}(S_c, S_j) =$

$2(k-1) \Sigma_{j\neq c} D(S_c, S_j)$

(d) $k \sum_{j=1..k, j\neq c} D(S_c, S_j) = \sum_{i=1..k} \sum_{j=1..k, j\neq c} D(S_c, S_j) \leq$

$\sum_{i=1..k} \sum_{j=1..k, j\neq i} D(S_i, S_j) \leq$

$\sum_{i=1..k} \sum_{j=1..k, j\neq i} a_{Mopt}(S_i, S_j) =$

$2\, D(M_{opt})$

(e) $\rightarrow$ $2\, D(M_c) \leq 2(k-1) \sum_{j\neq c} D(S_c, S_j)$

$k \sum_{j\neq c} D(S_c, S_j) \leq 2\, D(M_{opt})$

$\rightarrow$ $D(M_c)/(k-1) \leq \sum_{j\neq c} D(S_c, S_i)$

$\sum_{j\neq c} D(S_c, S_i) \leq 2\, D(M_{opt})/k$

$\rightarrow$ $D(M_c) / D(M_{opt}) \leq 2(k-1)/k$