



# Computational Molecular Biology

Erwin M. Bakker

Lecture 3, mainly from material by R. Shamir [2]  
and H.J. Hoogeboom [4].



# Pairwise Sequence Alignment

- Biological Motivation
- Algorithmic Aspect
  - Recursive functions
  - Formal definitions
  - Dynamic Programming
  - Complexity
- Heuristics



# Molecular Biology Sequences

DNA      A, T, C, G

RNA      U, A, G, C

Protein    A, R, D, N, C  
            E, Q, G, H, I  
            L, K, M, F, P  
            S, T, W, Y, V

# Global Alignment

## Dynamic Programming

Sequence S -acgctg

Sequence T catg-t-

acgctg-

-ca-tgt

acgctg-

-c-atgt

Sequence S

$\langle \begin{matrix} \text{acgctg} \\ \text{catgt} \end{matrix} \rangle$

Sequence T

	-	c	a	t	g	t
-	0	-1	-2	-3	-4	-5
a	-1	-1	1	0	-1	-2
c	-2	1	0	0	-1	-2
g	-3	0	0	-1	2	1
c	-4	-1	-1	-1	1	1
t	-5	-2	-2	1	0	3
g	-6	-3	-3	0	3	2

-  
c

a  
a

c  
t

g  
g

c  
-

t  
t

g  
-

-  
t

Trace back to obtain an optimal global alignment.

Note that, here three such optimal global alignments exist.



# Local Alignment

## Local Alignment Problem

Given two sequences  $S$  and  $T$ , find subsequences  $s$  of  $S$  and  $t$  of  $T$  whose similarity is maximal over all pairs of subsequences of  $S$  and  $T$ .

Note that, a subsequence here is a **contiguous** subsequence.

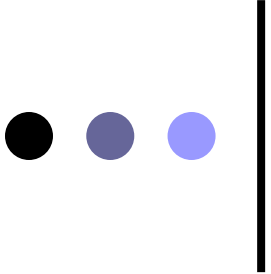


# Local Alignment

## Motivation (1/2)

### **Coding and non-coding regions of DNA**

- Mutations in **non-coding regions (introns)** are expected to be more likely than mutations in **coding regions (exons)**. As mutations in **exons** will have a direct impact on the organism.
- Therefore a best match between two stretches of DNA from different species is most likely between **2 exons (i.e., subsequences)**.



# Local Alignment

## Motivation (2/2)

### Protein Domains

- o Different kind of proteins and proteins of different species often show **local similarities**, so called **homeoboxes** (most probably functional **subunits**).



# Local Alignment Example

S:    G G T C T G A G  
T:    A A A C G A

Match = 2, indel/substitution = -1

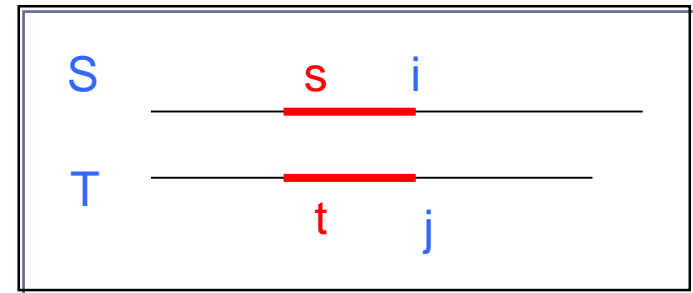
Best local alignment

S:    G G T C T G A G  
T:    A A A C \_ G A



# Local Alignment

## Local Suffix Alignment



### Definition

Given sequences  $S$  and  $T$ , and indices  $i$  and  $j$ , the **local suffix alignment problem** is finding a (possibly empty) suffix  $s$  of  $S_{1\dots i}$  and a (possibly empty) suffix  $t$  of  $T_{1\dots j}$  such that the score of the alignment of  $s$  and  $t$  is maximal over all alignments of suffixes of  $S_{1\dots i}$  and  $T_{1\dots j}$ .

### Remark:

The solution of the **local alignment problem** is the same as the **maximal solution** to the **local suffix alignment problem** over all  $i$  and  $j$ .



# Local Alignment Algorithm

Let  $A(i,j)$  the value of the optimal **local suffix alignment** for a given pair  $i,j$  of indices

Let the weights be limited to

$$\begin{aligned} \sigma(x,y) &\geq 0, && \text{if } x, y \text{ match, and} \\ \sigma(x,y) &\leq 0, && \text{if } x, y \text{ do not match or} \\ &&& \text{one of them is equal to} \\ &&& \text{a space} \end{aligned}$$

**Note:** the maximal  $A(i,j)$  over all  $i, j$  is the value we are looking for.



# Local Alignment Algorithm

## Algorithm Sketch

- Compute the *local suffix alignment* (for all  $i$  and  $j$ ) of  $S'_i = S_{1\dots i}$  and  $T'_j = T_{1\dots j}$ . Using the *global alignment algorithm* where the prefixes of  $S'$  and  $T'$  whose alignments are  $\leq 0$  are discarded, i.e., subsequences may start from indices  $\geq 1$ .
- Search the results and find the indices  $i^*$  and  $j^*$  of  $S$  and  $T$  respectively, after which the similarity (*obtained by local suffix alignment*) only decreases.

# Local Alignment Algorithm

Let  $A(i,j)$  be the optimal local suffix alignment score of  $S_{1\dots i}$  and  $T_{1\dots j}$ , where  $0 \leq i \leq n$ , and  $0 \leq j \leq m$ , then:

$$\forall i, j: A(i,0) = 0, A(0, j) = 0$$

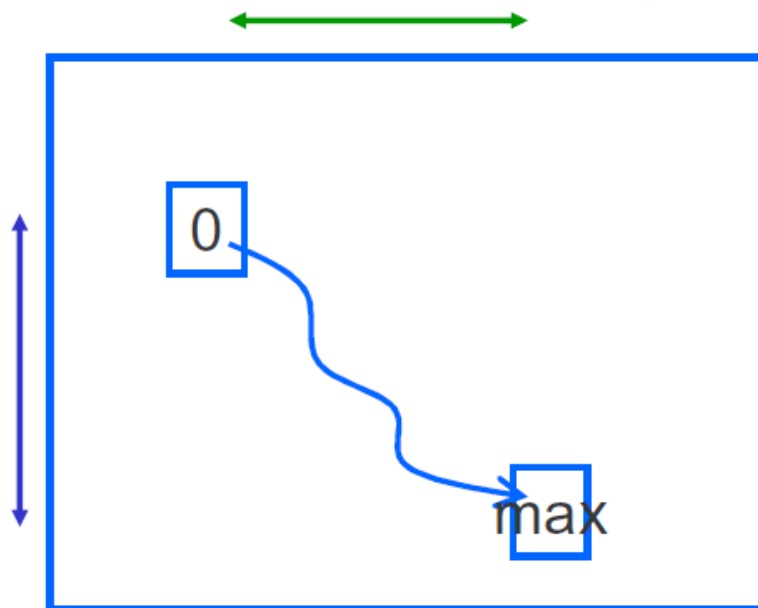
$$A(i, j) = \max \begin{cases} 0 \\ A(i-1, j-1) + \sigma(S_i, T_j) \\ A(i-1, j) + \sigma(S_i, -) \\ A(i, j-1) + \sigma(-, T_j) \end{cases}, \text{ for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m$$

Compute  $i^*$  and  $j^*$  such that  $A(i^*, j^*) = \max_{1 \leq i \leq n, 1 \leq j \leq m} A(i, j)$ .  
This value is the optimal local alignment score.

# Local Alignment

Obtain optimal local alignment sequences by backtracking

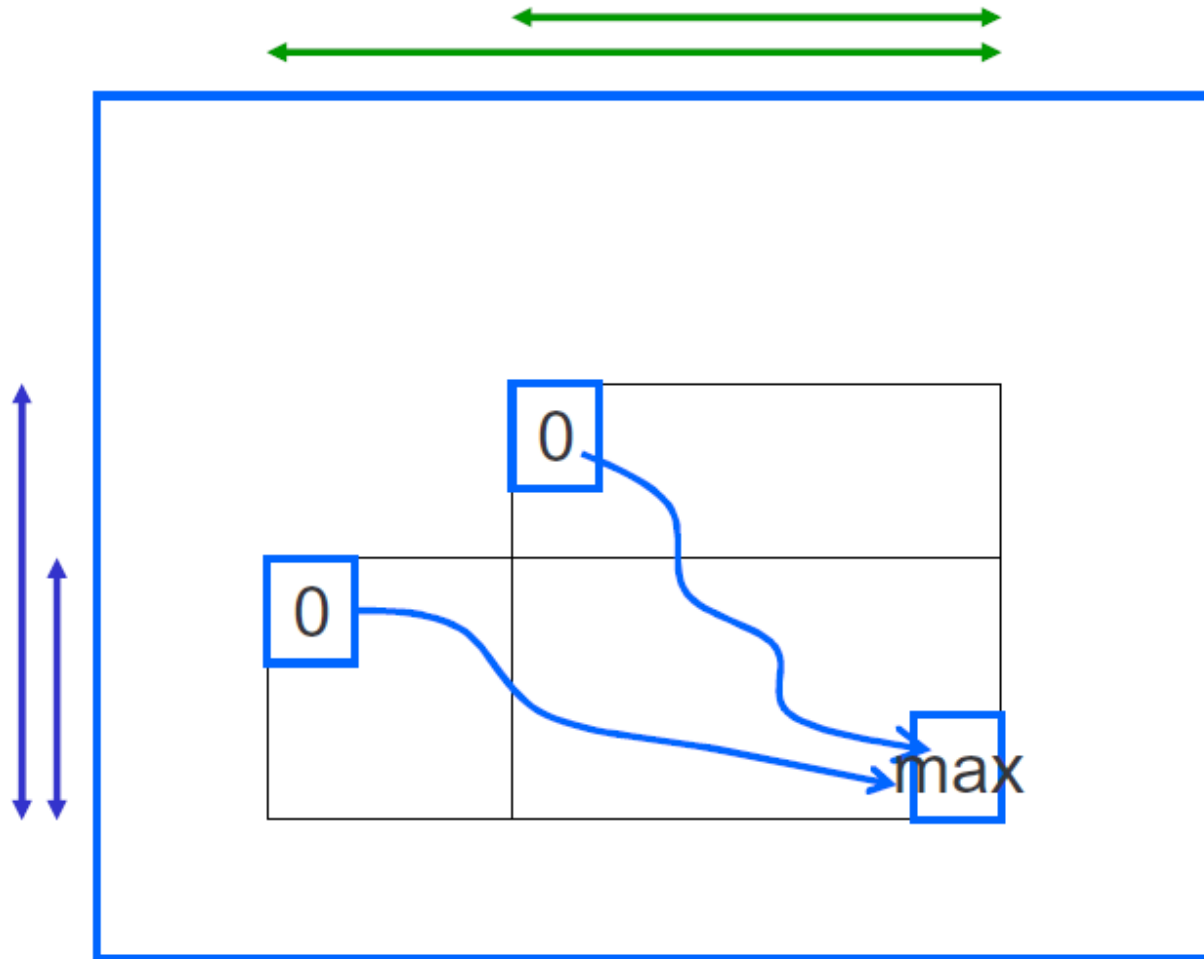
$$a[i, j] = \max \begin{cases} a[i, j-1] + g \\ a[i, j] + \sigma( s[i], t[j] ) \\ a[i-1, j] + g \\ 0 \end{cases}$$



solution (*traceback*)  
from max to 0

# Local Alignment

Obtain optimal local alignment sequences by backtracking





# Local Alignment Complexity

**Lemma:** Local alignment can be solved in linear space

- The optimal local alignment identifies the subsequences  $s$  and  $t$  whose global alignment is optimal over all pairs of subsequences.
- Hirschberg's method for global alignment can then be used to find the actual alignment of subsequences  $s$  and  $t$ .
- Using the recursion  $i^*$  and  $j^*$  can be calculated using two rows only.
- Hence the end points  $(i^*, j^*)$  can be computed in linear space.
- Finding the start positions can be done using reverse dynamic programming.

□



# Local Alignment Complexity

Time complexity  $O(mn)$

Space complexity  $O(n+m)$





# End-Space Free Alignment

## End-Space Free Alignment Problem

**Input:** Two sequences  $S$  and  $T$ .

**Question:**

Find a best alignment 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, i.e, complete overlap is possible) is a suffix.

Hereby costs of **indels** at the end or beginning of the sequences are not counted.



# End-Space Free Alignment

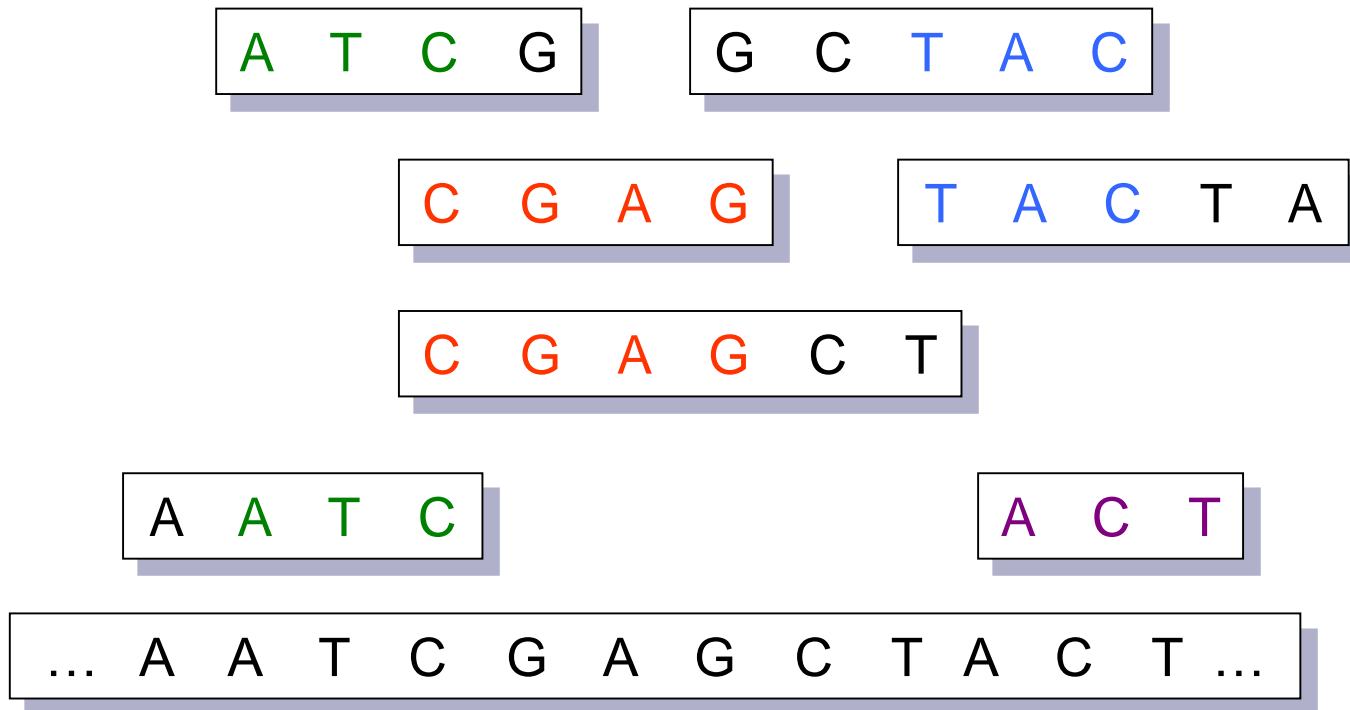
## Motivation

### Shotgun Sequence Assembly

- A large number of partially overlapping sequences coming from many copies of one original but unknown DNA sequence  $R$  has to be searched for pairs of overlapping subsequences in order to reconstruct the original DNA sequence.
- Two subsequences from **different parts** of  $R$  will have a **low global alignment score** as well as a **low end-space free alignment score**.
- Two overlapping subsequences from **the same part** of  $R$  will still have a **low global alignment score** but a **high end-space free alignment score**.

# End-Space Free Alignment

## Example





# End-Space Free Alignment

## Example

Example Consider the sequences:

$S = c a c t g t a c$

$T = g a c a c t t g$

Assigning value of 2 for match, and -1 for indel/substitution, the best Global alignment will have value 1 and will look like this:

$S = c a c - - t - g t a c$

$T = g a c a c t t g - - -$

while End-space free alignment will have value 9 and will look like this:

$S = - - c a c - t g t a c$

$T = g a c a c t t g - - -$

The two leading spaces at the left end of the alignment are free, as well as the three trailing spaces at the right end.



# End-Space Free Algorithm

## Initial Conditions

Set the initial conditions to allow zero weight to leading **indel** operations in (at most) one of the sequences.

## Compute Optimal Value

- Fill the table with the values of  $A(i,j)$  (as before).
- Then search for the maximal value in either of the 'ending rows', thus allowing (at most) one sequence to end before the other, with zero weight for all **indel** operations from there on.
- This value is the best value.

## Determine Sequence

The aligned sequence is tracked from cell  $(0,0)$  in the table until the end of one sequence (bottom row /right most column). From there on, all indel operations until cell  $(n,m)$  are not counted in the total value (though they are present in the table).

# End-Space Free Alignment

Define the End-Space Free Alignment score as  $A(S,T)$ , then:

$$\forall i, j: A(i,0) = 0, A(0, j) = 0$$

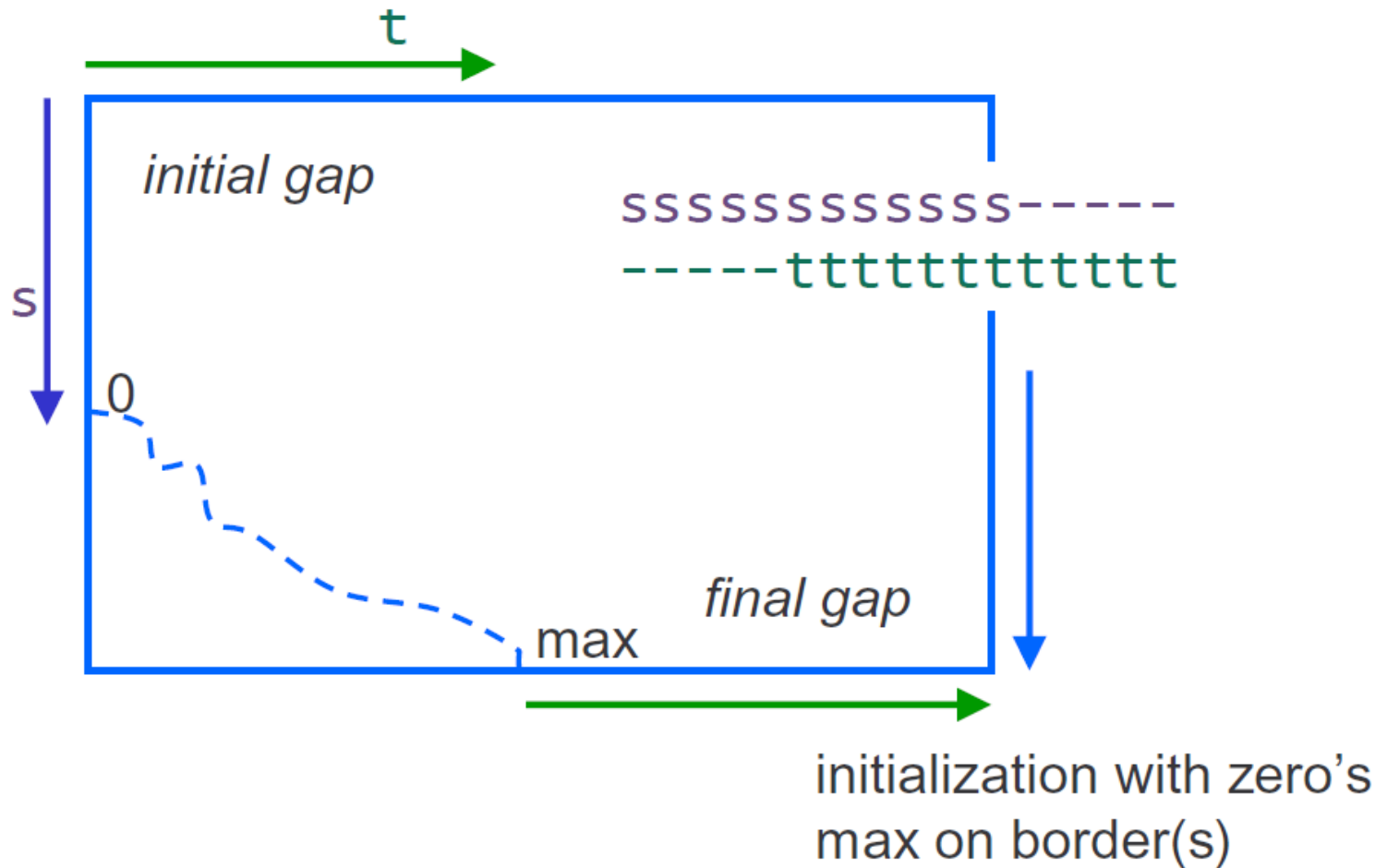
$$A(i, j) = \max \begin{cases} A(i-1, j-1) + \sigma(S_i, T_j) \\ A(i-1, j) + \sigma(S_i, -) \\ A(i, j-1) + \sigma(-, T_j) \end{cases}, \text{ for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m$$

$$\text{Search for } i^* \text{ such that: } A(i^*, m) = \max_{1 \leq i \leq n, m} A(i, j)$$

$$\text{Search for } j^* \text{ such that: } A(n, j^*) = \max_{n, 1 \leq j \leq n} A(i, j)$$

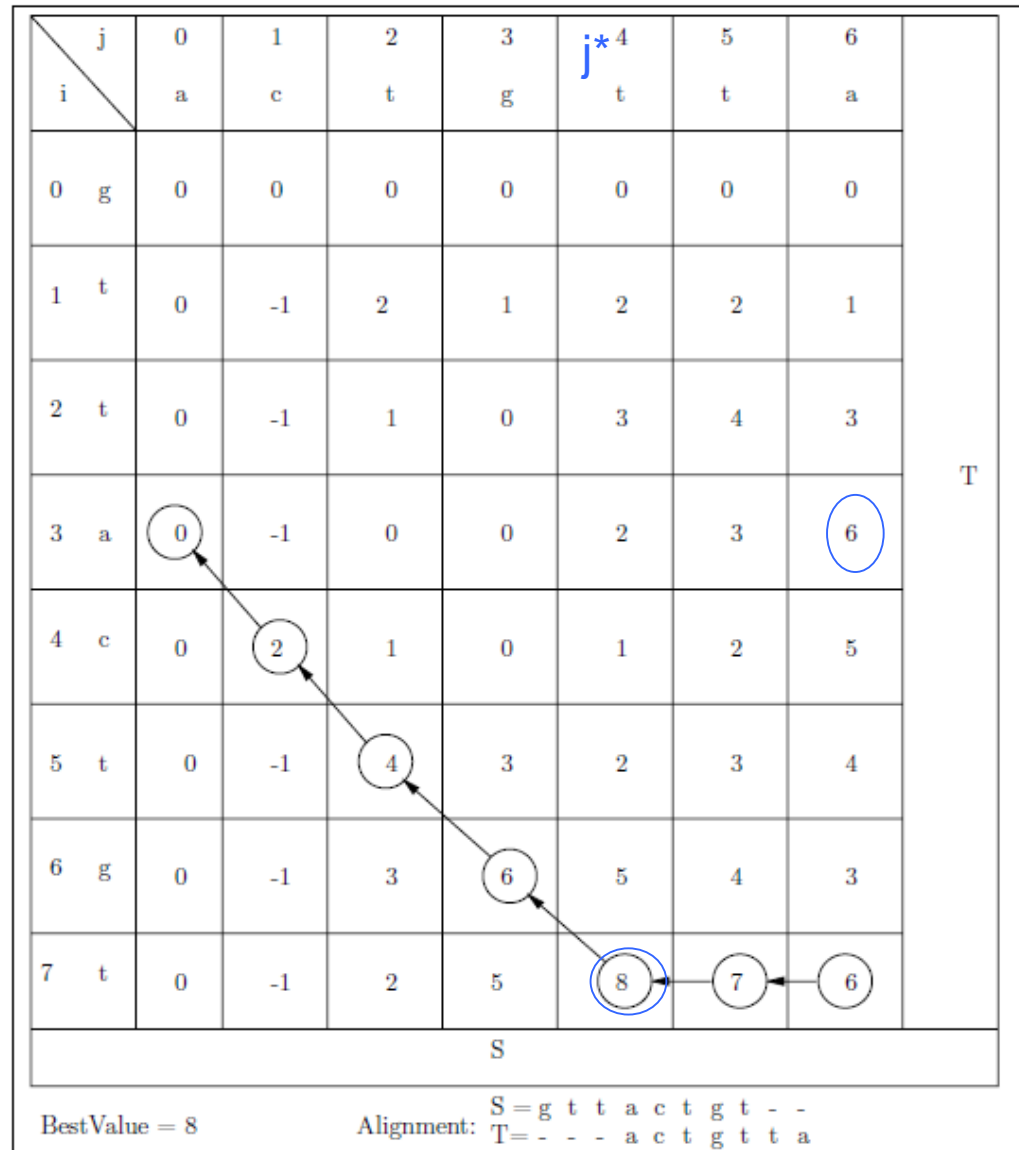
$$\text{Define alignment score: } A(S, T) = \max \begin{cases} A(n, j^*) \\ A(i^*, m) \end{cases}$$

# End-Space Free Alignment



# End-Space Free Alignment

$j^*$  is where S ends before T  
 $i^*$  is where T ends before S







# Gap Penalty

## Definition:

A **gap** is a maximal, consecutive run of spaces in a single sequence of a given alignment.

## Definition:

The **length of a gap** is the number of *indel* operations.



# Gap Penalty

## Motivation

### DNA Sequences

- Insertion or deletion of an entire subsequence often occurs as a single mutational event.
- A set of these events can create many gaps of varying sizes.

### Protein Sequences

- Two protein sequences may be similar except for some **subunits** that exist in the one but not the other.



# Gap Penalty

## Motivation

### cDNA matching

- DNA transcribes to pre-mRNA the complement of the gene's DNA (with introns and exons). After splicing mRNA (only transcribed exons) results.
- When mRNA is captured from the cell, so called cDNA can be transcribed which has to be matched with the DNA in order to find the gene from which it originally resulted.
- cDNA does not contain the gaps that the original DNA exhibits because of the intron regions.

# Gap Penalty

Constant Gap Penalty

$$g(k) = k * g$$

Affine Gap Penalty

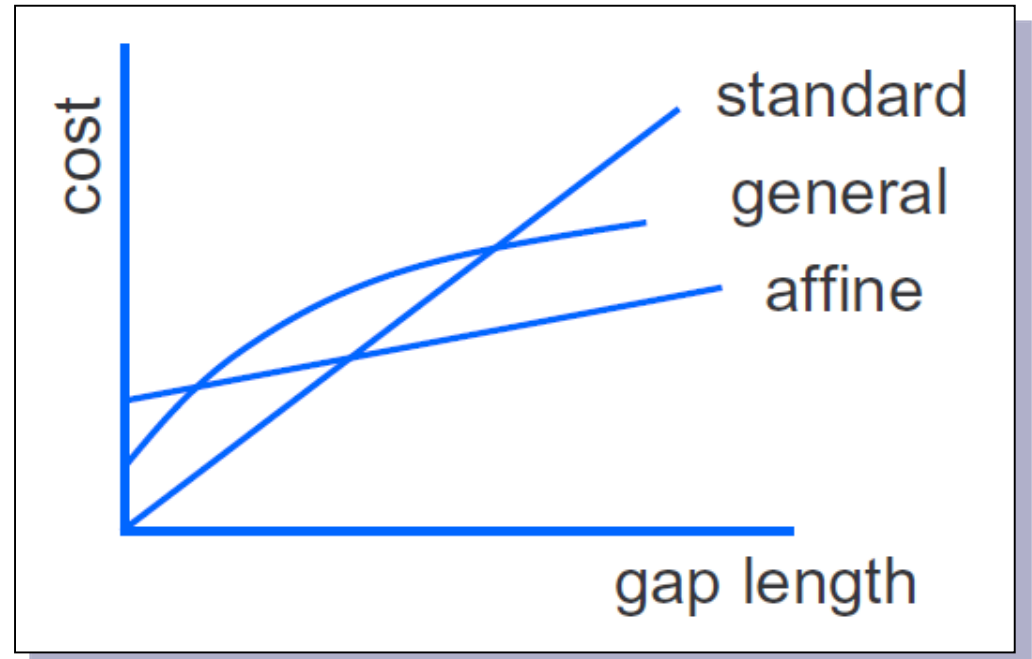
$$g(k) = k * g + s$$

Convex Gap Penalty

each additional gap contributes less to the gap than the previous space.

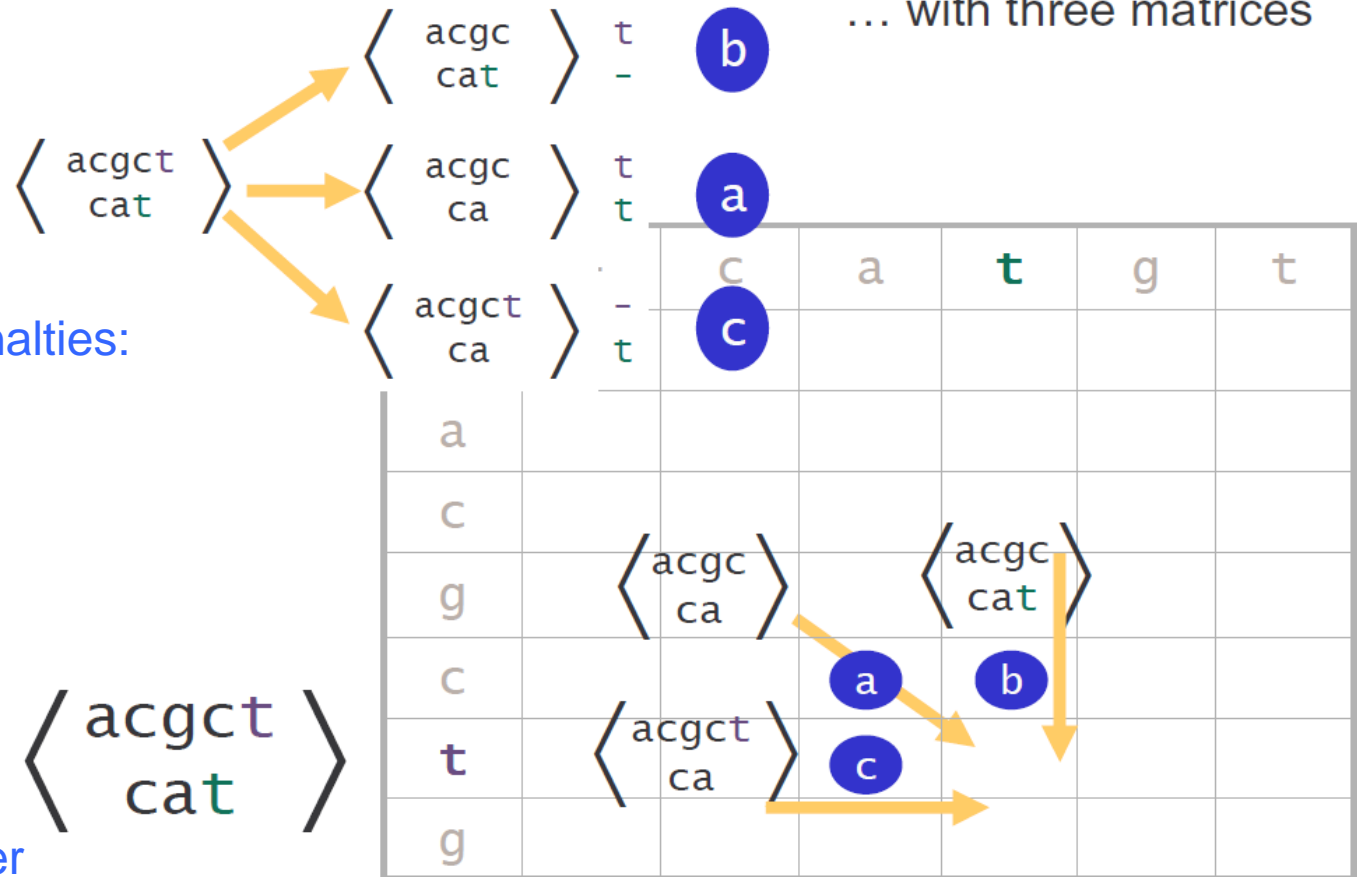
General Gap Penalty

$g(k)$  arbitrary



# Algorithm for Sequence Alignment with Affine Gap Penalty Model

... with three matrices



Using 3 Matrices for tracking the gap penalties:

B: S -----i  
T ----j\_\_\_\_

A: S -----i  
T -----j

C: S ----i\_\_\_\_  
T -----j

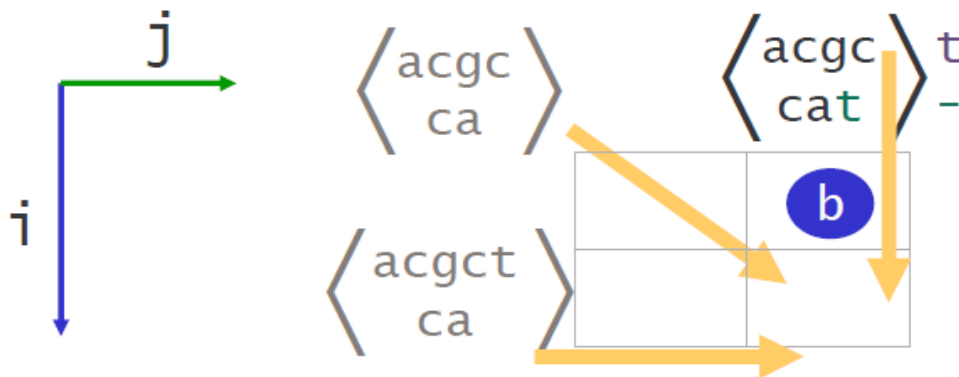
V taken the max over the three matrices.

# Algorithm for Sequence Alignment with Affine Gap Penalty Model

dynamic programming  
with three matrices

$$g(k) = g + ke \quad \text{open gap, extend gap}$$

$$b[i, j] = \max \begin{cases} a[i-1, j] & -g-e \\ b[i-1, j] & -e \\ c[i-1, j] & -g-e \end{cases} \quad \leftarrow \text{gap was already open}$$



Still a time complexity of  
 $O(mn)$  vs.  
 $O(mn^2 + m^2n)$  for a  
general gap penalty



# References

- [1] R. Shamir, [Algorithms in Molecular Biology \(2009 version\)](#), 2001.
- [2] R. Shamir, [Pairwise Alignment](#), Scribe from Lecture, 2001.
- [3] H.J. Hoogeboom, A.P. Gultyaev, [Lecture Notes](#), under development, 2009.
- [4] H.J. Hoogeboom, [Lecture Slides](#), 2009.



# Problems to be Solved

## Humane Genome

- 23 pairs of chromosomes
- $3.1 \times 10^9$  bases
- 30.000 – 40.000 genes
- Protein variants from splicing  $\sim 1 \times 10^6$
- 99.9% similarity between humans





# Problems to be Solved

## GenBank (February 2014)

- $1,58 \times 10^{11}$  bases
- $1,71 \times 10^8$  sequence records

## Whole Genome Shotgun (WGS) submissions (February 2014)

- $5,91 \times 10^{11}$  bases
- $1,40 \times 10^8$  sequence records



# Problems to be Solved

The optimal alignment algorithms are too slow

- Long strings
- Huge databases

Multiple Alignment

- NP-Complete
- Until now exponential solutions only

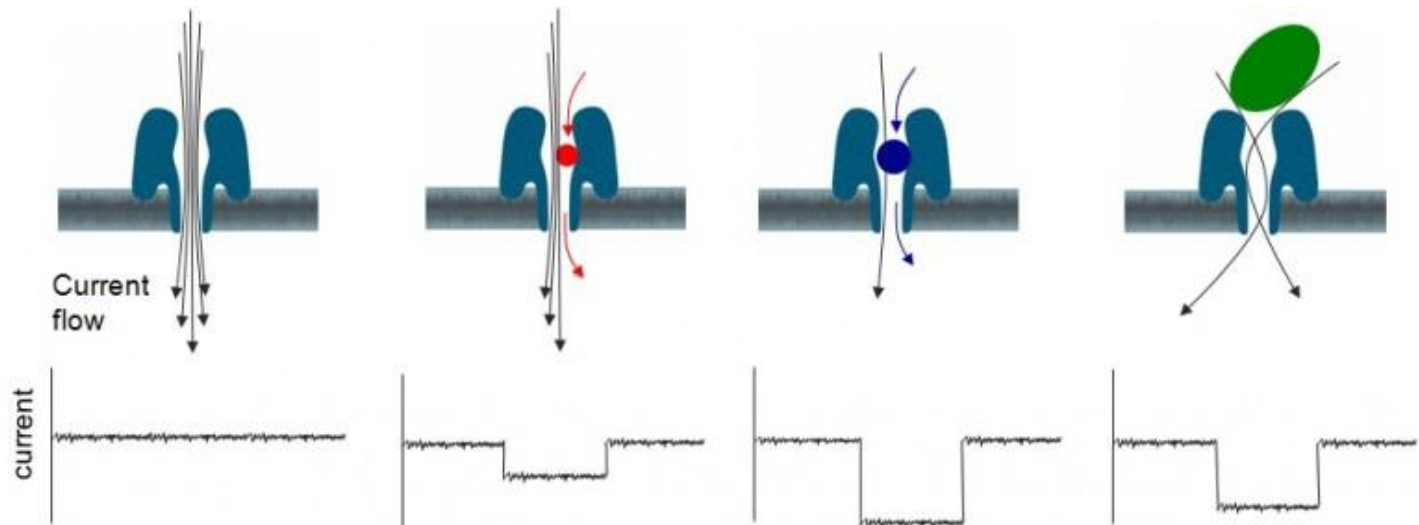
Heuristics are needed

- FASTA along diagonals
- BLAST extend from minimal close matches

● ● ● | A USB-powered  
DNA-Sequencer  
MinION NanoPores



- 512 nanopores x 15bp/sec => ~7500 bp/sec
- 6 hours lifetime =>  $150 \times 10^6$  bp
- \$900 usb-powered DNA-Sequencer



- ● ● | A USB-powered  
DNA-Sequencer  
MinION NanoPores



UNREGISTERED :)  
downloadhelper.net



# Possible Solutions

## Problem

- Database size:  $>10^{11}$
- Query size:  $\sim 10^3$
- Complexity of optimal (local) alignment algorithms such as the Smith-Waterman Algorithm (1981):  $O(nm)$ .
- May take many hours of processing time.

## Solutions

- Hardware dynamic programming implementations
- Parallel implementation of sequence alignment algorithms (GPUs and FPGAs)
- Using heuristics to improve speed



# Some Figures

- Intel i7 ~ 0.15 TFlop/s
- NVidia Kepler ~ 1 – 3 TFlop/s
- Tianhe-2 ~ 33 862.7 TFlop/s

1 TFlop/s =  $1.0 \times 10^{12}$  Flop/s

Time complexity Smith-Waterman:

- $O(10^3 \times 10^{11}) = O(10^{14})$
- => Several hours or days on an Intel i7 machine

## Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P



Site:	National Super Computer Center in Guangzhou
Manufacturer:	NUDT
Cores:	3,120,000
Linpack Performance (Rmax)	33,862.7 TFlop/s
Theoretical Peak (Rpeak)	54,902.4 TFlop/s
Power:	17,808.00 kW
Memory:	1,024,000 GB
Interconnect:	TH Express-2
Operating System:	Kylin Linux
Compiler:	icc
Math Library:	Intel MKL-11.0.0
MPI:	MPICH2 with a customized GLEX channel



# Heuristic Alignment Algorithms

## Heuristics

- Heuristic algorithms aim at speeding up to  $O(\max(n,m))$  at the price of possibly missing the best scoring alignment
- Allow a preprocessing phase on not frequently updated databases
- Homologous sequences are expected to contain many segments with matches or substitutions without indels and gaps
- Substitutions are expected to be much likely than indels



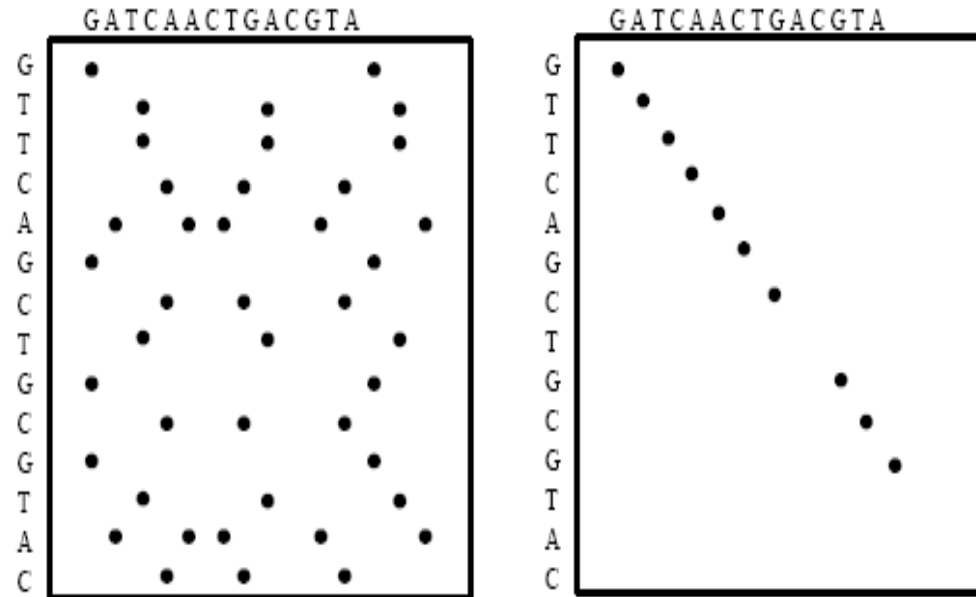


# Heuristic Alignment Algorithms

Two well known programs

- **BLAST**: Basic Local Alignment Search Tool
- **FASTA**: Fast Alignment Tool
- Both find high scoring local alignments between a query sequence and a target database
- **Basic idea**: first locate high-scoring short stretches and then extend them

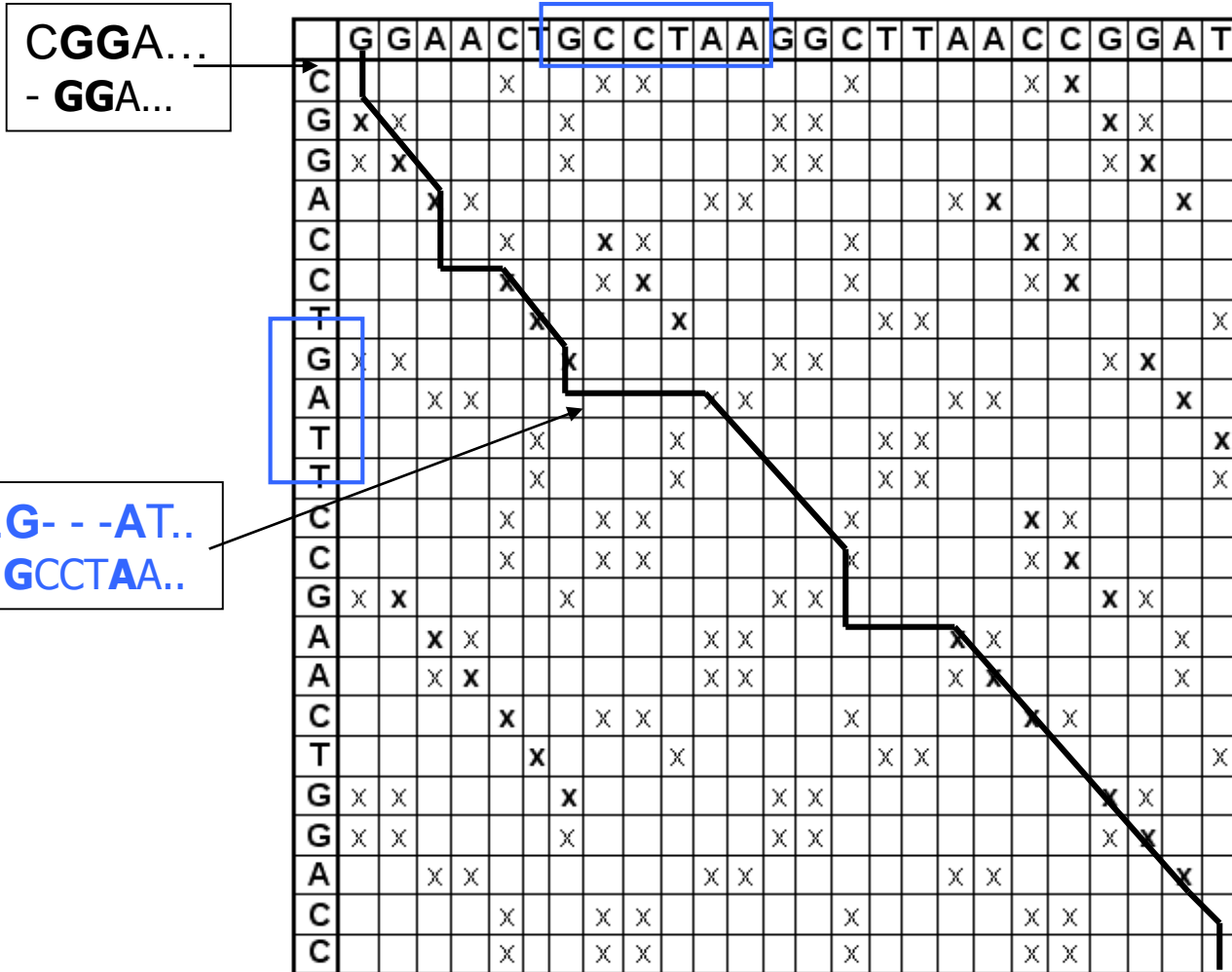
# Dot Matrix Alignment Method



**Dot Matrix Plot:** Boolean matrices representing possible alignments that can be detected visually

- Extremely simple but
- $O(n^2)$  in time and space
- Visual inspection

# TGCA Matrix Plot





# FASTA

## Fast All [9, 10]

**FASTA** a heuristic algorithm for sequence alignment by Pearson & Lipman (1985, improved in 1988)

### Motivation

- Good alignments should contain many exact matches
- Focus on segments of the compared sequences with absolute identity
- Finding these segments by using the logic of the dot matrix method



# FASTA

**FASTA** a heuristic algorithm for sequence alignment by Pearson & Lipman (1985, improved in 1988)

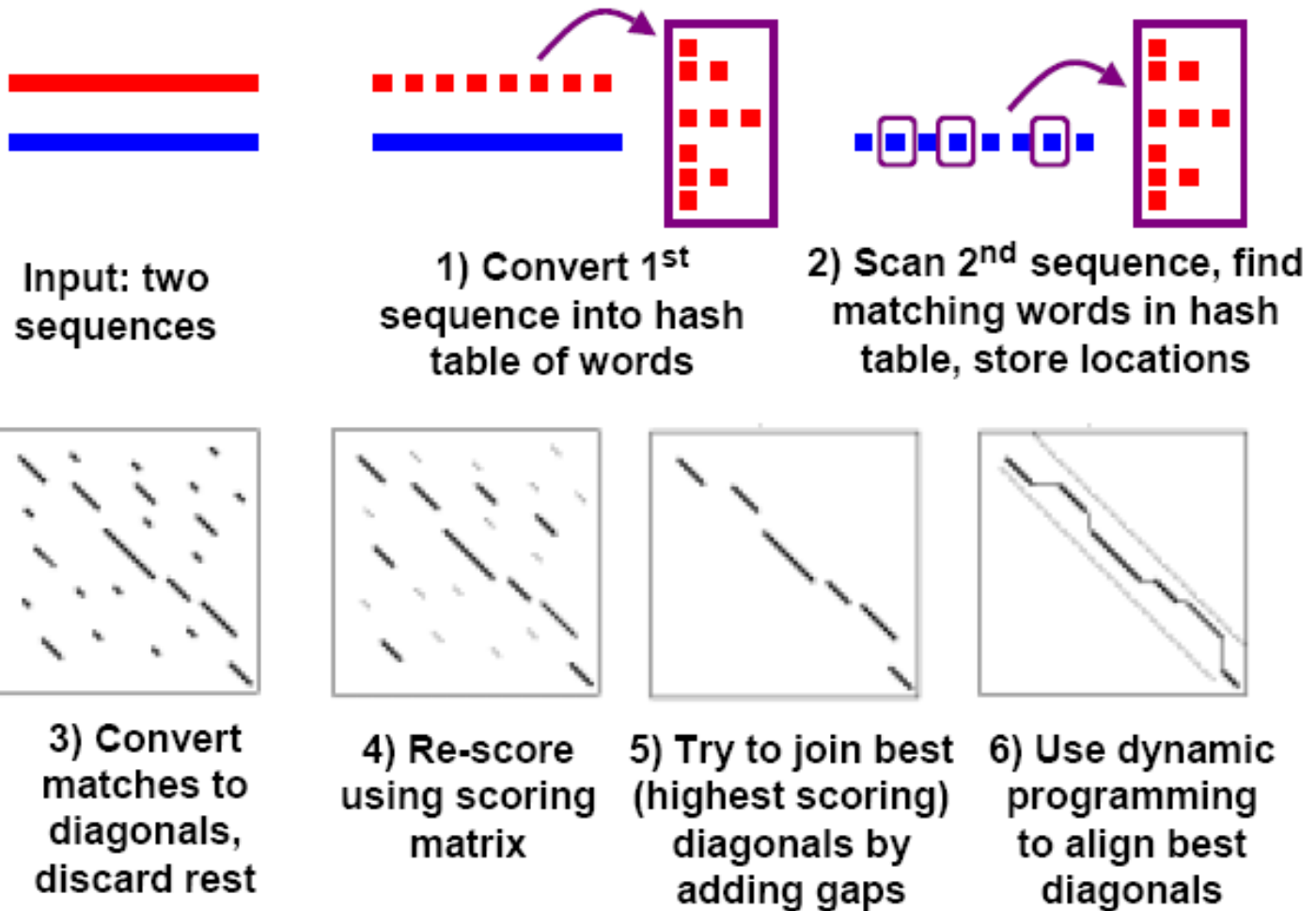
- View sequences as sequences of *short words* (*k-tuple*)
  - DNA:  $k = 6$  bases, protein:  $k = 1$  or  $2$  amino acids
  - $k$  is a trade-off between speed and sensitivity
- *Hot-spots* are matching *k-tuple* substrings of the two compared strings. Clearly, *hot-spots* are located along the diagonals of the dot-plot matrix.
- A *diagonal run* is a sequence of nearby hot spots on the same diagonal
  - spaces between these *hot-spots* are allowed
  - possibly multiple *diagonal runs* along the same diagonal



# FASTA Algorithm

1. Look for **hot spots**
2. Find **10 best diagonal runs**
3. Compute the **best scoring diagonal run  $init_1$**  and discard low scoring runs
4. Combine **close diagonal runs** in order to compute the best single larger high scoring **alignment  $init_n$** .
5. Determine a diagonal band (using parameter  $k$ ) around  **$init_1$**  and use dynamic programming to compute the alternative optimal local alignment  **$opt$**  within the band.
6. Rank the sequences in the database according to  **$init_n$**  or  **$opt$** ,
7. Use dynamic programming to align query against each of the highest scoring results (sequences) from the database.

# FASTA (Fast Alignment)





# FASTA Algorithm

## 1. Look for hot spots

- Make a hash table containing an entry for each possible *k-tuple* (DNA =>  $4^k$  with  $k = 6$ , Protein =>  $20^k$  with  $k = 2$ ).
- Preprocess the complete database and store for each sequence where a *k-tuple* appeared.
- Now scan the query by shifting a window with width *k* along the query sequence and use the hash table for retrieving the locations in the database.

Preprocessing the database takes  $O(c.m)$  where  $c$  is the number of sequences of length  $m$  in the database.

Query processing has complexity  $O(n)$ , where  $n$  the length of the query sequence





# FASTA Algorithm

## 2. Find 10 best diagonal runs

- Each **hot-spot** is given a **positive score**.
- The space between consecutive **hot-spots** along the diagonal is given a negative score that decreases with length.
- The score of a **diagonal run** is the sum of the **hot-spots** plus the scores of the spaces in between.
- Find the **10** highest scoring **diagonal runs**.

Note that:

- We saw that hashing can find **hot-spots** in  $O(n)$  time
- Now **diagonal runs** can be formed from **hot-spots** quickly by sorting the **hot-spots** by *diagonal-position* ( $i - j$ ).



# FASTA Algorithm

3. Compute the best scoring diagonal-run  $init_1$  and discard low scoring diagonal-runs
  - The (*10 best*) diagonal-runs found thus far are evaluated using a substitution matrix based scoring function (1988 improvement).
    - Note that there are *no indels* along a (diagonal) run.
  - The highest scoring diagonal-run  $init_1$  is determined.
  - Furthermore, diagonal-runs that score below a certain threshold are discarded.

# FASTA Algorithm: $init_n$

4. Combine close diagonal-runs in order to compute the best single larger high scoring alignment  $init_n$ .
  - Take the high scoring diagonal-runs found in Step 3.
  - Make a directed graph  $G$  as follows:
    - Each vertex is a diagonal-run found in Step 3 with weight equal to that score.
    - There is an edge from  $u$  to  $v$ , if  $v$  starts at a higher row and column than those at which  $u$  ends.
    - The (negative) weight of  $(u,v)$  depends on the number of gaps that would be created to make an alignment that contains  $u$  followed by  $v$ .
  - Find a maximum weight path in  $G$  and call this  $init_n$

**Complexity:** Given a DAG  $G=(V,E)$ , finding a max-weight path can be done in  $O(|V|+|E|)$  time using dynamic programming.



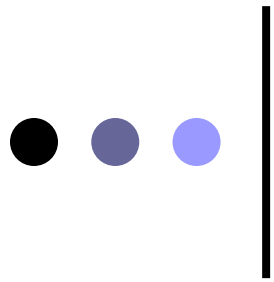
# FASTA Algorithm: *opt*

5. Determine a diagonal band (using parameter  $k$ ) around diagonal-run  $init_1$ 
  - The idea behind this is that the best local alignment containing  $init_1$  is most likely within this band, as  $init_1$  is a high scoring diagonal-run lying within this band.
  - The bandwidth is depending on  $k$ .
  - Use dynamic programming to compute an optimal local alignment called *opt* within the band.



# FASTA Algorithm

6. Rank the sequences in the database and determine optimal local alignments to the query sequence of the highest scoring database sequences.
  - Each sequence in the database has an  $init_n$  or  $opt$  score with respect to the query sequence.
  - Rank the sequences in the databases according to their  $init_n$  or  $opt$  score.
  - Select the highest scoring sequences from the database.
  - Use the full dynamic programming algorithm to align the query against each of these highest scoring results from the database.



# FASTA Algorithm

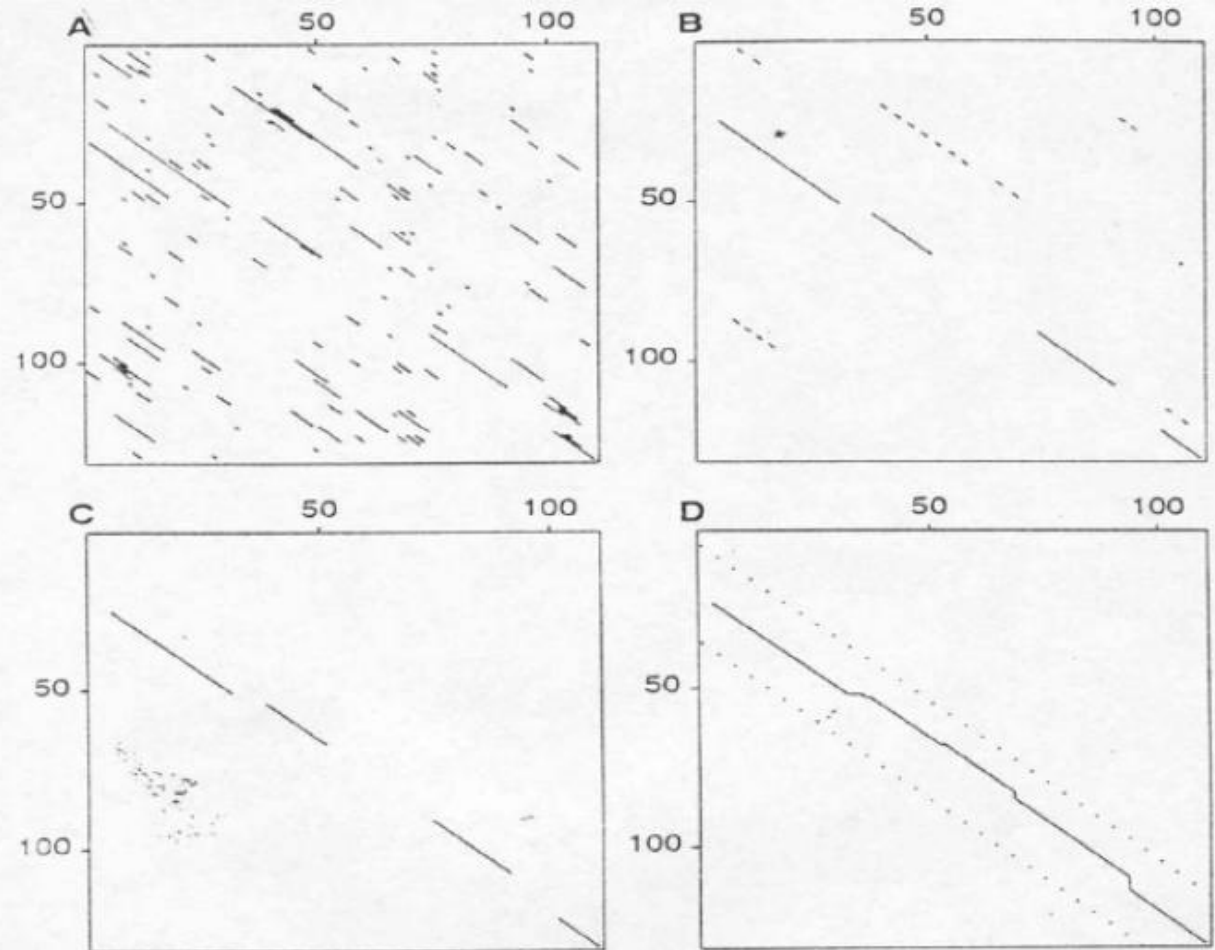


FIG. 1. Identification of sequence similarities by FASTA. The four steps used by the FASTA program to calculate the initial and optimal similarity scores between two sequences are shown. (A) Identify regions of identity. (B) Scan the regions using a scoring matrix and save the best initial regions. Initial regions with scores less than the joining threshold (27) are dashed. The asterisk denotes the highest scoring region reported by FASTP. (C) Optimally join initial regions with scores greater than a threshold. The solid lines denote regions that are joined to make up the optimized initial score. (D) Recalculate an optimized alignment centered around the highest scoring initial region. The dotted lines denote the bounds of the optimized alignment. The result of this alignment is reported as the optimized score.

# FASTA Ranking [9]

Score	Number of sequences
< 2	1 :*
4	0 :
6	1 :*
8	4 :***
10	10 :*****
12	35 :*****
14	75 :*****
16	215 :*****
18	283 :*****
20	390 :*****
22	362 :*****
24	313 :*****
26	307 :*****
28	246 :*****
30	157 :*****
32	87 :*****
34	65 :*****
36	36 :*****
38	24 :*****
40	21 :*****
42	11 :*****
44	8 :*****
46	3 :***
48	1 :*
50	2 :*
52	2 :*
54	0 :
56	0 :
58	0 :
60	0 :
62	2 :*
64	0 :
66	0 :
68	0 :
70	0 :
72	0 :
74	0 :
76	3 :***
78	1 :*
80	1 :*
> 80	10 :*****



# FASTA Conclusions

- **Scores compare well** to optimal alignment algorithms.
- **It remains an heuristic:** it is always possible to construct examples in which FASTA will not find the optimal alignments.
- **Much faster** than ordinary dynamic programming algorithm for sequence alignment.





# BLAST

## Basic Local Alignment Search Tool [5]

BLAST, developed by Altschul, Gish, Miller, Myes, and Lipman in 1990

- Increase speed over FASTA
- Find fewer and better hot-spots
- Therefore, integrate the substitution matrix already at the first stage, when finding hot-spots.
- Originally developed for protein alignments.



# BLAST

Given two sequences  $S_1$  and  $S_2$ .

A **segment pair** is a pair of equal length subsequences of  $S_1$  and  $S_2$  that are aligned without gaps.

A **locally maximal segment** is a segment whose alignment score does not increase when extending or shortening it.

A **maximum segment pair MSP** of  $S_1$  and  $S_2$  is a segment pair with maximum score over all segment pairs in  $S_1$  and  $S_2$ .



# BLAST Algorithm

General idea:

- Given a database with sequences.
- Find all sequences that have an *MSP* with the query string that has a score  $\geq$  *threshold S* (these are so called *high scoring pairs HSP*).



# BLAST Algorithm

## Step 1

- Given are length  $w$ , and threshold  $T$ .
- Find all  $w$ -length substrings (words) of database sequences that align with words from the query string with a score (using a substitution matrix score)  $\geq T$ . If found it is called a *hit*.
- Note: we build for each  $w$ -length word in the query a **data structure** containing all the  $w$ -length words that are similar with a score  $\geq T$ .

## Note that:

- $w$  can be relatively big ( $\Rightarrow$  speed) without losing sensitivity:  $w = 3$  to  $5$  for protein sequences, and  $w \sim 12$  for DNA sequences.
- $w$  in general is kept fixed, while  $T$  is varied for a trade-off between speed and sensitivity.



# BLAST Algorithm

## Step 2

- Extend each *hit* to a *locally maximal segment*.
- The extension will be stopped, if the reduction of the score relative to the maximum value encountered  $\geq$  *given drop-off threshold*.
- If the *MSP* has a score  $\geq S$ , it will be maintained as an *HSP*.

## Note:

- This version of BLAST does not allow *indels*.
- It can be shown that most of the correct alignments are found much more efficiently than using the standard optimal method.



# BLAST Improvement [7]

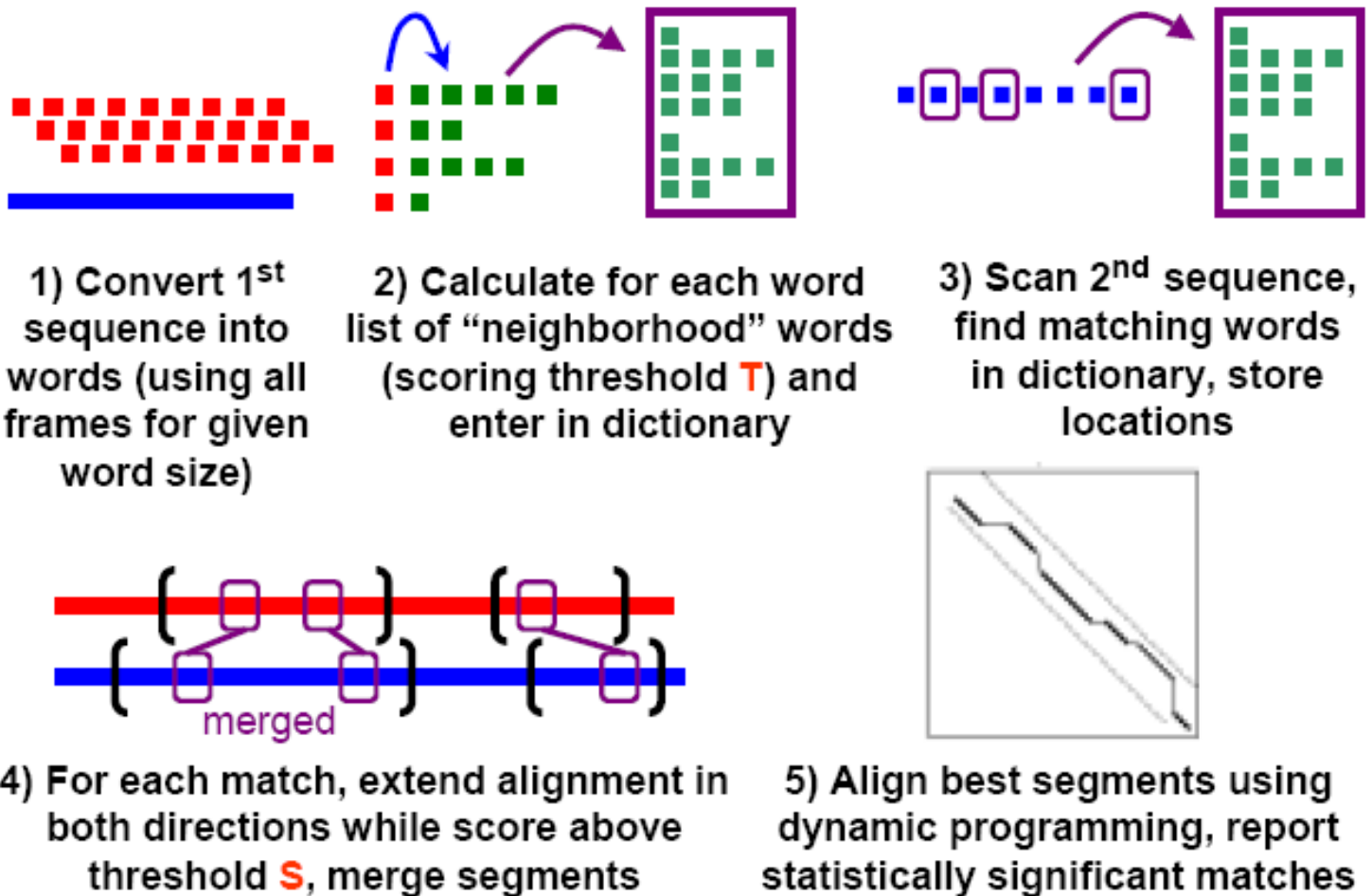
A version that allows *indels*:

1. Again consider the dot-plot matrix: search along the diagonal for *two w-length* words with distance  $\leq A$  and score  $\geq T$ . ( $T$  can be lower here.) Expansion is done only to these pairs of hits.
2. *Two* local alignments from different diagonals are merged into a new local alignment: *alignment-1* followed by *indels* followed by *alignment-2*, as long as the resulting alignment has score  $\geq$  *some given threshold*.

Note:

- This method is about *3* times faster than the original BLAST as only *two-hit words* are expanded, i.e., much less hits have to be processed.

# BLAST (Basic Local Alignment Search Tool)



# BLAST

Fixed word size  $w (=11)$



1) Convert 1<sup>st</sup> sequence into words (using all frames for given word size)

- Both the DNA sequence and its complement are directed.
- The reading takes place in the designated direction.
- Per sequence there are 3 reading frames for reading codons.

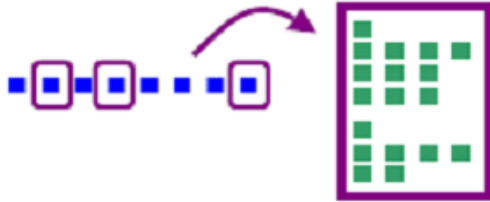


2) Calculate for each word list of "neighborhood" words (scoring threshold  $T$ ) and enter in dictionary

- Fixed alignment score  $T$
- Calculate for each word  $a$  of the first string all the  $length-w$  words  $v$  that are similar to  $a$  with scoring at least  $T$
- Using a keyword tree this can be done in linear time (in length of the initial list of  $length-w$  words)

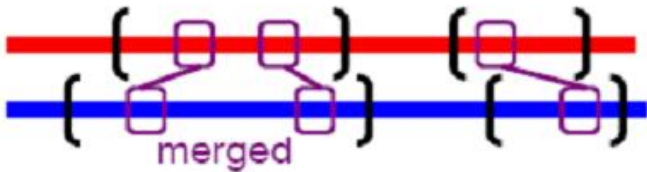


# BLAST



3) Scan 2<sup>nd</sup> sequence, find matching words in dictionary, store locations

- The **length-w** words of the first string and its high scoring similar counterparts are stored in a **dictionary**
- The **dictionary** is used for finding **exact matches** with the **length-w** words from the 2<sup>nd</sup> sequence
- If we find an exact match, we know that the scoring with the original **length-w** word from the 1<sup>st</sup> sequence is always above threshold **T**



4) For each match, extend alignment in both directions while score above threshold **S**, merge segments

- Fixed threshold **S** for scoring extensions
- For each '**exact**' match that we found we extend the alignment in both directions while the score is above threshold **S**
- Close segments are merged

# BLAST



**5) Align best segments using dynamic programming, report statistically significant matches**

- We now have **long high scoring (above threshold  $S$ ) segments**
- The number of different segments is like in the case of FASTA restricted to a diagonal band
- Again **dynamic programming/max weight path** can be used to align the best segments and find the global alignment
- The **scoring function** can be used to derive the **significance of the matches**



# PSI-BLAST [7]

Yet another improved version of **BLAST** is called **Position Specific Iterated- (PSI-) BLAST**.

## Motivation

- A group of aligned amino acid sequences will show per column a so called **profile** of amino acids.
- If the amino acids belong to the same family it is expected that there will be some regions that have **profiles that show little variance**. These are the **conserved regions** that define the structures and functionality specific for the family.
- How can we take into account this kind of statistical information in **BLAST**. (Note, this info will in general not be available in the scoring matrices used.)



# PSI-BLAST sketch

1. Perform **BLAST** while using a different cost vector  $V_i$  for each column  $i$ . Where  $V_i$  initially is equal to the row of the substitution matrix corresponding to the  $i$ -th character in the query sequence. ( $V_i$  is a per query character substitution vector cost function.)
2. Use the high scoring results from step 1 to **build profiles per query character position**.
3. Perform **BLAST** but now **using as query the collection of profiles**, i.e., use a histogram at each column.
4. **Update the profiles** according to the obtained result sequences. (basically, this is the same as updating the position dependent cost vectors  $V_i$ )
5. Go to step 3, while we still find meaningful new matches, otherwise stop.

**Note:** **PSI-BLAST** is able to find more distantly related sequences than **FASTA** and **BLAST**, but may also find too distant sequences.



# Amino Acids Substitution Matrices

- Used **scorings** have a very important impact on the results.
- Scorings should reflect **biological meaningfulness, molecular phenomena, etc.**
- Empirical observations on **ancestral sequences and their descendants** should be used to derive meaningful scoring functions.



# Amino Acids Substitution Matrices

No biological phenomena are employed by:

- 1) The *unit matrix*  $M$ , where
  - $M_{ij} = 1$  if  $i = j$ , and
  - $M_{ij} = 0$  otherwise.
- 2) The *genetic code matrix*  $M$ , where  $M_{ij}$  equals the minimal base substitutions that are necessary to convert the codon of amino acid  $i$  into the codon of amino acid  $j$ .

Note:

- 1) measures similarity
- 2) measures distance.



# PAM

Margaret Dayhoff et al. 1979 [6, 11]

## Data set

71 super-families of protein sequences with  
1572 accepted mutations

## Observations

- Substitutions within protein families are not random
- Some substitutions occurred more frequent because apparently there was no major effect on structure and function
- As a consequence evolutionary related proteins do not need to have the same amino acids at the same position, comparable ones suffice.



# PAM

Point Accepted Mutations

Percent Accepted Mutations

## Definition

Sequences  $S_1$  and  $S_2$  are at an evolutionary distance of  $1$  PAM, if  $S_1$  evolved to  $S_2$  with an average of  $1$  accepted point-mutation per  $100$  amino acids.

## Note

- Accepted means the mutation was incorporated into the protein and was passed to its progeny.
- If  $d_{PAM}(S_1, S_2) = 1$ , it does not follow that  $S_1$  and  $S_2$  differ  $1$  percent. There can be more than  $1$  accepted point-mutation on the same position.
- Additive:  $d_{PAM}(S_1, S_2) + d_{PAM}(S_2, S_3) = d_{PAM}(S_1, S_3)$ .





# PAM units

## Problems

- Ancestor-descendant relations between proteins are unknown in general.
  - It is assumed that amino acids mutations are reversible and the probability of either direction is the same.
  - If  $S_i$  and  $S_j$  have a mutual ancestor  $S_{ij}$ , then
$$d_{PAM}(S_i, S_j) = d_{PAM}(S_i, S_{ij}) + d_{PAM}(S_{ij}, S_j).$$
- *Indels* are ignored  $\Rightarrow$  matching position are uncertain.
  - For sequences that are evolutionary distant this is a major problem.
  - Therefore consider *conserved regions* of protein families.



# PAM matrices

**PAM** matrices are amino acid substitution matrices encoding the expected evolutionary change.

For any pair  $(A_i, A_j)$  the  $(i, j)$  entry in the **PAM- $N$**  matrix is a score equal to the **log** of the **probability** that  $A_i$  replaces  $A_j$  in two sequences that are  **$N$  PAM** units apart.

**Question:** How do we gather the valid statistics to determine for example a **PAM 120** matrix?

# PAM matrix generation

- *Dayhoff* used aligned highly similar sequences with known evolutionary trees.
- **PAM 1** computation
  - $M_{ij}$  is equal to the observed relative frequency of  $A_i \rightarrow A_j$  during **1 PAM**
  - $M_{ij}$  is a **20x20** matrix where each column adds up to **1**.

	<i>A</i>	<i>R</i>	<i>N</i>	<i>D</i>	<i>C</i>
<i>A</i>	9867	2	9	10	3
<i>R</i>	1	9913	1	0	1
<i>N</i>	4	1	9822	36	0
<i>D</i>	6	0	42	9859	0
<i>C</i>	1	1	0	0	9973



# PAM-N matrix computation

- If we know  $M$ , then  $M^N$  gives the probabilities of any amino acid mutating to any other during  $N$  PAM units  $\Rightarrow$  PAM- $N$  matrix.
- From PAM- $N$  the matrix  $(c_{ij})$  is derived:

$$c_{ij} = \log \frac{f(j)M^n(i, j)}{f(i)f(j)} = \log \frac{M^n(i, j)}{f(i)}$$

where  $f(i)$  and  $f(j)$  are the observed frequencies of amino acids  $A_i$  and  $A_j$ , respectively.



# BLOSUM

## Block Substitution Matrix

An amino acid substitution matrix by *Henikoff and Henikoff* in 1992 [8].

- **PAM** derived from globally aligned proteins
- **BLOSUM** derived from alignments of **short blocks** of sequences with a defined level of similarity.

# BLOSUM matrix calculation

- **Blocks** of over 500 groups of related family members are used.
- **Blocks** act as signatures of the family.

## Definition

A *block* is a conserved regions of a protein family with the regions of the family members **aligned**.

```
A ABCDA . . . BBCDA
D ABCDA . A . BBCBB
B BBCDABA . BCCAA
A AACDAC . DCBCDB
C CBADAB . DBBDCC
A AACAA . . . BBCCC
```



# BLOSUM matrix calculation

The **BLOSUM** matrix calculation stages:

1. Eliminate sequences that are identical more than  $x\%$  (**this is to avoid bias**).
  - Either remove the sequence, or replace the sequences by  $1$  representative.
  - Matrix built from **blocks** with no more than  $x\%$  similarity is called **BLOSUM X**.
2. Calculate **substitution statistics** by counting each pair in each column of the blocks of multiple alignment.
  - For example: column **AABACA**  $\Rightarrow$  **6 AA-pairs, 4 AB-pairs, 4 AC-pairs, 1 BC pair**.
3. Normalize the results.



# BLOSUM matrix calculation

## 3) Normalization

- Let  $q_{ij}$  be the observed probability that a pair of amino acids in the same column are  $A_i$  and  $A_j$ .
- Then  $p_i$  is the observed probability of a certain amino acid to be  $A_i$ :

$$p_i = q_{ii} + \sum_{i \neq j} \frac{q_{ij}}{2}$$

- Assume that  $p_i$  and  $p_j$  are independent, then actually a given pair should occur with relative frequencies  $e_{ij} = p_i p_j$ , if  $i \neq j$ .
- Now the odds matrix is given by  $(q_{ij}/e_{ij})$
- The log odd ratio  $s_{ij} = \log_2 (q_{ij}/p_{ij})$
- Finally, BLOSUM matrix **B** has as  $(i,j)$  entry = ( rounded  $(s_{ij})$  )

## Note:

- $S_{ij}=0$ , if observed = expected # differences between amino acid pairs
- $S_{ij}>0$ , if observed > expected # differences between amino acid pairs
- $S_{ij}<0$ , if observed < expected # differences between amino acid pairs





# BLOSUM vs PAM

**BLOSUM** uses more sequences to compute the necessary statistics than **PAM** ...

Some tests showed:

- **BLOSUM 45, 62, and 80** outperform **PAM 120, 160, or 250**.
- **BLOSUM** missed **6-9** positions while **PAM** missed **30-31** positions.
- **BLOSUM62** is found better than **BLOSUM-X (X≠62)** and **PAM-X**.



# BLOSUM62

The BLOSUM62 scoring matrix:

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



# References

- [1] R. Shamir, [Algorithms in Molecular Biology \(2009 version\)](#), 2001.
- [2] R. Shamir, Sequence Alignment Heuristics, Scribe from Lecture, 2001.
- [3] H.J. Hoogeboom, A.P. Gultyaev, [Lecture Notes](#), under development, 2009.
- [4] H.J. Hoogeboom, [Lecture Slides](#), 2009.



# References

- [5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215:403–10, 1990.
- [6] M. Dayhoff and R. Schwartz. Matrices for detecting distant relationship. *Atlas of Protein Sequences*, pages 353–358, 1979.
- [7] S. F. Altschul et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25(17):3389–402, 1997.
- [8] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Science USA*, 89(22):10915–10919, November 1992.
- [9] D. Lipman and W. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.
- [10] D. Lipman and W. Pearson. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Science USA*, 85:2444–2448, 1988.
- [11] R. Schwartz M. Dayhoff and B. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.