Sequence Alignment Algorithms

DEKM book

Notes from Dr. Bino John

and Dr. Takis Benos

To Do

- Global alignment
- Local alignment
- Gaps
  - Affine Gaps
  - Algorithm (blackboard)
- Statistical Significance
  - Notes (blackboard)
- Read up on database searches
  - BLAST
  - FASTA
  - CS tricks: suffix tree, …
- PSSMs and Multiple Sequence Alignments

Why compare sequences?

- Given a new sequence, infer its function based on similarity to another sequence

- Find important molecular regions – conserved across species

Why compare sequences? Do more..

- Determine the evolutionary constraints at work
- Find mutations in a population or family of genes
- Find similar looking sequence in a database
- Find secondary/tertiary structure of a sequence of interest – molecular modeling using a template (homology modeling)

Sequence alignment

- Are two sequences related?
  - Align sequences or parts of them
  - Decide if alignment is by chance or evolutionarily linked?
- Issues:
  - What sorts of alignments to consider?
  - How to score an alignment and hence rank?
  - Algorithm to find good alignments
  - Evaluate the significance of the alignment

How do we use the matrices for sequence alignment?

AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC

-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

Definition

Given two strings $x = x_1x_2...x_M$, $y = y_1y_2...y_N$,

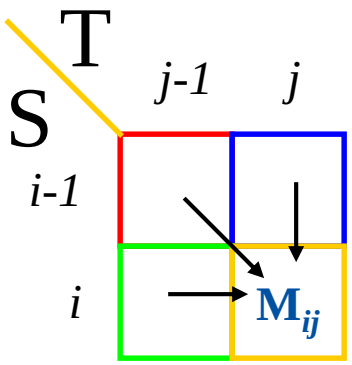
An **alignment** of two sequences *x* and *y* is an arrangement of *x* and *y* by position, where *a* and *b* can be padded with gap symbols to achieve the same length.

Dynamic Programming

We apply dynamic programming when:

- There is only a polynomial number of subproblems
  - Align $x_1 \ldots x_i$ to $y_1 \ldots y_j$
- Original problem is one of the subproblems
  - Align $x_1 \ldots x_M$ to $y_1 \ldots y_N$
- Each subproblem is easily solved from smaller subproblems

Global alignment



PAM

DNA matrix

BLOSUM

$$M_{i,j} = MAX \begin{cases} M_{i-1,\,j-1} + Score(S_i, T_j) \\ M_{i,j-1} + \gamma \\ M_{i-1,j} + \gamma \end{cases}$$

Gap penalty

Needleman & Wunsch, 1970

Dynamic programming for global alignment – simple case

We want the best alignment between two sequences x and y
Consider two sequences: $x_1 .. x_M$, and $y_1 .. y_M$
we have ONLY three choices to get the best score *F(i,j)*

1. x_i aligns to y_j

   $x_1 \ldots\ldots x_{i-1}$ x_i

   $y_1 \ldots\ldots y_{j-1}$ y_j

   Align x[1 .. i] with y[1 .. j]

2. x_i aligns to a gap

   $x_1 \ldots\ldots x_{i-1}$ x_i

   $y_1 \ldots\ldots y_j$ -

   x[1 ..(i-1)] is already aligned with y[1 ..(j)], so align x[i] with a gap in y

3. y_j aligns to a gap

   $x_1 \ldots\ldots x_i$ -

   $y_1 \ldots\ldots y_{j-1}$ y_j

   x[1 .. i] is already aligned with y[1 .. (j-1)], so align a gap in x to y[j]

1. x_i aligns to y_j

   $x_1 \ldots\ldots x_{i-1}\ \ x_i$

   $y_1 \ldots\ldots y_{j-1}\ \ y_j$

   $F(i,j) = F(i-1, j-1) + s(i,j)$

2. x_i aligns to a gap

   $x_1 \ldots\ldots x_{i-1}\ \ x_i$

   $y_1 \ldots\ldots y_j\ \ -$

   $F(i,j) = F(i-1, j) - gap\_open\_penalty\ (go)$

3. y_j aligns to a gap

   $x_1 \ldots\ldots x_i\ \ -$

   $y_1 \ldots\ldots y_{j-1}\ \ y_j$

   $F(i,j) = F(i, j-1) - gap\_open\_penalty\ (go)$

If we could make $F(i, j-1)$, $F(i-1, j)$, $F(i-1, j-1)$ optimal, then we can make the next ones optimal as well

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

Where $s(x_i, y_j) =$ *Score for a match*, if $x_i = y_j$;
score for a mismatch, if $x_i \neq y_j$;

The Needleman-Wunsch Algorithm

– pioneering application of DP to biological sequences

1. **Initialization.**
   - a. $F(0, 0) = 0$
   - b. $F(0, j) = -j \times go$
   - c. $F(i, 0) = -i \times go$

   O(NM)

2. **Main Iteration.** Filling-in partial alignments

   For each i = 1……M

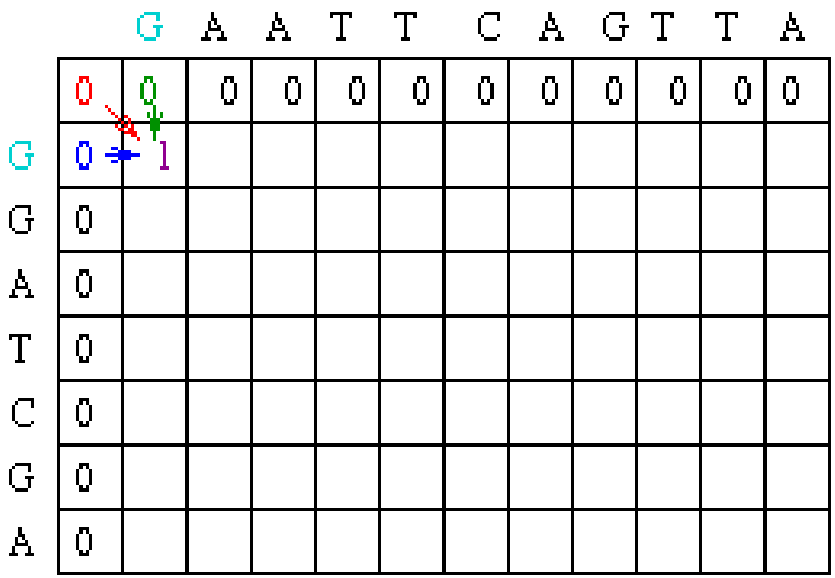
   For each j = 1……N

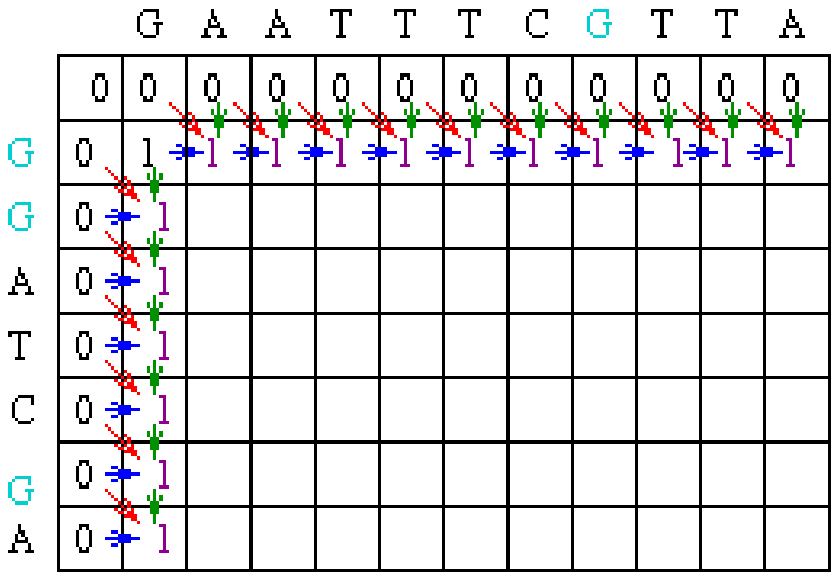
$$F(i, j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) & \text{[case 1]} \\ F(i-1, j) - go & \text{[case 2]} \\ F(i, j-1) - go & \text{[case 3]} \end{cases}$$

$$Ptr(i,j) = \begin{cases} DIAG, & \text{if [case 1]} \\ UP, & \text{if [case 2]} \\ LEFT, & \text{if [case 3]} \end{cases}$$

3. **Termination.** F(M, N) is the optimal score, and from Ptr(M, N) can trace back optimal alignment

Alignment: adding scores (cntd)





Score(match) = 1
Score(mismatch) = 0
Score(gap) = 0

Alignment: adding scores

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

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	2	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	3	4	4	4	4	4
G	0	1	2	2	3	3	3	4	4	5	5	5
A	0	1	2	3	3	3	3	4	5	5	5	**6**

Alignment: adding scores (cntd)

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	1	1	1	1	1	1	1	1	1	1
G	0	1	1	1	1	1	1	1	2	2	2	2
A	0	1	1	2	2	2	2	2	2	2	2	3
T	0	1	2	2	3	3	3	3	3	3	3	3
C	0	1	2	2	3	3	4	4	4	4	4	4
G	0	1	2	2	3	3	4	4	5	5	5	5
A	0	1	2	3	3	3	4	5	5	5	5	6

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	
G	0	1	1	1	1	1	1	1	1	1	1	
G	0	1	1	1	1	1	1	1	2	2	2	
A	0	1	2	2	2	2	2	2	2	2	2	
T	0	1	2	2	3	3	3	3	3	3	3	
C	0	1	2	2	3	3	4	4	4	4	4	
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

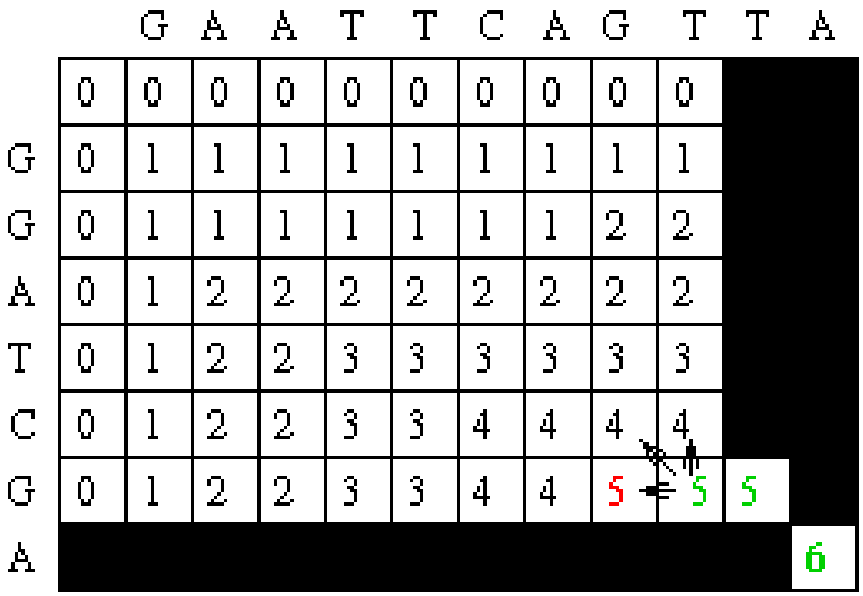
Alignment:

(Seq #1) A

|

(Seq #2) A

Alignment: adding scores (cntd)

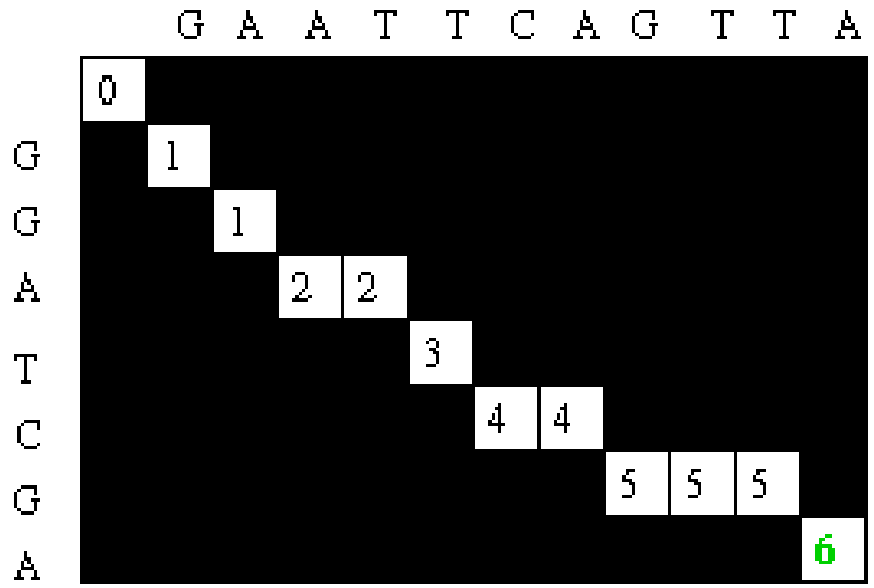


		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0		
G	0	1	1	1	1	1	1	1	1	1		
G	0	1	1	1	1	1	1	1	2	2		
A	0	1	2	2	2	2	2	2	2	2		
T	0	1	2	2	3	3	3	3	3	3		
C	0	1	2	2	3	3	4	4	4	4		
G	0	1	2	2	3	3	4	4	5	5	5	
A												6

```
Alignment:   (Seq #1)    T A
                           |
             (Seq #2)    - A
```

Alignment: adding scores (cntd)

		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0		
G	0	1	1	1	1	1	1	1	1	1		
G	0	1	1	1	1	1	1	1	2	2		
A	0	1	2	2	2	2	2	2	2	2		
T	0	1	2	2	3	3	3	3	3	3		
C	0	1	2	2	3	3	4	4	4	4		
G	0	1	2	2	3	3	4	4	5	5	5	
A											6	

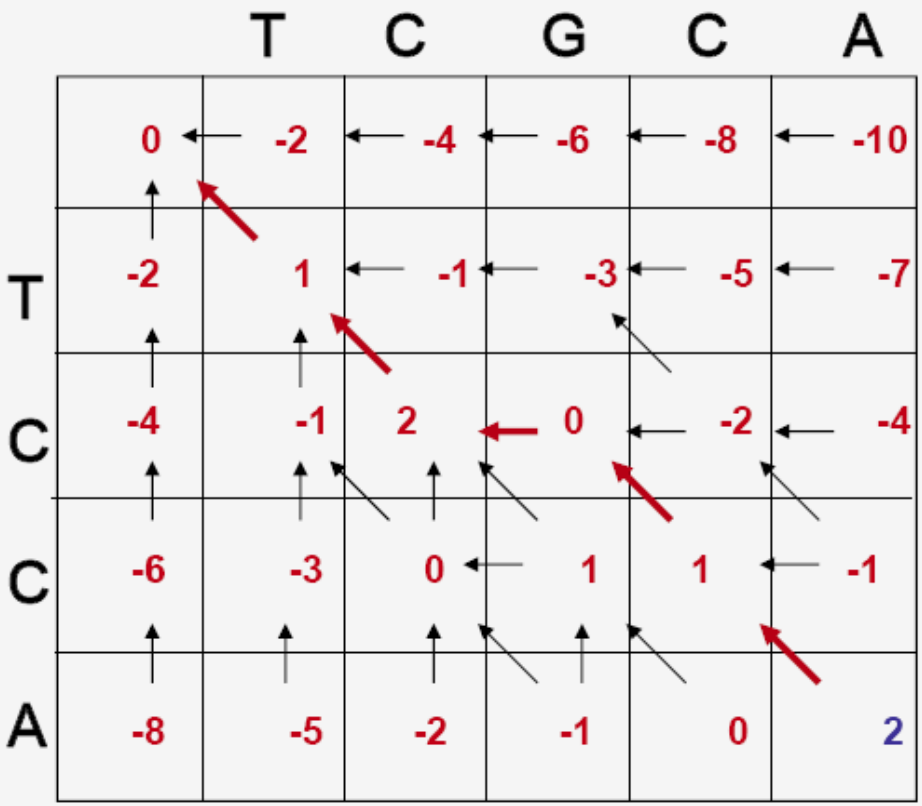


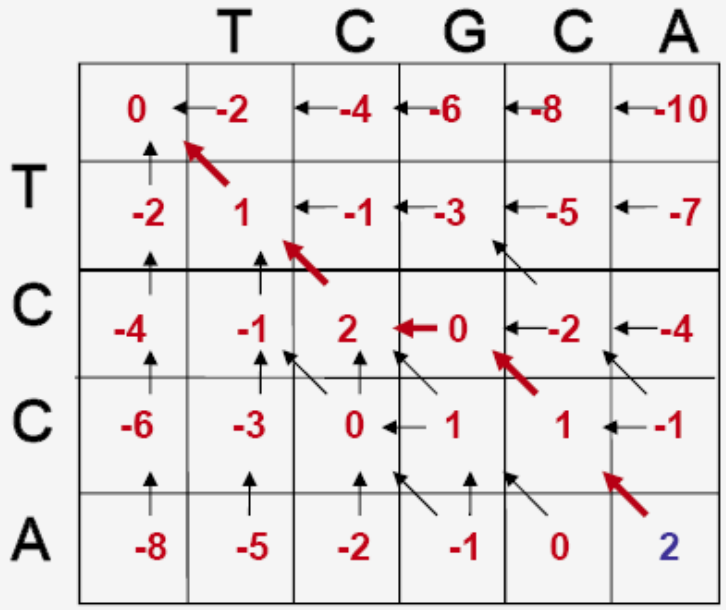

```
(Seq #1) G A A T T C A G T T A
         |   |   | |   |     |
Alignment:
(Seq #2) G G A - T C - G - - A
```

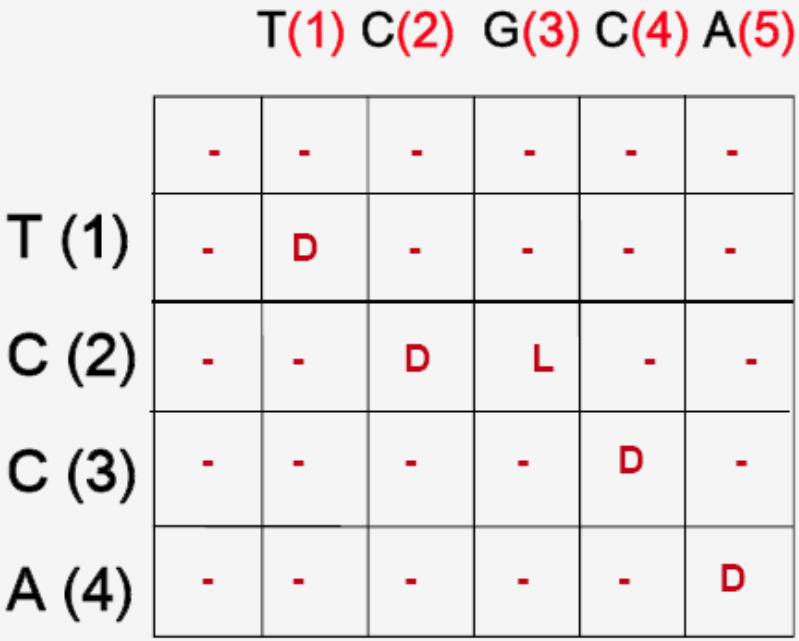
6 matches, 1 mism., 4 gaps

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

$s(x_i, y_j)$=1 for match,
-1 for mismatch
go=2







$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, y_j) \text{ [case 1]} \\ F(i-1,j) - \text{go [case 2]} \\ F(i,j-1) - \text{go [case 3]} \end{cases}$$

$$Ptr(i,j) = \begin{cases} \text{DIAG OR D, if [case 1]} \\ \text{UP OR U, if [case 2]} \\ \text{LEFT OR L,if [case 3]} \end{cases}$$

D at 1,1 =>1,1 is paired & decrease i and j by 1 => T,T & go to 0,0 - END

D at 2,2 =>2,2 is paired & decrease i and j by 1 => C,C & go to 1,1

L at 2,3 =>-,3 is paired & decrease j by 1 => -,G & go to 2,2

D at 3,4 =>3,4 is paired and decrease i and j by 1 => CC & go to 2,3

D at 4,5=>4,5 are paired and decrease i and j by 1 => AA & go to 3,4

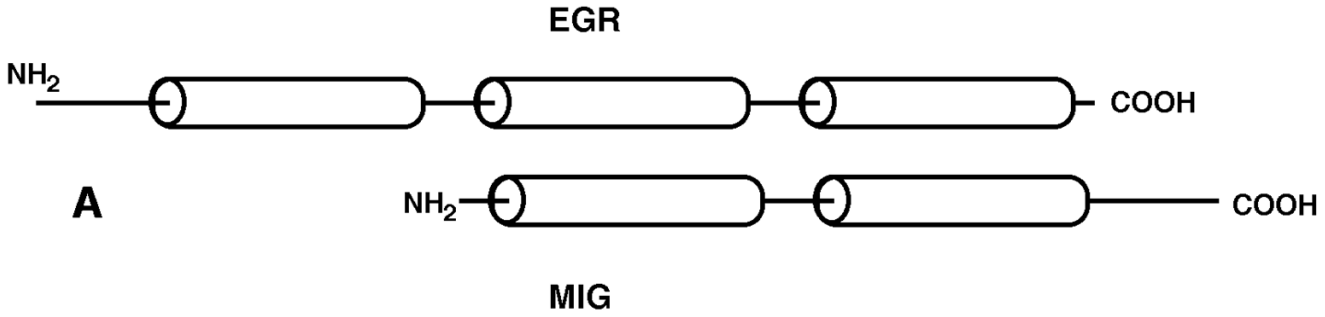
```
T C - C A
T C G C A
```

Local alignment

Given two sequences, *S* and *T*, find two subsequences, *s* and *t*, whose alignment has the highest "score" amongst all subsequence pairs.

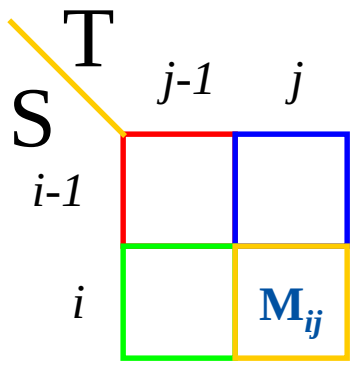
Question: Why do we need local alignment, if we have the global one?

Local alignment: an example




```
EGR4_HUMAN   KA [FACPVESCVRSFARSDELNRHLRIH] TGHKP [FQCRICLRNFSRSDHLTSHVRTH] TGEKP [FACDV--CGRRFARSDEKKRHSKVH]
EGR4_RAT     KA [FACPVESCVRTFARSDELNRHLRIH] TGHKP [FQCRICLRNFSRSDHLTTHVRTH] TGEKP [FACDV--CGRRFARSDEKKRHSKVH]
EGR3_HUMAN   RP [HACPAEGCDRRFSRSDELTRHLRIH] TGHKP [FQCRICMRSFSRSDHLTTHIRTH] TGEKP [FACEF--CGRKFARSDERKRHAKIH]
EGR3_RAT     RP [HACPAEGCDRRFSRSDELTRHLRIH] TGHKP [FQCRICMRSFSRSDHLTTHIRTH] TGEKP [FACEF--CGRKFARSDERKRHAKIH]
EGR1_HUMAN   RP [YACPVESCDRRFSRSDELTRHIRIH] TGQKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDI--CGRKFARSDERKRHTKIH]
EGR1_MOUSE   RP [YACPVESCDRRFSRSDELTRHIRIH] TGQKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDI--CGRKFARSDERKRHTKIH]
EGR1_RAT     RP [YACPVESCDRRFSRSDELTRHIRIH] TGQKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDI--CGRKFARSDERKRHTKIH]
EGR1_BRARE   RP [YACPVETCDRRFSRSDELTRHIRIH] TGQKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACEI--CGRKFARSDERKRHTKIH]
EGR2_RAT     RP [YPCPAEGCDRRFSRSDELTRHIRIH] TGHKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDY--CGRKFARSDERKRHTKIH]
EGR2_XENLA   RP [YPCPAEGCDRRFSRSDELTRHIRIH] TGHKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDY--CGRKFARSDERKRHTKIH]
EGR2_MOUSE   RP [YPCPAEGCDRRFSRSDELTRHIRIH] TGHKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDY--CGRKFARSDERKRHTKIH]
EGR2_HUMAN   RP [YPCPAEGCDRRFSRSDELTRHIRIH] TGHKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDY--CGRKFARSDERKRHTKIH]
EGR2_BRARE   RP [YPCPAEGCDRRFSRSDELTRHIRIH] TGHKP [FQCRICMRNFSRSDHLTTHIRTH] TGEKP [FACDF--CGRKFARSDERKRHTKIH]
MIG1_KLULA   -- [-------------------------] ---RP [YVCPICQRGFHRLEHQTRHIRTH] TGERP [HACDFPGCSKRFSRSDELTRHRRIH]
MIG1_KLUMA   -- [-------------------------] ---RP [YMCPICHRGFHRLEHQTRHIRTH] TGERP [HACDFPGCAKRFSRSDELTRHRRIH]
MIG1_YEAST   -- [-------------------------] ---RP [HACPICHRAFHRLEHQTRHMRIH] TGEKP [HACDFPGCVKRFSRSDELTRHRRIH]
MIG2_YEAST   -- [-------------------------] ---RP [FRCDTCHRGFHRLEHKKRHLRTH] TGEKP [HHCAFPGCGKSFSRSDELKRHMRTH]
                [                         ]    :* [. *  * * * * :* . *:* *] ***:* [. *    * : *:**** .** : *]
```

Local alignment (cntd)



$$M_{i,j} = MAX \begin{cases} 0 \\ M_{i-1,\,j-1} + Score(S_i, T_j) \\ M_{i,j-1} + \gamma \\ M_{i-1,j} + \gamma \end{cases}$$

PAM

DNA matrix

BLOSUM

Gap penalty

Smith & Waterman, 1981

Similarity Scoring Expected value:
negative for random alignments
positive for highly similar sequences

The Smith-Waterman Algorithm

1. **Initialization**

   $F(0,0) = F(0,j) = F(i,0) = 0$

2. **Iteration**

   for i=1,…,M

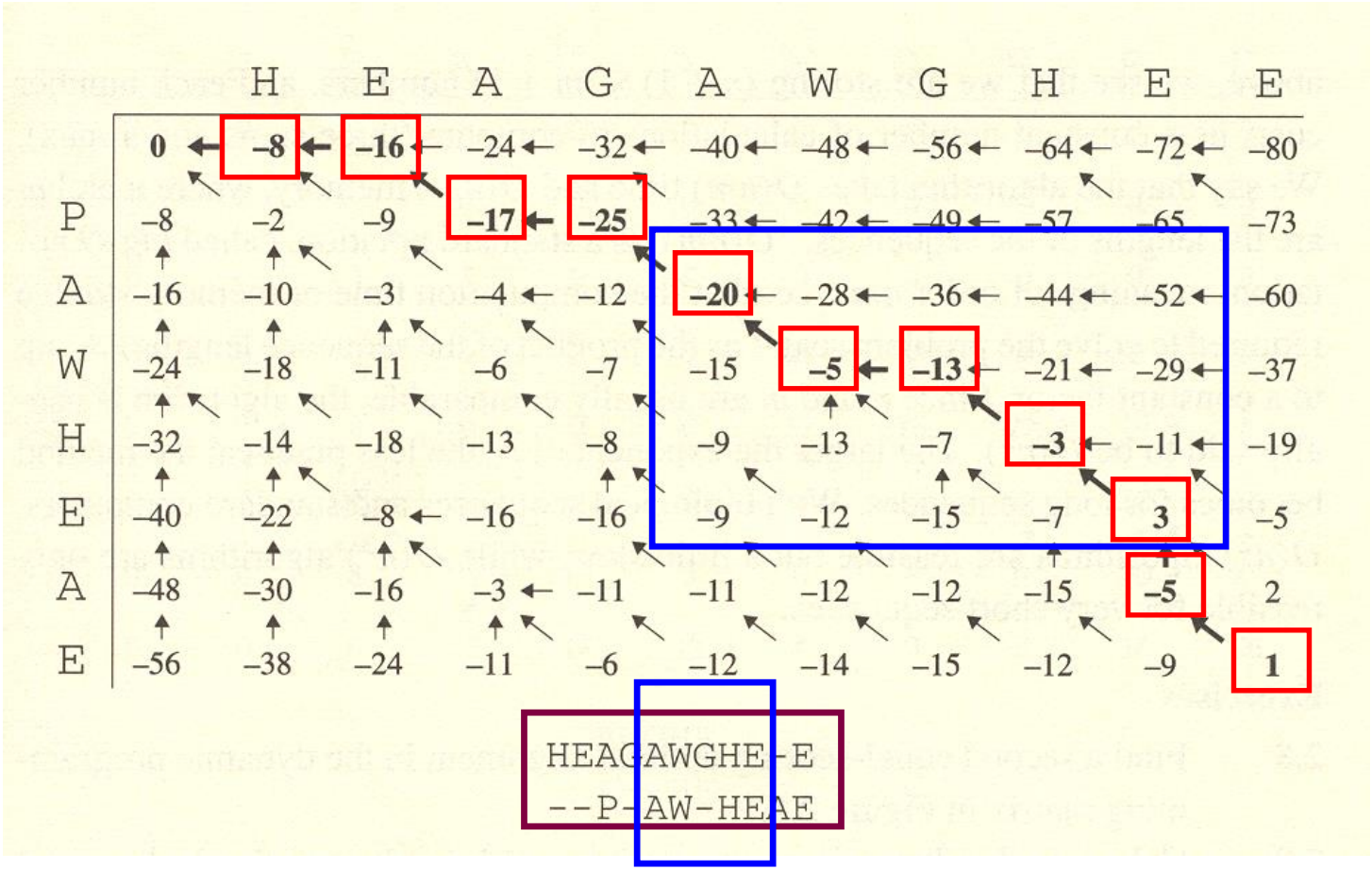
   for j=1,…,N

   - calculate optimal $F(i,j)$
   - store $Ptr(i,j)$

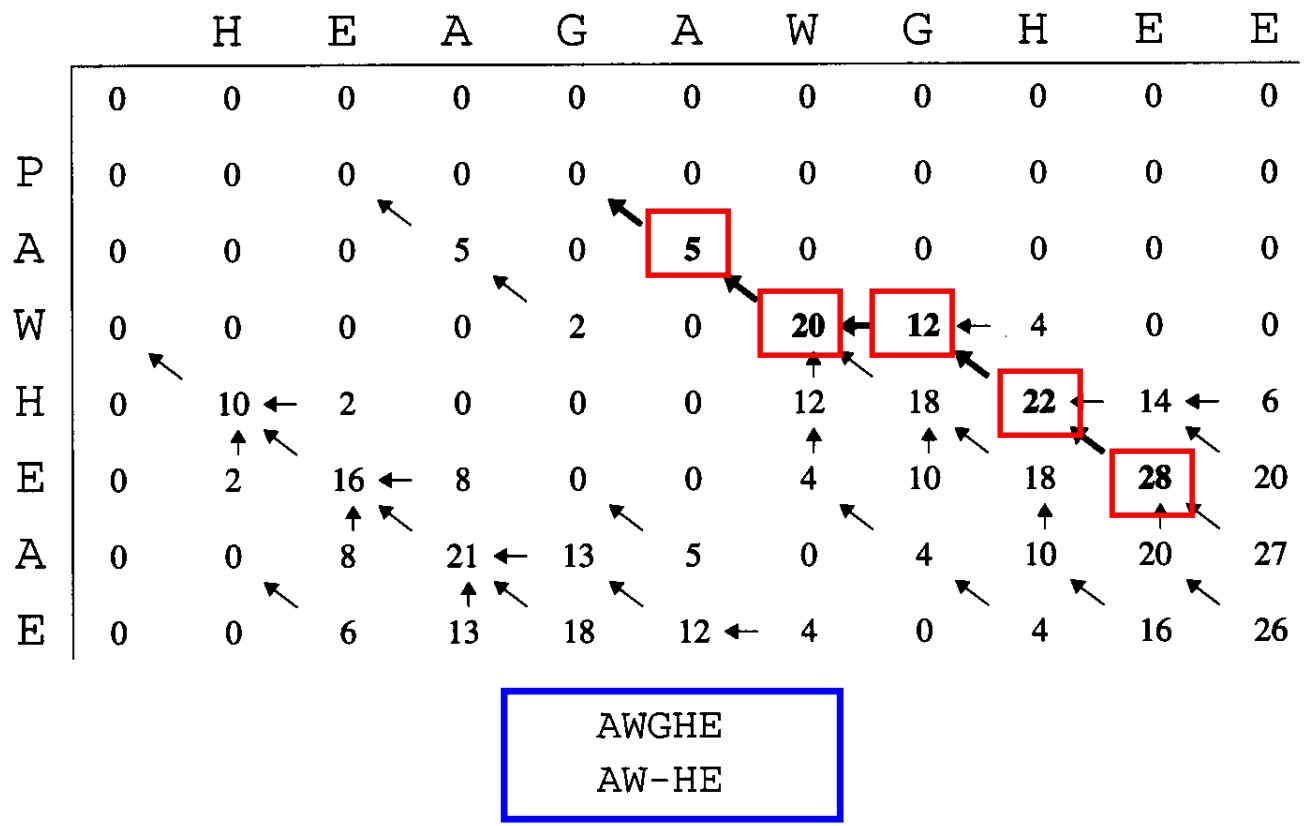
3. **Termination**

   - Find the end of the best alignment with $F_{OPT} = \max_{\{i,j\}} F(i,j)$ and trace back **OR**
   - Find all alignments with $F(i,j) >$ threshold and trace back

Local *vs.* global alignment



Local *vs.* global alignment (cntd)



Local alignment (cntd)

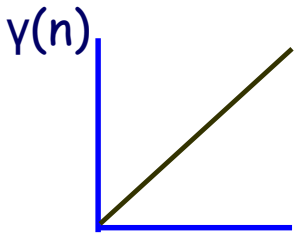
Characteristics of local alignments:

- The alignment can start/end at any point in the matrix.
- No negative scores in the alignment.
- The mean value of the scoring matrix (e.g. PAM, BLOSUM) should be negative, but there should be positive scores in the scoring matrix.

Scoring the gaps more accurately

- **A naive model**

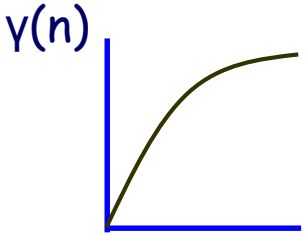
  Gap penalty is linear to the gap length



Nature "prefers" to place gaps where other gaps exist

- **Convex gap penalty function**

  $\gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1)$



Time $O(N^2M)$ Space $O(NM)$

(assume N>M)

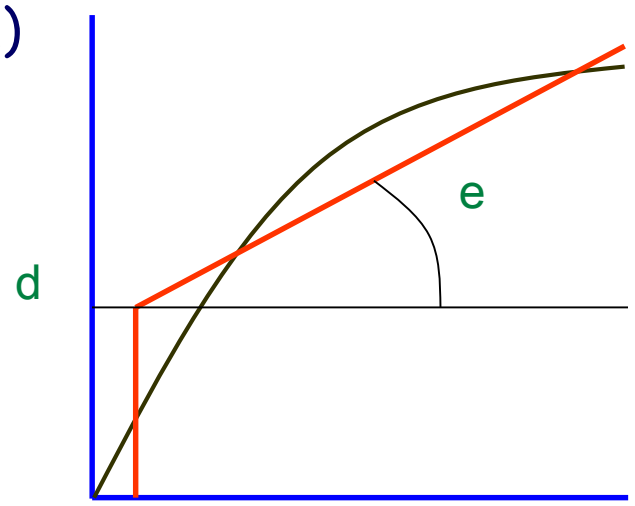
Scoring gaps: affine gaps

- **Affine gaps: a compromise between linear and convex gap penalties**

γ(n) = -d - e * (n-1)

d: gap initiation penalty

e: gap extension penalty



Convex gap dynamic programming

Initialization: same as before

Iteration:

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

Termination: same

Running Time: $O(N^2M)$ (assume N>M)

Space: $O(NM)$

Smith, waterman, and Beyer Algorithm for affine gap

1. **Initialization.**
   - a. $F(0, 0) = 0$
   - b. $F(0, j) = -j \times go$
   - c. $F(i, 0) = -i \times go$

   $O(N^3)$

2. **Main Iteration.** Filling-in partial alignments

   For each i = 1……M

   For each j = 1……N

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

Fast implementation of affine gap penalty

We need three matrices for tracking scores

Matrix-1: a[i,j]= to store maximum score of an alignment that

ends in x[i] matched to y[j]

$\ldots x_i$
$\ldots y_j$

Matrix-2: b[i,j]= to store maximum score of an alignment that

ends in gap matched to y[j]

$\ldots -$
$\ldots y_j$

Matrix-3: c[i,j]= to store maximum score of an alignment that

ends in gap matched to x[i]

$\ldots . x_i$
$\ldots -$

Implementation – Cont'd

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

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

Pointer-matrices: Three matrices to figure out which state within each score matrix maximization was used to obtain the optimal alignment of position i,j. Of course you need to know which matrix yielded the best score in the end (m,n) as well

The twist with affine gap penalty

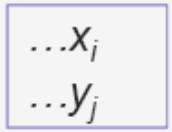
$$score\begin{pmatrix} WA-- \\ -AGC \end{pmatrix} \quad vs \quad score\begin{pmatrix} WA- \\ -AG \end{pmatrix} + score\begin{pmatrix} - \\ C \end{pmatrix}$$

$$g(1)+s(A,A)+g(2) \quad \neq \quad g(1)+s(A,A)+g(1)+g(1)$$

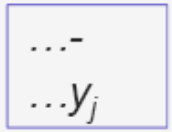
Fast implementation of affine gap penalty

We need three matrices for tracking scores

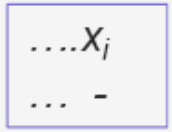
Matrix-1: a[i,j]= to store maximum score of an alignment that ends in x[i] matched to y[j]



Matrix-2: b[i,j]= to store maximum score of an alignment that ends in gap matched to y[j]



Matrix-3: c[i,j]= to store maximum score of an alignment that ends in gap matched to x[i]



Implementation – Cont'd

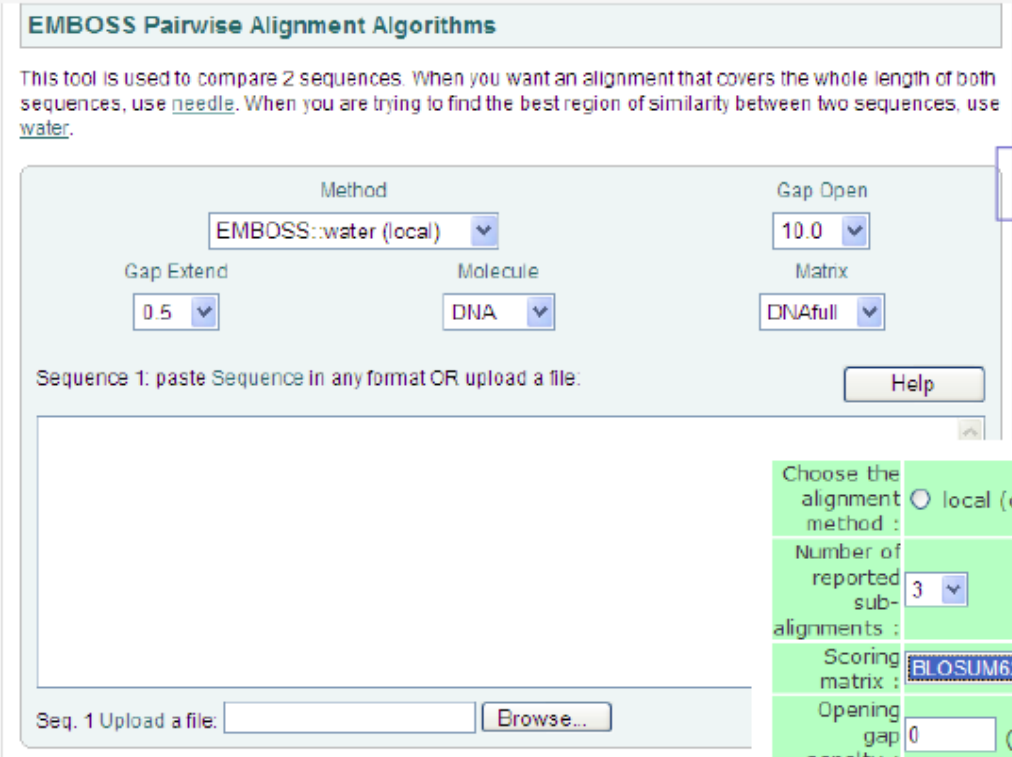
$$a[i,j] = \max \begin{cases} a[i-1,j-1] \\ b[i-1,j-1] + s(i,j) \\ c[i-1,j-1] \end{cases}$$

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

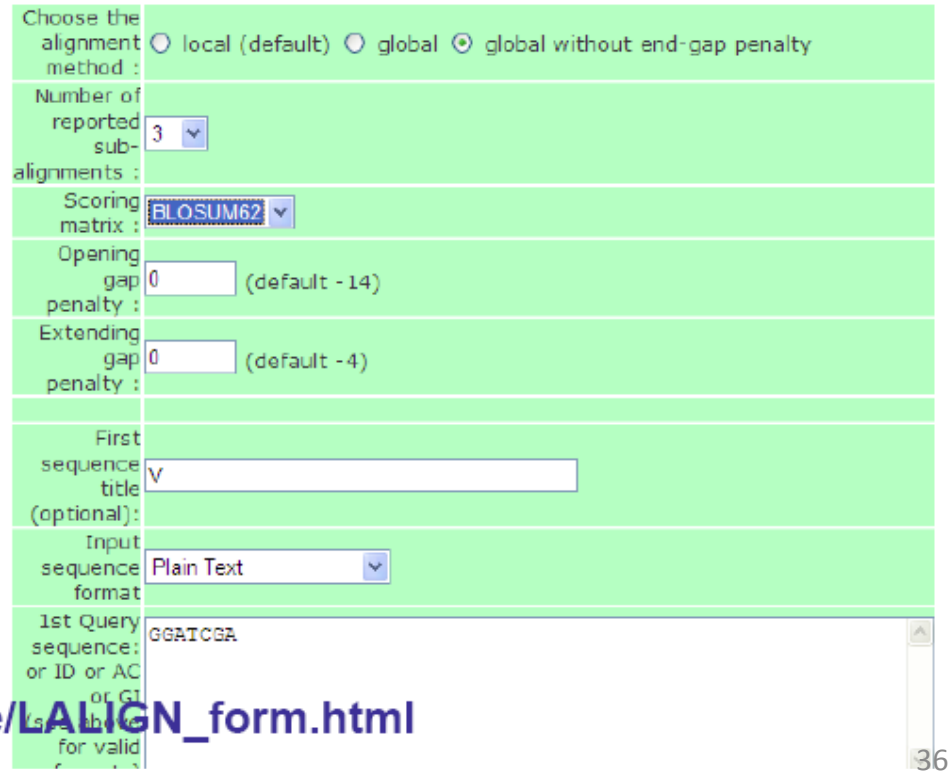
$$c[i,j] = \max \begin{cases} a[i-1,j] + go \\ b[i-1,j] + go \\ c[i-1,j] + ge \end{cases}$$

Pointer-matrices: Three matrices to figure out which state within each score matrix maximization was used to obtain the optimal alignment of position i,j. Of course you need to know which matrix yielded the best score in the end (m,n) as well

http://www.ebi.ac.uk/Tools/emboss/align/index.html



For checking results of your code



http://www.ch.embnet.org/software/LALIGN_form.html

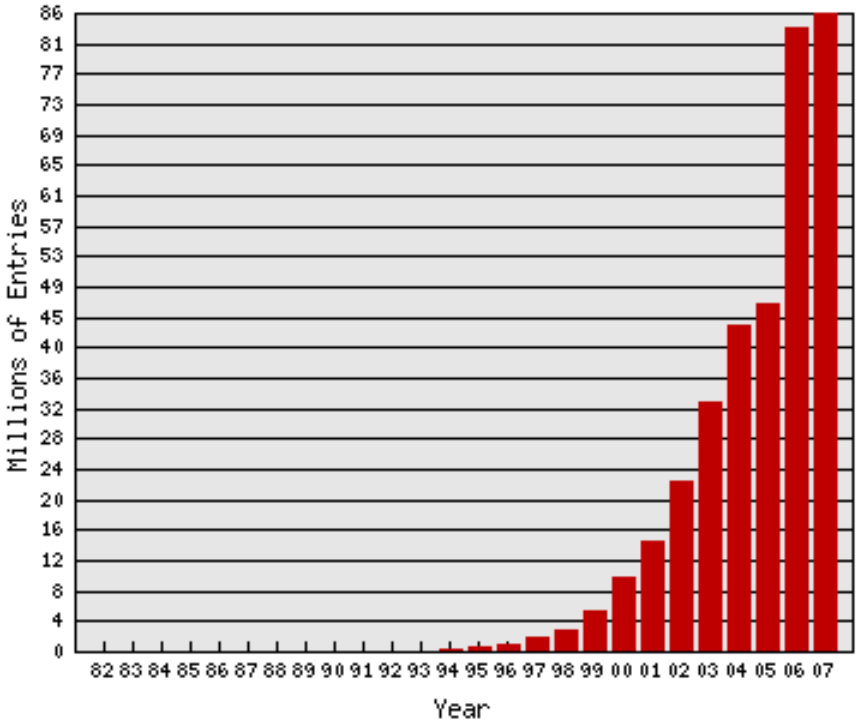
Alignment Programs

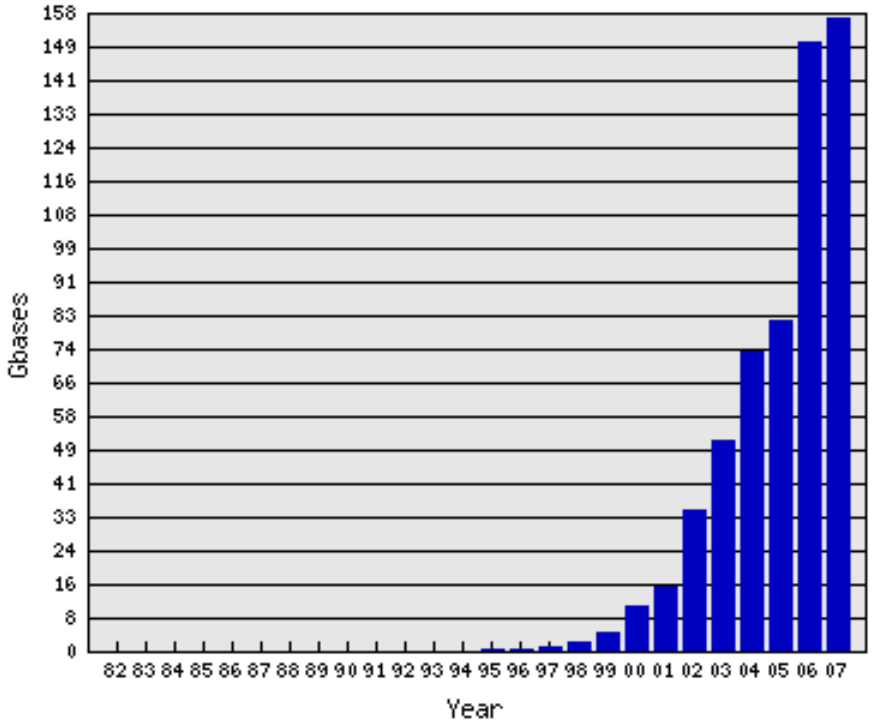
- The Fasta program **Align** –Global.
- EMBOSS **Needle/ Stretcher**–Global
- Fasta program LALIGN – Local
- BLAST 2 Sequences @ NCBI – Local
  - **http://www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html**
- AVID –Genome Scale alignment (LONG Seqs)

Database searches

DNA and protein databases

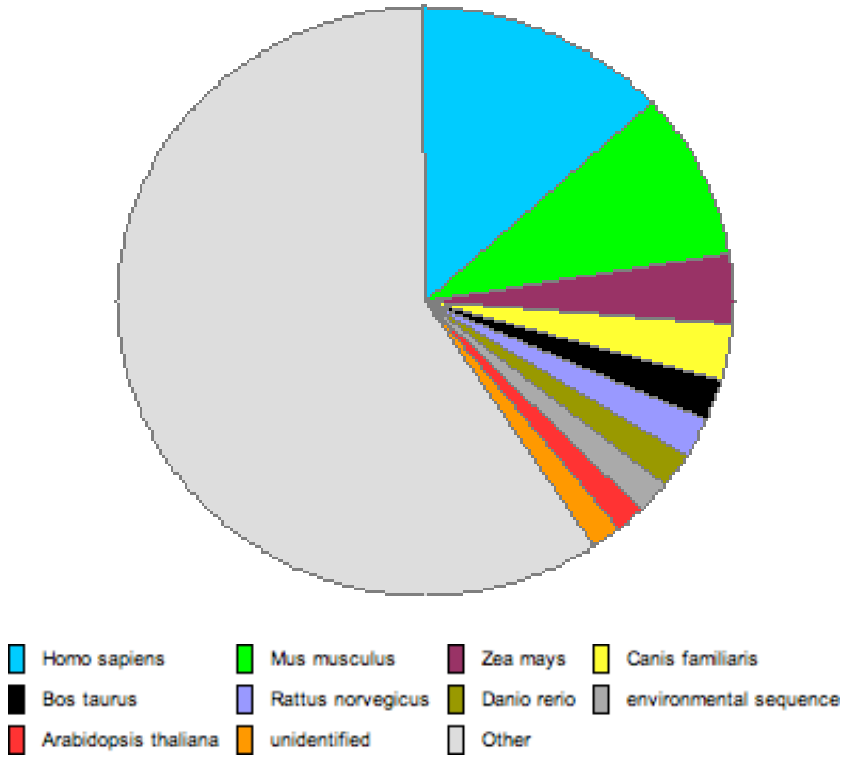
- EMBL/GenBank/DDBJ database of nucleic acids

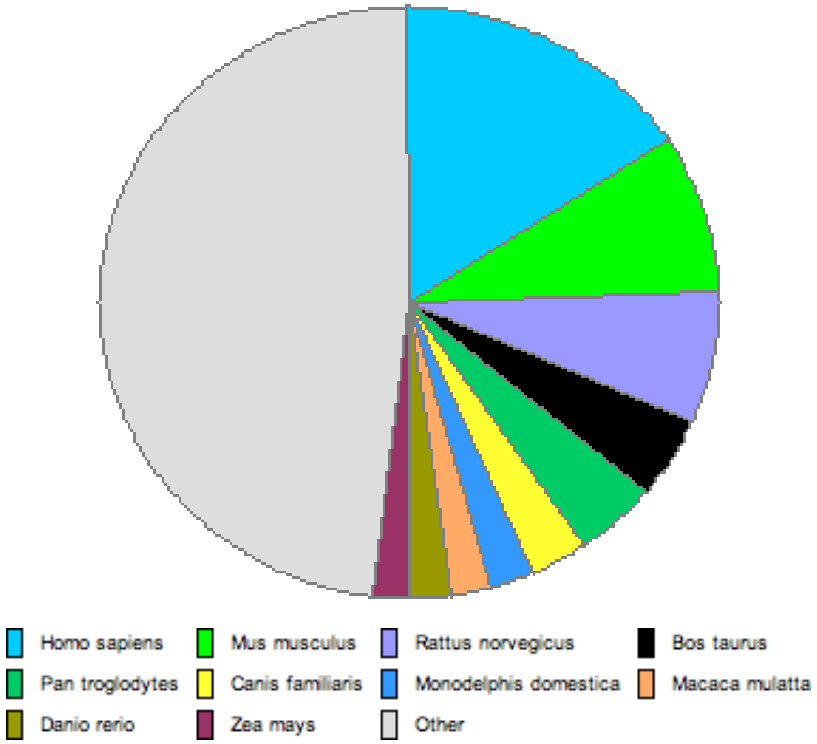




DNA and protein databases

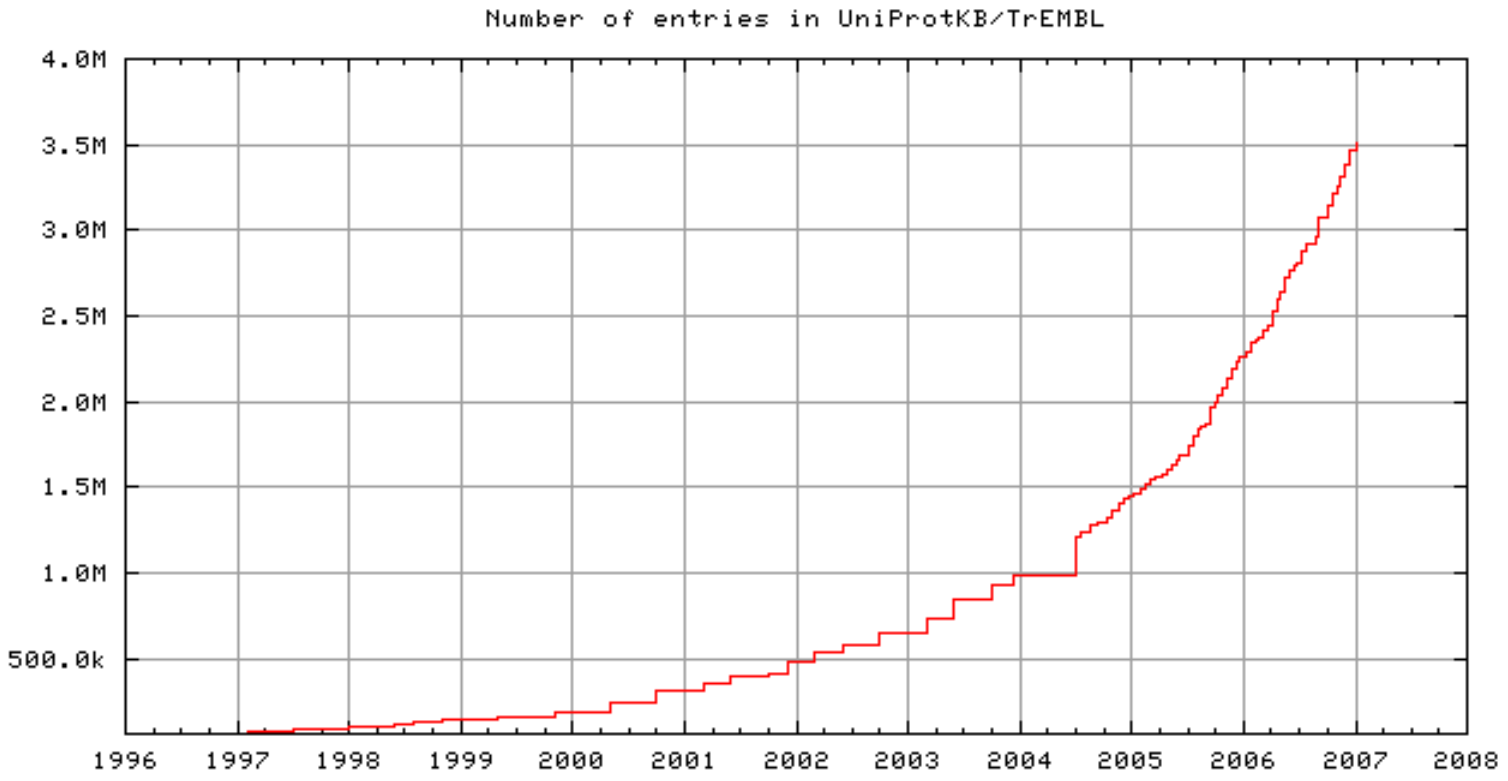
- EMBL/GenBank/DDBJ database of nucleic acids (cntd)





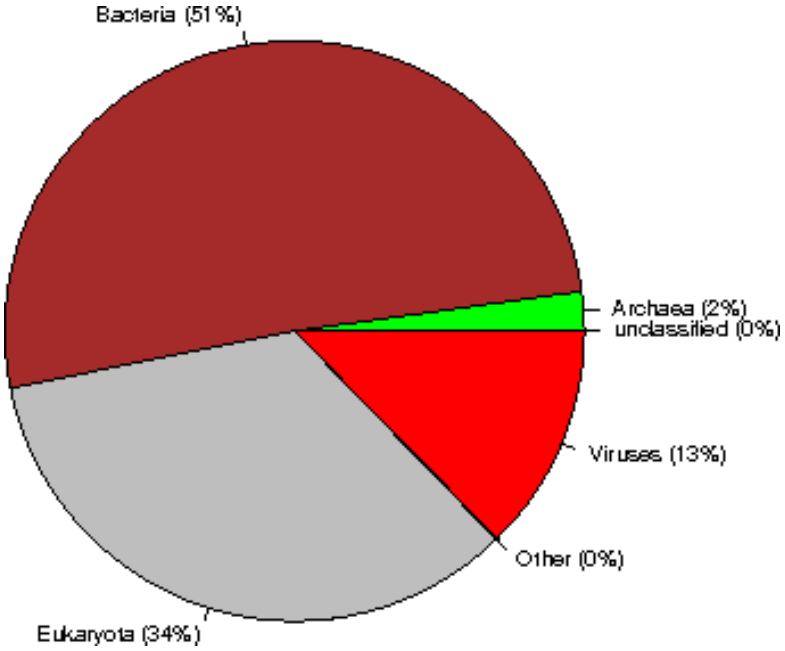
DNA and protein databases

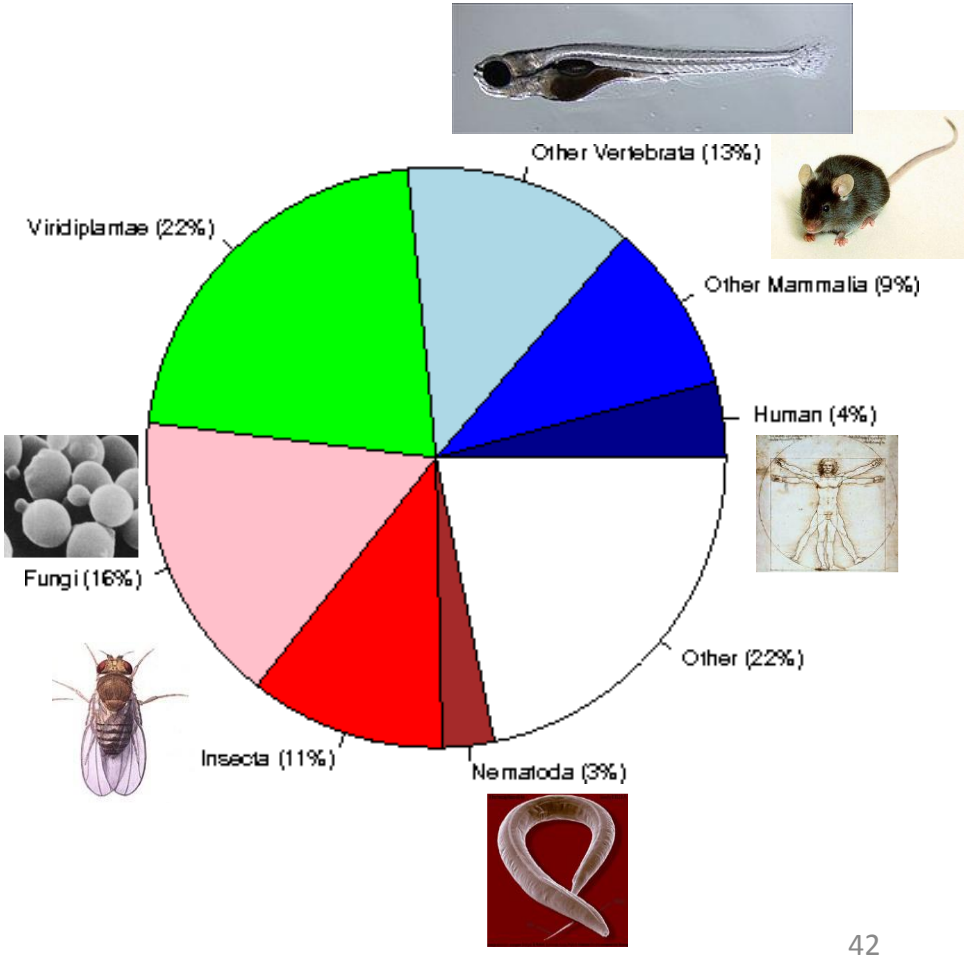
- SWISS-PROT & TrEMBL database of proteins



DNA and protein databases

- SWISS-PROT & TrEMBL database of proteins





Database searches

- Database searching consists of **many** pairwise alignments combined in one search.
- It helps determining the *function* and the evolutionary relationships
- Heuristic algorithms are used instead of DP. *Why?*
  - Size of SWISS-PROT + TrEMBL (Rel. 9.5):

    3.9M entries or 1,276M residues.

  - Exact algorithms are O(*NM*) fast.
- Heuristic methods can look at a small fraction of the searching space that will include all (or most) of the high scoring pairs.

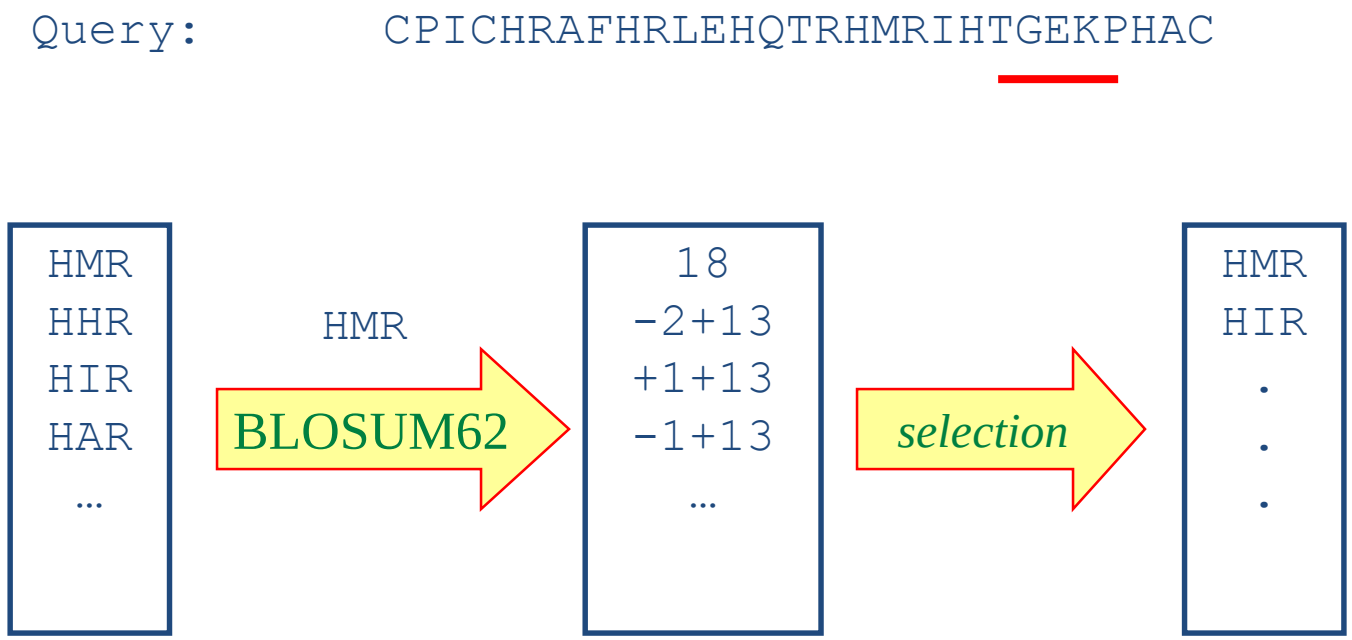
BLAST algorithm

- **B**asic **L**ocal **A**lignment **S**earch **T**ool - The method:

  - For each “word” (of fixed-length) in the query sequence, make a list of all neighbouring “words” that score above some threshold.
  - Scan the database for these words.
  - Perform (ungapped) “hit extension” until score < threshold.
  - Stop at maximum scoring extension.

BLAST algorithm (cntd)

- An example:



BLAST algorithm (cntd)

- An example:

```
Query:      CPICHRAFHRLEHQTRHMRIHTGEKPHAC
                              H+R
Sbjct:      CPLCDKAFHRLEHQTRHIRTHTGEKPHAC
```

BLAST algorithm (cntd)

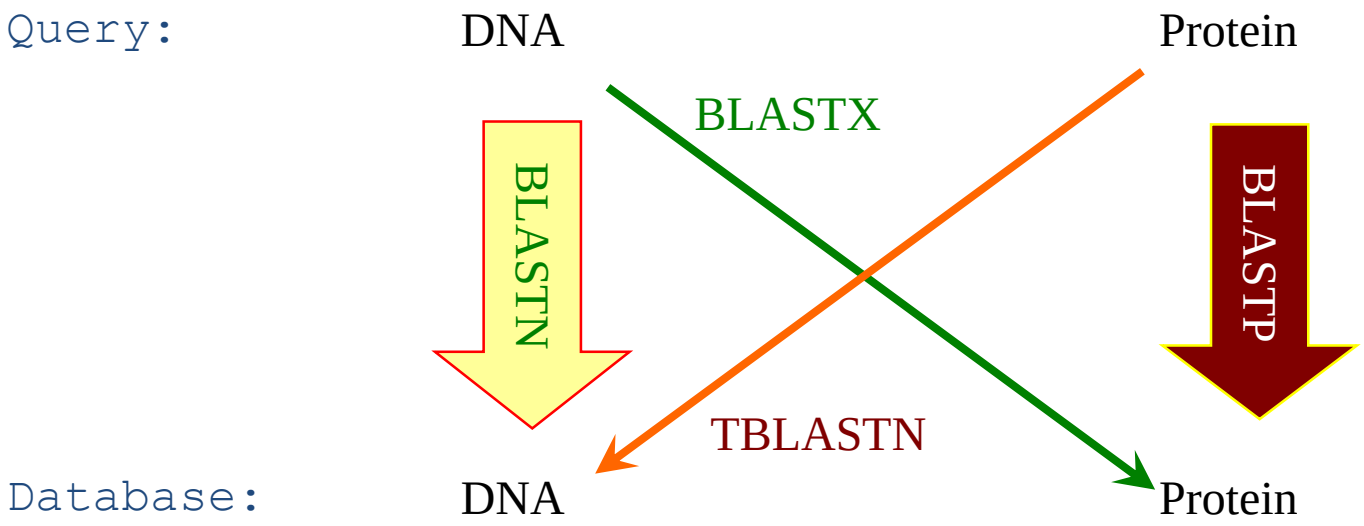
- An example:

```
Query:      CPICHRAFHRLEHQTRHMRIHTGEKPHAC
            CP+C +AFHRLEHQTR     H+R  HTGEKPHAC
Sbjct:      CPLCDKAFHRLEHQTRHIRTHTGEKPHAC
```

BLAST algorithm (cntd)

- The idea: a high scoring match alignment is very likely to contain a short stretch of very high scoring matches.
- Word length: 3 (proteins) and 11 (DNA).
- HSSP: multiple HSSPs can be reported for each database entry.
- Gapped alignments: more recent BLAST versions perform gapped alignments.

BLAST flavours

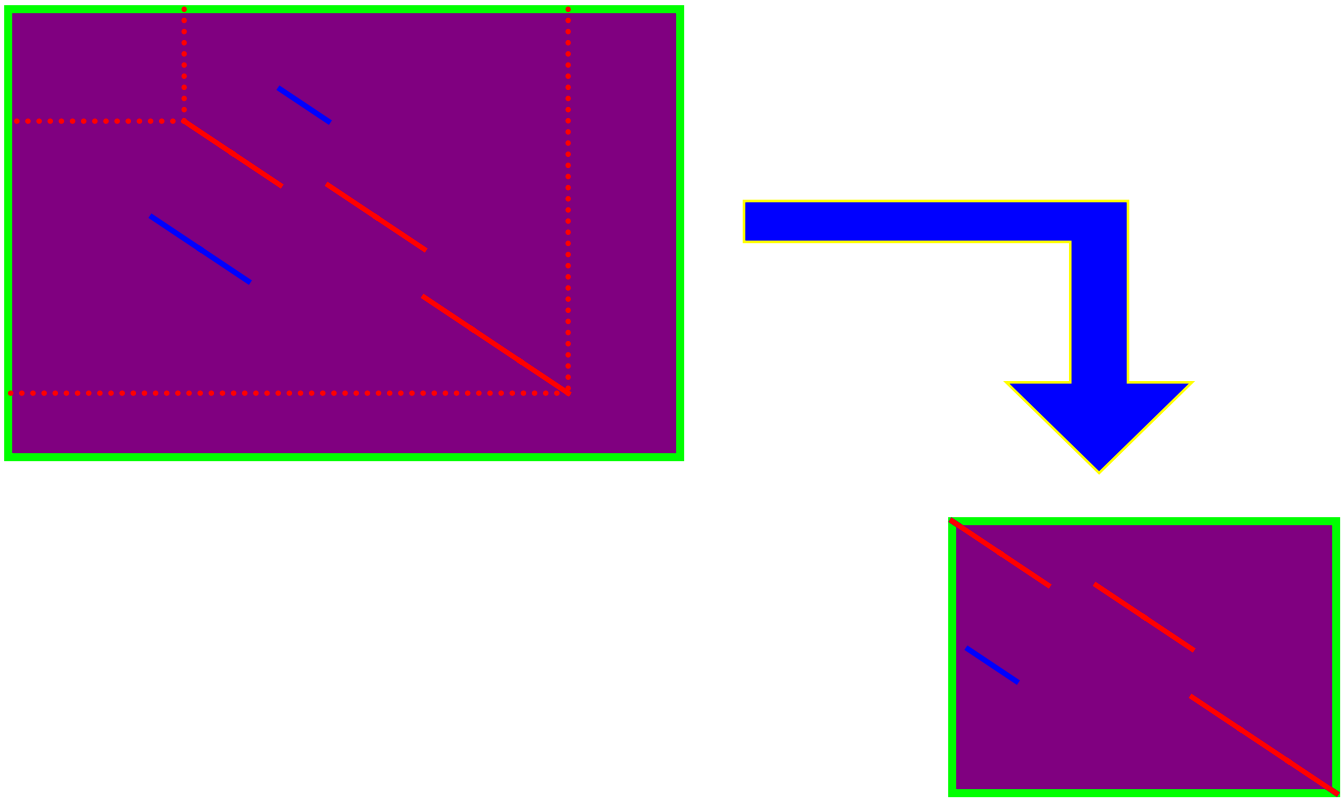


TBLASTX: DNA Query to DNA Database *via* translation

FASTA algorithm

- The method:
  - For each pair of sequences (query, subject), identify all identical “word” matches of (fixed) length.
  - Look for diagonals with many mutually supporting “word” matches.
  - The best diagonals are used to extend the word matches to find the maximal scoring (ungapped) regions.
  - Join ungapped regions, using gap costs.
  - Align the two (sub)regions using full dynamic programming techniques.

FASTA algorithm (cntd)



FASTA algorithm (cntd)

- The idea: a high scoring match alignment is very likely to contain a short stretch of identities.
- Word length: 2 (proteins) and 4-6 (DNA).
- HSSP: usually one (extended) gapped alignment is presented.

FASTA flavours