1-month Practical Course Genome Analysis (Integrative Bioinformatics & Genomics)

Lecture 3: Pair-wise alignment

Centre for Integrative Bioinformatics VU (IBIVU)
Vrije Universiteit Amsterdam
The Netherlands

www.ibivu.cs.vu.nl
heringa@cs.vu.nl

Pair-wise alignment

```
T D W V T A L K
T D W L - - I K
```

Combinatorial explosion

- 1 gap in 1 sequence: $n+1$ possibilities
- 2 gaps in 1 sequence: $(n+1)n$
- 3 gaps in 1 sequence: $(n+1)n(n-1)$, *etc.*

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

→ 2 sequences of 300 a.a.: ~10^{88} alignments
2 sequences of 1000 a.a.: ~10^{600} alignments!

Technique to overcome the combinatorial explosion: Dynamic Programming

- Alignment is simulated as Markov process, all sequence positions are seen as independent
- Chances of sequence events are independent
  - Therefore, probabilities per aligned position need to be multiplied
  - Amino acid matrices contain so-called log-odds values ($\log_{10}$ of the probabilities), so probabilities can be summed

To say the same more statistically…

- To perform statistical analyses on messages or sequences, we need a reference model.
- **The model:** each letter in a sequence is selected from a defined alphabet in an *independent and identically distributed* (*i.i.d.*) manner.
- This choice of model system will allow us to compute the statistical significance of certain characteristics of a sequence, its subsequences, or an alignment.
- Given a probability distribution, P_i, for the letters in a *i.i.d.* message, the probability of seeing a particular sequence of letters *i, j, k, ... n* is simply $P_i P_j P_k \cdots P_n$.
- As an alternative to multiplication of the probabilities, we could sum their logarithms and exponentiate the result. The probability of the same sequence of letters can be computed by exponentiating $\log P_i + \log P_j + \log P_k + \cdots + \log P_n$.
- In practice, when aligning sequences we only add log-odds values (residue exchange matrix) but we do not exponentiate the final score.

Sequence alignment History of the Dynamic Programming (DP) algorithm

1970 **Needleman-Wunsch global pair-wise alignment**

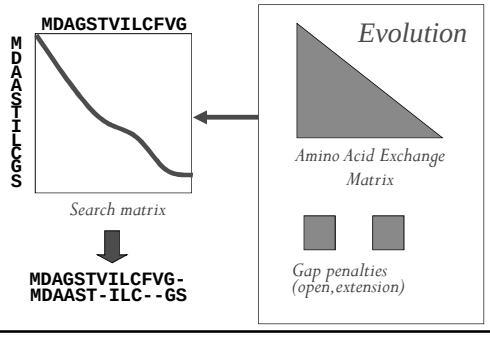
Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J Mol Biol.* 48(3):443-53.

1981 **Smith-Waterman local pair-wise alignment**

Smith, TF, Waterman, MS (1981) Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195-197.

Pairwise sequence alignment

Global dynamic programming



How to determine similarity

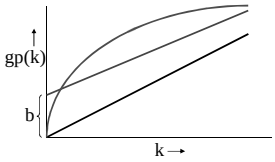
Frequent evolutionary events at the DNA level:

1. Substitution
2. Insertion, deletion
3. Duplication
4. Inversion

We will restrict ourselves to these events

Dynamic programming

Three kinds of Gap Penalty schemes:



- **Linear:** gp(k)=ak
- **Affine:** gp(k)=b+ak (most commonly used)
- **Concave (general)**: e.g. gp(k)=log(k)

k is the gap length

Dynamic programming

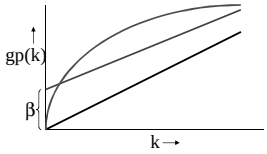
Scoring alignments

– Substitution (or match/mismatch)

- DNA
- proteins

– Gap penalty

- Linear: gp(k)=αk
- Affine: gp(k)=$\beta+\alpha k$
- Concave, e.g.: gp(k)=log(k)



General alignment score:

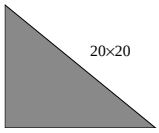
$$S_{a,b} = \sum s(a,b) - \sum_k N_k \bullet gp(k)$$

The score of an alignment is the sum of the scores over all alignment columns

Dynamic programming

Scoring alignments

```
T D W V T A L K
T D W L - - I K
```



Amino Acid Exchange Matrix

Gap penalties (open, extension)

Score: s(T,T) + s(D,D) + s(W,W) + s(V,L) –P_o -P_x + + s(L,I) + s(K,K)

Constant vs. affine gap scoring

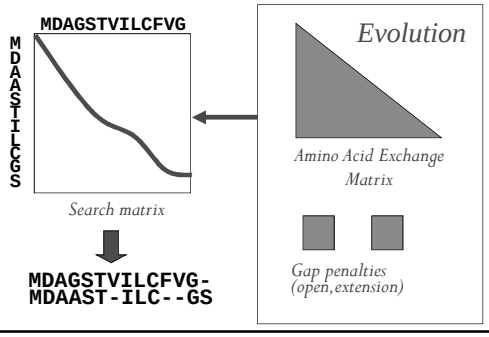
Scoring \ Gap	opening	extension
Constant	-2	-2
Affine	-3	-1

…and +1 for match

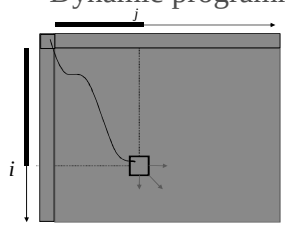
```
Seq1   G  T  A  -  -            G  -  T  -  A
Seq2   -  -  A  T  G            -  A  T  G  -
Const  -2 -2 1  -2 -2 (SUM = -7) -2 -2 1  -2 -2 (SUM = -7)
Affine -4  1   -4  (SUM = -7)  -3 -3 1  -3 -3 (SUM = -11)
```

Pairwise sequence alignment

Global dynamic programming



Dynamic programming



The cell [*i*, *j*] contains the alignment score of the best scoring alignment of subsequence 1..*i* and 1..*j*, that is, the subsequences up to [*i*, *j*]

Cell [i, j] does not 'know' what that best scoring alignment is (it is one out of many possibilities)

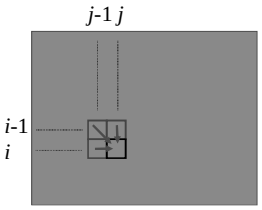
Extend alignment from cell [i, j]

The DP algorithm

- Goal: find the maximal scoring alignment
- Dynamic programming
  - Solve smaller subproblem(s)
  - Iteratively extend the solution
- Scores: m match, -s mismatch, -g for insertion/deletion
- 3 ways to extend the alignment:

	X[1…i-1] Y[1…j-1] X[i] Y[j]	X[1…i] Y[1…j-1] - Y[j]	X[1…i-1] X[i] Y[1…j] -
M[i,j] -	M[i-1, j-1] + m - s	M[i, j-1] - g	M[i-1, j] - g

Global dynamic programming



Value from residue exchange matrix

$$H(i,j) = Max \begin{cases} H(i\text{-}1,j\text{-}1) + S(i,j) & \text{diagonal} \\ H(i\text{-}1,j) - g & \text{vertical} \\ H(i,j\text{-}1) - g & \text{horizontal} \end{cases}$$

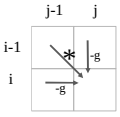
This is a recursive formula

The algorithm – final equation

$$M[i,j] = \max \begin{cases} M[i\text{-}1, j\text{-}1] + score(X[i],Y[j]) \\ M[i, j\text{-}1] - g \\ M[i\text{-}1, j] - g \end{cases}$$

Corresponds to:

$$\begin{matrix} X_1 \ldots X_{i-1} & X_i \\ Y_1 \ldots Y_{j-1} & Y_j \end{matrix} \quad \begin{matrix} X_1 \ldots X_i & - \\ Y_1 \ldots Y_{j-1} & Y_j \end{matrix} \quad \begin{matrix} X_1 \ldots X_{i-1} & X_i \\ Y_1 \ldots Y_j & - \end{matrix}$$



Example: *global* alignment of two sequences

- Align two DNA sequences:
  - GAGTGA
  - GAGGCGA (note the length difference)
- Parameters of the algorithm:
  - Match: score(A,A) = 1
  - Mismatch: score(A,T) = -1
  - Gap: g = -2

$$M[i,j] = \max \begin{cases} M[i\text{-}1, j\text{-}1] \pm 1 \\ M[i, j\text{-}1] - 2 \\ M[i\text{-}1, j] - 2 \end{cases}$$

The algorithm. Step 1: init

- Create the matrix
- Initiation
  - 0 at [0,0]
  - Apply the equation…

$$M[i,j] = \max \begin{cases} M[i\text{-}1, j\text{-}1] \pm 1 \\ M[i, j\text{-}1] - 2 \\ M[i\text{-}1, j] - 2 \end{cases}$$

	j→	0	1	2	3	4	5	6
i↓		-	G	A	G	T	G	A
0	-	0						
1	G							
2	A							
3	G							
4	G							
5	C							
6	G							
7	A							

The algorithm. Step 1: init

$$M[i,j] = \max \begin{cases} M[i-1,j-1] \pm 1 \\ M[i,j-1] - 2 \\ M[i-1,j] - 2 \end{cases}$$

- Initiation of the matrix:
  - 0 at pos [0,0]
  - Fill in the first row using the "→" rule
  - Fill in the first column using "↓"

		j					
	-	G	A	G	T	G	A
-	0	-2	-4	-6	-8	-10	-12
G	-2						
A	-4						
G	-6						
G	-8						
C	-10						
G	-12						
A	-14						

i

The algorithm. Step 2: fill in

$$M[i,j] = \max \begin{cases} M[i-1,j-1] \pm 1 \\ M[i,j-1] - 2 \\ M[i-1,j] - 2 \end{cases}$$

- Continue filling in of the matrix, remembering from which cell the result comes (arrows)

		j					
	-	G	A	G	T	G	A
-	0	-2	-4	-6	-8	-10	-12
G	-2	1	-1	-3			
A	-4	-1	2				
G	-6						
G	-8						
C	-10						
G	-12						
A	-14						

i

The algorithm. Step 2: fill in

$$M[i,j] = \max \begin{cases} M[i-1,j-1] \pm 1 \\ M[i,j-1] - 2 \\ M[i-1,j] - 2 \end{cases}$$

- We are done…
- Where's the result?

The lowest-rightmost cell

		j					
	-	G	A	G	T	G	A
-	0	-2	-4	-6	-8	-10	-12
G	-2	1	-1	-3	-5	-7	-9
A	-4	-1	2	0	-2	-4	-6
G	-6	-3	0	3	1	-1	-3
G	-8	-5	-2	1	2	2	0
C	-10	-7	-4	-1	0	1	1
G	-12	-9	-6	-3	-2	1	0
A	-14	-11	-8	-5	-4	-1	2

i

The algorithm. Step 3: traceback

- Start at the last cell of the matrix
- Go against the direction of arrows
- Sometimes the value may be obtained from more than one cell (which ones?)

$$M[i,j] = \max \begin{cases} M[i-1,j-1] \pm 1 \\ M[i,j-1] - 2 \\ M[i-1,j] - 2 \end{cases}$$

		j					
	-	G	A	G	T	G	A
-	0	-2	-4	-6	-8	-10	-12
G	-2	1	-1	-3	-5	-7	-9
A	-4	-1	2	0	-2	-4	-6
G	-6	-3	0	3	1	-1	-3
G	-8	-5	-2	1	2	2	0
C	-10	-7	-4	-1	0	1	1
G	-12	-9	-6	-3	-2	1	0
A	-14	-11	-8	-5	-4	-1	2

i

The algorithm. Step 3: traceback

- Extract the alignments

a)
```
GAGT-GA
GAGGCGA
```

b)
```
GA-GTGA
GAGGCGA
```

c)
```
GAG-TGA
GAGGCGA
```

		j					
	-	G	A	G	T	G	A
-	0	-2	-4	-6	-8	-10	-12
G	-2	1	-1	-3	-5	-7	-9
A	-4	-1	2	0	-2	-4	-6
G	-6	-3	0	3	1	-1	-3
G	-8	-5	-2	1	2	2	0
C	-10	-7	-4	-1	0	1	1
G	-12	-9	-6	-3	-2	1	0
A	-14	-11	-8	-5	-4	-1	2

a
b

i

Global dynamic programming

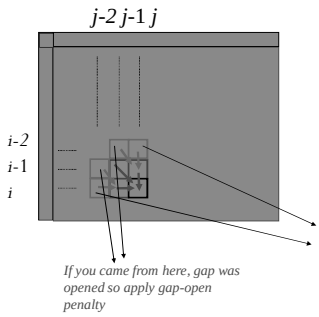
$$H(i,j) = Max \begin{cases} H(i-1,j-1) + S(i,j) & \text{diagonal} \\ H(i-1,j) - g & \text{vertical} \\ H(i,j-1) - g & \text{horizontal} \end{cases}$$

Problem with simple DP approach:

•Can only do linear gap penalties

•Not suitable for affine or concave penalties, but algorithm can be extended to deal with affine penalties

Global dynamic programming
using affine penalties



Looking back from cell (*i, j*) we can adapt the algorithm for *affine gap penalties* by looking back to four more cells (magenta)

Affine gaps

- Penalties:
  g_o - gap opening (e.g. -8)
  g_e - gap extension (e.g. -1)

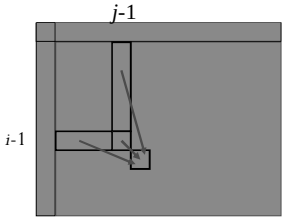
$$\begin{matrix}X_1\ldots X_i\\ Y_1\ldots Y_j\end{matrix}\ M[i,j] = \text{score}(X[i], Y[j]) + \max\begin{cases} M[i-1, j-1] \\ I_x[i-1, j-1] \\ I_y[i-1, j-1]\end{cases}$$

$$I_x[i,j] = \max\begin{cases} M[i, j-1] + g_o \\ I_x[i, j-1] + g_e\end{cases} \qquad \begin{matrix}X_1\ldots X_i\\ Y_1\ldots Y_{j-1}\end{matrix}\ \begin{matrix}-\\ Y_j\end{matrix}$$

$$I_y[i,j] = \max\begin{cases} M[i-1, j] + g_o \\ I_y[i-1, j] + g_x\end{cases} \qquad \begin{matrix}X_1\ldots\\ Y_1\ldots Y_{j-1}\end{matrix}\ \begin{matrix}-\\ Y_j\end{matrix}$$

- @ home: think of boundary values M[*,0], I[*,0] *etc.*

Global dynamic programming
all types of gap penalty

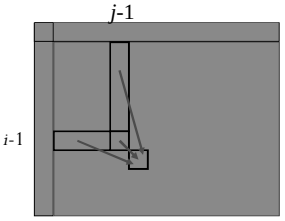


The complexity of this DP algorithm is increased from $O(n^2)$ to $O(n^3)$

The gap length is known exactly and so any gap penalty regime can be used

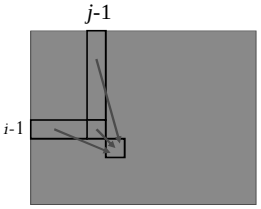
$$S_{i,j} = s_{i,j} + Max \begin{cases} Max\{S_{0<x<i-1,\,j-1} - Gap(i\text{-}x\text{-}1)\} \\ S_{i-1,j-1} \\ Max\{S_{i-1,\,0<y<j-1} - Gap(j\text{-}y\text{-}1)\}\end{cases}$$

Global dynamic programming
if affine penalties are used



$$S_{i,j} = s_{i,j} + Max \begin{cases} Max\{S_{0<x<i-1,\,j-1} - G_o - (i\text{-}x\text{-}1)*G_e\} \\ S_{i-1,j-1} \\ Max\{S_{i-1,\,0<y<j-1} - G_o - (j\text{-}y\text{-}1)*G_e\}\end{cases}$$

Global dynamic programming
Linear, Affine or Concave gap penalties



All penalty schemes are possible because the exact length of the gap is known

Gap opening penalty

$$S_{i,j} = s_{i,j} + Max \begin{cases} Max\{S_{0<x<i-1,\,j-1} - Pi - (i\text{-}x\text{-}1)Px\} \\ S_{i-1,j-1} \\ Max\{S_{i-1,\,0<y<j-1} - Pi - (j\text{-}y\text{-}1)Px\}\end{cases}$$

Gap extension penalty

DP is a two-step process

- *Forward step*: calculate scores
- *Trace back*: start at lower-right corner cell and reconstruct the path leading to the highest score
  - These two steps lead to the highest scoring global alignment (the optimal alignment)
  - This is guaranteed when you use DP!

Time and memory complexity of DP

- The *time complexity* is **O(n^2)**: for aligning two sequences of *n* residues, you need to perform n^2 algorithmic steps (square search matrix has n^2 cells that need to be filled)
- The *memory (space) complexity* is also **O(n^2)**: for aligning two sequences of *n* residues, you need a square search matrix of *n* by *n* containing n^2 cells