

Lecture 3:

Dynamic Programming and Alignment

Main Algorithms with Applications in Bioinformatics

Grégory Nuel

Laboratoire Statistique et Génome
University of Evry, CNRS (8071), INRA (1152)
France

Bioinformatics and Comparative Genome Analysis, Institut
Pasteur of Tunis from March 18 to April 6, 2007

Outline

- 1 **Dynamic Programming**
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 **Local Score of One Sequence**
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 **Alignment of Sequences**
 - Global Alignment
 - Local Alignment
 - Heuristics

Outline

- 1 **Dynamic Programming**
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 **Local Score of One Sequence**
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 **Alignment of Sequences**
 - Global Alignment
 - Local Alignment
 - Heuristics

Computing $n!$

Definition ($n!$)

For all $n \geq 1$ we have $n! = 1 \times 2 \times \dots \times n$ and $0! = 1$. Hence $1! = 1$, $2! = 1 \times 2 = 2$, $3! = 1 \times 2 \times 3 = 6, \dots$

function factorial(n)

```
1:  $r = 1$   
2: for  $i = 2 \dots n$  do  
3:    $r = r \times i$   
4: return  $r$ 
```

$\Rightarrow O(n)$ to compute $n!$ hence $O(n^2)$ to compute $1!, 2!, 3!, \dots, n!$.

Computing $n!$

Definition ($n!$)

For all $n \geq 1$ we have $n! = 1 \times 2 \times \dots \times n$ and $0! = 1$. Hence
 $1! = 1$, $2! = 1 \times 2 = 2$, $3! = 1 \times 2 \times 3 = 6, \dots$

function smartfactorial(n)

```
1: static  $n_{\text{top}} = 3$  and static  $r[33] = \{1, 1, 2, 6\}$ 
2: if  $n > 32$  then
3:   we need more memory
4: else
5:   while  $n_{\text{top}} < n$  do
6:      $n_{\text{top}} = n_{\text{top}} + 1$ 
7:      $r[n_{\text{top}}] = r[n_{\text{top}} - 1] \times n_{\text{top}}$ 
8:   return  $r[n]$ 
```

$\Rightarrow O(n)$ to compute $1!, 2!, 3!, \dots, n!$ but we need $O(33)$ in space.



What is Dynamic Programming ?

Dynamic programming is a method to solve a problem (like computing $n!$) using solutions of subproblems (like the values of $1!$, $2!$, \dots ($n-1!$)) that takes **much less time** than naive approaches. Such an approach usually relies on

- a **recurrence relation** (like $n! = (n-1)! \times n$)
- a data structure to memorize subproblems solutions (thus **more space requirements** than with naive approaches)

Example (Problem solved by dynamic programming)

- n factorial, Fibonacci numbers, \dots
- longest common subsequence
- Viterbi algorithm
- local score of one sequence, alignment of sequences, \dots

Outline

- 1 **Dynamic Programming**
 - The Principle
 - **Application: Longest Common Subsequence**
 - Application: Viterbi
- 2 **Local Score of One Sequence**
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 **Alignment of Sequences**
 - Global Alignment
 - Local Alignment
 - Heuristics

Longest Common Subsequence

The problem

Given two sequence $X_1^p = X_1 \dots X_p$ and $Y_1^q = Y_1 \dots Y_q$ over the same alphabet \mathcal{A} , we want to find their **Longest Common Subsequence** $\text{LCS}(X_1^p, Y_1^q)$.

Example: $\text{LCS}(\text{XMJYAUZ}, \text{MZJAWXU}) = \text{MJAU}$

The recurrence relation

For all $i \leq p$ and $j \leq q$ we have $\text{LCS}(X_1^i, Y_1^j) =$

$$\begin{cases} \emptyset & \text{if } i \leq 0 \text{ or } j \leq 0 \\ \text{LCS}(X_1^{i-1}, Y_1^{j-1}) + X_i & \text{if } X_i = Y_j \\ \text{LCS}(X_1^{i-1}, Y_1^j) \text{ or } \text{LCS}(X_1^i, Y_1^{j-1}) & \text{(the longest) otherwise} \end{cases}$$

where $+$ is the concatenation symbol.

The algorithm

Length

- 1: initialize $(L_{i,j})_{0 \leq i \leq p, 0 \leq j \leq q}$ to 0
- 2: **for** $i = 1 \dots p$ **do**
- 3: **for** $i = 1 \dots q$ **do**
- 4: **if** $X_i = Y_j$ **then**
- 5: $L_{i,j} = L_{i-1,j-1} + 1$
- 6: **else**
- 7: $L_{i,j} = \max(L_{i-1,j} \text{ and } L_{i,j-1})$
- 8: **return** $L_{p,q}$

Traceback

- 1: find a “path” connection $L_{0,0}$ to $L_{p,q}$
- 2: return the concatenation of the matching letters of this path

A simple example

Example ($X_1^7 = \text{XMJYAUZ}$ and $Y_1^6 = \text{MZJAWXU}$)

	j	0	1	2	3	4	5	6	7
i			M	Z	J	A	W	X	U
0		0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	1	1	1	1	1	1	1
3	J	0	1	1	2	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	3	3	3	3
6	U	0	1	1	2	3	3	3	4
7	Z	0	1	2	2	3	3	3	4

A simple example

Example ($X_1^7 = \text{XMJYAUZ}$ and $Y_1^6 = \text{MZJAWXU}$)

	j	0	1	2	3	4	5	6	7
i			M	Z	J	A	W	X	U
0		0	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	1	1	1	1	1	1	1
3	J	0	1	1	2	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	3	3	3	3
6	U	0	1	1	2	3	3	3	4
7	Z	0	1	2	2	3	3	3	4

Outline

- 1 **Dynamic Programming**
 - The Principle
 - Application: Longest Common Subsequence
 - **Application: Viterbi**
- 2 **Local Score of One Sequence**
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 **Alignment of Sequences**
 - Global Alignment
 - Local Alignment
 - Heuristics

Best hidden path of a HMM

The Problem

How to compute the **best path** $s^* = \operatorname{argmax}_s \mathbb{P}_\theta(S = s \mid X = x)$ of a HMM ?

Proposition

$$Z_i(u) = \max_{s_1, \dots, s_{i-1}} \mathbb{P}_\theta(S_1 = s_1, \dots, S_{i-1} = s_{i-1}, S_i = u \mid X = x)$$

then we have the following **recurrence relation**

$$Z_i(u) = \max_{t \in \mathcal{S}} Z_{i-1}(t) \nu(t, u) \mu_u(X_i)$$

Viterbi Algorithm for a MOM1 model

The log-likelihood of s^*

- 1: initialize $L_1(t) = \log \mu_0(t) + \log \mu_t(x_1)$ for all $t \in \mathcal{S}$
- 2: **for** $i = 2 \dots \ell$ **do**
- 3: **for all** $u \in \mathcal{S}$ **do**
- 4: $L_i(u) = \max_{t \in \mathcal{S}} L_{i-1}(t) + \log \nu(t, u) + \log \mu_u(x_i)$
- 5: $T_{i-1}(u) = \operatorname{argmax}_{t \in \mathcal{S}} L_{i-1}(t) + \log \nu(t, u) + \log \mu_u(x_i)$
- 6: **return** L_ℓ

Traceback

- 1: $s_\ell^* = \operatorname{argmax}_{t \in \mathcal{S}} Z_\ell(t)$
- 2: **for** $i = \ell - 1 \dots 1$ **do**
- 3: $s_i^* = T_i(s_{i+1}^*)$

A simple example (1)

Example ($\mathcal{S} = \{1, 2, 3\}$ and $\mathcal{A} = \{a, c, g, t\}$)

ν gives the transition between hidden states

$$N = \log \nu = \begin{pmatrix} 3.0 & -1.0 & -1.0 \\ 1.6 & 2.5 & 1.6 \\ -1.0 & -1.0 & 3.0 \end{pmatrix}$$

and μ_t gives the distribution of X_i the letter if $S_i = t$

$$M = \log \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} = \begin{pmatrix} 2.0 & 1.5 & 2.5 & 1.0 \\ 1.5 & 2.0 & 1.0 & 2.5 \\ 1.9 & 1.9 & 1.9 & 1.9 \end{pmatrix}$$

A simple example (2)

Example ($\mathcal{S} = \{1, 2, 3\}$ and $\mathcal{A} = \{a, c, g, t\}$)

$x = \text{ctagacc}$ and (as a simplification) we assume that $L_1(t) = M(t, c)$ we hence get the following table:

	c	t	a	g	a	c	c
$L_i(1)$	1.5	→ 5.5	11.1	→ 16.1	→ 21.6	→ 26.1	→ 30.6
$L_i(2)$	2.0	→ 7.0	→ 10.5	→ 14.5	→ 18.0	→ 22.6	→ 27.6
$L_i(3)$	1.9	→ 6.8	→ 11.7	→ 16.6	→ 21.5	→ 26.6	→ 31.3

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 **Local Score of One Sequence**
 - **Motivation and Notation**
 - Computing the Local Score
 - Assessing the Significance
- 3 Alignment of Sequences
 - Global Alignment
 - Local Alignment
 - Heuristics

Homogeneous Regions in Biological Sequences

Example (gc rich regions in DNA)

$X = \text{aaa}\mathbf{gaaa}\mathbf{ggg}\mathbf{cacac}\mathbf{agcc}\mathbf{agaaataatttt}\mathbf{ctt}$

is there one (or more) **gc rich** region in this DNA sequence ?

Example (hydrophobic regions in proteins)

$X = \text{YVP}\mathbf{I}\mathbf{SM}\mathbf{Y}\mathbf{CLQ}\mathbf{WLL}\mathbf{PVLL}\mathbf{IPKPL}\mathbf{N}\mathbf{W}\mathbf{SDG}\mathbf{VAS}$

T, I, L, M, F, W and C are very **hydrophobic** amino-acids. Is there any hydrophobic region in this protein ?

Sliding Window

Idea

Given a **window size** h , for each $i = 1 \dots \ell - h + 1$ score the feature of interest (gc content, hydrophobic content) in the **sliding window** $[i, i + h - 1]$.

Example (with $h = 5$ and score = frequency)

```

aaagaaa gggcaccagcaga aataatttctt
111223444332334344322100000111----

YVPISM YCLQWLLPVLLIPKPLN WSDGVAS
122333344333333433232211----
    
```

Another Approach

Remarks

The sliding window method is very **simple** to understand, have a low **linear complexity**, but suffers **several drawbacks**:

- how to choose the **window size** h ?
- where are exactly the **limits** of our regions of interest ?
- how to choose the **scoring function** ?

⇒ we need **another approach** !

Definition (Local Score)

Given a scoring function $S : \mathcal{A} \rightarrow \mathbb{R}$, the **local score** H of the sequence $X = X_1 \dots X_\ell$ is defined by:

$$H = \max_{1 \leq i < i' \leq \ell} \sum_{j=i}^{i'} S(X_j)$$

Examples

Example ($S([gc]) = +1$ and $S([ac]) = -1$)

$X = \text{aaagaaa} \boxed{\text{gggcacacagccag}} \text{aaataattttctt}$

Example ($S([gc]) = +1$ and $S([ac]) = -2$)

$X = \text{aaagaaa} \boxed{\text{gggc}} \text{acacagccagaaataattttctt}$

Example ($S([TILMFWC]) = +1$ and $S(\{TILMFWC\}) = -1$)

$X = \text{YVP} \boxed{\text{ISMYCLQWLLPVLIPKPLNW}} \text{SDGVAS}$

Example ($S([TILMFWC]) = +1$ and $S(\{TILMFWC\}) = -2$)

$X = \text{YVP} \text{ISMY} \boxed{\text{CLQWLL}} \text{PVLIPKPLNWSDGVAS}$

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 Local Score of One Sequence
 - Motivation and Notation
 - **Computing the Local Score**
 - Assessing the Significance
- 3 Alignment of Sequences
 - Global Alignment
 - Local Alignment
 - Heuristics

Brut Force has a Cubic Complexity

Brut force

Score **each of the possible** segments and pick up the best score.

- ℓ segments of length 1
- $\ell - 1$ segments of length 2
- ...
- 1 segment of length ℓ

⇒ resulting complexity is $O(\ell^3)$

Proposition

If we denote by H_i the local score of $X_1 \dots X_i$ then we have the following **recurrence relation**:

$$H_0 = 0 \quad \text{and} \quad H_i = \max(0, H_{i-1} + S(X_i)) \quad \forall 1 \leq i \leq \ell$$

Linear Algorithm

Algorithm

- 1: $H_0 = 0$
- 2: **for** $i = 1 \dots \ell$ **do**
- 3: $H_i = \max(0, H_{i-1} + S(X_i))$
- 4: **return** $H = \max_{1 \leq i \leq \ell} H_i$

\Rightarrow complexity is $O(\ell)$ in space and time

Example ($S([gc]) = +1$ and $S([ac]) = -1$)

X_i - aaagaaa gggc a c a c a g c c a g a a a t a a t t t t c t t
 H_i 00001000123434343456567654321000100
 we get $H = 7$ and can easily find the corresponding segment

Interest of the Local Score Approach

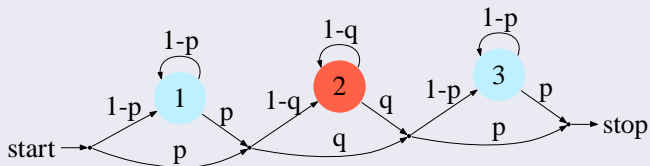
Remarks

- **simple** and **efficient** linear algorithm
 - directly point out **segments** of interest
 - can be used with **complex scoring function** (ex: Kyte-Doolittle hydrophobic scale)
 - all **suboptimal segments** can be found in $O(\ell)$ thanks to the algorithm from Ruzzo and Tompa (1999)
- ⇒ far more **elegant approach** than sliding windows from a wide range of problem encountered in sequence analysis.

Connexion with HMM (1)

M0M1 HMM model

$\mathcal{S} = \{1, 2, 3\}$, transition between **hidden states** is given by



and for all $a \in \mathcal{A}$ we have

$$\begin{cases} \mathbb{P}(X_i = a | S_i = 1) = \mu(a) \\ \mathbb{P}(X_i = a | S_i = 2) = \nu(a) \\ \mathbb{P}(X_i = a | S_i = 3) = \mu(a) \end{cases}$$

Connexion with HMM (2)

Proposition

With $\theta = (p, q, \mu, \nu)$ we have:

$$L(\theta | X, S) = \ell \log(1 - p) - 2 \log p - \log q \\ + \sum_{i=1}^{\ell} \log \mu(X_i) + \sum_{i=1}^{\ell} \mathbb{I}_{S_i=3} \log \frac{(1 - q)\nu(X_i)}{(1 - p)\mu(X_i)}$$

hence with the scoring function

$$S(a) = \log \frac{(1 - q)\nu(a)}{(1 - p)\mu(a)} \quad \forall a \in \mathcal{A}$$

we get

$$L(\theta | X, S = s^*) = H + \text{constant}$$

where s^* is the Viterbi path and H is the local score.

Connexion with HMM (3)

Remarks

- The local score is **totally equivalent** to a M0M1 model
- it is hence possible to use HMM parameter estimation to **estimate scoring function**
- HMM provides a natural path to **extend** the local score:
 - best score on **two segments** (or three, or four, ...)
 - **dependant** scoring function
 - **generalized HMM** to avoid geometric distribution for the segment lengths
 - ...

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 Local Score of One Sequence
 - Motivation and Notation
 - Computing the Local Score
 - **Assessing the Significance**
- 3 Alignment of Sequences
 - Global Alignment
 - Local Alignment
 - Heuristics

Notion of p-value

Problem

Using $S([\text{gc}] = +1$ and $S([\text{at}] = -1$, how to interpret that $H(X) = H(Y)$ if we assume either of:

A_1 X is 10 times smaller than Y

A_2 $\mathbb{P}_X([\text{gc}]) = 90\%$ while $\mathbb{P}_Y([\text{gc}]) = 20\%$

A_3 we have both H_1 and H_2

\Rightarrow need of **p-values**

Definition (p-value)

If we assume that $X = X_1 \dots X_\ell$ is generated according to a **random model** (ex: M_0, M_1, \dots) then the **p-value** of an observed result H_{obs} is given by:

$$\text{p-value} = \mathbb{P}(H \geq H_{\text{obs}})$$

Asymptotic Approximations

Proposition (Iglehart, 1972 and Karlin et al., 1990)

$$H \sim \text{Gumble} \left(\frac{\log \ell - \log K}{\lambda} ; \frac{1}{\lambda} \right)$$

which mean that

$$\mathbb{P} \left(H \geq \frac{\log \ell}{\lambda} + u \right) \simeq 1 - \exp \left(-K e^{-\lambda u} \right)$$

Two way to compute K and λ

- **analytically** in the M0 case (λ is easy, K requires more complex computations)
- by **simulations** using

$$\log \left(-\log \mathbb{P}(H < x) \right) \simeq -\lambda x + \log K + \log \ell$$

Exact Computations

Remark

- once λ and K are computed, asymptotic approximations are **very fast to compute**
- these approximations are only valid for **large ℓ**

Proposition (Mercier & Daudin 2001, Nuel 2006)

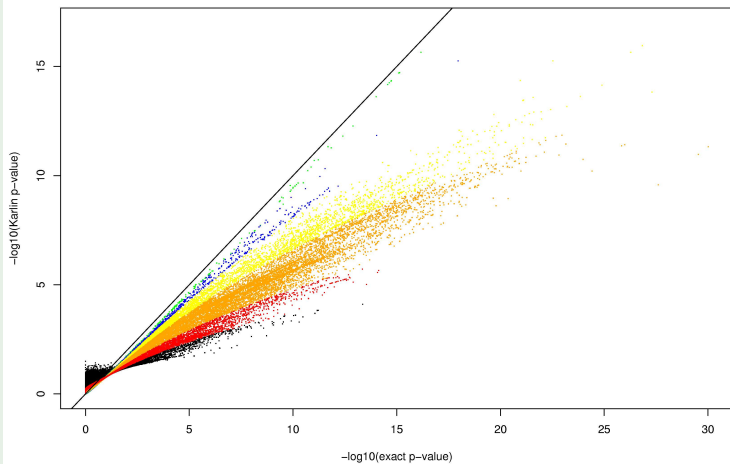
One can use **FMCI** to compute a exact p-value for a Mm model with a complexity

$$O\left(10^n \times k^{m+1} \times H_{\text{obs}} \times \log \ell\right)$$

where n is the number of digits in the scoring function and k the alphabet size.

Gumble approximations vs exact p-values (1)

Example (Kyte-Doolittle hydrophobic scale on SwissProt)



Gumble approximations vs exact p-values (2)

- asymptotic approximations could be **completely false** (for 99.5% of the data in our example)
 - exact computations take **much more time** (in our example 20 exact p-values computed per second)
 - there is a program called **pLocalScore** allowing to perform both the asymptotic approximations and the exact computations
- ⇒ **advice**: use exact computation rather than asymptotic approximations **as often as possible**

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 Local Score of One Sequence
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 Alignment of Sequences
 - **Global Alignment**
 - Local Alignment
 - Heuristics

How to compare sequences ?

Problem

How to compare two biological sequences (DNA, proteins) $X = X_1 \dots X_p$ and $Y = Y_1 \dots Y_q$? Should we:

- compare their respective lengths ?
- compare their compositions (letters, word of size 2, 3, ...)
- look for repetitions ?
- ...

⇒ What about their **proximity** in the evolution process ?

Mutation and Indel

Definition

During the evolution of a biological sequence, a letter that changes is called a **mutation** and we called **indel** either the insertion of a letter or the deletion of a letter.

Example (a DNA sequence)

a	c	c	g	t	t	a	c	a	a	g	a	c	a		
a	c	c	g	t	t	a	c	a	a	g	a	c	a		
a	c	c	g	t	t	a	c	a	a	g	a	c	a		
									•						
a	c	•	g	t	t	a	c	a	t	g	a	c	a		
a	c		g	t	t	a	•	g	c	a	t	g	a	c	a

What is an alignment ?

Example (Two DNA sequences)

$X = a c g t a g c a t g a c a$

$Y = a c c g t a c a a g c a$

We denote by A a **common ancestral sequence**:

A	a	c	c	g	t	t	a	c	a	a	g	a	c	a	
X	a	c		g	t		a	g	c	a	t	g	a	c	a
Y	a	c	c	g	t		a		c	a	a	g		c	a

here is the **alignement** we get:

\tilde{X}	a	c	-	g	t	-	a	g	c	a	t	g	a	c	a
\tilde{Y}	a	c	c	g	t	-	a	-	c	a	a	g	-	c	a

Scoring Alignments

Definition (Score of an Alignment)

Using the **scoring function** $S : \mathcal{A} \cup \{-\} \times \mathcal{A} \cup \{-\} \rightarrow \mathbb{R}$ we define the **score of an alignment** as the sum of the scoring function over all the columns of the alignment.

Example ($S(\text{match}) = +1$ $S(\text{mismatch}) = -1$ $S(\text{gap}) = -2$)

the first alignment scores $8 \times 1 - 0 \times 1 - 9 \times 2 = -10$

\tilde{X}	a	c	-	g	t	a	-	-	-	g	c	a	t	g	a	c	a
\tilde{Y}	a	c	c	g	t	a	c	a	a	g	c	a	-	-	-	-	-

and the second alignment scores $10 \times 1 - 1 \times 1 - 2 \times 2 = 5$

\tilde{X}	a	c	-	g	t	a	g	c	a	t	g	a	c	a
\tilde{Y}	a	c	c	g	t	a	c	a	a	g	c	a	-	-

How to find the best alignment ?

Brut

Scores individually **all possible alignments** and pick up the best one.

⇒ **How many** possible alignments ?

Proposition

If $N(p, q)$ is the number of alignments between two sequences of lengths p and q we have the following **recurrence relation**:

$$N(1, q) = 2q + 1 \quad N(p, 1) = 2p + 1$$

$$N(p, q) = N(p, q - 1) + N(p - 1, q) + N(p - 1, q - 1)$$

for all $p, q \geq 1$.

Number of Alignments

Example

	$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$	$q = 6$	$q = 7$
$p = 1$	3	5	7	9	11	13	15
$p = 2$	5	13	25	41	61	85	113
$p = 3$	7	25	63	129	231	377	575
$p = 4$	9	41	129	321	681	1289	2241
$p = 5$	11	61	231	681	1683	3653	7183

Approximation

Idea: $N(p, q) \sim \rho^{p+q}$ we hence get
 $\rho^2 - 2\rho - 1 = 0 \Rightarrow \rho = 1 + \sqrt{2}$

this gives us:

$$N(20, 20) \simeq 10^{15} \quad N(100, 100) \simeq 10^{76} \quad N(1000, 1000) \simeq 10^{764}$$

Dynamic Programming

Needleman and Wunsch (1970)

We denote by $B(i, j)$ the best score of an alignment of $X_1 \dots X_i$ and $Y_1 \dots Y_j$ and we get

1: $B(0, 0) = 0$

2: $B(i, 0) = \sum_{k=1}^i S(X_k, -)$ and $B(0, j) = \sum_{k=1}^j S(-, Y_k)$

3: **for** $i = 1 \dots p$ **do**

4: **for** $i = 1 \dots q$ **do**

5:

$$B(i, j) = \max \begin{cases} B(i-1, j-1) + S(X_i, Y_j) \\ B(i-1, j) + S(X_i, -) \\ B(i, j-1) + S(-, Y_j) \end{cases}$$

6: return $S(p, q)$ and use a **traceback** to find the alignment

Example

Example ($X = \text{gcgacgtgcaag}$ $Y = \text{aggcacgca}$ $+3, -1, -2$)

	-	a	g	g	c	a	c	g	c	a			
-	0	-2	-4	-6	-8	-10	-12	-14	-16	-18			
g	-2	-1	1	-1	-3	-5	-7	-9	-11	-13			
c	-4	-3	-1	0	2	0	-2	-4	-6	-8			
g	-6	-5	0	2	0	1	-1	1	-1	-3			
a	-8	-3	-2	0	1	3	1	-1	0	2			
c	-10	-5	-4	-2	3	1	6	4	2	0			
g	-12	-7	-2	-1	1	2	4	9	7	5			
t	-14	-9	-4	-3	-1	0	2	7	8	6			
g	-16	-11	-6	-1	-3	-2	0	5	6	7			
c	-18	-13	-8	-3	2	0	1	3	8	6			
a	-20	-15	-10	-5	0	5	3	1	6	11			
a	-22	-17	-12	-7	-2	3	4	2	4	9			
g	-24	-19	-14	-9	-4	1	2	7	5	7			
a	g	g	c	-	a	c	g	-	-	c	a	-	-
-	g	-	c	g	a	c	g	t	g	c	a	a	g

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 Local Score of One Sequence
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 Alignment of Sequences
 - Global Alignment
 - **Local Alignment**
 - Heuristics

Limits of Global Alignment

Remarks

Global alignment is **not suitable** to detect:

- **overlaps** (ex: the end of X and beginning of Y are the same)
- **insertions** (ex: X is included in Y)
- all kind of **local similarities**

Definition (local alignment)

The **local alignment** of two sequences is a global alignment of **two segments** of these sequences.

$$H = \max_{1 \leq i \leq i' \leq p, 1 \leq j \leq j' \leq q} B(X_i \dots X_{i'}, Y_j \dots Y_{j'})$$

Algorithm

Smith and Waterman (1981)

We denote by $H(i, j)$ the best score of local alignment of $X_1 \dots X_i$ and $Y_1 \dots Y_j$ and we get

1: $H(i, 0) = H(0, j) = 0$ for all i, j

2: **for** $i = 1 \dots p$ **do**

3: **for** $j = 1 \dots q$ **do**

4:

$$H(i, j) = \max \begin{cases} H(i-1, j-1) + S(X_i, Y_j) \\ H(i-1, j) + S(X_i, -) \\ H(i, j-1) + S(-, Y_j) \\ 0 \end{cases}$$

5: return $\max_{i,j} H(i, j)$ and use a **traceback** to find the alignment

Example

Example ($X = \text{gcgacgtgcaag}$ $Y = \text{aggcacgca} +3, -1, -2$)

	-	a	g	g	c	a	c	g	c	a	
-	0	0	0	0	0	0	0	0	0	0	
g	0	0	3	3	1	0	0	3	1	0	
c	0	0	1	2	6	4	3	1	6	4	
g	0	0	3	4	4	5	3	6	4	5	
a	0	3	1	2	3	7	5	4	5	7	
c	0	1	2	0	5	5	10	8	7	5	
g	0	0	4	5	3	4	8	13	11	9	
t	0	0	2	3	4	2	6	11	12	10	
g	0	0	3	5	3	3	4	9	10	11	
c	0	0	1	3	8	6	6	7	12	10	
a	0	3	1	1	6	11	9	7	10	15	
a	0	3	2	0	4	9	10	8	8	13	
g	0	1	6	5	3	7	8	13	11	11	
		g	c	g	a	c	g	t	g	c	a
		g	c	-	a	c	-	-	g	c	a

Outline

- 1 Dynamic Programming
 - The Principle
 - Application: Longest Common Subsequence
 - Application: Viterbi
- 2 Local Score of One Sequence
 - Motivation and Notation
 - Computing the Local Score
 - Assessing the Significance
- 3 Alignment of Sequences
 - Global Alignment
 - Local Alignment
 - Heuristics

Complexity

Remarks

- local alignment with Smith & Waterman has a complexity $O(p \times q)$ both in time and space
 - a slightly more complex algorithm has a complexity $O(p \times q)$ in time but only $O(\min(p, q))$ in space
- ⇒ **too slow** for massive sequences comparisons (complete genomes, databases, ...) we need **heuristics** !

BLAST, FASTA, ...

Heuristic to compare a query to a database

- 1: **pre-process** the database to speed up hit search (once per database) and compute λ and K (for Gumble approximations)
- 2: scan the database for **query hits** (ex: the same word of length 7 appears both in the database and in the query)
- 3: try to **combine hits** together
- 4: perform a **Smith and Waterman** algorithm in the vicinity of the hits
- 5: asses the **significance** of the result with the Gumble approximation

Summary

Dynamic programming

- need of **recurrence relation**
- **trade** memory for speed
- **many** applications

Local score of one sequence

- an elegant replacement for **sliding window** methods
- strong connexion with **HMM**
- **Gumbel approximations** are not much reliable

Alignment of sequences

- suitable to **compare** biological sequences
- **choice** of the scoring function
- only **heuristics** are used in practice

Summary

Dynamic programming

- need of **recurrence relation**
- **trade** memory for speed
- **many** applications

Local score of one sequence

- an elegant replacement for **sliding window** methods
- strong connexion with **HMM**
- **Gumbel approximations** are not much reliable

Alignment of sequences

- suitable to **compare** biological sequences
- **choice** of the scoring function
- only **heuristics** are used in practice

Summary

Dynamic programming

- need of **recurrence relation**
- **trade** memory for speed
- **many** applications

Local score of one sequence

- an elegant replacement for **sliding window** methods
- strong connexion with **HMM**
- **Gumbel approximations** are not much reliable

Alignment of sequences

- suitable to **compare** biological sequences
- **choice** of the scoring function
- only **heuristics** are used in practice