

Computational Biology

Approximate Patter Matching (Dyn. Prg, Alignments&Co)

Department of Mathematics
Stockholm University

Dynamic Programming is . . .

- . . . a general, powerful algorithm design technique for solving optimization problems.
- . . . a type of “very smart” exhaustive search that can be applied when the problem can be “subdivided” into overlapping subproblems.
- . . . solves problems by combining the solutions to subproblems
- . . . computes the value of an optimal solution first. Optionally, the optimal solution can be constructed from computed information ([backtracking](#)).

Example: Fibonacci Numbers...

... are recursively defined:

- ▶ $f(1) = f(2) = 1$
- ▶ $f(n) = f(n-1) + f(n-2), n > 2.$

naive recursive way:

$F(\text{positive integer } n)$

- 1: **if** $n \leq 2$ **then** $f = 1$
- 2: **else**
- 3: $f = F(n-1) + F(n-2)$
- 4: **return** f

recursive way with memo:

$F(\text{positive integer } n)$

- 1: **if** $\text{memo}[n] \neq \text{NIL}$ **then**
- 2: **return** $\text{memo}[n]$
- 3: **if** $n \leq 2$ **then** $f = 1$
- 4: **else**
- 5: $f = F(n-1) + F(n-2)$
- 6: $\text{memo}[n] = f$
- 7: **return** f

Which algorithm is more efficient and why? WHITEBOARD

Example: Fibonacci Numbers...

... are recursively defined:

- ▶ $f(1) = f(2) = 1$
- ▶ $f(n) = f(n-1) + f(n-2), n > 2.$

naive recursive way:

$F(\text{positive integer } n)$

- 1: **if** $n \leq 2$ **then** $f = 1$
- 2: **else**
- 3: $f = F(n-1) + F(n-2)$
- 4: **return** f

recursive way with memo:

$F(\text{positive integer } n)$

- 1: **if** $\text{memo}[n] \neq \text{NIL}$ **then**
- 2: **return** $\text{memo}[n]$
- 3: **if** $n \leq 2$ **then** $f = 1$
- 4: **else**
- 5: $f = F(n-1) + F(n-2)$
- 6: $\text{memo}[n] = f$
- 7: **return** f

Which algorithm is more efficient and why? WHITEBOARD

Example: Fibonacci Numbers...

... are recursively defined:

- ▶ $f(1) = f(2) = 1$
- ▶ $f(n) = f(n-1) + f(n-2), n > 2.$

naive recursive way:

F (positive integer n)

- 1: **if** $n \leq 2$ **then** $f = 1$
- 2: **else**
- 3: $f = F(n-1) + F(n-2)$
- 4: **return** f

recursive way with memo:

F (positive integer n)

- 1: **if** $\text{memo}[n] \neq \text{NIL}$ **then**
- 2: **return** $\text{memo}[n]$
- 3: **if** $n \leq 2$ **then** $f = 1$
- 4: **else**
- 5: $f = F(n-1) + F(n-2)$
- 6: $\text{memo}[n] = f$
- 7: **return** f

Which algorithm is more efficient and why? WHITEBOARD

Example: Fibonacci Numbers...

... are recursively defined:

- ▶ $f(1) = f(2) = 1$
- ▶ $f(n) = f(n-1) + f(n-2), n > 2.$

naive recursive way:

F (positive integer n)

- 1: **if** $n \leq 2$ **then** $f = 1$
- 2: **else**
- 3: $f = F(n-1) + F(n-2)$
- 4: **return** f

recursive way with memo:

F (positive integer n)

- 1: **if** $\text{memo}[n] \neq \text{NIL}$ **then**
- 2: **return** $\text{memo}[n]$
- 3: **if** $n \leq 2$ **then** $f = 1$
- 4: **else**
- 5: $f = F(n-1) + F(n-2)$
- 6: $\text{memo}[n] = f$
- 7: **return** f

Which algorithm is more efficient and why? **WHITEBOARD**

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (WHITEBOARD)

Longest common subsequence (LCS)

Classical problem in bioinformatics is to understand how “close” to genes or genomes are. There are several ways to address this problem. A simple approach is the “Longest common subsequence” problem.

- ▶ String $X = x_1x_2 \dots x_m$
- ▶ $Z = z_1z_2 \dots z_k$ is subsequence of X , if there are indices $i_1, i_2, \dots, i_k \in \{1, \dots, m\}$ such that $i_1 < i_2 < \dots < i_k$ and $z_j = x_{i_j}$
E.g. $Z = BCDB$ is subsequence of $X = ABCBDAB$
 $Z \neq X[i..j]$ may hold!
- ▶ A subsequence Z of X and Y is a common subsequence of X and Y

Aim: Find longest subsequence of X and Y .

Solution: via Dynamic Programming (**WHITEBOARD**)

Longest common subsequence (LCS) (WHITEBOARD)

LCS(strings X, Y)

```
1: m= X.length, n= Y.length
2: Let b[1 ... m, 1 ... n] be new array
3: Let c[0 ... m; 0 ... n] be new array
4: for i = 1 ... m do c[i, 0] = 0
5: for j = 0 ... n do c[0, j] = 0
6: for i = 1 ... m do
7:   for j = 1 ... n do
8:     if xi = yj then
9:       c[i, j] = c[i - 1, j - 1] + 1
10:      b[i, j] = "↖"
11:     else if c[i - 1, j] ≥ c[i, j - 1] then
12:       c[i, j] = c[i - 1, j]
13:       b[i, j] = "↑"
14:     else
15:       c[i, j] = c[i, j - 1]
16:       b[i, j] = "←"
17: return c and b
```

PRINT_LCS(b, X, i, j)

// Initial call PRINT_LCS(b, X, m, n)

```
1: if i = 0 or j = 0 then return
2: if b[i, j] = "↖" then
3:   PRINT_LCS(b, X, i - 1, j - 1)
4:   print xi
5: else if b[i, j] = "↑" then
6:   PRINT_LCS(b, X, i - 1, j)
7: else
8:   PRINT_LCS(b, X, i, j - 1)
```

Theorem

4.3 LCS() and PRINT_LCS() correctly returns length and LCS of two strings $X = x_1 \dots x_n$ and $Y = y_1 \dots y_m$ in $O(mn)$ time.

"Non-exact" matching

Aim:

Compare strings to score/evaluate the (dis)similarity between them.

"Non-exact" matching arises in many fields:

- ▶ Molecular biology
- ▶ Inexact text matching (e.g. spell checkers; web page search)
- ▶ Speech recognition

Biology:

In biomolecular sequences (DNA, RNA, Proteins) high sequence similarity often implies significant functional or structural similarity.

Important:

similar function \nrightarrow similar structure \nrightarrow similar sequences

How to measure "sequence similarity"?

Naive/simple ways:

- ▶ Hamming distance (board)
- ▶ LCS (board)
- ▶ ..

Edit Distance

Edit Operations:

- ▶ Insertion of character
- ▶ Deletion of character
- ▶ Replacement of one character by some other one

Edit Distance = Min. Nr. of Edit Operations to transform string u to string v
(equivalent transform string v to string u)

D	M	M	R	M	M	I	
<hr/>							(M = Match)
w	r	i	t	e	r	-	
-	r	i	d	e	r	s	

Edit Script = string over alphabet $\{I, D, R, M\}$ that describes transformation from u to v .

Edit Distance Problem: For two strings compute edit distance and optimal edit script.

Example

$u = \text{TGCATAT}$ $v = \text{ATCCGAT}$

$u = \text{TGCATAT}$ $\xrightarrow{\text{del. last T}}$ TGCATA $\xrightarrow{\text{del. last A}}$ TGCAT $\xrightarrow{\text{add A 1.pos}}$ ATGCAT $\xrightarrow{\text{repl. G by C 3.pos}}$
 ATCCAT $\xrightarrow{\text{insert G 5.pos}}$ $\text{ATCCGAT} = v$ Edit Distance ≤ 5

$u = \text{TGCATAT}$ $\xrightarrow{\text{ins. A 1.pos}}$ ATGCATAT $\xrightarrow{\text{del. T 6.pos}}$ ATGCATAT $\xrightarrow{\text{repl. A by G 5.pos}}$
 ATCCGAT $\xrightarrow{\text{repl. G by C 3.pos}}$ $\text{ATCCGAT} = v$

Edit Distance ≤ 4

(How to find OPTIMAL one?)

Example

$u = \text{TGCATAT}$ $v = \text{ATCCGAT}$

$u = \text{TGCATAT}$ $\xrightarrow{\text{del. last T}}$ TGCATA $\xrightarrow{\text{del. last A}}$ TGCAT $\xrightarrow{\text{add A 1.pos}}$ ATGCAT $\xrightarrow{\text{repl. G by C 3.pos}}$
 ATCCAT $\xrightarrow{\text{insert G 5.pos}}$ $\text{ATCCGAT} = v$ Edit Distance ≤ 5

$u = \text{TGCATAT}$ $\xrightarrow{\text{ins. A 1.pos}}$ ATGCATAT $\xrightarrow{\text{del. T 6.pos}}$ ATGCAT $\xrightarrow{\text{repl. A by G 5.pos}}$
 ATCCGAT $\xrightarrow{\text{repl. G by C 3.pos}}$ $\text{ATCCGAT} = v$

Edit Distance ≤ 4

(How to find OPTIMAL one?)

Example

$u = \text{TGCATAT}$ $v = \text{ATCCGAT}$

$u = \text{TGCATAT}$ $\xrightarrow{\text{del. last T}}$ TGCATA $\xrightarrow{\text{del. last A}}$ TGCAT $\xrightarrow{\text{add A 1.pos}}$ ATGCAT $\xrightarrow{\text{repl. G by C 3.pos}}$
 ATCCAT $\xrightarrow{\text{insert G 5.pos}}$ $\text{ATCCGAT} = v$ Edit Distance ≤ 5

$u = \text{TGCATAT}$ $\xrightarrow{\text{ins. A 1.pos}}$ ATGCATAT $\xrightarrow{\text{del. T 6.pos}}$ ATGCAT $\xrightarrow{\text{repl. A by G 5.pos}}$
 ATCCGAT $\xrightarrow{\text{repl. G by C 3.pos}}$ $\text{ATCCGAT} = v$

Edit Distance ≤ 4

(How to find OPTIMAL one?)

Example

$u = \text{TGCATAT}$ $v = \text{ATCCGAT}$

$u = \text{TGCATAT}$ $\xrightarrow{\text{del. last T}}$ TGCATA $\xrightarrow{\text{del. last A}}$ TGCAT $\xrightarrow{\text{add A 1.pos}}$ ATGCAT $\xrightarrow{\text{repl. G by C 3.pos}}$
 ATCCAT $\xrightarrow{\text{insert G 5.pos}}$ $\text{ATCCGAT} = v$ Edit Distance ≤ 5

$u = \text{TGCATAT}$ $\xrightarrow{\text{ins. A 1.pos}}$ ATGCATAT $\xrightarrow{\text{del. T 6.pos}}$ ATGCAT $\xrightarrow{\text{repl. A by G 5.pos}}$
 ATCCGAT $\xrightarrow{\text{repl. G by C 3.pos}}$ $\text{ATCCGAT} = v$

Edit Distance ≤ 4

(How to find OPTIMAL one?)

Example

$u = \text{TGCATAT}$ $v = \text{ATCCGAT}$

$u = \text{TGCATAT}$ $\xrightarrow{\text{del. last T}}$ TGCATA $\xrightarrow{\text{del. last A}}$ TGCAT $\xrightarrow{\text{add A 1.pos}}$ ATGCAT $\xrightarrow{\text{repl. G by C 3.pos}}$
 ATCCAT $\xrightarrow{\text{insert G 5.pos}}$ $\text{ATCCGAT} = v$ Edit Distance ≤ 5

$u = \text{TGCATAT}$ $\xrightarrow{\text{ins. A 1.pos}}$ ATGCATAT $\xrightarrow{\text{del. T 6.pos}}$ ATGCAT $\xrightarrow{\text{repl. A by G 5.pos}}$
 ATGCGTAT $\xrightarrow{\text{repl. G by C 3.pos}}$ $\text{ATCCGAT} = v$

Edit Distance ≤ 4

(How to find OPTIMAL one?)

(global pairwise) Alignment

Alternative way to edit script: Alignment

For two strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ an *alignment* \mathcal{A} is a matrix with two rows and entries $\mathcal{A}[i, j]$ that are characters from Alphabet Σ (e.g. $\Sigma = \{A, C, G, T\}$) or a *gap* “-” s.t.

- ▶ 1st row = u after deleting all gaps
- ▶ 2st row = v after deleting all gaps
- ▶ in no column are two gaps

w	r	i	t	e	r	-	-	T	G	C	A	T	A	T
-	r	i	d	e	r	s	A	T	C	C	G	-	A	T

Cost-Function $\delta : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$

Unit-Cost-Function $\delta(a, b) = 1$ if $a \neq b$

$\delta(a, b) = 0$ if $a = b$

Alignment Costs $\delta(\mathcal{A}) := \sum_{i=1} \delta(a_i, b_i)$

(global pairwise) Alignment

Alternative way to edit script: Alignment

For two strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ an *alignment* \mathcal{A} is a matrix with two rows and entries $\mathcal{A}[i, j]$ that are characters from Alphabet Σ (e.g. $\Sigma = \{A, C, G, T\}$) or a *gap* “-” s.t.

- ▶ 1st row = u after deleting all gaps
- ▶ 2nd row = v after deleting all gaps
- ▶ in no column are two gaps

w	r	i	t	e	r	-	-	T	G	C	A	T	A	T
-	r	i	d	e	r	s	A	T	C	C	G	-	A	T

Cost-Function $\delta : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$

Unit-Cost-Function $\delta(a, b) = 1$ if $a \neq b$

$\delta(a, b) = 0$ if $a = b$

Alignment Costs $\delta(\mathcal{A}) := \sum_{i=1} \delta(a_i, b_i)$

(global pairwise) Alignment

Alternative way to edit script: Alignment

For two strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ an *alignment* \mathcal{A} is a matrix with two rows and entries $\mathcal{A}[i, j]$ that are characters from Alphabet Σ (e.g. $\Sigma = \{A, C, G, T\}$) or a *gap* “-” s.t.

- ▶ 1st row = u after deleting all gaps
- ▶ 2nd row = v after deleting all gaps
- ▶ in no column are two gaps

w	r	i	t	e	r	-	-	T	G	C	A	T	A	T
-	r	i	d	e	r	s	A	T	C	C	G	-	A	T

Cost-Function $\delta : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$

Unit-Cost-Function $\delta(a, b) = 1$ if $a \neq b$

$\delta(a, b) = 0$ if $a = b$

Alignment Costs $\delta(\mathcal{A}) := \sum_{i=1} \delta(a_i, b_i)$

(global pairwise) Alignment

Alternative way to edit script: Alignment

For two strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$ an *alignment* \mathcal{A} is a matrix with two rows and entries $\mathcal{A}[i, j]$ that are characters from Alphabet Σ (e.g. $\Sigma = \{A, C, G, T\}$) or a *gap* “-” s.t.

- ▶ 1st row = u after deleting all gaps
- ▶ 2st row = v after deleting all gaps
- ▶ in no column are two gaps

w	r	i	t	e	r	-	-	T	G	C	A	T	A	T
-	r	i	d	e	r	s	A	T	C	C	G	-	A	T

Cost-Function $\delta : \Sigma \cup \{-\} \times \Sigma \cup \{-\} \rightarrow \mathbb{R}$

Unit-Cost-Function $\delta(a, b) = 1$ if $a \neq b$

$\delta(a, b) = 0$ if $a = b$

Alignment Costs $\delta(\mathcal{A}) := \sum_{i=1} \delta(a_i, b_i)$

Lemma 1

Edit Distance of two strings u, v equals the min. alignments costs $\delta(\mathcal{A})$ between u and v with unit-cost function.

How to compute Edit Distance? **Dynamic Programming!**

Recurrence Function D (Needleman-Wunsch Algorithm)

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Assume $D[i, j]$ are the costs for an optimal alignment of substrings $u_1 \dots u_i$ and $v_1 \dots v_j$, $1 \leq i \leq m$, $1 \leq j \leq n$

$i = 0$: alignment empty string ϵ and $v_1 \dots v_j$

$j = 0$: alignment $u_1 \dots u_i$ and empty string ϵ

Init: $D[i, 0] = D[i - 1, 0] + \delta(s_i, -)$; $D[0, j] = D[0, j - 1] + \delta(-, t_j)$, $i, j \geq 0$;

Compute

$$D[i, j] = \min \begin{cases} D[i - 1, j] & + \delta(u_i, -) \\ D[i - 1, j - 1] & + \delta(u_i, v_j) \\ D[i, j - 1] & + \delta(-, v_j) \end{cases}$$

(if $\delta =$ unit-cost-function, then $D[i, 0] = i$ and $D[0, j] = j$ for $i, j \geq 1$)

Lemma 2

$D[m, n]$ = cost of optimal alignment between u and v .

Recurrence Function D (Needleman-Wunsch Algorithm)

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Assume $D[i, j]$ are the costs for an optimal alignment of substrings $u_1 \dots u_i$ and $v_1 \dots v_j$, $1 \leq i \leq m$, $1 \leq j \leq n$

$i = 0$: alignment empty string ϵ and $v_1 \dots v_j$

$j = 0$: alignment $u_1 \dots u_i$ and empty string ϵ

Init: $D[i, 0] = D[i - 1, 0] + \delta(s_i, -)$; $D[0, j] = D[0, j - 1] + \delta(-, t_j)$, $i, j \geq 0$;

Compute

$$D[i, j] = \min \begin{cases} D[i - 1, j] & + \delta(u_i, -) \\ D[i - 1, j - 1] & + \delta(u_i, v_j) \\ D[i, j - 1] & + \delta(-, v_j) \end{cases}$$

(if $\delta =$ unit-cost-function, then $D[i, 0] = i$ and $D[0, j] = j$ for $i, j \geq 1$)

Lemma 2

$D[m, n]$ = cost of optimal alignment between u and v .

Recurrence Function D (Needleman-Wunsch Algorithm)

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Assume $D[i, j]$ are the costs for an optimal alignment of substrings $u_1 \dots u_i$ and $v_1 \dots v_j$, $1 \leq i \leq m$, $1 \leq j \leq n$

$i = 0$: alignment empty string ϵ and $v_1 \dots v_j$

$j = 0$: alignment $u_1 \dots u_i$ and empty string ϵ

Init: $D[i, 0] = D[i - 1, 0] + \delta(s_i, -)$; $D[0, j] = D[0, j - 1] + \delta(-, t_j)$, $i, j \geq 0$;

Compute

$$D[i, j] = \min \begin{cases} D[i - 1, j] & + \delta(u_i, -) \\ D[i - 1, j - 1] & + \delta(u_i, v_j) \\ D[i, j - 1] & + \delta(-, v_j) \end{cases}$$

(if $\delta =$ unit-cost-function, then $D[i, 0] = i$ and $D[0, j] = j$ for $i, j \geq 1$)

Lemma 2

$D[m, n]$ = cost of optimal alignment between u and v .

Recurrence Function D (Needleman-Wunsch Algorithm)

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Assume $D[i, j]$ are the costs for an optimal alignment of substrings $u_1 \dots u_i$ and $v_1 \dots v_j$, $1 \leq i \leq m$, $1 \leq j \leq n$

$i = 0$: alignment empty string ϵ and $v_1 \dots v_j$

$j = 0$: alignment $u_1 \dots u_i$ and empty string ϵ

Init: $D[i, 0] = D[i - 1, 0] + \delta(s_i, -)$; $D[0, j] = D[0, j - 1] + \delta(-, t_j)$, $i, j \geq 0$;

Compute

$$D[i, j] = \min \begin{cases} D[i - 1, j] & + \delta(u_i, -) \\ D[i - 1, j - 1] & + \delta(u_i, v_j) \\ D[i, j - 1] & + \delta(-, v_j) \end{cases}$$

(if $\delta =$ unit-cost-function, then $D[i, 0] = i$ and $D[0, j] = j$ for $i, j \geq 1$)

Lemma 2

$D[m, n]$ = cost of optimal alignment between u and v .

Recurrence Function D (Needleman-Wunsch Algorithm)

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Assume $D[i, j]$ are the costs for an optimal alignment of substrings $u_1 \dots u_i$ and $v_1 \dots v_j$, $1 \leq i \leq m$, $1 \leq j \leq n$

$i = 0$: alignment empty string ϵ and $v_1 \dots v_j$

$j = 0$: alignment $u_1 \dots u_i$ and empty string ϵ

Init: $D[i, 0] = D[i - 1, 0] + \delta(s_i, -)$; $D[0, j] = D[0, j - 1] + \delta(-, t_j)$, $i, j \geq 0$;

Compute

$$D[i, j] = \min \begin{cases} D[i - 1, j] & + \delta(u_i, -) \\ D[i - 1, j - 1] & + \delta(u_i, v_j) \\ D[i, j - 1] & + \delta(-, v_j) \end{cases}$$

(if $\delta =$ unit-cost-function, then $D[i, 0] = i$ and $D[0, j] = j$ for $i, j \geq 1$)

Lemma 2

$D[m, n]$ = cost of optimal alignment between u and v .

Backtracing

Given the strings $u = u_1 \dots u_m$ and $v = v_1 \dots v_n$

Tracematrix is an $m \times n$ matrix with $T[i, j] \subseteq \{\leftarrow, \swarrow, \uparrow\}$.

Init: $T[0, 0] = \emptyset$, $T[i, 0] = \uparrow$, $T[0, j] = \leftarrow$ for $1 \leq i \leq m$, $1 \leq j \leq n$

Set: $\begin{array}{ll} \uparrow \in T[i, j] & \text{if } D[i-1, j] + \delta(u_i, -) \\ \swarrow \in T[i, j] & \text{if } D[i-1, j-1] + \delta(u_i, v_j) \\ \leftarrow \in T[i, j] & \text{if } D[i, j-1] + \delta(-, v_j) \end{array}$

Runtime: $O(mn)$

Alignment with variable Gap-Costs

ACCGTCTGCT ACCGTCTGCT $\delta(\mathcal{A}) = 5$
A-C--C-G-T ACCGT-----

This contradicts “biological intuition”:

Insertion of gap of length k is “evolutionary simpler to realize” than insertion of k gaps of length 1.

gap penalty function $g : \mathbb{N} \rightarrow \mathbb{R}$

$g(k)$ is penalty for inserting a gap of length k .

we need:

$$g(k + l) \leq g(k) + g(l),$$

as otherwise it might be better to insert 2 gaps of length k and l than one gap of length $k + l$.

Alignment with variable Gap-Costs (Smith-Waterman-Alg.)

Init: $D[0, 0] = 0$; $D[0, k] = D[k, 0] = g(k)$, $k \geq 1$;

$$D[i, j] = \min \begin{cases} D[i-1, j-1] & + \delta(u_i, v_j) \\ \min_{1 \leq k \leq i} D[i-k, j] & + g(k) \\ \min_{1 \leq k \leq j} D[i, j-k] & + g(k) \end{cases}$$

Tracematrix is an $m \times n$ matrix with $T[i, j] \subseteq \{\leftarrow_k, \nearrow, \uparrow_k, k \in \mathbb{N}\}$

Distance VS Scoring Function

Note: Instead of using a distance matrix D we can use a Similarity/Scoring Matrix S and maximize.

Init: $S[i, 0] = -i * gap - cost$; $S[0, j] = -j * gap - cost$; for $i, j \geq 0$;

Compute

$$S[i, j] = \max \begin{cases} S[i-1, j] & + \delta(u_i, -) \\ S[i-1, j-1] & + \delta(u_i, v_j) \\ S[i, j-1] & + \delta(-, v_j) \end{cases}$$

with e.g.

$$\delta(a, b) = \begin{cases} 1 & \text{if } a = b \\ -1 & \text{if } a \neq b \text{ and } a, b \neq - \\ -3 & \text{else (gap-costs)} \end{cases}$$

Local vs Global Alignment

Needleman-Wunsch computes a *global optimal Alignment*

NW reasonable if sequences have almost same length

If sequences have quite different length, then the sequences are “shredded”:

```
R-----LCPMNLGCSQ-----KY
```

```
RCGEQGSNMECPNNLC-CSQYGYCGMGGDYCGKGCQNGACWTSKR
```

Reason: gaps are penalized equally on each position

Reasonable: less penalization of gaps at end and beginning

Local Alignment: find best alignment of two substrings of two sequences
(Smith-Waterman-Algorithm)

Local vs Global Alignment

Dynamic programming table for global alignment between sequences X and Y. The value 12 is circled in red, indicating the maximum score for a local alignment.

		Y														
		ε	T	A	T	A	T	G	C	G	G	C	G	T	T	T
X	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
	G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
	T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
	A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
	T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
	G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
	C	0	0	0	0	0	0	2	10	4	0	4	0	0	0	0
	T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
	G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
	G	0	0	0	0	0	0	2	0	2	8	4	4	0	0	0
	C	0	0	0	0	0	0	0	4	0	2	10	4	0	0	0
	G	0	0	0	0	0	0	2	0	6	2	4	12	6	0	0
	C	0	0	0	0	0	0	0	4	0	2	4	6	8	2	0
	T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
	A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

Dynamic programming table for local alignment between sequences X and Y. A red arrow traces the path of the maximum local alignment from the bottom-right cell (12) back to the start of the alignment (0,0).

		Y														
		ε	T	A	T	A	T	G	C	G	G	C	G	T	T	T
X	ε	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	G	0	0	0	0	0	0	2	0	2	2	0	2	0	0	0
	G	0	0	0	0	0	0	2	0	2	4	0	2	0	0	0
	T	0	2	0	2	0	2	0	0	0	0	0	0	4	2	2
	A	0	0	4	0	4	0	0	0	0	0	0	0	0	0	0
	T	0	2	0	6	0	6	0	0	0	0	0	0	2	2	2
	G	0	0	0	0	2	0	8	2	2	2	0	2	0	0	0
	C	0	0	0	0	0	0	2	10	4	0	4	0	0	0	0
	T	0	2	0	2	0	2	0	4	6	0	0	0	2	2	2
	G	0	0	0	0	0	0	4	0	6	8	2	2	0	0	0
	G	0	0	0	0	0	0	2	0	2	8	4	4	0	0	0
	C	0	0	0	0	0	0	0	4	0	2	10	4	0	0	0
	T	0	2	0	2	0	2	0	0	0	0	0	0	8	10	4
	A	0	0	4	0	4	0	0	0	0	0	0	0	2	4	6

Y TATGC - GGCG
 ||||| ||||
 X TATGCTGGCG

↙ traceback

Example Multiple Alignment

Q5E940_BOVIN	-----M*PREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_HUMAN	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_MOUSE	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_RAT	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_CHICK	-----MPREDRATWKSNYFMKIIQLDDY*PKCFVYVADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_RANSY	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--SALE	76
Q7ZUG3_BRARE	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0 ICTPU	-----MPREDRATWKSNYFLKIIQLDDY*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PALE	76
RLA0_DROME	-----MVRENKAANKAQYFIKVV*ELDFE*PKCFIVGADNVGSKMQQIRMSLRGK-AVVLMGKNTMMRKAIRGHLENN--PQLE	76
RLA0_DICDI	-----MSGAG-SKRKNVFEIETATKLF*FTYDKMIVAEAD*FVGS*SQLOKIRKSLRGI-GAVLMGKNTMIRKVIIRD*LDASK--PELD	75
Q54LP0_DICDI	-----MSGAG-SKRKNVFEIETATKLF*FTYDKMIVAEAD*FVGS*SQLOKIRKSLRGI-GAVLMGKNTMIRKVIIRD*LDASK--PELD	75
RLA0_PLAF8	-----MAKLSKQQKQMYIEKLS*SLIQQYSKILVHV*DNVGS*SNMASVYKSLRGA-AILMGKNTMIRKVIIRD*LDASK--PQLE	76
RLA0_SULAC	-----MIGLAVTTTKKIAKWKVDEVAELTKL*TKHT*IIIANIRGFPADKLHEIRKKLRGK-ADIKVTKNNL*FNIALKNAG---YDLK	79
RLA0_SULTO	-----MRIMAVITQERKIAKWKIEEYKELTKL*REYHT*IIIANIRGFPADKLHDIRKKMRGMAEIKVTKNNL*FIAAKNAG---LDVS	80
RLA0_SULSO	-----MKRLALALQKQKVASW*KEEYKELTKL*IKNSNT*ILIGNL*EGFPADKLHEIRKKLRGK-ATIKVTKNNL*LFKIAAKNAG---IDIE	80
RLA0_AERPE	MSVVS*LVGQMYKREK*PIPEW*KLMLRELE*ELFSK*RRV*VLEADLT*G*P*FVY*OR*YK*KLW*KK-Y*P*MM*V*AK*KR*IL*RAM*KAAGLE---LDDN	86
RLA0_PYRAE	MMLAIGKRRYVRTQ*Y*PARKYKIVSEATE*LLQK*Y*YV*VFL*DL*H*GL*SL*RL*HE*RY*RL*RY-G*V*K*I*IK*P*LFK*IA*F*TK*V*Y*GG---TPAE	85
RLA0_METAC	-----MAEERHHT*E*HP*Q*WK*DE*IE*NI*KE*LI*Q*SH*K*V*G*MG*V*IE*G*IL*AT*K*OK*IR*RD*L*K*DV-A*V*L*K*V*SR*NT*L*LE*RA*LN*Q*L*G---ETIP	78
RLA0_METMA	-----MAEERHHT*E*HP*Q*WK*DE*IE*NI*KE*LI*Q*SH*K*V*G*MG*V*IE*G*IL*AT*K*OK*IR*RD*L*K*DV-A*V*L*K*V*SR*NT*L*LE*RA*LN*Q*L*G---ESIP	78
RLA0_AFCFU	-----MAAVRGS---PPEYKVR*AVEEIKR*MI*SS*Q*Y*V*V*AL*V*SR*NV*P*AG*DM*OK*IR*RE*FR*GK-AEIKVTKNNL*LE*RA*LD*AL*G---GDYL	75
RLA0_METKA	MAVKAKGQPPSGYE*PKVAEWKRR*EYKELTKL*MD*E*Y*EN*V*GL*VD*LE*G*IP*AD*Q*LO*E*TRAK*LR*ERD*TI*RM*SR*NT*LMR*IA*LE*EK*LD*ER--PELE	88
RLA0_METH	-----MAHVAEWK*KEVE*Q*EL*H*DL*IK*G*Y*E*Y*V*G*IAN*LA*DI*P*AR*LO*Q*MR*QT*LR*DS-ALIR*MS*KK*TL*SL*LA*EK*AG*RE*L---ENVY	74
RLA0_METTL	-----MITAESEHK*AP*WK*IE*Y*V*KL*KE*LL*KN*G*Q*V*AL*V*DM*ME*V*P*AR*LO*E*IR*DK*IR-G*TM*IL*KM*SR*NT*L*LE*RA*IK*E*VA*E*ET*GN*PE*FA	82
RLA0_METVA	-----MIDAKSEHK*AP*WK*IE*Y*V*AL*KE*LL*KS*AN*V*IAL*DM*ME*V*P*AR*LO*E*IR*DK*IR-D*QM*IL*KM*SR*NT*L*LE*RA*V*E*E*V*AB*ET*GN*PE*FA	82
RLA0_METJA	-----METKVK*AH*V*AP*WK*IE*Y*V*TL*KL*IK*SK*Y*V*V*AL*V*DM*DV*P*AR*LO*E*IR*DK*IR-D*K*V*L*RM*SR*NT*L*LE*RA*KE*AA*E*LN*NP*KL*LA	81
RLA0_PYRAB	-----MAHVAEWK*KEVE*E*EL*AN*LI*KS*Y*P*V*IAL*V*DV*SS*MP*AY*PL*SQ*MR*LI*RE*NG*GL*LR*V*SR*NT*L*LE*IA*IK*KA*AE*LG*KP*E*LE	77
RLA0_PYRHO	-----MAHVAEWK*KEVE*E*EL*AK*LI*KS*Y*P*V*IAL*V*DV*SS*MP*AY*PL*SQ*MR*LI*RE*NG*GL*LR*V*SR*NT*L*LE*IA*IK*KA*AE*LG*KP*E*LE	77
RLA0_PYRFU	-----MAHVAEWK*KEVE*E*EL*AN*LI*KS*Y*P*V*V*AL*V*DV*SS*MP*AY*PL*SQ*MR*LI*RE*NG*GL*LR*V*SR*NT*L*LE*IA*IK*KA*AE*LG*KP*E*LE	77
RLA0_PYRKO	-----MAHVAEWK*KEVE*E*EL*AN*LI*KS*Y*P*V*V*AL*V*DV*AG*V*P*AY*PL*SK*MR*DK*LE-G*K*ALL*V*SR*NT*L*LE*IA*IK*KA*AE*LG*Q*P*E*LE	76
RLA0_HALMA	MSAESEKRT*E*TP*E*WK*Q*E*E*V*DA*IV*E*MI*E*SY*E*SV*G*V*V*NI*AG*IP*SR*LO*DM*RR*DL*HG*T-A*E*L*V*SR*NT*L*LE*RA*LD*DD*V---DGLE	79
RLA0_HALVO	MSESEVR*QT*E*V*P*Q*WK*RE*E*Y*DL*V*DF*IE*SY*E*SY*G*V*G*V*AG*IP*SR*LO*DM*RR*DL*HG*S-A*AV*RM*SR*NT*L*V*NR*AL*DE*Y*V---DGFE	79
RLA0_HALSA	MSAEEQRT*E*V*P*Q*WK*Q*E*V*AE*LV*DL*LET*DS*Y*G*V*V*NT*G*IP*SR*KL*OD*MR*RR*DL*HG*Q-A*AL*RM*SR*NT*L*V*RA*LE*E*E*G---DGLD	79
RLA0_THEAC	-----MKEV*SQ*Q*KE*LV*NE*IT*RI*KA*SR*V*AI*V*D*AG*IR*E*R*IO*D*IR*G*KN*RG*K-IL*MK*V*IK*K*LL*F*KA*EN*LG*D---EKLS	72
RLA0_THEVO	-----MRKIN*PK*KE*IV*SE*LA*Q*IT*KS*KA*V*AI*V*DI*G*V*R*E*R*O*M*IR*AK*NR*DK-V*K*I*V*V*K*LL*F*KA*LD*S*IN*D---EKLT	72
RLA0_PICTO	-----MTED*P*Q*WK*ID*F*V*KN*LE*NE*IN*SR*K*V*AA*V*S*IK*GL*RN*NE*F*OK*IR*NS*IR*DK-A*RI*K*V*SR*AR*LL*RA*IE*NT*E*G*K---NNIV	72
ruler	1.....10.....20.....30.....40.....50.....60.....70.....80.....90	

First 90 positions of a protein multiple alignment of instances of the acidic ribosomal protein P0 (L10E) from several organisms. (wikipedia)

Now a short overview of three classical (collections) of algorithm that are based on or concerned with Alignments.

- ▶ BLAST
- ▶ Clustal
- ▶ MUSCLE

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = **B**asic **L**ocal **A**lignment **S**earch **T**ool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST = Basic Local Alignment Search Tool

Umbrella term for a collection of the world's most widely used programs for analyzing biological sequence data

- ▶ BLAST is used to compare experimentally determined DNA or protein sequences with sequences already existing in a database.
- ▶ Basic idea: BLAST divides query sequences into short strings and initially only looks for (exact) matches of those strings in database strings. This is afterwards extended to get the entire alignment.
- ▶ very fast local alignment heuristic, but no optimality guarantee
- ▶ output: series of local alignments, i.e. comparisons of pieces of the searched sequence with similar pieces from the database. In addition, BLAST indicates how significant of the hits that have been found.

Databases e.g. for nucleotide sequences (Genbank of NCBI, EMBL, ...) or protein databases (SwissProt, RefSeq, Pfam, ...).

BLAST homepage: blast.ncbi.nlm.nih.gov

Tutorial: digitalworldbiology.com/BLAST

BLAST“Types”

type	query	target
blastn	nucleotide	nucleotide
blastp	protein	protein
blastx	nucleotide (transl)	protein
tblastn	protein	nucleotide (transl)
tblastx	nucleotide (transl)	nucleotide (transl)

Online Example

<https://blast.ncbi.nlm.nih.gov/Blast.cgi> → nucleotide blast → copy&paste →
press button BLAST

>Sequence_experimental

```
GACATTACGGCGACCCAGTCTCCCCCGGTGTTGTCAGTGGGACTGGGCC  
AGACCGCAACCATCACTTGTACGGCCAGTCAAAGCATCTACAGTAACCT  
TGCTTGGTACCAGCAGAGAGAAGGACAGAAGCCCTCTCTCCTGATCTAT  
GCTGCGACAACGCGATACGAAGGAGTCTCCGAGCGATTGAGCGGCAGTG  
GATCAGGGACCAGTTTCACCCTGACAATCAGCAACGTTGAGAATGAGGA  
TGTCGCTGACTATTACTGTGAGATCGCATATTCGATCTACTCCGGTTCC  
GTTGTTTTTCGGTGAAGGAACCAAGCTCAGACTGAGCCGT
```

specific mRNA of a nurse shark.

Clustal is a series of computer programs used in bioinformatics for multiple sequence alignment.

Brief History:

- ▶ Clustal (1981, first version)
- ▶ CLustalW (1994, great improvements)
- ▶ ClustalX (1997, first time with GUI)
- ▶ ClustalΩ (latest standard version, 2011)

Basic idea explained on ClustalW (3 steps for input $\zeta = \text{set } \{S_1, \dots, S_k\}$ of sequences):

W1 Compute for all pairs $S_i, S_j \in \zeta$
a pairwise alignment \implies
pairwise distances $D(S_i, S_j)$

W2 Use distance matrix D to
compute phylogenetic tree T
(via NeighborJoining-method)

W3 Use T to carry out a multiple
alignment

146

Algorithms in Bioinformatics — A Practical Introduction

(a)
 S_1 : PPGVKSDCAS
 S_2 : PADGVKDCAS
 S_3 : PPDGKSDS
 S_4 : GADGKDCCS
 S_5 : GADGKDCAS

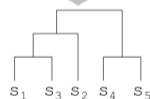
	S_1	S_2	S_3	S_4	S_5
S_1	0	0.111	0.25	0.555	0.444
S_2		0	0.375	0.222	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0

(a)

(b)

(d)
 S_1 : PPGVKSDCAS
 S_2 : PATGVKDCAS
 S_3 : PPTGKSD--S
 S_4 : GATGK-DCCS
 S_5 : GATGK-DCAS

(d)



(c)

FIGURE 6.6: The three steps of ClustalW (a progressive alignment method). Five input sequences are given in (a). Step 1 computes the pairwise distance scores for these five sequences (see (b)). Then, Step 2 generates the guide tree such that similar sequences are grouped together first (see (c)). Step 3 aligns the sequences one by one according to the branching order of the guide tree, yielding the multiple alignment of all input sequences (see (d)).

MUltiple SEquence Comparison by Log-Expectation (2004)

computer software used in bioinformatics for multiple sequence alignment.

Online available via <https://www.ebi.ac.uk/Tools/msa/muscle/>

MUltiple SEquence CComparison by Log-Expectation (2004)

computer software used in bioinformatics for multiple sequence alignment.

Online available via <https://www.ebi.ac.uk/Tools/msa/muscle/>

Basic idea for input $\zeta = \text{set } \{S_1, \dots, S_k\}$ of sequences (2nd and 3rd steps similar to ClustalW):

1 Compute k -mer distances

=(dis)similarities $D(S_i, S_j)$ between the sets of k -mers for all pairs $S_i, S_j \in \zeta$

Much(!) faster than [W1] in Clustal

W2 Use distance matrix D to compute phylogenetic tree T (via UPGMA-method)

W3 Use T to carry out a multiple alignment

4 Several re-iteration and refinement steps follow

MU**L**tiple **S**equence **C**omparison by **L**og-**E**xpectation (2004)

computer software used in bioinformatics for multiple sequence alignment.

Online available via <https://www.ebi.ac.uk/Tools/msa/muscle/>

Online Example:

>Sequence_1

GTTTATTAGTGATCATGGCTAAGTTTGCGTCCATCATCGCACTTCTTTTT

>Sequence_2

CTCGAGACAGTGATCATGGCTTCTCTCTCTCGTGCCGCATCTCACACC

>Sequence_3

TCTTGGTGAGGATCCGTTGAGAGTGATCATGGCTCGCCCCATCGCCCTNGTTAGA

>Sequence_4

GACATTACGGCGACCCAGTCTCCCAGTGATCATGGCTTCAGTGGGACTGGGCC