Part II

2 Global pairwise alignment

2.1 Pairwise alignment

A pairwise alignment is a basic sequence structure that consists of two sequences. A global alignment stretches to the whole part of two sequences, whereas a local alignment usually contains only part of the sequences.

Components of pairwise alignment

We name two sequences as database or d and query or q through this course. They may represent sequences from two different species or organisms.

Identical sequences.

```
q: ACGT
d: ACGT
```

One mismatch.

```
q: ACGT
d: ACGA
```

The '-' symbol represents a blank. A single or a set of multiple blanks further represents a gap, which is an indication of insertion or deletion in the course of evolution between two organisms.

```
q: ACGT
d: A-GT
```

N.B. A gap cannot be aligned with another gap.

Example of a simple scoring scheme

- Match: 1
- Mismatch: 0
- Gap penalty: 1 (use -1 for the actual calculation)

We may use the following notation.

- $R_{ab} = 1$ for a = b
- $R_{ab} = 0$ for a $\neq$ b
- g = 1

Exercise 2.1

Use the simple scoring scheme above and calculate the scores of the following two alignments.

```
Alignment 1                       Alignment 2
   q: GCA-GCA                        q: GCA-GCA
   d: GA-TG-A                        d: GA-TG-A
```

2.2 Alignment by brute–force

A brute–force approach finds the alignment with the highest score by simply considering all possible alignments and calculates the score for each of them.

An example of brute–force approach

We find the optimal alignment for the following sequences by using the scoring scheme below.

Sequences:

`q: AG, d: ACG`

Scoring scheme:

$R_{ab} = 1$ for a = b
$R_{ab} = 0$ for a $\neq$ b
g = 1

1. **The length of alignment**
   - Maximum length: length(q) + length(d)
   - Minimum length: max(length(q), length(d))
2. **All possible alignments when length = 5**

```
---AG    A---G    A--G-    AG---    --A-G
ACG--    -ACG-    -AC-G    --ACG    AC-G-

--AG-    -AG--    -A--G    -A-G-    A-G--
AC--G    A--CG    A-CG-    A-C-G    -A-CG
```

3. **All possible alignments when length = 4**

```
A--G    A-G-    AG--    A--G    -A-G    -AG-
ACG-    AC-G    A-CG    -ACG    ACG-    AC-G

-AG-    A-G-    --AG    --AG    -A-G    AG--
A-CG    -ACG    ACG-    AC-G    A-CG    -ACG
```

4. **All possible alignments when length = 3**

```
-AG    A-G    AG-
ACG    ACG    ACG
```

5. **Alignment with the best score**

```
ACG
A-G
```

Score: 1

Search space size of the brute-force approach

The search space size is the number of all possible alignments. It is 25 (10 + 12 + 3) for the example above.

Rapid growth of search space size

```
Example 1
    q: ACGACG, d: AGAG
```

Search space size: 1289

```
Example 2
    q: ACGACGACGACG, d: AGAGAGAG
```

Search space size: 4,673,345

Exercise 2.2

Find the alignment with the best score for the sequences. Use the simple scoring scheme below.

Sequences:

`q: A, d: AC`

Scoring scheme:
$R_{ab} = 1$ for a = b
$R_{ab} = 0$ for a $\neq$ b
g = 1

1. What are the maximum and minimum lengths of the alignment?
2. What is the search space size when the brute-force approach is used?
3. Identify all possible alignments.
4. What is the best score?

2.3 Table representation of alignment

Several data structures can be used to represent an alignment. The table representation is frequently used and also makes the process clear when we combine it with dynamic programming (DP) later.

Data structures and algorithms

It is important to consider the following aspects before solving computational problems.

1. Identify and analyze the problem you want to solve
2. Pick up an algorithm that can efficiently solve the problem
3. Decide a data structure that works with the algorithm of your choice

We use a table format (2D array) to solve global alignments by dynamic programming.

Example of table format

Alignment:

```
q: -AG-
d: A-CG
```

1. Initial setup

1. Make a table with the size of (1 + length(q)) by (1 + length(d))
2. Add the database sequence as column labels
3. And the query sequence as row labels

q/d		A	C	G
	S			
A				
G				**E**

2. Add arrows

We use three types of arrows to form an alignment.

- Move diagonally: add the letters from q and d to the alignment
- Move vertically: add - and the letter from d to the alignment
- Move horizontally: add the letter from q and - to the alignment

It should start from S and stops at E.

q/d		A	C	G
	S →			
A		↓	↘	
G				→ **E**

Exercise 2.3

Find the corresponding alignments for Table 1, 2 and 3.

Table 1

q/d		A	C	G
	S ↘			
A		↘		
G				→ **E**

Table 2

q/d		A	C	G
	S →	→	→	↓
A				↓
G				**E**

Table 3

q/d		A	C	G
	S ↓			
A	↓			
G	→	→	→	**E**

2.4 Global alignment with DP

Dynamic programming (DP) provides a solution for a multi-stage decision process, in which larger decisions recursively nest smaller decisions.

Memorize the best score in a table cell

For global alignment, the core procedure of DP is updating a cell with the highest score from the three different scores calculated from its adjacent cells. DP ends when the entire table is updated.

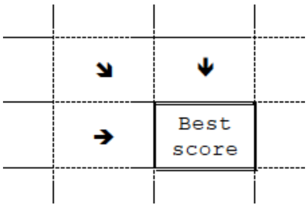


Table notation and indices

$H_{i,j}$ represents the score of the cell for the current update. $H_{i-1,j}$, $H_{i,j-1}$,and $H_{i-1,j-1}$ are the scores of the adjacent cells.

Cell $H_{i,j}$ and its adjacent cells

	d_{j-1}	d_j
q_{i-1}	$H_{i-1,j-1}$	$H_{i-1,j}$
q_i	$H_{i,j-1}$	$H_{i,j}$

Example

	d_1	d_2
q_1	$H_{1,1}$	$H_{1,2}$
q_2	$H_{2,1}$	$H_{2,2}$

Calculation of three candidate scores

$H^{(0)}_{i,j}$, $H^{(1)}_{i,j}$, and $H^{(2)}_{i,j}$ represent the three candidate scores of $H_{i,j}$. They are respectively calculated as:

$$\begin{aligned} H^{(0)}_{i,j} &= H_{i-1,j} - g && (\textit{vertical}) \\ H^{(1)}_{i,j} &= H_{i,j-1} - g && (\textit{horizontal}) \\ H^{(2)}_{i,j} &= H_{i-1,j-1} + R_{a,b} && (\textit{diagonal}) \end{aligned}$$

Exercise 2.4

Calculate the scores of $H^{(0)}_{4,6}$, $H^{(1)}_{4,6}$, and $H^{(2)}_{4,6}$ first and then update $H_{4,6}$.

	j	5	6
i		A	C
3	C	1	3
4	T	1	$H_{4,6}$

Scoring scheme:
$R_{ab} = 1$ for a = b
$R_{ab} = 0$ for a $\neq$ b
g = 1

Initialization

The first row and the first column can be calculated independently from the adjacent cells.

$$H_{0,j} : j * -1 * g$$
$$H_{i,0} : i * -1 * g$$

Example

	j	0	1	2
i			A	C
0		0	-1	-2
1	G	-1		
2	T	-2		

Exercise 2.5

Update all cells of Table 1 and 2. Use the scoring scheme in Exercise 2.4.

Table 1

		A	C
G			

Table 2

		A
G		
T		

Sub-solutions

In DP, larger decisions recursively nest smaller decisions. For instance, Table S is included in Table L.

Table S

		A
	$H_{0,0}$	$H_{0,1}$
A	$H_{1,0}$	$H_{1,1}$

Table L

		A	G
	$H_{0,0}$	$H_{0,1}$	$H_{0,2}$
A	$H_{1,0}$	$H_{1,1}$	$H_{1,2}$
C	$H_{2,0}$	$H_{2,1}$	$H_{2,2}$

Pseudo-code of updating DP table for global alignment

Algorithm 2.1: Update dynamic programming table for global alignment

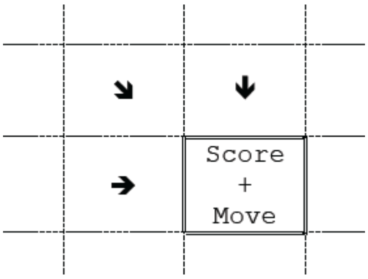
```
$H_{i,j}$ : Dynamic programming table
$R_{a,b}$: Match/mismatch scores
g   : Gap penalty

// Initialization
for $i \leftarrow 0$ to $m$ do
    $H_{i,0} \leftarrow i * -1 * g$;
end
for $j \leftarrow 1$ to $n$ do
    $H_{0,j} \leftarrow j * -1 * g$;
end

// Main loop for table update
for $i \leftarrow 1$ to $m$ do
    for $j \leftarrow 1$ to $n$ do
        $H_{i,j} \leftarrow max(H_{i-1,j} - g, H_{i,j-1} - g, H_{i-1,j-1} + R_{a,b})$;
    end
end
```

2.5 Backtracking

Backtracking is a post-processing procedure to find the alignments that have yielded the best score.

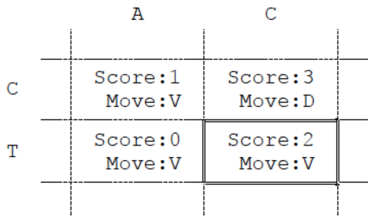
Store movement in cells

A table cell can be used for storing the movement.



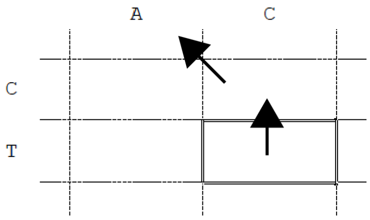
Example

Cells with scores and directions



	A	C
C	Score:1 Move:V	Score:3 Move:D
T	Score:0 Move:V	Score:2 Move:V

Use arrows to indicate backtracking



Exercise 2.6

Complete the DP table with scores and directions. What is the alignment with the best score?

		A	C
A			

Scoring scheme:
$R_{ab} = 1$ for a = b
$R_{ab} = 0$ for a $\neq$ b
g = 1

Re-calculate candidate scores

Re-calculating the three candidate scores also reveals the movement.

$$\begin{aligned} H^{(0)}_{i,j} &= H_{i-1,j} - g && (vertical) \\ H^{(1)}_{i,j} &= H_{i,j-1} - g && (horizontal) \\ H^{(2)}_{i,j} &= H_{i-1,j-1} + R_{a,b} && (diagonal) \end{aligned}$$

Example

	A	C
C	1	3
T	1	2

$$\begin{aligned} H^{(0)}_{i,j} &= 3 - 1 = 2 = H_{i,j} && \checkmark (vertical) \\ H^{(1)}_{i,j} &= 1 - 1 = 0 \neq H_{i,j} && (horizontal) \\ H^{(2)}_{i,j} &= 1 + 0 = 1 \neq H_{i,j} && (diagonal) \end{aligned}$$

Common mistake with backtracking

For the re-calculation approach, it is not to find $max(H_{i-1,j}, H_{i,j-1}, H_{i-1,j-1})$. You must re-calculate the candidates and then $max(H^{(0)}_{i,j}, H^{(1)}_{i,j}, H^{(2)}_{i,j})$ to find the actual direction.

Implementation with recursive call

Recursive calls are usually used to implement DP backtracking.

Algorithm 2.2: DP backtracking

```
$S_q$   : Sequence q
$S_d$   : Sequence d
$H_{i,j}$ : Dynamic programming table
$R_{a,b}$: Match/mismatch scores
g   : Gap penalty

proc backTrack(i, j, $A_q$, $A_d$, k)
    i   : Index of sequence q
    j   : Index of sequence d
    $A_q$ : q part of alignment (stored in reverse order)
    $A_d$ : d part of alignment (stored in reverse order)
    k   : Index for $A_q$ and $A_d$

    //
    // Need to implement recursion termination here
    // ...
    //
    if $H_{i,j} = H_{i-1,j} - g$ then                          // vertical
        $A_{q,k} \leftarrow S_{q,i}$;
        $A_{d,k} \leftarrow$ '-';
        backTrack(i − 1, j, $A_q$, $A_d$, $k + 1$);
    end
    if $H_{i,j} = H_{i,j-1} - g$ then                          // horizontal
        $A_{q,k} \leftarrow$ '-';
        $A_{d,k} \leftarrow S_{d,i}$;
        backTrack(i, j − 1, $A_q$, $A_d$, $k + 1$);
    end
    if $H_{i,j} = H_{i-1,j-1} + R_{S_{q,i},S_{d,i}}$ then          // diagonal
        $A_{q,k} \leftarrow S_{q,i}$;
        $A_{d,k} \leftarrow S_{d,i}$;
        backTrack(i − 1, j − 1, $A_q$, $A_d$, $k + 1$);
    end
end
```

Exercise 2.7

Find the alignment with the best score.

		A	C
G			
T			

Scoring scheme:
$R_{ab} = 1$ for a = b
$R_{ab} = 0$ for a ≠ b
g = 1

2.6 Needleman-Wunsch algorithm

The method of using DP to solve global pairwise alignment is called the Needleman-Wunsch algorithm in the field of bioinformatics.

Complexity

- Time: O(nm)
- Space: O(nm)

Comparisons with other algorithms

The Needleman-Wunsch algorithm is similar to several algorithms.

Divide and conquer algorithms

Sub-solutions must be independent with divide and conquer.

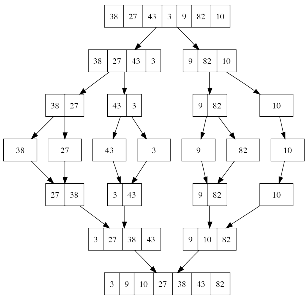


Figure 2.1: Merge sort (source: VineetKumar, Wikimedia Commons)

Dijkstra's algorithm

Worst-case performance of Dijkstra: $O(|E| + |V| \log |V|)$

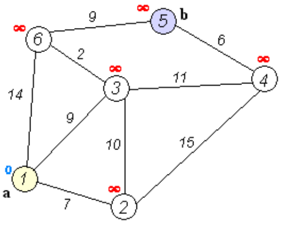


Figure 2.2: Dijkstra's algorithm (source: Ibmua, Wikimedia Commons)