BMEG3105
Lecturer: Yu LI (李煜)
Liyu95.com
liyu@cse.cuhk.edu.hk

Lecture 4: Dynamic Programming

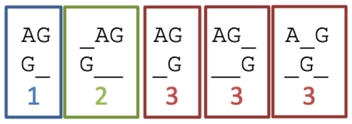
Fall 2023
Friday, 15 September 2023
Scriber: Alpha Cheung
1155133865@link.cuhk.edu.hk

What is dynamic programming? How do we do it?

- A method to identify the best alignment that can be used to identify sequence similarity
- Basic idea: divide and conquer
  1. Divide a big problem into smaller problems
  2. Solve the smaller problems
     - If a smaller problem is still too difficult to solve, we divide it further into even smaller problems
     - Otherwise, we solve it directly
  3. Combine the results to solve the original big problem

Procedure for dynamic programming (recursion tree)

- Suppose we want to align these sequences:
  - *r* = AG
  - *s* = G
- Possible alignments (we want to find out the best alignment without listing them in this way):

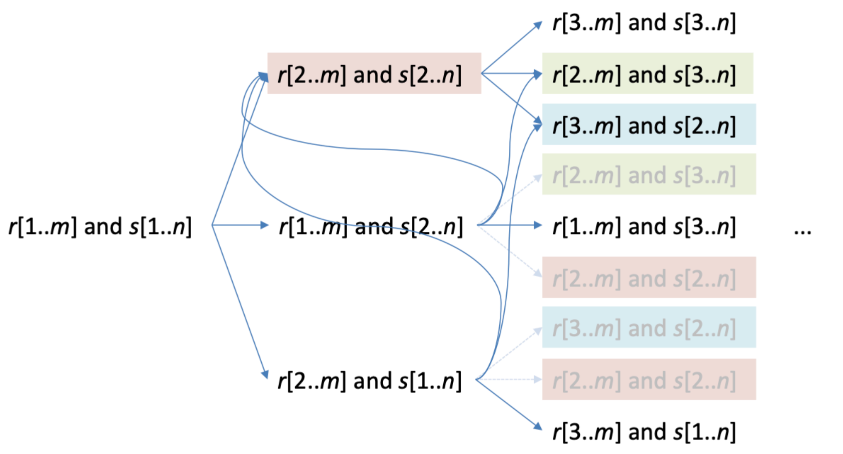


- Notation: The original problem is to align *r*[1..2] and *s*[1..1]
- To solve this problem, we want to see which of the following is the best:
  1. Align *r*[1] with *s*[1], and find the best way to align the remaining (*r*[2..2] with ϕ, i.e., empty sequence)
  2. Align a gap with *s*[1], and find the best way to align the remaining (*r*[1..2] with ϕ)
  3. Align *r*[1] with a gap, and find the best way to align the remaining (*r*[2..2] with *s*[1..1])

- We can summarize the process using a "recursion tree":

$$\begin{tikzcd}
 & & r[3..m]\text{ and }s[3..n] & \\
 & r[2..m]\text{ and }s[2..n] \arrow[ur] \arrow[r] \arrow[dr] & r[2..m]\text{ and }s[3..n] & \\
 & & r[3..m]\text{ and }s[2..n] & \\
 & & r[2..m]\text{ and }s[3..n] & \\
r[1..m]\text{ and }s[1..n] \arrow[uuur] \arrow[r] \arrow[dddr] & r[1..m]\text{ and }s[2..n] \arrow[ur] \arrow[r] \arrow[dr] & r[1..m]\text{ and }s[3..n] & \ldots \\
 & & r[2..m]\text{ and }s[2..n] & \\
 & & r[3..m]\text{ and }s[2..n] & \\
 & r[2..m]\text{ and }s[1..n] \arrow[ur] \arrow[r] \arrow[dr] & r[2..m]\text{ and }s[2..n] & \\
 & & r[3..m]\text{ and }s[1..n] &
\end{tikzcd}$$

- Important observation: many sub-problems are the same
- Idea: Store the intermediate results and reuse them



–Note that each sub-problem involves a suffix of *r* and a suffix of *s*

Procedure for dynamic programming (table representation)

1. Define a scoring matrix
2. Fill in a dynamic programming table, including arrows
3. Value in the last cell represents the best alignment score
4. Trace back arrows to obtain alignment

Scoring matrix

- Define scores for:
  - Match
  - Mismatch (mutations)
  - Gaps (indels, gene duplications)
- Scores can be any numbers of your choice (based on your needs, or built from different databases), e.g.:

	A	**C**	**G**	**T**	**ø (gap)**
A	2	-1	-1	-1	-2
C	-1	2	-1	-1	-2
G	-1	-1	2	-1	-2
T	-1	-1	-1	2	-2
ø (gap)	-2	-2	-2	-2	0

BMEG3105
Lecturer: Yu LI (李煜)
Liyu95.com
liyu@cse.cuhk.edu.hk

Fall 2023
Friday, 15 September 2023
Scriber: Alpha Cheung
1155133865@link.cuhk.edu.hk

Filling in dynamic programming table

Suppose we have a simple example: alignment between sequences ACCG and ACG

Step 1: Draw out the relevant table as below, with gap coming before the sequence itself:

	ø (gap)	**A**	**C**	**C**	**G**
ø (gap)					
A					
C					
G					

Step 2: Fill out the gap penalties first, using the scoring matrix in section 1.2. Mark progressions with an arrow:

	ø (gap)	**A**	**C**	**C**	**G**
ø (gap)	0 →	-2 →	-4 →	-6 →	-8
A	↓ -2				
C	↓ -4				
G	↓ -6				

Step 3: For the cell highlighted with red border, there are 3 options, choose the option that provides the highest score:

<u>Case 1</u>: Align row[2], col[2]

A

A

Score: 2 (match)

+2 from row[1], col[1]

Diagonal arrow

<u>Case 2</u>: Align row[2] with gap

_

A

Score: -4 (gap)

-2 from row[2], col[1]

Horizontal arrow

<u>Case 3</u>: Align col[2] with gap

A

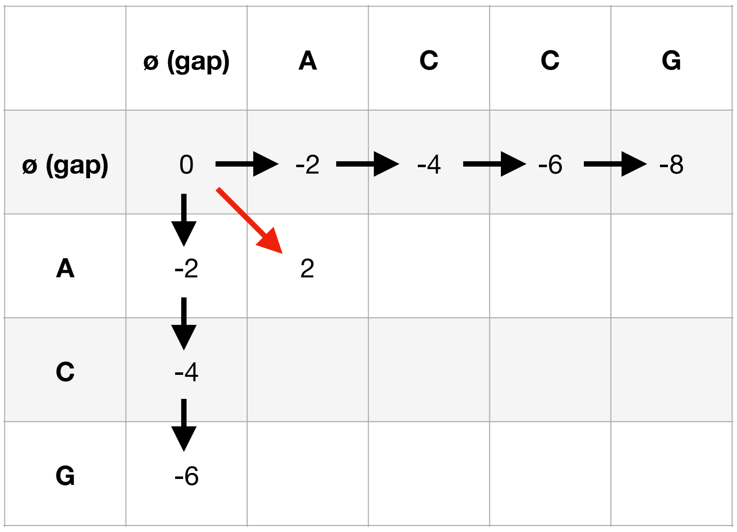
_

Score: -4 (gap)

-2 from row[1], col[2]

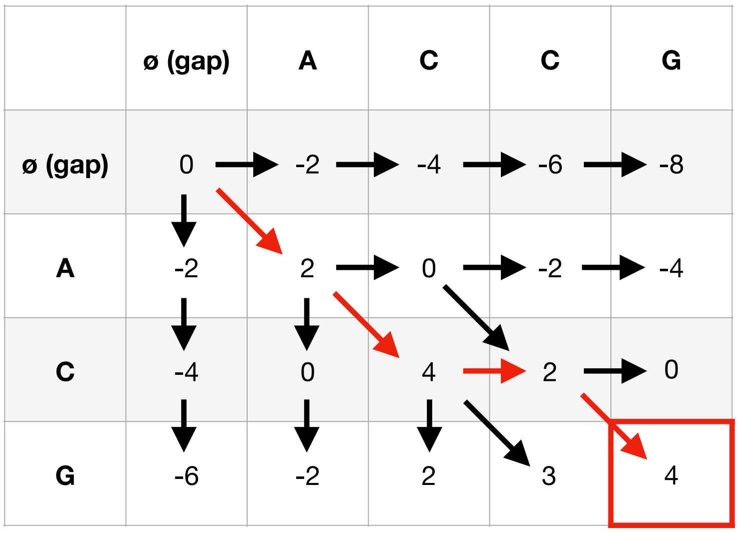
Vertical arrow

In this example, case 1 provides the highest score amongst the three options. Therefore, we select case 1, and fill in the value and the arrow in the table accordingly:



	ø (gap)	A	C	C	G
ø (gap)	0	→ -2	→ -4	→ -6	→ -8
A	↓ -2	↘ 2			
C	↓ -4				
G	↓ -6				

Step 4: Repeat step 3 for all remaining cells:



	ø (gap)	A	C	C	G
ø (gap)	0	→ -2	→ -4	→ -6	→ -8
A	↓ -2	↘ 2	→ 0	→ -2	→ -4
C	↓ -4	↓ 0	↘ 4	↘ → 2	→ 0
G	↓ -6	↓ -2	↓ 2	↘ 3	↘ $\boxed{4}$

Step 5: Track back arrows to get the sequence alignment(s):

ACCG

AC_G

Highest score = the value in the bottommost right corner = 4