Sequence Alignment

It is a way of arranging [illegible] sequences of characters to identify regions of similarity.

Sequence:- Can be taken as order of string of letters.

↳ These sequences can be DNA, RNA, Proteins sequences.

Dynamic Programming:- The process of breaking down the problem into subproblem. Stores the results of the subproblem - which will helps to avoid the computing same results again and again.

⇒ Dynamic programming will not be useful when there are no common (overlaping) subproblems.

Needleman - Wunsch Algorithm → This algorithm will implements the dynamic Algorithm paradigm - which will provide the optimal solution - with the corresponding score.

Sequence Alignment:- The process of aligning of nucleotide / characters which will maximize the number of matches and minimize the number of mismatches and gap.

The algorithm which defines a recurrence relation which will provide an reward for a nucleotide / character match, and penalty for a mismatch.

Results of Sequence Alignment:

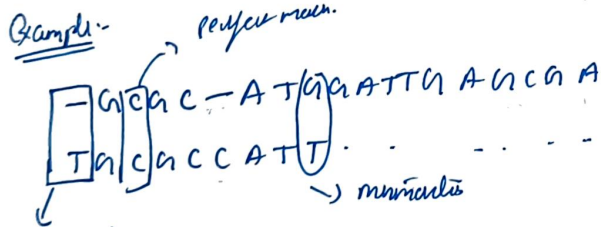
When we are given two sequences. The output of the sequence alignment will be. (consists of)

(i) Insertion / deletion (indel)

(ii) mismatches

(iii) Perfect match.

Example:-



Approaches

Sequence Alignment

- Qualitative
  - Dot plot
- Quantitative
  - global
  - local

-> Pictorial representation & relationship between two sequences

-> uses a table.

-> doesn't guarantee

Alignment:

⇒ Construction of the best alignment between two sequences

Global sequence:- The best alignment over the entire length of two sequences.

↓

when similar:- if two sequence are of same length, with a significant degree of similarity throughout.

Local sequence:- Compare the shortest proportional of sequences

↳ when similar:- when short length of similarity and different in storing length.

How to analyse?

⇒ Introduced a scoring schema. → we will have an alignment score. → goodness of alignment.

Substitution matrix:

It means which contain A C G T. which also contain a scoring function (σ) → which will depict in some to negative substitution matrix.

↳ Match → 2

Mis-match → −1

	A	C	G	T
A	2	−1	−1	−1
C	−1	2	−1	−1
G	−1	−1	2	−1
T	−1	−1	−1	2

Gap: → consecutive run of spaces in an alignment (insertion vs deletion).

Gap penalty: → defines a score to give insertion vs deletion

↳ -1 // (a) 0

for any dynamic programming, we need to define a recurrence relation.

$$T(i,j) = \max \begin{cases} T(i-1, j-1) + \sigma(S_1(i), S_2(j)) \\ T(i-1, j) + \text{gap penalty} \\ T(i, j-1) + \text{gap penalty} \end{cases}$$

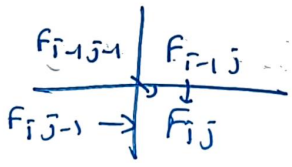
→ max / minimize

Initialization:

$F(0,0) = 0$

$T(i,0) = F(i-1, 0) - \text{gap penalty}$

$T(0,j) = T(0, j-1) - \text{gap penalty}$.



Termination: Bottom right.

Steps involved:

↳ Step 1: Fill up the matrix table (T) using the recursive relation.

→ Step 2: Traceback the matrix to work out your a best alignment.

Sequences:

S_1 = T C G T C

S_2 = A T C G T

Table initialization:

		A	C	G	T
	A	2	-1	-1	-1
	C	-1	2	-1	-1
	G	-1	-1	2	-1
	T	-1	-1	-1	2

		i=0	i=1	i=2	i=3	i=4	i=5
			T	G	G	T	G
j=0		0	0	0	0	0	0
j=1	A	0	-1	-1	-1	-1	-1
j=2	T	0	2	1	0	1	0
j=3	C	0	1	1	0	0	0
j=4	G	0	0	3	3	2	2
j=5	T	0	2	2	2	5	4

Alignment = 5 //

→ Step-1 => The value of T(0,0) is zero → Starting condition

→ Step-2 => Gap penalty → 0

→ Step-3 => Match → +2, mismatch = -1

→ Traceback.

→ Look for shared changes //

S_1 = T C G T C

S_2 = A T C G T

-	T	G	G	T	G
	|		|	|	
A	T	C	G	T	-

Score = Number of matches + sum of mismatch + gap

= 3(2) + -1 + 0

= 5 //

Performance:

- N-w → approach → n^2 //
- accessing all possible combination → $^{2n}C_n$

$N^2 < {}^{2n}C_n$ //

N-w first then access all possible element one by one.