STRING / SEQUENCE COMPARISON

Biological sequences $\cong$ strings over a fixed alphabet

DNA $\simeq$ $\{a, c, g, t\}$
RNA $\simeq$ $\{a, c, g, u\}$
protein $\simeq$ $\{$ 20 amino acid residues $\}$

Motivation for sequence comparison:

Let g_1 ~~and~~ and g_2 be 2 genes, obtained from (i.e. related species eg., human & chimp

g_1 : a c a g a g t a a c
g_2 : a c a t a b a g a c

Hypothesis: g_1 and g_2 are evolutionarily closely related

$\Rightarrow$ $\{$ g_1 is an ancestor of g_2
g_2 is an ancestor of g_1
g_1 and g_2 share the same ancestor $\}$

Testing the hypothesis:

$\Rightarrow$ How many changes/edits does one have to make to g_1 to make it look like g_2? (i.e. vice versa)

Allowed edits — mutation/substitution
— insertion } "gaps"
— deletion }

$\Rightarrow$ Need a way to compare and evaluate the similarity (or differences) between two strings.

Sequence/String Alignment:

Given two strings s_1 and s_2, an "alignment" between them is an order-preserving way of mapping characters in one string against characters in the other string or against a "gap character ("-")

Computing Optimal Alignments

- Combinatorially optimal
- statistically optimal

↗ First let us discuss this.

Example alignments:

Input: s_1: a c a g a g t a a c
s_2: a c a a g t g a t c

Alignment #1:

```
s1: a  c  a  g  a  g  t  a  a  c
    |  |  *  *  *  *  *  |  *  |
s2: a  c  a  a  g  t  g  a  t  c
```

#matches = 4
#mismatches = 6

can this be improved?

Alignment #2:

deletion, insertion, substitution

```
s1: a  c  a  g  a  g  t  _  a  a  c
    |  |  |  |  |  |  |     |  *  |
s2: a  c  a  _  a  g  t  g  a  t  c
```

#matches = 8
#mismatches = 1
#gaps = 2

How good are the above two alignments?

Need a Scoring mechanism!

Scoring an alignment:

Step 1) Count #matches, #mismatches, #gaps

	#matches	#mismatches	#gaps
Alignment #1:	4	6	0
Alignment #2:	8	1	2

Step 2)

Reward matches ⟶ (+)ve score weight

Penalize mismatches and gaps ⟶ (−)ve score weight

E.g: Let the score weights be: match = +1
mismatch = −1
gap = −1

∴ Score (Alignment #1) = ~~4−6+0~~ 4−6+0 = −2

Score (Alignment #2) = 8 − 1 − 2 = +5

⇒ Score(Alignment #2) > Score (Alignment #1)

⇒ Alignment #2 is more plausible than #1.

The Sequence Alignment Problem:

Given S_1 and S_2, find an alignment that maximizes the score of the alignment.

Note: 1) There could be more than one optimal alignments for S_1 & S_2.

2) When computing an opt. alignment we always assume that a scoring scheme is specified by the user.

Alignment score = #matches × match score
+ #mismatches × mismatch penalty
+ #gaps × gap penalty

Variations:

i) Treat insertions different from deletions
ii) Give/assign different penalties for mismatches vs. gaps

Question: If the Optimal alignment score between S_1 & S_2 is x, then what will be the optimal alignment score between S_2 & S_1, ? i.e., $OPT(S_1, S_2) = OPT(S_2, S_1)$?

Algorithm to compute an optimal Alignment:

Input: $S_1[1..m]$ vs. $S_2[1..n]$

Algo:
Let $T(i, j)$ = the optimal score of aligning prefixes $S_1[1..i]$ vs. $S_2[1..j]$

$\Rightarrow T(m, n) = opt(S_1, S_2)$

How to compute $T(i, j)$ $(1,1 \leq \forall i, j \leq m, n)$

Let $S_1 = a_1 a_2 a_3 \ldots\ldots a_i \ldots\ldots a_m$

$S_2 = b_1 b_2 b_3 \ldots\ldots b_j \ldots\ldots b_n$

Let $\delta(a_i, b_j)$ → substitution score of a_i vs. b_j

g → gap penalty

For $T(i, j)$:

There are only three possible ways of ~~alignment~~ an alignment of $s_1[1..i]$ vs. $s_2[1..j]$ can end:

case (i) $\begin{array}{|l|l|} \hline a_1\ a_2 \cdots a_{i-1} & a_i \\ \hline b_1\ b_2 \cdots b_{j-1} & b_j \\ \hline \end{array}$. a_i substitutes b_j

case (ii) $\begin{array}{|l|l|} \hline a_1\ a_2 \cdots a_{i-1} & a_i \\ \hline b_1\ b_2 \cdots b_j & - \\ \hline \end{array}$ a_i is deleted in s_2

case (iii) $\begin{array}{|l|l|} \hline a_1\ a_2 \cdots a_i & - \\ \hline b_1\ b_2 \cdots b_{j-1} & b_j \\ \hline \end{array}$ b_j is inserted in s_2

- Each $T(i, j)$ is a subproblem
- There are only $O(mn)$ subproblems
- $T(m, n)$ can be computed recursively from its subproblems

Use dynamic programming!

Recurrence:

$$T(i, j) = \max \begin{cases} T(i-1, j-1) + \delta(a_i, b_j) & \rightarrow (i) \\ T(i-1, j) + g & \rightarrow (ii) \\ T(i, j-1) + g & \rightarrow (iii) \end{cases}$$

Initialization:

$T(0, 0) = 0$

$T(0, j) = g \times j$

$T(i, 0) = g \times i$

Final optimal score $= T(m, n)$

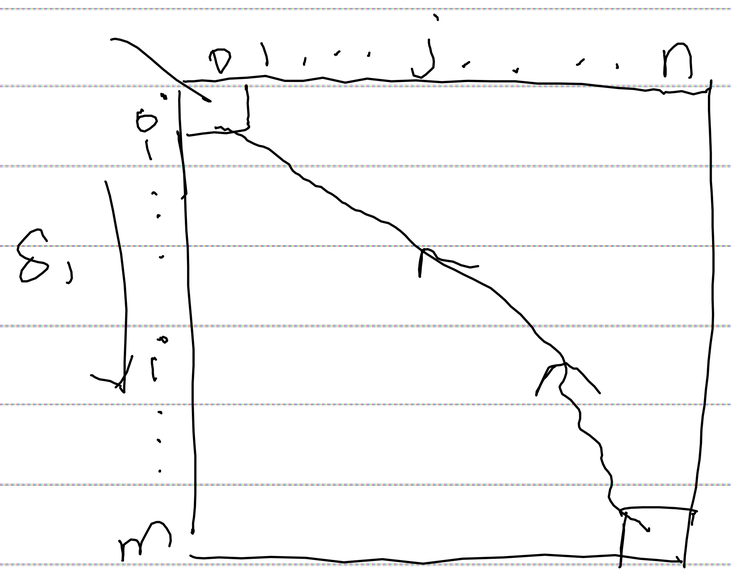
GLOBAL ALIGNMENT algorithm

Algorithm: (Needleman & Wunsch, 1970)

```
int Align ( S1 , S2 )
{
  // Init:  Table T[0..m][0..n]
        For i = 0 to m      T(i, 0) = g × i ;
        For j = 0 to n      T(0, j) = g × j ;

  // Alignment:

        For( i = 1 to m ) {
        For( j = 1 to n ) {
              T(i, j) = max { T(i-1, j-1) + δ(ai, aj)
                            { T(i, j-1) + g
                            { T(i-1, j) + g
        }
  }
  return T[m][n];
}
```



Time Complexity = O(mn)

Space Complexity = O(mn)

This is called "global" alignment because S_1 is wholly aligned against S_2.

Alignment Retrace (optimal path retrace)

Use backtracking