COMP 571: Homework #1

Vi Nguyen (vkn1)

February 7, 2017

On my honor, I have neither given nor received any unauthorized aid on this assignment.

1. (a) The highest score is **4**. An example of an optimal alignment is

-	C	-	D	A	A
A	E	E	C	-	A

This was computed by filling in the dynamic programming table as follows:

		A	E	E	C	A
	0	-2	-4	-6	-8	-10
C	-2	-2	-2	-4	-5	-7
D	-4	-4	-4	-4	-4	-6
A	-6	-2	-4	-5	-6	-2
A	-8	-4	-3	-5	-7	-4

(b) The completed matrix is as follows:

		A	E	E	C	A
	0.0	-2.2	-2.6	-3.0	-3.4	-3.8
C	-2.2	-2.0	-2.2	-2.6	-2.0	-4.2
D	-2.6	-4.2	-4.0	-4.2	-2.6	-4.0
A	-3.0	-0.6	-2.8	-3.2	-3.0	-0.6
A	-3.4	-1.0	-1.6	-2.0	-2.4	-1.0

An optimal alignment is

-	-	C	D	A	A
A	E	E	C	-	A

(c) For (a), a minimum of 5 mutations occurred. For (b), a minimum of 5 mutations occurred.

2. When $\mu < -20$ and $g(k) = -10k$, it is always more optimal to insert a gap rather than mismatch two residues, since mismatching will result in a less than -20 penalty while adding two gaps (to avoid a mismatch) on either side of the sequence alignemt will also result in a -20 penalty. Therefore, in this case, the optimal sequence alignments will be the same, since in any case where two residues would be mismatched, simply match a gap to each residue.

3. (a) This is reasonable when one of the sequences is a (potential) subsequence of the other, so that we are "searching" the larger sequence for where the subsequence aligns.

   (b) i. When initializing the DP matrix, the leftmost column and topmost row will be initialized to 0, which scores gaps at the beginning of the alignment as 0.

   ii. When computing the bottom right cell of the matrix, let the gap penalty for using the top or left cell by 0. If the top or left cell is selected to compute the bottom right cell, let the gap penalty in the "same" direction be 0, and so on. This allows end gaps to be scored as 0.

   (c) The following alignments all have a score of 1.

-	-	A	R	T	-	-
A	A	R	R	T	R	T

-	A	R	T	-	-	-
A	A	R	R	T	R	T

-	-	-	-	A	R	T
A	A	R	R	T	R	T

4. Essentially, we want to split A and B into non-anchored "chunks" and then perform the alignments between the anchors.

   (a) Align (using Needleman-Wunsch) $A[0 : i_1]$ and $B[0 : j_1]$

   (b) Append $A[i_1]$ and $B[j_1]$ to the resulting alignment

   (c) For k from 2 to n:

   - Align $A[i_{k-1} + 1 : i_k]$ and $B[j_{k-1} + 1 : j_k]$ and append to running alignment
   - Append $A[i_k]$ and $B[j_k]$ to running alignment

   (d) Align $A[i_n + 1 : len(A - 1)]$ and $B[j_n + 1 : len(B - 1)]$ and append to running alignment

   (e) Return alignment

   This algorithm is correct because it optimally aligns the sections of the sequence that aren't anchored. The sections between anchors are independent of each other, since we cannot align "across" anchor points. Thus, we can simply align corresponding unanchored substrings and combine all of these alignments (including the anchors) to obtain a full alignment.

5. Consider an m by n alignment matrix. Assume (without loss of generality) that $m < n$. We will then show that the number of possible pairwise alignments is

$$\sum_{k=0}^{m} \binom{n+m-k}{k, m-k, n-k}$$

We can do this by illustrating that the number of possible paths through the alignment matrix is equal to the above expression.

We want to find the number of paths from the top-left cell to the bottom-right cell. We can consider that we have some how traversed to cell (i, j) and want to find a path to the bottom right cell (m, n). In the sum above, we would be in iteration i. We then compute the number valid paths to (m, n). Then, by summing for all iterations, we can compute the total number of paths through the matrix.