Introduction to Bioinformatics Algorithms
Homework 3 Solution

Saad Mneimneh
Computer Science
Hunter College of CUNY

Problem 1: Concave penalty function
We have seen in class the following recurrence for alignment with a general penalty function:

$$A(i,j) = \max \begin{cases} A(i-1,j-1) + s(x_i, y_j) & \\ A(i-k,j) - \gamma(k) & k = 1,\dots,i \\ A(i,j-k) - \gamma(k) & k = 1,\dots,j \end{cases}$$

where $A(i,0) = -\gamma(i)$ and $A(0,j) = -\gamma(j)$, and $\gamma(k)$ is the penalty of a gap of length k.

(a) Show by a counter example that this algorithm fails to correctly find the optimal score. *Hint*: construct a γ that is not concave.

Solution: Consider the penalty function $\gamma(0) = 0, \gamma(1) = 2, \gamma(x) = 10$ for $x > 1$. Assume that a match has a $+1$ score, and a mismatch a -1 score. Consider the two strings A and ABA. Then the optimal alignment has score -5:

```
A B A
- A -
```

However, the algorithm will compute the optimal score as -3 for the following alignment:

```
A B A
A - -
```

This is the score of a gap of length 1 plus the score of optimally aligning AB and A, which is -1. The problem is that the penalty of a gap of length 2 is higher then the penalty of two gaps of length 1. The algorithm will fail because splitting this gap falsely suggests a smaller penalty. This will not happen if the condition in part (b) below is satisfied.

(b) Show that if $\gamma(0) = 0$ and $\gamma(k) - \gamma(k-1)$ is non-increasing in k, then we have (sub-additive property)

$$\gamma(k_1 + k_2) \leq \gamma(k_1) + \gamma(k_2)$$

Explain why this property makes the algorithm above correct.

Solution:

$$\gamma(x_1+x_2) = \gamma(x_1)+[\gamma(x_1+1)-\gamma(x_1)]+[\gamma(x_1+2)-\gamma(x_1+1)]+\ldots+[\gamma(x_1+x_2)-\gamma(x_1+x_2-1)]$$

$$\leq \gamma(x_1) + [\gamma(1) - \gamma(0)] + [\gamma(2) - \gamma(1)] + \ldots + [\gamma(x_2) - \gamma(x_2 - 1)]$$

$$= \gamma(x_1) + \gamma(x_2) - \gamma(0) = \gamma(x_1) + \gamma(x_2)$$

Since the penalty of a gap becomes higher when the gap is scored as separate smaller pieces, the algorithm will not favor such erroneous breaks.

(c) Modify the algorithm to work with any gap function.

Solution: Let $A(i,j)$ be the score of the optimal alignment that ends in x_i and y_j aligned. Let $B(i,j)$ be the score of the optimal alignment when x_i is aligned with a gap. Finally, let $C(i,j)$ be the score of the optimal alignment when y_j is aligned with a gap.

$$A(i,j) = \max \begin{cases} A(i-1,j-1) + s(x_i,y_j) \\ B(i-1,j-1) + s(x_i,y_j) \\ C(i-1,j-1) + s(x_i,y_j) \end{cases}$$

$$B(i,j) = \max \begin{cases} A(i-k,j) - \gamma(k) & k = 1 \ldots i \\ C(i-k,j) - \gamma(k) & k = 1 \ldots i \end{cases}$$

$$C(i,j) = \max \begin{cases} A(i,j-k) - \gamma(k) & k = 1 \ldots j \\ B(i,j-k) - \gamma(k) & k = 1 \ldots j \end{cases}$$

For initialization: $A(0,0) = 0$, $A(i,0) = A(0,j) = -\infty$, $B(i,0) = -\gamma(i)$, $B(0,j) = -\infty$, $C(0,j) = -\gamma(j)$, $C(i,0) = -\infty$. The running time of this algorithm is $O(m^2n + n^2m)$.

Problem 2: Substitution matrices

Let's say that we would like to build a DNA substitution matrix (4x4 matrix) optimized for finding alignments among sequences that have 99% conservation (so mutation rate is 0.01, evolutionary distance 1). Assume that $p_i = 0.25$ for every nucleotide i (A, G, C, or T), and that all matches are equally probable, and all mismatches are equally probable (uniform model).

(a) Find the matrix Q, where Q_{ij} is the probability of seeing nucleotides i and j aligned.

Solution:

$$Q = \begin{bmatrix} \frac{0.99}{4} & \frac{0.01}{12} & \frac{0.01}{12} & \frac{0.01}{12} \\ \frac{0.01}{12} & \frac{0.99}{4} & \frac{0.01}{12} & \frac{0.01}{12} \\ \frac{0.01}{12} & \frac{0.01}{2} & \frac{0.99}{4} & \frac{0.01}{12} \\ \frac{0.01}{12} & \frac{0.01}{12} & \frac{0.01}{12} & \frac{0.99}{4} \end{bmatrix}$$

(b) After constructing the matrix M, where $M_{ij} = Q_{ij}/p_i$, find the match and mismatch scores in the matrix S given by $S_{ij} = \log_2 M^k_{ij}/p_j$, for evolutionary distances $k = 1, 2, 5, 25, 50, 75, 100$. What is the percentage conservation in each case (e.g. for $k = 1$ it is 0.99)?

Solution: Here are the match and mismatch scores found for the different evolutionary distances, and the percentage conservation:

	match	mismatch	% conservation
1	1.98	-6.23	99
2	1.97	-5.25	98.0
5	1.93	-3.94	95.1
10	1.86	-2.99	90.6
25	1.65	-1.81	78.6
50	1.34	-1.03	63.3
75	1.07	-0.66	52.4
100	0.83	-0.44	44.6

(c) [optional] Show that regardless of the probabilities p_i and Q_{ij}, S will always be symmetric for any evolutionary distance. *Hint*: $M = DQ$ for some diagonal matrix D (what is D?), and $S = \log_2[(DQ)^k D]$.

Solution: D is the diagonal matrix defined as $D_{ii} = 1/p_i$ and $D_{ij} = 0$ for $i \neq j$. Observe that D and Q are symmetric. A matrix is symmetric iff it is equal to it's transpose. In addition $(AB)^T$ is $B^T A^T$. Moreover, matrix multiplication is associative. So

$$[(DQ)^k D]^T = (DQDQ \dots DQD)^T = D^T Q^T D^T \dots Q^T D^T Q^T D^T$$

$$= DQD \dots QDQD = (DQ)^k D$$

Problem 3: Longest increasing subsequence

Given a sequence $x_1, x_2, \dots, x_n$, an increasing subsequence of length k is a sequence $x_{i_1}, x_{i_2}, \dots, x_{i_k}$ such that $i_1 < i_2 < \dots < i_k$ and $x_{i_1} < x_{i_2} < \dots < x_{i_k}$. For example, in the sequence $8, 2, 1, 6, 5, 7, 4, 3, 9$, an increasing subsequence is $1, 5, 7, 9$. It has length 4, and it is a longest possible increasing subsequence.

(a) Describe a dynamic programming algorithm to find a longest increasing subsequence. *Hint:* you can think about alignment between the sequence and its sorted version (what scores will you assign for matches, mismatches, and gaps?).

Solution: We can align the sequence x with it's sorted version y. For any given pair (i, j), the score of the optimal alignment of $x_1 \dots x_i$ and $y_1 \dots y_j$ is given by (gaps score 0):

$$A(i,j) = \max \begin{cases} A(i-1, j-1) + s(x_i, y_j) \\ A(i-1, j) \\ A(j, j-1) \end{cases}$$

where $A(i, 0) = A(j, 0) = 0$ and $s(x_i, y_j) = 1$ if $x_i = y_j$ and -1 otherwise. This way, we will never align x_i with y_j if they are not equal, because the mismatch can be replaced by two gaps (one in each sequence).

(b) A 2-increasing subsequence is one that can be partitioned into two subsequences that are increasing. For example, while $2, 1, 6, 5, 7, 9$ is not an increasing subsequence, it is a 2-increasing subsequence because it can be partitioned into: $2, 6$ and $1, 5, 7, 9$. Show by a counter example that the longest 2-increasing subsequence cannot be obtained by a greedy strategy, i.e. removing the longest increasing sequence, then finding the longest increasing sequence among the remaining elements, and finally interleaving the two.

Solution: Consider the sequence

$$i + 1, 1, \dots, i - 1, i + 2, \dots, n, i$$

The longest increasing sequence has length $n - 2$, leaving two elements $i + 1, i$, which define an increasing sequence of length 1. So in total, we achieve length $n - 1$. However, the entire sequence of length n can be partitioned into $1, \dots, i$ and $i + 1, \dots, n$.

(c) Describe a dynamic programming algorithm to find the longest 2-increasing subsequence. *Hint:* you can think about aligning the sequence with two of its sorted versions. But this is not the standard multiple alignment because an alignment up to x_i, y_j, and z_k can end in one of 5 possibilities:

$$\begin{array}{ccccc} x_i & x_i & x_i & - & - \\ y_j & - & - & y_j & - \\ - & z_k & - & - & z_k \end{array}$$

Solution: This generalizes part (a). $A(-,-,-)$ is dropped from the maximization if any of the indices is negative. This will simplify the initial condition to $A(0, 0, 0) = 0$.

$$A(i,j,k) = \max \begin{cases} A(i-1, j-1, k) + s(x_i, y_j) \\ A(i-1, j, k-1) + s(x_i, z_k) \\ A(i-1, j, k) \\ A(i, j-1, k) \\ A(i, j, k-1) \end{cases}$$

where $A(0, 0, 0) = 0$ and s defined as before.

Problem 4: Homodeletions
Problem 6.40 in the book.

Solution: We only allow matches and gaps. A match has a score of $+1$. The score of the gap will depend on whether it's an opening of the gap (-1), or an extension of it (zero). Let $A(i, j)$ be the score of the optimal alignment that ends in x_i and y_j aligned. Let $B(i, j)$ be the score of the optimal alignment when x_i is aligned with a gap. Finally, let $C(i, j)$ be the score of the optimal alignment when y_j is aligned with a gap.

$$A(i,j) = \min \begin{cases} A(i-1, j-1) + s(x_i, y_j) \\ B(i-1, j-1) + s(x_i, y_j) \\ C(i-1, j-1) + s(x_i, y_j) \end{cases}$$

$$B(i,j) = \min \begin{cases} A(i-1, j) + 1 \\ B(i-1, j) + \gamma(x_{i-1}, x_i) \\ C(i-1, j) + 1 \end{cases}$$

$$C(i,j) = \min \begin{cases} A(i,j-1)+1 \\ B(i,j-1)+1 \\ C(i,j-1)+\gamma(y_{j-1},y_j) \end{cases}$$

where $s(x_i, y_j) = 0$ if $x_i = y_j$ and $-\infty$ otherwise (no mismatches), and $\gamma(a,b) = 1$ if $a \neq b$ and 0 otherwise. For initialization: $A(0,0) = 0$, $A(i,0) = A(0,j) = -\infty$, $B(1,0) = 1$, $B(i,0) = B(i-1,0)+\gamma(x_{i-1},x_i)$, $B(0,j) = -\infty$, $C(0,1) = 1$, $C(0,j) = C(0,j-1) + \gamma(y_{j-1},y_j)$, $C(i,0) = -\infty$. The running time of this algorithm is $O(mn)$.