Homework 4

CS 4390/5390
Fall 2019

Due: 13 November 2019

This homework is worth 6 points out of the total 25 points of homework in the class.

1. **(1 Point)** *Assume that we compare the sequences below by counting the shared number of* 3*-mers. For the following sequences, which pair is more similar.*

   - S_1 = `ACGTCGATC`
   - S_2 = `CCGGCGTCA`
   - S_3 = `ACGCTCGAT`

3-mer	S_1	S_2	S_3
ACG	1		1
ATC	1		
CCG		1	
CTC			1
CGA	1		1
CGC			1
CGT	1	1	
CGG		1	
TCA		1	
TCG	1		1
GAT	1		1
GCT			1
GCG		1	
GTC	1	1	
GGC		1	

	S_1	S_2	S_3
S_1	-	2	4
S_1	2	-	0
S_1	4	0	-

S_1 and S_3, are most similar under this measure, sharing 4/7 3-mers.

2. **(2 points)** *Given a set of sequences $S_1, S_2, S_3, ..., S_k$, we would like to find k substrings $T_1, T_2, T_3, ..., T_k$ of $S_1, S_2, S_3, ..., S_k$ respectively, such that the optimal SP score of the multiple sequence alignment of $T_1, T_2, T_3, ..., T_k$ is maximized. (a) Design a dynamic programming algorithm to solve the problem with match, mismatch, and indel penalties of α, β, γ respectively (i.e. not affine gap scoring). (b) What is the time complexity? (Note that when $k = 2$, this problem is the same as pairwise local alignment).*

define a recurrence formula as follows:

$$V[(i_1, i_2, ..., i_k)] =: \max \Bigg\{ 0, \max_{1 \le j \le 2^k} \Bigg\{ V[(i_1, i_2, ..., i_k) - \mathbb{B}_j] \tag{1}$$

$$+ \alpha \sum_{1 \le \ell \le k} \sum_{(\ell+1) \le p \le k} \mathbb{B}_j[\ell] \mathbb{B}_j[p] \mathbb{I}(S_\ell[i_\ell] - 1] = S_p[i_p - 1]) \} \tag{2}$$

$$+ \beta \sum_{1 \le \ell \le k} \sum_{(\ell+1) \le p \le k} \mathbb{B}_j[\ell] \mathbb{B}_j[p] \mathbb{I}(S_\ell[i_\ell] - 1] \neq S_p[i_p - 1]) \} \tag{3}$$

$$+ \gamma \sum_{1 \le \ell \le k} \sum_{1 \le p \le k} \mathbb{B}_j[\ell] \left(1 - \mathbb{B}_j[p]\right) \Bigg\} \Bigg\} \tag{4}$$

Here $\mathbb{B}_j$ is the binary vector encoding of j, and $\mathbb{I}$ is an indicator function that returns 1 when the condition is true, and 0 otherwise. The traceback would be performed just as it normally is in local alignment, first finding the maximum alignment value in the table, then running traceback using the (stored) indicator directions.

The matrix V has n^k elements, when each one is filled in the internal max has 2^k different binary vectors to examine, each one taking k^2 operations. Therefore the total running time of this algorithm is $O(n^k 2^k k^2)$. Finding the maximum can be done in a similar amount of time, as can traceback and alignment recovery.

3. **(1 point)** *Given an additive tree $T = (E, V)$ for n species. (a) Describe an algorithm for reconstructing the distance matrix between all of the species. (b) What is the time complexity of this algorithm?*

   Algorithm 1:
   $\forall (A, B) \in V^2, A \neq B$:
   $L :=$ the edge list of the path from A to B in T
   $D(A, B) = \sum_{e \in L} \omega(e)$

   where $\omega(e)$ is the weight of edge $e \in E$ in T.

   The algorithm above for $n = |V|$ would take $O(n^3)$ time since for each of the $O(n^2)$ pairs of groups, we might need to do an $O(n)$ traversal to find the path, and the summation is $O(n)$.

   If you order the vertices, you could only cut the constant in half since you would fill in $D(A, B)$ and $D(B, A)$ at the same time but it would still be $O(n^3)$.

   If you use one depth first traversal, for each node, you can fill in the table in $O(n^2)$ time:

   Algorithm 2:
   $\forall A \in V$, run a depth first traversal keeping a sum, s of the edge weights. When you move down an edge e add it's weight to the running sum ($s+ = \omega(e)$), when you come back up subtract the value ($s- = \omega(e)$). When you reach a leaf, w.l.g. call it B, assign $D(A, B) = s$.

   Each traversal takes $O(n)$ time ($|E| = O(n)$ since its a binary tree), and this is performed for each of the n nodes.

4. **(2 point)** *When constructing a neighbor-joining tree from a set of taxa, we select the pair to merge that is equally balances the desire to find the pair that is closest in distance and the desire to find the pair that is on average furthest from everything else. We could also look at the minimum distance between a group and the other groups. (a) Create an algorithm that merges pairs of groups that:*

   - *unequally tries to (weighted using parameter α)*
   - *the minimize distance between the two groups, and*
   - *the maximize the minimum distance between either of the two groups and any other group not in the pair.*

   (b) What is the time complexity of your new algorithm?

   within the neighbor joining algorithm update the value in the argmin to the following:

$$\underset{\substack{(A,B)\in S^2\\ A\neq B}}{\operatorname{argmin}} \left\{ \alpha D(A,B) - (1-\alpha) \left(\min_{C\in S\setminus\{A,B\}} \left\{ \min\left\{ D(A,C), D(B,C) \right\} \right\} \right) \right\}$$

   The addition of this change make it so the running time of each iteration increases by a factor of $O(n')$ where n' is the number of groups left at that stage. This means the new total running time is $O(n^4)$ using the analysis from class.