Algorithms in Computational Biology – Homework Exercise 2

Publication date: *Thursday, November 26*
Due date: *Sunday, December 13 (9pm IST)*

Problem 1:

A **non-contiguous substring** of S is a string obtained by deleting a subset of characters from S. For instance, **`TCA`** is a **non-contiguous substring** of **`ATCGA`** (deleting the first and fourth characters), but **`AGT`** is not.

An **alignment with omissions (AWO)** of two sequences S and T, is an alignment of two non-contiguous substrings of S and T.

a) When does a maximum score AWO contain indels (gaps)? Specify a condition on the scoring function that will enable gaps in the max score AWO and give an example (two sequences, a scoring function, and the maximum score AWO).

Professor Crude suggested the following algorithm for computing a maximum score AWO: First, run the Needleman-Wunsch algorithm to compute a max-score global alignment (S', T') of S and T. Then eliminate all columns (S'_i, T'_i) in the alignment s.t. $\sigma(S'_i, T'_i) <0$. Output the resulting alignment.

b) Prove that Crude's algorithm always returns an AWO.

c) Prove by example that Crude's algorithm does not always return a maximum score AWO. Give an explicit example and explain it in detail – specify the sequences, scoring function, output of Crude's algorithm, and AWO with higher score.

d) Suggest an efficient ($O(nm)$) algorithm for finding the maximum score AWO. Describe your algorithm in detail and analyze its time and space complexity. **Hint:** you might find it useful to modify the input scoring function σ.

e) Prove the correctness of your algorithm. Use formal arguments and avoid hand waving.

f) Suggest an efficient ($O(nm)$) algorithm for finding the longest common non-contiguous subsequence of S and T. The algorithm should return the longest sequence w s.t. w is a non-contiguous subsequence of S and T. Clearly describe your algorithm, analyze its complexity and formally prove its correctness.

Problem 2:

This question deals with the problem of finding common substrings between two sequences S and T. We do this by considering alignments between the two sequences that <u>do not contain gaps, but may contain mismatches</u>. For two sequences, x and y, of the same length we define by $d(x,y)$ the number of positions i in which $x_i \neq y_i$.

a. Start by defining an algorithm that finds completely matching substrings with <u>no mismatches</u>. Describe an algorithm that for every pair i,j of positions along S and T computes the largest number k, such that $\boldsymbol{d(S_{i-k+1..i}, T_{j-k+1..j}) = 0}$. Note that $\boldsymbol{S_{i-k+1..i}}$ and $\boldsymbol{T_{j-k+1..j}}$ are the substrings of length k, of S and T (respectively) that end in positions i and j. Since we require that $\boldsymbol{d(S_{i-k+1..i}, T_{j-k+1..j}) = 0}$, then these substrings should be completely identical. Clearly describe your algorithm and analyze its complexity.

b. Prof. Las wants to modify this algorithm so that it computes longest common substrings with <u>at most two mismatches</u>. In other words, he is interested in computing for every pair of positions i,j along S and T the largest number k, such that $\boldsymbol{d(S_{i-k+1..i}, T_{j-k+1..j}) \leq 2}$. He tries to argue that this can be done using a simple dynamic programming algorithm, meaning that if A[i,j] is the value defined above, then it can be computed using a simple function of A[i-1,j-1], A[i-1,j], and A[i-1,j]. Prof. Las is very passionate about this approach and spreads this idea in the media, but you have a hunch that he may be wrong. Suggest an example of two sequences S and T that disproves Prof. Las' idea. In particular, the longest suffix of S that matches a suffix of T with up to two mismatches cannot be obtained by elongating longest suffixes with up to two mismatches of S and T without their last characters.

c. Suggest an efficient dynamic programming algorithm for computing for every pair of positions i,j along S and T the largest number k, such that $\boldsymbol{d(S_{i-k+1..i}, T_{j-k+1..j}) \leq 2}$. Explain the modification you had to apply to address the issue raised in (b), and clearly describe your algorithm. Provide a complete formal proof for the correctness of your algorithm.

Problem 3:

In Lecture #4, we discussed a measure for the homology of two sequences S and T that sums over all possible alignment of the two sequences (slide 17). Formally, given a model for homology H with parameters Θ_H,

$$\text{sumHomology}(S,T) \;=\; \sum P(A|H,\Theta_H)\ ,$$

where the sum is taken over all valid alignments A of S and T.

a. Describe an $O(nm)$ algorithm that receives two sequence $S_{1..n}$ and $T_{1..m}$ and parameters Θ_H, and computed sumHomology(S,T). Describe your algorithm in detail and argue its correctness. You may assume a linear gap penalty here, by assuming $p_{\text{gap}}=1$.

   Hint: notice that the inductive structure we used when proving the correctness of the algorithm for max-score alignment partitions all alignments based on the identity of the last column. This can be used to sum over all alignments efficiently. Use this idea to write an expression for sumHomology(S,T) as a function of sumHomology(S',T'), sumHomology(S',T), and sumHomology(S,T'), where S' and T' are the prefixes of S and T (respectively) without their last characters.

b. Write a short computer program (in your language of choice) that computes sumHomology(S,T) for the two sequences below:

   S = **ATAAGGCATTGACCGTATTGCCAA**

   T = **CCCATAGGTGCGGTAGCC**

   under Θ_H defined as follows:
   $p(x,x) = 32/160$, $p(x,y) = 2/160$ $(x\neq y)$, and $p(x,-)=\ p(-,x)= 1/160$.

   - Notice that these probabilities are normalized to sum up to 1 when considering all 24 options for possible alignment columns. These values are consistent (up to a constant additive factor) with the scoring function used in the previous homework assignment: $(\sigma(*,*) = \log_2(p(*,*)) + \log_2(20))$

     $\sigma(x,x) = 2$, $\sigma(x,y) = -2$ $(x\neq y)$, and $\sigma(x,-)= \sigma(-,x)=-3$.

   - Make sure that your program's **space requirements are linear** in the short sequence (T in this case). Note that you do not have to output an alignment here, so no need to use Hirschberg's trick that uses the "mid-point" of the alignment.

   - Write down the sumHomology score that you received for S and T, and submit your code as appendix.

c. **Food for thought:** run your program with the same Θ_H and sequence T, but change sequence S to be S = $\mathbf{T}^n$**GATTAAGCCAAGGTTCCCCG**, where $\mathbf{T}^n$ is a sequence of n consecutive **T**'s. Increase n until you get a sumHomology value of 0. If your programming language uses double precision floating point representation (8 bytes), this will be obtained around n=150. Now run the program you wrote in the first assignment to compute the global alignment of two sequences. What is the score of the most probable alignment? (do not print out the DP matrix or the alignment, just compute its score). If you were to run the NW algorithm with products of probabilities instead of sums of log-probabilities, you would have also received a probability of 0. This demonstrates why in probabilistic models we represent probabilities in log-scale. The logarithms give you improved accuracy when representing near-zero probabilities, which are typical of very large data sets.

Explain briefly why this can be done when computing the probability of the most likely alignment but cannot be easily done when computing sumHomology.

Note: we will learn a neat trick that will allow us to use log-probabilities also in this case.

Submission Instructions:

- Submit your work on the course **Moodle website** by Sunday, Dec 13 @21:00.
- Type your solutions or write legibly and scan. **If you scan, make sure the scan came out fine.**
- **Submit your work in pairs!** One student should submit a solution file with both of your student IDs specified. The other student should submit a simple text file with your two student ids, to help us match back the grade to both of you.
- If you consult with other pairs on ideas, specify their names clearly on the first page and make sure that they **acknowledge your collaboration** as well.
- You have two weeks to complete the assignment. **Plan your time wisely**. Extensions due to special circumstances, will be granted only upon **request by e-mail at least 48 hours** before the deadline. No last minute extensions!
- Please post any questions that you have on the course Piazza website: https://piazza.com/idc.ac.il/fall2020/cs3571/.