Fall 2014

ECS 124A Theory and Practice of Bioinformatics

Lab/Homework Assignment #2

(100 points total)

Instructor: Ilias Tagkopoulos (iliast@ucdavis.edu)

TA: Linh Huynh (huynh@ucdavis.edu)

Please address all technical questions to the TA directly

Assigned: 10/14/2014

Due: 10/28/2014, 1:30pm

Please note that a 20% penalty per day will be in place for late submissions.

Scope: The goal of this lab is to (a) introduce you to dynamic programming (DP) and perform a DP alignment computation by hand, (b) modify a Perl alignment program to incorporate user-defined match and penalty scores, (c) learn how to use regular expressions in Perl, and (d) learn how to compute longest common subsequences between two strings.

PART I: Local and global alignment

Exercise 1 *(20 points)*

Consider two following strings:

AATCCGGTACA
ATCCTCGTTAA

1.1. What is the total number of global alignments for these two strings?

1.2. Compute (i.e. layout and fill in) the dynamic programming alignment table that computes the value of the optimal alignment of these two strings; use the following objective function:

maximize {#matches - #mismatches - #spaces}

You must do this BY HAND. It is a bit tedious, but it will force you to pay attention to the recurrence relations used to compute the table.

1.3. Now find at least one optimal trace-back path in the table and write out the associated optimal alignment. Use one of the tools (*see additional comments in the end*) to check that your alignment is in fact optimal. Turn in a print-out of the output for your alignment (from the program of your choice).

1.4. Find the local alignment with the highest value using the **Smith-Waterman algorithm** that we learned in class. How many local alignments of non-zero value do we have?

Exercise 2 *(10 points)*

Download Perl program `needleman.pl`. Be sure you have it running, and try to understand how it works. This is a Perl version of the **Needleman-Wunsch alignment algorithm** that we studied in class, using the dynamic programming recurrence relations, instead of an alignment graph. Even though you have not learned all the Perl constructs used in this program, you should be able to understand what the different parts of the program does, based on you prior exposure to some programming language.

Modify the program so that it asks the user for a match value *V*, a mismatch cost *Cm*, and an indel cost *Im*. It should read the inputs from the keyboard and assigns them to variables. Modify the program so that it finds the maximum value of any possible alignments of the two input strings, where the objective function is

maximize{*V* * (#matches) - *Cm* * (#mismatches) - *Im* * (#indels)}

Exercise 3 *(10 points)*

Extend the `needleman.pl` program of exercise 2 to incorporate gaps. Note that you will have to modify the way spaces are handled.

PART II: Perl and Regular Expressions

Exercise 4 *(25 points)*

Please read `Introduction_to_Perl_2.pdf` *by D. Gusfield and K. Stevens*. Do exercises 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.14 (Please number them as 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.14 in your report ☺). Read about lists and arrays in *Johnson* or elsewhere.

Exercise 5 *(15 points)*

In the next exercises you will work with short regular expression programs. Run the first three programs (`ex5_program1.pl`, `ex5_program2.pl`, and `ex5_program3.pl`) and understand how the regular expression works in them. In the starting comments of each program, you are asked to run the program in certain ways. Be sure you do those, but please **DON'T turn in scripts of those executions**.

5.1. Program `accession.pl` is very similar to `ex5_program3.pl`, but has some extensions. Modify `accession.pl` so that it only prints the accession numbers it finds, and prints each one on a separate line. That is, it prints a list of accession numbers it finds in the input file, but does not print anything else. Use `ex5_testfile.txt` as the test input file.

5.2. The meaning of [,.;:?] in the regular expression is that the digits of an accession number must be followed by any ONE of the six characters listed between the brackets [...]. So this is an OR of the six characters. Now in the program `ex5_program2.pl` (it looks for a DNA string in an input line) we used "|" to indicate an OR. That regular expression had (A|T|C|G|a|t|c|g). Replace that with [ATCGatcg] to see if the program works. Script the result. Use `ex5_testfile.txt` as the test input file.

PART III: Longest Common Subsequence

Exercise 6 *(20 points)*

6.1. By setting *V* to 1, and *Cm* and *Im* to zero, the program `needleman.pl` (*see Exercise 2*) will produce the length of the **longest common subsequence** (LCS) between the two sequences. That is the alignment that simply maximizes the number of matches that can be obtained, without regard for how many space and mismatches are involved. The LCS between two strings is sometimes taken as a measure of the similarity of two strings. By letting your figures make up "random" strings of length 20 say, and computing the LCS of those two strings, try to find in this way the expected length of the LCS of two random strings.

6.2. In this exercise, you will determine as best you can what is the correct expected length of the LCS is. `Randomdna.pl` is a Perl program to produce random DNA strings. Get this program running and make sure you understand how it works. Then modify it so that it asks the user how many random strings to produce, and then it generates that many strings and writes them into a file. The main modification is to put a loop around part of the existing program.

Now generate gobs (*that is a technical term for a number that is at least 10*) of random strings of length 10 each, and using your modified Needleman-Wunsch from Exercise 6.1, compute the length of the LCS between the first of your strings and each of the other ones. Compute the average LCS length obtained. Then repeat with strings of length 20, then 50, then 100, 200. What do you observe about the average LCS length? If you are comfortable enough with Perl and programming at this point, you may want to put everything together into a single program that generates the random strings, computes the LCS lengths, computes and reports the averages. This may save you a lot of effort compared to running programs over and over again. In fact, if you do have a single program, then for each string length you should generate at least 100 random strings.

ADDITIONAL COMMENTS

A comprehensive list of alignment tools can be found at:

http://en.wikipedia.org/wiki/List_of_sequence_alignment_software

and:

http://molbiol-tools.ca/Alignments.htm

emboss is The European Molecular Biology Open Software Suite

http://emboss.sourceforge.net/

has links to online tools at:

http://emboss.sourceforge.net/interfaces/#web