7.36/7.91/20.390/20.490
Computational and Systems Biology

Homework 5

Due: Friday, April 13, 2007

General

Files to be downloaded for problem sets can be found on the course website at: http://stellar.mit.edu/S/course/7/sp07/7.91J/homework/index.html

Python Scripts

All Python scripts must work on athena using /usr/athena/bin/python. Scripts using input files must be able to run if placed in a directory also containing these input files, and should not alter the contents of the host file system (i.e. should not add, delete, or modify any files) unless explicitly requested by the question in the problem set. Electronic submissions are subject to the same late homework policy as outlined in the syllabus and submission times are assessed according to the server clock.

DO NOT SUBMIT YOUR OUTPUT FILES - CODE ONLY PLEASE!

NO HTML FILES WILL BE ACCEPTED AS CODE

PLEASE DO NOT ZIP YOUR FILES

Written Portion

The written portion of the problem set must be turned into the **Biology Education Office, 68-120, by 3 PM** on the due date. Late problem sets should be turned into the same location by noon two work days following the due date (see the syllabus for late homework policy).

Problems

1. **DNA Motif Discovery**

One of your collaborators is interested in RNA signals involved in splicing in the fission yeast *Schizosaccharomyces pombe*. You happen to mention that you're taking this computational biology class, and she quickly recruits you to help her find some motifs! Download the file **pombeIntrons.fa** from the course website which contains 250 intron sequences that she has sequenced.

(a) Browse through these sequences. Do you see any clear motifs by eye? (not graded)

(b) Now try the motif-finding tool MEME (http://meme.sdsc.edu/meme/meme.html). Run MEME using these sequences as input and the following settings: zero or one motifs per sequence, minimum and maximum width 7, maximum number of motifs 3, and search given strand only (check box under Optional). Of the three motifs returned (you requested MEME to return a maximum of 3 motifs), which are you most confident in and why?

(c) Based on the motif positions in the input sequences (see block diagrams in output) and what you've learned about splicing in lecture, what do you think these motifs represent?

(d) Take a look at the information content plots for each motif. Which of the information content values for positions in motif 2 are not possible for a position in a PSPM using the formula for calculating information content introduced by Prof. Yaffe?

$$I(bits) = H_{before} - H_{after}$$

with $H = -\sum_k p_k log_2(p_k)$

(e) Information is sometimes defined as the *relative entropy* (as in your MEME output):

$$I(bits) = \sum_k p_k log_2(p_k/q_k)$$

where p_k is the probability of k in the observed motif (*foreground*) and q_k is the probability of k based on the background distribution. Using the background letter frequencies reported in the **command line summary** in your MEME output, what is the maximum value that the relative entropy can attain at a position in any PSPM?

(f) Each motif has a corresponding PSPM and a PSSM. What is the relationship between these two matrices?

(g) Using the PSSM for motif 1, what is the score for an occurrence of TACTAAC?

(h) Based on the score above, how much more likely is it that the sequence TACTAAC comes from the motif model than from the background model? (*hint*: values in the PSSM have been multiplied by 100 and are in log base 2).

2. **DNA Motif Searching**

For this question, you will write a python script that finds occurrences of an input motif in an input DNA sequence - a very common task in computational biology. Let's use some of the motifs, represented by PSSMs, generated by MEME in Question 1. You can download the PSSMs from the course website (**motif_1.txt** and **motif_2.txt**) together with an *S. pombe* gene sequence file, **niftySeqToSearch.fa**.

S. pombe genes typically have 1 or 2 introns that are relatively short, with a mean length of 81 nts, and are very rarely larger than 350 nts. The introns in the pombeIntrons.fa file from Question 1 are typical for this organism.

(a) Using a threshold of 0.94, how many occurrences of motif 1 do you find (**python motifFinder.py motif_1.txt niftySeqToSearch.fa 0.94**)?

(b) Now try running your code using motif 2 (**python motifFinder.py motif_2.txt niftySeqToSearch.fa 0.94**). How many occurrences of motif 2 do you find?

(c) Based on what you deduced about the function of these motifs in Question 1, do you expect all of these motif occurrences to be biologically active (Yes/No)? Why/why not? (You can assume that your motif searching algorithm has a sensitivity of essentially 100%, such that you never miss an authentic site).

(d) Briefly describe how this initial script might be expanded to build a simple *S. pombe* intron finder, i.e. a program that predicts the locations of introns in an input primary transcript or genomic sequence (limit your answer to 1 or 2 sentences).

The details:

A given sequence (of length 7 for these motifs) can be evaluated by summing the scores of each nucleotide in the sequence based on the relevant position in the PSSM. Then the total score is compared to some predetermined cutoff.

For this problem, your cutoff value is determined as follows: (i) calculate the maximum score possible, $maxScore$, by adding up the maximum values for each position in the PSSM (**the rows in the PSSMs output by MEME represent positions and the columns correspond to nucleotides**). (ii) calculate the minimum score possible, $minScore$, by adding up the minimum values for each position in the PSSM. (iii) calculate the cutoff score as $cutoff = minScore + threshold * (maxScore - minScore)$, where $threshold$ is the value between 0 and 1 input to your script. Sequences scoring above this cutoff are reported along with their position in the sequence, score (based on PSSM), and score as a fraction of the best score (as above).

There is a starter script (**motifFinder.py**) available on the course website that provides some useful functions. ## indicates pseudocode where you are to add python code, but # is simply a comment line. You may ignore this file and write your own from scratch, however, such code that (i) does not run, (ii) does not handle the input filenames from the command line, (iii) mis-parses the motif PSSM files, or (iv) does not produce correctly formated output will be **significantly** penalized. Ensure you compare the formatting of your output to that of the sample output provided, **motifFinderOutputSample.txt** (generated by searching for motif 1 in a different sequence).

The rules:

Your script should be named **motifFinder.py** and should take three arguments in the following order: (1) name of motif PSSM text file, (2) name of fasta format sequence file to search against, (3) the threshold value (value between 0 and 1). You will be penalized if you assume a different order of arguments.

Your script should output a file named **motifFinderOutput.txt** which should be formated exactly like the sample output file **motifFinderOutputSample.txt**. You will be penalized for different output file names, formats, or for not writing your output to a file.

3. **Finding High-Scoring Local Alignments**

   The theory behind BLAST statistics is based on the properties of the highest scoring ungapped alignment between two sequences, often called the maximal segment pair (MSP). In this problem, you will implement a simple algorithm that is guaranteed to return the MSP and then explore how the MSP changes with the mismatch penalty. (The BLAST algorithm itself is not guaranteed to find the highest scoring ungapped alignment, but rather implements a computationally efficient heuristic that has been shown to find almost all such alignments.) For this problem, download **query.fa** containing a query sequence and use **niftySeqToSearch.fa** as your database sequence (same file as used in Question 2).

   (a) With pen and paper, first plot the cumulative score for the alignment of ATCCAGTGAGGTAAC (a query) with TCGAATCAAGGTATT (a database, albeit a small one) using a match score of +1 and a mismatch score of -1. Plot the cumulative score on y-axis and the position in the query sequence along the x-axis (an example plot is included below). **Circle** the points corresponding to the MSP and **write** out the corresponding local alignment.

   (b) Using your script, findMSP.py (see details below) and a mismatch score of -1, what is the MSP score and length (**python findMSP.py query.fa niftySeqToSearch.fa -1**)?

   (c) (i) How do the MSP score and length change when you run your script using a mismatch penalty of -2 and then -3 (using the same input sequences)? (ii) Is this what you expect (Yes/No)? Why/why not? (Limit your answer to 1 or 2 sentences).

The details:

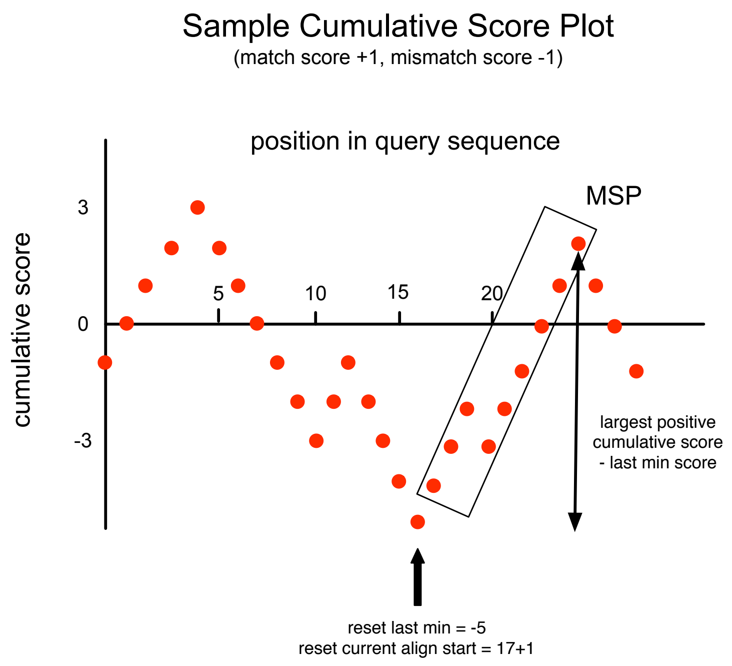
The algorithm is quite simple. Walk through the database, calculating the score of local alignments of the query sequence with each possible offset from the beginning of the database, by accumulating a score that depends on comparing successive nucleotides of the query sequence to successive nucleotides of the database sequence. We will only consider **ungapped** alignments (as done by the original BLAST algorithm widely used in the 1990s).

First, initialize global variables for the score of the best alignment, *maxScore*, and the start position of the best alignment in the query, *alignStartInQuery*, and in the database, *alignStartInDB*. Then for each offset in the database, initialize a cumulative score, *currScore*, a minimum score, *minScore*, and the start position of current alignment, *currAlignStartInQuery*. Score each position in the query against the corresponding position in the database (ungapped alignment). As you do this, the cumulative score may become positive or negative (as shown in the figure below). Each time you encounter a new minimum score, record this score, *minScore*, and update *currAlignStartInQuery*. Each time the *currScore* increases, check to see whether the difference between the *currScore* and the *minScore* is a larger positive number than the current *maxScore*, and if so, update *maxScore*, *alignStartInQuery* and *alignStartInDB*. This is the score for the MSP found so far. The final MSP can thus be recognized as the region of the ungapped alignment producing the largest positive change in the cumulative score.

Note, the MSP does not start at the position with the minimum score, but rather at the neighboring nucleotide (i.e. where the alignment score begins to improve again).

Use a nucleotide match score of +1

A starter script (**findMSP.py**) is available on the course website. ## indicates pseudocode where you are to add python code, but # is simply a comment line. You may ignore this file and write your own from scratch. However, see Question 2 for the rules on code and output.



The rules:

Your script should be named **findMSP.py** and should take three arguments in the following order: (1) name of the query fasta file (2) name of fasta format sequence file to search against (3) the mismatch penalty (integer value). You will be penalized if you assume a different order of arguments.

Your script should output a file named **findMSPoutput.txt** which should be formatted exactly like the sample output file **findMSPoutputSample.txt**. You will be penalized for different output file names, formats, or for not writing your output to a file.