6.047/6.878/HST.507
Fall 2015 Problem Set 1: Aligning and Modeling Genomes

Due: Wednesday, September 30 at 8pm (submit on the course website)

When you submit this pset, please turn in the following files in a zip file:

- Your answers to the problem set questions in a pdf file
- A directory named "code" with all the code you are submitting
- A directory named "data" with all other results you are submitting

In your answers to the questions please refer to the appropriate file name where your results/code for that problem are located.

1. **Evolutionary distances of orthologs and paralogs**

   In this problem, you will implement the Needleman-Wunsch algorithm for pairwise sequence alignment, apply it to the protein-coding sequences of related genes from several mammalian genomes, and use the results to learn about their evolution.

   (a) On the class web site, we have provided a python skeleton program `ps1-seqalign.py`, which you will complete. We provide a traceback routine, but you will write the code to fill in the score and traceback matrices. The skeleton program specifies a substitution matrix and gap penalty. If you so choose, you may rewrite the program in any programming language. Please submit (1) the portion of the code that you wrote; (2) an optimal alignment of the two sequences `CTAAGTACT` and `CATTA`, and the corresponding score matrix F with the optimal path indicated; and (3) the score of the alignment of the human and mouse HoxA13 genes, which we also provide on the web site.

   The command to run the program is:
   `python ps1-seqalign.py <FASTA 1> <FASTA 2>`

   The Hox cluster is a set of genes that are crucial in determining body plan formation during embryo development. They are found in all bilateral animals, in species as distant as the fruit fly. The fruit fly has one Hox cluster, while most vertebrates have four. It is thought that vertebrates have undergone two rounds of whole-genome duplication, giving rise to four Hox clusters from the ancestral one, although the hypothesis remains controversial.

   In the remainder of this problem, you will use your Needleman-Wunsch alignment program to analyze the sequences of several Hox genes, and estimate the date of the most recent vertebrate whole-genome duplication. In particular, we are interested in using the N-W alignment score as a distance metric between two sequences.

   (b) Make minor adjustments to your alignment program so that the score it computes can be interpreted as a distance metric. That is, the score of a sequence aligned with itself should be zero, all scores should be non-negative, and sequences that are more dissimilar should be given a score with a greater magnitude. Describe the changes you made in your handin; no code is necessary.

   (c) Apply your modified program to compute a distance between the human HoxA13 gene and the mouse HoxA13 gene.

   (d) The modern mammalian genes HoxA13 and HoxD13 arose from a single ancestral gene by whole-genome duplication, long before the human-mouse divergence. We provide the sequences of the human and mouse HoxD13 genes on the web site. Given that the fossil record shows that human and mouse diverged about 70 million years ago, use your distance metric and your results from part (c) to estimate the date of the whole-genome duplication that gave rise to HoxA13 and HoxD13. Make sure to state the assumptions underlying your estimate.

2. **Sequence hashing and dotplot visualization**

As you have seen in problem 1, sequence alignment is a quadratic time algorithm. Full sequence alignment is therefore only feasible for sequences near the length of a single gene. To align larger regions of a genome, heuristic approximations are typically used. In this problem, you will use hashing techniques to guide the alignment of a 1 megabase (1 million nucleotides) region surrounding the HoxA cluster in human (`human-hoxa-region.fa`) and mouse (`mouse-hoxa-region.fa`). You will use dotplots to visualize the performance of various hashing methodologies.

The code provided (`ps1-dotplot.py`) finds all 30-mers in the human that also appear in mouse. On a dotplot, each of these matches is represented as a single dot at (x, y), where x is a coordinate for the beginning of a 30-mer in human and y is a coordinate for the beginning of a matching 30-mer in mouse. We provide a plotting function that will produce dotplot images. The format of the image is determined by the file extension (*.ps, *.png, *.jpg). There is also code for heuristically judging the *specificity* of the matches (the fraction of matches that occur near the diagonal of the dotplot).

(a) Run the script unchanged to generate a dotplot for all exact matching 30-mers. It must be run in the same directory where it is located, since it also requires `utils.py` and `plotting.py`. This script also requires `gnuplot` which is available on athena.

Before running the script, do:
`athena% add gnu`

The command to run the program is:
`athena% python ps1-dotplot.py <FASTA 1> <FASTA 2> <PLOT FILE (*.ps, *.png, *.jpg)`

Alternatively, you can run on your own machine if you have gnuplot installed.

Describe what you see. How many hits are there and what percentage fall near the diagonal? Do you observe any structure in the off-diagonal hits? What types of genomic elements could cause such a pattern? Why are matches that are close to the diagonal more likely than off-diagonal matches to represent "correct", or orthologous, alignments?

(b) Make the following modifications to the script and report how the plot changes qualitatively and quantatively (how many hits, what percentage are near the diagonal). Also briefly describe how you implemented each change.

i. Modify the script to find all *exact* matching 100-mers
ii. Modify the script to find all 60-mers that match every *other* base
iii. Modify the script to find all 90-mers that match every *third* base
iv. Modify the script to find all 120-mers that match every *fourth* base
v. Modify the script to find all 100-mers that allow *at most two* mismatches in each contiguous block of six bases. Instead of producing a plot, focus on describing how you would implement this modification.

(c) Although parts a, b.ii, b.iii, and b.iv require the same number of matching bases ($30 = 60/2 = 90/3 = 120/4$), one of them is more specific to the diagonal. Explain why this might be so.

(d) Explain the trade-off you see between number of hits near the diagonal (sensitivity) and the percentage of hits near the diagonal (specificity). How is the trade-off affected by the hashing parameters?

(e) Modify the script to also detect inversions. An inversion occurs when a stretch of DNA is spliced out and reinserted in reverse orientation. For example,

CGT[GATT]AGA

⇓

CGT[AATC]AGA

The `human-hoxa-region-modified.fa` file contains a version of the Hox region with an artificial inversion. Use the dotplot to locate the inversion in human. (Note: ignore the sensitivity measure, and only test all sizes necessary to detect the inversion.)

3. **HMMs for GC-rich regions: State durations and limitations**

An important use of HMMs is to decode or parse a genome into its biological components: exons, introns, regulatory regions, etc. In this problem, we will examine how the accuracy of HMM predictions is affected by certain inherent properties of the model.

In this problem, we will use GC content (the fraction of letters that are a C or a G) to classify the genome into high-GC regions (on average 60% G or C) and Low-GC regions (on average 60% A or T). These have different melting temperatures, different replication times across the cell cycle, and different gene density. They have also been hypothesized to have different evolutionary origins (see *isochores*), but this hypothesis remains controversial.

Our simple model requires only two states. We have provided a program, `viterbi.py`, which you will complete and use to decode several artificial genomes, and then compare the resulting predictions of High-GC and Low-GC regions to a provided (correct) annotation. More details about this program are included at the end of the problem.

(a) In most HMMs, the self-loop transition probabilities a_{kk} are large, while the transition probabilities between different states a_{kl} are small. Once a Markov chain with these transition probabilities enters state k, it tends to stay in state k for a while. The *state duration* is the total number of consecutive steps at which the Markov chain stays in the same state, before switching to another state (e.g. transitioning into state k and then transitioning out to a different state is a state duration of 1). What is the expected (mean) state duration of state k as a function of the transition probability a_{kk}? What is the distribution of state durations $P(D_k = d)$?

(b) Complete the implementation of the Viterbi algorithm in `viterbi.py`. Based on the HMM parameters hard-coded into the program, what are the expected state durations for High-GC and Low-GC regions? Apply the finished program to the data file `hmmgen`, which was generated using the same HMM, and verify that your program achieves ∼83% accuracy.

(c) Now apply your program to the files `mystery1`, `mystery2`, and `mystery3`. How do the (correct) state duration distributions in the mystery sequences differ and what do they have in common? What accuracy levels does your HMM achieve on these sequences? How does each Viterbi-predicted state duration distribution differ from the correct distribution? (You do not need to include the plots in your solutions.)

(d) Would re-training the HMM parameters according to the procedure described in lecture, using the correct annotations as training data, improve the accuracy of the Viterbi annotation for the mystery sequences? Why or why not?

(Extra credit) Try to make the decoder perform better by adjusting the hard-coded model parameters. If you succeed, can you explain why?

(e) As you are now aware, the length distribution of genomic elements can strongly affect the predictive accuracy of an HMM used to decode them. Unfortunately, most elements in real genomes do not follow the length distribution you derived in part (a). By reading the following paper (or any other sources), describe how the gene finder GENSCAN addresses this issue. How is it possible, algorithmically, to use state duration distributions that differ from the one you derived in part (a)?

Burge C, Karlin S. Prediction of complete gene structures in human genomic DNA. *J Mol Bio* 268(1):78-94, 1997.

Details about `viterbi.py`

Note that like in problem set 1, the plotting portion of this code relies on gnuplot. Therefore, you should run this on athena after running "add gnu" if you want plotting to work.

The nearly complete program `viterbi.py` performs the following:

- Reads in a data file containing a DNA sequence and an authoritative (correct) annotation, consisting of a string of pluses and minuses, specifying where the High-GC and Low-GC regions are, respectively.
- Calculates the base composition of the High-GC and Low-GC regions, calculates the mean length of High-GC and Low-GC regions, and plots a histogram of the lengths of the High-GC and Low-GC regions. (All with respect to the authoritative annotation.)
- Performs Viterbi decoding on the DNA sequence, using a hard-coded HMM designed to detect High-GC and Low-GC regions. (This is the part you will complete.)
- Calculates the base composition of the High-GC and Low-GC regions, calculates the mean length of High-GC and Low-GC regions, and plots a histogram of the lengths of the High-GC and Low-GC regions. (All with respect to the Viterbi annotation.)
- Calculates the accuracy of the Viterbi decoding, defined as the percentage of predicted plus and minus states that match the authoritative annotation.

4. **Final project preparation** This course aims to both introduce you to the field of computational biology, and to enable you to become active members of its research community. This involves being able to plan, set up, carry out, and report your independent research, which will be the goal of the final project. While the bulk of the work for the final project will be carried out during the second half of the term, it is important to begin thinking about possible projects early on.

   To begin the process of identifying a good project that matches your background and your interest, this part of the first problem set asks you to begin that process by writing a paragraph or two on each of the following questions:

   (a) **Skill set:** Detail your academic background, and in particular, your computational/algorithmic training and your biological knowledge/experience. We encourage you to select a project that matches your skills, and benefits from your strengths. Certainly, you will learn new areas and new applications of your skills, but you are more likely to accomplish a successful project if you think carefully about your strengths, and perhaps identify partners that complement your background early on.

   (b) **Research experience:** Outline your previous research experience, if any. This can be in any field and can be either independent research or class-related research. Think back at the projects you have accomplished over the years, list them here, and give a brief description of the kind of skills that you gained in accomplishing them.

   (c) **Interests:** What areas of computational biology do you find the most interesting for your own research? Read ahead on the syllabus, and find the lecture topics that seem most interesting to you for a final project. You can find more information about each of these topics on the web, or by pulling up the slides for these lectures from previous years. Stepping back and taking a look at the whole term ahead of time will help you pick a topic without biasing yourself to only consider early lectures.

   (d) **Project types:** Think ahead about the type of project you might prefer, for example: algorithmic, theoretical, tool-building, analysis, or method development. This can help you identify areas that are more inclined to the type of project that youre looking for, and also pair up with partners that share similar interests, or complementary interests on the same topic.

   A unique aspect of computational biology is how collaborative the field is. In order to learn more about your classmates, identify potential partners for the final project, and potentially even collaborators that will come in handy for the longer term, we ask you to fill out the information above (or an abbreviated version thereof) in an MS Word document and upload it with your homework. We have provided a template `StudentProfileTemplate.doc`. Note that these summaries will be shared among

your classmates (if you are uncomfortable with this, please contact the course staff). Name your profile `LastnameFirstname_Profile.doc`. There will be more files that complement this through the term, so maintaining a consistent naming scheme is important.

6.047 / 6.878 / HST.507 Computational Biology
Fall 2015