7.36/7.91/BE.490
Homework 2
Due March 11 at 1:00 PM

Note: Please see the class website for a handout describing how to submit your programming problems electronically. Also a list of useful programming hints is included at the end of this assignment.

1. Paleogenomics I - BLAST searches with nucleotide sequences

Your lab has developed a revolutionary technology to extract DNA from dinosaur fossils. In order to learn about the evolutionary history of dinosaurs and the origins of flight, you extract and sequence DNA from a fossilized pterodactyl.
This sequence is stored in the file dino1.fa.

a. Do a BLASTN search of the 'dino1.fa' sequence against the nr database. What is the top hit with the default settings? Does it look real? Why/why not?
b. Now change the mismatch penalty from the default (-3) to –1 and do the same BLASTN search. What is the top hit now? What is the E-value of this hit? Qualitatively, how did the alignments change from part a? Is this what you expect when reducing the magnitude of the mismatch penalty?
c. Now do a BLASTX search with dino1.fa. What is the top BLASTX hit? What are the E-values and bit scores of this hit? Does it look real? What type of organism is this? What could this mean for the evolution of flight in dinosaurs?
d. Explain why the BLASTN and BLASTX results are so different.

2. Paleogenomics II – the effect of introns on BLAST searches

You obtain a new pterodactyl sequence to study in the file 'dino2.fa'. You try BLASTN and BLASTX but can't find any related sequence in the nr nucleotide and protein databases (do the searches to convince yourself). Because this sequence comes from a very gene-rich part of the pterodactyl genome, you suspect that there may be a gene hidden in the sequence which for some reason is undetectable by BLAST. To explore this possibility, you decide to investigate the splice site motifs recognized by the pterodactyl splicing machinery to see whether you can extract the exons from the dino2.fa sequence. You do some more sequencing and identify a set of 56 pterodactyl genomic DNA fragments that have BLASTX hits to known proteins. You infer these hits are likely to contain exons and splice sites. Since BLASTX hits often do not correspond precisely to the boundaries of exons (e.g., because of amino acid changes near splice sites resulting in truncated hits, or spurious hits to translates splice sites/introns resulting in extended hits), you construct a dataset including ~25 bases on each side of the START of the BLASTX hit and call it '3primesplicesites.txt' (for finding the 3′ splice site motif) and another dataset including ~25 bases on each side of the END of the BLASTX hit, which you call '5primesplicesites.txt' (for finding the 5′ splice site motif).

a. Run MEME (http://meme.sdsc.edu/meme/website/meme.html) on the two datasets to find a consensus motif common to the 56 sequences that dinosaurs may have used as 5′ and 3′ splice sites. What motifs are found? Write the consensus for each motif using the standard one-letter code shown below:

N=A,T,G,C		Y=C,T	R=G,A
K=G,T	S=C,G	W=A,T	M=A,C
B=C,G,T	V=A,C,G	D=A,G,T	H=A,C,T

Note: A glitch in the output from MEME for the 'Multilevel consensus sequences' may not print the fourth nucleotide at a given position in the motif. Therefore, examine the 'Simplified pos.-specific probability matrix' or the 'Information content' graph to determine whether any of the positions in the motif can have all four nucleotides.

b. Write a Python program that reads in the two consensus sequences, finds putative introns in your 'dino2.fa' sequence, and prints out the predicted spliced mRNA. Your program should accept two file names as command line arguments. The first is the name of a file with two consensus sequences – each on a line by itself (the 5′ and 3′ splice site consensus sequences respectively). A consensus sequence is any strings composed of characters in the table above as well as A, G, C, and T. The second argument is the name of a file containing the DNA sequence of interest in FASTA format. You may not assume that the sequence in this file will entirely be on one line (many of you made this assumption in the last homework). See http://ngfnblast.gbf.de/docs/fasta.html for a description of the FASTA format. Your program should search for the 5′ and 3′ splice site consensus sequences and print out the predicted spliced mRNA. Assume that, unlike many modern organisms, in dinosaurs the first nucleotide of the intron is the 5′ nucleotide of the 5′ splice site motif and the last nucleotide of the intron is the 3′ nucleotide of the 3′ splice site motif. Name your program splicing.py and submit it online. Hint: from the spacing of the BLASTX hits in pterodactyl genomic sequences, you have inferred that pterodactyl introns are always ~300-600 bases long. It is always a good idea to implement error checking in your programs, but it is not required for this assignment. You can assume that your program will always be called with exactly two file names and that the files contain exactly what is outlined above. See the website for a sample run of this program (splicing.run).

c. Perform a BLASTN with your predicted spliced exon sequences. Do you find any plausible hits?
d. Perform a BLASTX search. Do you find any plausible hits now? If so, explain why you didn't find these hits in your original search (before removing the introns).

3. Motif finding

Download the file 'pombe.fa' from the course website. This is a file containing the 3' ends of 75 introns from the fission yeast *Schizosaccharomyces pombe* in FASTA format. Each sequence represents the bases –35 to –6 relative to the 3' splice junction, the region of the intron that is expected to normally contain the site of branch formation in this organism. (Part a of question not graded.)

a. Examine the sequences in the file. Can you see any motifs shared by most or all of the sequences by eye?
b. Now run the sequences through the Gibbs Motif Sampler at http://bayesweb.wadsworth.org/gibbs/gibbs.html. Try a motif width of 7 and leave the field specifying the number of different motifs blank to engage the default Site Sampler behavior (see the manual available at http://bayesweb.wadsworth.org/gibbs/bernoulli.html for details). What consensus motif does it find? Run the same sequences through the server 4 more times. Does it always produce the same consensus motif sequence? Are the base frequencies always the same? Explain.
c. Now run the sequences through MEME at http://meme.sdsc.edu/meme/website/intro.html. Use the default parameters. How do the consensus motif(s) compare to your results from the Gibbs Motif Sampler? Explain why your results are either different or the same. Notice that MEME gives the information content of the discovered motif(s). What is/are the information content(s)?
d. Write a python program, which takes as input the name of a file containing an alignment of arbitrary length containing an arbitrary number of aligned occurrences of a particular DNA motif and computes the weight matrix model (position-specific base frequencies) as well as the

Shannon Entropy and the information content in bits at each position. A sample input file is available on the website as 'motifs.txt'. Assume a uniform background distribution of nucleotide frequencies (25% frequency for each base). Also, calculate the total information content of the entire motif.

e. Extend your python program so that it also computes the 'bit score' (see formula given in lecture) of each provided motif sequences in the input file using the frequencies in your weight matrix model. Print the sequences and their scores. Also compute the mean score. How does the mean bit score compare to the calculated information content of your motif?

f. Based on your calculated information content of the motif, how many times would you expect the motif to occur in a random DNA sequence of length 500,000 base pairs? Search for the motif in the random DNA sequence provided in 'random.fa' using a cutoff score of 9.8. How many times do you find this motif? How do these two numbers compare? Discuss the validity of your prediction. (We suggest that you implement the search in your python program but we will not be testing this aspect of your program explicitly.)

Note:
For this problem you may assume that each line contains one DNA sequence with no spaces. All sequences are of the same length (since they are part of the same motif alignment derived from a Gibbs sampler run) and there are an arbitrary number of lines. It is always a good idea to perform error checking in your code, but for this assignment it is not required. You can assume that the program is called with exactly one argument, which is the name of an existing file, which indeed contains a list of aligned DNA sequences (i.e. no errors on the part of the user). The following is a sample run of such a program. Name your program 'motif.py' and submit it online.

```
[Computer:~/be490] user% python motif.py motifs.txt
Frequency Matrix:
A 0.067 0.627 0.000 0.000 0.893 1.000 0.000
T 0.773 0.240 0.120 1.000 0.027 0.000 0.133
G 0.093 0.120 0.000 0.000 0.080 0.000 0.000
C 0.067 0.013 0.880 0.000 0.000 0.000 0.867

Position 1: entropy = 1.127, information = 0.873
Position 2: entropy = 1.367, information = 0.633
Position 3: entropy = 0.529, information = 1.471
Position 4: entropy = 0.000, information = 2.000
Position 5: entropy = 0.576, information = 1.424
Position 6: entropy = 0.000, information = 2.000
Position 7: entropy = 0.567, information = 1.433

Total Information Content = 9.834

Sequence AGCTGAC has score s = 2.999
Sequence GACTAAT has score s = 6.650
Sequence GGCTAAT has score s = 4.266
Sequence CATTAAC has score s = 5.991
Sequence CTCTAAC has score s = 7.481
Sequence TTTTAAC has score s = 8.142
Sequence TACTAAC has score s = 12.401
Sequence TACTAAT has score s = 9.701
Sequence TACTAAC has score s = 12.401
Sequence TGCTAAC has score s = 10.017
Sequence TATTAAC has score s = 9.527
Sequence TTCTAAC has score s = 11.017
Sequence TTCTAAC has score s = 11.017
Sequence TGCTAAC has score s = 10.017
Sequence TTCTAAC has score s = 11.017
Sequence CACTAAC has score s = 8.865
Sequence TGCTAAC has score s = 10.017
Sequence GACTAAC has score s = 9.351
Sequence TATTAAC has score s = 9.527
Sequence CGCTAAC has score s = 6.481
Sequence AACTAAC has score s = 8.865
Sequence TACTAAT has score s = 9.701
Sequence AACTAAC has score s = 8.865
Sequence TACTAAC has score s = 12.401
Sequence TACTTAC has score s = 7.335
Sequence TACTAAT has score s = 9.701
Sequence TACTAAT has score s = 9.701
Sequence TTCTAAC has score s = 11.017
Sequence TTCTAAC has score s = 11.017
Sequence CGCTGAC has score s = 2.999
Sequence TACTAAT has score s = 9.701
Sequence TACTAAT has score s = 9.701
Sequence TACTGAC has score s = 8.920
Sequence TACTAAC has score s = 12.401
Sequence TACTGAC has score s = 8.920
Sequence TGCTAAC has score s = 10.017
Sequence AACTAAC has score s = 8.865
Sequence TTCTAAC has score s = 11.017
Sequence GACTAAC has score s = 9.351
Sequence AACTAAC has score s = 8.865
Sequence TTCTAAT has score s = 8.316
Sequence TTCTAAC has score s = 11.017
Sequence TATTAAC has score s = 9.527
Sequence TACTGAC has score s = 8.920
Sequence TTCTAAC has score s = 11.017
Sequence TACTAAT has score s = 9.701
Sequence TCCTAAC has score s = 6.847
Sequence TACTAAC has score s = 12.401
Sequence TACTAAC has score s = 12.401
Sequence TTCTAAC has score s = 11.017
Sequence GACTAAC has score s = 9.351
Sequence TATTAAC has score s = 9.527
Sequence TACTGAC has score s = 8.920
Sequence TGCTAAC has score s = 10.017
Sequence TACTAAC has score s = 12.401
Sequence TTTTAAC has score s = 8.142
Sequence TTCTAAC has score s = 11.017
Sequence TACTAAC has score s = 12.401
Sequence TTCTAAC has score s = 11.017
Sequence TACTAAC has score s = 12.401
Sequence TTCTAAC has score s = 11.017
Sequence TACTAAC has score s = 12.401
Sequence TACTAAC has score s = 12.401
Sequence TACTAAC has score s = 12.401
Sequence TACTAAC has score s = 12.401
Sequence TATTAAC has score s = 9.527
Sequence TACTAAC has score s = 12.401
Sequence GACTAAC has score s = 9.351
Sequence GACTAAC has score s = 9.351
Sequence TACTAAC has score s = 12.401
Sequence TACTTAC has score s = 7.335
Sequence TTCTAAC has score s = 11.017
Sequence TACTAAC has score s = 12.401
Sequence TACTAAC has score s = 12.401
Sequence TTTTAAC has score s = 8.142
Mean score = 9.834
```

4. Probability and Statistics

The genome of the mycoplasma *Mesoplasma floracea* is somewhat unusual in that it has a very low G+C content (25%) and has a very small (734 kb) circular genome. There is a convention for which strand reads 'clockwise' and which strand reads 'counterclockwise'. All questions refer to the clockwise strand.

a. Assume that you start sequencing bases from the genome at an arbitrary position in the genome. Sequencing clockwise, how many bases would you expect to sequence before seeing a G+C in the genome? Same question, sequencing counterclockwise from the same position?

b. What is the average spacing between consecutive G+C bases in the genome (i.e. average number of intervening A+T bases)? Explain why this number is the same or different from the sum of the two values you obtained in part a.

c. During the annotation of the genome, a gene finding program was used to find 686 predicted coding regions in the genome. Of the 686 putative coding regions, 432 of these were confirmed on the basis of similarity to coding regions in other species. The other predicted coding regions remain unconfirmed. You had the idea that perhaps coding regions have a higher G+C content than noncoding regions in the genome because of the necessity of encoding certain amino acids. Out of the 432 "confirmed" coding regions, 337 have a G+C content higher than 25%. In a set of 203 "confirmed" non-coding regions (regions which have been experimentally shown not to express any transcripts), only 17 have C+G content higher than 25%. If the G+C content of a particular predicted coding region is above 25%, what is the probability that it is a true coding region? Show your work including the joint probability table. Assume that "unconfirmed" coding/non-coding regions are similar in composition to confirmed coding/non-coding regions, respectively.

d. In the next step of annotating the *Mesoplasma florum* genome, you want to be able to find all transcriptional terminators in the genome. Based on the number of predicted coding regions and the size of the genome, a very rough calculation suggests that about ~3% of the genome consists of transcriptional terminators. Based on previous tests, the TerminatorScan program is known to predict a transcriptional terminator in a DNA sequence, if one is present, with a probability of 0.79. If a terminator is not present in the sequence, the program will detect one falsely with a probability of 0.15. Use Bayes rule to determine the probability that a terminator is actually present if the program predicts a terminator in an arbitrarily chosen segment of the genome. Show your work. How useful will this program be in annotating the *Mesoplasma* genome?

e. Another research group has published a paper that includes estimates for the mutation probabilities between nucleotides in *Mesoplasma florum*. Their analysis accounts for the fact that transitions are more common than transversions, that A+T nucleotides are more prevalent than G+C nucleotides and that the organism lacks basic DNA repair machinery. Based on these transition probabilities, calculate the steady-state probabilities of any given nucleotide in the genome being an A, T, G or C assuming a first order Markov model for DNA mutation. Is the base composition of *Mesoplasma florum* in equilibrium? (Note: P_{XY} denotes the probability of a nucleotide mutating from X to Y.)

$$P = \begin{bmatrix} P_{AA} & P_{AC} & P_{AT} & P_{AG} \\ P_{CA} & P_{CC} & P_{CT} & P_{CG} \\ P_{TA} & P_{TC} & P_{TT} & P_{TG} \\ P_{GA} & P_{GC} & P_{GT} & P_{GG} \end{bmatrix} = \begin{bmatrix} 0.95 & 0 & 0.03 & 0.02 \\ 0.04 & 0.91 & 0.05 & 0 \\ 0.03 & 0.02 & 0.95 & 0 \\ 0.05 & 0 & 0.04 & 0.91 \end{bmatrix}$$

5. Viterbi Algorithm

Your lab has developed a first order Hidden Markov Model for predicting helical segments based on experimentally determined amino acid helix propensities published by Karyn T. O'Neil and William F. DeGrado in "A Thermodynamic Scale for the Helix-Forming Tendencies of the Commonly Occurring Amino Acids", *Science*, New Series, Vol. 250, No. 4981. (Nov. 2, 1990), pp. 646-651. The model has two hidden states – helical (H) and non-helical (N).

a. Which parameter(s) do you think was/were derived using the amino acid helix propensity data?

b. The emission probabilities of the HMM for each amino acid in each state are given below in Table 1. The transition probabilities between states are given in Table 2. Assume that the prior probabilities for the first position in the sequence are $P_H = 0.3$ and $P_N = 0.7$. What is the joint probability of generating the sequence FSRSNHLSPC and parse HHHHHHHNNN using the given HMM? Show your answer and how you arrived at it.

Amino Acid	Pem in H	Pem in N
A	0.1352	0.05
C	0.0224	0.05
D	0.0247	0.05
E	0.0407	0.05
F	0.0369	0.05
G	0.0166	0.05
H	0.0224	0.05
I	0.0334	0.05
K	0.0907	0.05
L	0.0907	0.05
M	0.0742	0.05
N	0.0334	0.05
P	0.0001	0.05
Q	0.0742	0.05
R	0.1224	0.05
S	0.0407	0.05
T	0.0247	0.05
V	0.0302	0.05
W	0.0498	0.05
Y	0.0369	0.05

Transition Probabilities		
	Helix	non-Helix
Helix	0.9	0.1
non-Helix	0.17	0.83

Table 2. Transition probabilities of the HMM.

Table 1. Emission probabilities (Pem) in the Helix (H) state and the non-Helix (N) state.

c. You are studying a protein, which you suspect to have a helical segment. You would like to use the HMM developed by your lab to find the most likely helical parts of your protein. The sequence under consideration is: CGQALFAASP. Use a trellis diagram to determine the optimal parse of this sequence. Make two rows of 10 circles, one immediately below the other. Label the top row 'H' and bottom row 'N', and number the columns 1..10. Fill in the probability of the optimal parse of the subsequence *1..k* (of the given sequence) that ends in a Helix state in circle *k* of the Helix row, and the probability of the optimal parse of the subsequence *1..k* ending in an non-Helix state in circle *k* of the non-Helix row. Start by filling in the two circles in the first column using the prior probabilities and the emission probabilities. To fill in the Helix circle in the second column, compare the probabilities of the of two parses: a) the optimal parse up to position 1 ending in H, followed by a transition from H → H; versus b) the optimal parse up to position 1 ending in N, followed by a transition from N → H. Fill in the maximum of these two values in the H circle in the second column, and draw an arrow from the circle in the first column that contributed to this maximal probability parse to the circle in the second column. Continue this way, filling in the whole trellis diagram. What is the joint probability of the optimal parse of the sequence ending in state H? What is the joint probability of the optimal parse of the sequence ending in state N? What is the optimal parse overall? (Hint: backtrack through the trellis diagram, following the arrows in reverse.)

d. After going through the above exercise, you find out that someone in your lab has already implemented the Viterbi algorithm in python. You use the program to evaluate a 1000 residue sequence and it completes in 0.1 seconds. Based on this information and what you know about the running time of the Viterbi Algorithm, what is your best guess as to how long it would take to apply the above HMM onto all the sequences in SwissProt (53,205,430 characters)? How long would it take if you added two new states – beta strand and turn (assume that these states are implemented in exactly the same way as the other two)?

Hints for Programming Problems:

- see Peter's python tutorial notes for information on reading in files, command line arguments, and regular expressions

- *f = open("file.txt");*
  lines = f.readlines()
  f.close()f.readlines()

  Reads the contents of the file "*file.txt*" and stores it as an array of strings ending with newlines.
- Regular expressions are useful for parsing fasta files since you can parse on the '>' character and then on the first newline character. To use regular expressions you have to 'import re' at the top of your script. Some useful regular expression functions:
  - *new_str = re.sub(regexp, subst, str)*. This takes the string *str*, replaces every occurrence of regular expression *regexp* with string *subst*. If you don't want to replace every occurrence, you can replace only the first *n*, by specifying *n* as the 4th argument. Examples:
    - seq = re.sub(">.*\n", "", fasta)
    - seq = re.sub("\n*", "", seq)
  - *match = re.match(regexp, str)* – returns a match object if regular expression *regexp* is found at the start of string *str*. Returns false (python special value None) otherwise – you can check whether the match exists or not with an *if* statement.
  - *match = re.search(regexp, str)* – the same as *re.match*, but succeeds if the regular expression *regexp* matches any position in the string (not just the beginning).
  - *list = re.split(regexp, str)* – splits the string *str* into sub-strings delimited by the regular expression *regexp*. Returns the sub-strings in a list.
  - See http://www.amk.ca/python/howto/regex/ for a good explanation of regular expressions. The split() function is useful.
- One useful type of regular expressions for this assignment – character class regular expression. "[ABC]" will match either A, B, or C. Thus, "C[GT][CA]GA" will only match CGCGA, CGAGA, CTCGA, and CTAGA.
- nucleotides = { 'A':0, 'T':1, 'G':2, 'C':3 } -> This declares a dictionary to convert between bases and array indices.
- v = [] -> creates an empty array.
- v.append(0) -> appends the value 0 to the array v.
- v.append([]) -> append an empty array to the array v.

- s.upper() -> returns a new string in which s has been converted to uppercase.
- range(m,n,p) -> creates an array of numbers between m and n-1 in increments of p, which is useful in for loops.
- math.log(x,b) -> return the log of x to the base b. Must “import math” at the top of your script.
- s.replace(‘\n’,’’) -> deletes the newlines in string s.
- See http://www.python.org/doc/current/lib/typesseq-strings.html#l2h-206 for information on formatting strings for printing