CS 3824
Project Description — Motif Finding

Given: October 2, 2015 **Due:** December 14, 2015

Introduction. This is a project to find motifs, possibly with "don't care" characters, in a list of sequences. The project is to be accomplished as a team (teams will be assigned after an online survey is completed) with everyone on each team working together towards the overall goal. Each of you will write up a report on the project separately for grading, due on December 14, 2015 at 8:00 AM. The steps in the project are detailed below.

Problem Definition. While the computational problem at the heart of this assignment is a variant of the motif finding problem discussed in class, it is sufficiently different to deserve a completely new presentation. This will also be covered in a future class.

The primary input is a tuple

$$S = (S_1, S_2, \ldots, S_n)$$

of strings over the DNA alphabet $\Sigma = \{A, C, G, T\}$. The lengths of the strings are in general different. We assume that every character in Σ occurs at least once in at least one of the strings in S. For each $\sigma \in \Sigma$, let τ_σ be the count of occurrences of σ in all the strings in S. For $\sigma \in \Sigma$, define

$$\rho_\sigma = \frac{\tau_\sigma}{\sum_{\gamma \in \Sigma} \tau_\gamma},$$

which we take to be the probability distribution for the characters in Σ in a null model for DNA strings.

A *motif* $M = \alpha_1\alpha_2 \cdots \alpha_k$ *of length* k *and having* d *"don't cares"* is a string of length k over the alphabet $\{A, C, G, T, *\}$ having exactly d $*$'s, where neither the first nor the last character of M is $*$. An example for $k = 8$ and $d = 3$ is $CG{*}AT{*}{*}T$, while ${*}A{*}CGT{*}C$ is not. The idea behind a "don't' care" character is that nucleotides in that position will not influence our scoring of the motif.

Fix a motif length $k \geq 2$. A tuple of *starting loci* for S is a tuple $s = (s_1, s_2, \ldots, s_n)$, where $1 \leq s_i \leq |S_i| - k + 1$ for all i. In the manner previously described in class, s defines an $n \times k$ alignment of k-mers, one from each S_i, starting at index s_i. From the alignment, we derive a $4 \times k$ profile matrix $\Psi = (\psi_{\sigma,j})$, where $\sigma \in \Sigma$ and $\psi_{\sigma,j}$ is the count of the character σ in column j of the alignment. From Ψ, we obtain a probability matrix $\Pi = (\pi_{\sigma,j})$ by defining

$$\pi_{\sigma,j} = \frac{\psi_{\sigma,j}}{n}.$$

Each column of Π defines the empirical probability of finding each $\sigma \in \Sigma$ in column j, given the starting loci s. Comparing the empirical probability of M to the null model, we get the *likelihood*[1] of M given s:

$$L(M, s) = \prod_{\substack{j=1 \\ \alpha_j \neq *}}^{k} \frac{\pi_{\alpha_j,j}}{\rho_{\alpha_j}}.$$

[1]More correctly, the *likelihood ratio*.

The corresponding *log-likelihood* is[2]

$$LL(M, s) \quad = \quad \lg L(M, s).$$

We will use $LL(M, s)$ as the score for motif M at starting loci s, where a higher score is a greater indication that M matches well at s.

The optimization problem that you are to solve is

MOTIF FINDING

INPUT: Tuple $S = (S_1, S_2, \ldots, S_n)$ of strings over $\{A, C, G, T\}$; k, where $2 \leq k$; d, where $0 \leq d \leq k - 2$.

OUTPUT: A motif M of length k with d "don't cares" and a tuple $s = (s_1, s_2, \ldots, s_n)$ of starting loci that maximizes $LL(M, s)$.

Program Requirements. Each team will develop a motif finding program to solve the MOTIF FINDING problem. It is anticipated that solving the problem optimally will be too expensive computationally, so a program will be evaluated on its ability to achieve as large an $LL(M, s)$ value as it can, for particular test inputs.

Your program should be called `MotifFinder` and should run on a Linux command line as follows:

```
        $ MotifFinder -k 8 -d 3 -t 5 input.fasta > MotifFinder.out
```

Here, k is an optional argument for the length of the motif; it defaults to 6; d is an optional argument for the number of $*$'s; it defaults to 0; t is the maximum number of seconds for the program to run; it defaults to 2 seconds; the FASTA-formatted input comes from file `input.fasta`. Output goes to standard output, which is formatted like this:

```
Best motif of length 8 with 3 don't cares is AAC**G*C
Log likelihood is 6.5852896695299385
Loci of the best motif are here:
63
70
104
78
113
54
29
34
71
42
```

You are free to use any programming language for which your program can run (and compile, if necessary) on the `rlogin.cs.vt.edu` resource. You are encouraged to do research into existing motif finders to get ideas that you might implement to achieve high quality results.

[2] Recall that lg is the logarithm base 2.

Available Programs. Three Python 3 programs are available from the course web site for demonstration purposes. To run Python 3 programs on `rlogin.cs.vt.edu`, you need to run

```
$ scl enable python33 bash
```

first. All three programs take various command line options that you can find out about by using the `--help` option. Of especial importance are the `-k` option, which specifies the motif length, and the `-d` option, which specifies the number of "don't care" characters. Here is a sample execution of the three in order:

```
$ EmbedMotif.py -k 8 -d 3 -n 20 -m 500 --mu 0.05 -a 0.30
    -c 0.20 -g 0.20 sequences.fasta
$ RandomMotifFinder.py -k 8 -d 3 -t 10 sequences.fasta >
    RandomMotifFinder.out
$ EvaluateLogLikelihood.py -k 8 -d 3 sequences.fasta <
    RandomMotifFinder.out
```

`EmbedMotif.py` generates a FASTA file that is a test set for a motif finder; the generated motif can be found in `EmbedMotif.log`. `RandomMotifFinder.py` is a simple-minded motif finder that just looks at random starting loci to find a "best" motif; note that the `-t` parameter controls how many seconds the program searches for that motif. `EvaluateLogLikelihood.py` reads the output from a motif finder and verifies that it found the right motif with the right log-likelihood for particular starting loci.

Testing. You can use `EmbedMotif.py` to generate test data for your `MotifFinder` or you can find sample sequences elsewhere. Your program should have (at least) the same command line options as `RandomMotifFinder`, including the `-t` option to control how long it executes. You should use `EvaluateLogLikelihood.py` as a double check on your calculations of the right motif for your starting loci and of the log-likelihood.

Oral Report. On Tuesday, November 17, each team will give a 10 minute presentation, which will be an overview of its plan for implementing a high-quality motif finder for the MOTIF FINDING problem. Every team member is expected to participate.

Written Report. Write a report (preferably in LaTeX) detailing what you did in the project and what results you obtained. Be certain to clarify what part of the work of your team you actually did. Include a table or graph summarizing the testing you did. The report should be 4–5 pages long to be comprehensive. Submit the report as a PDF to Scholar by 8:00 AM on December 14, 2015; use the "Assignments" tab.

Program Submission. For each team, the team leader will submit a `tar` file consisting of the `MotifFinder` source code, an informative `README` file that details how to build and run the program, and the files for at least one test run that the team believes exemplifies the quality of its product. Submit the `tar` file to Scholar by 8:00 AM on December 14, 2015; use the "Drop Box" tab.

Basis for Grade	Points
Oral report	100
Written report	200
Performance	100
Total	400

Table 1: Project grading.

Grading. The project is worth a total of 400 points. Of those, 100 points are reserved for an instructor-evaluated measure of performance. This performance will be measured on a small suite of hidden test cases. Each test case will be run for differing amounts of time (possibly extending to several minutes) to evaluate how quickly your program reaches a high quality solution. At least one test case will have hundreds of input sequences. For each test case, the higher the log-likelihood achieved, the better the performance. A project that achieves highest performance on all test cases will receive the full 100 points. Other projects will receive fewer points; it is possible that a weakly-performing project will receive 0 points. The grading breakdown is as in Table 1.

Ethics. The Honor Code applies in the sense that you may not seek help outside your team except from the instructor and the GTA. The cooperation you have within the team is not limited.