Motif Finding:

Finding binding sites for transcription factors using randomized algorithms

Andrew Buxa

Integrated Research Component I

Prof. Louis Oliphant

April 12, 2016

**Abstract**

In the field of biology, genes which are able to be turned on and off at the same time are co-regulated. This can be attributed to a common pattern found upstream from a gene which are called motifs. There are many ways of finding motifs, however, the most common method of finding motifs are randomized algorithms. We use these randomized algorithms because they produce good results in a reasonable amount of time. My research compares three of these randomized search algorithms for accuracy and efficiency. These algorithms are Gibbs Sampling1, Randomized Motif Search2 and the MEME3 algorithm.

**Introduction**

Motif finding is a very hot topic in the field of Bioinformatics because there aren’t any solidified ways of locating these binding sites before genes. These motifs are presumed to have some kind of biological function. I planned on implementing multiple algorithms which would perform the same task, which is finding these binding sites but in different ways.

**Relevant Work**

Since locating binding sites is such an important task in Bioinformatics, a lot of brilliant minds have come up with various algorithms which perform the same function. However, these algorithms have a slight twist to how they locate and determine where these binding sites are. By looking at many previous papers about motif finding, a lot of material helped me get started.

**Project Description**

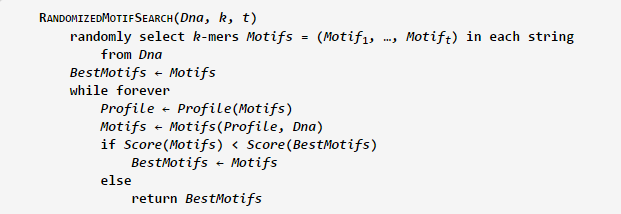
 From the very beginning of the semester, I was always intrigued by the idea of motif finding and I did not really understand the significance as to why we must find these motifs. Initially, the project looked relatively easy to complete because our Bioinformatics Algorithms book had the pseudo-code for some motif finding algorithms. In order to design and implement these algorithms into a python format, I wanted to first understand what a motif was and what the algorithms were trying to achieve. First, I focused on the Randomized Motif Search algorithm which was the easiest to implement and design. I managed to get it working the moment I finished the code and the pseudo-code I followed is below (figure 1):

Figure 1Randomized Motif Search pseudo-code.

Originally for testing purposes, I created my own artificial test sequences with implanted motifs in order to see if the implemented algorithm actually returned the absolute best motifs. Needless to say, it did, however the way I calculated the score which was given in the book as well was not real efficient. The way the program calculated the score for a randomly selected list of motifs was first getting the consensus sequence of the list. Then, you would have to go through each motif in the list and get the hamming distance between the consensus sequence and the selected motif in the list.

Randomized Motif Search works by taking in a list of DNA sequences, a *k*-size or size of the motif and *t* which is the amount of iterations the algorithm will perform. After the information is sent in to the function as parameters, the program will randomly select a position in each sequence and take a motif of size *k* and append it to a list. Next, this first list of motifs are set as the best motifs because the algorithm needs a set of motifs to compare newly acquired ones to. After calculating the score for the newly selected motifs, a comparison between the two scores is needed. If the newly selected motifs score is less than the best motifs score, then set the new motifs to best motifs. And this algorithm will keep performing this function *t* amount of times. After all of the iterations, the algorithm should return a list of the best motifs that are *k*-size long.

Second, I began working on Gibbs Sampling which does is similar to Randomized Motif Search with a few differences. One of the major differences with Gibbs Sampler is that the selected list of motifs does not all change at once, instead one motif in the list is changed. Below is the pseudo-code for Gibbs Sampling and a thorough explanation of how it works (figure 2):

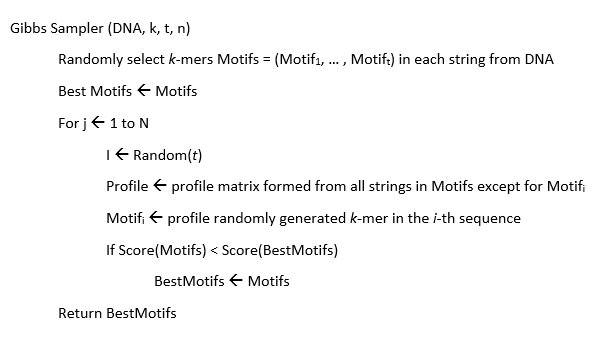


Figure 2 Gibbs Sampler pseudo-code

The way in which my Bioinformatics group and I pieced Gibbs Sampling together is slightly different from the pseudo-codes depiction. We started off by calculating the nucleotide frequencies which returns a list of dictionaries (in python) that contains the frequencies of each nucleotides in each sequence. Instead of having a for-loop from one to *n* iterations, I had a while loop that essentially did the same thing. Next, you want to generate a random start location for each sequence in the list of sequences and then load them into a motif List. Another while loop is needed so Gibbs Sampler does not get “stuck” because of various stopping rules that it relies on. Essentially, you want to repeat the first two steps and then construct a profile which is a representation profile of the *k*-mers. Now, you must find the frequencies of each nucleotide in each sequence by applying the profile and store this new profile. A random index must be selected from the probabilities of each profile and a new motif must be selected for that individual element in the motif list while the others remain the same. Then the edited motif list must have a profile constructed and finally, you must check the scores of both profiles that were created. Since relative entropy was being used, the maximal score will be taken as the best, if this case is true set best motifs to be the temporary motif list. The program will run *n* amount of times and then return the best motifs.

Next, I wanted to go out and look for a complex algorithm that solves the motif finding problem. I stumbled across the MEME algorithm which stands for Multiple EM for Motif Elicitation and what it does is given a data set of sequences, returns the best motifs. However, unlike Randomized Motif Search and Gibbs Sampler, MEME can return multiple motifs in the same sequence. Below is the pseudo-code for the MEME algorithm (figure 3):

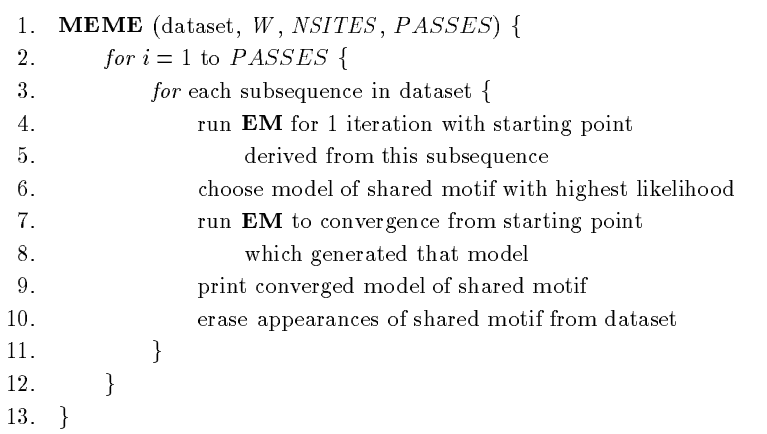


Figure 3 MEME Algorithm pseudo-code

Like before, I put together the MEME algorithm differently than the pseudo-code shows, but it follows along the same rules. First, we want to have a loop that goes up to *PASSES* and inside this loop we want to calculate the nucleotide frequencies for the dataset. I chose to re-use code from Gibbs Sampling to make the work easier. Then you want to split each sequence into a subsequence of size *w* for each sequence in the list. Now construct a profile of each subsequence and then add the profile, sequence and position of the sequence into a list. Sort the list based off the score in ascending order and take *nsites* (expected amount of motifs in sequences) amount of worst scoring motifs and construct a profile using these *nsites* amount of matches. Run a modified version of the Randomized Motif Search that allows multiple or no matches per sequence up to *nsites* to be found. Remove those motifs from the dataset, which you have the index and position of these motifs since you previously added them into a list. Finally, repeat these steps *passes* amount of times and the best motifs will be returned to you.

**Results and Evaluation**

In terms of results, there are not any unique results I have to show other than the general output of my algorithms. Though, the only difference in outputs are the results of MEME and Randomized Motif Search/Gibbs Sampler. Below are what the results of running both Randomized Motif Search and Gibbs Sampler will look like (figure 4):



Figure 4 Expected results of both Gibbs Sampling and Randomized Motif Search

This output shows the best selected motifs in each of the lists starting from the first element on the left going to the last on the right. I believe I was successful at implementing the first two algorithms because the results are relatively similar. It’s just a matter of performing more iterations and finding the right *k*-size of the motifs, which in this case for the data I used, the *k*-size is between ten and twenty. In my proposal, I was expecting to program three algorithms, where I had to change the last one from ALSE to MEME. My Randomized Motif Search and Gibbs sampling do fit the description of my proposal perfectly, however, at the time I wrote my proposal I did not have a good idea of what a motif was. If I were to start over again, I would begin by programming MEME first in order to successfully implement it. If that was the case I would definitely feel more confident in getting all of the work done and successfully working.

**Conclusion and Future Work**

I learned quite a lot about motif finding and how significant the motif finding problem is in the field of Bioinformatics. It is a very interesting topic to me because of the various ways of seeking out these motifs upstream from a gene. Needless to say, if you want to find motifs, you must have a program not for a set amount of iterations.

If I had more time to work on the project, I would make sure absolutely everything worked with no bugs. I’m confident that mostly everything I have implemented works, however, some users figure out ways to stumble across bugs one way or the other. As I previously stated, I would start working on the MEME algorithm first to get rid of any hard comings in the future if I had to start over.

**References**

1. Compeau, Phillip, and Pavel Pevzner. *Bioinformatics Algorithms: An Active Learning Approach*. La Jolla, CA: Active Learning, 2015. Print.
2. Compeau, Phillip, and Pavel Pevzner. *Bioinformatics Algorithms: An Active Learning Approach*. La Jolla, CA: Active Learning, 2015. Print.
3. Bailey, Timothy L., and Charles Elkan. "Unsupervised Learning of Multiple Motifs in Biopolymers Using Expectation Maximization." *Mach Learn Machine Learning* 21.1-2 (1995): 51-80. Web.
4. Das, Modan K., and Ho-Kwok Dai. "A Survey of DNA Motif Finding Algorithms." *BMC Bioinformatics* 8.Suppl 7 (2007). Web.
5. Leung, H. C. M., and F. Y. L. Chin. "Finding Motifs from All Sequences with and without Binding Sites." *Bioinformatics* 22.18 (2006): 2217-223. Web.