CS 4223 Final Project Report

Jeff Dong - kqm018

Date: December 4, 2022

1 Abstract

For this project, I decided to implement and analyze two different algorithms for the motif finding problem; Multiple Expectation-Maximization for Motif Elicitation (MEME) and Gibbs Sampling. In contrast to the exhaustive search styles of Sample-Driven or Pattern-Driven Motif Finding algorithms, MEME and Gibbs Sampling take a probabilistic approach to the problem. As a result, both of the algorithm's output is sensitive to the initial starting conditions provided. To solve this issue, MEME performs an exhaustive search of all possible starting points to select the best one to use in the algorithm. In contrast, Gibbs Sampling adopts a stochastic approach and allows non-optimal steps to avoid local optima. As a result of these two differing solutions, Gibbs Sampling leads to an unequivocally better runtime compared to MEME.

2 Introduction

DNA motif finding is an important and difficult problem in the field of bioinformatics. In biology, a DNA motif is defined as a recurring subsequence that appears in part or in whole among a set of DNA sequences. As a result, motif finding is a vital step into studying gene functions and investigating the mechanisms that regulate gene expression. Previous algorithms tackling this problem include Pattern-Driven Motif Finding (PDMF) and Sample-Driven Motif Finding (SDMF). PDMF and SDMF are combinatorial algorithms that enumerate a set of possible motif candidates and find which candidate is the best match across all of the sequences. PDMF and SDMF only differ in how they generate the set of candidates; PDMF enumerates all possible candidates whereas SDMF only enumerates the candidates found within the sequences. Although PDMF and SDMF perform well for sequences that have strong motifs (few mutations), their performance drops off with larger sequences or with sequences with weak motifs. This primarily due to their runtimes increasing significantly as the sequence or motif length grows, but also due to random strings/noise potentially scoring better than any single instance of the true motif as well. To alleviate this issue, probabilistic algorithms such as MEME and Gibbs Sampling were proposed to solve the motif finding problem. Rather than doing any enumeration, these algorithms construct probabilistic models to distinguish motifs versus non-motifs.

3 Algorithms and Implementation

Everything algorithm, testing, and plotting related for this project was done in Python utilizing the standard data science libraries (numpy, matplotlib). All of the following code, data files, and more detailed installation/version documentation can be found on my GitHub:
`https://github.com/Jeffafuh/CS-4223-Bioinformatics-Final-Project`

For both of the following algorithms, assume that the motif has a fixed length of l, each sequence has the same length L, and the set of N sequences is denoted as X where $X_{i,j}$ represents the jth character (starting at 1) in sequence i of X. Additionally, the algorithms will utilize a Position-Weighted Matrix (PWM) denoted as W. The PWM is a probabilistic representation of the motif where $W_{c,k}$ represents the probability of character c appearing at position k (starting at 1) of the motif. Additionally, the information of the background probabilities will be encoded in column $k = 0$ of the PWM. For example, $W_{A,2}$ is the probability that A is the second character of the motif and $W_{C,0}$. is the probability of a C occurring by pure chance. An important note is that in my code, sequences aren't treated strings of characters but rather an array of numbers 0 through 3, inclusive.

3.1 Multiple Expectation-Maximization for Motif Elicitation (MEME)

In addition to the PWM, MEME also introduces a new matrix Z such that $Z_{i,j}$ represents the probability the motif starts in position j in sequence i. MEME consists of two primary steps: the E-step and the M-step.

In the E-step, Z is re-estimated using the PWM according to the following formula:

$$Z_{ij} = \prod_{k=1}^{j-1} W_{c_k,0} \prod_{k=j}^{j+l-1} W_{c_k,k-j+1} \prod_{k=j+l}^{L} W_{c_k,0}$$

In my code, this takes 5 for-loops: a nested loop iterating through the Z matrix, two loops for multiplying the background probabilities before and after the assumed motif starting point, and one loop multiplying the actual motif probabilities. After updating the Z, every row is normalized such that its sum is one.

In the M-step, the PWM is re-estimated using Z according to the following formula:

$$\text{For } k > 0,\ W_{c,k} = \sum_{i=1}^{L} \sum_{\{j|X_{i,j+k-1}=c\}} Z_{i,j} \quad \text{For } k = 0,\ W_{c,k} = n_c - \sum_{j=1}^{l} W_{c,j}$$

Where n_c is the number of times c appears in all of the sequences. After updating the PWM, every column is normalized such that its sum is one. Thanks to various numpy functions, this can be done using one for loop iterating over the columns of the PWM combined with clever slicing and calls to the sum() function.

Using the E-step and M-step, MEME simply repeats these two steps (E then M) until the PWM converges. For my implementation of MEME, I start the algorithm by considering a random sampling of 10% of the unique subsequences of length l found in the sequences (primarily to reduce the runtime to practical times). For each subsequence, I run EM for one iteration then select the subsequence that had the highest probability of occurring to use for the rest of the algorithm.

3.2 Gibbs Sampling

Gibbs Sampling adopts a more stochastic approach to avoid being potentially trapped in a local optimal like in MEME. Similar to the Z matrix in MEME, Gibbs Sampling will instead keep track of the predicted location for the motif in each sequence. For initialization, predictions of the positions for motif are chosen at random. Afterwards, Gibbs Sampling follows 3 primary steps: the removal step, the predictive update step, and the sampling step.

For the removal step, a random sequence in X is removed and excluded. This sequence is added back in after the predictive update and used for the sampling step.

In the predictive update step, the PWM is updated to reflect the sequences without the newly excluded sequence. This is performed with the following formula:

$$\text{For } k > 0,\, W_{c,k} = \frac{n_{c,k}}{N-1} \quad \text{For } k = 0,\, W_{c,k} = \frac{n_{c,k}}{N*(L-l)}$$

Where $n_{c,k}$ is the number of times c appears in the kth position of the predicted motifs if $k > 0$ or the number of times c appears in the background if $k = 0$.

In the sampling step, a new predicted position of the motif for the excluded sequence is chosen. For each possible starting position of the motif, the ratio of the motif actually being there versus being generated by the background is calculated. This can be represented by the following formula:

$$S_{x,y} = \prod_{j=x}^{x+l-1} \frac{W_{X_{y,j},j}}{W_{X_{y,j},0}}$$

Where $S_{x,y}$ is the probability of the motif starting at position x in sequence y. After S is generated for the excluded sequence, it is normalized such that all elements in S sum to one. Then, one of the motif locations is sampled based on the distribution of S itself.

The removal step, predictive step, and sampling step are then repeated until the predicted locations of the motif converge or have sufficiently little change.

4 Results and Discussion

4.1 Experimental Setup

I measured the runtime for both algorithms on a randomly generated set of sequences while altering various parameters used to generate the sequences. I first generate the appropriate number of sequences using a uniform distribution, then randomly choose one location in each sequence to plant the motif (which itself is randomly mutated accordingly). For each unique configuration of the sequence generation parameters, 5 sets of sequences/motifs were generated. The runtimes reported in the graphs are the average runtime of the algorithms across the 5 sets. Unless stated otherwise, the default values used to generate the sequences are as follows: Number of Sequences = 20, Sequence Length = 40, Motif Length = 7, Motif Mutations = 1. The exact sequences used to test each parameter can be found in plain-text files in the GitHub repo.

4.2 Results

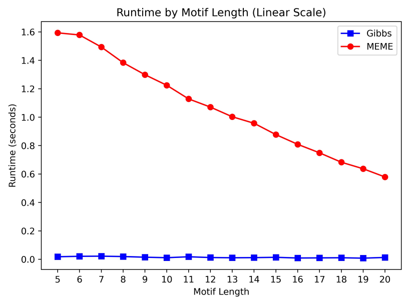

Figure 1(a)

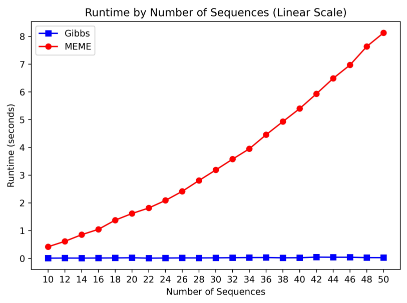

Figure 2(a)

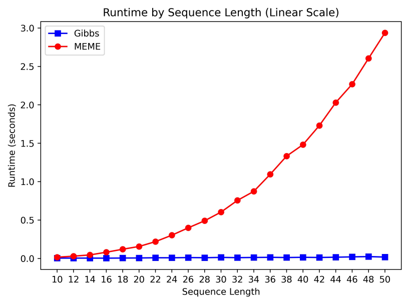

Figure 3(a)

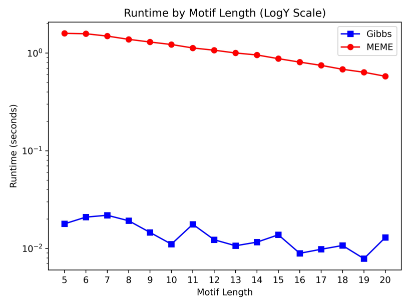

Figure 1(b)

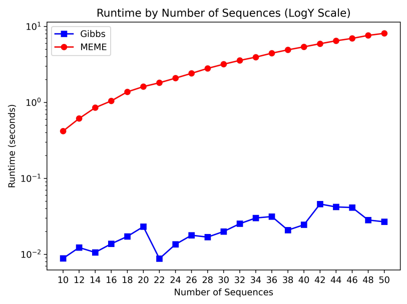

Figure 2(b)

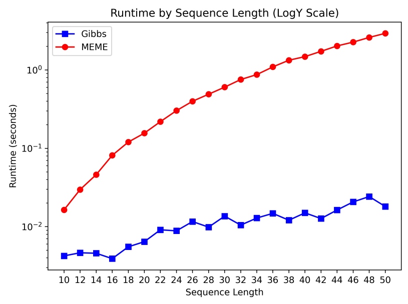

Figure 3(b)

4.3 Discussion

Notably, the ”Runtimes by Motif Mutations” graphs are absent due to the trivial fact that the runtimes for both algorithms are invariant to the number of mutations in the motifs. For all of the following figures, (a) is the graph in linear scale whereas (b) is the graph in log-y scale.

Figure 1 shows the change in runtime when the motif length of the sequences is altered. Although Figure 1(a) shows that Gibbs Sampling is relatively invariant to the motif length changing compared to MEME, Figure 2(b) better illustrates that both algorithms experience a downward trend in the runtime as the motif length increases. In MEME, this is due to there being less possible subsequences to search through with a longer motif length, leading to a relatively larger decrease in runtime. In Gibbs Sampling, this decrease is due to there being one less position to calculate a sampling probability for leading to a very minimal (but non-zero) decrease on average.

Figure 2 shows the change in runtime when the number of sequences is altered. Figure 2(a) shows that MEME increases linearly as the number of sequences increases whereas Gibbs Sampling remains largely invariant. However, Figure 2(b) better illustrates that both functions experience an upward trend in their runtime as the number of sequences increases. Inversely compared to the motif length, more sequences leads to a larger set of possible subsequences to search through, resulting in a much larger increase in runtime. For Gibbs Sampling, this marginal increase in runtime is due to there being additional sequences to consider when calculating the PWM in the predictive update.

Figure 3 shows the change in runtime when the sequence length is altered. Figure 3(a) illustrates that MEME increases polynomially when the sequence length increases. The reason that I say polynomially instead of exponentially is due to Figure 3(b). In log-y space, an exponential curve (e.g. 4^x) would be transformed into a linear curve whereas a polynomial curve (e.g. x^4) would be transformed into a log curve. Since the MEME curve in Figure 3(b) is not linear, I conclude that it must follow a polynomial growth. Once again illustrated Figure 3(b), both algorithms experience an upward trend in their runtime as the sequence length increases. Similar to the number of sequences, a longer sequence will also lead to a larger set of possible subsequences to search through, resulting again in a much larger increase in runtime. However for Gibbs Sampling, this minimal increase is due to needing to consider one more position when calculating the sampling probability in the sampling step.

As shown by all (b) figures in log-y scale, there are very minor fluctuations in the runtime for Gibbs Sampling that are not present in MEME. This is due to the fact that different sets of sequences with similar parameters can take a different number of iterations to converge with Gibbs Sampling, resulting in the noise seen in the graph. Overall, Gibbs Sampling is (for the most part) invariant to all sequence parameter changes as shown by all the (a) figures. This is in contrast to the runtime of MEME which is highly dependent of the sequence parameters also shown by the (a) figures.

5 Conclusion

The runtime of Gibbs Sampling is unmatched compared to the runtime of MEME. Although the main loops of each algorithm take approximately the same time, the reason for this discrepancy comes from how the algorithms are initialized. Gibbs Sampling chooses random predictions for the motif, but MEME performs an exhaustive search of all (or part of) the substrings for initialization which drastically increases the runtime. However, one key element missing in this comparison between the MEME and Gibbs Sampling is the actual performance and output of the algorithms; how well did each algorithm find the planted motif? This was mainly excluded due to difficulty and time restraints for the project. Often, the algorithms would find a partially-shifted version of the planted motif, but there was difficulty deciding how to score that versus other possible permutations of the planted motif. Improvements in the comparison between these two algorithms can possibly come from implementing a proper scoring system of the outputs of MEME and Gibbs Sampling that takes into consideration the probabilistic nature of the algorithms.