CSCI 4314/5314 Benchmark - Algorithms for Molecular Biology

Motif Discovery

Members:

Monte Anderson, Byron Bearden, Sushma Colanukudhuru, Rasmi Lamichhane, Han Ngo, and Christine Samson.

Dataset Generation:

Dataset is generated randomly from a python code. During the gene creation, we randomly insert one of our motifs from the list into the gene multiple times, but we can change the frequency of random insertions however we want. We then perform this insertion again, to allow overlaps between motifs. This is an edge case that we should check for. We also track where the motifs were inserted into the gene, as well as into each sequence, for comparisons later. Once we have a gene with our motifs inserted, we can create sequences of length L (in our case, 180). Again, we can change this parameter for testing. This also allows for cutting off of motifs, which would interfere with how an algorithm finds that motif. This is another edge case we can test. We can use our data from above (about where we inserted the motifs) to check and see if the algorithms are finding all the motifs we inserted. Due to how the algorithm works, it may miss a few in edge cases, so we need the actual insertion point data to compare it to. We are using MEME as a benchmarking tool. Due to the fact that we generate both a .txt file with all the known data, as well as the .fasta file, we can simply upload the .fasta file to MEME, and compare the output to our known data. There is a constraint in the online version of MEME,

which does not allow a .fasta file to contain more than 1,000 sequences OR 60,000 total characters in a gene. This is somewhat limiting to us, as it prevents us from testing huge files.

Benchmarking Purpose:

The purpose of benchmarking using a random dataset is to test edge cases. In most real DNA, motifs are extremely sparse between sequences, and usually do not overlap or get cut off during sequence splitting. However, we need to test these cases to see what happens if they do occur, and how algorithms handle them. In some cases, we may only see half the motif, and need to decide on what counts as the motif and what does not. This also sets a goal for us, since using these benchmarks will give us a good standard to strive towards. It will also show any errors in the MEME algorithm, but that we can see and hopefully correct. Benchmarking will also allow us to see how complex or large data sets take to complete, and how accurate they are. This will allow us to potentially make trade-offs between accuracy and speed, or find a good balance between them. One interesting feature of meme was it prints/shows the motif depending on the match.

MEME Benchmarking:

For benchmarking MEME, we are able to change a lot of the variables for testing edge cases. We want to try as many edge cases as we can, in order to break MEME and have it not detect our motifs anymore. We also want to see in which cases MEME works best. Since MEME outputs a .txt file containing the data, we can also see how long it takes for each benchmark to complete, and how large the output file is, and how it changes due to our parameters. Below, we selected the most interesting cases of the benchmarks we ran, since these typically deal with edge cases or

weird scenarios we did not predict MEME would do. MEME gives a list of possible motifs, but prioritizes the most matches on the top and least matches on the bottom.

Benchmark Results:

We have attached a zip file with all our data from MEME, as well as the actual .fasta and answer keys we generated using our programs. Below is the most interesting benchmark results.

Benchmark Interpretation:

For our interpretation, we tally up what MEME got right, and what it got wrong. Include positives, negatives, false positives, and false negatives. Since we know exactly how many motifs we inserted, what they are, and where they are, we can see how accurately MEME will find them. For “Small” cases, we use total strand lengths of 2,000 characters. For “Large”, we use 5,000, and for “Huge”, we use 10,000. For “Sparse” cases, we roughly have 2-6 motifs per 2,000 characters. For “Saturated”, we have 100-150 motifs per 2,000 characters. We keep the same sequence length, as we discovered that changing this did not have a huge effect on the outcome of the results from MEME.

Our interpretation of the true false, postive negative results of our trials:

True Positive: Any sites of the motifs we used that were found by MEME

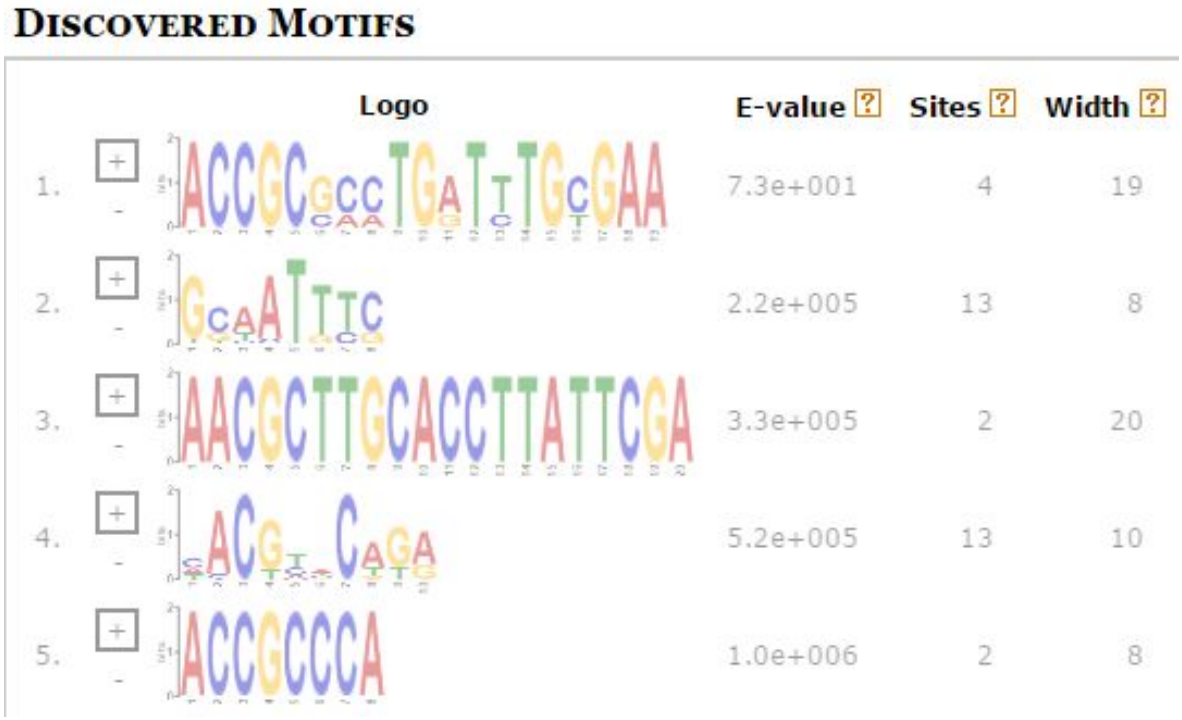
True Negative: Anything that doesn’t show up that shouldn’t but for our case we shall put 0

False Positive: Anything found by MEME that was not what we placed within the sequence

False Negative: Sites of motifs that should have been found but weren’t by MEME

Small | 20mer | Sparse | 180 Sequence Length | 2000 Character Length

Type of Result:	*Count:*
True Positive: Ones found that we placed in	Both motifs we placed were found at all five sites. (5/5)
True Negative	0
False Positive: Found but not put in by us	Three additional motifs were found at 28 separate sites.
False Negative	0 (Since we got all the motifs and sites.)

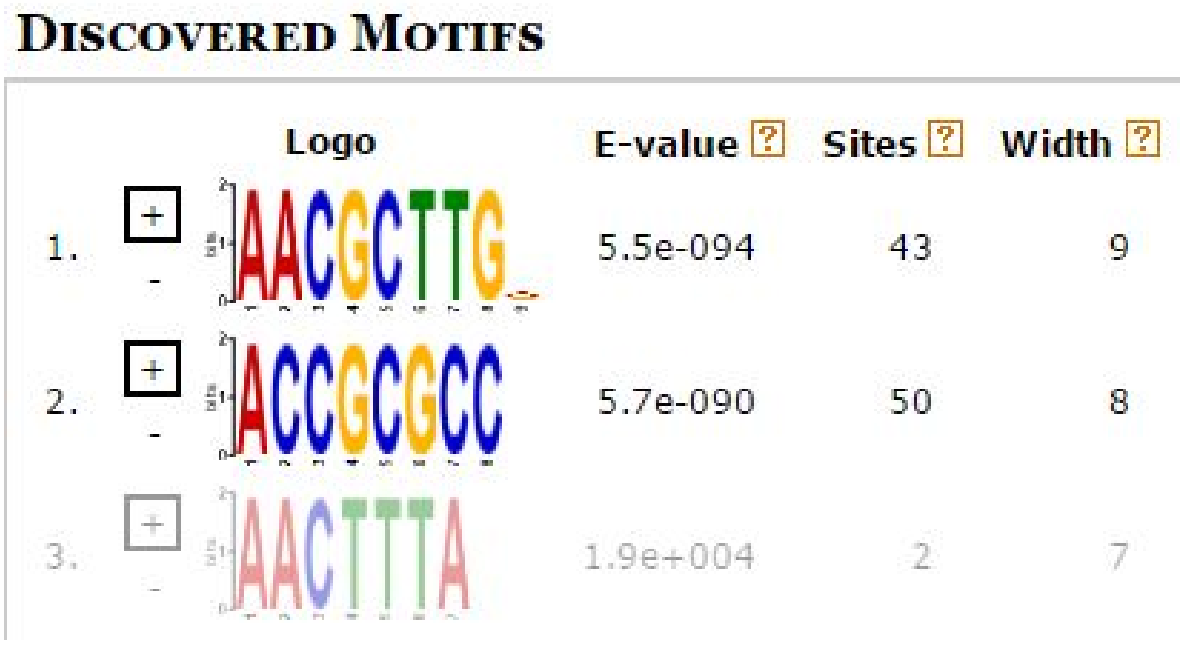


Time(seconds): 9.84

Summary: This trial showed that the accuracy at 20-mers is high but it also has a threshold for 20-mers to be slightly different, as it found a version of the motif that we placed in that had 6 pairs that were different. While we found all of our inserted motifs, it gave us a bit extra.

Small | 8mer | Saturated | 180 Sequence Length | 2000 Character Length

Type of Result:	*Count:*
True Positive	Found all 93 of the motifs we inserted.
True Negative	0
False Positive	Discovered an additional motif at two sites.
False Negative	Zero, all correct matches were found



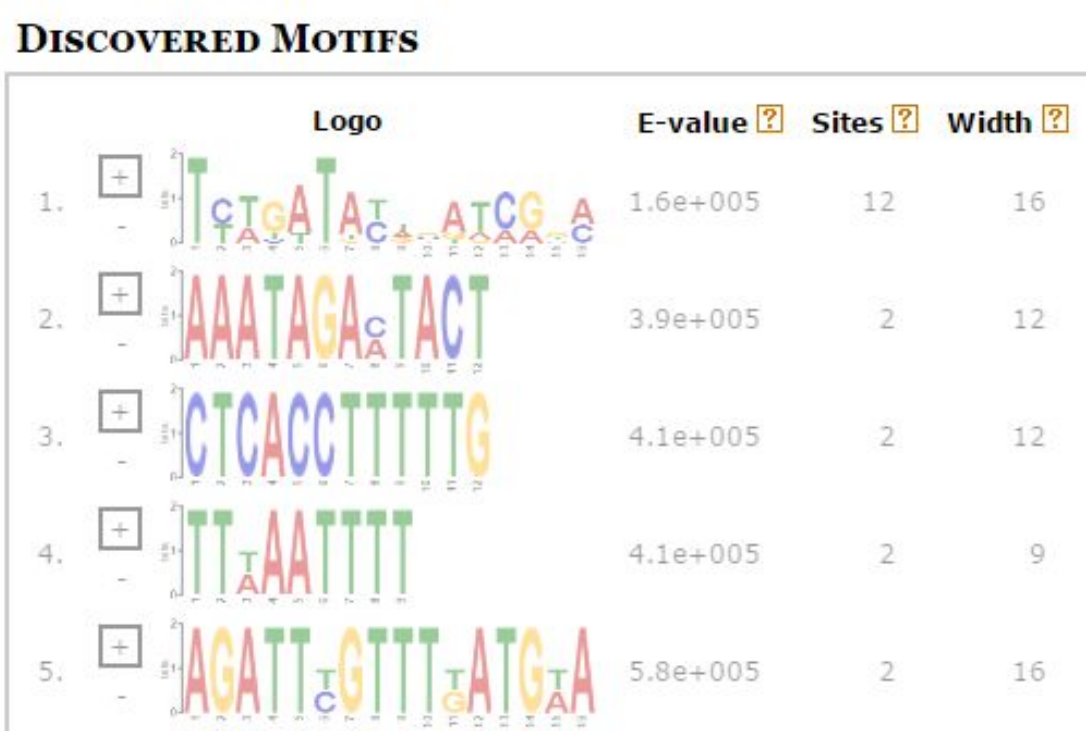
Time(seconds): 4.2

Summary**:** This trial was very accurate in that it found a vast majority of the motifs we were looking for. The only caveat was that on one of our 8-mers another character at the end. Because the original was intact and the last character was very ambiguous, we decided it was by chance that some characters were randomly inserted more than others. Because of the way we created

our code, it was possible for overwrites to occur. In this case, there was originally 99 insertions of our motif, but when running back through the fasta we found 93, which MEME found all of.

Large | 8mer | Sparse | 180 Sequence Length | 5000 Character Length | Shuffled

Type of Result:	*Count:*
True Positive	This trial showed no matches
True Negative	0
False Positive	Found five motifs at 20 sites
False Negative	Four, Since we missed all four of our known inserted motifs.



Time(seconds): 6.19

Summary**:** This trial was a very simple one, since we wanted to ensure that the shuffle was being done base pair by base pair. This means the entire sequence was shuffled around, and therefore we should not find any trace of our motifs that were placed before the shuffle. The motifs it did find were mainly just random.

Small | 8mer | Sparse | 180 Sequence Length | 2000 Character Length

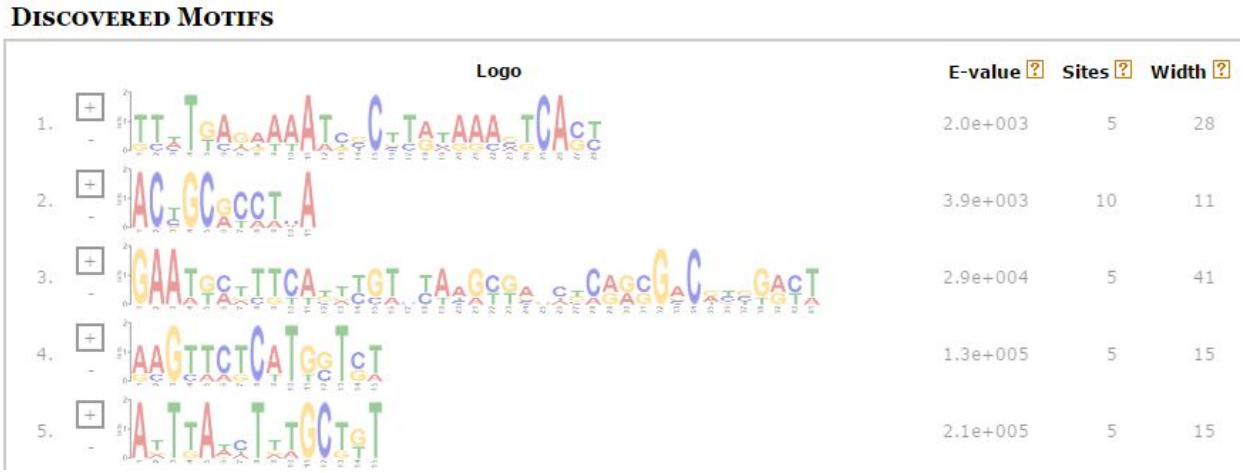
Type of Result:	*Count:*
True Positive	This trial showed no matches
True Negative	0
False Positive	Three additional motifs were found at 15 separate sites.
False Negative	Three, because we missed all three of our known inserted motifs.

Time(seconds): 23.47

Summary**:** This trial was pretty simple in that because it is a shorter kmer at 8 and that the motifs were placed sparsely it was harder for MEME to find that at only 2 matches max for one motif there was significance. Therefore it did not find that either of our motifs mattered statistically.

Large | 20mer | Sparse | 180 Sequence Length | 5000 Character Length

Type of Result:	*Count:*
True Positive	This trial showed no matches
True Negative	0
False Positive	25 additional motifs were found at 76 separate sites.
False Negative	4 of the sites from the two motifs we placed were not found

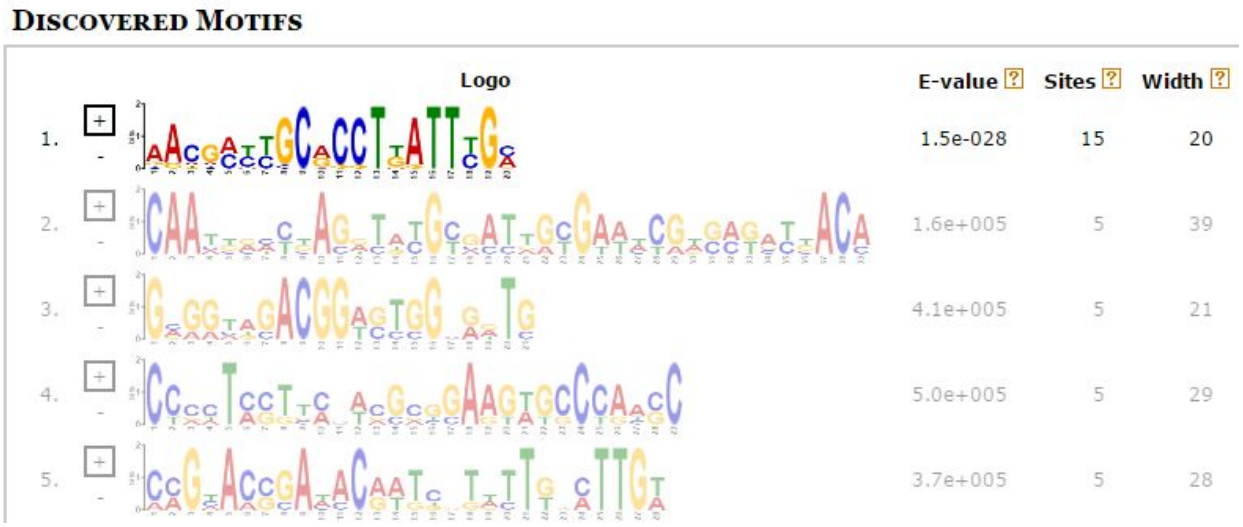


Time(seconds): 141.45

Summary**:** We found with most trials were the motifs were sparsely placed that MEME had a hard time seeing only two sites for a motif as likely to be significant. This surprised us as for a kmer of size 20, since the likelihood of there being two exact matches is very low in a sequence of only 5000 base pairs long. Therefore in our minds this test was fairly inaccurate.

Huge | 20mer | Sparse | 180 Sequence Length | 10000 Character Length

Type of Result:	*Count:*
True Positive	7 sites were found for one motif out of 14 total
True Negative	0
False Positive	13 motifs at 57 sites were found that were not placed by our code
False Negative	7 sites were not found (but were found within a different motif)

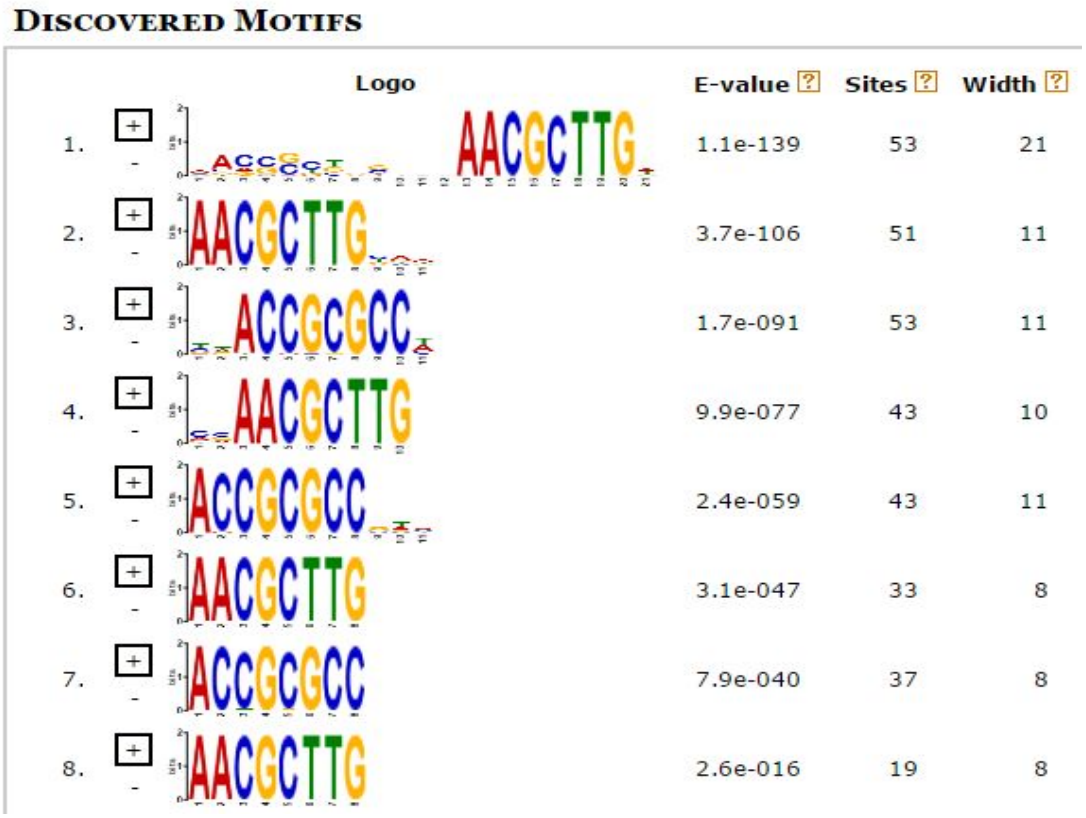


Time(seconds): 293.97

Summary**:** While 7 out of the 14 sites for one of the motifs placed within the sequences was found, the other seven were only partially found within a smaller motif that included only 16 of the base pairs. Because it was part of a different motif we believe that once sections of a sequence are used within a motif they may not be found as part of a different one within MEME. More testing would need to occur to ensure that this observation goes for all examples.

Large | 8mer | Saturated | 180 Sequence Length | 5000 Character Length

Type of Result:	*Count:*
True Positive	366 sites for our two motifs were found.
True Negative	0
False Positive	540 sites for 90 motifs were found that we did not place
False Negative	94 sites for our two motifs were not found.



Time(seconds): 595.72

Summary: Strangely enough, MEME actually separated some of the motifs away from each other even though they were the exact same kmer. Their E-Values were calculated as different

and each site is a true find of our kmer that we placed in, however it still decided to keep them apart.

Accuracy:

The accuracy of the benchmarks is shown above each picture in the tables. Overall, as the motif frequency increased (saturated), so did the accuracy. This makes sense, since MEME is finding more common motifs that appear very often, and is able to determine that these are motifs. However, as we inserted less and less motifs (sparse), MEME was still able to get the majority of our motifs, but often added a few extra base pairs before and/or after our actual motif. Although this is not terribly inaccurate, we can see that it still become less accurate the less often a motif appears. It appears as if having large or small .fasta files did not affect the accuracy in any major way, but it was more dependent on how often the specific motif we want was inserted. Increasing the size of the .fasta file, however, did lead to many more random motifs being found after our initial inserted motifs. Although MEME usually got our motifs correct, it would often find additional motifs that we did not intend for it to find. We found that MEME is most accurate when finding shorter motifs (8-14 in length) in a saturated file, and that it is least accurate when finding long motifs (18-20 in length) in sparse files.

Conclusion:

MEME uses statistical modeling techniques to automatically choose the best width, number of occurrences, and description for each motif. For many of the benchmarking runs, we noticed some interesting things. Firstly, sometimes MEME would discover our motifs, but add an

additional character or characters to the beginning or ends of the motif. For instance, if we inserted “AACGCTTG”, it may find “AACGCTTG(C/A/T)”. We believe this was just due to our randomly generated data placing a certain character more often than the others either before or after our motif, leading MEME to think it was part of the motif. This happens because the input length of the motif can be from eight to twelve, so if the motif is length eight, then there is a possibility of having extra remaining spaces, which eventually MEME could fill up with random variables either in beginning or end. Another similar case was meme did not just find the exactly same motif we expected, but it found a somewhat similar motif too. It aligned the sequence using an insertion/deletion technique like in local/global alignment.

We also notice that meme was finding a different motif for single long sequence. This should not be happening, as random chance dictates that we would not have many similar motifs of large lengths. These motifs would be long, but have a lot of variance in the actual motif, such as on page #9 and #10. We can attribute this to just a statistical anomaly, therefore cutting MEME some slack in the fact that true random means you will find patterns, however it was oddly long in some cases.

We found that using small, large, and huge data sets increased the time it took MEME to run a full motif discovery, but this was expected, as the files are larger, and it has to do more calculations. This is shown in our comparison of how long it took a small file to run when searching for 25 motifs (39.65 seconds), and how long it took for a huge file to search for 14 motifs (293.97). In addition, we also found that increasing the number of motifs for MEME to search for greatly affected time. This is because MEME must do a lot more calculations to discover more motifs, and usually does not pick exactly the same motifs. For example, using the

same small .fasta file, it took MEME 9.84 seconds to find 5 motifs. However, it took MEME 71.58 seconds to find 50 motifs. We can also see this in the fact that it took 595.72 seconds for MEME to find 100 motifs in our large file, but took 141.45 seconds to find only 25. So, both file size and motifs to search for both greatly affect how long it takes MEME to complete a motif find.

In conclusion, we found that MEME works best on saturated files with motif lengths of 8 to 14, and does these fairly quickly when only searching for 5 to 10 motifs total. We also found out that MEME does not work well when finding motifs of lengths 18 to 20 when searching for over 50 motifs total. These results are expected, as it makes sense from a computer science perspective, since doing more calculations with larger files will take longer and probably not be as accurate (in our case). However, we discovered some interesting things that MEME does that we do not want, such as not finding a motif of length 20 in a large file that is sparsely populated with our motifs. Although we expected our motif to be discovered at some point, since having 20 characters in a row match up between multiple sequences is not very likely at all, we expected MEME to find these motifs, even though there were only 4 to 8 per large and sparse file. However, we found that MEME did not find these, but found random other motifs of varying length instead. We attribute this to random generation, and how our data could have generated more common, but shorter, motifs that MEME was looking at instead. Overall, we were able to find how certain parameters change MEME to be better or worse at finding our motifs, what cases MEME fails or passes at, and how accurate MEME is with different parameters, which lets us see how we can potentially improve upon these errors in MEME.

Work cited:

Timothy L. Bailey, Mikael Bodén, Fabian A. Buske, Martin Frith, Charles E. Grant, Luca Clementi, Jingyuan Ren, Wilfred W. Li, William S. Noble, "MEME SUITE: tools for motif discovery and searching", Nucleic Acids Research, 37:W202-W208, 2009.