A hidden Markov model that finds genes in E.coli DNA Re-Implementation

Sophie Zheng (sz374), Matthew Xu (mx68)

Abstract

Hidden Markov Models are commonly used in the realm of machine learning. Previously, we explored variants of the Viterbi Algorithm, which is an algorithm that uses HMM to guess the most likely path through the states of the model. In the paper *A Hidden Markov Model that Finds Genes in E.coli DNA* (Krogh et al), the authors delineate two HMMs designed to find protein coding genes in E.coli DNA using E.Coli genome DNA sequences. These HMMs proposed include states that model the codons and their frequencies in E.coli genes, as well as other patterns, including overlapping start and stop codons, as well as different intergenic models. The paper states that the parameters for the HMM are generalized across the dataset, and the model was able to achieve 80% accuracy for the exact locations of known E.coli genes, and 10% for approximate locations. In this project, we will attempt to reimplement both the models proposed.

1. Introduction

The problem of sequencing genomes of organisms and organelles has and will continue to produce large quantities of complex map and DNA sequence data. The development of algorithms is crucial in interpreting these data in a robust and automated manner. The method described in (Krogh et al) is one of such algorithms attempting to automate this tedious process. E.coli DNA sequences have been historically thoroughly studied and annotated in the field of genomics, and this made it easier for the authors of this paper to evaluate the performance of machine learning methodologies. In this paper, we will re-implement the models they proposed, and evaluate the two models presented. Originally, (Krogh et al) utilized the EcoSeq6 database, since this was one of the most well maintained

E.coli sequences when the paper was published. However, we will be utilizing a different dataset since EcoSeq6 is not publicly available.

2. Background

(Krogh et al) suggest to model E.coli genome structure with a Hidden Markov Model (HMM) and use the Viterbi Algorithm to find the most likely path given the DNA sequence. Once the path is found, we can easily locate the start and stop codon states, and thus find the location of the genes. However, due to the complexity of the structure of E.coli genomes, the HMMs in question require almost a thousand hyperparameters. We combine some data presented in (Krogh et al), and give a rough initial guess for the rest of the parameters. Then, we use the Baum-Welch algorithm, a special case of the Expectation Maximization algorithm (EM), to fit the parameters to the training data. In this section, we will discuss the background of these algorithms mentioned above.

2.1. Hidden Markov Models (HMM)

Hidden Markov Model (HMM) is a statistical model that represents systems which are assumed to be a Markov process. These models are utilized in our previous study on high and low GC contents within DNA strands. This problem required a simple HMM: one that contains only a “high” state and a “low” state.

Our job here is similar: there are states representing the genes, and states that represent the intergenic regions. We want to use the Viterbi Algorithm, which we will discuss in the next subsection, to find the most likely path through the HMM given the DNA sequence. Since the path will contain states that mark the start and end of a gene, it will be simple to locate the genes in the E.coli genome sequence.

2.2. Viterbi Algorithm

Viterbi Algorithm is a dynamic programming algorithm that finds the most likely path given the data sequence and the HMM that models the system. It makes a DP table of size (number of states, length of sequence) and fills it accordingly. For each step, Viterbi computes the probability of arriving at this state given the previous maximum, as well as the probability of this state emitting the corresponding base. Algorithm 1 demonstrates the pseudocode for the Viterbi Algorithm.

Algorithm 1 Viterbi Algorithm

```
Input: sequence $x$, HMM states $st$
Initialize empty dp table $V$ of size $st$ by length of $x$
Initialize $V_k(1) = a_{0k}e_k(x_1) = P(z_1 = k)P(x_1|z_1 = k)$
for $l$ in range(len($st$)) do
    for $i$ in range(len($x$)) do
        $V_l(i) = e_l(x_i) \cdot \max_k(V_k(i-1) \cdot a_{kl})$
    end for
end for
Return $\max_z P(x, z) = \max_k V_k(\text{len}(x))$
```

Algorithm 2: The Baum-Welch Algorithm

```
Initialization: $\Theta_0$, $\{O_{1:T}\}$
for $l = 1, \ldots, l_{\max}$ do
    1. Forward-Backward calculations:
        $\alpha_1(i) = \pi_i b_i(O_1),\ \beta_T(i) = 1,$
        $\alpha_t(i) = \Big[\sum_{j=1}^{K} \alpha_{t-1}(j)a_{ji}\Big] b_j(O_t),\ \beta_t(i) = \sum_{j=1}^{K} a_{ij}b_j(O_{t+1})\beta_{t+1}(j)$
        for $1 \le i \le K,\ 1 \le t \le T-1$
    2. E-step:
        $\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{K}\alpha_t(j)\beta_t(j)},\ \xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1}\beta_t(j))}$
        for $1 \le i \le K,\ 1 \le j \le K,\ 1 \le t \le T-1$
    3. M-step:
        $\pi_i = \frac{\gamma_1(i)}{\sum_{j=1}^{K}\gamma_1(j)},\ a_{ij} = \frac{\sum_{t=1}^{T}\varepsilon_t(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T}\varepsilon_t(i,k)},\ w_{kd} = \frac{\sum_{t=1}^{T}\gamma_t(k,d)}{\sum_{t=1}^{T}\sum_{r=1}^{D}\gamma_t(k,r)}$
        for $1 \le i \le K,\ 1 \le j \le K,\ 1 \le k \le K, 1 \le d \le D$
end
Result: $\{\Theta_l\}_{l=0}^{l_{\max}}$
```

2.3. Expectation Maximization (EM)

Expectation maximization, or more specifically the Baum-Welch algorithm in our case is an iterative algorithm used to compute the maximum likelihood estimates of unknown parameters for a given hidden markov model utilizing the forward-backward algorithm. The Baum-Welch algorithm was utilized previously in problem set 5 to find the transition and emission probabilities of the hidden markov model for locating GC rich and GC poor regions. Similarly, in our reimplementation, we utilize the Baum-Welch algorithm to find the transition and emission probabilities of the states of our Hidden Markov Model. Algorithm 2 demonstrates the pseudocode for the Baum-Welch algorithm.

3. Dataset

The original dataset utilized by (Krogh et al) was the EcoSeq6 database. The authors utilized this dataset for its proper maintenance and previously existing annotations and also because the authors were in close contact with the owner of the database. Since the EcoSeq6 database is not publicly available and we did not have the means to contact the owner, we instead used the E.coli K-12 MG1655 DNA sequence found publicly available online on the University of Wisconsin ASAP database. This sequence is thoroughly maintained by the University of Wisconsin-Madison staff and the entire sequence has been annotated since 1997 but continues to be updated even to this day as new genomes are discovered. The sequence is 4641652 nucleotides long, but due to computational limitations, we will be utilizing only a small portion of this sequence, namely the first 20000 nucleotides.

4. Simple Intergenic Model

The Simple Intergenic Model is an HMM that attempts to model the structure of E.coli genomes. It contains the following structures: start codons, stop codons, intergenic regions, 61 smaller HMMs that represent the codon triplets that are not stop codons, and a central state to tie everything together. The codons are split into 3 or more states so that each state emits one base. The 61 smaller HMMs have identical structure, and aim to model individual codons. To take into consideration the possibility of insertions and deletions within the codon HMMs, (Krogh et al) adds "circular" states to represent deletion, and "diamond" states to represent insertion. The structure of these small HMMs are shown in Figure 1.

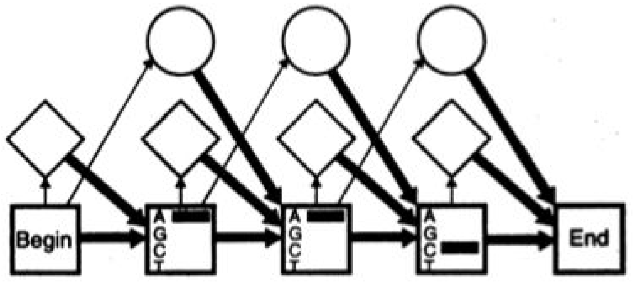

Figure 1. Structure of codon HMMs (AAC) proposed in (Krogh et al)

In this section, we will discuss the implementation of the Simple Intergenic Model, as well as other related algorithms, and modifications to the original model described above.

4.1. Model Creation

Since our ultimate goal is to use the Viterbi Algorithm on the HMM we create, the original model proposed by (Krogh et al) became problematic. This is because there are states that do not emit anything, like the central state and the circular states within the 61 codon HMMs, and this complicates the dynamic programming process: if there are "blank" states, which "previous" state do we go to? Thus, we decided to simplify this problem by deleting these states from the original model. Before we discuss the new edges created as a byproduct of this deletion, let us first talk about the states in this model.

There are three standard stop codons, including TAA, TGA, and TAG. We represent TAA and TGA as a group, and TAG as another group due to typical frequencies, as (Krogh et al) described. Thus, there are 6 states that represent the stop codons. Similarly, we represent all three standard start codons, including ATG, GTG, and TTG, as one group. Thus, there are 3 states that represent the start codons. There is 1 state representing the intergenic region. Finally, because of the simplification described above, each codon HMM is represented by 7 states, with 3 being the codon base triplet, and the rest representing the possibility of insertion. Thus, there are a total of 437 states in this simplified model.

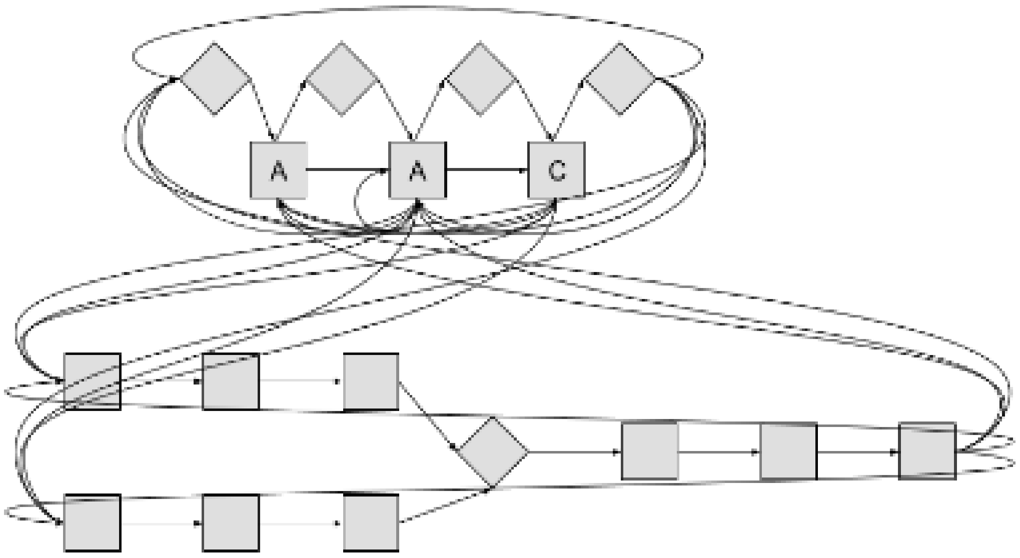

Figure 2. Modified Structure of the Simple Intergenic Model (with codon AAC as an example and the reset of the 61 codon HMMs omitted)

Because of the deletion of the central state and the circular states in the codon HMMs, as well as the "begin" and "end" states in the codon HMMs, we need to add many more edges in addition to the original edges proposed in (Krogh et al). For example, all "square" states in the codon HMMs must connect to all "square" states to the right because of the deletion of the "circular" states; the last "square" state, the last "diamond" state in all codon HMMs, the second "square" state, and the last start codon state must connect to the first "diamond" state, the first “square” state, the second “square” state of all codon HMMs, as well as the first states of the two groups of stop codons. This is portrayed in Figure 2.

4.2. Model Implementation

In addition to the creation of the states as described in subsection 4.1, the final step to creating the Simple Intergenic Model is creating the transition probability matrix, the emission probability matrix, and the initial probability matrix. We first create a NumPy array of zeros, representing disjoint states. Then, we modify the entries for all the edges in the graph as shown in Figure 2. We calculate the probabilities with the following parameters:

We set p_{indel}, the probability of insertion and deletion in the codon HMMs, to 10^{-8}, as given by (Krogh et al). Through parsing our data, we found that n_{codon}, the average number of codons per gene, is 313. Thus, p_{gene}, the probability of staying in the 61 codon HMMs is $1 - \frac{1}{n_{codon}}$. In addition the probability of staying in the intergenic state is 0.9879, obtained by taking the average of the length of the intergenic regions in the training data. In addition, parameters like emission rates of the intergenic region and relative frequencies of start and stop codons are all parsed from the training data.

Because the sequence can start in any state of this model, we set the initial probabilities to be equal. Thus, each state had initial probability of $\frac{1}{437}$.

Finally, we pipeline what we constructed into the Viterbi Algorithm. The implementation is simple, and can be referenced from our previous study of the GC contents in DNA. After obtaining the most likely path, we simply filter for the first start codon state or the last stop codon states and return the indices. This algorithm will also be used for the Complex Intergenic Model in subsection 5.2.

5. Complex Intergenic Model

The Complex Intergenic Model is an HMM that attempts to model the structure of E.coli genomes. It contains all the structures of the Simple Intergenic Model, as well as a long intergenic model, a short intergenic model, an overlap model of overlap length 1, and an overlap model of overlap length 4. In this section, we will discuss the implementation of the Complex Intergenic Model, as well as other related algorithms, and modifications to the original model described above.

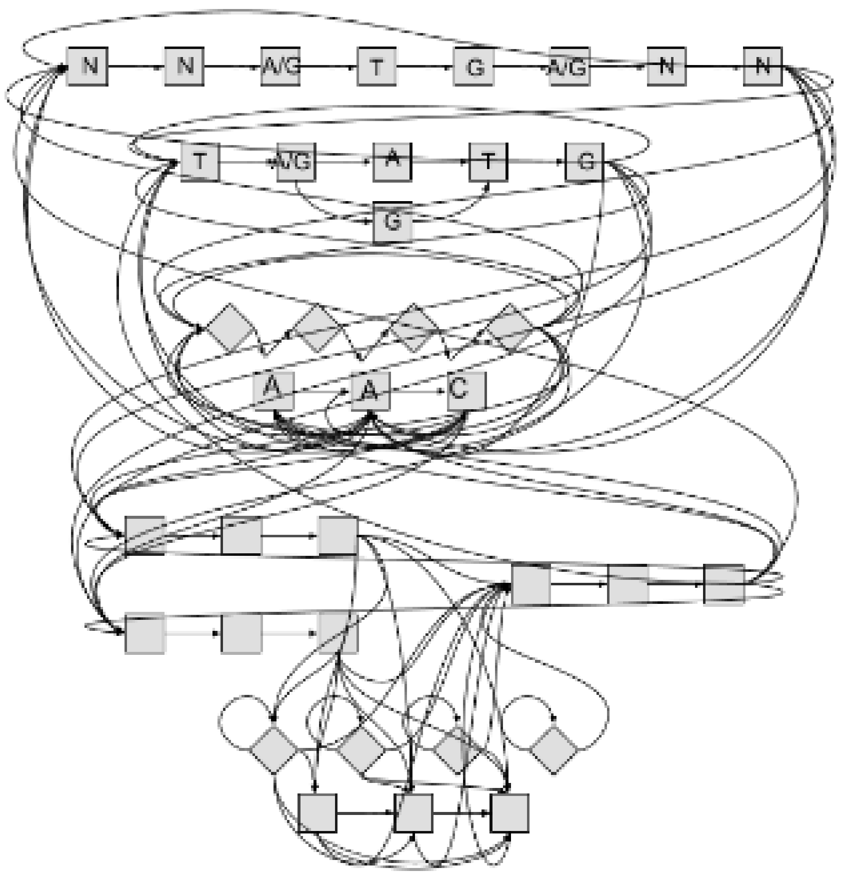

Figure 3. Modified Structure of the Complex Intergenic Model (with codon AAC as an example and the reset of the 61 codon HMMs omitted, as well as a shortened intergenic region)

5.1. Model Creation

The creation of the Complex Intergenic Model includes all components discussed in the Simple Intergenic Model (subsection 4.1). However, the complex model has an additional overlap model, as well as a long intergenic model and a short intergenic model (Krogh et al). Similar to the construction of the simple model, we had to remove all nodes that did not emit anything, so the resulting model is shown in Figure 3.

(Krogh et al) mentioned that start and stop codons generally have overlap of 1 or 4 genes. Thus, we make this into two different groups (as shown at the top of Figure 3). The overlapping genes of 1 tend to be either TA(/G)ATG or TA(/G)GTG, while the overlapping genes of 4 tend to fall in the regex of NNA(/G)TGA(/G)NN, where N represents any of the 4 bases.

In addition, the long and short intergenic regions (as shown at the bottom of Figure 3) have the same structure, just different lengths (shown in Figure 4). The long intergenic region is described to have 44 square states and 45 diamond states, while the short intergenic region has 9 square states and 10 diamond states (Krogh et al). Figure 3 only shows one such region for simplicity.

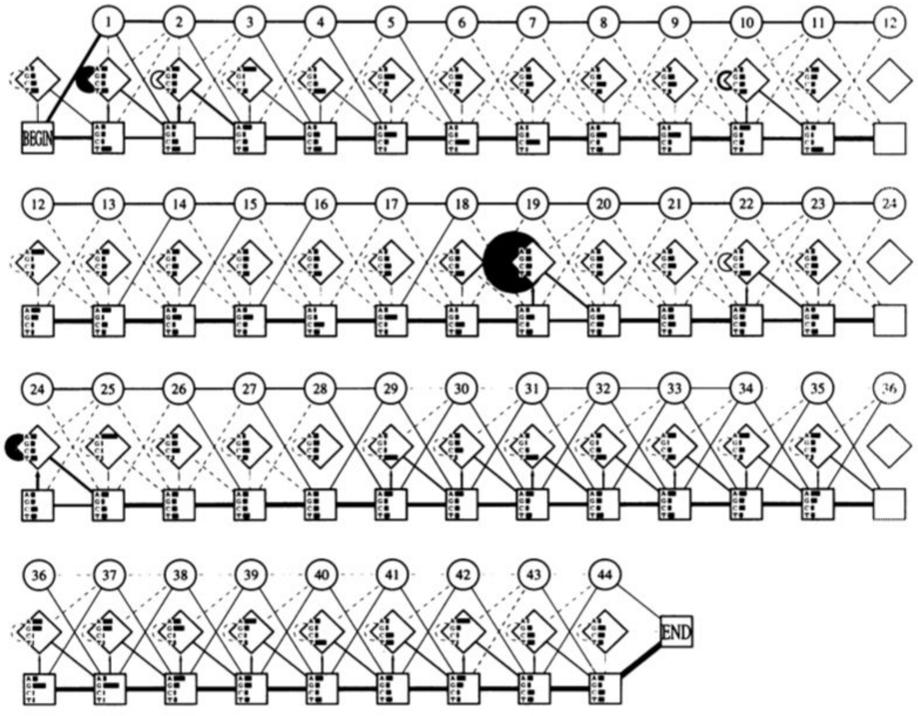

Figure 4. Structure of the long/short intergenic regions from (Krogh et al)

In addition to the 436 states (without the intergenic state) from the Simple Intergenic Model, we add in the newly constructed states: for the long intergenic model, we have 44 square states and 45 diamond states; for the short intergenic model, we have 9 square states and 10 diamond states. Finally, the overlap model has 14 states. Thus, there are 558 total states.

5.2. Model Implementation

Because no details on parameters were presented in (Krogh et al) for the long and short intergenic models, as well as the overlap models, we had to guess the parameters, and hope that EM improves them. We started by guessing equal probabilities for all unknown transition probabilities, as well as emission probabilities in the intergenic regions. Then, we ran EM on these probabilities. However, due to the size of both the dataset and our model, this was not a good plan by nature. The runtime of one iteration of EM is the runtime of the forward-backward algorithm plus the runtime of EM, which gives $O(2Tn + Tn^2 + Tn) = O(Tn^2)$, where T is the length of the data (4 million characters), and n is the number of states (558 states). This is only one epoch of the EM algorithm. In addition, due to limited memory, we cannot compute all 4 million nucleotides at once. Thus, we had to split the data up into segments of 1000s, and that gave $O(1mil \cdot n^2)$ per epoch. We will discuss this issue in detail in Section 6.

6. Experiments

We ran our simple and complex models on the first 20000 nucleotides of the E.coli K-12 MG1655 DNA sequence. Due to computational limitations, 20000 nucleotides was our limit seeing as how EM ran for over 10 hours on our systems. Since we did not need to run EM for the Simple Intergenic Model (Krogh et al provided most of the parameters needed), this particular experiment was not bounded by time. However, for better comparison with the complex experiment, we still ran it on the first 20000 nucleotides. In each case, we keep track of the same four metrics as (Krogh et al): genes found with perfect nucleotide matches, genes found within 10 nucleotides of the actual gene, genes found with at least 60 overlapping nucleotides, and finally genes completely missed.

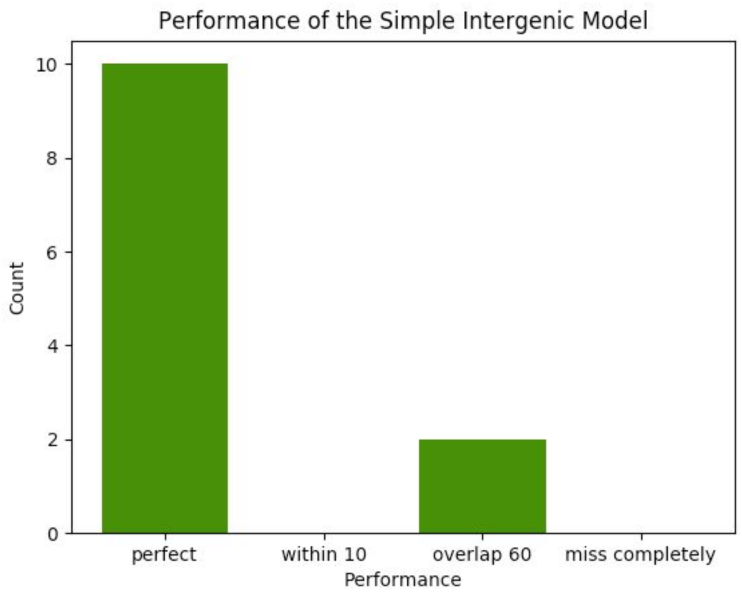

Figure 5. Performance of the Simple Intergenic Model on the four metrics defined by (Krogh et al)

6.1 Simple Intergenic Model

On the first 20000 nucleotides, our simple intergenic model seems to have performed very well. After preprocessing the genes found in the first 20000 nucleotides (provided by the annotations from the University of Wisconsin-Madison data) by removing genes without proper start codons or stop codons as described in (Krogh et al), we have 12 annotated genes. Our simple intergenic model located 10 of the 12 genes perfectly and overlapped with the other 2 genes by over 60 codons (see Figure 5). Thus, we have that the Simple Intergenic Model was able to achieve 83.3% perfectly aligned genes, and 16.7% with at least 60 overlapping bases. This is certainly on par with, if not better than, the results noted in (Krogh et al).

6.2 Complex Intergenic Model

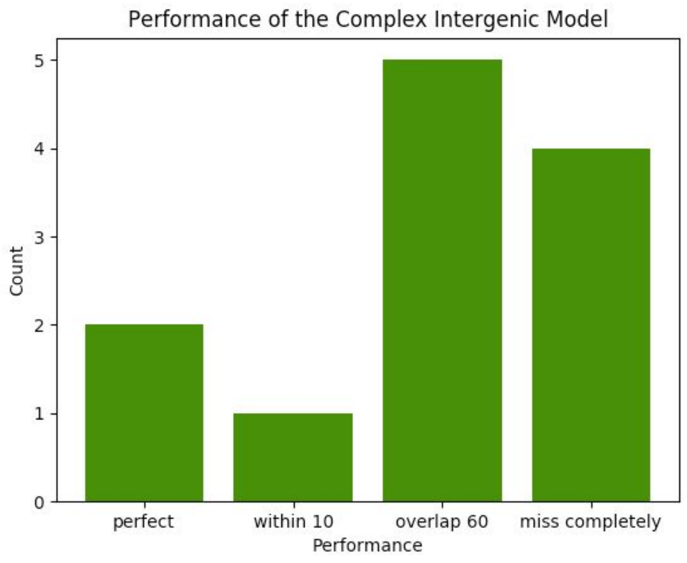

Figure 6. Performance of the Complex Intergenic Model on the four metrics defined by (Krogh et al)

As mentioned in subsection 5.2, due to limitations of computing power and time, and the complex nature of the Complex Intergenic Model, we were only able to run the EM algorithm for two epochs on only the first 20000 nucleotides (one epoch took over 12 hours). Because our implementation was in NumPy, we were also unable to efficiently utilize GPU hardware acceleration. We are fairly certain that 2 epochs of EM does not reach convergence, and thus, we believe that the suboptimal parameters for the Complex

Intergenic Model will perform suboptimally. We preprocess the first 20000 nucleotides, as described in subsection 6.1, and have 12 annotated genes.

As shown in Figure 6, the Complex Intergenic Model (with barely any training) was able to achieve 16.7% perfectly aligned genes, 8.3% within 10 nucleotides of the start codon, 41.7% overlap of at least 60 bases, and 33.3% complete misses. This definitely does not match the proposed 80% perfect accuracy from (Krogh et al), but was achieved under very limited conditions. We believe that if we had the computational power and time to fully run EM until it converges, this model will achieve better results.

7. Conclusion

As we've discovered, the hidden Markov Models as described in (Krogh et al) correctly identifies most of the genes in the E.coli MG 1655 DNA sequence. Despite the paper being written over two decades ago, the model still performs well on the most up to date E.coli annotations. Our simple model, utilizing the parameters as provided by the paper and found in our dataset, achieved nearly perfect results on the first 20000 nucleotides. We hypothesize that our model would achieve similar results to those found by the paper, achieving a roughly 80% accuracy in finding genes perfectly in the DNA sequence. As (Krogh et al) mentioned, this high accuracy is achieved at the cost of a large number of false positives, or unidentified sequences.

Due to limited computing power and time, we were not able to finish running EM on the complex model. We were also unable to use parameters from (Krogh et al) like we did for the simple model, since the paper omitted all details on the complex model. Nonetheless, our complex model was able to find some genes perfectly, and most genes within reasonable bounds, even though it performed poorly in comparison to our simple model. We theorize however, with more training with the EM algorithm (until parameter convergence), that the complex model will achieve better results. This is because despite the lower accuracy in comparison to the simple model, the complex model produced significantly fewer false positives.

During our experimentation, we found that our Hidden Markov Models also identified a number of sequences (not annotated as genes by our dataset) as genes. While it is unlikely that these sequences are new undiscovered genes in the E.coli MG 1655 DNA strand considering the strand's thorough annotation history, we hypothesize that it is through similar efforts how new genes were discovered in the past and in the present.

References

Krogh, A., Mian, I. S., & Haussler, D. (1994). A hidden Markov model that finds genes in E. coli DNA. Nucleic acids research, 22(22), 4768–4778. https://doi.org/10.1093/nar/22.22.4768

Glasner, Jeremy & Liss, Paul & Plunkett, Guy & Darling, Aaron & Prasad, Tejasvini & Rusch, Michael & Byrnes, Alexis & Gilson, Michael & Biehl, Bryan & Blattner, Frederick & Perna, Nicole. (2003). ASAP, a systematic annotation package for community analysis of genomes. Nucleic acids research. 31. 147-51. 10.1093/nar/gkg125.