DNA Region Grammatical Inference

Cory Cook
Computer Science, San Jose State University
San Jose, California, United States
corycook@ieee.org

***Abstract*—An accurate language definition capable of distinguishing between coding and non-coding DNA has important applications and analytical significance to the field of computational biology. The method proposed here uses positive sample grammatical inference and statistical information to infer languages for coding DNA.**

I. Introduction

There is a large amount of gene sequence data available and many methods are currently available for analyzing and interpreting that data. However, little is understood about the specific structure of nucleotide sequences. Erwin Chargaff first discovered rough statistical patterns in DNA when he discovered his parity rules: 1) DNA will have 1:1 ratio of pyrimidine and purine bases and 2) each strand of DNA will have approximately 1:1 ratio of pyrimidine and purine bases. This first of Chargaff's rules went on to help develop the double helix structure of DNA as we understand it today [1]; however, the reason behind the second rule is still mysterious. Chargaff went on to say, "[f]or I saw before me in dark contours the beginning of a grammar of biology" [2]. He was speaking of expression of DNA as a language and interpreting its grammatical structure relating the process to those similar in other contexts.

Little progress has been made in determining the specific nucleotide patterns of DNA. Many have discovered statistical patterns in the nucleotide and oligonucleotide emission frequencies [3] [4]. These statistical patterns show that DNA has patterns independent of the specific species. There has been success in identifying the major components of DNA through observation of functional behavior and sequence comparisons at different stages of transcription and translation. There has also been success in being able to locate these components of DNA using statistical information and analysis of the sequences. However, an accurate language definition for DNA or its components has not yet been created.

An accurate language definition for DNA would have important analytical and applicable value to the field of computational biology. Being able to determine the language of DNA would provide application to gene sequencing, gene finding, and gene analysis. A sufficiently accurate language definition could also provide additional insight to the functionality of components of DNA and provide additional evidence to support or refute current theories about these components as some of the functionality is still unknown.

A method is proposed here to infer generalized regular grammars for annotated coding sequences of DNA that allow differentiation between coding and non-coding sequences in the human genome. The results and example here show how the languages for these components are both useful and accurate for application in computational biology.

II. Theory

Until recently it was believed that gene positions in eukaryotic DNA were random due to the fact that chromosomal domains were not necessary and chromosomal inversions had little effect on genetic expression [5]. Recent studies show that genes with similar expression cluster more often than randomly [6]. During protein synthesis genes are copied from the DNA to mRNA. This process looks for a promoter on the DNA and begins copying at the start codon until it reaches the stop codon and finishes transcription. The strand is then put through an RNA splicing process that removes all of the introns from the transcribed mRNA. Intron sequences can be determined by aligning the mRNA sequence after RNA splicing with the source gene in the DNA. Once the two sequences are aligned, the missing regions indicate the introns [7]. Ribosomes synthesize proteins utilizing gene sequences independent of their genomic context; therefore, valid gene analysis independent of its genomic context should be possible.

Gene identification and structure prediction are key problems in computational biology. The current industry standard for gene identification is through the use of Hidden Markov Models. A Hidden Markov Model (HMM) utilizes a set of hidden states that transition between each other with a predefined probability. At each state transition there is a visible emission from that state. The form of the emission is predicted by some probability based on the current hidden state of the HMM. The N-order Markov property assumes that the current state is only dependent of the previous N states. You can then use the HMM with Viterbi, Baum-Welch, and Forward-Backward algorithms to determine probabilistic information about a sequence of emissions. For example, in the case of gene finding an HMM may indicate a high probability of being in a gene or exon at a particular position in the genomic sequence [8].

GenMark was developed to find genes in Escherichia coli and was the first gene finding software that utilized HMM. The current standard software for gene finding is GENSCAN which is also an HMM-based application that utilizes a large amount of structural information about the human genome in the development of its model [9]. GENSCAN is very accurate compared with other gene finding software; however, it still

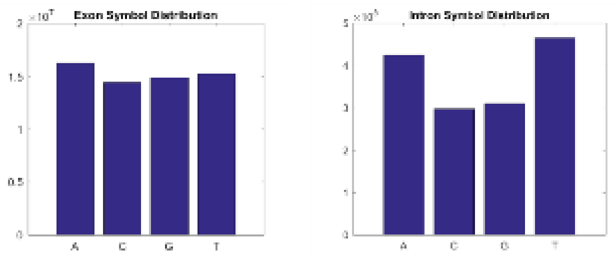

Figure 2: Comparison of symbol distribution in human exon and intron sequences

misses nine percent of all exons and five percent of its predicted exons do not overlap any actual exon. GENSCAN accuracy falls off for longer sequences and most HMM-based software have trouble detecting splice sites between introns and exons. To improve the accuracy beyond these limitations we should explore other methods of pattern analysis.

Genes contain the coding regions of DNA that provide the sequences necessary for the synthesis of proteins. Statistical analysis has revealed that coding regions are structurally distinct from the non-coding regions of DNA [4]. Counting the nucleotides in human genes reveals that the nucleotide distributions are not equal for exon and intron sequences. This indicates that there may be a pattern difference between genes and intergenic regions or introns and exons.

Hidden Markov Models detect patterns in sequential data through local probabilistic information; however, there is also a language or pattern expressed by DNA that identifies its structure and behavior. E Mark Gold introduced language identification in the limit stating that an adequate algorithm will eventually successfully learn a language when provided a complete presentation including both examples belonging to the language and examples that do not belong to the language for examples defined by a finite alphabet [10]. However, Dana Angluin went on to prove that languages can be identified using only positive examples “drawn independently according to some probability distribution” [11]. There are several algorithms for identifying regular languages based on Gold’s theorem that utilize both positive and negative samples and a couple based on Angluin’s findings. These algorithms accomplish the language learning task through state-merging finite automata.

Finite automata are theoretical constructs composed of an alphabet ∑, a finite set of states Q, a set of final or accepting states F where F is a subset of Q, an initial state q_0, and a transition function $\delta(q, a)$ that indicates the next states given the current state q and an input symbol a. An alphabet is a finite set of symbols. Given an input sequence composed of symbols in the alphabet the states can be followed using the transition function. The sequence is accepted if the terminus is at an accepting state and rejected otherwise. Finite automata are deterministic if their transition function always returns a single state [12]. ∑* is the set of all finite sequences composed of symbols in the alphabet ∑. The language expressed by finite automata are the sets of all finite strings accepted by the automata and are a subset of ∑*. For nucleotide DNA the alphabet is defined as ∑ = {A, T, C, G}.

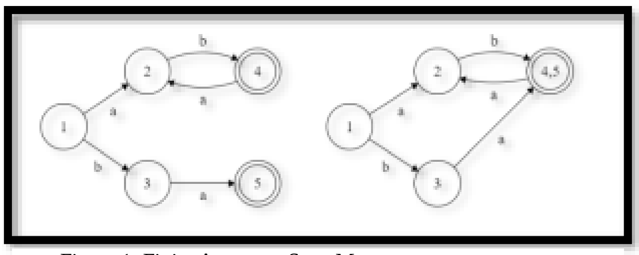

Figure 1: Finite Automata State Merge

The automaton prior to the merge accepts the language ba|ab(ab)* which is a subset of the language accepted after the merge: (ba|ab)(ab)*. Image generated using Finite State Machine Designer [21].

State merging on finite automata results in generalized automata that accept at least as many strings as the unmerged automata. Meaning, that the language expressed by the unmerged automata is a subset of the language expressed by the merged automata (see Figure 1). In this sense, the goal of language learning is to generalize and express the language presented in the positive samples and not the negative samples.

Genetic analysis using finite automata will attempt to learn a language for coding regions and non-coding regions and compare the accuracy of the deterministic automata generated based on positive and negative samples [4]. The positive samples will come from either coding or non-coding regions based on the target automata while the negative samples will come from the opposing set.

III. HYPOTHESIS

Given the theory, the following hypotheses are presented:

1. A language definition can be inferred for coding or non-coding DNA that will differentiate target sequences from their counterparts.
2. A language inferred for coding or non-coding DNA will be consistently accurate for any uniformly randomly sampled sets of test sequences from the genome.

IV. METHOD

In order to analyze sequence information about DNA we need to express it through symbolic information. DNA can be expressed as a sequence of characters where the characters represent nucleotides or some abstraction of the nucleotides. Nucleotide representation can be expressed using only one symbol for each of the four nucleotides. Nucleotide representation can be abstracted to codon representation where each symbol corresponds to three consecutive nucleotides. Since all permutations of the four nucleotides are possible there are sixty-four possible symbols in codon representation.

```
   function SearchAlergia(S, P, N, m, α)
Input: A positive training sample set
      S, a positive test sample set
      P, a negative test sample set
      N, a bounding value m, and an
      ALERGIA alpha confidence value
      alpha
Output: The finite automata with the
      highest accuracy, the subset
      used to generate the automata,
      and the accuracy of the
      automata
   size = number of samples in S
   if size <= m then
       fa = ALERGIA(S, α)
       a = Test(fa, P, N)
       return (fa, S, a)
   else
       p = $\left\lfloor \frac{size}{2} \right\rfloor$
       (bfa, bS, ba) =
SearchAlergia(S(1:p), P, N, m, α)
       (tfa, tS, ta) =
SearchAlergia(S(p+1:size), P, N, m, α)

       subset = Merge(bS, tS)
       fa = ALERGIA(subset, α)
       a = Test(fa, P, N)

       if ta > a and ta > ba then
           return (tfa, tS, ta)
       else if ba > a then
           return (bfa, bS, ba)
       else
           return (fa, subset, a)
       end if
   end if
   end function
```

Codon representation can be further abstracted to amino-acid representation. The sixty-four codons only code for twenty-two amino acids so the number of symbols necessary for this representation is much smaller. Amino acid representation can be abstracted to amino acid group representation to bring the number of symbols down to five. However, amino acid group representation does not express all of the information in the sequence; amino acid representation is only applicable to exon sequences that actually code for amino acids, and codon representation is only applicable to RNA. Nucleotide representation expresses all of the information available and applies to all components of DNA.

Most language learning algorithms begin with the building of prefix tree acceptor (PTA) deterministic finite automata (DFA) using the positive input samples. The basic method of building a PTA is for every sequence in your positive set and every symbol in that sequence if a node does not exist for the subsequence then add a new state for that symbol and transition to that state from the state indicating the sequence prefix of the symbol. The end result will be a DFA that accepts all of the sequences in the positive example set. The language learning algorithm then iterates through pairs of states and decides whether to merge the two states or not.

Algorithms based on Gold’s method usually merge the nodes then determine whether the merged automata are acceptable by checking against the negative examples. One example of this approach is Regular Positive and Negative Inference (RPNI) [13]. Algorithms based on Angluin’s findings use a statistical approach to determining merge compatibility. The method applied here utilizes the ALERGIA algorithm [14] that uses Heoffding’s probability inequality for sums of bounded random variables as its merge constraint [15]. Tests will be done using the ALERGIA algorithm; however, there is a dependency in the algorithm on the test samples provided. To help overcome this dependency and optimize the results obtained the SearchAlergia algorithm is proposed.

SearchAlergia recursively divides the sample set into smaller subsets until the set size is below a bounding value provided. It then runs ALERGIA on each of the small sets. It then merges two smaller subsets and runs ALERGIA on the merged set. The most accurate of the three subsets is promoted to the next level to be merged and tested. Eventually the most accurate subset found is promoted to the top and returned. The algorithm does not provide an exhaustive search; however, it does search many possible subsets in a reasonable amount of time.

The only requirement of the Test function used in the SearchAlergia algorithm is that it returns a reasonable value that is comparable with other values having larger values more desirable. Here the Test function will return the average conditional probability (ACP) of the language correctly identifying a sequence in the positive and negative test sets [16]. When testing the sequences in the test sets against the automata generated: positive sequences accepted are true positives (TP), negative sequences accepted are false positive (FP), positive sequences not accepted are false negatives (FN), and negative sequences not accepted are true negatives (TN). ACP is then calculated using the following formula [17]:

$$ACP = \frac{1}{4}\left[\frac{TP}{TP+FP} + \frac{TP}{TP+FN} + \frac{TN}{TN+FN} + \frac{TN}{TN+FP}\right]$$

V. Design of Experiment

The experiment begins with the testing of the second hypothesis. The goal of the test is to determine that the relative accuracy is maintained across many different test sets of varying sizes. A random sample will be collected from the positive set to infer the language. The average conditional probability will then be calculated for the inferred language against randomly sampled positive and negative samples of varying size. The test is repeated many times and the average is taken to resolve the overall behavior. If the results show that the ACP is approximately equivalent for all test sample sizes then the test is considered successful and the second hypothesis is supported by the evidence.

The goal for testing the first hypothesis is to show that the inferred language can distinguish between the positive and negative samples. The language has to be able to correctly classify a sequence with average conditional probability better than random. The results will need to be averaged over many test cases to capture the true behavior and filter noise. If the average ACP is greater than 50% then the test is considered successful and the first hypothesis is supported. If the average ACP is significantly higher than 50% then the first hypothesis is heavily supported by the evidence.

The SearchAlergia algorithm is tested against the ALERGIA algorithm to indicate any improvement in the ACP and running time of the algorithm. Each algorithm is run using an increasing number of input samples as the running time of the ALERGIA algorithm is dependent on the number of states in the PTA. The results are the averaged over many test cases. If the results show an improvement in ACP, then the SearchAlergia algorithm is considered an improvement over the basis algorithm for optimizing ACP and the results from SearchAlergia will be used to support the first hypothesis.

VI. EXPERIMENT

This experiment uses known gene data for the human genome. The sample sets come from exons as the coding regions and introns as the non-coding regions of the genome. The data is downloaded using the University of California Santa Cruz Table Browser application [18]. The data is downloaded in such a way as to simplify parsing the genes into coding and non-coding samples.

The grammatical inference toolbox (gitoolbox) is a set of algorithms written for MATLAB that perform automata approximation and state merging algorithms [19]. Among the algorithms implemented are RPNI and ALERGIA. This experiment uses the ALERGIA algorithm implemented in the toolbox with modification (see Appendix A: Improving gitoolbox).

Introns and exons are sampled from the genome by uniformly random distribution. This is due to gene expression and structure similarity based on proximity in the genome. Positive example learning methods based on statistical merge criteria perform best when repetitive elements are minimized and a more representative example of the language can be provided [14].

Intron languages will be inferred for the purposes of providing evidence for second hypothesis; however, intron sequences will not be used for the purposes of providing evidence for the first hypothesis. This is due to the nature of annotated intron sequences to always begin with GT and end with AG. The language defined by GT[ACGT]*AG has 94% ACP for identifying intron sequences; however, this language does not provide useful application or analysis as the internal structure of the intron is still unknown (see **Error! Reference source not found.**). Application of this process to intron language will require additional preprocessing or an alternate accuracy metric outside of the scope of this experiment. Inferred languages for introns are still useful and are accurate as evidence for the second hypothesis; however, optimizing on ACP leads to an overgeneralized language for introns that is not useful.

The first test determines the consistent accuracy of inferred languages for exons and introns. The test is repeated 1000 times and data is collected at each repetition. Each iteration of the test randomly samples 200 positive sample sequences from the genome for exons and introns. It then performs the ALERGIA algorithm using an alpha confidence value of 0.5 on each positive set. Then random test sequence sets are sampled from the genome for exons and introns ranging in size from 200 to 2000 samples per set. The ACP is calculated for each test set and recorded.

In order to find a language with high ACP the proper input parameters must be determined. The first parameter to consider is the alpha confidence value for the ALERGIA algorithm as this parameter is the most likely to have an apparent effect on the accuracy of the model. The expected effect of an increased alpha value is an increase in the stringency of the merge constraint of the algorithm. The alpha confidence value controls the number of states in the final automaton and the generalization of the language inferred.

This test will determine the optimal alpha confidence value to use to infer exon and intron languages with high ACP. The test is repeated 300 times and data is collected on each iteration of the test. At each iteration sequence sets of size 200 are randomly sampled for testing the inferred languages. Finite automata are generated using alpha values ranging from 0 to 1 at intervals of 0.1 on randomly sampled sequence sets of size 500. The results are averaged to determine possible correlations between ACP and alpha confidence.

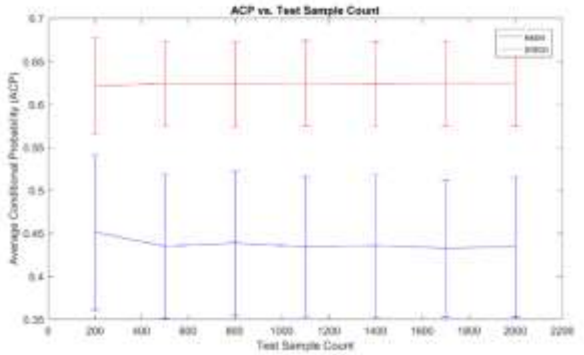

Figure 4: ACP vs. Test Sample Size

VII. Results

The results of the first test show that the average conditional probability is consistently accurate across many different uniformly random test samples of different sizes. There is a small 0.01% difference for exon languages at low sample counts. This could be due to a loss in granularity with fewer test samples causing the result to be overly optimistic. Intron sequences tend to perform better than exon sequences for this test, and the exon languages have worse than random ACP. This is likely due to the alpha value chosen for this test.

This result provides the evidence for the second hypothesis and sets the focus on finding a language that can identify its target components with very high probability. Knowing that the found language will be consistently accurate provides the motivation for the finding of an accurate language, because once the language is found it can be applied to the solving of other problems in computational biology with consistent accuracy.

The alpha confidence test results reveal that the ACP of exon languages scales with the alpha confidence value while the ACP of intron languages do not. Lower alpha confidence will result in over-generalized languages with high percentages of false positives. This is not captured by the intron languages as over-generalizing the language results in a GT[ACGT]* language that has relatively high ACP despite not being very useful.

The next parameter tested was the number of input samples used to build the PTA. ALERGIA was run using sequence sample sizes ranging from 100 to 1100 in increments of 100 using an alpha value of 0.8. This test was repeated 20 times to produce a recordable average; however, the accuracy fluctuated between different sample sizes without indication of direct relationship.

Through exhaustive search using ALERGIA and the optimal parameters determined by the previous tests, automata were found having up to 65% ACP for exon language. This is slightly better than random and therefore provides evidence for the first hypothesis; however, this is not accurate enough for useful application of the inferred languages.

The next tests determine the accuracy and performance of the SearchAlergia algorithm proposed in this paper.

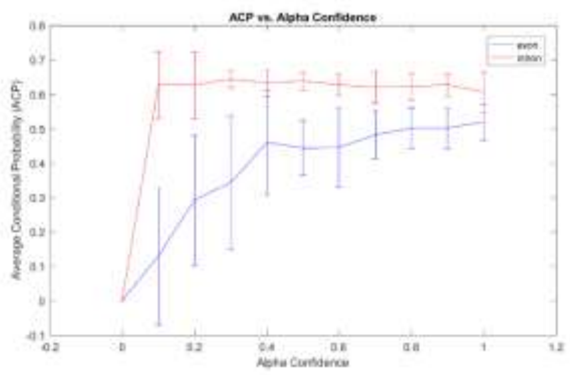

Figure 6: ACP vs. Alpha Confidence

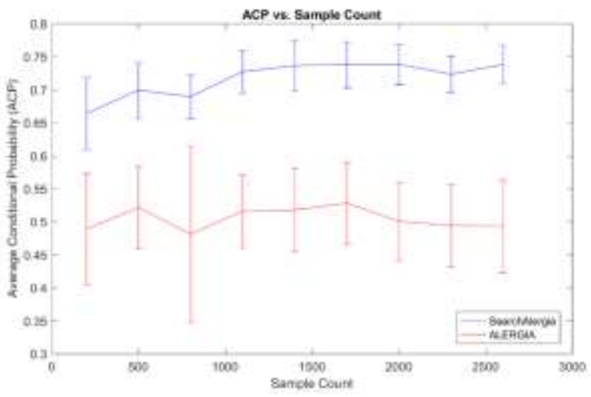

Figure 5: ALERGIA vs. SearchAlergia - ACP vs. Sample Count

SearchAlergia and ALERGIA were timed on input sequence sets ranging from 200 to 2600 sequences in size. The accuracy of the output automata was determined for each algorithm. SearchAlergia was more accurate than ALERGIA for all sample sizes and the accuracy of SearchAlergia improved as the sample size increased. The accuracy of ALERGIA decreased as the sample size increased and the inferred automata positively identified more negative samples than positive ones for larger sample sizes. The run time of SearchAlergia was longer than ALERGIA for smaller sample sizes less than 1200 samples; however, for sample sizes larger than 1200 SearchAlergia outperformed ALERGIA. The difference in run time increased as the number of samples increased indicating an improvement in complexity from ALERGIA to SearchAlergia.

Using the SearchAlergia algorithm, automata were discovered for the exon language having up to 84% ACP. This provides evidence for the first hypothesis and has accuracy effective for applications in computational biology.

VIII. Application

One aspect for determining the accuracy of gene prediction software is to count the number of predicted exons that do not overlap any true exon in the gene. This is metric is called wrong exons. GENSCAN reports its wrong exon accuracy as far exceeding its competitors at 5%. Most gene prediction software predicts 10-15% wrong exons. A post-processing method for these software using the methods proposed in this paper could take the predicted exons and run them through automata created for the exon language. If the positive match percentage of the automata is high and it rejects an exon, then the exon can be removed from the set of positive matches.

Exons are located using a simple 1st order HMM which returned mostly wrong exons. While most gene finding software that rely on HMMs use 4th or 5th order HMMs with complex state transitions to improve accuracy, the scope of this example is limited to distinguishing exons and introns in intragenic regions so a simpler model is used. Through application of the method proposed in this paper an automaton was created to recognize exons versus introns with approximately 65% ACP. The HMM generated predictions for

200 genes and was tested for accuracy using standard measures [17]. The predictions were fed into the inferred automata which filtered out exons that did not match its language definition. The filtered output was then tested on the same metrics as the unfiltered HMM output. The mean result of filtering was about 20% improvement to the exon specificity (Sp) and a 30% reduction in the number of wrong exons (see Figure 7). These improvements were made without altering the HMM method at all and simply applying the proposed method as post processing using an automaton that has only 65% ACP.

Using the SearchAlergia algorithm an 80% ACP language definition is inferred for the exon language. Using the same method as before an HMM made exon predictions for 200 genes. Wrong exons were then filtered using the 80% ACP automaton. The mean result of filtering was a 100% improvement in exon specificity (Sp) and an 80% reduction in the number of wrong exons detected (see Figure 8). The HMM for this example produced a very large number of wrong exons compared to the average case; however, it highlights the positive impact of grammatical inference filtering on Hidden Markov predictions.

In both of these examples some true positive exons were removed as a result of filtering. Improving the ACP of the language used for filtering should reduce the number of true positives that are removed by filtering and increase the number of wrong exons removed. Standard metrics express missing and wrong exons as a percent of predicted exons where here they are instead expressed as average number of missing and wrong exons.

IX. Conclusions

There are languages that can be inferred to differentiate coding and non-coding regions of DNA. These languages describe the patterns of the genetic constructs from which they are inferred. A language that is inferred for exons or introns can be tested against other annotated exons or introns by determining the percentage of sequences in that class that are accepted by the language. A language that is tested on a randomly sampled subset of the genome will be consistently accurate for all randomly sampled subsets. This verifies the theory that there is a structured language common to coding DNA and a similar structured language common to non-

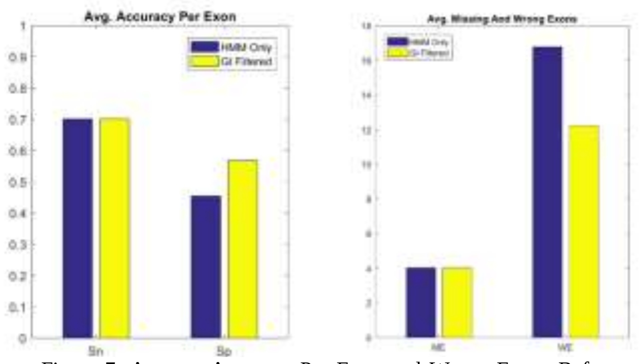

Figure 7: Average Accuracy Per Exon and Wrong Exons Before and After Filtering with 40% Distinct Automata

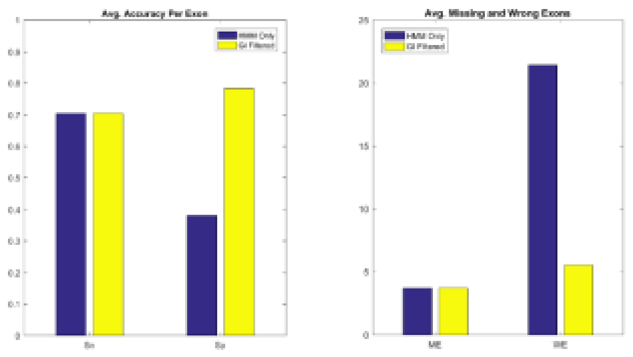

Figure 8: Average Accuracy Per Exon and Wrong Exons Before and After Filtering with 80% ACP Automaton

coding DNA. This also opens up possibilities for application of inferred automata in bioinformatics and the ability to search for an efficient automaton.

Using grammatical inference, automata can be inferred that approximate the language of components of DNA capable of differentiating coding and non-coding nucleotide sequences. This indicates that there is a difference in the patterns and language between the coding and non-coding sequences.

Test results indicate that the accuracy for automata inferred for exon language scaled with the alpha confidence level. Conversely, test results also indicate that accuracy for automata inferred for intron language did not scale with the alpha confidence value. This appears to indicate that exon language has more intricacies than that of the intron language since under-fitting the exon language results in a language definition that also matches introns; whereas, under-fitting the intron language does not necessarily result in a language definition that matches for exons.

The test results for ALERGIA only became apparent after averaging results over many test iterations and the accuracy of the majority of inferred languages was such that they were unusable. Also, the accuracy of ALERGIA does not scale with the number of sequences provided to the algorithm. As such, determining the optimal parameters and inputs for ALERGIA is an arduous task. The SearchAlergia algorithm proposed in this paper helps to alleviate some of the issues with the ALERGIA algorithm. It uses a divide and conquer approach to efficiently scan over many possible subsets of the input samples and combines and promotes the subsets that provide

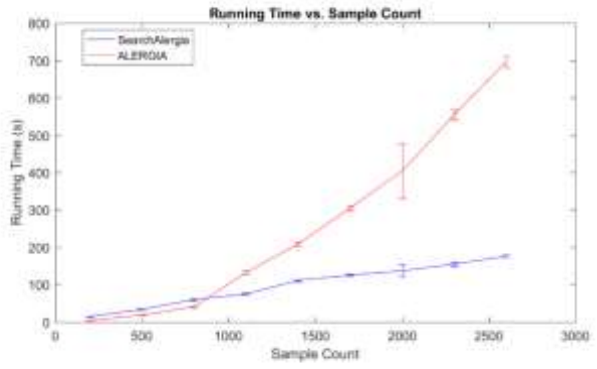

Figure 9: ALERGIA vs. SearchAlergia - Running Time vs. Sample Count

the highest accuracy. It does not guarantee that the optimal subset will be found; however, testing shows that the accuracy and running time of the algorithm improve over ALERGIA alone.

X. Future Work

More information should be considered than if a string is accepted by the automaton or not. Intron language can be defined as GT[ACGT]*AG and this language definition will be 90% accurate; however, this is not a particularly useful observation. This language does not tell us anything about the internal structure of an intron and only provides us with possible start and ending sequence information. We could not use this language to detect exons within intron sequences since anything could be in an intron as long as the intron starts with GT and ends with AG.

Since positive and negative samples are available for coding and non-coding DNA, ALERGIA may not be the best algorithm to use for inferring the languages of these genetic components. Algorithms that utilize negative samples, such as RPNI, should be of particular interest.

The work in this paper was tested against the human genome exclusively; however, according to the work done by Seneff, Wang, and Burge, languages inferred for human DNA may have application to other species [20]. It should be tested to see if the accuracy of a learned language is maintained across different species or if each species carries its own particular internal structure.

References

[1] J. D. Watson and F. H. Crick, "A Structure for Deoxyribose Nucleic Acid," *Nature,* pp. 737-738, 1953.

[2] E. Chargaff, "Preface to a Grammar of Biology: A hundred years of nucleic acid research," *Science,* pp. 637-642, 1971.

[3] M. E. B. Yamagishi and R. H. Herai, "Chargaff's "Grammar of Biology": New Fractal-like Rules," *Genomics*, 2011.

[4] K. J. Locey and E. P. White, "Simple Structural Differences between Coding and Noncoding DNA," *PLoS ONE,* 2011.

[5] J. Maynard-Smith, Evolutionary Genetics, Oxford: Oxford University Press, 1998.

[6] J. F. a. L. D. H. 8. (. R. P. W. 8. M. 2. Poyatos, "The Determinants of Gene Order Conservation in Yeasts," *Genome Biology,* p. R233, 2007.

[7] M. a. L. C. 3. (. 5. P. W. 8. M. 2. Chorev, "The Function of Introns," *Frontiers in Genetics,* p. 55, 2012.

[8] M. Stanke, "Gene Prediction with a Hidden Markov Model," University of Göttingen, Göttingen, 2003.

[9] C. Burge and S. Karlin, "Prediction of Complete Gene Structures in Human Genomic DNA," *Journal of Molecular Biology,* pp. 78-94, 1997.

[10] E. M. Gold, "Language Identification in the Limit," *Information and Control,* pp. 447-474, 1967.

[11] D. Angluin, "Identifying Languages From Stochastic Examples," Yale University, New Haven, 1988.

[12] J. E. Hopcroft, R. Motwani and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Boston: Addison-Wesley, 2001.

[13] J. Oncina and P. Garcia, "Inferring Regular Languages in Polynomial Update Time," *Pattern Recognition and Image Analysis,* 1992.

[14] R. C. Carrasco and J. Oncina, "Learning Stochastic Regular Grammars by Means of a State Merging Method," in *ICGI '94 Proceedings of the Second International Colloquium on Grammatical Inference and Applications* , London, UK, 1994.

[15] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables," *Journal of the American Statistical Association,* pp. 13-30, 1963.

[16] M. R. Anderberg, Cluster Analysis for Applications, New York: Academic Press, 1973.

[17] M. Burset and R. Guigo, "Evalutation of Gene Structure Prediction Programs," *Genomics,* pp. 353-367, 1996.

[18] K. D, H. AS, F. TS, R. KM, S. CW, H. D and K. WJ, "The UCSC Table Browser data retrieval tools," *Nucleic Acids Research,* 2004.

[19] H. Akram, C. d. l. H. Ibne, H. Xiao and C. Eckert, "Grammatical Inference Algorithms in MATLAB," in *ICGI 2010: Proceedings of the 10th International Colloquium on Grammatical Inference*, Valencia, Spain, 2010.

[20] S. Seneff, C. Wang and C. B. Burge, "Gene Structure Prediction Using an Othologous Gene of Known Exon-Intron Structure," *Applied Bioinformatics,* pp. 81-90, 2004.

[21] E. Wallace, "Finite State Machine Designer," 7 March 2015. [Online]. [Accessed 8 May 2016].

Appendix A: Improving gitoolbox

The grammatical inference toolbox provided an excellent starting point for the work presented in this paper; however, there were several issues with the software that needed to be fixed.

The AlergiaCompatible function was not implemented according to the definition of the algorithm given by Carrasco and Oncina. The implementation was not recursively checking the subtrees for compatibility and was only checking the node acceptance and emission frequencies. Produced automata were overly optimistic and represented much larger languages than they should have. That is, the languages that should have been produced by true ALERGIA were subsets of the languages produced by this implementation. This skewed the results and produced some interesting effects. For instance, exon languages generated by this implementation would consistently match over 99% of all introns and exons indicating that the language generated was very close to $\sum^*$. Whereas, intron languages generated with this implementation would consistently match over 95% introns and only about 12% exons with the highest accuracy produced languages reaching close to 91%. It is very interesting that an overly optimistic approximation of the language could produce an extremely viable result; however, there is no formal language theory capable of explaining the phenomena so the results had to be thrown out. More research should be done on the recursive compatibility matching of ALERGIA and determine the probabilistic effect of ignoring subtree compatibility. Fixing the implementation produced the results presented in the paper.

The Build_FPTA function is implemented properly in the gitoolbox; however, the algorithm used is inefficient and suffers from poor performance. Instead of using insertion sort to place the sequences read from the file in order, read all of the sequences in the file then use MATLAB's sort method to sort the input set.

In the primary block of the Build_FPTA function the poor performance implementation iterates over all sequences in the stochastic set then iterates over every symbol in each sequence. At each symbol it checks against all of the previously discovered prefixes for the prefix up to the symbol in the current sequence. If a prefix does not exist, then it adds a new found prefix and initializes a new state in the output deterministic frequency finite automata (DFFA). It then increments the visited and accepted frequencies for the prefix at the symbol in the sequence. The complexity of the algorithm is in O (N^2K^2) for N samples and K sample string length with slightly better amortized complexity.

The improved Build_FPTA function implementation iterates over all sequences in the stochastic set. For each sequence use binary search to find the longest existing prefix in the set of previously discovered prefixes. Then initialize memory for all new states where the number of new states is the length of the current sequence minus the longest prefix. Then for each symbol up to the longest prefix increment the visited frequencies. Then for each symbol beyond the longest prefix add a new state to the resulting DFFA. The complexity of the improved implementation is in O ($2N^2 \log K + NK$) with slightly better amortized complexity. There are also some environment performance enhancements in this implementation as memory is allocated in blocks rather than incrementally.

The Build_FPTA function is further improved by not finding the longest prefix beforehand. The implementation iterates through the states indicated by the sequence updating the frequencies until it reaches a state that does not exist. It then iterates through all of the remaining symbols in the sequence adding a new state for each symbol. This implementation does not require any comparisons to the previously discovered prefixes and is in O (NK).