2009

DNA Sequence Representation by Use of Statistical Finite Automata

Asmi Shah
San Jose State University

DNA Sequence Representation by Use of Statistical Finite Automata

A Project

Presented to

Prof. Dr. T.Y. Lin

In Partial Fulfillment

of the Requirements for the Degree

Masters of Science

By

Asmi Shah

December 2009

SAN JOSÉ STATE UNIVERSITY

The Undersigned Project Committee Approves the Project Titled

DNA SEQUENCE REPRESENTATION BY USE OF STATISTICAL FINITE AUTOMATA

By

Asmi Shah

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Prof. T.Y Lin, Department of Computer Science	Date
Prof. Robert Chun, Department of Computer Science	Date
Prof. Chris Pollett, Department of Computer Science	Date

APPROVED FOR THE UNIVERSITY

Associate Dean	Date

ABSTRACT

DNA SEQUENCE REPRESENTATION BY USE OF STATISTICAL FINITE AUTOMATA THEORY

By Asmi Shah

This project defines and intends to solve the problem of representing information carried by DNA sequences in terms of amino acids, through application of the theory of finite automata. Sequences can be compared against each other to find existing patterns, if any, which may include important genetic information. Comparison can state whether the DNA sequences belong to the same, related or entirely different species in the 'Tree of Life' (phylogeny). This is achieved by using extended and statistical finite automata. In order to solve this problem, the concepts of automata and their extension, i.e. Alergia algorithm have been used. In this specific case, we have used the chemical property - polarity of amino acids to analyze the DNA sequences.

ACKNOWLEDGEMENT

I would like to thank my advisor Dr. Tsau Young Lin for his guidance throughout the duration of my project. He has been extremely supportive and without his mentorship, this project would not have been possible. The enthusiasm and interest shown by him in my project helped me get through the challenging task with relative ease.

I am thankful to the committee members for their help, and insightful discussions on the progress of the project in general or certain minor details.

I would like to express my sincere thanks to the Department of Computer Science at San Jose State University to provide me an opportunity to explore and publish my views and ideas.

Finally, I am grateful to my parents and family members who have always been there for me to cherish my successes and support my failures. Without them, I would not have achieved my career milestones. My friends have played an equally important role towards my achievements so far. I am especially thankful to Sujay, Chintan, Varun, and Maulik with whom I was able to discuss the technical or logical aspects of my project.

Table of Contents

List of Figures

List of Tables

1. Overview

This chapter gives a small introduction to the problem. The outline of the report that follows is briefly summarized here. Chapter 2 provides the background on DNA sequences and protein synthesis. Chapter 3 justifies why automata theory works for representing DNA sequences as data. Chapter 4 provides the background for automata theory and describes how the Alergia algorithm is applied to DNA sequences to extract information. In Chapter 5, the implementation details of the program are discussed. In Chapter 6, various experimental data and results obtained using the program on these data sets are described. The report ends with suggestions for future work and enhancements that could be done to the program.

1.1 Introduction

DNA (DeoxyriboNucleic Acid) is the nucleic acid which contains the necessary genetic information used in the development and functioning of any prokaryotic or eukaryotic organism. Molecules like mRNA (RiboNucleic Acid) and proteins are synthesized using the information carried by DNA. DNA is a double helical structure of two strands of nucleotides connected via hydrogen bonds. There are four nucleotide bases in any DNA sequence: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). In RNA, Thymine is replaced by Uracil (U). The DNA template (one of the two strands) is used to read the genetic code and pass on the information for protein synthesis. Thus, DNA can be

thought of as a mere sequence of four letters A, C, G, and T. In case of RNA, these letters are A, C, G, and U.

Automata theory is an intelligent approach to represent any regular expression. An Automaton learns the pattern of the data fed to it. This approach has been used in the real world to solve several scientific problems successfully as described below. It allows a certain tolerance of error in the results, and hence can provide approximations as well.

1.2 Prior Work

So far, Automata have been used to study intrusion detection in the system calls made in any computer program [5], to identify the author with respect to their writing pattern and to compare different books [6], and also for numerical data sequences. Moreover, cellular automata have also been used to generate image representation of DNA sequences [20]. Thus, the application of automata theory to study DNA sequences in terms of the regular expressions of amino acids could prove to be a powerful tool to understand genetic patterns phylogenetically.

Many machine learning methods have been used to analyze the information carried by DNA sequences. Majority of the approaches concentrate on the 3D model of DNA [23], and the secondary structure of DNA which involves both the strands in parallel [9]. Apart from this, some different approaches like FPGAs [13], Optical pattern recognition [15], neural networks [18] and H curves [23] [25] are used for representation and pattern

matching in DNA sequences. Representation of protein sequences has also been proposed by use of amino acid subalphabets [22].

Thus, the representation of gene sequence has been the topic of research for several years. The first 3D H curve representation of DNA was discussed in [23]. Attempts to visualize the DNA sequence as images have been made using automata. They achieved noticeable results with the cellular automata on an abstract level, where they were trying to improve the quality of predicting protein attributes through images [20].

The idea of modeling DNA sequences as automata was first proposed in 1984 as discussed in [21]. In those days the biological sequence database was limited and the idea was to encourage the researchers to try to implement automata based models which may be applied to primary or 3D structure of DNA.

Here, we will propose an approach to represent the primary structure of DNA data in terms of automata, and further to compare various DNA sequences to answer the similarity and difference among species as per phylogeny.

2. DNA Sequence Representation with respect to Amino Acids

In order to understand how the data in DNA sequences transform into automata, we need to go through a brief introduction to the biological concepts. We need to understand what DNAs are and how they are responsible for the life and formation of any species.

The basic ingredients of any organism (whether a prokaryote or eukaryote) are proteins. For example, hair, skin and muscles are largely made up of proteins. Other biochemical compounds like fats and carbohydrates are synthesized by enzymes or by other proteins which are synthesized using information from DNA. Cells die and new cells are generated in a timely manner, for which the genetic information of the organism is necessary. An organism has a necessary set of chromosomes by which it differs from other organisms (homo-sapiens (humans) have 23 pairs of chromosomes). These chromosomes carry the basic information of that organism in terms of shape, size, form and other phenotypic characteristics in their respective genes [7]. These genes, which are short sequences of DNA, are located at different loci on the chromosomes.

Proteins are polymers, and the monomeric units are amino acids. In general, there are 20 amino acids in any organism. It is the length and the sequence of the amino acid chain which makes a protein unique. This in turn, depends on the number of nucleotides and its sequence in the DNA. The entire process of protein synthesis is shown in figure 2. During the transcription process, information from DNA is passed on to RNA where Thymine (T) is replaced by Uracil (U) base in presence of RNA polymerase.

The promoters at specific locations on the DNA indicate the presence of next gene in the sequence. Introns are spliced (only exons contain valuable genetic information which are just 5-10% of the gene and rest are the introns, also known as the non-coding part of DNA) and the mRNA is capped by a modified Guanine base and terminated by a repeated sequence of Adenine base A preparing a matured mRNA.

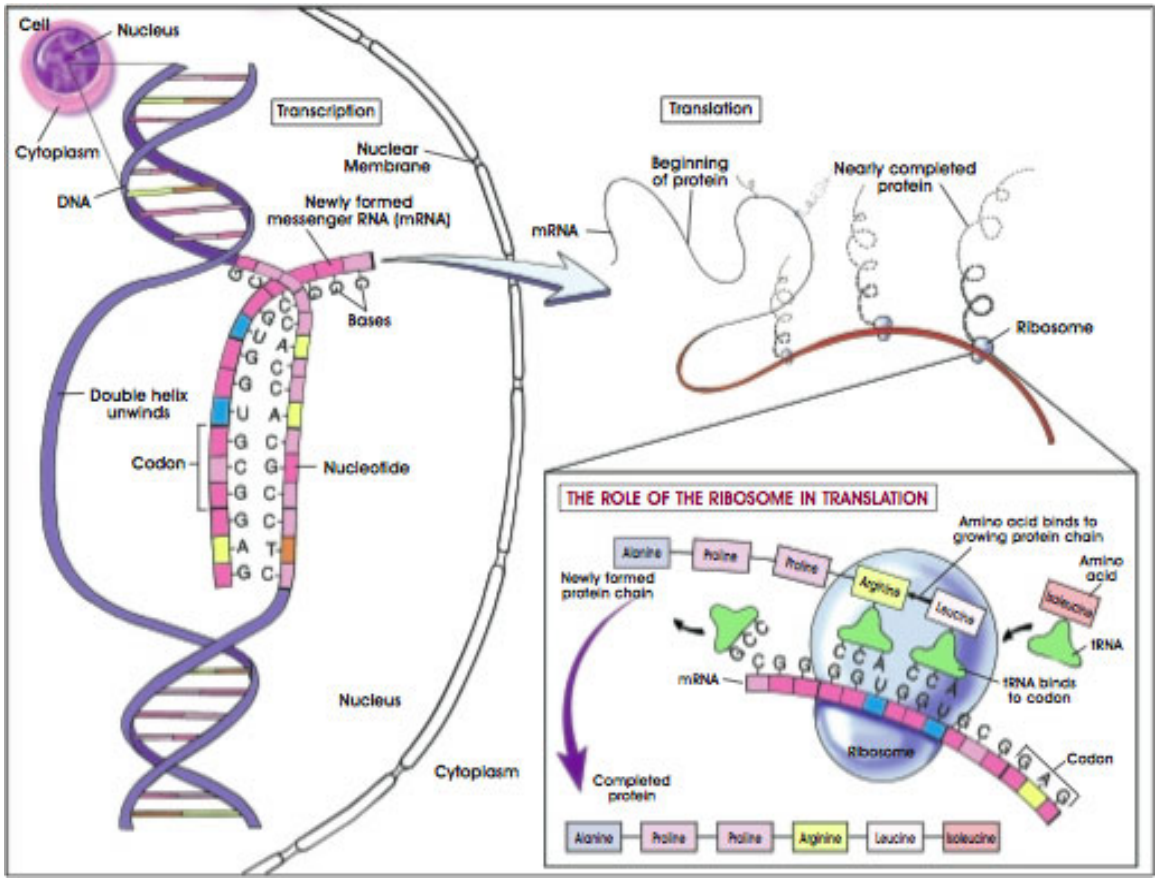


Figure 1: Gene Transcription, Translation, and Protein Synthesis [28]

Transcription takes place in the nucleus where the information from DNA is transferred to mRNA, replacing the Thymine with Uracil. Ribosomes, with help of other cytoplasmic RNAs start reading the mRNA three base pairs at a time, which is called a codon. For

each codon, there is an amino acid mapped in the genetic code as shown in figure 2. The translation of information from mRNA starts with the first hit of 'AUG' codon which results in Methionine amino acid. Then, the translation goes on and the amino acids are linked to each other creating a chain till the time any of the stop codons (UAA, UGA or UAG) are encountered, which indicates an end of the mRNA sequence to be translated. The adjacent amino acids join together with polypeptide bonds and result in a newly synthesized protein. Thus, taking the amino acids into consideration, we can have a ground to analyze DNA sequences.

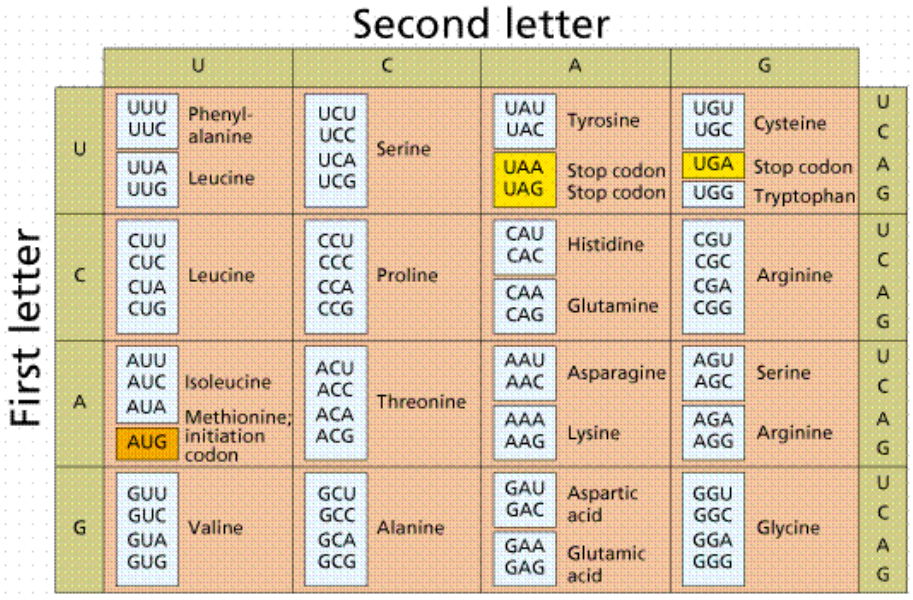


Figure 2: Genetic Code [2]

3. Why Finite Automata for DNA Representation?

This chapter introduces Finite Automata and justifies its use in representing the DNA sequences.

Kolmogorov theory combines information theory, probability theory and randomness. According to Kolmogorov theory, a string, which has patterns, can always be represented and written by some "simple" Turing machine. There is no practical way to express Turing machine. We can use finite automata to approximate this turing machine. Here, we use Alergia algorithm to approximate this simple Turing machine by finite automata.

From a Granular Computing (GrC) point of view, finite automata form granules in the category of Turing machines. So the approximation theory of GrC was one of the factors responsible for the success of this approach in previous text representations [5], and computer security [6]. We apply this approach to DNA representation, which is more complex and deeper. We have to differentiate the species as individuals and have to group them to show their common ancestor or as belonging to the same taxon. Say, humans and mice are distinct, but in the tree of life they are closer than other mammals like marsupials. We will see this through an example explained in detail in a later chapter.

The synthesis of proteins involves a mere repetition of various amino acids in the chain. Each codon is a permutation of any 3 nucleotides out of 4; so this makes a total of 64 codons. They code for 20 amino acids. Now these 20 amino acids can be categorized

further depending upon their chemical property - polarity. Each amino acid has an amino group and a side chain which varies in polarity, depending on its structure. Some are hydrophobic being non-polar and the rest are hydrophilic being polar, which again can be subcategorized in neutral, acidic (negatively charged) and basic (positively charged) polar amino acids depending on their acidic/basic behavior.

It has been proved that during the translation process replacement of one amino acid by another can result in *silent* replacement if they fall in the same category due to their similar chemical and structural properties [7] [8]. This indicates that 'intra' amino acid group replacement results in no major harm and the protein still results in the supposed one with no major dysfunction in its properties.

Hence, we have four main amino acid groups and the mapping of the amino acids with their codons is shown below. We enumerate the groups from 0 to 3 for the amino acid groups and -1 for the stop codons. (Refer genetic code in figure 2.)

0 – NonPolar

Glycine (G) – GGU, GGC, GGA, GGG;

Alanine (A) – GCU, GCC, GCA, GCG;

Valine (V) – GUU, GUC, GUA, GUG;

Leucine (L) – CUU, CUC, CUA, CUG, UUA, UUG;

Isoleucine (I) – AUU, AUC, AUA;

Proline (P) – CCU, CCC, CCA, CCG;

Methionine (M) – AUG;

Phenylalanine (F) – UUU, UUC;

Tryptophan (W) – UGG

1 – Polar Neutral

Serine (S) – UCU, UCC, UCA, UCG;

Threonine (T) – ACU, ACC, ACA, ACG;

Cysteine (C) – UGU, UGC;

Asparagine (N) – GAU, GAC;

Glutamine (Q) – CAA, CAG;

Tyrosine (Y) – UAU, UAC

2 – Polar Acidic

Aspartic Acid (D) – GAU, GAC;

Glutamic Acid (E) – GAA, GAG

3 – Polar Basic

Lysine (K) – AAA, AAG;

Arginine (R) – CGU, CHC, CGA, CGG, AGA, AGG;

Histidine (H) – CAU, CAC

-1 – Stop Codons

UAA,

UAG,

UGA

Thus, we can think of the DNA sequence as having a regular expression of amino acids group being 0, 1, 2, 3, and -1.

The DNAs are sequenced in laboratories and are experimental data. The online databases such as GenBank, NCBI, Swissprot available for the DNA/protein sequences are updated with correct and exchanged sequence data almost every day. Moreover, mutations are possible in the cellular process itself which may result in a silent mutant protein (like due to the intra amino acid replacements), totally deviated (missense), or senseless protein (nonsense) or no protein at all [7].

In order to tackle with the silent point mutation, we need some fault tolerance acceptance probability to represent the DNA sequence and also to align two sequences. Finite automata can not only learn the sequence pattern, but also have the capability of fault tolerance which is integrated using the Alergia Algorithm with acceptable confidence probability.

We deal with the point mutation of substitution here. For example, in a DNA sequence we encountered G G U (which is Glycine falling under the group of nonpolar amino acid – 0) as a codon instead of G C U (which is a different amino acid – Alanine falling under the same group – 0) due to mutation where G got replaced by the base C, then it would

translate into and represent the same amino acid group even though they code for different amino acids.

Moreover, there are many nucleotide sequence patterns in DNA sequences. One can come to know the evolution of different species by their DNA patterns. Many of the disorders or diseases seen in the species happen to have a repetitive pattern of certain nucleotides on specific loci on chromosomes. There are certain repetitions of nucleotides significant in gene location too (e.g., TATA box, gene promoter, is a repetition of 'T' and 'A' as TAATAATATATA.) This kind of patterns can be viewed as Regular Expression. Finite Automaton is one of the best methods to catch the pattern and represent them as regular expressions, and hence the same holds for DNA sequences.

4. Automata Modeling with Alergia Algorithm

Machine learning approach has been used to analyze the secondary structure of DNA sequences [9]. Deterministic finite Automaton can be used to study the sequence where it can easily identify the patterns in the sequence in terms of branches and loops of the substrings. Regular expressions can compress the repetition and represent them in lesser space [11], and these regular expressions can be studied well by use of Stochastic Finite Automata with its state merging techniques described in [13].

Deterministic Finite Automata in its five tuple notation is

$$A = (Q, \Sigma, \delta, q_0, F) \text{ where,}$$

- Q is a finite set of states,
- Σ is the finite state of input symbols, here

$$\Sigma = \{A, C, G, U\}$$

- δ is the transition function which takes a state and an input symbol as arguments and return other state showing the transition, here

$$\delta: Q \times \Sigma \rightarrow Q$$

- q_0 is the intial state,
- F is the set of the accepting states,
- The language L defined by this DFA over Σ is a subset of Σ^*.

A Stochastic deterministic finite automata SFA = (Q, Σ, δ, q_0, F, P) consists of the DFA and P, a probability function $Q \times \Sigma \cup \{\varepsilon\} \rightarrow Q$ such that:

$$\forall q \in Q, \sum_{w \in \Sigma \cup \{\varepsilon\}} \mathrm{P(q, w)} = 1$$

P is a set of probability matrices of $p_{ij}(a)$, which states the probability of state *i* ending in state *j* with symbol $a \in \Sigma$. We let p_{if} be the probability of the string *w* ending in state *i* then the following applies:

$$p_{if} + \sum_{q_j \in Q} \sum_{a \in \Sigma.} p_{ij}(a) = 1$$

The probability of string w generated by Σ is defined by:

$$p(w) = \sum_{\mathrm{q_j \in Q}} \mathrm{p_{ij}(w)\, p_{if}}$$

The language generated here by the SFA, known as stochastic regular language, is given as:

$$L = \{w \in \Sigma^*: p(w) \neq 0\} \qquad (1)$$

Now for two languages to be equivalent, it needs to have the probability distribution to be identical over Σ^*, meaning, not only the strings should be the same, but their probabilities should be equal too.

$$L_1 \equiv L_2 \Leftrightarrow p_1(w) = p_2(w)\,,\ \forall w \in \Sigma^* \qquad (2)$$

We use Alergia algorithm [13] to build the prefix tree acceptor (PTA) from the sequence of DNA which at every node evaluates the probability of the transitions from that node. Then the state merging technique tries to merge the equivalent nodes.

Taking a DNA sequence of any species (taxon), we can have a set of strings where each starts with Methionine (AUG) as it is a start codon for being translated into protein and ends with one of the stop codons (UAA, UGA, and UAG), and we get rid of rest of the nucleotides in the sequence.

We take a DNA sequence to understand the steps followed for the representation here.

The following is the DNA sequence taken from NCBI [27] in fasta format:

```
>gi|116686129|ref|NM_000024.4| Homo sapiens adrenergic, beta-2-,
receptor, surface (ADRB2), mRNA

GCACATAACGGGCAGAACGCACTGCGAAGCGGCTTCTTCAGAGCACGGGCTGGAACTGGCAGGCACCGCG
AGCCCCTAGCACCCGACAAGCTGAGTGTGCAGGACGAGTCCCCACCACACCCACACCACAGCCGCTGAAT
GAGGCTTCCAGGCGTCCGCTCGCGGCCCGCAGAGCCCCGCCGTGGGTCCGCCCGCTGAGGCGCCCCCAGC
CAGTGCGCTCACCTGCCAGACTGCGCGCCATGGGGCAACCCGGGAACGGCAGCGCCTTCTTGCTGGCACC
CAATAGAAGCCATGCGCCGGACCACGACGTCACGCAGCAAAGGGACGAGGTGTGGGTGGTGGGCATGGGC
ATCGTCATGTCTCTCATCGTCCTGGCCATCGTGTTTGGCAATGTGCTGGTCATCACAGCCATTGCCAAGT
TCGAGCGTCTGCAGACGGTCACCAACTACTTCATCACTTCACTGGCCTGTGCTGATCTGGTCATGGGCCT
GGCAGTGGTGCCCTTTGGGGCCGCCCATATTCTTATGAAAATGTGGACTTTTGGCAACTTCTGGTGCGAG
TTTTGGACTTCCATTGATGTGCTGTGCGTCACGGCCAGCATTGAGACCCTGTGCGTGATCGCAGTGGATC
...
...
AGTTCAGTTCCTCTTTGCATGGAATTTGTAAGTTTATGTCTAAAGAGCTTTAGTCCTAGAGGACCTGAGT
CTGCTATATTTTCATGACTTTTCCATGTATCTACCTCACTATTCAAGTATTAGGGGTAATATATTGCTGC
TGGTAATTTGTATCTGAAGGAGATTTTCCTTCCTACACCCTTGGACTTGAGGATTTTGAGTATCTCGGAC
CTTTCAGCTGTGAACATGGACTCTTCCCCCACTCCTCTTATTTGCTCACACGGGGTATTTTAGGCAGGGA
TTTGAGGAGCAGCTTCAGTTGTTTTCCCGAGCAAAGTCTAAAGTTTACAGTAAATAAATTGTTTGACCAT
GCC
```

As explained, we get the strings by separating a set of three bases as one word, starting from AUG to one of the stop codons. Then for each three base word we map it to our

code of numerical amino acid group and get a file of sequences of those mapped numbers. The set of strings is as follows:

String 1: AUG GCG CCA AGA GCG CAG GCU CCA CAC GCC CCA AGG UGA

String 2: AUG AGA GUG UCA AAG CCA CUU UGU ACG UGG UGG UCU GGG AGG AAA UCC UUG CUG GCU UCC UUA UGG AUG CGG GAC GAU UGA

String 3: AUG AGG UCA AGG ACA UAA

String 4: AUG GCU GCA ACA GAU UGG AGA AUA UGU ACC GCA ACU GCC GUA CUA ACA CAC GGA GUA ACU UGU GUU CUU ACC CCC ACA AGA GUA UUA AUU AGA GAA GCA UGC UAU AAG AAA AAA UGA

The numerical amino group sequences corresponding to the above training strings:

String 1: 0 0 0 3 0 1 0 0 3 0 0 3 -1

String 2: 0 3 0 1 3 0 0 1 1 0 0 1 1 3 3 1 0 0 0 1 0 0 0 3 1 1 -1

String 3: 0 3 1 3 1 -1

String 4: 0 0 0 1 1 0 3 0 1 1 0 1 0 0 0 1 3 1 0 1 1 0 0 1 0 1 3 0 0 0 3 2 0 1 1 3 3 3 -1

Now, having a set of these strings, we can create a prefix tree adapter (PTA) from it, which has the codons as the transition and the nodes being states for the finite automata. We start creating states (nodes) where, with the input of the number 0, 1, 2, 3 we have a transition from one state to other state. So we get the following PTA (shown partial for understanding):

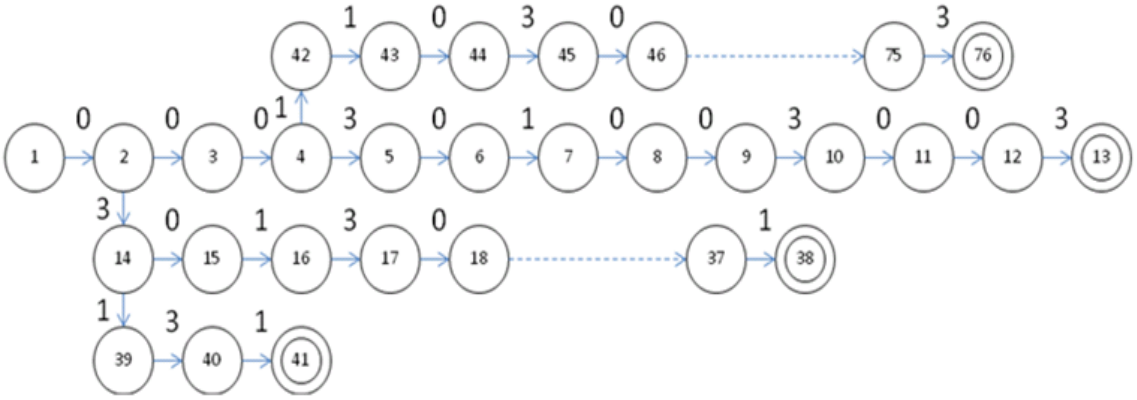


Figure 3: A Prefix Tree Acceptor (PTA) accepting the Training Strings

This automata model represents the PTA which accepts 4 training strings precisely. But it is too rigid as it would not accept any other string with even a little deviation in these strings. As described, in DNA sequences there are chances of errors in experimental data or genetic mutations by nature. Thus, to generalize the automata, we would like our system to identify the pattern from the training DNA sequence strings and "learn" this pattern. This learning should then be helpful to extend the acceptance level. Thus, the automata start reducing on rigidness and accept more strings with a little specified deviation. This is achieved by Alergia algorithm, a state merging method which is shown in Figure 4. Depending on the statistics of the transitions, we merge the nodes if they are equivalent and recalculate the statistics of the frequency of transition and get the final stochastic finite automata.

```
Algorithm Alergia

Input:

        S: sample set of strings

        α: 1 - confidence level

Output:

        SFA

Begin

        A = stochastic prefix tree acceptor from S

        Do (for j = successor(first node(A) to last
        node(A))

             Do (for i = firstnode(A) to j)

                  If compatible(i,j)

                       Merge (A,i,j)

                       Determinize(A)

                       Exit (i loop)

                  End if

             End for

        End for

        Return A

End algorithm
```

Figure 4: Alergia Algorithm [13]

We track the number of strings passing through the node and strings accepted by the node. We denote by n_i the number of strings arriving at node q_i, $f_i(a)$ the number of strings following transition $\delta_i(a)$ (viz., transition by input $a \in A$) and $f_i(\#)$ the number of

strings ending at node q_i. The quotients $f_i(a)/n_i$ and $f_i(\#)/n_i$ give an estimate of the probabilities $p_i(a)$ and p_{if} i.e., probabilities of strings leaving from and strings ending at the node respectively. Now to add the approximation to the acceptance level of the strings by our SFA we need to merge the equivalent nodes. For this we have to compare the statistics of all nodes against each other.

Two nodes are believed to be equivalent if for all symbols a belonging to A, relative transition and termination probabilities are equal and the destination nodes from the transitions of these nodes are equivalent too.

$$q_i \equiv q_j \Rightarrow a \in A, \text{ we have } p_i(a) = p_j(a) \text{ and } \delta_i(a) \equiv \delta_j(a) \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

As we deal with the substitution mutations, this equivalence should be within a range of confidence. The equivalent nodes which adhere to the confidence range are called compatible nodes. The confidence probability for a Bernoulli's variable having probability p with frequency f out of n total number according to Hoeffding's bound is:

$$\left| p - \frac{f}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}} \text{ with probability larger than } (1 - \alpha) \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

This algorithm rejects the equivalence of the nodes if their probabilities differ more than the sum of their confidence ranges as shown in the algorithm in Figure 5. So those nodes are not compatible.

To check the non-compatibility following formula is used.

$$\left|\frac{f}{n} - \frac{f'}{n'}\right| > \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}}\right)$$

```
Algorithm different

Input:
        n, n': number of strings arriving at each node
        f, f': number of strings ending/following a given
        arc
Output:
        Boolean
Begin
        Return   |f/n − f'/n'| > √(1/2 log 2/α) (1/√n + 1/√n')
End algorithm
```

Figure 5: Different Algorithm [13]

If the nodes are not different, the destination nodes are approached and checked to see the compatibility as shown in Figure 6. If they are compatible then we can merge them. In the end, the tree is compressed by merging the states which are compatible and the properties of each node are recalculated and changed like parent node, child nodes, and frequencies of transitions.

```
Algorithm compatible
Input:
        i, j nodes
Output:
        Boolean
Begin
        If different(n_i, f_i(#), n_j, f_j(#))
             Return false
        End if
        Do (∀a ∈ A)
             If different(n_i, f_i(a), n_j, f_j(a))
                  Return false
             End if
             If not compatible(δ(i,a), δ(j,a))
                  Return false
             End if
        End do
        Return true
End algorithm
```

Figure 6: Compatible Algorithm [13]

Thus, we extend our SFA for representing DNA sequences, using the original SFA while preserving its deterministic properties and order.

To summarize, a flow chart for the whole process is shown below:

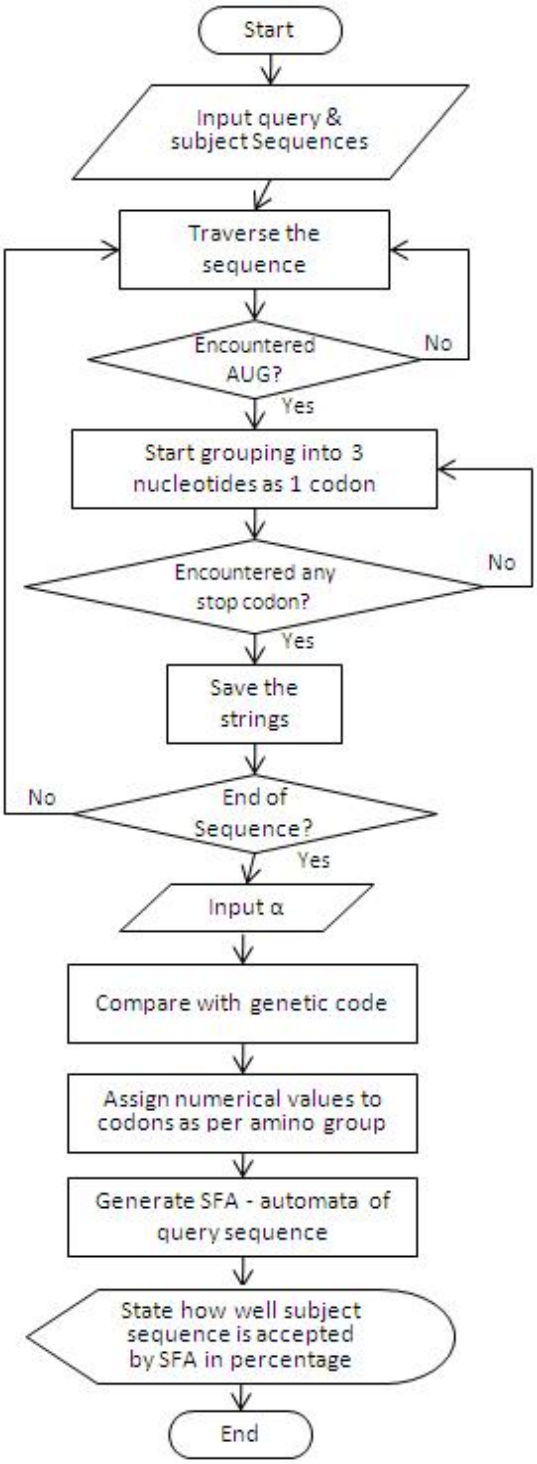


Figure 7: Flow chart of the logical process of program

To understand this with an example, let us take a set of strings as given below to create the automata, as the training set:

S = {λ, λ, 01, 01, 001, 001, 001, 011, 00101, 00101} where λ is an empty string

Now, as described above the Prefix Tree Acceptor would be:

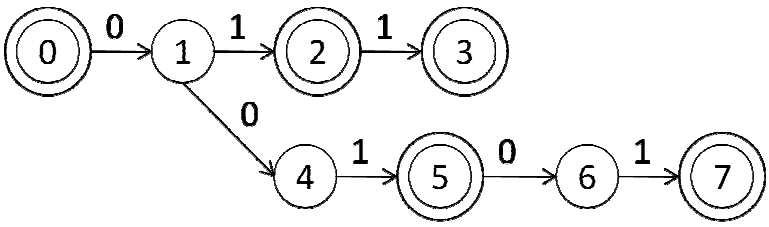


Figure 8: PTA of Set S

The following table shows the frequency statistics for the above PTA, where i is the node, n_i is the number of strings passing through the node i, $f_i(\#)$ is the number of strings which has the node i as their terminating node, $f_i(0)$ is the number of strings having a transition of 0 from node i, and $f_i(1)$ is the number of strings having transition of 1 from node i.

Table 1: Frequency Statistics for PTA of strings in set S

i	0	1	2	3	4	5	6	7
n_i	10	8	3	1	5	5	2	2
$f_i(\#)$	2	0	2	1	0	3	0	2
$f_i(0)$	8	5	0	0	0	2	0	0
$f_i(1)$	0	3	1	0	5	0	2	0

Here, we can clearly see that nodes 3 and 7 and nodes 4 and 6 are compatible as their frequencies are equivalent. Taking α = 0.7 and applying the formula for the difference of frequency fractions, we understand that 2, 3, 5, and 7 are compatible and 4 and 6 are compatible. So we can merge these nodes. The merging of these states is shown below. At last, the frequency statistics are calculated and updated.

Merging state 2 and 3:

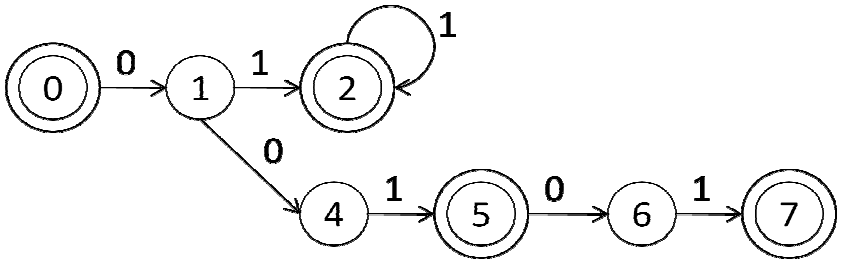


Figure 9: Merging – Step1

Merging states 2 and 5:

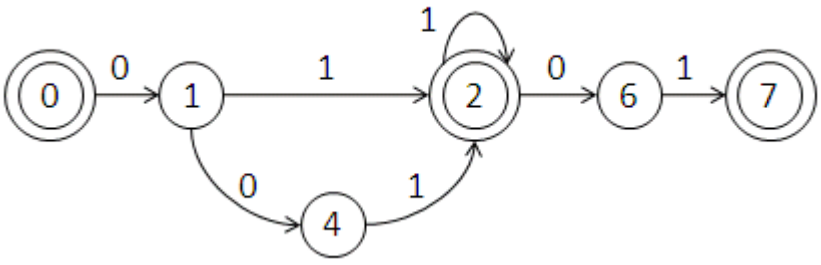


Figure 10: Merging – Step2

Merging states 2 and 7 and states 4 and 6:

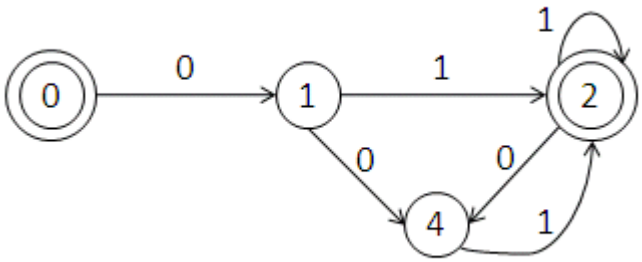


Figure 11: Merging – Step3

As we can see in Figure 11 above, the automaton defines a language by use of strings in a set S, a subset of the set which is represented by the SFA in above figure. Now if we have a set Q = {101, 01111, 0101010, 11011, 001010101} then strings 01111 and 001010101 are accepted by the SFA shown. Thus, the confidence (acceptance) probability will be 0.40 or 40% as each string carries 20% equal probability. We apply the same concept to S as the set of DNA sequence training strings and Q as the set of the DNA sequence testing strings.

5. Implementation

The implementation is done in Java using Eclipse as the IDE for Java. The program mainly consists of two Java files, AminoView.java and Train.java. AminoView.java defines the input output directories, training and test DNA sequence files, and makes functional calls to the functions defined in Train.java.

The state in the automata is an instance to the class Node as shown below:

```
class Node {
    int label;  // node label i
    int parent; // the predecessor: parent node of node i
    int num_thrustr; // number of strings passing through node i
    int num_acpstr; // number of strings terminating at node i
    boolean end;    // is it the end?
    boolean merged; // is node merged with other node?
    int merge_to;   // merged to the other nodes
    int[] child = new int[CodonType];   // child array of the node i
    int[] num_appear = new int[CodonType];  // number of strings passing
                                            // through node i for each codon type
}
```

The function GetCodons() takes the input file of DNA sequences which is used as training data to generate the automaton as an argument, parses through it, and provides the numerical amino group notation for each codon. It then stores an intermediate file as shown below. It also takes care of the mRNA sequences (containing U instead of T) along with DNA as a prior step:

```
int GetCodons(String dir, String in, String out_dir, String out) {
        ……
        ……
         while (fr.read(str) != -1) {

             if("AUG".equalsIgnoreCase(String.valueOf(str)))
             {
                 start = true;
             }

             if(start){
             // nonpolar: 0
                 if ("GCU".equalsIgnoreCase(String.valueOf(str))
                         || "GCC".equalsIgnoreCase(String.valueOf(str))
                         || "GCA".equalsIgnoreCase(String.valueOf(str))
                         || "GCG".equalsIgnoreCase(String.valueOf(str))
                         ……
                         ……
                         || "AUG".equalsIgnoreCase(String.valueOf(str))
                         || "UUU".equalsIgnoreCase(String.valueOf(str))
                         || "UUC".equalsIgnoreCase(String.valueOf(str))
                         || "UGG".equalsIgnoreCase(String.valueOf(str))) {

                     fw.write("0 " + String.valueOf(str) + "\n");
                 }
                 // polar: 1
                 else if ("GGU".equalsIgnoreCase(String.valueOf(str))
                         || "GGC".equalsIgnoreCase(String.valueOf(str))
                         || "GGA".equalsIgnoreCase(String.valueOf(str))
                         ……
                         ……
                         || "UAU".equalsIgnoreCase(String.valueOf(str))
                         || "UAC".equalsIgnoreCase(String.valueOf(str))) {

                     fw.write("1 " + String.valueOf(str) + "\n");

                 } // acidic: 2
                 else if ("GAU".equalsIgnoreCase(String.valueOf(str))
                         || "GAC".equalsIgnoreCase(String.valueOf(str))
                         || "GAA".equalsIgnoreCase(String.valueOf(str))
                         || "GAG".equalsIgnoreCase(String.valueOf(str))) {

                     fw.write("2 " + String.valueOf(str) + "\n");

                 } // basic: 3
                 else if ("AAA".equalsIgnoreCase(String.valueOf(str))
                         || "AAG".equalsIgnoreCase(String.valueOf(str))
                         || "CGU".equalsIgnoreCase(String.valueOf(str))
                         ……
                         ……
```

```
                    || "CAU".equalsIgnoreCase(String.valueOf(str))
                    || "CAC".equalsIgnoreCase(String.valueOf(str))) {
                fw.write("3 " + String.valueOf(str) + "\n");

            } // stop codon: -1
            else if ("UAA".equalsIgnoreCase(String.valueOf(str))
                    || "UAG".equalsIgnoreCase(String.valueOf(str))
                    || "UGA".equalsIgnoreCase(String.valueOf(str))) {
                start = false;
                fw.write("-1 " + String.valueOf(str) + "\n");
            }
        }
        ……
        ……
}
```

Next, the function CreatePTA() generates the prefix tree acceptor ((PTA) from the numeric file created by GetCodons(). Here, it manipulates the class Node, where for each state a new Node is created. For each state, its parameters are updated as per the SFA.

Then, there are Compatible() and Differ() functions which are described by algorithms previously to check if the nodes in question are compatible to each other.

```
boolean Compatible(int node_i, int node_j) {
    if ((node_i > 0) && (node_j > 0)) {
        if (Differ(state[node_i].num_thrustr, state[node_j].num_thrustr,
                state[node_i].num_acpstr, state[node_j].num_acpstr))
            return false;
        for (int j = 0; j < CodonType; j++) {
            if (Differ(state[node_i].num_thrustr, state[node_j].num_thrustr,
                    state[node_i].num_appear[j],
                    state[node_j].num_appear[j]))
                return false;
            if (!Compatible(Delta(node_i, j), Delta(node_j, j)))
                return false;
        }
        return true;
    }

    else
        return true;
}
```

```
boolean Differ(double n_1, double n_2, double f_1, double f_2) {
    return ((f_1 / n_1 - f_2 / n_2) * (f_1 / n_1 - f_2 / n_2) > 0.5
            * Math.log(1 / a) * (1 / Math.sqrt(n_1) + 1 / Math.sqrt(n_2))
            * (1 / Math.sqrt(n_1) + 1 / Math.sqrt(n_2)));
}
```

Delta() is a transition function which transits from one state to its child depending on the input. Recall, δ: Q x $\Sigma \rightarrow$ Q

```
int Delta(int i, int t) {
    return state[i].child[t];
}
```

Once, we know the states that are compatible, they need to be merged. This is done by Combine() and MergeAll() functions. In Combine(), flags are set for the states to merge, where for state *q,* to be merged to *r,* the *merge_to* property is set to *r*. In MergeAll(), the nodes are finally merged and all properties for the nodes are changed accordingly. A part of the code in Combine() is shown below:

```
if (pass) {                         //if compatible
    state[j].merge_to = temp;
    state[j].merged = true;
    if (state[j].end == true) {
        state[temp].end = true;
    }
    j = state[j].parent;
    temp = state[temp].parent;
    m++;
}
```

6. Results

Alergia based Automata modeling can be used to align and compare two DNA sequences against each other. This can even answer the evolution gaps between the DNA sequences that have been compared.

Now, automata have two different notations: a) Transition Diagram, b) Transition table. The code generates the automata in terms of the transition table, which is the tabular representation of the transition function $\delta(q_i, a) = q_j$. It shows the transition of the current state q_i to the destination state q_j with the input alphabet $a \in \{0,1,2,3\}$.

Let us consider the same DNA sequence taken in Chapter 4 to explain the automaton implementation. Shown in Table 2 is the automata generated when $\alpha = (1 - confidence\ level)$ is set between 0.7 and 1. There are no merging states as for the DNA sequence a value of 0.7 to 1 is too rigid to have compatible nodes.

When the other sequence comes as input, it is checked against the SFA that was created by the first DNA sequence, whether it is acceptable or not. If it is acceptable, the system gives the percentage value for the acceptance.

Table 2: Transition Table for the example discussed

q \ a	0	1	2	3
1	2			
2	3			14
3	4			
4		42		5
5	6			
6		7		
7	8			
8	9			
9				10
10	11			
11	12			
12				13
13				
14	15	39		
…				
…				
71		72		
72		73		
73				74
74				75
75				76
76				

The programs tells you if the sequences taken as inputs are the same, or closely related or at the far end of evolutionary tree for different species.

When it comes to comparing the DNA or protein sequences, or to search homologous species or to map evolutionary relations, BLAST (Basic Local Alignment Search Tool) [29] is a defacto standard in search and alignment tools in bioinformatics. It's a heuristic approach where it finds a few short matches between two sequences and then starts to look for more similar alignments locally, which means it does not cover the entire

sequence space. It may not be the optimal solution to a DNA sequence comparison. To match a query and subject sequence, it starts with the matching words of size seven to eleven bases. From there, it starts comparing the nearby sequence areas to find more matches. The limitation is it only aligns locally, but considers gaps and substitutions.

We took a human sequence i.e., **Homo sapiens nebulin (NEB), transcript variant 1, mRNA (locus: NM_001164507)** which has 26202 base pairs to create the automata and the problem was if the mouse parallel sequence i.e., **Mus musculus nebulin (Neb), mRNA (locus NM_010889)** which has 22489 base pairs was accepted by the language generated by the automata.

Following are the search results with the varying values of confidence parameter α:

Table 3: Results for the program with respect to different confidence parameter α values for Human and Mouse Nebulin sequences

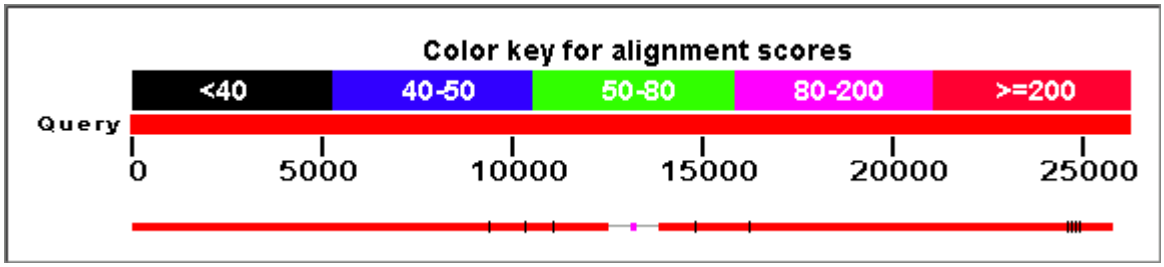
	confidence parameter α (varies between 1 and 0)	**acceptance of the mouse's sequence (percentage)**
Case 1	0.7	15%
Case 2	0.5	25%
Case 3	0.3	76%

When these two sequences are aligned by BLAST it gives 88% identity where the coverage of the sequence considered was 93%, while we consider the whole 100% of the sequence.

The reason we have to take the lower values of α is that the dataset we consider is huge. The string lengths considered are bigger values than generally taken as a unit. Each string stands for a protein, which normally has 400 – 1000s of amino acids. The smallest protein chignolin has 10 amino acids and the largest one being titin has around 27000 amino acids.

Here, if we analyze the results, we can say that when α is 0.7, the automata is too rigid to accept the strings, so by 15% similarity it supports the fact that mouse and human are very different species in evolution but with decrease in α we can see 76% similarity, which can state the fact that human and mouse are similar, as in taxonomically, in the group of mammals (mouse is one of the best model organisms to study human processes.)

Moreover, as discussed, by our grouping methodology we can take care of the substitution mutations, but not the insertion and deletion, as with those two the whole sequence of the amino acid changes and so does the protein, which is again a rare possibility in gene sequences. This scenario is taken care of by BLAST, which does local alignments, and is able to have the gaps in between for a sequence when it has an insertion on one or deletion on the other. The results of the BLAST tool are shown below for the same sequence alignment, where the grey lines in the middle of the red thick line is a part of subject string (Mouse NEB). It does not match with the query sequence (Human NEB) at all which is 7% of the later one, and so, is not counted in the local alignment:



Sequences producing significant alignments:
(Click headers to sort columns)

Accession	Max score	Total score	Query coverage	E value	Max ident	Links
16529	1.128e+04	4.699e+04	93%	0.0	88%	

Figure 12: BLAST results, aligned Nebulin sequences of Human and Mouse

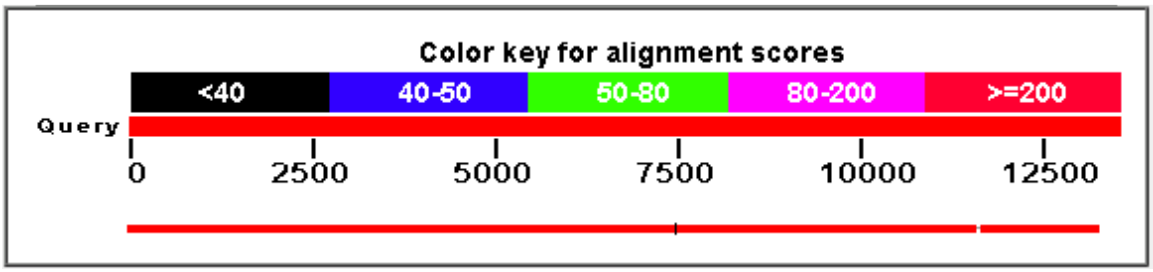
Thus, taking this into consideration, we can say that our method is powerful enough to generate a language by studying one DNA sequence, and also can say whether the other sequence is accepted well with a slight change in selecting the parameter α than the conventional way.

For the second experiment, we took mitochondrion genome sequence of the lungworm found in rats as the query string (**Angiostrongylus costaricensis mitochondrion, complete genome – LOCUS NC_013067)** having 13585 base pairs which generates the language, and mitochondrion genome sequence of the hookworm found in human intestine as the subject string (**Necator americanus complete mitochondrial genome – LOCUS AJ417719)** having 13605 base pairs, and we got the following results:

Table 4: Results for the program with respect to different confidence parameter α values for lungworm and hookworm mitochondrion sequences

	confidence parameter α (varies between 1 and 0)	acceptance of the hookworms's sequence (percentage)
Case 1	0.7	44%
Case 2	0.5	71%
Case 3	0.3	71%

When these two sequences are aligned using the BLAST tool we get 78% identity over 97% of the coverage of the query string, which means that 3% of the sequences don't match at all and so it is not considered while aligning the sequences. This is visible as the gap, white space in the red thick line which is the subject sequence. So there is an identity of 78% of the hookworm with respect to the lungworm mitochondrion sequence, while we get as 71% which is fairly close to the results we get from BLAST, even when we consider the gaps which are shown below with the other results:



Sequences producing significant alignments:

Accession	Max score	Total score	Query coverage	E value	Max ident	Links
AJ417719.2	4985	9677	97%	0.0	78%	G

Figure 13: BLAST results, aligned mitochondrion sequences of Lungworm and Hookworm

When the value is varied from 0 towards 1 for the confidence level, the results get tougher as the automata get rigid with lesser merges and lose the automata handling of repetition and loops with the comparison among sequences.

Even with the large datasets, the runtime of the program remains linear, that is O(n). It may need some virtual space to run on the machine as eclipse is used as the IDE for the Java program. With many applications and tasks running at a time on a machine like IBM thinkpad, Intel core 2 duo, 1 GB of RAM, it still remains linear for its runtime (takes a second or two depending on the size of the sequences in question).

7. Literature Review: Alternative Methods for DNA Representation

The analysis of DNA sequences for finding patterns in the sequence as repetition of nucleotides, to find important characteristic sites such as the TATA box, and comparison the DNA sequences with each other for evolutionary results have been done since decades. This has been one of the most important research areas in bioinformatics. Thus, there are many alternative techniques that researchers have come up with. Each method has its own particular reason of invention and success; we will discuss some of them here.

7.1 3D Technique of DNA Pattern Matching

When the textual DNA sequences (the mere sequence of letters (nucleotides)) are aligned and matched against each other, they fail to answer many of the characteristic information about the species involved in the analysis [16]. This happens because DNA is something more than just the sequence of characters; they are stored in the cells in form of double helical structure, where hydrogen bonds are responsible for their build. So the 3D model of the DNAs is considered and matched to solve the problem. The dimensions and the double helix structure of DNA sequences are compared.

This approach is towards the biological studies for the species behavioral characteristics as included in protein folding, whereas, our method works on the primary structure of DNA.

7.2 DNA Pattern Matching using FPGA

Here, FPGAs (Field Programmable Gate Arrays) are used to analyze DNA sequences and to match patterns between them. This approach uses the hardware directly, and so it happens to be efficient enough and expensive too. “The novel aspect of this approach is the technique of converting a matching problem into a boolean satisfiability problem and then to a circuit, exploiting the reconfigurability of FPGAs [15]”. But, the limitation is of the DNA size, meaning the string capacity that FPGAs support.

In our approach, hardware is not used, the computational time increases in order of the size of DNA which is very nominal as compared to the cost of hardware used here.

Apart from the above described methods, there are solutions such as optical pattern recognition [17], neural network based pattern recognition [18], and linear time filtering approach to analyze the DNA patterns.

7.3 Comparison with Alternative Methods

There are several solutions to DNA pattern analysis. Some solutions lack accurate findings, some fail in providing cost effective solutions, some are inappropriate as they don’t justify the capacity of the input data as large as several DNA sequences, some happen to concentrate more on the data of DNA sequence locally and missing out the bigger picture, while some distribute the concentration over the whole DNA sequence from a limited species.

The better solution may be Automata Theory, where capacity to handle and study the sequences and their pattern is higher and there is a control over the fault tolerance which is user defined. Automaton can easily identify the hidden patterns in a sequence and the comparison is not a mere alignment.

8. Future Work

Many of the protein behavioral patterns depend on the secondary and 3D structure of the DNA sequence which the proposed approach cannot handle [16]. So this can be taken care of in the future implications.

We have used polarity as the property to classify the amino acids. There are even other properties of DNAs like their size and shape [14] which may lead us to a total different classification with different results.

We are not handling the mutations in DNA sequences like insertion and deletion which will result in a frame shift of the coding sequence [8], which means by inserting even a single nucleotide in a sequence somewhere the codons will code for a total different genetic code, which may result in ‘missense’ or ‘nonsense’ mutation coding for different meaningless or no protein respectively [7]. This is still not a case with the gene sequences because in genes, it hardly occurs that they have mutations of insertion or deletion.

9. References

1. Hopcroft, J. E., Motwani R., & Ullman, J.D.: Introduction to Automata Theory, Language, and Computation. Addison Wesley. (2001)

2. Pevsner, J. & Wiley, J.: Bioinformatics and Functional Genomics. (2003)

3. Watson J., Baker T., Bell S., Gann A., Levine M., Losick R.: Molecular Biology of the Gene, Fifth Edition. New York: Benjamin Cummings (2003)

4. Crick F.: The Great Ideas of Today 1980, Encyclopedia Britannica, 644-683 (1980)

5. Baliga P., Lin T.: Kolmogorov Complexity Based Automata Modeling for Intrusion Detection. In: Proceeding of the 2005 IEEE International Conference on Granular Computing, July 25-27, Beijing, China, 387-392 (2005)

6. Lin T., Zhang S.: Automata Based Authorship Identification System. In: New Frontier in Applied Data Mining, Springer Berlin/Heidelberg, 134-142 (2009)

7. Jones S.: The Britannica guide to Genetics, Running Press (2009)

8. Agutter P., Wheatley D.: About Life – Concepts in Modern Biology, Springer (2007)

9. Baldi P., Brunak S.: Bioinformatics: The Machine Learning Approach, MIT Press (2001)

10. Chakraborty, S.: Formal Languages and Automata Theory - Regular Expressions and Finite Automata (2003)

11. Sekar R., Bendre M., Dhurjati D., Bollineni P.: A Fast Automaton-Based for Detecting Anomalous Program Behoviors. In: Proceedings IEEE Symposium on Security and Privacy (2001)

12. Lin T.: Patterns in Numerical Data: Practical Approximations to Kolmogorov Complexity. In: RSFDGrC, 509-513 (1999)

13. Carraso R., Oncina J.: Learning Stochastic Regular Grammars by means of a State Merging Method. In: Proceedings of the 2nd International Colloquium on Grammar Inference. Lecture notes in Artificial Intelligence, 139-152 (1994)

14. Bosnacki D., Eikelder H.M.M., Steijaert M., Vink E.: Stochastic Analysis of Amino Acid Substitution in Protein Synthesis. In: CMSB 2008, LNBI 5307, 367–386, Springer-Verlag Berlin Heidelberg (2008)

15. Lipson, A. & Scott, H.: DNA Pattern Matching using FPGAs [Electronic Version] (2002)

16. Herrison, J., Payen, G., & Gherbi, R.: A 3D pattern matching algorithm for DNA sequences (2007)

17. Martin, J. C. & Hawk, J. F.: DNA sequence analysis by optical pattern recognition. In: The International Society for Optical Engineering, 938, 238-45 (1988)

18. Cotter, N., Gesteland, R., & Murdock, M.: Neural network based pattern recognition for sequenced DNA autoradiograms. In: International Joint Conference on Neural Networks, 2, 909 (1991)

19. Quang V.: A natural number based linear time filtering approach to finding all occurrences of a DNA pattern. In: Fourth International Conference on Intelligent Sensing and Information Processing, 108-111 (2007)

20. Xiao X., Shao S., Ding Y., Huang Z., Chen X., Chou K.: Using cellular automata to generate image representation for biological sequences. In: Amino Acids, 29-35 (2005)

21. Burks C., Farmer D.: Towards Modeling DNA Sequences as Automata. In: Physica D: Nonlinear Phenomena, Volume 10, 157-167 (1984)

22. Anderson C., Brunak S.: Representation of Protein Sequence Information by Amino Acid Subalphabets. In: American Association for Artificial Intelligence, Volume 1, 97-104 (2004)

23. Hamori E., Ruskin J.: H Curves, A Novel Method of Representation of Nucleotide Series Especially Suited for Long DNA Sequences. In: The Journal of Biological Chemistry, Volume 258, 1318-1327 (1983)

24. Yin C., Yau S.: Numerical Representation of DNA Sequences based on Genetic Code Context and its Applications in Periodicity Analysis of Genomes. In: CIBCB – IEEE symposium, 223-227 (2008)

25. Lantin M., Carpendale M.: Supporting Detail-in-Context for the DNA Representation, H-Curves. In: Proceedings of the Conference on Visualization, 443-448 (1998)

26. Ichinose N., Yada T., Takadi T.: Quadtree Representation of DNA Sequences. In: Genome Informatics 12, 510-511 (2001)

27. National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov

28. Stem Cell Information, N.I.H., http://stemcells.nih.gov

29. BLAST, Basic Local Alignment Search Tool, http://blast.ncbi.nlm.nih.gov/Blast.cgi

30. GPCRDB: Information System for G Protein Coupled Receptors, http://www.gpcr.org/7tm/