University of Texas at El Paso
DigitalCommons@UTEP

Open Access Theses & Dissertations

2011-01-01

Computational Methods of Hidden Markov Models With Respect To CpG Island Prediction in DNA Sequences

Roberto Angel Ortega
University of Texas at El Paso, raortega3@miners.utep.edu

Follow this and additional works at: https://digitalcommons.utep.edu/open_etd

 Part of the Bioinformatics Commons, and the Statistics and Probability Commons

Recommended Citation
Ortega, Roberto Angel, "Computational Methods of Hidden Markov Models With Respect To CpG Island Prediction in DNA Sequences" (2011). *Open Access Theses & Dissertations*. 2354.
https://digitalcommons.utep.edu/open_etd/2354

This is brought to you for free and open access by DigitalCommons@UTEP. It has been accepted for inclusion in Open Access Theses & Dissertations by an authorized administrator of DigitalCommons@UTEP. For more information, please contact lweber@utep.edu.

Computational Methods of Hidden Markov Models With Respect To CpG Island Prediction in DNA Sequences

Roberto Angel Ortega Jr.

Department of Mathematical Sciences

APPROVED:

Ming-Ying Leung, Ph.D., Chair

Ori Rosen, Ph.D.

Naijun Sha, Ph.D.

Max Shpak, Ph.D,

Benjamin Flores, Ph.D.

Acting Dean of the Graduate School

To my mother, father, and sister,

- Robert

Computational Methods of Hidden Markov Models With Respect To CpG Island Prediction in DNA Sequences

by

Roberto Angel Ortega Jr.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Mathematical Sciences

THE UNIVERSITY OF TEXAS AT EL PASO

August 2011

Acknowledgements

My gratitude for those who have been a part of my graduate education is great for many reasons. It should be no surprise though, that most of the gratitude sits heavily on the patience and trust of those who have helped to push and support me over the past two years. Coming from a background of little mathematics/statistics training and entering with a class of developed minds, the road was certainly difficult at times. I won't forget the effort and assistance provided to me, and I hope that my future puts to use everything that I have received from those who have extended their hand in help.

I would like to thank Dr. Staniswalis, Dr. Moschopoulos, Dr. Rosen, Dr. Sha, Dr. Wagler, and Dr. Leung of the statistics department at UTEP for giving me this opportunity. The past two years of my life within Bell Hall extends beyond the experience of an education, which is both invaluable and unforgettable.

Specifically, I would like to thank Dr. Staniswalis for offering her guidance while I was an undergraduate, providing time outside of the classroom to listen and help.

My interest in statistics began with Dr. Rosen, who has continuously inspired me throughout my years of education at UTEP. Many of my aspirations for my own personal abilities within the discipline were established because of his.

I would also like to thank my peers, with whom I have worked closely over the past two years. The experiences that we have had will not be forgotten. I wish nothing but the best for all of you in your professional and personal lives from here on out. I hope that our paths will cross in the future.

I want to offer my sincere appreciation to Rahul Simham who, beyond providing his time and effort by collecting the data for this thesis, worked with me for the first weeks in the summer of 2010, to begin to develop an understanding of the HMM algorithms.

Thanks to Dr. Rosen, Dr. Sha, and Dr. Shpak, who served for this thesis' defense committee. Your help and assistance throughout the process is very much appreciated.

Probably the most influential person in my training and eventual interests is my advisor, Dr. Leung. I couldn't have asked for a more appropriate person to be my advisor, and I am endlessly grateful that she agreed to work with me. I was able to witness the rigor that she pours into her work and the level of expectations that she has of herself. A student's academic lineage is a defining aspect of their work and interests, and I hope that I can live up to expectations that I have created for myself, because of Dr. Leung. Beyond my academic work, Dr. Leung was aware of my life outside of school, and took extra strides to offer her support and advice, as well as provide me with financial support. My appreciation for her assistance and guidance is inexpressible.

My family has been a large supporting factor throughout my entire life. I want to thank all of the members of my family, both those alive today and those not with us, who I have taken much from in my life. My parents, sister, brother in-law, nephews, aunt, uncles, cousins, grandparents, and great-grandparents, are my backbone. I am constantly watching and taking notes.

Lastly, I want to thank the love of my life, Katrina, who has served to be everything from my support system, psychiatrist, to a constant reminder of what we are working to accomplish. My world is an infinitely better place for having you in it. I am grateful that you have been a part of it. I love you.

This work was supported in part by NIH grant 5T36GM078000-03.

Abstract

HMM's are a specific case of Markov models where, contrary to Markov chains, the observer is unaware of what state the model was in when the symbol is observed. Like Markov chains, HMM's assume that the future state of a sequence is dependent only on the current state of the sequence. The parameters associated with Markov Models are transition and emission probabilities, where transition probabilities are associated with transitioning from one state to another, and emission probabilities are the probabilities associated with observing a symbol given it came from a specific state.

The structure of DNA sequences is made up of the nucleotides adenine (A), cytosine (C), guanine (G), and thymine (T). CpG islands are regions within the DNA sequence where there is a higher occurrence of the two nucleotides, cytosine and guanine, referred to as the CG dinucleotide.

The HMM algorithms used to analyze the DNA sequences were the Viterbi, Baum-Welch, and Viterbi training algorithms. The Viterbi algorithm determines the state-sequence that is most likely to have produced the given sequence, given the model. The Baum-Welch and Viterbi training algorithms estimate the parameters associated with an HMM.

In specific we have assessed the accuracy of the aforementioned Viterbi algorithm at predicting the location of CpG islands within DNA sequences as well as determine the strength of the parameter estimating algorithms at recovering the model parameters.

Table of Contents

List of Tables

List of Figures

Chapter 1

DNA and CpG Islands

1.1 The Basics of DNA

DNA is a biological construct that is used as the method of long-term storage of information, much like a blueprint for living organisms. DNA itself is a nucleic acid, which are biological molecules essential for life, that contains genetic information used for the development and functionality of a living organism. It serves as a means of transferring this information to future generations of organisms. The many purposes of DNA include carrying genetic information as well as regulating the genetic information.

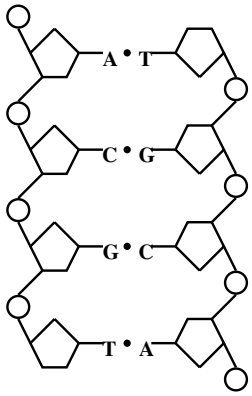

Figure 1.1: Nucleotides of DNA

DNA is much like a coded message which uses a four character alphabet, called nucleotides, which are Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). It should be noted that the nucleotide A bonds with T, C bonds with G, and vice-versa. Together

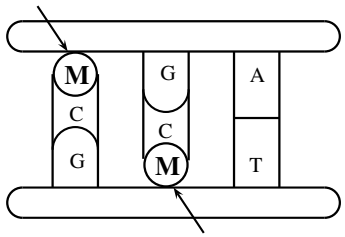

Figure 1.2: Methylation (M) of cytosine nucleotide.

the nucleic acids combine to form interpretable genetic messages that are translated for the purpose of biological function. Similar to the way a specific sequence of letters from a given language convey a message to the reader, so does the order of the nucleotides determine the way an organism is developed and maintained. An important occurrence within DNA sequences are dinucleotides. Dinucleotides are a sequence of two nucleotides, such as AT or CG occurring within a DNA strand. The CG dinucleotide is of interest in this thesis, and the increased occurrence of the CG dinucleotide over a region can lead to what is called a CpG island.

1.2 CpG Islands and Methylation

CpG islands are a phenomena that occur in DNA nucleotide sequences characterized by a high concentration of the dinucleotide CG, occurring repeatedly over a given region in the sequence. Wherever the CG dinucleotide occurs in a DNA strand, the C nucleotide is regularly modified by a process known as methylation and as a result tends to become a T. Methylation is a biologically-programmed modification of DNA that occurs in mammals such that a methyl group (chemically denoted CH_3) is added to a cytosine nucleotide (Tost, 2009) [2].

The largest-noted heritable modification to DNA is methylation (Nephew, Balch, and Skalnik, 2009) [5]. CG dinulceotides occurring less frequently than expected in the genome is the consequence of being an area of DNA where mutations occur at an elevated rate and

thus methylation is necessary for stability in the DNA sequence. As a result, the distribution of CG dinucleotides in a DNA sequence occur less frequently than expected.

Regions that aren't methylated tend to have a higher count of the CG dinucleotide and lead to the development of CpG islands. It has been estimated that mutation rates are increased anywhere from 10 to 50 times within CpG islands (Tost, 2009) [2]. In fact, the increased mutation rate has had an evolutionary impact on DNA sequences, depleting the rate of occurrence of CG dinucleotides over time. The methylation process can be suppressed in areas of the DNA strand though, and as a result such regions have a higher occurrence of the CG dinucleotide. These areas are termed CpG islands (Durbin, Eddy, Krogh, Mitchison, 1998) [7]. A frequently used criterion for determining a CpG island is a region where $\frac{\text{Observed CG count}}{\text{Expected CG count}} > .6$, and the cytosine and guanine content is greater than 50% (Tost, 2009) [2].

The continued study of CpG islands is the result of limited information on CpG Islands, the lack of a formal definition of what characterizes a CpG island, and the difficulty of identifying regions of nucleotide sequences that contain CpG islands. The origin of the study of DNA methylation occurred in 1948, when Rollin Hotchkiss discovered the first modified base, which was during a time period where little was known about DNA itself or its constructs (Weissbach, 1993) [1]. The relevance in the study of CpG islands and DNA methylation is not limited to the advancement of the understanding of the genetic process though. It is also known that an abnormal methylation process is associated with the occurrence of cancer, and recent work has linked it to many other diseases including bipolar disorder and schizophrenia (Brena and Plass, 2009) [10].

This has increased the demand for information on methylation and CpG islands and as a result, numerical methods have been looked at as possible method of analysis. Numerical methods that use Markov properties have been used to assist in the analysis of the phenomena and are used by DNA databases to estimate the location of CpG Islands.

Chapter 2

Markov Chains and Hidden Markov Models

2.1 Background

Markov Models are named after the Russian Mathematician responsible for their discovery, Andreyevich Markov. In 1906, Markov introduced the idea of what is now known as a Markov chain. Hidden Markov Models (HMM's) were introduced by Leonard E. Baum and his colleagues in the late 1960's (Rabiner, 1989) [4]. The original work in HMM's was applied to speech recognition in the 1970's, and has since found applications for multiple disciplines.

2.2 Introduction To Markov Chains

A specific instance of stochastic process, a Markov chain is defined such that regardless of time, the state at any given time is an element of a finite set, the process occurs in discrete time intervals, and the probability distribution of a state at any point in time depends only on a number of states immediately occurring in the past (DeFonzo, Aluffi-Pentini, and Parisi, 2007) [8]. Both Markov chains and hidden Markov models (HMM's) are statistical models that assume the Markov property. A Markov chain $(X_1, X_2, \ldots)$ is a first-order Markov process. In other words, a Markov chain is a random process which consists of a random sequence of state variables, such that the future state in the sequence depends on

the past states only through the current state (first-order Markov process). This is termed the Markov property and can be represented mathematically as:

$$P(X_{j+1} = m|X_1 = m_1, X_2 = m_2, \ldots, X_j = m_j) = P(X_{j+1} = m|X_j = m_j)$$

The collection of the previous terms being conditioned on can be reduced to the single most recent term under the Markov property. Hence, the probability that the $j + 1^{st}$ state assumes some value depends only on the value of the j^{st} state.

In the most general case, the process can have any number of states, and be in any state at any time. Each iteration of the process leads to a movement out of one state and into another, possibly returning back to the same state. The possible values that the $X_i's$ can assume form a set S, called the state space of the chain. This thesis will consider only a discrete state space, although other scenarios may require a continuous state space.

2.2.1 Transition Probability

In a Markov chain, each transition from one state to another has a specific rate of occurrence. The parameter of interest is one that is associated with the individual transitions. Each transition from one state to the next has a probability associated with it and the probability defines the distribution of the Markov chain. Moving from one state to another is referred to as a transition and the probability associated with that move is referred to as a transition probability. A transition probability can be denoted:

$$a_{qk} = P(X_j = k|X_{j-1} = q),$$

where a_{qk} refers to the probability that the state in the sequence is k at time j, given that the state is q at time $j - 1$. The probabilities of individual transitions from one state to another can be organized using a transition matrix. If working with state 1 and state 2, then the transition probability matrix would look like:

$$\begin{array}{cc} \text{State} & \begin{array}{cc} 1 & 2 \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \end{array}$$

where a_{11} is the probability of transitioning from state 1, back into state 1, a_{21} is the probability of transitioning out of state 2, into state 1, etc.

A set of parameters that need to be accounted for deal with entering and exiting the sequence. Generally in Markov chains, the ending of a sequence is arbitrary as it is assumed that a sequence can end at any time (Durbin, Eddy, Krogh, and Mitchison, 1998) [7]. Nonetheless, the ending of a sequence can be represented using a_{k0}. This would represent the occurrence of the sequence being in state k, when exiting the model.

The beginning of a sequence can be denoted using the same notation. The probability a_{0k} can be thought of as the probability of the sequence starting in state k.

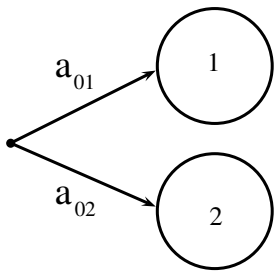

Figure 2.1: In this diagram, the model selects a state to start in. In this case, it can either select starting in state 1 with probability a_{01} or can select state 2 with probability a_{02}.

2.2.2 Applying Notation

Using the definition of a transition probability, the probability of observing a sequence of states can be retrieved. The probability of a sequence $(x_1, x_2, \ldots, x_L)$, where L represents the length of the sequence, can be found using the joint probability identity, $P(A, B) =$

$P(A|B)P(B)$. This now allows for the probability of a sequence to be represented as:

$$P(x) = P(x_L|x_{L-1}, \ldots, x_1)P(x_{L-1}|x_{L-2}, \ldots, x_1) \ldots P(x_1).$$

Again, using the Markov property the probability of a sequence can be reduced to:

$$P(x) = P(x_L|x_{L-1})P(x_{L-1}|x_{L-2}) \ldots P(x_1).$$

Using the newly discussed transition probability, this representation of $P(x)$ is equivalent to,

$$P(x) = P(x_1) \prod_{i=2}^{L} a_{x_i x_{i-1}}.$$

2.3 Introduction To Hidden Markov Models

In a Markov chain, the observer is always aware of which state the sequence is in at any given point of the sequence. In fact, in a Markov chain the observable symbol, O_j, and the state that produced it, X_j, are one in the same. Moving from Markov chains to Hidden Markov Models, it is no longer possible to determine the state that the model was in when the observation was generated. That is to say, the Markov chain is hidden. Colloquially speaking, an HMM is a Markov chain occurring in noise (Cappe, Moulines, and Rydén, 2005) [9]. Mathematically speaking, an HMM no longer holds a 1-1 correspondence between the sequence and the state. Although the state is not directly visible anymore, what is visible is emitted from the unobservable Markov chain.

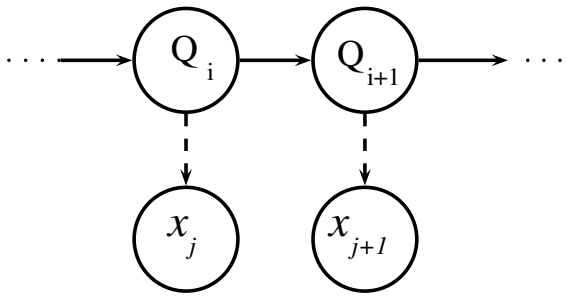

Figure 2.2: As can be seen, the observation x_j depends only on the state the produced it, Q_j. Alternatively, the distribution of x_{j+1} conditional on the past states of the sequence, is dependent only on the state, Q_{j+1}. This is the previously discussed Markov property.

As an example, if the two states of our HMM is CpG and non-CpG, where the CpG state is denoted as "+" and the non-CpG state as "−", the list of possible ways a nucleotide can be represented is given as, A_+, A_-, C_+, C_-, G_+, G_-, T_+, T_-, where A_+ is the instance that the nucleotide A is emitted from a CpG state, and G_- is the instance that the nucleotide G is emitted from a non-CpG state.

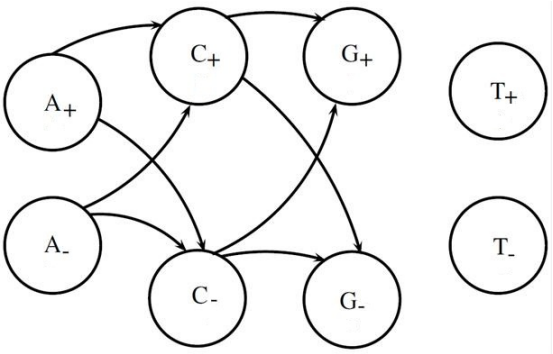

Figure 2.3: Illustration of the number of ways to arrive at the nucleotide sequence, "ACGT".

HMM's extend into many disciplines. In the biological setting specifically, HMM's can be used to look at protein, DNA, or RNA sequences, which often have some elements of information that needs to be recovered (secondary structures, CpG Islands, etc.).

2.3.1 Definitions and Notations

The methods used to analyze sequences need to accommodate for the possibility of arriving at a given observation from a list of states. Because an HMM allows for multiple states to produce the same symbol, the previously discussed transition probability needs to be redefined and a new parameter introduced. The sequence of states that emit the observations can be denoted Q. Therefore the ith state in the path is denoted Q_i. Now transition probabilities can be defined as:

$$a_{kl} = P(Q_i = l | Q_{i-1} = k).$$

Because it can no longer be determined which state the model was in when the symbol was generated, a new parameter should be introduced that signifies the probability that an observation was emitted from a given state. An emission probability is defined as:

$$e_k(b) = P(x_i = b | Q_i = k),$$

which represents the probability that symbol b is seen when in state k. The vector of emission probabilities for state k can be represented:

$$e_k(\bullet) = \begin{pmatrix} e_k(1) \\ e_k(2) \\ \vdots \\ e_k(n) \end{pmatrix}$$

, where n refers to the number of possible observations that can be emitted from state k.

2.3.2 Using The New Notation

Using the transition and emission probabilities, the joint probability between an observed sequence x and a state sequence Q can be represented by $P(x, Q) = a_{0Q_1} \prod_{i=1}^{L} e_Q(X_i) a_{Q_{i,i+1}}$. If we have interest in the probability of observing a specific sequence x, given the specific state sequence Q and the model λ, then we can calculate $P(x|Q, \lambda)$,

$$P(x|Q, \lambda) = \prod_{t=1}^{L} P(x_t|Q_t, \lambda)$$

which because of independence can be expressed:

$$P(x|Q, \lambda) = e_{q_1}(x_1) \times e_{q_2}(x_2) \times \ldots \times e_{q_L}(x_L).$$

The probability of observing the specific state sequence is given by:

$$P(Q|\lambda) = a_{0q_1} a_{q_1 q_2} a_{q_2 q_3} ... a_{q_{L-1} q_L}.$$

The total probability of the sequence can be calculated by summing over all possible state sequences:

$$P(O|\lambda) = \sum_{Q} P(O|\lambda) P(Q|\lambda) \tag{2.1}$$

$$= \sum_{q_1,\ldots,q_L} a_{0q_1} e_{q_1}(x_1) a_{q_1 q_2} e_{q_2}(x_2) ... a_{q_{L-1} q_L} e_{q_L}(x_L). \tag{2.2}$$

To this point, a background and introduction to Markov Models has been laid out.

The parameters of an HMM, the transition and emission probabilities, help to uncover the probabilities associated with a given sequence. It may be the case though that the parameters are not known and must be estimated. Also, calculating the total probability

of a sequence consisting of thousands of observations might be impractical if done using standard methods of calculation. It now becomes important to identify methods able to take on these specific problems, as well as any others that might exist.

Chapter 3

Computational Methods of Hidden Markov Models

3.1 Introduction

A main concern of any task or problem at hand is not only whether or not it can be solved, but also the amount of effort that must be given in order to do so. This is the central concern of HMM algorithms and such methods have been addressed by Lawrence Rabiner in his tutorial on HMM's, written in 1988. As previously mentioned, the application of HMM's to applied problems have been around since the 1960's and 1970's, when Baum and his peers applied HMM's to speech processing (Baum, Petrie, Soules, and Weiss, 1970) [12]. Although the idea of using HMM's to explore real world problems has been around since Baum and his colleagues, it was not until the 1980's when HMM's gained attention from researchers. The main reason that HMM's are reasonably new to research is that the theory behind applying HMM's wasn't readily available in literature, making it difficult for any individual without a mathematical background to approach (Rabiner,1989) [4].

3.2 Central Problems of HMM's

There are three problems which are central to HMM's, which will also be the main focus of this chapter. The three problems are:

(1) - Given a model, how likely is a given observed sequence?

(2) - What is the most probable path (state sequence) that gives rise to the observed sequence?
(3) - What are the HMM parameters given a set of sequences?

Questions (1) and (2) are similar in that they both require uncovering the unobservable sequence of states in order to arrive at the corresponding answer. The first question deals with looking at every possible sequence of states that could give rise to the observed sequence, given you are dealing with a specific set of parameters. This question is solved using one of two algorithms, the forward and backward algorithms.

The second question deals with finding the sequence of states that is most likely to have been responsible for the observed sequence, given a set of parameters and a sequence. A term frequently used to describe this process is "decoding", which originated in the field of speech recognition (Rabiner,1989) [4]. The algorithm that is used to solve this problem is the Viterbi algorithm.

The third question is concerned with uncovering the optimal parameters of the HMM in the case that they are unknown. It is often the case that the parameters of the model are unknown to the researcher and need to be estimated. The methods used to do so is either the Baum-Welch algorithm or Viterbi training.

In the following section, a description of each algorithm will be laid out. Ultimately, this thesis aims to uncover and compare the differences in state-sequence prediction accuracy using the parameter estimating algorithms, by way of the Viterbi algorithm to uncover the hidden state-sequences. It should also be noted that in this thesis, the forward and backward algorithms will be used only through the Baum-Welch algorithm, and not for their individual purposes of uncovering the total probability of a given observed sequence.

3.3 Algorithms

3.3.1 Viterbi Algorithm

The Viterbi algorithm is concerned with combing through every possible combination of state sequences and determining the one that is most likely to have produced the observed sequence. Using the previously mentioned CpG example, the state sequences $(A_+C_+G_+)$, $(A_-C_-G_-)$, (A_+, C_-, G_+) give rise to the symbol sequence ACG. These do so though with different probabilities. If we were to choose just one path for the prediction, the Viterbi algorithm says the one with the highest probability should be chosen. By listing all possible sequences of hidden states and finding the probability of the observed sequence for each of the combinations, selecting the most probable sequence of hidden states is the sequence that maximizes:

$$P(\text{A,C,G}|+,+,+),\ P(\text{A,C,G}|+,+,-),\ P(\text{A,C,G}|+,-,+),\ldots,P(\text{A,C,G}|-,-,-)$$

This approach is an option, but to find the most probable sequence by calculating each combination is computationally intensive. Let the path with maximum probability be denoted as π^* such that:

$$\pi^* = argmax_\pi P(x, \pi).$$

Finding π^* can be done recursively. Define $v_k(i)$ as the most probable path ending in state k at time i, over all possible states k. This new variable is termed the Viterbi variable, and is defined as:

$$v_l(i+1) = e_l(x_{i+1}) max_k(v_k(i) a_{kl}).$$

Table 3.1: Viterbi Algorithm

Step	Action
1	$(i = 0)$: $v_0(0) = 1, v_k(0) = 0$, for $k > 0$.
2	$(i = 1, \ldots, L)$: $v_l(i) = e_l(x_i) max_k(v_k(i-1)a_{kl})$.
3	$P(x, \pi^*) = max_k(v_k(L)a_{k0})$. $\pi^*_L = argmax_k(v_k(L)a_{k0})$.

3.3.2 Forward Algorithm

The Viterbi algorithm is limited in providing information about a sequence of states in that it returns only the most likely sequence of states that would produce the observed sequence. It may be that the total probability of observing a sequence is of interest. Also, the most likely sequence may not be the only state sequence that holds a large probability of producing the sequence being observed. In the case where the total probability is of interest, the forward algorithm is used.

The method of total probability is done by replacing the maximization steps with summations. Again using the example "ACG", the forward algorithm finds each possible sequence of the hidden states, and sum these probabilities, using:

$$P(x) = \sum_{\pi} P(x, \pi) \tag{3.1}$$

$$= P(\text{A,C,G}|+,+,+) + P(\text{A,C,G}|+,+,-) + \ldots + P(\text{A,C,G}|-,-,-). \tag{3.2}$$

Calculating the probability using this method is also computationally exhausting, even more so with large models or long sequences. In fact using this method, $P(x) = \sum_{\pi} P(x, \pi)$, the number of paths increases exponentially with the length of the sequence, so calculations may not be entirely reasonable.

The probability $f_k(i)$ can be represented as:

$$f_k(i) = P(x_1, \ldots, x_i, \pi_i = k),$$

which is the total probability of observing the sequence up to and including x_i. The recursive equation corresponding to the probability above is written:

$$f_l(i+1) = e_l(x_{i+1}) \sum_k f_k(i) a_{kl}.$$

Table 3.2: Forward Algorithm

Step	Action
1	$(i = 0)$: $f_0(0) = 1, f_k(0) = 0$, for $k > 0$.
2	$(i = 1, \ldots, L)$: $f_l(i) = e_l(x_i) \sum_k f_k(i-1) a_{kl}$.
3	$P(x) = \sum_k f_k(L) a_{k0}$

3.3.3 Backward Algorithm

An analogous approach to the forward computation previously discussed is termed the backward algorithm, which arrives at the total probability through backward recursion. The approach to this method is to first consider the total probability of an observed sequence, given the ith symbol is emitted from state k:

$$P(x, \pi_i = k) = P(x_1, \ldots, x_i, \pi_i = k) P(x_{i+1}, \ldots, x_L | x_1, \ldots, x_i, \pi_i = k) \tag{3.3}$$

$$= P(x_1, \ldots, x_i, \pi_i = k) P(x_{i+1}, \ldots, x_L | \pi_i = k), \tag{3.4}$$

which is the case because everything following state k depends only on state k. The first probability is the previously discussed forward probability, $f_k(i)$. The second probability is denoted $b_k(i)$, such that:

$$b_k(i) = P(x_{i+1}, \ldots, x_L | \pi_i = k).$$

Table 3.3: Backward Algorithm

Step	Action
1	$(i = 0)$: $b_k(L) = a_{k0}$, for all k.
2	$(i = 1, \ldots, L)$: $b_k(i) = \sum_l e_l(x_i + 1) b_l(i + 1) a_{kl}$.
3	$P(x) = \sum_l b_l(1) a_{0l} e_l(x_1)$

3.3.4 Baum-Welch Algorithm

An instance of the EM (Expectation-Maximization) algorithm, the Baum-Welch algorithm is also termed the forward-backward algorithm which stems from the fact that, at each possible step, in each possible path of a given sequence, it computes the forward probability of arriving at the current state (given the current model approximation) and the backward probability of generating the final state of the model (again given the current approximation). Corrections can be made to the approximated HMM parameters to improve the developing probabilities, where the adjustments form the basis of the algorithm iterations.

In other words, in the first pass the Baum-Welch algorithm computes a set of forward probabilities which provide the probability of ending up in any particular state given the first k observations in the sequence. In the second pass, the algorithm computes a set of backward probabilities which provide the probability of observing the remaining observations given any starting point k.

3.3.5 Estimation when the state sequence is known

A set of genomic sequences where the CpG islands are already labeled based on experimental data, is an example of a time the parameters can be calculated using the already known state-sequence. When all of the paths are known, the number of times each particular transition or emission is used in the set of training sequences can be counted. Training sequences are a set of sequences assumed to be of the type that fit the model well, defined as $x_1, \ldots, x_n$. The assumption is that they are independent and thus the joint probability of all sequences given a set of parameters is the product of the probabilities of the individual sequences. Ultimately, $\log P(x_1, \ldots, x_n|q) = \sum \log P(x_j|q)$, where q represents the entire current set of values of the parameters in the model (as and es). This is equal to the log likelihood of the model.

For a fixed set of data and underlying probability model, the method of maximum likelihood selects values of the model parameters that maximize the likelihood function. The maximum likelihood estimators for a_{kl} and $e_k(b)$ are:

$$a_{kl} = A_{kl} / \sum_{l'} A_{kl'}$$

$$e_k(b) = E_k(b) / \sum_{b'} E_k(b').$$

where A_{kl} is the number of times that the sequence transitions from state k to state l. Similarly $E_k(b)$ is the number of emissions of b while in state k.

3.3.6 Deriving The Number of Transitions and Emissions

The expected count of transitions out of state k and into state l can be written:

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1),$$

where f_k^j is the forward variable for the jth sequence and similarly b_l^j is the backward variable.

Similarly,

$$E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{i|x_i^j=b} f_k^j(i) b_k^j(i),$$

where $E_k(b)$ is the expected number of times seeing observation b while in state k.

Table 3.4: Baum-Welch Algorithm

Step	Action
1	Set the A_{kl} and $E_k(b)$ to estimated values, or leave as 0.
2	Calculate $f_k(i)$ and $b_k(i)$ for sequence j using the forward/backward algorithm.
3	Add the newly calculated contributions to the A and E variables.
4	Calculate the new model parameters using the MLE estimates for a_{kl} and $e_k(b)$.
$\updownarrow^{\text{False - 1}}_{\text{True - 5}}$	Number of iterations met or other requirement satisfied?
5	Stop.

3.3.7 Viterbi Training

An alternative to the Baum-Welch algorithm is called Viterbi training, which is also used to estimate the parameters of an HMM. Essentially, Viterbi training uses the Viterbi algorithm to find π^*. Once π^* is found, the estimated state-sequence is used to estimate the transition and emission probabilities. Viterbi training maximizes $P(x|\lambda, \pi^*)$, but unlike the Baum-Welch algorithm doesn't maximize $P(x|\lambda)$ (DeFonzo, Aluffi-Pentini, Parisi, 2007) [8].

Table 3.5: Viterbi Training Algorithm

Step	Action
1	Set the A_{kl} and $E_k(b)$ to estimated values, or leave as 0.
2	Calculate π_i^* for training sequence $i = 1, \ldots, n$ using the Viterbi algorithm.
3	Calculate the A_{kl} and $E_k(b)$ using the calculated π_i^*'s.
4	Calculate the new model parameters, a_{kl} and $e_k(b)$.
$\updownarrow^{\text{False - 1}}_{\text{True - 5}}$	Number of iterations met or other requirement satisfied?
5	Stop.

3.3.8 Differences Between the Baum-Welch and Viterbi Training Algorithms

Key differences exist between the Baum-Welch and the Viterbi Training algorithm. Different from the Baum-Welch, the Viterbi training algorithm finds the values of the parameters that maximizes the contribution to the likelihood from the most probable paths for all sequences, but does not maximize the true likelihood (Durbin, Eddy, Krogh, and Mitchison, 1998) [7]. In other words, the Baum-Welch algorithm maximizes

$$P(x^1, \ldots, x^n|\theta),$$

while the Viterbi training algorithm maximizes

$$P(x^1, \ldots, x^n|\theta, \pi^*(x^1), \ldots, \pi^*(x^n)).$$

In general, the Viterbi training method performs less well than the Baum-Welch, yet is widely used when the main method of state sequence decoding is through the Viterbi method (Durbin et al., 1998) [7].

3.4 Underflow and Overflow errors

Due to the exceedingly small values that are seen when working with emission and transition probabilities, it can be expected that the calculated forward and backward values would be small as well. This can create a problem for calculations when the sequences being dealt with are large or even moderate in length. As the sequences grow, the forward and backward values tend to zero, vanishing over time and hindering the approximation process. In some instances the forward and backward values tend to zero within very few iterations.

There are two solutions to the problem of vanishing probabilities that will be addressed. One solution to the problem is taking the log of the values which transforms the data, making it more reasonable for calculations.

3.4.1 Scaling The Forward Variable

In order to keep the forward variable in a manageable range, scaling is used (Durbin et al., 1998) [7]. The new scaled forward variable is defined as:

$$\tilde{f}_l(i+1) = \frac{1}{s_{i+1}} e_l(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kl}$$

The scaling variable is defined:

$$s_{i+1} = \sum_l e_l(x_{i+1}) \sum_k \tilde{f}_k(i) a_{kl}$$

3.4.2 Log Transformation: Viterbi Algorithm

The logarithm of probabilities should be used when working with the Viterbi Algorithm (Durbin et al., 1998) [7]. As a result, the log of the product becomes the sum and the underflow errors are eliminated. The log of the Viterbi variable becomes:

$$V_l(i+1) = \tilde{e}_l(x_{i+1}) + max_k(V_k(i) + \tilde{a}_{kl})$$

Chapter 4

Data & Results

4.1 Implementation of Algorithms

Each of the algorithms, Viterbi, Baum-Welch (including the forward and backward algorithm), and Viterbi training algorithms were programmed in java. The Viterbi algorithm's results were cross checked with Matlab's hmmgenerate() and hmmviterbi() functions, and all of the algorithms were cross checked using corresponding problems in, "Problems and Solutions in Biological Sequence Analysis", by Mark Borodovsky and Svetlana Ekisheva. A random number generator that randomly selects sequences for training and estimation was programmed using R. All of the algorithms can be found in the appendices of this thesis.

4.2 Description of the Data and Methods

The data used for the initial stages of analysis consisted of sequences of DNA nucleotides from the NCBI database. The data that was retrieved from the NCBI database was presented in FASTA format, and also contained the locations of confirmed CpG islands. A total number of 19 sequences were used, with the longest sequence being 152141 nucleotides in length and the shortest being 1870 nucleotides in length. In the table above, "# of CpG islands" refers to the total number of CpG islands within that sequence, and "% CpG" refers to the amount of nucleotides within the total number of CpG islands for a specific sequence, proportional to the total sequence length. Table 4.1 has a full description of the sequences used during the research for this thesis.

Table 4.1: Description of the DNA sequences used for data analysis

Sequence ID	Seq. Length	# CpG Islands	% CpG
U07000.1	152141	6	0.0192
U10687.1	7411	4	0.3283
U12202.1	11495	5	0.1203
X12811.1	4942	4	0.3567
V01510.1	2231	5	0.8126
M23533.1	8658	4	0.2176
M58508.1	3604	4	0.2679
M95740.1	1870	3	0.5941
M59856.1	4480	3	0.4167
M57424.1	2585	3	0.3749
D13370.1	4982	3	0.2033
X69907.1	3730	3	0.2488
X69908.1	9457	3	0.1087
J05451.1	15016	3	0.0665
M27132.1	15067	3	0.0644
X72861.1	10186	3	0.0916
Z15025.1	3683	3	0.3519
U07561.1	84539	6	0.0231
X62695.1	35962	4	0.0493

As can be seen from the table above, the maximum number of CpG islands within a sequence is 6, and the minimum is 3. Typically, CpG islands occur in large blocks spanning hundreds or thousands of nucleotides in length. The gaps that occur between CpG islands, non-CpG regions, display similar behavior spanning hundreds or thousands of nucleotides.

Table 4.2: Nucleotide composition.

Seq - DNA Region	Nucleotide Distribution
1 - CpG	A - 0.1763, C - 0.3208, G - 0.3806, T - 0.1223
1 - Non-CpG	A - 0.2198, C - 0.2595, G - 0.2665, T - 0.2543
2 - CpG	A - 0.2737, C - 0.2581, G - 0.2565, T - 0.2117
2 - Non-CpG	A - 0.1763, C - 0.3246, G - 0.3206, T - 0.1784
3 - CpG	A - 0.1974, C - 0.5322, G - 0.0889, T - 0.1815
3 - Non-CpG	A - 0.2232, C - 0.3018, G - 0.2770, T - 0.1980
4 - CpG	A - 0.2087, C - 0.2660, G - 0.2655, T - 0.2598
4 - Non-CpG	A - 0.2696, C - 0.2076, G - 0.2356, T - 0.2872
5 - CpG	A - 0.1224, C - 0.3789, G - 0.2934, T - 0.2052
5 - Non-CpG	A - 0.1388, C - 0.3397, G - 0.3541, T - 0.1675
6 - CpG	A - 0.1704, C - 0.3471, G - 0.3217, T - 0.1608
6 - Non-CpG	A - 0.2634, C - 0.2319, G - 0.2396, T - 0.2651
7 - CpG	A - 0.1474, C - 0.3624, G - 0.3273, T - 0.1630
7 - Non-CpG	A - 0.1925, C - 0.2981, G - 0.2415, T - 0.2679
8 - CpG	A - 0.1872, C - 0.2097, G - 0.4104, T - 0.1926
8 - Non-CpG	A - 0.2227, C - 0.2345, G - 0.2727, T - 0.2701
9 - CpG	A - 0.1237, C - 0.3958, G - 0.3508, T - 0.1296
9 - Non-CpG	A - 0.1485, C - 0.3176, G - 0.3295, T - 0.2044
10 - CpG	A - 0.1744, C - 0.3220, G - 0.3498, T - 0.1538
10 - Non-CpG	A - 0.2772, C - 0.2110, G - 0.2222, T - 0.2896
11 - CpG	A - 0.1530, C - 0.3544, G - 0.3406, T - 0.1520
11 - Non-CpG	A - 0.2661, C - 0.2184, G - 0.2318, T - 0.2837
12 - CpG	A - 0.2112, C - 0.2446, G - 0.3319, T - 0.2123
12 - Non-CpG	A - 0.2584, C - 0.2473, G - 0.2063, T - 0.2880
13 - CpG	A - 0.2344, C - 0.2899, G - 0.2977, T - 0.1780
13 - Non-CpG	A - 0.2686, C - 0.2266, G - 0.2387, T - 0.2661
14 - CpG	A - 0.1622, C - 0.3053, G - 0.3103, T - 0.2222
14 - Non-CpG	A - 0.2573, C - 0.2301, G - 0.2281, T - 0.2845
15 - CpG	A - 0.1617, C - 0.4192, G - 0.2503, T - 0.1689
15 - Non-CpG	A - 0.2221, C - 0.2819, G - 0.2885, T - 0.2074
16 - CpG	A - 0.1672, C - 0.2937, G - 0.2905, T - 0.2487
16 - Non-CpG	A - 0.2499, C - 0.2178, G - 0.2274, T - 0.3050
17 - CpG	A - 0.1227, C - 0.3434, G - 0.3279, T - 0.2060
17 - Non-CpG	A - 0.1864, C - 0.2941, G - 0.2660, T - 0.2535
18 - CpG	A - 0.1914, C - 0.3045, G - 0.3414, T - 0.1627
18 - Non-CpG	A - 0.2611, C - 0.2260, G - 0.2250, T - 0.2879
19 - CpG	A - 0.1505, C - 0.3253, G - 0.3281, T - 0.1962
19 - Non-CpG	A - 0.2577, C - 0.2150, G - 0.2500, T - 0.2773

Table 4.3: Actual Transition Composition.

Sequence ID	a_{00}	a_{01}	a_{10}	a_{11}
U07000.1	.99795	.00205	.00004	.99996
U10687.1	.99836	.00164	.00804	.99196
U12202.1	.99639	.00362	.00049	.99951
X12811.1	.99773	.00227	.00126	.99874
V01510.1	.99724	.00276	.00157	.99843
M23533.1	.99788	.00212	.00059	.99941
M58508.1	.99870	.00130	.00152	.99848
M95740.1	.99730	.00270	.00395	.99605
M59856.1	.99839	.00161	.00114	.99885
M57424.1	.99690	.00310	.00186	.99814
D13370.1	.99704	.00296	.00076	.99924
X69907.1	.99677	.00323	.00107	.99893
X69908.1	.99708	.002928	.00036	.99644
J05451.1	.99700	.00300	.00021	.99978
M27132.1	.99690	.00309	.00021	.99979
X72861.1	.99679	.00322	.00032	.99967
Z15025.1	.99843	.00157	.00170	.99830
U07561.1	.99693	.00307	.00007	.99993
X62695.1	.99774	.00226	.00012	.99988

4.3 Estimated Sequence Information

The following state sequence estimation process was done by using the individual extracted values from the data to calculate the individual sequence parameters. Using the individual sequence parameters, the Viterbi algorithm was used to estimate the corresponding state

sequences. The parameters and the estimation results are in the following tables.

Tables 4.4 and 4.5 give individual and summary statistics, that refer to the accuracy of the state sequence predictions using individual sequence parameter estimation. In table 4.4, accuracy refers to the percent of the total states that the Viterbi algorithm estimated correctly with respect to the corresponding actual sequence.

Table 4.4: Individual accuracies for the Viterbi algorithm.

Seq - DNA Region	Viterbi Accuracy
U07000.1	0.99205
U10687.1	0.790582
U12202.1	0.884646
X12811.1	0.714290
V01510.1	0.802331
M23533.1	0.909563
M95740.1	0.906215
M59856.1	0.840107
M57424.1	0.927902
D13370.1	0.959768
X69907.1	0.962667
X69908.1	0.88150
J05451.1	0.86032
M27132.1	0.93533
X72861.1	0.81901
Z15025.1	0.90605
U07561.1	0.86412
X62695.1	0.98418

Table 4.5: Summary statistics for state sequence prediction using individual sequence parameter estimation.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max	SD
.7143	.8926	.8846	.8809	.9316	.9920	.0740

Table 4.6: Actual CpG island location.

Sequence ID	CpG Islands
U07000.1	(9167-9368), (14460-15861), (16078-16383), (74401-74645), (96703-97186), (97190-97483)
U10687.1	(94-406), (476-1893), (1907-2401), (2460-2670)
U12202.1	(1194-1419), (1818-2096), (3185-3503), (3872-4178), (4546-4802)
X12811.1	(196-527), (1175-1497), (2662-2869), (3656-4559)
V01510.1	(48-361), (381-752), (759-1352), (1380-1669), (1695-1942)
M23533.1	(228-610), (615-1109), (1111-1432), (7328-8015)
M58508.1	(241-550), (561-813), (856-1615), (1752-3506)
M95740.1	(274-594), (633-873), (916-1467)
M59856.1	(1478-2289), (2328-2782), (2814-3416)
M57424.1	(208-510), (529-737), (778-1237)
D13370.1	(1668-1996), (2025-2241), (2359-2828)
X69907.1	(290-512), (821-1075), (1158-1610)
X69908.1	(662-871), (2952-3212), (4164-4723)
J05451.1	(2545-2947), (3621-3981), (3996-4233)
M27132.1	(4962-5164), (6520-7007), (7054-7336)
X72861.1	(890-1280), (2056-2377), (2427-2649)
Z15025.1	(767-1516), (1635-1948), (1995-2229)
U07561.1	(16956-17162), (29143-29370), (31649-31929), (37194-37849), (38120-38349), (79936-80293)
X62695.1	(7153-7370), (8361-8902), (27258-27578), (27632-28328)

Table 4.7: Viterbi estimated CpG island location.

Sequence ID	CpG Islands
U07000.1	(14431-1636), (68541-68989), (96677-97462)
U10687.1	(1-1688), (2075-3626)
U12202.1	(319-378),(567-767),(1190-1391),(1807-2113),(2495-2766),(3177-3443), (3852-4118),(4527-4799),(5212-5483),(5896-6044),(6462-6609)
X12811.1	(1-641), (3846-4941)
V01510.1	(259-2091)
M23533.1	(69-1092), (7318-8286)
M58508.1	(1-91), (272-1595), (1796-3603)
M95740.1	(243-1653)
M59856.1	(1530-3616)
M57424.1	(251-1238)
D13370.1	(1631-2826)
X69907.1	(262-1632)
X69908.1	(278-884), (2945-3221), (3809-5276)
J05451.1	(2677-4362)
M27132.1	(4251-5004), (6530-7649)
X72861.1	(561-1286), (2009-2820)
Z15025.1	(435-1883)
U07561.1	(11247-17392), (28800-32846)
X62695.1	(7204-7490), (8457-8954), (27327-28455)

When looking at the table 4.7 above, it can be seen that, although the HMM doesn't match the actual CpG island count per sequence (sometimes the Viterbi algorithms will either miss a CpG island, predict a CpG island that doesn't exist, or combine multiple CpG islands into one CpG island), or recover every portion of the sequence that is a CpG

island, it does do a good job of finding the CpG islands overall. In other words, the Viterbi algorithm, when given the exact parameters, recovers the general location of CpG islands within a sequence. As mentioned, it can combine the locations of multiple CpG islands into one CpG island. It could be that when given such extreme parameters, since either unlikely to enter or leave a given state, it rarely does so. For example, sequence U07000.1 has CpG islands from nucleotide 14,660 to 15,861. The Viterbi algorithm estimates the CpG island starting at nucleotide 14,431 to 16,036. It is interesting that, although this sequence is over 150,000 nucleotides in length, the algorithm was able to pick out this region, about 2,000 nucleotides in length, as a CpG island. Another example for the same sequence, a CpG island occurs starting at nucleotide 97,190 and ending at nucleotide 97,483. The Viterbi algorithm has estimated that a CpG island occurs beginning at nucleotide 96,677 and ending at nucleotide 97,462. This is a reoccurring theme for the Viterbi algorithm, given this data set and its parameters.

On the other hand though, the Viterbi algorithm tends to estimate CpG islands where they do no occur, perhaps phantom CpG islands. For example, for sequence U12202.1 the Viterbi algorithm seemed to struggle, estimating about 5 CpG islands that are non-CpG regions. The results for this sequence are the exception, and not the norm, although it gives a good example of how the Viterbi algorithm can mistake non-CpG regions for CpG islands. Another example using sequence U07000.1 again, the Viterbi algorithm estimates that a CpG island occurs starting at nucleotide 68,541 and ending at nucleotide 68,989, although no CpG island exists within the given span of nucleotides. Hence, this sequence only has one "phantom" CpG island. The Viterbi algorithm usually incorrectly estimates either one CpG island, or none at all. It can also miss CpG islands. Although there exist a CpG island starting at nucleotide 9,167 and ending at nucleotide 9,368, the Viterbi algorithm is oblivious to this, labeling this region non-CpG. On second glance though, we can see that this CpG island spans only about 200 nucleotides. Once again, given the extreme parameters, this CpG island may not be of large enough length to be detected by the Viterbi algorithm.

In the following section, a comparison of parameter estimation techniques will be examined using the Viterbi algorithm to estimate the state sequence. Realistically, in normal situations actual parameters will not be available to the researcher. Although we have been able to determine the Viterbi algorithms performance under best-case situations, it is more useful to determine how well parameter estimation techniques, combined with the Viterbi algorithm, will estimate state sequences.

4.4 Comparison of Parameter Estimation Methods

4.4.1 Methods

Three means of acquiring parameters were compared which were, the extracted value method, the Baum-Welch algorithm, and the Viterbi training algorithm. The extracted value method was trained using the actual parameters from the data. Both the Baum-Welch and the Viterbi training method used an initial guess for transition probabilities of

$$a_{00} = .99998,\ a_{01} = .00002,\ a_{10} = .0001, \text{ and } a_{11} = .9999,$$

which were chosen as an arbitrarily close guess. The initial guess for the emission probabilities were

$$e_0(A) = .15,\ e_0(C) = .35,\ e_0(G) = .35,\ e_0(T) = .15,\ e_1(A) = .25,\ e_1(C) = .25,\ e_1(G) = .25,$$
$$e_1(T) = .25,$$

which were again chosen as an arbitrarily close guess to the actual parameters, where 0 represents the CpG state and 1 represents the non-CpG state. Each method randomly sampled 9 sequences and used them for training sequences. Nine of the 10 remaining sequences not selected for training were then used for state sequence estimation. This

represented one simulation and a total of 100 simulations were done for each parameter estimation method. The estimated state sequence was then compared to the corresponding actual sequence. The total state accuracy was the measure of interest. Each estimated state was compared to the actual state corresponding the the same position in the sequence. If the estimated state equaled the actual state at the given position in the sequence, a match was marked. The number of matches were divided by the sequence length, a measure of the total estimation accuracy. Each simulation gave nine accuracies which were averaged to get the average simulation accuracy. A total of 100 average simulation accuracies were calculated.

Two methods of sampling were used. The first method used a completely randomized sampling method for each parameter estimation method. Each simulation, between and within methods randomly generated the sequences to train and estimate on. The second method used 9 random sequences for training and 9 not-selected sequences for estimation, similar to the previous method. The difference is that the sequences used for both training and estimation for the second sampling method are the same across parameter estimation methods. The sampling is random within each method, but not random between the methods. For example, if sequences 1,3,5,7,9,11,13,15,17 were chosen randomly selected for training in an individual simulation, then each method, extracted value, Baum-Welch, and Viterbi training would train on those sequences for the respective simulation. The same method would be applied for the estimation process.

4.5 Results

4.5.1 Descriptive Statistics

Tables 4.8 and 4.9 are the descriptive statistics of the three algorithmic simulations done for both sampling methods. As can be seen, the median and mean accuracy for the extracted value and Baum-Welch algorithms are higher than the Viterbi training method for both sampling methods. Also, the spread of average accuracies is less for the extracted and Baum-Welch algorithms when compared to the Viterbi training algorithm. For both the extracted and Baum-Welch the minimum accuracy is around .6 and the maximum accuracy is around .9, while for the Viterbi training algorithm, the minimum for the completely random sampling method is about .2 and for the repeated measures is about .5. The maximum accuracy for the Viterbi training algorithm in the completely random sampling and repeated measures simulation is about .9. Also, the mean and median for all of the simulations are almost equal, signaling little skewness in the values.

Table 4.8: Randomized Simulation Summary

Algorithm	Min.	1st Qu.	Median	Mean	3rd Qu.	Max	SD
Extracted Value	.5807	.6847	.7377	.7327	.7769	.9069	.0710
Baum-Welch	.6293	.7159	.7538	.7547	.7961	.8926	.0552
Viterbi Training	.2281	.5953	.6712	.6544	.7421	.8935	.1173

Table 4.9: Repeated Measures Simulation Summary

Algorithm	Min.	1st Qu.	Median	Mean	3rd Qu.	Max	SD
Extracted Value	.5928	.7063	.7495	.7511	.7940	.9033	.0600
Baum-Welch	.6348	.7224	.7578	.7625	.8025	.8853	.0508
Viterbi Training	.4725	.6115	.6658	.6656	.7215	.8788	.0799

4.5.2 Comparison of Algorithm Estimation Results

Assessment of Normality

The tables below give an idea to the spread of the average accuracies of the algorithms for both sampling methods.

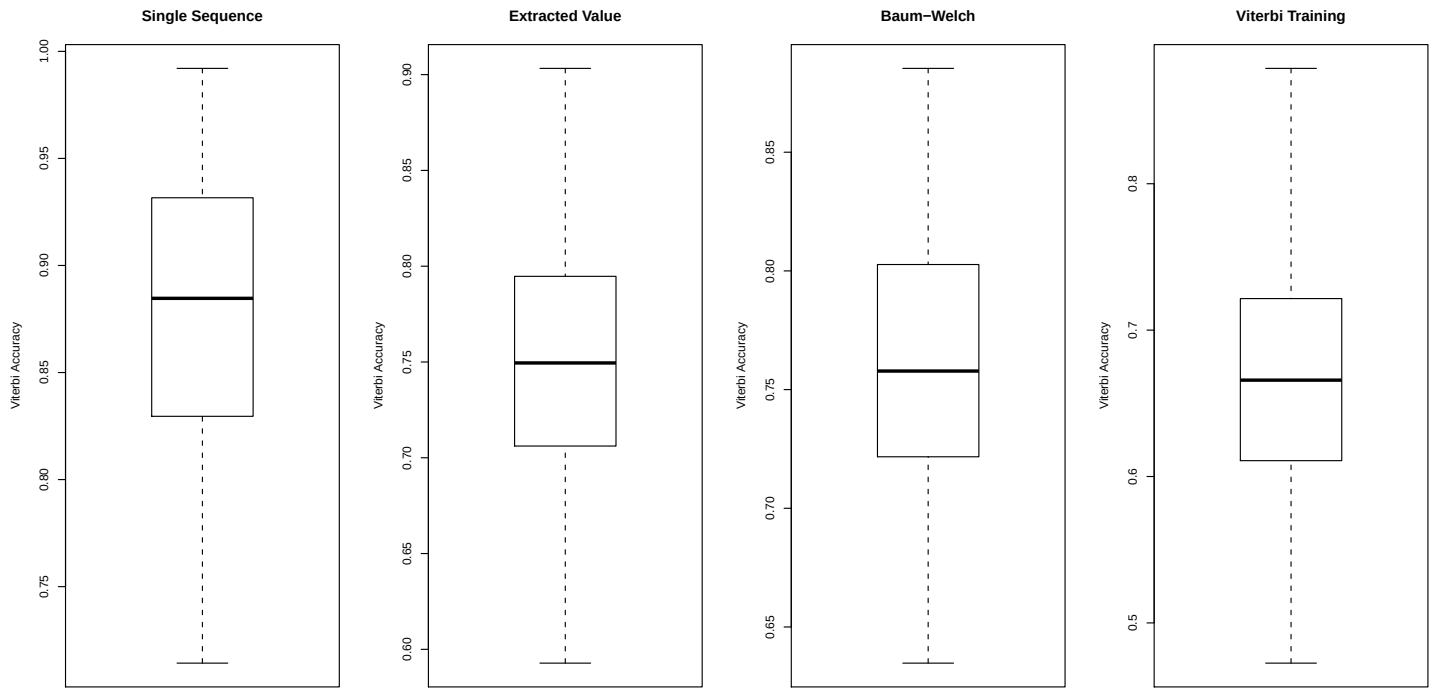

Figure 4.1: Single sequence and completely random data box plots for the average accuracies of the algorithms.

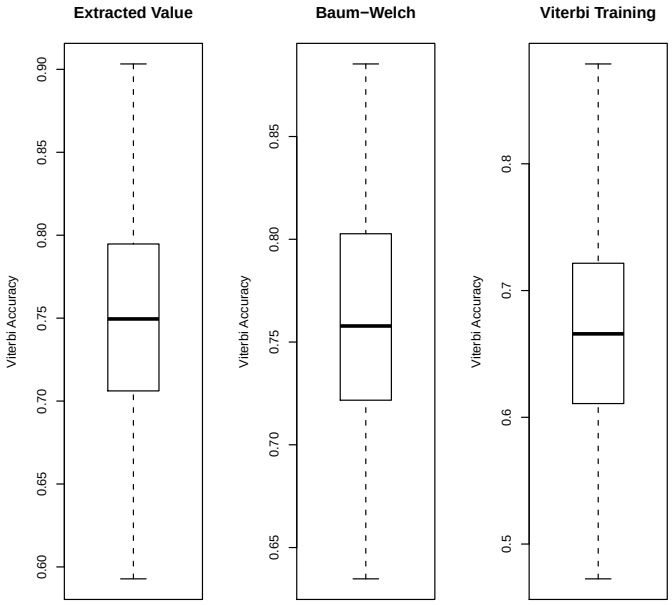

Figure 4.2: Repeated measures boxplots for the average accuracies of the algorithms.

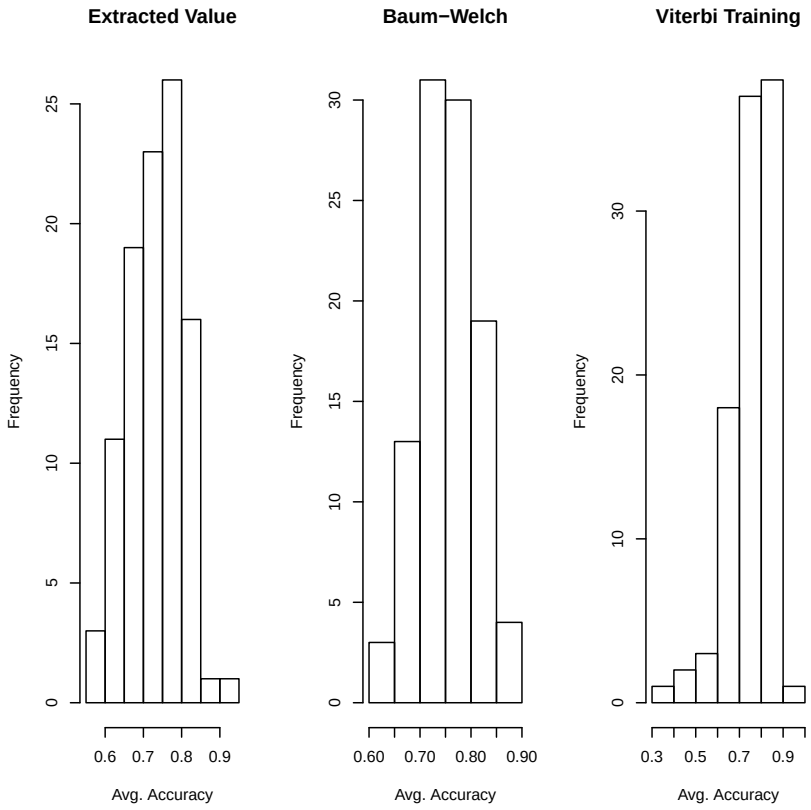

Figure 4.3: Single sequence and completely random data histograms for the average accuracies of the algorithms.

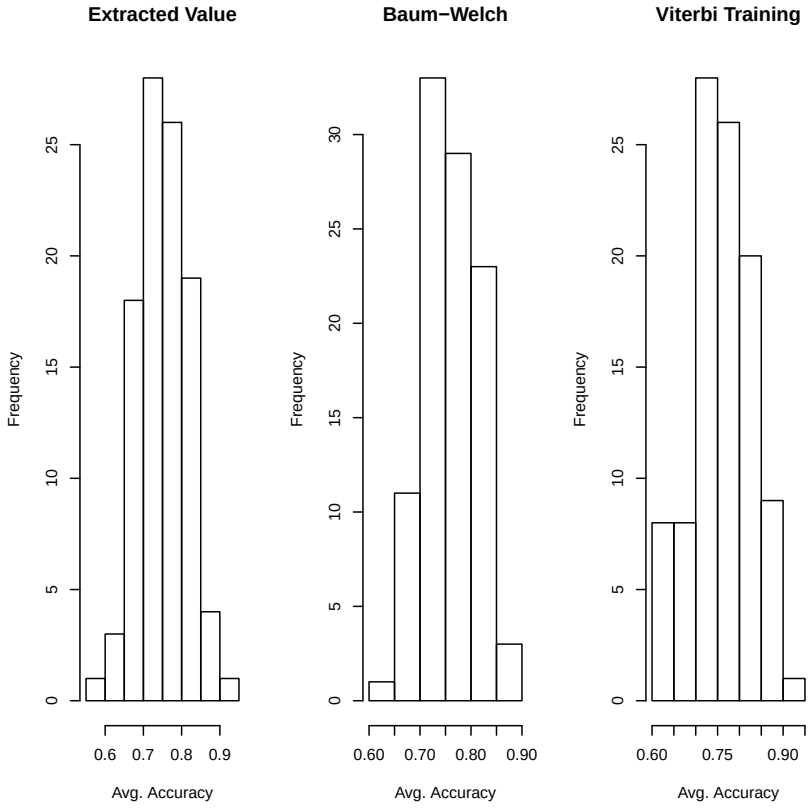

Figure 4.4: Repeated measure histograms for the average accuracies of the algorithms.

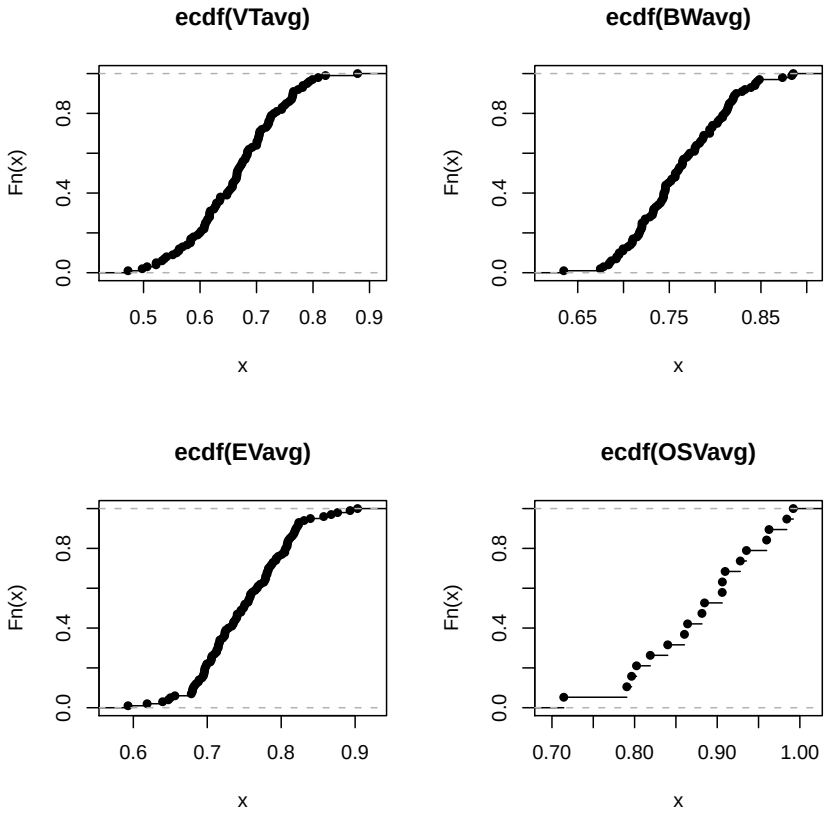

Figure 4.5: Single sequence and repeated measures empirical cdf for the average accuracies of the algorithms.

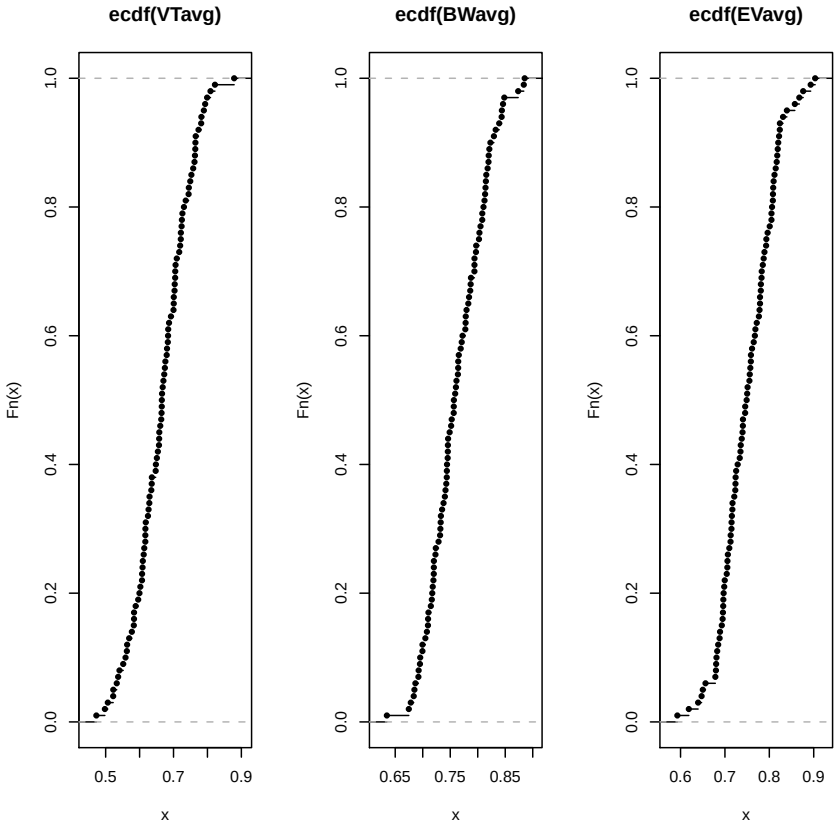

Figure 4.6: Repeated measures empirical cdf for the average accuracies of the algorithms.

To assess the normality of the two sets of data, the Lilliefors test was used, which is a distributional test of the Kolmogorov-Smirnoff type.

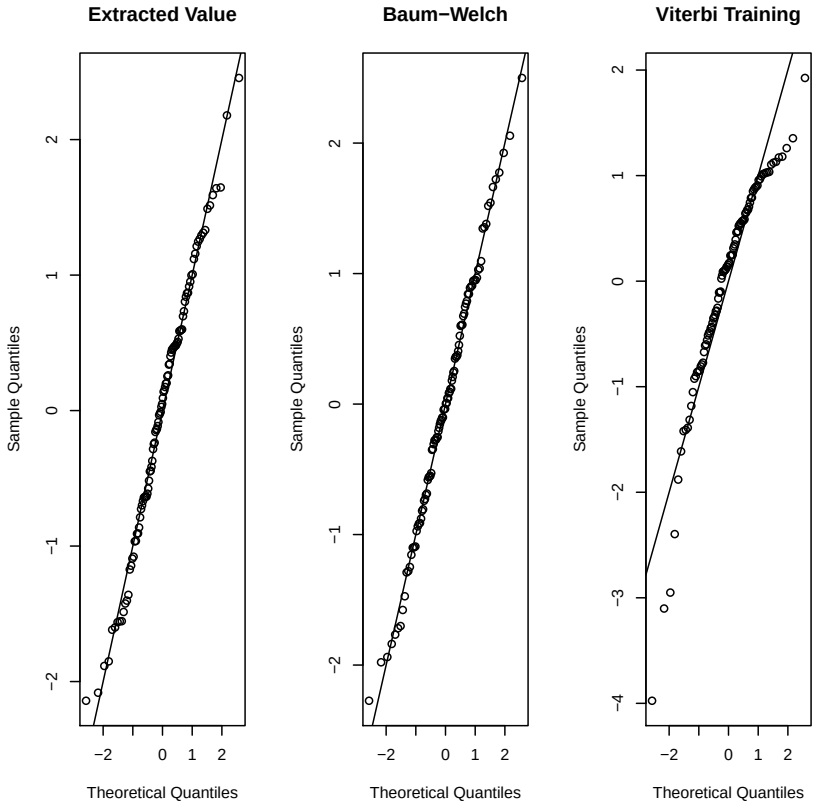

Figure 4.7: Single sequence and completely random qq plots for the average accuracies of the algorithms.

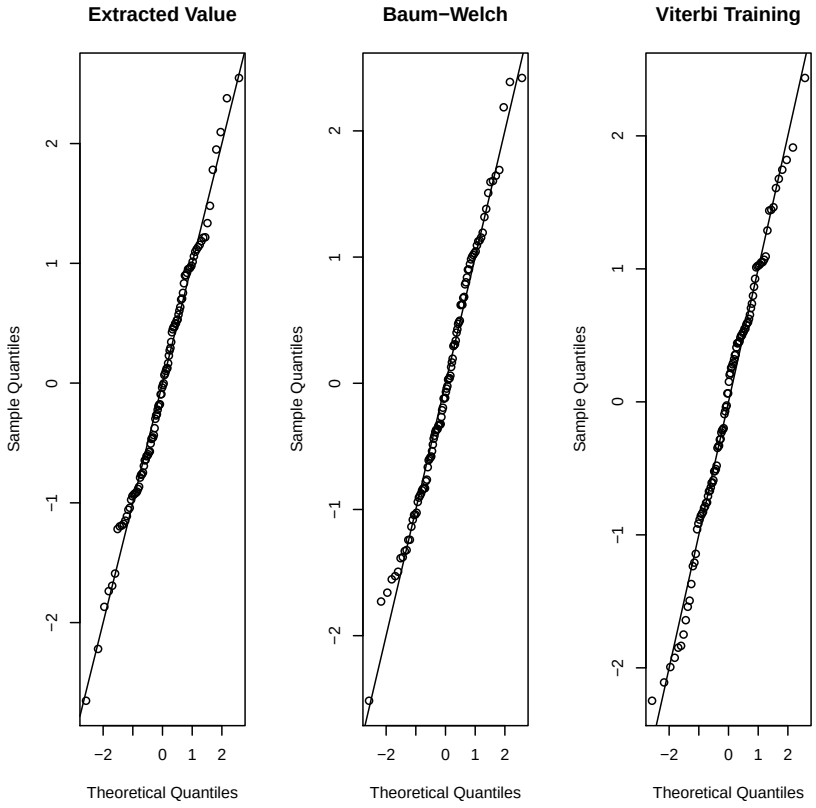

Figure 4.8: Repeated measures qq plots for the average accuracies of the algorithms.

For the completely randomized sample, both the extracted value ($p - value = .6214$) and Baum-Welch (p-value = .9216) data set were normal according to the Lilliefors test, $\alpha = .05$. The only question of normality was with the Viterbi training ($p - value = .01416$) data set, which can be seen visually by use of a qq-plot.

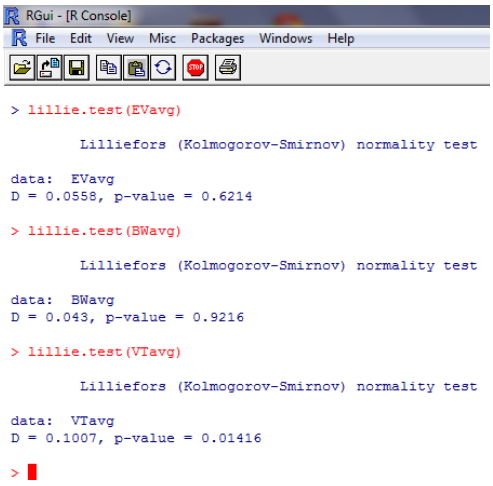

Figure 4.9: Lilliefors for normality, completely random data.

For the repeated measures sample, all data sets were normal according to the Lilliefors test. However, the variances of the means of the three algorithms happened to be unequal for both sampling methods, as was shown by Levene's test for homogeneity of variance. For the completely random and repeated measures data set, Levene's test returned $p-value \approx 0$, $\alpha = .05$, signaling heterogeneity of the variances. There are methods that describe a way for correcting unequal variances, for cases within one-way analysis of variance. When the variances are unknown and differ among groups, replacing the standardized variables with Behrens-Fisher T values is suggested, which does not require equality of group variances (Rice and Gaines, 1989) [3]. For the case of unequal variances within repeated measures sampling, a method for testing a number of hypotheses by taking the generalized p-value approach has worked to correct for unequal variances (Ho and Weerhandi, 2005) [6].

Several transformation methods were performed. The methods of transformation performed include $Y' = \sqrt{Y}$, $Y' = \log Y$, $Y' = \frac{1}{Y}$, and $Y' = 2arcsin\sqrt{Y}$. One of the methods mentioned helped to correct for normality, $Y' = 2arcsin\sqrt{Y}$, which normalized the data as was determined by Lilliefors tests for normality. The transformation however had no effect on heterogeneity of variances, using a Levene's test for both sampling methods to

determine the nature of the variances of the transformed data.

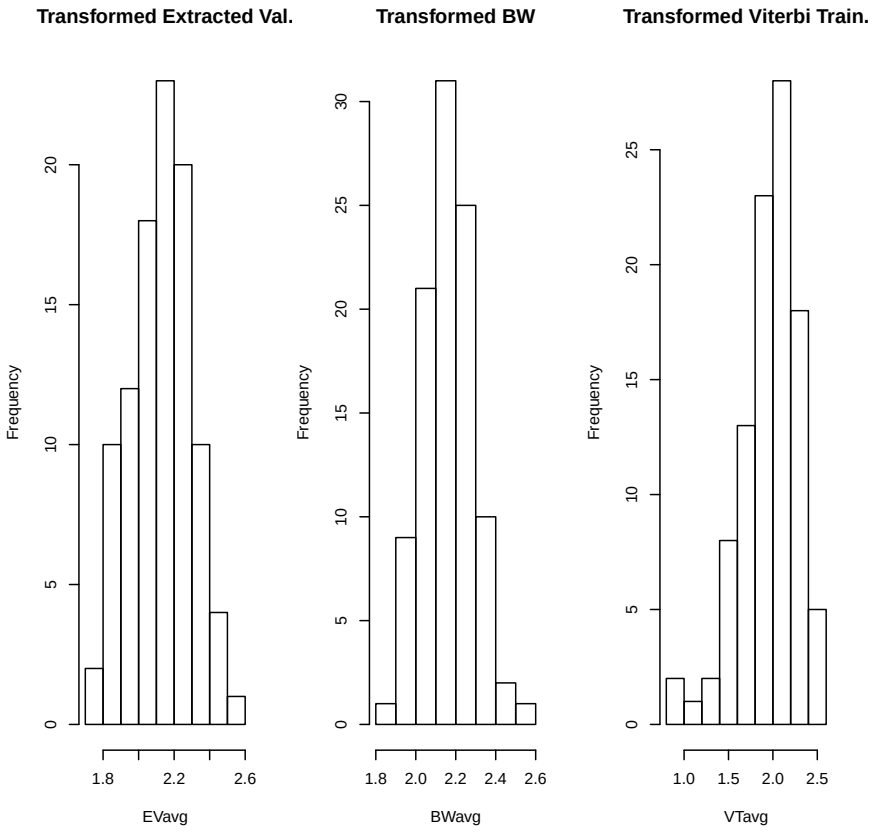

Figure 4.10: Transformed completely random data.

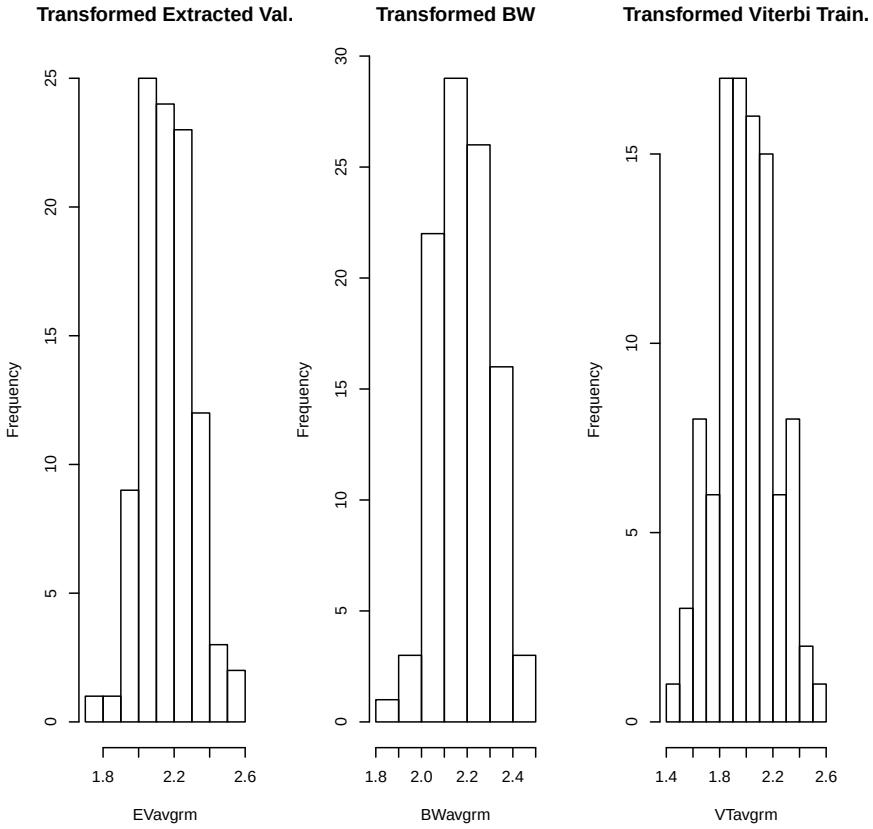

Figure 4.11: Transformed repeated measures data.

For the completely random sampling method, a Kruskal-Wallis test was performed to compare the three algorithms' average results. The test showed that the average accuracies of the three algorithms were significantly different, with a $p-value = 1.944e-12$, $\alpha = .05$. Wilcox rank-sum tests were performed in place of multiple comparisons to assess individual differences between average accuracies of the algorithms, $\alpha = .01$ (Bagdonavicius, Kruopis, and Nikulin, 2011) [13]. The results are displayed in the table below:

Table 4.10: Wilcox Rank Sum Summary, Completely Random

Comparison	p-value	Confidence Interval
Ext. Value vs. Baum-Welch	0.07507	(-0.10070762, 0.01742877)
Ext. Value vs. Viterbi Training	0.00037	(0.03129856, 0.21743697)
Baum-Welch vs. Viterbi Training	0.0	(0.07679606, 0.25048071)

For the repeated measures sampling method, a Freidman's non-parametric test was performed to compare the three algorithms' average results. The test showed that the average accuracies of the three algorithms were significantly different, with a $p - value = 1.087e - 07$, $\alpha = .05$. Wilcox rank-sum tests were also performed to assess individual differences between average accuracies of the algorithms, $\alpha = .01$. The results are displayed in the table below:

Table 4.11: Wilcox Rank Sum Summary, Repeated Measures

Comparison	p-value	Confidence Interval
Ext. Value vs. Baum-Welch	0.2689	(-0.07086251, 0.02789050)
Ext. Value vs. Viterbi Training	0.0	(0.1042269, 0.2454892)
Baum-Welch vs. Viterbi Training	0.0	(0.1284062, 0.2645160)

A common result, the Viterbi training algorithm doesn't seem to perform as well on average as its counterpart, the Baum-Welch algorithm, or the extracted value algorithm. Perhaps more appropriate data analysis would reveal differing results, though analysis is difficult given the nature of the data. The deviation from normality for the Viterbi training algorithm in the completely random data set probably was corrected for using the previously mentioned transformation, although the heterogeneity of variances across algorithms complicated possible options for data analysis. The results could have been expected since, as mentioned in chapter 3, the Baum-Welch algorithm tends to outperform the Viterbi training algorithm.

Chapter 5

Discussion and Conclusion

5.1 Non-Unique Most-Likely Path

It is possible that given a sequence of observations, the underlying most probable path can be produced by multiple state-sequences. It is conceivable that at time points in a sequence that the most likely state associated with a given time can be a number of states with equal probability. For example, if all of the parameters of the HMM are set so that they are all equally likely (all transition probabilities are equal and all emission probabilities are equal), then at each time point any path chosen would be the most likely. Though the previous example is extreme, the point should be clear. Which state should be chosen at time i when a number of states are equally likely? Furthermore, how are the multiple state-sequences that are most likely recovered? The latter is the question of interest.

The importance of this question stems from the descrepenices in the behavior of a sequence that can arise between an estimated sequene and the actual sequene. It would be of best interest to use the most-likely state sequence, or sequences, that best fits the data. Given that there are multiple state-sequences that are equally most-likely, a number of the sequences may not demonstrate the correct behavior of the data being used. For example, in DNA sequences CpG islands occur in blocks and are reasonably sized in length. The transition from non-CpG to CpG, and vice-versa, is infrequent. A most probable state sequence that has frequent transitions in and out of CpG states, or no transitions into the CpG state would not be a good fitting estimation of the state sequence, even if most-likely.

Equally most likely state sequences can occur due to the occurrence of equal-valued pointers during the backtracking process of the Viterbi algorithm. In other words there

may be a point in time where the pointer associated with the previously selected state is equally likely to be pointing from multiple states. With respect to this thesis, this implies that at some point in time, either the CpG state or the non-CpG state is equally likely to be pointing to the previously selected state in the state-sequence. The issue that arises should this happen is that a decision must be made as to which state-pointer to select. The moment that this occurs, there are most likely state-sequences that are left behind. It is interesting that, though a number of most probable state-sequences exist, statistical software will only produce an individual most probable state-sequence, and the same sequence each time an individual set of data and parameters are used. Using Matlab's hmmgenerate() function to generate an observed and hidden state-sequence and estimating the most probable state sequence using the hmmviterbi() function will consistently produce the same state-sequence, when using the same sequences and parameters. This can be a large setback when a another most probable state-sequence, or sequences, is a better fit with the given set of data.

It is of interest to determine if a method exists that would determine all state paths that are most-likely, and if one doesn't, to development such a method. Let Π^* denote the set of state-sequences that are most-likely, given the observed sequence and the model. Then we can define Π^* s.t.:

$$\Pi^* = \{\pi^* : \pi^* = argmax_\pi P(x, \pi)\}.$$

If Π^* is possible to find, then the user can for themselves, based off of some criteria established by them, determine the sequence best fit to model the data that they are using.

A small example of a situation where this occurs can be demonstrated using the following scenario. Let there be two states, 1 and 2, and four possible emissions per state, A, C, G, T. Let the transition probabilities, $a_{11} = .5$, $a_{12} = .5$, $a_{21} = .6$, and $a_{22} = .4$, and the emission probabilities for A, C, G, T respectively for state 1 are .1, .2, .3, .4, and similarly for state 2 are .4, .05, .05, and .5. If using sequence "GTGAG", then there are $2^5 = 32$ total possible state-sequences. When the likelihood of all 32 state sequences are cpmputed,

two of the state sequences are most likely, which are, "21121" and "21221". Thus, the set of most likely state-sequences, $\Pi^* = \{21121, 21221\}$.

Of course, calculations used to find these state-sequences were done by hand and had the observed sequence been of even moderate length, calculation time wouldn't have been practical. The main foreseeable problem surrounding the development of such a method used to find Π^* is that as the length of the sequence grows, the number of possible most likely state-sequences increases exponentially, ultimately increasing the amount of computation time required. The reward though, is a set of sequences that can be selected according to the level of relevance, determined by previous standards, the researcher, etc.

A possible solution to the problem is, during the state selection process, whenever a number of most probable pointers equal each other, instead of choosing only one state, choose those that are equal and most likely. The problem that arises with this solution is that the number of pathways could grow out of control, creating hundreds of possible pathways that need to be accounted for. A potential solution is, instead of holding onto the entire list of state pathways, disregarding or storing the state-sequence prior to the point where multiple pointers were most probable, and continuing the process until finished with the sequence analysis. The state sequence prior to the "split in pathway" could be converted into a string and stored. By storing the strings in sequential order, the path would not get mixed with the others. Once finished with the sequence analysis, connecting all of the stored sequence pieces, like a puzzle, would reveal all possible pathways.

If using the previous example where $\Pi^* = \{21121, 21221\}$, the Viterbi algorithm would select, working backwards, state 1, followed by state 2. It is at this point that the pointer associated with state 2 is equally likely to be pointing from state 1 and state 2. The algorithm would then chop off the already visited states and store them. It would continue by considering the pointers for each individual state processing in the same manner as before. What the finished result would be, in this case, is three pieces of sequence, 21, 211, and 212. By reconstructing the sequences using iteration values and states as markers, the result would give 21121 and 21221. A longer example with many possible pathways would

give a better illustration, as more paths could be chopped off, stored, and give better detail of the idea.

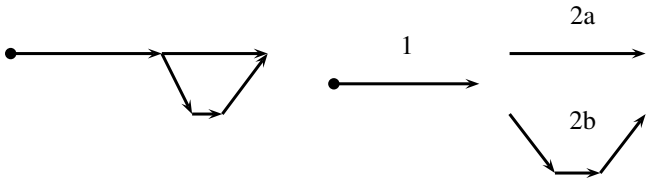

Figure 5.1: Visual representation of solving for Π^*.

Of course, this method is one potential solution and has not yet been implemented past the harmless inquiry stages and other ideas may arise that lead to a more efficient solution. Nevertheless, finding a solution to this problem seems useful and would serve as a new means of sequence selection, more flexible and useful for the user.

5.2 Conclusion

The results have shown a method that is effective at recovering the locations of CpG islands and a method of recovering the parameters of an HMM with respect to CpG island sequence data. The Viterbi algorithm has shown to be effective at locating CpG islands within DNA sequences. The Baum-Welch and Viterbi training algorithm have both shown to be capable of parameter estimation, although the Baum-Welch algorithm outperformed the Viterbi training algorithm in both methods of sequence sampling within this thesis' research. The difficulty of working with DNA sequences in hopes of trying to estimate CpG islands is in the sequence lengths. Computational methods can cut down analysis time and effort. The computation time for the single sequence Viterbi algorithm is seconds. If the reliability of estimation is convincing enough, the benefits of using such methods could be large.

This thesis works with computational methods of HMM's applied to CpG island esti-

mation in DNA sequences. It should be known though that HMM's can be used, and have potential uses, in further areas of biological sequence analysis. Coding and non-coding region analysis as well as secondary structure analysis are two areas that have uses for HMM's. The applicability of HMM's extends beyond biological sequence analysis.

Selecting a single most probable path from a set consisting of most probable paths is an example of advances that can be made within the study. Further research in the area of HMM's also extends beyond a frequentist approach into Bayesian analysis. As this area of statistics and probability is recently developed, the area is still finding new uses. The areas of study within HMM's are possibly numerous.

References

[1] Weissbach A. (1993), "A Chronicle of DNA Methylation (1948-1975)", *Exs*. 64, 1–10.

[2] Tost J.(2009), "DNA Methylation: An Introduction to the Biology and the Disease Associated Changes of a Promising Biomarker", *Methods of Molecular Biology*. 507, 3–20.

[3] Rice, W., Gaines, S (1989), "One-way analysis of variance with unequal variances" *Proceedings of the National Academy of Sciences*. 86, 8183–8143.

[4] Rabiner R.L. (1989), "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition" *IEEE*. 77, 257–286.

[5] Nephew P., Balch C., Skanlnik G. (2009), "Methyl Group Acceptance Assay for the Determination of Global DNA Methylation Levels", *Methods of Molecular Biology*. 507, 35–41.

[6] Ho Y, Weerhandi S. (2005), "Analysis of repeated measures under unequal variances", *Journal of Multivariate Analysis*. 98, 493-504.

[7] Durbin R. and Eddy S. and Mitchison G. (1998), *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge, New York, New York

[8] De Fonzo V. and Aluffini-Pentini F. and Parisi V. (2007), "Hidden Markov Models in Bioinformatics" *Current Bioinformatics*. 2, 140–145.

[9] Cappe O., Moulines E., Ryden T. (2005), *Inference in Hidden Markov Models*, Springer, New York, New York

[10] Brena M. and Plass C. (2009), "Bio-COBRA: Absolute Qualification of DNA Methylation in Electrofluidics Chips", *Methods of Molecular Biology*. 507, 257–269.

[11] Borodovsky, M. and Ekisheva, S. (2006), *Problems and Solutions in Biological Sequence Analysis*, Cambridge, New York, New York

[12] Baum, L. and Petrie, T. and Soules, G. and Normann, W. (1970), "A maximization technique occuring in the statistical analysis of probabilistic functions in Markov chains" *The Annals of Mathematical Statistics*. 41, 164–171.

[13] Bagdonavicius V. and Kruopis J. and Nikulin M. (2011), *Non-parametric Tests for Complete Data*, Wiley, Hoboken, NJ

Appendix A

Individual Sequence Viterbi Code (Java)

```
import java.io.*;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;


public class BHViterbi
{
   public static void main(String args[]) throws IOException
  {


//int nRep = 1;


//int length = 7411;
int numSeq = 1;


   FileInputStream fstream = new FileInputStream("Sequence.txt");
   DataInputStream in = new DataInputStream(fstream);
   BufferedReader br = new BufferedReader(new InputStreamReader(in));

       int nRep = 1;


       // Training Data Compilation

    String strLine = null;
    int ii = 0;
    String[] S = new String[100000];
    //Read File Line By Line

    while ((strLine = br.readLine()) != null)
    {
       S[ii] = strLine;
       ii++;
    }
```

```
String Snew = "";
int counter = 0;

       for(int k = 0; k < ii; k++)
       {
          //System.out.print("\n" + S[k]);
          Snew = Snew.concat(S[k]);
          counter = Snew.length();
          //System.out.print("\n" + counter);
       }


int i = counter;
char[] nullDNA = new char[ii];
       nullDNA = Snew.toCharArray();



//char[][][] symbol = new char[nRep][numSeq][counter];
char[][] DNA = new char[numSeq][counter];


int y7 = 0;
int y8 = 0;
int x9 = 0;
int multi = 0;


 while(y7 < numSeq)
  {
   y8 = 0;
   while(y8 < counter)
   {
        DNA[y7][y8] = nullDNA[x9];
        //System.out.print(y8 + " " + DNA[y7][y8]);
        x9++;
        y8++;
   }
        //System.out.print(" " + y7 + "\n");
   y7++;
 }


//********************************** R A N D O M N U M B E R R E A D

int nTrain = 1;

char[][][] DNA1 = new char[nRep][nTrain][counter];
int M0 = 0;
int M = 0;
int M2 = 0;
int M3 = 0;


while(M3 < nRep)
{
  M = 0;
```

```
while(M < nTrain)
{
   //M2 = randArray[M3][M];
   M0 = 0;
 while(M0 < counter)
 {
   DNA1[M3][M][M0] = DNA[M][M0];
        //System.out.print(M0 + " " + DNA1[M3][M][M0]);
   M0++;
 }
   M++;
}

        //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   //System.out.print("\n\n");
M3++;
}

/*
int trial = 97189;
while(trial < 97482)
{
 System.out.print(DNA1[0][0][trial]);
 trial++;
}
*/
// ******************** I D E N T I F Y C p G I S L A N D S



//******R E A D C p G I S L A N D L O C A T I O N


Scanner Location = new Scanner(new File("CpGLocation.txt"));

int[] B = new int[100];
int[] E = new int[100];

int v = 0;
double nA = 0;
double nC = 0;
double nG = 0;
double nT = 0;
double nA1 = 0;
double nC1 = 0;
double nG1 = 0;
double nT1 = 0;
int cAseq = 0;
int cCseq = 0;
int cGseq = 0;
int cTseq = 0;
double totCpG = 0;
double totNon = 0;

while(Location.hasNextInt())
{
```

```
   B[v] = Location.nextInt();
   E[v] = Location.nextInt();
   //System.out.print(B[v] + " " + E[v] + "\n");
   v++;
}

//System.out.print(counter);


int x4 = 0;
int CpGcount = 0;
char[] symbol = new char[counter];
int nCpG = 4;
int x5 = 0;

while(CpGcount < nCpG)
{
  x4 = B[CpGcount];
 while(x4 < counter)
 {
  if(x4 >= (B[CpGcount]-1) && x4 <= (E[CpGcount]-1))
  {
        x5 = x4;
        symbol[x5] = '+';

     //Count Nucleotides
        if(DNA1[0][0][x4] == 'A')
        {
               nA++;
               cAseq++;
        }
        else if(DNA1[0][0][x4] == 'C')
        {
               nC++;
               cCseq++;
        }
        else if(DNA1[0][0][x4] == 'G')
        {
               nG++;
               cGseq++;
        }
        else if(DNA1[0][0][x4] == 'T')
        {
               nT++;
               cTseq++;
        }

        //System.out.print(nA);
        totCpG++;
  }
/*
  else
  {
        symbol[x4] = '-';

        if(DNA1[0][0][x4] == 'A')
        {
```

```
                nA1++;
        }
        else if(DNA1[0][0][x4] == 'C')
        {
                nC1++;
        }
        else if(DNA1[0][0][x4] == 'G')
        {
                nG1++;
        }
        else if(DNA1[0][0][x4] == 'T')
        {
                nT1++;
        }

        totNon++;

  }
*/
//System.out.print(symbol);
  x4++;
 }
 CpGcount++;
}

int symbolCounter = 0;

while(symbolCounter < counter)
{
        if(symbol[symbolCounter] != '+')
        {

symbol[symbolCounter] = '-';
                totNon++;
        if(DNA1[0][0][symbolCounter] == 'A')
        {
                nA1++;
        }
        else if(DNA1[0][0][symbolCounter] == 'C')
        {
                nC1++;
        }
        else if(DNA1[0][0][symbolCounter] == 'G')
        {
                nG1++;
        }
        else if(DNA1[0][0][symbolCounter] == 'T')
        {
                nT1++;
        }
        }

        //System.out.print(symbol[symbolCounter]);
   symbolCounter++;
}
```

```
double a00 = (totCpG-nCpG)/totCpG;
double a01 = nCpG/totCpG;
double a10 = nCpG/totNon;
double a11 = (totNon-nCpG)/totNon;


double eCA = nA/totCpG;
double eCC = nC/totCpG;
double eCG = nG/totCpG;
double eCT = nT/totCpG;

double eNA = nA1/totNon;
double eNC = nC1/totNon;
double eNG = nG1/totNon;
double eNT = nT1/totNon;

//System.out.print("\n" + eCA + " " + eNA + "\n" + eCG + " " + eNG);

int transCounter = 0;
double c00 = 0;
double c01 = 0;
double c10 = 0;
double c11 = 0;

while(transCounter < (counter-1))
{
 if(symbol[transCounter] == '+' && symbol[transCounter + 1] == '+')
       c00++;
 if(symbol[transCounter] == '+' && symbol[transCounter + 1] == '-')
       c01++;
 if(symbol[transCounter] == '-' && symbol[transCounter + 1] == '+')
       c10++;
 if(symbol[transCounter] == '-' && symbol[transCounter + 1] == '-')
       c11++;
 transCounter++;
}

//System.out.print(c11 + " " + c10 + " " + c01 + " " + c00 + "\n");

double[][][] tState = new double[nRep][2][2];


int T3 = 0;
int T13 = 0;
int T23 = 0;

while(T3 < nRep)
{
 T13 = 0;
 while(T13 < 2)
 {
  T23 = 0;
  while(T23 < 2)
  {
   tState[T3][T13][T23] = pr(T13, T23, a00, a01, a10, a11);
   //System.out.print(tState[T3][T13][T23] + "\n");
  T23++;
```

```
  }
 T13++;
 }
T3++;
}
```

```
       double[][][][] eState22 = new double[nRep][nTrain][2][2000000];
       int g22 = 0;
       int r2 = 0;
       int g2 = 0;
       int T6 = 0;
       int T7 = 0;
```

```
while(T6 < nRep)
{
 T7 = 0;
 while(T7 < nTrain)
 {
  r2 = 0;
  while(r2<2)
  {
     g2 = 0;
     while(g2 < counter)
     {
        eState22[T6][T7][r2][g2] = emiss(DNA1[T6][T7][g2], r2, eCA, eCC, eCG, eCT, eNA, eNC, eNG, eNT);
        //System.out.print("\n" + T7 + " " + eState22[T6][T7][r2][g2] + DNA1[T6][T7][g2]);
        g2++;
     }
         //System.out.print("\n");
    r2++;
  }
  T7++;
 }
 T6++;
}
```

```
double[][][] vh = new double[nRep][nTrain][2000000];
double[][][] vl = new double[nRep][nTrain][2000000];
char[][][] state = new char[nRep][nTrain][2000000];
```

```
// ************** Using Log Transformation
```

```
//Log estates
```

```
       double[][][][] LogE = new double[nRep][nTrain][2][2000000];
```

```
        int rrr2 = 0;
        int rrr = 0;
        int gggg = 0;
        int rrr3 = 0;

      while(rrr2 < nRep)
      {
       rrr3 =0;
       while(rrr3 < nTrain)
       {
         rrr =0;
         while(rrr<2)
         {
            gggg = 0;
            while(gggg < counter)
            {

                 LogE[rrr2][rrr3][rrr][gggg] = Math.log(eState22[rrr2][rrr3][rrr][gggg]);
                 //System.out.print("\n" + gggg + " " + LogE[rrr2][rrr3][rrr][gggg]);
                 gggg++;
            }

           rrr++;
         }
           //System.out.print("\n");
         rrr3++;
         }
        rrr2++;
       }
```

```
//Log tState

double[][][] LogT = new double[nRep][2][2];
int ck = 0;
int ck1 = 0;
int ck2 = 0;
int ck3 = 0;
int ck4 = 0;

while(ck < nRep)
{
 ck4 = 0;
 while(ck1 < 2)
 {
   ck2 = 0;
  while(ck2 < 2)
   {
        LogT[ck][ck1][ck2] = Math.log(tState[ck][ck1][ck2]);
        //System.out.print("\n" + LogT[ck][ck1][ck2]);
        ck2++;
   }
   ck1++;
  }
```

```
 ck++;
}



double[] begin = new double[2];
   begin[0] = .5;
   begin[1] = .5;



//**** L O G B E G I N

double[] Logb = new double[2];
Logb[0] = Math.log(begin[0]);
Logb[1] = Math.log(begin[1]);



int Vit = 0;
int Vit2 = 0;
int vCount = 0;
char MAX = ' ';
double[] MAX0 = new double[2000000];
double[] MAX1 = new double[2000000];
double[][][] V0 = new double[nRep][nTrain][2000000];
double[][][] V1 = new double[nRep][nTrain][2000000];
double[][][] V0B = new double[nRep][nTrain][2000000];
double[][][] V1B = new double[nRep][nTrain][2000000];
char[][][] vPath = new char[nRep][nTrain][2000000];
double recursive1 = 0;
double recursive2 = 0;
double[][][] H = new double[nRep][nTrain][2000000];
double[][][] L = new double[nRep][nTrain][2000000];
double Vt00 = 0;
double Vt01 = 0;
double Vt10 = 0;
double Vt11 = 0;
char[][] ptr = new char[2][counter];


while(Vit < nRep)
{
 Vit2 = 0;
while(Vit2 < nTrain)
{
 vCount = 0;

 while(vCount < counter)
 {
   if(vCount == 0)
   {
     V0[Vit][Vit2][vCount] = LogE[Vit][Vit2][0][0] + Logb[0];
     V1[Vit][Vit2][vCount] = LogE[Vit][Vit2][1][0] + Logb[1];

        Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
```

```
    Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
    Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
    Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

    if(Vt00 >= Vt10)
    {
       MAX0[vCount] = Vt00;
       ptr[0][vCount] = '+';
    }
    else
    {
       MAX0[vCount] = Vt10;
       ptr[0][vCount] = '-';
    }

    if(Vt01 >= Vt11)
    {
       MAX1[vCount] = Vt01;
       ptr[1][vCount] = '+';
    }
    else
    {
       MAX1[vCount] = Vt11;
       ptr[1][vCount] = '-';
    }

          //System.out.print(DNA1[Vit][Vit2][vCount] + " " + LogE[Vit][Vit2][0][0] + " " + Logb[Vit2] + " " + V0[Vit][Vit2][vCount]);
          //System.out.print(LogT[Vit][0][0] + " " + V0[Vit][Vit2][vCount]);
          //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
}
else
{
    V0[Vit][Vit2][vCount] = VH1(LogE[Vit][Vit2][0][vCount], MAX0[vCount-1], vCount);
    V1[Vit][Vit2][vCount] = VL1(LogE[Vit][Vit2][1][vCount], MAX1[vCount-1], vCount);

    Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
    Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
    Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
    Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

    if(Vt00 >= Vt10)
    {
       MAX0[vCount] = Vt00;
       ptr[0][vCount] = '+';
    }
    else
    {
       MAX0[vCount] = Vt10;
       ptr[0][vCount] = '-';
    }

    if(Vt01 >= Vt11)
    {
       MAX1[vCount] = Vt01;
       ptr[1][vCount] = '+';
    }
    else
```

```
        {
           MAX1[vCount] = Vt11;
           ptr[1][vCount] = '-';
        }


        //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
        //System.out.print(LogT[Vit][0][vCount] + " " + V0[Vit][Vit2][vCount] + "\n");
   }
         //System.out.print(MAX0[vCount] + " " + MAX1[vCount] + "\n");
         //System.out.print(V0[Vit][Vit2][vCount]+ " " + V1[Vit][Vit2][vCount] + " " + DNA1[0][0][vCount] + "\n");
         //System.out.print(vCount + " " + ptr[0][vCount]+ " " + ptr[1][vCount] + "\n");

 vCount++;
 }
Vit2 ++;
//System.out.print("\n");
}
//System.out.print("\n");
 Vit++;
}


// ***************************** Select pi*

if(V0[0][0][counter-1] > V1[0][0][counter-1])
       MAX = '+';
else
       MAX = '-';

//System.out.print("MAX: " + MAX + " " + V0[0][0][counter-1] + " " + V1[0][0][length-1] + "\n");

char[] Path = new char[counter];

 int pathCount = (counter-1);



while(pathCount >= 0)
{
   if(pathCount == (counter-1))
   {
        Path[pathCount] = MAX;
   }
   else
   if(Path[pathCount+1] == '+')
   {
      Path[pathCount] = ptr[0][pathCount];
   }
   else
        Path[pathCount] = ptr[1][pathCount];
 pathCount--;
}
```

```
int pathView = 0;

while(pathView < counter)
{
   //System.out.print(Path[pathView] /*+ " " + MAXO[pathView] + "\n"*/);
   pathView++;
}



int stateCount = 0;
double nMatch = 0;

while(stateCount < counter)
{
   if(Path[stateCount] == symbol[stateCount])
   {
        nMatch++;
   }
 stateCount++;
}

System.out.print("Viterbi Accuracy: " + nMatch/counter);

/*
PrintWriter pw = new PrintWriter(new FileWriter("out.txt", true));
pw.println("20");
pw.print(nMatch/counter);
pw.println();
pw.close();
*/
/*
PrintWriter pw2 = new PrintWriter(new FileWriter("NucInfo.txt", true));
pw2.println("20");
pw2.println(counter);
pw2.println(" %A " + nA/totCpG + " %C " + nC/totCpG + " %G " + nG/totCpG + " %T " + nT/totCpG);
pw2.println(" %A " + nA1/totNon + " %C " + nC1/totNon + " %G " + nG1/totNon + " %T " + nT1/totNon);
pw2.println(nCpG);
pw2.println(totCpG/counter);
pw2.println();
pw2.close();
*/


// ************************** D E T E R M I N E V I T E R B I P R E D I C T E D C p G I S L A N D

PrintWriter pw2 = new PrintWriter(new FileWriter("VitCpG.txt", true));
pw2.println("1");
int CpGWriter = 1;

while(CpGWriter < counter)
{
  if(CpGWriter == 1 && Path[CpGWriter-1] == '+')
  {
    System.out.println("\nB: " + CpGWriter);
  }
  if(Path[CpGWriter-1] == '-' && Path[CpGWriter] == '+')
```

```
  {
    System.out.println("\nB: " + CpGWriter);
  }
  if(Path[CpGWriter-1] == '+' && Path[CpGWriter] == '-')
  {
    System.out.println("\nE: " + CpGWriter);
  }
  if(CpGWriter == (counter-1) && Path[CpGWriter] == '+')
  {
    System.out.println("\nB: " + CpGWriter);
  }
 CpGWriter++;
}


//************** E N D M A I N

 }



//******************************** T R A N S I T I O N


public static double pr(int T2, int T3, double a00, double a01, double a10, double a11)
{
       double tState = 0;

       if(T2 == 0 && T3 == 0)
          tState = a00;
       else if(T2 == 0 && T3 == 1)
          tState = a10;
       else if(T2 == 1 && T3 == 0)
          tState = a01;
       else if(T2 == 1 && T3 == 1)
          tState = a11;

       return (tState);
}


//*************************** E M I S S I O N


public static double emiss(char nullDNA1, int g, double A0, double C0, double G0, double T0, double A1, double C1, double G1, double T1)
{
              double emST2 = 0;


                        if(nullDNA1 == 'A')
                        {
                            if(g==0)
                            emST2= A0;
                            else
                            emST2 = A1;
                        }
                        else if(nullDNA1 == 'C')
```

```
                        {
                            if(g==0)
                            emST2 = C0;
                            else
                            emST2 = C1;
                        }
                        else if(nullDNA1 == 'G')
                        {
                            if(g==0)
                            emST2 = G0;
                            else
                            emST2 = G1;
                        }
                        else if(nullDNA1 == 'T')
                        {
                            if(g==0)
                            emST2 = T0;
                            else
                            emST2 = T1;
                        }



       return (emST2);
}




public static double emiss2(char predDNA, double emcA, double emcC, double emcG, double emcT, double emnA, double emnC, double emnG, double
      emnT, int g)
{
              double emST2 = 0;


                        if(predDNA == 'A')
                        {
                            if(g==0)
                            emST2= emcA;
                            else
                            emST2 = emnA;
                        }
                        else if(predDNA == 'C')
                        {
                            if(g==0)
                            emST2 = emcC;
                            else
                            emST2 = emnC;
                        }
                        else if(predDNA == 'G')
                        {
                            if(g==0)
                            emST2 = emcG;
                            else
                            emST2 = emnG;
```

```
                        }
                        else if(predDNA == 'T')
                        {
                             if(g==0)
                             emST2 = emcT;
                             else
                             emST2 = emnT;
                        }



       return (emST2);
}


//************************ V I T E R B I A L G O R I T H M

public static double VH1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VL1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VH2(double E, double Vl, double T, int i)
{
   double q = E + Vl + T;
   return q;
}
public static double VL2(double E, double Vh, double T, int i)
{
   double q = E + Vh + T;
   return q;
}
}
```

Appendix B

Train/Predict Viterbi Code Using Extracted Values (Java)

```
import java.io.*;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;


public class ExtractedValue
{
   public static void main(String args[]) throws IOException
  {

//************************** F I R S T U S E T H I S 'R' C O D E T O R A N D O M L Y S E L E C T T R A I N I N G S E Q U N C E S

/*


# * * * * * * * * * * S A M P L E
W1 = 0:19
sW1 = sample(W1,10)
   #sort(sW1)
W2 = W1[-(sW1+1)]
   #sort(W2)
testVal = sample(W2,1)

# * * * * * * * * * * W R I T E T O F I L E

write.infile(W2, "C:\\Users\\Robert\\Desktop\\School\\HMM\\Rprint.txt")
write.infile(testVal, "C:\\Users\\Robert\\Desktop\\School\\HMM\\trialSeqNUMBER.txt")

*/



   Scanner scan = new Scanner(System.in);

// *********** I N F O F O R T R A I N I N G S E Q U E N C E S **********


   FileInputStream fstream = new FileInputStream("oNuc2.txt");
   DataInputStream in = new DataInputStream(fstream);
```

```
  BufferedReader br = new BufferedReader(new InputStreamReader(in));

       int nRep = 5;


       // Training Data Compilation

    String strLine = null;
    int ii = 0;
    String[] S = new String[100000];
    //Read File Line By Line

    while ((strLine = br.readLine()) != null)
    {
       S[ii] = strLine;
       ii++;
    }


String Snew = "";
int counter = 0;

       for(int k = 0; k < ii; k++)
       {
          //System.out.print("\n" + S[k]);
          Snew = Snew.concat(S[k]);
          counter = Snew.length();
          //System.out.print("\n" + Snew);
       }


int i = counter;
char[] nullDNA = new char[ii];
       nullDNA = Snew.toCharArray();


  /* System.out.print("\nHow many training sequences will be used?\n");
       int nTrain = scan.nextInt();
  */

int nTrain = 10;


int numSeq = 20;
char[][] DNA = new char[numSeq][i];
int y3 = 0;
int y4 = 0;
int[] nCpG = new int[numSeq];




// ******************** I D E N T I F Y C p G I S L A N D S


Scanner numCPG = new Scanner(new File("numCPG.txt"));
```

```
int[][] Bloc = new int[numSeq][i];
int[][] Eloc = new int[numSeq][i];


while(numCPG.hasNextInt())
{
   nCpG[y3] = numCPG.nextInt();
   //System.out.print("N" + y3 + " " + nCpG[y3]);
   y3++;
}



//******R E A D C p G I S L A N D L O C A T I O N


Scanner Location = new Scanner(new File("Loc.txt"));

int[] B = new int[100];
int[] E = new int[100];

int v = 0;

while(Location.hasNextInt())
{

   B[v] = Location.nextInt();
   E[v] = Location.nextInt();
   //System.out.print(B[v] + " " + E[v] + "\n");
   v++;
}



int v1 = 0;
int v2 = 0;


 while(y4 < numSeq)
 {
  v1 = 0;
  while(v1 < nCpG[y4])
  {
    Bloc[y4][v1] = B[v2];
    Eloc[y4][v1] = E[v2];
    //System.out.print("\n" + y4 + " " + Bloc[y4][v1] + " " + Eloc[y4][v1]);
    v1++;
    v2++;
  }
 y4++;
}



// ************************* D E T E R M I N E L E N G T H O F S E Q U E N C E S
```

```
int y5 = 0;
int y6 = 0;

char[][] sCpG = new char[numSeq][i];

int[] lengthNull = new int[i];
int[] length = new int[numSeq];
int[] iLength = new int[numSeq]; // postion at which individual sequence ends
int z = 0;


while(y5 < numSeq)
{
   y6 = 0;
   while(y6 < i)
   {
      if(nullDNA[y6] == 'X')
      {
        length[y5] = lengthNull[y5];
        iLength[y5] = y6;
        //System.out.print("\nLength" + y5 + " " + length[y5]);
        y5++;
      }
      else
        lengthNull[y5]++;
        y6++;
    }
   y5++;
}



//System.out.print(length[0] + " " + length[1]);




//********************************** R A N D O M N U M B E R R E A D


Scanner Random = new Scanner(new File("Rprint.txt"));


Random rand = new Random();
int rCount = 0;
int rCheck = 0;
int[][] randArray = new int[nRep][nTrain];

while(rCheck < nRep)
{
   rCount = 0;
 while(rCount < nTrain)
 {
   randArray[rCheck][rCount] = Random.nextInt();
```

```
   //System.out.print(randArray[rCheck][rCount] + " ");
   rCount++;
 }
       //System.out.print("\n");
 rCheck++;
}



// ******************************** B U I L D S E Q U E N C E


//FileWriter outFile = new FileWriter("SeqConfirm.txt");
PrintWriter out1 = new PrintWriter("SeqConfirm.txt");


char[][][] symbol = new char[nRep][numSeq][i];

int y7 = 0;
int y8 = 0;
int x9 = 0;
int multi = 0;


  while(y7 < numSeq)
  {
    x9 = 0;
   while(y8 < iLength[y7])
   {
        if(nullDNA[y8+1] == 'X')
        {
          DNA[y7][x9] = nullDNA[y8];
          y8 = (y8+1);
        }
        else
        {
          DNA[y7][x9] = nullDNA[y8];
        }
        //System.out.print(DNA[y7][x9]);
        x9++;
        y8++;
   }
        //System.out.print("\n\n S  E      Q");
   y7++;
 }
//******************** B U I L D S E Q U E N C E B A S E D O F F O F R A N D O M A S S I G N M E N T


char[][][] DNA1 = new char[nRep][nTrain][i];
int M0 = 0;
int M = 0;
int M2 = 0;
int M3 = 0;

while(M3 < nRep)
{
  M = 0;
```

```
while(M < nTrain)
{
   M2 = randArray[M3][M];
   M0 = 0;
 while(M0 < length[M2])
 {
   DNA1[M3][M][M0] = DNA[M2][M0];
   //System.out.print(M + " " + DNA1[M3][M][M0]);
   M0++;
 }
   //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   //System.out.print("\n\n");
   M++;
}
M3++;
}
```

```
// ***************** A S S I G N S T A T E & C O U N T c p G / N O N
```

```
int A = 0;
int C = 0;
int G = 0;
int T = 0;

int x2 = 0;
int x3 = 0;
int x4 = 0;
int x5 = 0;
int[] totCpG = new int[100];
int[] totNon = new int[100];

double[] nA = new double[100];
int[][] cAseq = new int[100][nTrain];
double[] nC = new double[100];
int[][] cCseq = new int[100][nTrain];
double[] nG = new double[100];
int[][] cGseq = new int[100][nTrain];
double[] nT = new double[100];
int[][] cTseq = new int[100][nTrain];

double[] nA1 = new double[100];
int[][] nAseq = new int[100][nTrain];
double[] nC1 = new double[100];
int[][] nCseq = new int[100][nTrain];
double[] nG1 = new double[100];
int[][] nGseq = new int[100][nTrain];
double[] nT1 = new double[100];
int[][] nTseq = new int[100][nTrain];


while(C < nRep)
```

```
{
 x2 = 0;
while(x2 < nTrain)
{
  x3 = 0;
  while(x3 < nCpG[randArray[C][x2]])
  {
    x4 = 0;
        //System.out.print("\nLength" + " " + length[x2]);
    while(x4 < length[randArray[C][x2]])
    {
      if(x4 >= (Bloc[randArray[C][x2]][x3]-1) && x4 <= (Eloc[randArray[C][x2]][x3]-1))
      {
        symbol[C][x2][x4] = '+';

        //Count Nucleotides
        if(DNA1[C][x2][x4] == 'A')
        {
                nA[C]++;
                cAseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'C')
        {
                nC[C]++;
                cCseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'G')
        {
                nG[C]++;
                cGseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'T')
        {
                nT[C]++;
                cTseq[C][x2]++;
        }
        totCpG[C]++;
        x5++;
        //System.out.print(tCpG);
      }
      else
        {
          symbol[C][x2][x4] = '-';

        //Count Nucleotides
        if(DNA1[C][x2][x4] == 'A')
        {
                nA1[C]++;
                nAseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'C')
        {
                nC1[C]++;
                nCseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'G')
        {
```

```
                nG1[C]++;
                nGseq[C][x2]++;
        }
        else if(DNA1[C][x2][x4] == 'T')
        {
                nT1[C]++;
                nTseq[C][x2]++;
        }

          totNon[C]++;
          x5++;
        }
      //System.out.print(symbol[C][x2][x4]);
     x4++;
     }
         //System.out.print("\n");
   x3++;
   }
         //System.out.print("\n\nCpG: " + nA[C]/totCpG[C] + " " + nC[C]/totCpG[C] + " " + nG[C]/totCpG[C] + " " + nT[C]);
         //System.out.print("\nNon: " + nA1[C] + " " + nC1[C] + " " + nG1[C] + " " + nT1[C] + "\n");
  x2++;
}
        System.out.print("\n\nCpG: " + (nA[C] + nA1[C])/(totCpG[C]+totNon[C]) + " " + (nC[C] + nC1[C])/(totCpG[C]+totNon[C]) + " " + (nG[C] +
              nG1[C])/(totCpG[C]+totNon[C]) + " " + (nT[C] + nT1[C])/(totCpG[C]+totNon[C]));
        System.out.print("\nNon: " + nA1[C]/totNon[C] + " " + nC1[C]/totNon[C] + " " + nG1[C]/totNon[C] + " " + nT1[C]/totNon[C] + "\n");
        System.out.print("\n\n\n");
C++;
}



// ************** K N O W N S T A T E E M I S S I O N P R O B A B I L I T I E S


double[] ecA = new double[100];
double[] ecC = new double[100];
double[] ecG = new double[100];
double[] ecT = new double[100];

double[] enA = new double[100];
double[] enC = new double[100];
double[] enG = new double[100];
double[] enT = new double[100];

int E1 = 0;
ExtractedValue util2 = new ExtractedValue();
String file2 = "Results.txt";
boolean append2 = true;


while(E1 < nRep)
{
  ecA[E1] = nA[E1] / (nA[E1] + nC[E1] + nG[E1] + nT[E1]);
  ecC[E1] = nC[E1] / (nA[E1] + nC[E1] + nG[E1] + nT[E1]);
  ecG[E1] = nG[E1] / (nA[E1] + nC[E1] + nG[E1] + nT[E1]);
  ecT[E1] = nT[E1] / (nA[E1] + nC[E1] + nG[E1] + nT[E1]);
```

```
//System.out.print("\nCpG Emission Probabilities\n" + ecA[E1] + "\n" + ecC[E1] + "\n" + ecG[E1] + "\n" + ecT[E1] + "\n");

  enA[E1] = nA1[E1] / (nA1[E1] + nC1[E1] + nG1[E1] + nT1[E1]);
  enC[E1] = nC1[E1] / (nA1[E1] + nC1[E1] + nG1[E1] + nT1[E1]);
  enG[E1] = nG1[E1] / (nA1[E1] + nC1[E1] + nG1[E1] + nT1[E1]);
  enT[E1] = nT1[E1] / (nA1[E1] + nC1[E1] + nG1[E1] + nT1[E1]);
//System.out.print("\nNon Emission Probabilities\n" + enA[E1] + "\n" + enC[E1] + "\n" + enG[E1] + "\n" + enT[E1] + "\n");
util2.writeLinesToFileE(file2, append2,ecA[E1],ecC[E1],ecG[E1],ecT[E1],enA[E1],enC[E1],enG[E1],enT[E1]);
E1++;
}




// *************** K N O W N S T A T E T R A N S I T I O N P R O B A B I L I T I E S



int T1 = 0;
int x6 = 0;
int tCount = 0;
double[] A11 = new double[100];
double[] A10 = new double[100];
double[] A01 = new double[100];
double[] A00 = new double[100];


while(T1 < nRep)
{
 x6 = 0;
 while(x6 < nTrain)
 {

       //System.out.print("\n" + length[randArray[T1][x6]]);
   tCount = 0;
  while(tCount < length[randArray[T1][x6]])
  {
       //System.out.print(symbol[T1][x6][tCount]);

       if(symbol[T1][x6][tCount] == '-' && symbol[T1][x6][(tCount+1)] == '-')
       {
              A11[T1]++;
       }
       else if(symbol[T1][x6][tCount] == '-' && symbol[T1][x6][(tCount+1)] == '+')
       {
              A10[T1]++;
       }
       else if(symbol[T1][x6][tCount] == '+' && symbol[T1][x6][(tCount+1)] == '-')
       {
              A01[T1]++;
       }
       else if(symbol[T1][x6][tCount] == '+' && symbol[T1][x6][(tCount+1)] == '+')
       {
              A00[T1]++;
       }
   tCount++;
```

```
  }
  x6++;
 }

    //System.out.print("\n" + T1 + " " + A11[T1] + " " + A10[T1] + " " + A01[T1] + " " + A00[T1]);
 T1++;
}




double[] T11 = new double[nRep];
double[] T10 = new double[nRep];
double[] T01 = new double[nRep];
double[] T00 = new double[nRep];

ExtractedValue util3 = new ExtractedValue();
String file3 = "ResultsAV.txt";
boolean append3 = true;
int T2 = 0;

while(T2 < nRep)
{
  T11[T2] = A11[T2] / (A11[T2]+A10[T2]);
  T10[T2] = A10[T2] / (A11[T2]+A10[T2]);
  T01[T2] = A01[T2] / (A01[T2]+A00[T2]);
  T00[T2] = A00[T2] / (A01[T2]+A00[T2]);
 //System.out.print("\nTransition" + "\na11: " + T11[T2] + "\na10: " + T10[T2] + "\na01: " + T01[T2] + "\na00: " + T00[T2]);

util3.writeLinesToFileT(file3, append3,T00[T2], T01[T2], T10[T2], T11[T2]);

T2++;
}


double[][][] tState = new double[100][2][2];


int T3 = 0;
while(T3 < nRep)
{
   tState[T3] = pr(T11[T3], T10[T3], T01[T3], T00[T3]);
   //System.out.print(tState[T3][nRep][0] + " ");
   T3++;
}




double[] begin = new double[2];
   begin[0] = .01;
   begin[1] = .99;
```

```
//***************************** B E G I N S E Q U E N C E P R E D I C T I O N
//*********************** R E A D S E Q U E N C E F O R P R E D I C T I O N




//FileInputStream fstream2 = new FileInputStream("Test.txt");
   //DataInputStream in2 = new DataInputStream(fstream2);
   //BufferedReader br2 = new BufferedReader(new InputStreamReader(in2));



       int[][] trialSeqValue = new int[nRep][nTrain];
       Scanner t = new Scanner(new File("trialSeqNUMBER.txt"));
       int T4 = 0;
       int T14 = 0;


while(T4 < nRep)
{
 T14 =0;
 while(T14 < nTrain)
 {
       trialSeqValue[T4][T14] = t.nextInt();
       //System.out.print("\n"+trialSeqValue[T4][T14]);
       T14++;
 }
T4++;
}



//********************************************** Prediction Sequence Assemble

char[][][] predDNA = new char[nRep][nTrain][200000];
int pred1 = 0;
int pred2 = 0;
int pred3 = 0;


while(pred1 < nRep)
{
 pred3 = 0;
 while(pred3 < nTrain)
 {
  pred2 = 0;
  while(pred2 < length[trialSeqValue[pred1][pred3]])
  {
     predDNA[pred1][pred3][pred2] = DNA[trialSeqValue[pred1][pred3]][pred2];
     //System.out.print(pred3 + ":" + predDNA[pred1][pred3][pred2]);
     pred2++;
     //System.out.print("\n\n");
  }
 pred3++;
 }
pred1++;
}
```

```
int Z = 0;
int ZZ = 0;
int ZZZ = 0;
int ZZZZ = 0;

double[][][][] bwEState = new double[nRep][nTrain][2][200000];



while(Z < nRep)
{
 ZZ = 0;
 while(ZZ < nTrain)
 {
  ZZZ = 0;
  while(ZZZ<2)
  {
     ZZZZ = 0;
     while(ZZZZ < length[trialSeqValue[Z][ZZ]])
     {
        bwEState[Z][ZZ][ZZZ][ZZZZ] = emiss2(predDNA[Z][ZZ][ZZZZ], ecA[Z], ecC[Z], ecG[Z], ecT[Z], enA[Z], enC[Z], enG[Z], enT[Z], ZZZ);
        ZZZZ++;
     }
         //System.out.print("\n");
    ZZZ++;
  }
  ZZ++;
 }
 Z++;
}



int[][][] Blocb = new int[nRep][nTrain][200000];
int[][][] Elocb = new int[nRep][nTrain][200000];
char[][][] symbolb = new char[nRep][nTrain][200000];
int[][] nb = new int[nRep][nTrain];



int n2b = 0;
int n3b = 0;

while(n2b < nRep)
{
 n3b = 0;
 while(n3b < nTrain)
 {
  nb[n2b][n3b] = nCpG[trialSeqValue[n2b][n3b]];
  //System.out.print("\n" + nb[n2b][n3b]);
  n3b++;
 }
n2b++;
}
```

```
int v3 = 0;
int n23b = 0;
int n24b = 0;

while(n23b < nRep)
{
 n24b = 0;
 while(n24b < nTrain)
 {
  v3 = 0;
  while(v3 < nCpG[trialSeqValue[n23b][n24b]])
  {
    Blocb[n23b][n24b][v3] = Bloc[trialSeqValue[n23b][n24b]][v3];
    Elocb[n23b][n24b][v3] = Eloc[trialSeqValue[n23b][n24b]][v3];
    //System.out.print("\n\n" + n24b + " " + Blocb[n23b][n24b][v3] + " " + Elocb[n23b][n24b][v3]);
    v3++;
  }
 n24b++;
 }
n23b++;
}


int a3 = 0;
int n4b = 0;
int a13 = 0;

while(n4b < nRep)
{
 a13 = 0;
while(a13 < nTrain)
{
 a3 = 0;
 while(a3 < nb[n4b][a13])
 {
   for(int K3 = Blocb[n4b][a13][a3]; K3 <= Elocb[n4b][a13][a3]; K3++)
   {
        symbolb[n4b][a13][K3] = '+';

   }
  a3++;
 }
a13++;
}
 n4b++;
}


int b = 0;
int n5b = 0;
int n6b = 0;
double[][] nonGC = new double[nRep][nTrain];
double nonCount = 0;
```

```
while(n5b < nRep)
{
 n6b = 0;
 while(n6b < nTrain)
 {
  b = 0;
  while(b<length[trialSeqValue[n5b][n6b]])
  {
   if(symbolb[n5b][n6b][b] != '+')
   {
     symbolb[n5b][n6b][b] = '-';
   }
  //System.out.print(symbolb[n5b][n6b][b]);
  b++;
  }
 n6b++;
  //System.out.print("\n\n");
 }
n5b++;
}



//************************************** V I T E R B I



double[][][] vh = new double[nRep][nTrain][200000];
double[][][] vl = new double[nRep][nTrain][200000];
char[][][] state = new char[nRep][nTrain][200000];



// ************** Using Log Transformation


//Log estates

       double[][][][] LogE = new double[nRep][nTrain][2][200000];
       int rrr2 = 0;
       int rrr = 0;
       int gggg = 0;
       int rrr3 = 0;

     while(rrr2 < nRep)
     {
      rrr3 =0;
      while(rrr3 < nTrain)
      {
        rrr =0;
        while(rrr<2)
        {
           gggg = 0;
           while(gggg < length[trialSeqValue[rrr2][rrr3]])
           {
```

```
            LogE[rrr2][rrr3][rrr][gggg] = Math.log(bwEState[rrr2][rrr3][rrr][gggg]);
            //System.out.print("\n" + LogE[rrr2][rrr3][rrr][gggg]);
            gggg++;
          }

         rrr++;
        }
       rrr3++;
       }
      rrr2++;
     }



//Log tState

double[][][] LogT = new double[nRep][2][2];
int ck = 0;
int ck1 = 0;
int ck2 = 0;
int ck3 = 0;
int ck4 = 0;

while(ck < nRep)
{
 ck4 = 0;
while(ck4 < nTrain)
{
 ck3 = 0;
while(ck3 < length[trialSeqValue[ck][ck4]])
{
 ck1 = 0;
 while(ck1 < 2)
 {
   ck2 = 0;
  while(ck2 < 2)
   {
       LogT[ck][ck1][ck2] = Math.log(tState[ck][ck1][ck2]);
       //System.out.print("\n" + LogT[ck1][ck2]);
       ck2++;
   }
    ck1++;
  }
ck3++;
}
ck4++;
}
 ck++;
}


//**** L O G B E G I N

double[] Logb = new double[2];
Logb[0] = Math.log(begin[0]);
Logb[1] = Math.log(begin[1]);
```

```
int Vit = 0;
int Vit2 = 0;
int vCount = 0;
char[][] MAX = new char[nRep][nTrain];
double[] MAX0 = new double[2000000];
double[] MAX1 = new double[2000000];
double[][][] V0 = new double[nRep][nTrain][2000000];
double[][][] V1 = new double[nRep][nTrain][2000000];
double[][][] V0B = new double[nRep][nTrain][2000000];
double[][][] V1B = new double[nRep][nTrain][2000000];
char[][][] vPath = new char[nRep][nTrain][2000000];
double recursive1 = 0;
double recursive2 = 0;
double[][][] H = new double[nRep][nTrain][2000000];
double[][][] L = new double[nRep][nTrain][2000000];
double Vt00 = 0;
double Vt01 = 0;
double Vt10 = 0;
double Vt11 = 0;
char[][][][] ptr = new char[2][nRep][nTrain][200000];



while(Vit < nRep)
{
 Vit2 = 0;
while(Vit2 < nTrain)
{
 vCount = 0;

 while(vCount < length[trialSeqValue[Vit][Vit2]])
 {
   if(vCount == 0)
   {
     V0[Vit][Vit2][vCount] = LogE[Vit][Vit2][0][0] + Logb[0];
     V1[Vit][Vit2][vCount] = LogE[Vit][Vit2][1][0] + Logb[1];

        Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
        Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
        Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
        Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

        if(Vt00 >= Vt10)
        {
           MAX0[vCount] = Vt00;
           ptr[0][Vit][Vit2][vCount] = '+';
        }
        else
        {
           MAX0[vCount] = Vt10;
           ptr[0][Vit][Vit2][vCount] = '-';
        }
```

```
      if(Vt01 >= Vt11)
      {
         MAX1[vCount] = Vt01;
         ptr[1][Vit][Vit2][vCount] = '+';
      }
      else
      {
         MAX1[vCount] = Vt11;
         ptr[1][Vit][Vit2][vCount] = '-';
      }

            //System.out.print(DNA1[Vit][Vit2][vCount] + " " + LogE[Vit][Vit2][0][0] + " " + Logb[Vit2] + " " + V0[Vit][Vit2][vCount]);
            //System.out.print(LogT[Vit][0][0] + " " + V0[Vit][Vit2][vCount]);
            //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
  }
  else
  {
      V0[Vit][Vit2][vCount] = VH1(LogE[Vit][Vit2][0][vCount], MAX0[vCount-1], vCount);
      V1[Vit][Vit2][vCount] = VL1(LogE[Vit][Vit2][1][vCount], MAX1[vCount-1], vCount);

      Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
      Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
      Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
      Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

      if(Vt00 >= Vt10)
      {
         MAX0[vCount] = Vt00;
         ptr[0][Vit][Vit2][vCount] = '+';
      }
      else
      {
         MAX0[vCount] = Vt10;
         ptr[0][Vit][Vit2][vCount] = '-';
      }

      if(Vt01 >= Vt11)
      {
         MAX1[vCount] = Vt01;
         ptr[1][Vit][Vit2][vCount] = '+';
      }
      else
      {
         MAX1[vCount] = Vt11;
         ptr[1][Vit][Vit2][vCount] = '-';
      }


      //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
      //System.out.print(LogT[Vit][0][vCount] + " " + V0[Vit][Vit2][vCount] + "\n");
  }
       //System.out.print(MAX0[vCount] + " " + MAX1[vCount] + "\n");
       //System.out.print(V0[Vit][Vit2][vCount]+ " " + V1[Vit][Vit2][vCount] + " " + DNA1[0][0][vCount] + "\n");
       //System.out.print(vCount + " " + ptr[0][vCount]+ " " + ptr[1][vCount] + "\n");

vCount++;
```

```
 }
Vit2 ++;
//System.out.print("\n");
}
//System.out.print("\n");
 Vit++;
}



// ***************************** Select pi*
int piCount1 = 0;
int piCount2 = 0;

while(piCount1 < nRep)
{
 piCount2 = 0;
 while(piCount2 < nTrain)
 {
  if(V0[piCount1][piCount2][length[trialSeqValue[piCount1][piCount2]]-1] > V1[piCount1][piCount2][length[trialSeqValue[piCount1][piCount2]]-1])
       MAX[piCount1][piCount2] = '+';
  else
       MAX[piCount1][piCount2] = '-';
   piCount2++;
 }
 piCount1++;
}



char[][][] Path = new char[nRep][nTrain][200000];

int pathCount1 = 0;
int pathCount2 = 0;
int pathCount3 = 0;


while(pathCount3 < nRep)
{
 pathCount2 = 0;
 while(pathCount2 < nTrain)
 {
   pathCount1 = length[trialSeqValue[pathCount3][pathCount2]]-1;
  while(pathCount1 >= 0)
  {
   if(pathCount1 == (length[trialSeqValue[pathCount3][pathCount2]]-1))
   {
        Path[pathCount3][pathCount2][pathCount1] = MAX[pathCount3][pathCount2];
   }
   else
   if(Path[pathCount3][pathCount2][pathCount1 + 1] == '+')
   {
      Path[pathCount3][pathCount2][pathCount1] = ptr[0][pathCount3][pathCount2][pathCount1];
   }
   else
        Path[pathCount3][pathCount2][pathCount1] = ptr[1][pathCount3][pathCount2][pathCount1];
 pathCount1--;
```

```
  }
 pathCount2++;
 }
 pathCount3++;
}



int pathView1 = 0;
int pathView2 = 0;
int pathView3 = 0;

 while(pathView1 < nRep)
 {
  pathView2 = 0;
   while(pathView2 < nTrain)
   {
    pathView3 = 0;
     while(pathView3 < length[trialSeqValue[pathView1][pathView2]])
     {
      //System.out.print(Path[pathView1][pathView2][pathView3] /*+ " " + MAXO[pathView] + "\n"*/);
      pathView3++;
     }
        //System.out.print("\n\n\n\n");
    pathView2++;
   }
   pathView1++;
  }



//************************************* V I T E R B I A C C U R A C Y


int b12 = 0;
int b13 = 0;
int b14 = 0;
double[][] nMatch2 = new double[nRep][nTrain];

while(b13 < nRep)
{
 b14 = 0;
while(b14 < nTrain)
{
 b12 = 0;
 while(b12 < length[trialSeqValue[b13][b14]])
 {
   if(Path[b13][b14][b12] == symbolb[b13][b14][b12])
   {
        nMatch2[b13][b14]++;
   }
  b12++;
  }
 b14++;
 }
 b13++;
}
```

```
double[][] pMatch = new double[nRep][nTrain];
int n7b = 0;
int n8b = 0;

double pSUM = 0;

while(n7b < nRep)
{
 n8b = 0;
 while(n8b < nTrain)
 {
  pMatch[n7b][n8b] = nMatch2[n7b][n8b]/length[trialSeqValue[n7b][n8b]];
  pSUM = pSUM + pMatch[n7b][n8b];
  //System.out.print("\n\nViterbi Accuracy: " + pMatch[n7b][n8b]);
 n8b++;
 }
n7b++;
}



int printCount1 = 0;
int printCount2 = 0;

while(printCount1 < nRep)
{
 printCount2 = 0;
 while(printCount2 < nTrain)
 {

  PrintWriter pw2 = new PrintWriter(new FileWriter("Results.txt", true));
  pw2.println(printCount1 + " - " + printCount2 + ": " + pMatch[printCount1][printCount2]);
  pw2.println();
  pw2.close();

 printCount2++;
 }
printCount1++;
}



// ************************************ E N D M A I N
}




//****************************** T R A N S I T I O N


public static double[][] pr(double t1, double t2, double t3, double t4)
{
       double[][] tState = new double [2][2];
```

```
          tState[0][0] = t1;
          tState[0][1] = t2;
          tState[1][0] = t3;
          tState[1][1] = t4;

       return (tState);
}



public static double emiss2(char predDNA, double emcA, double emcC, double emcG, double emcT, double emnA, double emnC, double emnG, double
      emnT, int g)
{
              double emST2 = 0;


                        if(predDNA == 'A')
                        {
                            if(g==0)
                            emST2= emcA;
                            else
                            emST2 = emnA;
                        }
                        else if(predDNA == 'C')
                        {
                            if(g==0)
                            emST2 = emcC;
                            else
                            emST2 = emnC;
                        }
                        else if(predDNA == 'G')
                        {
                            if(g==0)
                            emST2 = emcG;
                            else
                            emST2 = emnG;
                        }
                        else if(predDNA == 'T')
                        {
                            if(g==0)
                            emST2 = emcT;
                            else
                            emST2 = emnT;
                        }



       return (emST2);
}


//************************ V I T E R B I A L G O R I T H M

public static double VH1(double E, double MAX, int i)
{
   double q = E + MAX;
```

```
   return q;
}
public static double VL1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VH2(double E, double Vl, double T, int i)
{
   double q = E + Vl + T;
   return q;
}
public static double VL2(double E, double Vh, double T, int i)
{
   double q = E + Vh + T;
   return q;
}

//************************* P R I N T T O F I L E Transition


public void writeLinesToFileT(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Transition:\n" + "A00: " + gc1);
          pw.println("Transition:\n" + "A01: " + gc2);
          pw.println("Transition:\n" + "A10: " + gc3);
          pw.println("Transition:\n" + "A11: " + gc4);
          arrays++;
          pw.println(" ");
          pw.flush();
}


//************************* P R I N T T O F I L E Emission


public void writeLinesToFileE(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4, double gc5, double gc6,
      double gc7, double gc8) throws IOException
```

```
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

        //If the file already exists, start writing at the end of it.
        pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

        pw = new PrintWriter(new FileWriter(filename));
        //this is equal to:
        //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Emiss CG: " + gc1 + " " + gc2 + " " + gc3 + " " + gc4);
          pw.println("Emiss NON: " + gc5 + " " + gc6 + " " + gc7 + " " + gc8);
          arrays++;
          pw.println(" ");
          pw.flush();
}



//************************ P R I N T T O F I L E


public void writeLinesToFile(String filename, boolean appendToFile, double gc1, int i) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

        //If the file already exists, start writing at the end of it.
        pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

        pw = new PrintWriter(new FileWriter(filename));
        //this is equal to:
        //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println(i + " " + gc1);
          arrays++;
          pw.println(" ");
          pw.flush();
```

```
}

}
```

Appendix C

Baum-Welch Algorithm (Java)

```
import java.io.*;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;


public class BaumWelch
{
   public static void main(String args[]) throws IOException
  {



  Scanner scan = new Scanner(System.in);

// ********** I N F O F O R T R A I N I N G S E Q U E N C E S **********


   FileInputStream fstream = new FileInputStream("oNuc2.txt");
   DataInputStream in = new DataInputStream(fstream);
   BufferedReader br = new BufferedReader(new InputStreamReader(in));

       int nRep = 5;


       // Training Data Compilation

    String strLine = null;
    int ii = 0;
    String[] S = new String[100000];
    //Read File Line By Line

    while ((strLine = br.readLine()) != null)
    {
       S[ii] = strLine;
       ii++;
    }


String Snew = "";
int counter = 0;

       for(int k = 0; k < ii; k++)
```

```
       {
          //System.out.print("\n" + S[k]);
          Snew = Snew.concat(S[k]);
          counter = Snew.length();
          //System.out.print("\n" + Snew);
       }


int i = counter;
char[] nullDNA = new char[ii];
       nullDNA = Snew.toCharArray();


   /*System.out.print("\nHow many training sequences will be used?\n");
        int nTrain = scan.nextInt();
   */

int nTrain = 9;


int numSeq = 19;

char[][] DNA = new char[numSeq][i];



// ************************* D E T E R M I N E L E N G T H O F S E Q U E N C E S


int y5 = 0;
int y6 = 0;

char[][] sCpG = new char[numSeq][i];

int[] lengthNull = new int[i];
int[] length = new int[numSeq];
int[] iLength = new int[numSeq]; // postion at which individual sequence ends
int z = 0;


while(y5 < numSeq)
{
   y6 = 0;
   while(y6 < i)
   {
      if(nullDNA[y6] == 'X')
      {
        length[y5] = lengthNull[y5];
        iLength[y5] = y6;
        //System.out.print("\nLength" + y5 + " " + iLength[y5]);
        y5++;
      }
      else
        lengthNull[y5]++;
        y6++;
    }
   y5++;
```

```
}


//System.out.print(length[0] + " " + length[1]);



//********************************** R A N D O M N U M B E R R E A D


Scanner Random = new Scanner(new File("Rprint.txt"));


Random rand = new Random();
int rCount = 0;
int rCheck = 0;
int[][] randArray = new int[nRep][nTrain];

while(rCheck < nRep)
{
   rCount = 0;
 while(rCount < nTrain)
 {
   randArray[rCheck][rCount] = Random.nextInt();
   //System.out.print(randArray[rCheck][rCount] + " ");
   rCount++;
 }
       //System.out.print("\n");
 rCheck++;
}



// ******************************** B U I L D S E Q U E N C E


//FileWriter outFile = new FileWriter("SeqConfirm.txt");
PrintWriter out1 = new PrintWriter("SeqConfirm.txt");


char[][][] symbol = new char[nRep][numSeq][i];

int y7 = 0;
int y8 = 0;
int x9 = 0;
int multi = 0;


  while(y7 < numSeq)
  {
    x9 = 0;
   while(y8 < iLength[y7])
   {
       if(nullDNA[y8+1] == 'X')
       {
```

```
        DNA[y7][x9] = nullDNA[y8];
        y8 = (y8+1);
      }
      else
      {
        DNA[y7][x9] = nullDNA[y8];
      }
      //System.out.print(DNA[y7][x9]);
      x9++;
      y8++;
  }
      //System.out.print("\n\n S  E      Q");
  y7++;
 }



//******************** B U I L D S E Q U E N C E B A S E D O F F O F R A N D O M A S S I G N M E N T


char[][][] DNA1 = new char[nRep][nTrain][i];
int M0 = 0;
int M = 0;
int M2 = 0;
int M3 = 0;

while(M3 < nRep)
{
  M = 0;
while(M < nTrain)
{
   M2 = randArray[M3][M];
   M0 = 0;
 while(M0 < length[M2])
 {
   DNA1[M3][M][M0] = DNA[M2][M0];
      //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   M0++;
 }
   M++;
}

      //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   //System.out.print("\n\n");
M3++;
}




double[][][] tState = new double[nRep][2][2];


int T3 = 0;
int T13 = 0;
```

```
while(T3 < nRep)
{
   tState[T3] = pr();
   //System.out.print(tState[T3][1][0] + " ");
T3++;
}

       double[][][][] eState22 = new double[nRep][nTrain][2][200000];
       int g22 = 0;
       int r2 = 0;
       int g2 = 0;
       int T6 = 0;
       int T7 = 0;



while(T6 < nRep)
{
 T7 = 0;
 while(T7 < nTrain)
 {
  r2 = 0;
  while(r2<2)
  {
     g2 = 0;
     while(g2 < length[randArray[T6][T7]])
     {
        eState22[T6][T7][r2][g2] = emiss(DNA1[T6][T7][g2], r2);
          //System.out.print("\n" + T6 + " " + T7 + " " + r2 + " " + eState22[T6][T7][r2][g2]);
        g2++;
     }
         //System.out.print("\n");
    r2++;
  }
  T7++;
 }
 T6++;
}


double[] begin = new double[2];
   begin[0] = .5;
   begin[1] = .5;



//*****************************************************************************



int recur = 5;
int nRecur = 0;

double[] emcA = new double[nRep];
double[] emcC = new double[nRep];
double[] emcG = new double[nRep];
double[] emcT = new double[nRep];
```

```
double[] emnA = new double[nRep];
double[] emnC = new double[nRep];
double[] emnG = new double[nRep];
double[] emnT = new double[nRep];

double[][][] bwTSTATE = new double [nRep][2][2];
double[][][][] bwEState = new double[nRep][nTrain][2][200000];



while(nRecur < recur)
{


//****************************** F O R W A R D S C A L I N G


//System.out.print("\nSCALED FORWARD VARIABLE\n");

 double[][][] scale = new double[nRep][nTrain][200000];
 double[][][] fsH = new double[nRep][nTrain][200000];
 double[][][] fsL = new double[nRep][nTrain][200000];
 double scaleProd = 1;
 int F1 = 0;
 int ss = 0;
 int ss2 = 0;

while(F1 < nRep)
{
 ss2 = 0;
 while(ss2 < nTrain)
 {
       ss = 0;
 while(ss < length[randArray[F1][ss2]])
 {
   if(ss == 0)
  {
    scale[F1][ss2][ss] = Scale1(eState22, begin, F1, ss2);
    fsH[F1][ss2][ss] = fHS1(eState22, begin, tState, scale, ss, F1, ss2);
    fsL[F1][ss2][ss] = fLS1(eState22, begin, tState, scale, ss, F1, ss2);
  }
  else
  {
   scale[F1][ss2][ss] = Scale2(eState22, tState, fsH, fsL, ss, F1, ss2);
   fsH[F1][ss2][ss] = fHS2(eState22, begin, tState, fsH, fsL, scale, ss, F1, ss2);
   fsL[F1][ss2][ss] = fLS2(eState22, begin, tState, fsH, fsL, scale, ss, F1, ss2);
  }
       //System.out.print(F1 + ":" + ss + " " + fsH[F1][ss2][ss]+ "\n");
       ss++;
 }
  ss2++;
}
       //System.out.print("\n");
 F1++;
}
```

```
//************************************************************ B A C K W A R D A L G O R I T H M



double[][][] scaleB = new double[nRep][nTrain][200000];

       double[][][] bh = new double[nRep][nTrain][200000];
       double[][][] bl = new double[nRep][nTrain][200000];


int B1 = 0;
int B2 = 0;
int B3 = 0;
int B14 = 0;
double[][][][] bEmiss = new double[nRep][nTrain][2][200000];



while(B1 < nRep)
{
B14 = 0;
while(B14 < nTrain)
{
 B3 = 0;
 while(B3 < 2)
 {
  B2 = (length[randArray[B1][B14]])-1;
  while(B2 > 0)
  {

       if(B2 == (length[randArray[B1][B14]]-1))
              {
                 scaleB[B1][B14][B2] = ScaleB1(eState22, begin, B1, B14);
                 bh[B1][B14][B2] = bHS1(scaleB, B2, B1, B14);
                 bl[B1][B14][B2] = bLS1(scaleB, B2, B1, B14);
              }
              else
              {
                 scaleB[B1][B14][B2] = ScaleB2(eState22, tState, bh, bl, B2, B1, B14);
                 bh[B1][B14][B2] = bh(eState22, tState, bh, bl, B2, B1, scaleB, B14);
                 bl[B1][B14][B2] = bl(eState22, tState, bh, bl, B2, B1, scaleB, B14);
              }
       //System.out.print("\n" + B1 + " " + bh[B1][B14][B2] + " " + bl[B1][B14][B2]);
     B2--;
  }
       //System.out.print("\n\n");
 B3++;
}
B14++;
}
 B1++;
}
```

```
//********************************** B A U M - W E L C H


// ******** T R A N S I T I O N


//double[][][] a1 = new double[nRep][1][2];
//double[][][] a2 = new double[nRep][1][2];

double[][] a0 = new double[nRep][2];
double[][] a1 = new double[nRep][2];


int L1 = 0;
int L2 = 0;
int L3 = 0;
int L4 = 0;

double BW1 = 0;
double BW2 = 0;


int BWc = 0;
int BWC = 0;
double[] SumA0 = new double[nRep];
double[] SumA1 = new double[nRep];

//double[][][] bwTSTATE = new double [nRep][2][2];
//double[][][][] bwEState = new double[nRep][nTrain][2][200000];

BW3 util3 = new BW3();
String file3 = "Results.txt";
boolean append3 = true;


int convergence = 0;
int iteration = 5;
int Z1 = 0;
int ZZ1 = 0;
int ZZZ1 = 0;
int ZZZZ1 = 0;

//double[][][][] bwEState = new double[nRep][nTrain][2][200000];

while(L1 < nRep)
{
 L4 = 0;
while(L4 < nTrain)
{
  L2 = 0;
  BWc = 0;
while(L2 < 2)
 {
```

```
   L3 = 0;
  while(L3 < length[randArray[L1][L4]])
  {


        BW1 = BWtrans(fsH[L1][L4][L3], bh[L1][L4][L3+1], tState[L1][0][L2], eState22[L1][L4][L2][L3+1]);
        BW2 = BWtrans(fsL[L1][L4][L3], bl[L1][L4][L3+1], tState[L1][1][L2], eState22[L1][L4][L2][L3+1]);


       //a1[L1][0][L2] = BW1;

          a0[L1][L2] = a0[L1][L2] + BW1;
          a1[L1][L2] = a1[L1][L2] + BW2;

       //a2[L1][0][L2] = BW2;


   L3++;
   }
       //System.out.print("\n" + a0[L1][L2]/length[randArray[L1][L4]] + " " + a1[L1][L2]/length[randArray[L1][L4]]);
        //SumA0[L1] = SumA0[L1] + a0[L1][L1];
        //SumA1[L1] = SumA1[L1] + a1[L1][L1];

  L2++;
  }
/*
         System.out.print("\nA00: " + a0[L1][0]/SumA0[L1]);
         System.out.print("\nA01: " + a0[L1][1]/SumA0[L1]);
         System.out.print("\nA10: " + a1[L1][0]/SumA1[L1]);
         System.out.print("\nA11: " + a1[L1][1]/SumA1[L1]);
*/
       //System.out.print("\n\n---------");
 L4++;
 }
L1++;
}




while(BWC < nRep)
{
 BWc = 0;
while(BWc < 2)
{
   SumA0[BWC] = SumA0[BWC] + a0[BWC][BWc];
   SumA1[BWC] = SumA1[BWC] + a1[BWC][BWc];
  BWc++;
}

  System.out.print("\nA00: " + a0[BWC][0]/SumA0[BWC]);
  System.out.print("\nA01: " + a0[BWC][1]/SumA0[BWC]);
  System.out.print("\nA10: " + a1[BWC][0]/SumA1[BWC]);
  System.out.print("\nA11: " + a1[BWC][1]/SumA1[BWC]);

util3.writeLinesToFileT(file3, append3,a0[BWC][0]/SumA0[BWC], a0[BWC][1]/SumA0[BWC], a1[BWC][0]/SumA1[BWC], a1[BWC][1]/SumA1[BWC]);
```

```
  bwTSTATE[BWC][0][0] = a0[BWC][0]/SumA0[BWC];
  bwTSTATE[BWC][0][1] = a0[BWC][1]/SumA0[BWC];
  bwTSTATE[BWC][1][0] = a1[BWC][0]/SumA1[BWC];
  bwTSTATE[BWC][1][1] = a1[BWC][1]/SumA1[BWC];


BWC++;
//System.out.print("\n\n");
}




//********** E M I S S I O N


double[] EAh = new double[nRep];
 double[] ehA = new double[nRep];
double[] ECh = new double[nRep];
 double[] ehC = new double[nRep];
double[] EGh = new double[nRep];
 double[] ehG = new double[nRep];
double[] ETh = new double[nRep];
 double[] ehT = new double[nRep];

double[] EAl = new double[nRep];
 double[] elA = new double[nRep];
double[] ECl = new double[nRep];
 double[] elC = new double[nRep];
double[] EGl = new double[nRep];
 double[] elG = new double[nRep];
double[] ETl = new double[nRep];
 double[] elT = new double[nRep];

int L11 = 0;
int L12 = 0;
int L13 = 0;
int L14 = 0;

while(L11 < nRep)
{
 L14 = 0;
 while(L14 < nTrain)
 {
  L12 = 0;
  while(L12 < 2)
  {
   L13 = 0;
   while(L13 < length[randArray[L11][L14]])
   {

    if(L12 == 0)
    {
        if(DNA1[L11][L14][L13] == 'A')
        {
           EAh[L11] = BWemiss(fsH[L11][L14][L13], bh[L11][L14][L13], scaleB[L11][L14][L13]);
                ehA[L11] = ehA[L11] + EAh[L11];
```

```
        }
        else if(DNA1[L11][L14][L13] == 'C')
        {
           ECh[L11] = BWemiss(fsH[L11][L14][L13], bh[L11][L14][L13], scaleB[L11][L14][L13]);
                ehC[L11] = ehC[L11] + ECh[L11];
        }
        else if(DNA1[L11][L14][L13] == 'G')
        {
           EGh[L11] = BWemiss(fsH[L11][L14][L13], bh[L11][L14][L13], scaleB[L11][L14][L13]);
                ehG[L11] = ehG[L11] + EGh[L11];
        }
        else if(DNA1[L11][L14][L13] == 'T')
        {
           ETh[L11] = BWemiss(fsH[L11][L14][L13], bh[L11][L14][L13], scaleB[L11][L14][L13]);
                ehT[L11] = ehT[L11] + ETh[L11];
        }
    }

    if(L12 == 1)
    {
        if(DNA1[L11][L14][L13] == 'A')
        {
          EAl[L11] = BWemiss(fsL[L11][L14][L13], bl[L11][L14][L13], scaleB[L11][L14][L13]);
                elA[L11] = elA[L11] + EAl[L11];
        }
        else if(DNA1[L11][L14][L13] == 'C')
        {
          ECl[L11] = BWemiss(fsL[L11][L14][L13], bl[L11][L14][L13], scaleB[L11][L14][L13]);
                elC[L11] = elC[L11] + ECl[L11];
        }
        else if(DNA1[L11][L14][L13] == 'G')
        {
          EGl[L11] = BWemiss(fsL[L11][L14][L13], bl[L11][L14][L13], scaleB[L11][L14][L13]);
                elG[L11] = elG[L11] + EGl[L11];
        }
        else if(DNA1[L11][L14][L13] == 'T')
        {
          ETl[L11] = BWemiss(fsL[L11][L14][L13], bl[L11][L14][L13], scaleB[L11][L14][L13]);
                elT[L11] = elT[L11] + ETl[L11];
        }
     }



   L13++;
   }
        //System.out.print("\n" + elA + " " + elC + " " + elG + " " + elT);
  L12++;
  }
        //System.out.print("\n\n---------");
 L14++;
 }
L11++;
}
```

```
BW3 util2 = new BW3();

int BWe = 0;
String file2 = "Results.txt";
boolean append2 = true;

while(BWe < nRep)
{
 emcA[BWe] = ehA[BWe] / (ehA[BWe] + ehC[BWe] + ehG[BWe] + ehT[BWe]);
 emcC[BWe] = ehC[BWe] / (ehA[BWe] + ehC[BWe] + ehG[BWe] + ehT[BWe]);
 emcG[BWe] = ehG[BWe] / (ehA[BWe] + ehC[BWe] + ehG[BWe] + ehT[BWe]);
 emcT[BWe] = ehT[BWe] / (ehA[BWe] + ehC[BWe] + ehG[BWe] + ehT[BWe]);

 emnA[BWe] = elA[BWe] / (elA[BWe] + elC[BWe] + elG[BWe] + elT[BWe]);
 emnC[BWe] = elC[BWe] / (elA[BWe] + elC[BWe] + elG[BWe] + elT[BWe]);
 emnG[BWe] = elG[BWe] / (elA[BWe] + elC[BWe] + elG[BWe] + elT[BWe]);
 emnT[BWe] = elT[BWe] / (elA[BWe] + elC[BWe] + elG[BWe] + elT[BWe]);

util2.writeLinesToFileE(file2, append2,emcA[BWe],emcC[BWe],emcG[BWe],emcT[BWe],emnA[BWe],emnC[BWe],emnG[BWe],emnT[BWe]);
//System.out.print("\n" + emcA[BWe] + " " + emcC[BWe] + " " + emcG[BWe] + " " + emcT[BWe]);
//System.out.print("\n" + emnA[BWe] + " " + emnC[BWe] + " " + emnG[BWe] + " " + emnT[BWe]);

BWe++;

//System.out.print("\n\n");
}


int L1b = 0;
int L4b = 0;
int L2b = 0;
int L3b = 0;

double[][] a0b = new double[nRep][2];
double[][] a1b = new double[nRep][2];



//*************************************************************************************************************************************



int T3b = 0;
int T13b = 0;

while(T3b < nRep)
{
   tState[T3b] = pr2(bwTSTATE[T3b][0][0], bwTSTATE[T3b][0][1], bwTSTATE[T3b][1][0], bwTSTATE[T3b][1][1]);
   //System.out.print(tState[T3b][1][0] + " ");
T3b++;
}

       int g22b = 0;
       int r2b = 0;
```

```
        int g2b = 0;
        int T6b = 0;
        int T7b = 0;



while(T6b < nRep)
{
 T7b = 0;
 while(T7b < nTrain)
 {
  r2b = 0;
  while(r2b<2)
  {
     g2b = 0;
     while(g2b < length[randArray[T6b][T7b]])
     {
        eState22[T6b][T7b][r2b][g2b] = emiss2(DNA1[T6b][T7b][g2b], emcA[T6b], emcC[T6b], emcG[T6b], emcT[T6b], emnA[T6b], emnC[T6b], emnG[T6b],
              emnT

[T6b],r2b);
          //System.out.print("\n" + T6 + " " + T7 + " " + r2 + " " + eState22[T6][T7][r2][g2]);
        g2b++;
     }
          //System.out.print("\n");
    r2b++;
  }
  T7b++;
 }
 T6b++;
}




nRecur++;
}


//***************************** B E G I N S E Q U E N C E P R E D I C T I O N
//********************** R E A D S E Q U E N C E F O R P R E D I C T I O N




//FileInputStream fstream2 = new FileInputStream("Test.txt");
   //DataInputStream in2 = new DataInputStream(fstream2);
   //BufferedReader br2 = new BufferedReader(new InputStreamReader(in2));



        int[][] trialSeqValue = new int[nRep][nTrain];
        Scanner t = new Scanner(new File("trialSeqNUMBER.txt"));
        int T4 = 0;
        int T14 = 0;
```

```
while(T4 < nRep)
{
 T14 =0;
 while(T14 < nTrain)
 {
       trialSeqValue[T4][T14] = t.nextInt();
       //System.out.print("\n"+trialSeqValue[T4][T14]);
       T14++;
 }
T4++;
}
```

```
//********************************************** Prediction Sequence Assemble

char[][][] predDNA = new char[nRep][nTrain][200000];
int pred1 = 0;
int pred2 = 0;
int pred3 = 0;


while(pred1 < nRep)
{
 pred3 = 0;
 while(pred3 < nTrain)
 {
  pred2 = 0;
  while(pred2 < length[trialSeqValue[pred1][pred3]])
  {
     predDNA[pred1][pred3][pred2] = DNA[trialSeqValue[pred1][pred3]][pred2];
     //System.out.print(trialSeqValue[pred1][pred3] + ":" + predDNA[pred1][pred3][pred2]);
     pred2++;
  }
 pred3++;
 }
pred1++;
}

int Z = 0;
int ZZ = 0;
int ZZZ = 0;
int ZZZZ = 0;

//double[][][][] bwEState = new double[nRep][nTrain][2][200000];



while(Z < nRep)
{
 ZZ = 0;
 while(ZZ < nTrain)
 {
```

```
  ZZZ = 0;
  while(ZZZ<2)
  {
     ZZZZ = 0;
     while(ZZZZ < length[trialSeqValue[Z][ZZ]])
     {
        bwEState[Z][ZZ][ZZZ][ZZZZ] = emiss2(predDNA[Z][ZZ][ZZZZ], emcA[Z], emcC[Z], emcG[Z], emcT[Z], emnA[Z], emnC[Z], emnG[Z], emnT[Z], ZZZ);
        ZZZZ++;
     }
         //System.out.print("\n");
    ZZZ++;
  }
  ZZ++;
 }
 Z++;
}




//************************************* V I T E R B I



double[][][] vh = new double[nRep][nTrain][200000];
double[][][] vl = new double[nRep][nTrain][200000];
char[][][] state = new char[nRep][nTrain][200000];



// ************** Using Log Transformation


//Log estates

       double[][][][] LogE = new double[nRep][nTrain][2][200000];
       int rrr2 = 0;
       int rrr = 0;
       int gggg = 0;
       int rrr3 = 0;

     while(rrr2 < nRep)
     {
      rrr3 =0;
      while(rrr3 < nTrain)
      {
        rrr =0;
        while(rrr<2)
        {
           gggg = 0;
           while(gggg < length[trialSeqValue[rrr2][rrr3]])
           {

                LogE[rrr2][rrr3][rrr][gggg] = Math.log(bwEState[rrr2][rrr3][rrr][gggg]);
                //System.out.print("\n" + LogE[rrr2][rrr3][rrr][gggg]);
                gggg++;
           }
```

```
         rrr++;
        }
       rrr3++;
       }
      rrr2++;
     }



//Log tState

double[][][] LogT = new double[nRep][2][2];
int ck = 0;
int ck1 = 0;
int ck2 = 0;
int ck3 = 0;
int ck4 = 0;

while(ck < nRep)
{
 ck4 = 0;
while(ck4 < nTrain)
{
 ck3 = 0;
while(ck3 < length[trialSeqValue[ck][ck4]])
{
 ck1 = 0;
 while(ck1 < 2)
 {
   ck2 = 0;
  while(ck2 < 2)
   {
       LogT[ck][ck1][ck2] = Math.log(bwTSTATE[ck][ck1][ck2]);
       //System.out.print("\n" + LogT[ck1][ck2]);
       ck2++;
   }
    ck1++;
  }
ck3++;
}
ck4++;
}
 ck++;
}


//**** L O G B E G I N

double[] Logb = new double[2];
Logb[0] = Math.log(begin[0]);
Logb[1] = Math.log(begin[1]);




int Vit = 0;
```

```
int Vit2 = 0;
int vCount = 0;
char[][] MAX = new char[nRep][nTrain];
double[] MAX0 = new double[2000000];
double[] MAX1 = new double[2000000];
double[][][] V0 = new double[nRep][nTrain][2000000];
double[][][] V1 = new double[nRep][nTrain][2000000];
double[][][] V0B = new double[nRep][nTrain][2000000];
double[][][] V1B = new double[nRep][nTrain][2000000];
char[][][] vPath = new char[nRep][nTrain][2000000];
double recursive1 = 0;
double recursive2 = 0;
double[][][] H = new double[nRep][nTrain][2000000];
double[][][] L = new double[nRep][nTrain][2000000];
double Vt00 = 0;
double Vt01 = 0;
double Vt10 = 0;
double Vt11 = 0;
char[][][][] ptr = new char[2][nRep][nTrain][200000];




while(Vit < nRep)
{
 Vit2 = 0;
while(Vit2 < nTrain)
{
 vCount = 0;

 while(vCount < length[trialSeqValue[Vit][Vit2]])
 {
   if(vCount == 0)
   {
     V0[Vit][Vit2][vCount] = LogE[Vit][Vit2][0][0] + Logb[0];
     V1[Vit][Vit2][vCount] = LogE[Vit][Vit2][1][0] + Logb[1];

        Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
        Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
        Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
        Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

        if(Vt00 >= Vt10)
        {
           MAX0[vCount] = Vt00;
           ptr[0][Vit][Vit2][vCount] = '+';
        }
        else
        {
           MAX0[vCount] = Vt10;
           ptr[0][Vit][Vit2][vCount] = '-';
        }

        if(Vt01 >= Vt11)
        {
           MAX1[vCount] = Vt01;
           ptr[1][Vit][Vit2][vCount] = '+';
```

```
        }
        else
        {
           MAX1[vCount] = Vt11;
           ptr[1][Vit][Vit2][vCount] = '-';
        }

              //System.out.print(DNA1[Vit][Vit2][vCount] + " " + LogE[Vit][Vit2][0][0] + " " + Logb[Vit2] + " " + V0[Vit][Vit2][vCount]);
              //System.out.print(LogT[Vit][0][0] + " " + V0[Vit][Vit2][vCount]);
              //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
   }
   else
   {
        V0[Vit][Vit2][vCount] = VH1(LogE[Vit][Vit2][0][vCount], MAX0[vCount-1], vCount);
        V1[Vit][Vit2][vCount] = VL1(LogE[Vit][Vit2][1][vCount], MAX1[vCount-1], vCount);

        Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
        Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
        Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
        Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

        if(Vt00 >= Vt10)
        {
           MAX0[vCount] = Vt00;
           ptr[0][Vit][Vit2][vCount] = '+';
        }
        else
        {
           MAX0[vCount] = Vt10;
           ptr[0][Vit][Vit2][vCount] = '-';
        }

        if(Vt01 >= Vt11)
        {
           MAX1[vCount] = Vt01;
           ptr[1][Vit][Vit2][vCount] = '+';
        }
        else
        {
           MAX1[vCount] = Vt11;
           ptr[1][Vit][Vit2][vCount] = '-';
        }


        //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
        //System.out.print(LogT[Vit][0][vCount] + " " + V0[Vit][Vit2][vCount] + "\n");
   }
         //System.out.print(MAX0[vCount] + " " + MAX1[vCount] + "\n");
         //System.out.print(V0[Vit][Vit2][vCount]+ " " + V1[Vit][Vit2][vCount] + " " + DNA1[0][0][vCount] + "\n");
         //System.out.print(vCount + " " + ptr[0][vCount]+ " " + ptr[1][vCount] + "\n");

 vCount++;
 }
Vit2 ++;
//System.out.print("\n");
}
//System.out.print("\n");
```

```
 Vit++;
}



// ***************************** Select pi*
int piCount1 = 0;
int piCount2 = 0;

while(piCount1 < nRep)
{
 piCount2 = 0;
 while(piCount2 < nTrain)
 {
  if(V0[piCount1][piCount2][length[trialSeqValue[piCount1][piCount2]]-1] > V1[piCount1][piCount2][length[trialSeqValue[piCount1][piCount2]]-1])
       MAX[piCount1][piCount2] = '+';
  else
       MAX[piCount1][piCount2] = '-';
   piCount2++;
 }
 piCount1++;
}



char[][][] Path = new char[nRep][nTrain][200000];

int pathCount1 = 0;
int pathCount2 = 0;
int pathCount3 = 0;


while(pathCount3 < nRep)
{
 pathCount2 = 0;
 while(pathCount2 < nTrain)
 {
   pathCount1 = length[trialSeqValue[pathCount3][pathCount2]]-1;
  while(pathCount1 >= 0)
  {
   if(pathCount1 == (length[trialSeqValue[pathCount3][pathCount2]]-1))
   {
        Path[pathCount3][pathCount2][pathCount1] = MAX[pathCount3][pathCount2];
   }
   else
   if(Path[pathCount3][pathCount2][pathCount1 + 1] == '+')
   {
      Path[pathCount3][pathCount2][pathCount1] = ptr[0][pathCount3][pathCount2][pathCount1];
   }
   else
        Path[pathCount3][pathCount2][pathCount1] = ptr[1][pathCount3][pathCount2][pathCount1];
 pathCount1--;
  }
 pathCount2++;
 }
 pathCount3++;
}
```

```
int pathView1 = 0;
int pathView2 = 0;
int pathView3 = 0;

 while(pathView1 < nRep)
 {
  pathView2 = 0;
   while(pathView2 < nTrain)
   {
    pathView3 = 0;
     while(pathView3 < length[trialSeqValue[pathView1][pathView2]])
     {
      //System.out.print(Path[pathView1][pathView2][pathView3] /*+ " " + MAXO[pathView] + "\n"*/);
      pathView3++;
     }
        //System.out.print("\n\n\n\n");
    pathView2++;
   }
   pathView1++;
  }




//************************************* V I T E R B I A C C U R A C Y



// ******************** I D E N T I F Y C p G I S L A N D S


Scanner numCPG = new Scanner(new File("numCPG.txt"));



int[][] Bloc = new int[numSeq][200000];
int[][] Eloc = new int[numSeq][200000];
int[] nCpG = new int[numSeq];
int y3 = 0;


while(numCPG.hasNextInt())
{
   nCpG[y3] = numCPG.nextInt();
   //System.out.print("N" + y3 + " " + nCpG[y3]);
   y3++;
}


int[][] nb = new int[nRep][nTrain];
char[][][] symbolb = new char[nRep][nTrain][200000];

int n2b = 0;
int n3b = 0;
```

```
while(n2b < nRep)
{
 n3b = 0;
 while(n3b < nTrain)
 {
  nb[n2b][n3b] = nCpG[trialSeqValue[n2b][n3b]];
  //System.out.print("\n" + nb[n2b][n3b]);
  n3b++;
 }
n2b++;
}



//******R E A D C p G I S L A N D L O C A T I O N


Scanner Location = new Scanner(new File("Loc.txt"));

int[] B = new int[200000];
int[] E = new int[200000];

int v = 0;

while(Location.hasNextInt())
{

   B[v] = Location.nextInt();
   E[v] = Location.nextInt();
   //System.out.print(B[v] + " " + E[v] + "\n");
   v++;
}

int v1 = 0;
int v2 = 0;
int y4 = 0;

int[][][] Blocb = new int[nRep][nTrain][200000];
int[][][] Elocb = new int[nRep][nTrain][200000];


// ************************* I D E N T I F Y A C T U A L (beg/end) C P G

 while(y4 < numSeq)
 {
  v1 = 0;
  while(v1 < nCpG[y4])
  {
    Bloc[y4][v1] = B[v2];
    Eloc[y4][v1] = E[v2];
    //System.out.print("\n" + y4 + " " + Bloc[y4][v1] + " " + Eloc[y4][v1]);
    v1++;
    v2++;
   }
  y4++;
 }
```

```
int v3 = 0;
int n23b = 0;
int n24b = 0;

while(n23b < nRep)
{
 n24b = 0;
 while(n24b < nTrain)
 {
  v3 = 0;
  while(v3 < nCpG[trialSeqValue[n23b][n24b]])
  {
    Blocb[n23b][n24b][v3] = Bloc[trialSeqValue[n23b][n24b]][v3];
    Elocb[n23b][n24b][v3] = Eloc[trialSeqValue[n23b][n24b]][v3];
    //System.out.print("\n\n" + Blocb[n23b][n24b][v3] + " " + Elocb[n23b][n24b][v3]);
    v3++;
  }
 n24b++;
 }
n23b++;
}


int a3 = 0;
int n4b = 0;
int a13 = 0;

while(n4b < nRep)
{
 a13 = 0;
while(a13 < nTrain)
{
 a3 = 0;
 while(a3 < nb[n4b][a13])
 {
   for(int K3 = Blocb[n4b][a13][a3]; K3 <= Elocb[n4b][a13][a3]; K3++)
   {
        symbolb[n4b][a13][K3] = '+';

   }
  a3++;
 }
a13++;
}
 n4b++;
}


int b = 0;
int n5b = 0;
int n6b = 0;
double[][] nonGC = new double[nRep][nTrain];
double nonCount = 0;
```

```
while(n5b < nRep)
{
 n6b = 0;
 while(n6b < nTrain)
 {
  b = 0;
  while(b<length[trialSeqValue[n5b][n6b]])
  {
   if(symbolb[n5b][n6b][b] != '+')
   {
     symbolb[n5b][n6b][b] = '-';
   }
  //System.out.print(symbolb[n5b][n6b][b]);
  b++;
  }
 n6b++;
 }
n5b++;
}



int b12 = 0;
int b13 = 0;
int b14 = 0;
double[][] nMatch2 = new double[nRep][nTrain];

while(b13 < nRep)
{
 b14 = 0;
while(b14 < nTrain)
{
 b12 = 0;
 while(b12 < length[trialSeqValue[b13][b14]])
 {
   if(Path[b13][b14][b12] == symbolb[b13][b14][b12])
   {
        nMatch2[b13][b14]++;
   }
  b12++;
  }
 b14++;
 }
 b13++;
}



double[][] pMatch = new double[nRep][nTrain];
int n7b = 0;
int n8b = 0;

double pSUM = 0;

while(n7b < nRep)
```

```
{
 n8b = 0;
 while(n8b < nTrain)
 {
  pMatch[n7b][n8b] = nMatch2[n7b][n8b]/length[trialSeqValue[n7b][n8b]];
  pSUM = pSUM + pMatch[n7b][n8b];
  //System.out.print("\n\nViterbi Accuracy: " + pMatch[n7b][n8b]);
 n8b++;
 }
n7b++;
}



int printCount1 = 0;
int printCount2 = 0;

while(printCount1 < nRep)
{
 printCount2 = 0;
 while(printCount2 < nTrain)
 {

  PrintWriter pw2 = new PrintWriter(new FileWriter("Results.txt", true));
  pw2.println(printCount1 + " - " + printCount2 + ": " + pMatch[printCount1][printCount2]);
  pw2.println();
  pw2.close();

 printCount2++;
 }
printCount1++;
}


// ************************************ E N D M A I N
}




//****************************** T R A N S I T I O N


public static double[][] pr()
{
       double[][] tState = new double [2][2];

          tState[0][0] = /*.99998*/ .75;
          tState[0][1] = /*.00002*/ .25;
          tState[1][0] = .0001;
          tState[1][1] = .9999;

       return (tState);
}


//*************************** E M I S S I O N
```

```
public static double emiss(char nullDNA1, int g)
{
              double emST2 = 0;


                        if(nullDNA1 == 'A')
                        {
                            if(g==0)
                            emST2= .15;
                            else
                            emST2 = .25;
                        }
                        else if(nullDNA1 == 'C')
                        {
                            if(g==0)
                            emST2 = .35;
                            else
                            emST2 = .25;
                        }
                        else if(nullDNA1 == 'G')
                        {
                            if(g==0)
                            emST2 = .35;
                            else
                            emST2 = .25;
                        }
                        else if(nullDNA1 == 'T')
                        {
                            if(g==0)
                            emST2 = .15;
                            else
                            emST2 = .25;
                        }



       return (emST2);
}


//********************************* S C A L I N G



public static double Scale1(double[][][][] emState, double[] begin, int F1, int nTrain)
{
       double sA = emState[F1][nTrain][0][0]*begin[0] + emState[F1][nTrain][1][0]*begin[1];
       return sA;
}
public static double Scale2(double[][][][] emState, double[][][] tState, double[][][] fsH, double[][][] fsL, int i, int F1, int nTrain)
{
       double sB = emState[F1][nTrain][0][i]*(tState[F1][1][0]*fsL[F1][nTrain][i-1] + tState[F1][0][0]*fsH[F1][nTrain][i-1]) +
             emState[F1][nTrain][1][i]*

(tState[F1][1][1]*fsL[F1][nTrain][i-1] + tState[F1][0][1]*fsH[F1][nTrain][i-1]);
```

```
       return sB;
}


//************************* F O R W A R D S C A L E D


public static double fHS1(double[][][][] emState, double[] begin, double[][][] tState, double[][][] s, int i, int F1, int nTrain)
{
       double fSH1 = (1/s[F1][nTrain][0])*emState[F1][nTrain][0][0]*begin[0];
       return fSH1;
}
public static double fLS1(double[][][][] emState, double[] begin, double[][][] tState, double[][][] s, int i, int F1, int nTrain)
{
       double fSL1 = (1/s[F1][nTrain][0])*emState[F1][nTrain][1][0]*begin[1];
       return fSL1;
}
public static double fHS2(double[][][][] emState, double[] begin, double[][][] tState, double[][][] h, double[][][] l, double[][][] s, int i,
      int F1, int

nTrain)
{
       double fSH2 = 1/s[F1][nTrain][i]*emState[F1][nTrain][0][i]*(tState[F1][1][0]*l[F1][nTrain][i-1] + tState[F1][0][0]*h[F1][nTrain][i-1]);
       return fSH2;
}
public static double fLS2(double[][][][] emState, double[] begin, double[][][] tState, double[][][] h, double[][][] l, double[][][] s, int i,
      int F1, int

nTrain)
{
       double fSL2 = 1/s[F1][nTrain][i]*emState[F1][nTrain][1][i]*(tState[F1][1][1]*l[F1][nTrain][i-1] + tState[F1][0][1]*h[F1][nTrain][i-1]);
       return fSL2;
}


//********************************* S C A L I N G B A C K W A R D


public static double ScaleB1(double[][][][] emState, double[] begin, int F1, int nTrain)
{
       double sA = emState[F1][nTrain][0][0]*begin[0] + emState[F1][nTrain][1][0]*begin[1];
       return sA;
}
public static double ScaleB2(double[][][][] emState, double[][][] tState, double[][][] bsH, double[][][] bsL, int i, int F1, int nTrain)
{
       double sB = emState[F1][nTrain][0][i+1]*(tState[F1][1][0]*bsL[F1][nTrain][i+1] + tState[F1][0][0]*bsH[F1][nTrain][i+1]) +
             emState[F1][nTrain][1][i

+1]*(tState[F1][1][1]*bsL[F1][nTrain][i+1] + tState[F1][0][1]*bsH[F1][nTrain][i+1]);

       return sB;
}
```

```
//************************ B A C K W A R D A L G O R I T H M


public static double bHS1(double[][][] s, int i, int F1, int nTrain)
{
       double fSH1 = (1/s[F1][nTrain][i]);
       return fSH1;
}
public static double bLS1(double[][][] s, int i, int F1, int nTrain)
{
       double fSL1 = (1/s[F1][nTrain][i]);
       return fSL1;
}

public static double bh(double[][][][] emState, double[][][] tState, double[][][] bh, double[][][] bl, int i, int i2, double[][][] s, int nTrain)
{
       double bh1 = 1/s[i2][nTrain][i]*(tState[i2][0][0]*emState[i2][nTrain][0][i+1]*bh[i2][nTrain][i+1] +
              tState[i2][0][1]*emState[i2][nTrain][1][i+1]*bl

[i2][nTrain][i+1]);

       return bh1;
}

public static double bl(double[][][][] emState, double[][][] tState, double[][][] bh, double[][][] bl, int i, int i2, double[][][] s, int nTrain)
{
       double bl1 = 1/s[i2][nTrain][i]*(tState[i2][1][0]*emState[i2][nTrain][0][i+1]*bh[i2][nTrain][i+1] +
              tState[i2][1][1]*emState[i2][nTrain][1][i+1]*bl

[i2][nTrain][i+1]);

       return bl1;
}

//************************ B A U M - W E L C H


// Estimated Transition


public static double BWtrans(double f, double b, double tState, double emState)
{
       double akl = 0;

       akl = f*tState*emState*b;

       return akl;
}


// Estimated Emissions


public static double BWemiss(double f, double b, double s)
{
```

```
        double e = 0;

        e = f*b*s;

        return e;
}


public static double emiss2(char predDNA, double emcA, double emcC, double emcG, double emcT, double emnA, double emnC, double emnG, double
       emnT, int g)
{
               double emST2 = 0;


                        if(predDNA == 'A')
                        {
                            if(g==0)
                            emST2= emcA;
                            else
                            emST2 = emnA;
                        }
                        else if(predDNA == 'C')
                        {
                            if(g==0)
                            emST2 = emcC;
                            else
                            emST2 = emnC;
                        }
                        else if(predDNA == 'G')
                        {
                            if(g==0)
                            emST2 = emcG;
                            else
                            emST2 = emnG;
                        }
                        else if(predDNA == 'T')
                        {
                            if(g==0)
                            emST2 = emcT;
                            else
                            emST2 = emnT;
                        }



        return (emST2);
}

public static double[][] pr2(double a00, double a01, double a10, double a11)
{
        double[][] tState = new double [2][2];

           tState[0][0] = a00;
           tState[0][1] = a01;
           tState[1][0] = a10;
           tState[1][1] = a11;
```

```
        return (tState);
}



//************************ V I T E R B I A L G O R I T H M

public static double VH1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VL1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VH2(double E, double Vl, double T, int i)
{
   double q = E + Vl + T;
   return q;
}
public static double VL2(double E, double Vh, double T, int i)
{
   double q = E + Vh + T;
   return q;
}


//************************* P R I N T T O F I L E Transition


public void writeLinesToFileT(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Transition:\n" + "A00: " + gc1);
          pw.println("Transition:\n" + "A01: " + gc2);
          pw.println("Transition:\n" + "A10: " + gc3);
          pw.println("Transition:\n" + "A11: " + gc4);
```

```
          arrays++;
          pw.println(" ");
          pw.flush();
}


//************************* P R I N T T O F I L E Emission


public void writeLinesToFileE(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4, double gc5, double gc6,
      double gc7,

double gc8) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Emiss CG: " + gc1 + " " + gc2 + " " + gc3 + " " + gc4);
          pw.println("Emiss NON: " + gc5 + " " + gc6 + " " + gc7 + " " + gc8);
          arrays++;
          pw.println(" ");
          pw.flush();
}


//*************************** E M I S S I O N


public static double emissB(char nullDNA1, int g, double emcA, double emcC, double emcG, double emcT,double emnA,double emnC,double emnG,double
      emnT)
{
              double emST2 = 0;


                        if(nullDNA1 == 'A')
                        {
                            if(g==0)
                            emST2= emcA;
                            else
                            emST2 = emnA;
                        }
```

```
                    else if(nullDNA1 == 'C')
                    {
                        if(g==0)
                        emST2 = emcC;
                        else
                        emST2 = emnC;
                    }
                    else if(nullDNA1 == 'G')
                    {
                        if(g==0)
                        emST2 = emcG;
                        else
                        emST2 = emnG;
                    }
                    else if(nullDNA1 == 'T')
                    {
                        if(g==0)
                        emST2 = emcT;
                        else
                        emST2 = emnT;
                    }



        return (emST2);
}



//************************ P R I N T T O F I L E


public void writeLinesToFile(String filename, boolean appendToFile, double gc1, int i) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

        //If the file already exists, start writing at the end of it.
        pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

        pw = new PrintWriter(new FileWriter(filename));
        //this is equal to:
        //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println(i + " " + gc1);
          arrays++;
          pw.println(" ");
          pw.flush();
```

```
}

}
```

Appendix D

Viterbi Training Algorithm (Java)

```
import java.io.*;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;


public class ViterbiTraining
{
   public static void main(String args[]) throws IOException
  {

//************************** F I R S T U S E T H I S 'R' C O D E T O R A N D O M L Y S E L E C T T R A I N I N G S E Q U N C E S

/*


# * * * * * * * * * * S A M P L E
W1 = 0:19
sW1 = sample(W1,10)
   #sort(sW1)
W2 = W1[-(sW1+1)]
   #sort(W2)
testVal = sample(W2,1)

# * * * * * * * * * * W R I T E T O F I L E

write.infile(W2, "C:\\Users\\Robert\\Desktop\\School\\HMM\\Rprint.txt")
write.infile(testVal, "C:\\Users\\Robert\\Desktop\\School\\HMM\\trialSeqNUMBER.txt")

*/



   Scanner scan = new Scanner(System.in);

// ********** I N F O F O R T R A I N I N G S E Q U E N C E S **********



double[] begin = new double[2];

begin[0] = .5;
begin[1] = .5;
```

```
   FileInputStream fstream = new FileInputStream("oNuc2a.txt");
   DataInputStream in = new DataInputStream(fstream);
   BufferedReader br = new BufferedReader(new InputStreamReader(in));

        int nRep = 5;


        // Training Data Compilation

    String strLine = null;
    int ii = 0;
    String[] S = new String[100000];
    //Read File Line By Line

    while ((strLine = br.readLine()) != null)
    {
        S[ii] = strLine;
        ii++;
    }


String Snew = "";
int counter = 0;

       for(int k = 0; k < ii; k++)
       {
          //System.out.print("\n" + S[k]);
          Snew = Snew.concat(S[k]);
          counter = Snew.length();
          //System.out.print("\n" + Snew);
       }


int i = counter;
char[] nullDNA = new char[ii];
       nullDNA = Snew.toCharArray();


  /* System.out.print("\nHow many training sequences will be used?\n");
       int nTrain = scan.nextInt();
  */

int nTrain = 9;


int numSeq = 19;
char[][] DNA = new char[numSeq][i];
int y3 = 0;
int y4 = 0;
int[] nCpG = new int[numSeq];
```

```
// ******************** I D E N T I F Y C p G I S L A N D S


Scanner numCPG = new Scanner(new File("numCPG.txt"));



int[][] Bloc = new int[numSeq][i];
int[][] Eloc = new int[numSeq][i];


while(numCPG.hasNextInt())
{
   nCpG[y3] = numCPG.nextInt();
   //System.out.print(nCpG[y3] + " ");
   y3++;
}

System.out.print("1");

//******R E A D C p G I S L A N D L O C A T I O N


Scanner Location = new Scanner(new File("Loc.txt"));

int[] B = new int[100];
int[] E = new int[100];

int v = 0;

while(Location.hasNextInt())
{

   B[v] = Location.nextInt();
   E[v] = Location.nextInt();
   //System.out.print(B[v] + " " + E[v] + "\n");
   v++;
}



int v1 = 0;
int v2 = 0;


 while(y4 < numSeq)
 {
  v1 = 0;
  while(v1 < nCpG[y4])
  {
    Bloc[y4][v1] = B[v2];
    Eloc[y4][v1] = E[v2];
    //System.out.print("\n" + y4 + " " + Bloc[y4][v1] + " " + Eloc[y4][v1]);
    v1++;
    v2++;
  }
 y4++;
```

```
}


// ************************* D E T E R M I N E L E N G T H O F S E Q U E N C E S


int y5 = 0;
int y6 = 0;

char[][] sCpG = new char[numSeq][i];

int[] lengthNull = new int[i];
int[] length = new int[numSeq];
int[] iLength = new int[numSeq]; // postion at which individual sequence ends
int z = 0;


while(y5 < numSeq)
{
   y6 = 0;
   while(y6 < i)
   {
      if(nullDNA[y6] == 'X')
      {
        length[y5] = lengthNull[y5];
        iLength[y5] = y6;
        //System.out.print("\nLength" + y5 + " " + length[y5]);
        y5++;
      }
      else
        lengthNull[y5]++;
        y6++;
    }
   y5++;
}



//System.out.print(length[0] + " " + length[1]);




//********************************** R A N D O M N U M B E R R E A D


Scanner Random = new Scanner(new File("Rprint.txt"));


Random rand = new Random();
int rCount = 0;
int rCheck = 0;
int[][] randArray = new int[nRep][nTrain];

while(rCheck < nRep)
```

```
{
   rCount = 0;
 while(rCount < nTrain)
 {
   randArray[rCheck][rCount] = Random.nextInt();
   //System.out.print(randArray[rCheck][rCount] + " ");
   rCount++;
 }
       //System.out.print("\n");
 rCheck++;
}



// ******************************** B U I L D S E Q U E N C E


//FileWriter outFile = new FileWriter("SeqConfirm.txt");
PrintWriter out1 = new PrintWriter("SeqConfirm.txt");


char[][][] symbol = new char[nRep][numSeq][i];

int y7 = 0;
int y8 = 0;
int x9 = 0;
int multi = 0;


  while(y7 < numSeq)
  {
    x9 = 0;
   while(y8 < iLength[y7])
   {
        if(nullDNA[y8+1] == 'X')
        {
          DNA[y7][x9] = nullDNA[y8];
          y8 = (y8+1);
        }
        else
        {
          DNA[y7][x9] = nullDNA[y8];
        }
        //System.out.print(DNA[y7][x9]);
        x9++;
        y8++;
   }
        //System.out.print("\n\n S  E      Q");
   y7++;
  }



//********************* B U I L D S E Q U E N C E B A S E D O F F O F R A N D O M A S S I G N M E N T
```

```
char[][][] DNA1 = new char[nRep][nTrain][i];
int M0 = 0;
int M = 0;
int M2 = 0;
int M3 = 0;

while(M3 < nRep)
{
  M = 0;
while(M < nTrain)
{
   M2 = randArray[M3][M];
   M0 = 0;
 while(M0 < length[M2])
 {
   DNA1[M3][M][M0] = DNA[M2][M0];
        //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   M0++;
 }
   M++;
}

        //System.out.print(M3 + " " + DNA1[M3][M][M0]);
   //System.out.print("\n\n");
M3++;
}
```

```
double[][][] tState = new double[nRep][2][2];


int T3 = 0;
int T13 = 0;
int T23 = 0;

while(T3 < nRep)
{
 T13 = 0;
 while(T13 < 2)
 {
  T23 = 0;
  while(T23 < 2)
  {
   tState[T3][T13][T23] = pr(T13, T23);
   //System.out.print(tState[T3][T13][T23] + "\n");
   T23++;
  }
  T13++;
 }
 T3++;
}
```

```
        double[][][][] eState22 = new double[nRep][nTrain][2][200000];
        int g22 = 0;
        int r2 = 0;
        int g2 = 0;
        int T6 = 0;
        int T7 = 0;



while(T6 < nRep)
{
 T7 = 0;
 while(T7 < nTrain)
 {
  r2 = 0;
  while(r2<2)
  {
     g2 = 0;
     while(g2 < length[randArray[T6][T7]])
     {
        eState22[T6][T7][r2][g2] = emiss2(DNA1[T6][T7][g2], r2);
          //System.out.print("\n" + T6 + " " + T7 + " " + r2 + " " + eState22[T6][T7][r2][g2]);
        g2++;
     }
         //System.out.print("\n");
    r2++;
  }
  T7++;
 }
 T6++;
}




// ************** Using Log Transformation


//Log estates

        double[][][][] LogE = new double[nRep][nTrain][2][200000];
        int rrr2 = 0;
        int rrr = 0;
        int gggg = 0;
        int rrr3 = 0;

     while(rrr2 < nRep)
     {
      rrr3 =0;
      while(rrr3 < nTrain)
      {
        rrr =0;
        while(rrr<2)
        {
```

```
          gggg = 0;
          while(gggg < length[randArray[rrr2][rrr3]])
          {

               LogE[rrr2][rrr3][rrr][gggg] = Math.log(eState22[rrr2][rrr3][rrr][gggg]);
               //System.out.print("\n" + LogE[rrr2][rrr3][rrr][gggg]);
               gggg++;
          }

         rrr++;
        }
       rrr3++;
       }
      rrr2++;
     }




//Log tState

double[][][] LogT = new double[nRep][2][2];
int ck = 0;
int ck1 = 0;
int ck2 = 0;
int ck3 = 0;
int ck4 = 0;

while(ck < nRep)
{
 ck4 = 0;
while(ck4 < nTrain)
{
 ck3 = 0;
while(ck3 < length[randArray[ck][ck4]])
{
 ck1 = 0;
 while(ck1 < 2)
 {
   ck2 = 0;
  while(ck2 < 2)
   {
        LogT[ck][ck1][ck2] = Math.log(tState[ck][ck1][ck2]);
        //System.out.print("\n" + LogT[ck][ck1][ck2]);
        ck2++;
   }
    ck1++;
  }
ck3++;
}
ck4++;
}
 ck++;
}
```

```
//**** L O G B E G I N

double[] Logb = new double[2];
Logb[0] = Math.log(begin[0]);
Logb[1] = Math.log(begin[1]);



//************** T R A I N I N G V I T E R B I



int Vit = 0;
int Vit2 = 0;
int vCount = 0;
char[][] MAX = new char[nRep][nTrain];
double[] MAX0 = new double[2000000];
double[] MAX1 = new double[2000000];
double[][][] V0 = new double[nRep][nTrain][2000000];
double[][][] V1 = new double[nRep][nTrain][2000000];
double[][][] V0B = new double[nRep][nTrain][2000000];
double[][][] V1B = new double[nRep][nTrain][2000000];
char[][][] vPath = new char[nRep][nTrain][2000000];
double recursive1 = 0;
double recursive2 = 0;
double[][][] H = new double[nRep][nTrain][2000000];
double[][][] L = new double[nRep][nTrain][2000000];
double Vt00 = 0;
double Vt01 = 0;
double Vt10 = 0;
double Vt11 = 0;
char[][][][] ptr = new char[2][nRep][nTrain][200000];




while(Vit < nRep)
{
 Vit2 = 0;
while(Vit2 < nTrain)
{
 vCount = 0;

 while(vCount < length[randArray[Vit][Vit2]])
 {
   if(vCount == 0)
   {
     V0[Vit][Vit2][vCount] = LogE[Vit][Vit2][0][0] + Logb[0];
     V1[Vit][Vit2][vCount] = LogE[Vit][Vit2][1][0] + Logb[1];

        Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
        Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
        Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
        Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

        if(Vt00 >= Vt10)
```

```
    {
       MAX0[vCount] = Vt00;
       ptr[0][Vit][Vit2][vCount] = '+';
    }
    else
    {
       MAX0[vCount] = Vt10;
       ptr[0][Vit][Vit2][vCount] = '-';
    }

    if(Vt01 >= Vt11)
    {
       MAX1[vCount] = Vt01;
       ptr[1][Vit][Vit2][vCount] = '+';
    }
    else
    {
       MAX1[vCount] = Vt11;
       ptr[1][Vit][Vit2][vCount] = '-';
    }

          //System.out.print(DNA1[Vit][Vit2][vCount] + " " + LogE[Vit][Vit2][0][0] + " " + Logb[Vit2] + " " + V0[Vit][Vit2][vCount]);
          //System.out.print(LogT[Vit][0][0] + " " + V0[Vit][Vit2][vCount]);
          //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
}
else
{
    V0[Vit][Vit2][vCount] = VH1(LogE[Vit][Vit2][0][vCount], MAX0[vCount-1], vCount);
    V1[Vit][Vit2][vCount] = VL1(LogE[Vit][Vit2][1][vCount], MAX1[vCount-1], vCount);

    Vt00 = LogT[Vit][0][0] + V0[Vit][Vit2][vCount];
    Vt10 = LogT[Vit][1][0] + V1[Vit][Vit2][vCount];
    Vt01 = LogT[Vit][0][1] + V0[Vit][Vit2][vCount];
    Vt11 = LogT[Vit][1][1] + V1[Vit][Vit2][vCount];

    if(Vt00 >= Vt10)
    {
       MAX0[vCount] = Vt00;
       ptr[0][Vit][Vit2][vCount] = '+';
    }
    else
    {
       MAX0[vCount] = Vt10;
       ptr[0][Vit][Vit2][vCount] = '-';
    }

    if(Vt01 >= Vt11)
    {
       MAX1[vCount] = Vt01;
       ptr[1][Vit][Vit2][vCount] = '+';
    }
    else
    {
       MAX1[vCount] = Vt11;
       ptr[1][Vit][Vit2][vCount] = '-';
    }
```

```
     //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
       //System.out.print(LogT[Vit][0][vCount] + " " + V0[Vit][Vit2][vCount] + "\n");
   }
        //System.out.print(MAX0[vCount] + " " + MAX1[vCount] + "\n");
        //System.out.print(V0[Vit][Vit2][vCount]+ " " + V1[Vit][Vit2][vCount] + " " + DNA1[0][0][vCount] + "\n");
        //System.out.print(vCount + " " + ptr[0][vCount]+ " " + ptr[1][vCount] + "\n");

 vCount++;
 }
Vit2 ++;
//System.out.print("\n");
}
//System.out.print("\n");
 Vit++;
}



// ***************************** Select pi*
int piCount1 = 0;
int piCount2 = 0;

while(piCount1 < nRep)
{
 piCount2 = 0;
 while(piCount2 < nTrain)
 {
  if(V0[piCount1][piCount2][length[randArray[piCount1][piCount2]]-1] > V1[piCount1][piCount2][length[randArray[piCount1][piCount2]]-1])
       MAX[piCount1][piCount2] = '+';
  else
       MAX[piCount1][piCount2] = '-';
   piCount2++;
 }
 piCount1++;
}




char[][][] Path = new char[nRep][nTrain][200000];

int pathCount1 = 0;
int pathCount2 = 0;
int pathCount3 = 0;


while(pathCount3 < nRep)
{
 pathCount2 = 0;
 while(pathCount2 < nTrain)
 {
   pathCount1 = length[randArray[pathCount3][pathCount2]]-1;
  while(pathCount1 >= 0)
  {
   if(pathCount1 == (length[randArray[pathCount3][pathCount2]]-1))
   {
       Path[pathCount3][pathCount2][pathCount1] = MAX[pathCount3][pathCount2];
```

```
   }
   else
   if(Path[pathCount3][pathCount2][pathCount1 + 1] == '+')
   {
      Path[pathCount3][pathCount2][pathCount1] = ptr[0][pathCount3][pathCount2][pathCount1];
   }
   else
       Path[pathCount3][pathCount2][pathCount1] = ptr[1][pathCount3][pathCount2][pathCount1];
 pathCount1--;
  }
 pathCount2++;
 }
 pathCount3++;
}




int pathView1 = 0;
int pathView2 = 0;
int pathView3 = 0;

 while(pathView1 < nRep)
 {
  pathView2 = 0;
   while(pathView2 < nTrain)
   {
    pathView3 = 0;
     while(pathView3 < length[randArray[pathView1][pathView2]])
     {
      //System.out.print(Path[pathView1][pathView2][pathView3] /*+ " " + MAX0[pathView] + "\n"*/);
      pathView3++;
     }
       //System.out.print("\n\n\n\n");
    pathView2++;
   }
   pathView1++;
  }



//******************** C a l c u l a t e P a r a m e t e r s

int p1 = 0;
int p2 = 0;
int p3 = 0;

double[] a00 = new double[nRep];
 double[] t00 = new double[nRep];
double[] a01 = new double[nRep];
 double[] t01 = new double[nRep];
double[] a10 = new double[nRep];
 double[] t10 = new double[nRep];
double[] a11 = new double[nRep];
 double[] t11 = new double[nRep];

String file3 = "Results.txt";
```

```
boolean append3 = true;
ViterbiTraining util3 = new ViterbiTraining();


while(p1 < nRep)
{
 p2 = 0;
 while(p2 < nTrain)
 {
  p3 = 0;
  while(p3 < (length[randArray[p1][p2]]-1))
  {
    if(Path[p1][p2][p3] == '+' && Path[p1][p2][p3+1] == '+')
        a00[p1]++;
    if(Path[p1][p2][p3] == '+' && Path[p1][p2][p3+1] == '-')
        a01[p1]++;
    if(Path[p1][p2][p3] == '-' && Path[p1][p2][p3+1] == '+')
        a10[p1]++;
    if(Path[p1][p2][p3] == '-' && Path[p1][p2][p3+1] == '-')
        a11[p1]++;
    p3++;
   }
  p2++;
 }

        t00[p1] = (a00[p1]/(a00[p1] + a01[p1]));
        t01[p1] = (a01[p1]/(a00[p1] + a01[p1]));
        t10[p1] = (a10[p1]/(a10[p1] + a11[p1]));
        t11[p1] = (a11[p1]/(a10[p1] + a11[p1]));

        //System.out.print(t00[p1] + " " + t01[p1] + " " + t10[p1] + " " + t11[p1] + "\n");
util3.writeLinesToFileT(file3, append3,t00[p1], t01[p1], t10[p1], t11[p1]);
p1++;
}


int p12 = 0;
int p22 = 0;
int p32 = 0;

double[] cA = new double[nRep];
double[] cC = new double[nRep];
double[] cG = new double[nRep];
double[] cT = new double[nRep];

double[] nA = new double[nRep];
double[] nC = new double[nRep];
double[] nG = new double[nRep];
double[] nT = new double[nRep];


double[] ecA = new double[nRep];
double[] ecC = new double[nRep];
double[] ecG = new double[nRep];
double[] ecT = new double[nRep];

double[] enA = new double[nRep];
```

```
double[] enC = new double[nRep];
double[] enG = new double[nRep];
double[] enT = new double[nRep];

String file2 = "Results.txt";
boolean append2 = true;
ViterbiTraining util2 = new ViterbiTraining();


while(p12 < nRep)
{
 p22 = 0;
 while(p22 < nTrain)
 {
  p32 = 0;
  while(p32 < (length[randArray[p12][p22]]))
  {
     if(Path[p12][p22][p32] == '+')
     {
       if(DNA1[p12][p22][p32] == 'A')
        cA[p12]++;
       if(DNA1[p12][p22][p32] == 'C')
        cC[p12]++;
       if(DNA1[p12][p22][p32] == 'G')
        cG[p12]++;
       if(DNA1[p12][p22][p32] == 'T')
        cT[p12]++;
     }

     if(Path[p12][p22][p32] == '-')
     {
       if(DNA1[p12][p22][p32] == 'A')
        nA[p12]++;
       if(DNA1[p12][p22][p32] == 'C')
        nC[p12]++;
       if(DNA1[p12][p22][p32] == 'G')
        nG[p12]++;
       if(DNA1[p12][p22][p32] == 'T')
        nT[p12]++;
     }
     p32++;
    }
   p22++;
  }


       //System.out.print(cA[p12] + " " + cC[p12] + " " + cG[p12] + " " + cT[p12] + "\n");

 p12++;
}


int eCount = 0;

while(eCount < nRep)
{
   ecA[eCount] = (cA[eCount]/(cA[eCount]+cC[eCount]+cG[eCount]+cT[eCount]));
```

```
   ecC[eCount] = (cC[eCount]/(cA[eCount]+cC[eCount]+cG[eCount]+cT[eCount]));
   ecG[eCount] = (cG[eCount]/(cA[eCount]+cC[eCount]+cG[eCount]+cT[eCount]));
   ecT[eCount] = (cT[eCount]/(cA[eCount]+cC[eCount]+cG[eCount]+cT[eCount]));

   enA[eCount] = (nA[eCount]/(nA[eCount]+nC[eCount]+nG[eCount]+nT[eCount]));
   enC[eCount] = (nC[eCount]/(nA[eCount]+nC[eCount]+nG[eCount]+nT[eCount]));
   enG[eCount] = (nG[eCount]/(nA[eCount]+nC[eCount]+nG[eCount]+nT[eCount]));
   enT[eCount] = (nT[eCount]/(nA[eCount]+nC[eCount]+nG[eCount]+nT[eCount]));
        //System.out.print(ecA[eCount] + " " + ecC[eCount] + " " + ecG[eCount] + " " + ecT[eCount] + "\n");

util2.writeLinesToFileE(file2, append2,ecA[eCount],ecC[eCount],ecG[eCount],ecT[eCount],enA[eCount],enC[eCount],enG[eCount],enT[eCount]);
   eCount++;
}




Scanner Random2 = new Scanner(new File("trialSeqNUMBER.txt"));


//Random rand2 = new Random();
int rCount2 = 0;
int rCheck2 = 0;
int[][] trialSeqValue = new int[nRep][nTrain];

while(rCheck2 < nRep)
{
   rCount2 = 0;
 while(rCount2 < nTrain)
 {
   trialSeqValue[rCheck2][rCount2] = Random2.nextInt();
   //System.out.print(trialSeqValue[rCheck2][rCount2] + " ");
   rCount2++;
 }
        //System.out.print("\n");
 rCheck2++;
}



//******************** B U I L D S E Q U E N C E B A S E D O F F O F R A N D O M A S S I G N M E N T


char[][][] DNA2 = new char[nRep][nTrain][i];
int M02 = 0;
int M2b = 0;
int M22 = 0;
int M32 = 0;

while(M32 < nRep)
{
  M2b = 0;
while(M2b < nTrain)
{
   M22 = trialSeqValue[M32][M2b];
   M02 = 0;
 while(M02 < length[M22])
```

```
 {
   DNA2[M32][M2b][M02] = DNA[M22][M02];
        //System.out.print(M32 + " " + DNA2[M32][M2b][M02]);
   M02++;
 }
   M2b++;
}
   //System.out.print("\n\n");
M32++;
}



double[][][] tState2 = new double[nRep][2][2];


int T3b = 0;
int T13b = 0;
int T23b= 0;

while(T3b < nRep)
{
 T13b = 0;
 while(T13b < 2)
 {
  T23b = 0;
  while(T23b < 2)
  {
   tState2[T3b][T13b][T23b] = pr2(T13b, T23b, t00[T3b], t01[T3b], t10[T3b], t11[T3b]);
   //System.out.print(tState2[T3b][T13b][T23b] + "\n");
   T23b++;
  }
  T13b++;
 }
 T3b++;
}




       double[][][][] eState22b = new double[nRep][nTrain][2][200000];
       int g22b = 0;
       int r2b = 0;
       int g2b = 0;
       int T6b = 0;
       int T7b = 0;



while(T6b < nRep)
{
 T7b = 0;
 while(T7b < nTrain)
 {
  r2b = 0;
  while(r2b<2)
  {
```

```
    g2b = 0;
    while(g2b < length[trialSeqValue[T6b][T7b]])
    {
       eState22b[T6b][T7b][r2b][g2b] = emiss3(DNA2[T6b][T7b][g2b], r2b, ecA[T6b], ecC[T6b], ecG[T6b], ecT[T6b], enA[T6b], enC[T6b], enG[T6b],
             enT[T6b]);
         //System.out.print("\n" + T6b + " " + T7b + " " + r2b + " " + eState22b[T6b][T7b][r2b][g2b]);
       g2b++;
    }
        //System.out.print("\n");
   r2b++;
  }
  T7b++;
 }
 T6b++;
}




//Log tState

double[][][] LogTb = new double[nRep][2][2];
int ckb = 0;
int ck1b = 0;
int ck2b = 0;
int ck3b = 0;
int ck4b = 0;

while(ckb < nRep)
{
 ck4b = 0;
while(ck4b < nTrain)
{
 ck3b = 0;
while(ck3b < length[trialSeqValue[ckb][ck4b]])
{
 ck1b = 0;
 while(ck1b < 2)
 {
   ck2b = 0;
  while(ck2b < 2)
   {
       LogTb[ckb][ck1b][ck2b] = Math.log(tState2[ckb][ck1b][ck2b]);
       //System.out.print("\n" + LogTb[ckb][ck1b][ck2b]);
       ck2b++;
   }
    ck1b++;
  }
ck3b++;
}
ck4b++;
}
 ckb++;
}
```

```
//Log estates

        double[][][][] LogEb = new double[nRep][nTrain][2][200000];
        int rrr2b = 0;
        int rrrb = 0;
        int ggggb = 0;
        int rrr3b = 0;

     while(rrr2b < nRep)
     {
      rrr3b =0;
      while(rrr3b < nTrain)
      {
        rrrb =0;
        while(rrrb<2)
        {
           ggggb = 0;
           while(ggggb < length[trialSeqValue[rrr2b][rrr3b]])
           {

                LogEb[rrr2b][rrr3b][rrrb][ggggb] = Math.log(eState22b[rrr2b][rrr3b][rrrb][ggggb]);
                //System.out.print("\n" + LogEb[rrr2b][rrr3b][rrrb][ggggb]);
                ggggb++;
           }

          rrrb++;
         }
        rrr3b++;
        }
       rrr2b++;
      }




//************** T R A I N I N G V I T E R B I



int Vitb = 0;
int Vit2b = 0;
int vCountb = 0;
char[][] MAXb = new char[nRep][nTrain];
double[] MAX0b = new double[2000000];
double[] MAX1b = new double[2000000];
double[][][] V0b = new double[nRep][nTrain][2000000];
double[][][] V1b = new double[nRep][nTrain][2000000];
double[][][] V0Bb = new double[nRep][nTrain][2000000];
double[][][] V1Bb = new double[nRep][nTrain][2000000];
char[][][] vPathb = new char[nRep][nTrain][2000000];
double recursive1b = 0;
double recursive2b = 0;
double[][][] Hb = new double[nRep][nTrain][2000000];
double[][][] Lb = new double[nRep][nTrain][2000000];
double Vt00b = 0;
double Vt01b = 0;
double Vt10b = 0;
```

```
double Vt11b = 0;
char[][][][] ptrb = new char[2][nRep][nTrain][200000];




while(Vitb < nRep)
{
 Vit2b = 0;
while(Vit2b < nTrain)
{
 vCountb = 0;

 while(vCountb < length[trialSeqValue[Vitb][Vit2b]])
 {
   if(vCountb == 0)
   {
     V0b[Vitb][Vit2b][vCountb] = LogEb[Vitb][Vit2b][0][0] + Logb[0];
     V1b[Vitb][Vit2b][vCountb] = LogEb[Vitb][Vit2b][1][0] + Logb[1];

        Vt00b = LogTb[Vitb][0][0] + V0b[Vitb][Vit2b][vCountb];
        Vt10b = LogTb[Vitb][1][0] + V1b[Vitb][Vit2b][vCountb];
        Vt01b = LogTb[Vitb][0][1] + V0b[Vitb][Vit2b][vCountb];
        Vt11b = LogTb[Vitb][1][1] + V1b[Vitb][Vit2b][vCountb];

        if(Vt00b >= Vt10b)
        {
           MAX0b[vCountb] = Vt00b;
           ptrb[0][Vitb][Vit2b][vCountb] = '+';
        }
        else
        {
           MAX0b[vCountb] = Vt10b;
           ptrb[0][Vitb][Vit2b][vCountb] = '-';
        }

        if(Vt01b >= Vt11b)
        {
           MAX1b[vCountb] = Vt01b;
           ptrb[1][Vitb][Vit2b][vCountb] = '+';
        }
        else
        {
           MAX1b[vCountb] = Vt11b;
           ptrb[1][Vitb][Vit2b][vCountb] = '-';
        }

              //System.out.print(DNA1[Vit][Vit2][vCount] + " " + LogE[Vit][Vit2][0][0] + " " + Logb[Vit2] + " " + V0[Vit][Vit2][vCount]);
              //System.out.print(LogT[Vit][0][0] + " " + V0[Vit][Vit2][vCount]);
              //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
   }
   else
   {
        V0b[Vitb][Vit2b][vCountb] = VH1(LogEb[Vitb][Vit2b][0][vCountb], MAX0b[vCountb-1], vCountb);
        V1b[Vitb][Vit2b][vCountb] = VL1(LogEb[Vitb][Vit2b][1][vCountb], MAX1b[vCountb-1], vCountb);

        Vt00b = LogTb[Vitb][0][0] + V0b[Vitb][Vit2b][vCountb];
```

```
        Vt10b = LogTb[Vitb][1][0] + V1b[Vitb][Vit2b][vCountb];
        Vt01b = LogTb[Vitb][0][1] + V0b[Vitb][Vit2b][vCountb];
        Vt11b = LogTb[Vitb][1][1] + V1b[Vitb][Vit2b][vCountb];

        if(Vt00b >= Vt10b)
        {
           MAX0b[vCountb] = Vt00b;
           ptrb[0][Vitb][Vit2b][vCountb] = '+';
        }
        else
        {
           MAX0b[vCountb] = Vt10b;
           ptrb[0][Vitb][Vit2b][vCountb] = '-';
        }

        if(Vt01b >= Vt11b)
        {
           MAX1b[vCountb] = Vt01b;
           ptrb[1][Vitb][Vit2b][vCountb] = '+';
        }
        else
        {
           MAX1b[vCountb] = Vt11b;
           ptrb[1][Vitb][Vit2b][vCountb] = '-';
        }

      //System.out.print(Vt00 + " " + Vt10 + " " + Vt01 + " " + Vt11 + "\n");
        //System.out.print(LogT[Vit][0][vCount] + " " + V0[Vit][Vit2][vCount] + "\n");
   }
         //System.out.print(MAX0[vCount] + " " + MAX1[vCount] + "\n");
         //System.out.print(V0[Vit][Vit2][vCount]+ " " + V1[Vit][Vit2][vCount] + " " + DNA1[0][0][vCount] + "\n");
         //System.out.print(vCount + " " + ptr[0][vCount]+ " " + ptr[1][vCount] + "\n");

 vCountb++;
 }
Vit2b++;
//System.out.print("\n");
}
//System.out.print("\n");
 Vitb++;
}




// ***************************** Select pi*
int piCount1b = 0;
int piCount2b = 0;

while(piCount1b < nRep)
{
 piCount2b = 0;
 while(piCount2b < nTrain)
 {
  if(V0b[piCount1b][piCount2b][length[trialSeqValue[piCount1b][piCount2b]]-1] >
        V1b[piCount1b][piCount2b][length[trialSeqValue[piCount1b][piCount2b]]-1])
```

```
        MAXb[piCount1b][piCount2b] = '+';
   else
        MAXb[piCount1b][piCount2b] = '-';
    piCount2b++;
 }
 piCount1b++;
}




char[][][] Pathb = new char[nRep][nTrain][200000];

int pathCount1b = 0;
int pathCount2b = 0;
int pathCount3b = 0;


while(pathCount3b < nRep)
{
 pathCount2b = 0;
 while(pathCount2b < nTrain)
 {
   pathCount1b = length[trialSeqValue[pathCount3b][pathCount2b]]-1;
  while(pathCount1b >= 0)
  {
   if(pathCount1b == (length[trialSeqValue[pathCount3b][pathCount2b]]-1))
   {
        Pathb[pathCount3b][pathCount2b][pathCount1b] = MAXb[pathCount3b][pathCount2b];
   }
   else
   if(Pathb[pathCount3b][pathCount2b][pathCount1b + 1] == '+')
   {
      Pathb[pathCount3b][pathCount2b][pathCount1b] = ptrb[0][pathCount3b][pathCount2b][pathCount1b];
   }
   else
        Pathb[pathCount3b][pathCount2b][pathCount1b] = ptrb[1][pathCount3b][pathCount2b][pathCount1b];
 pathCount1b--;
  }
 pathCount2b++;
 }
 pathCount3b++;
}




int pathView1b = 0;
int pathView2b = 0;
int pathView3b = 0;

 while(pathView1b < nRep)
 {
  pathView2b = 0;
   while(pathView2b < nTrain)
   {
    pathView3b = 0;
```

```
    while(pathView3b < length[trialSeqValue[pathView1b][pathView2b]])
    {
     //System.out.print(Pathb[pathView1b][pathView2b][pathView3b] /*+ " " + MAXO[pathView] + "\n"*/);
     pathView3b++;
    }
       System.out.print("\n\n\n\n");
   pathView2b++;
  }
  pathView1b++;
 }
```

```
char[][][] symbolb = new char[nRep][nTrain][200000];
int[][] nb = new int[nRep][nTrain];
int[][][] Blocb = new int[nRep][nTrain][200000];
int[][][] Elocb = new int[nRep][nTrain][200000];


int n2b = 0;
int n3b = 0;

while(n2b < nRep)
{
 n3b = 0;
 while(n3b < nTrain)
 {
  nb[n2b][n3b] = nCpG[trialSeqValue[n2b][n3b]];
  //System.out.print("\n" + nb[n2b][n3b]);
  n3b++;
 }
n2b++;
}

int a3 = 0;
int n4b = 0;
int a13 = 0;

while(n4b < nRep)
{
 a13 = 0;
while(a13 < nTrain)
{
 a3 = 0;
 while(a3 < nb[n4b][a13])
 {
   for(int K3 = Blocb[n4b][a13][a3]; K3 <= Elocb[n4b][a13][a3]; K3++)
   {
        symbolb[n4b][a13][K3] = '+';

   }
  a3++;
 }
a13++;
}
```

```
 n4b++;
}


int b = 0;
int n5b = 0;
int n6b = 0;
double[][] nonGC = new double[nRep][nTrain];
double nonCount = 0;


while(n5b < nRep)
{
 n6b = 0;
 while(n6b < nTrain)
 {
  b = 0;
  while(b<length[trialSeqValue[n5b][n6b]])
  {
   if(symbolb[n5b][n6b][b] != '+')
   {
     symbolb[n5b][n6b][b] = '-';
   }
  //System.out.print(symbolb[n5b][n6b][b]);
  b++;
  }
 n6b++;
 }
n5b++;
}



int b12 = 0;
int b13 = 0;
int b14 = 0;
double[][] nMatch2 = new double[nRep][nTrain];

while(b13 < nRep)
{
 b14 = 0;
while(b14 < nTrain)
{
 b12 = 0;
 while(b12 < length[trialSeqValue[b13][b14]])
 {
   if(Pathb[b13][b14][b12] == symbolb[b13][b14][b12])
   {
        nMatch2[b13][b14]++;
   }
  b12++;
  }
 b14++;
 }
 b13++;
}
```

```
double[][] pMatch = new double[nRep][nTrain];
int n7b = 0;
int n8b = 0;

double pSUM = 0;

while(n7b < nRep)
{
 n8b = 0;
 while(n8b < nTrain)
 {
  pMatch[n7b][n8b] = nMatch2[n7b][n8b]/length[trialSeqValue[n7b][n8b]];
  pSUM = pSUM + pMatch[n7b][n8b];
  //System.out.print("\n\nViterbi Accuracy: " + pMatch[n7b][n8b]);
 n8b++;
 }
n7b++;
}


int printCount1 = 0;
int printCount2 = 0;

while(printCount1 < nRep)
{
 printCount2 = 0;
 while(printCount2 < nTrain)
 {

  PrintWriter pw2 = new PrintWriter(new FileWriter("Results.txt", true));
  pw2.println(printCount1 + " - " + printCount2 + ": " + pMatch[printCount1][printCount2]);
  pw2.println();
  pw2.close();

 printCount2++;
 }
printCount1++;
}


// ************************************* E N D M A I N
}




//******************************* T R A I N I N G T R A N S I T I O N


public static double pr(int T2, int T3)
{
       double tState = 0;

       if(T2 == 0 && T3 == 0)
```

```
          tState = .99998;
       else if(T2 == 0 && T3 == 1)
          tState = .00002;
       else if(T2 == 1 && T3 == 0)
          tState = .0001;
       else if(T2 == 1 && T3 == 1)
          tState = .9999;

       return (tState);
}

//******************************** T R A I N I N G E M I S S I O N


public static double emiss2(char predDNA, int g)
{
              double emST2 = 0;


                        if(predDNA == 'A')
                        {
                            if(g==0)
                            emST2= .15;
                            else
                            emST2 = .25;
                        }
                        else if(predDNA == 'C')
                        {
                            if(g==0)
                            emST2 = .35;
                            else
                            emST2 = .25;
                        }
                        else if(predDNA == 'G')
                        {
                            if(g==0)
                            emST2 = .35;
                            else
                            emST2 = .25;
                        }
                        else if(predDNA == 'T')
                        {
                            if(g==0)
                            emST2 = .15;
                            else
                            emST2 = .25;
                        }



       return (emST2);
}


//******************************** P R E D I C T I O N T R A N S I T I O N
```

```
public static double pr2(int T2, int T3, double a00, double a01, double a10, double a11)
{
       double tState = 0;

       if(T2 == 0 && T3 == 0)
          tState = a00;
       else if(T2 == 0 && T3 == 1)
          tState = a01;
       else if(T2 == 1 && T3 == 0)
          tState = a10;
       else if(T2 == 1 && T3 == 1)
          tState = a11;

       return (tState);
}

//****************************** P R E D I C T I O N E M I S S I O N


public static double emiss3(char predDNA, int g, double cA, double cC, double cG, double cT, double nA, double nC, double nG, double nT)
{
              double emST2 = 0;


                        if(predDNA == 'A')
                        {
                            if(g==0)
                            emST2= cA;
                            else
                            emST2 = nA;
                        }
                        else if(predDNA == 'C')
                        {
                            if(g==0)
                            emST2 = cC;
                            else
                            emST2 = nC;
                        }
                        else if(predDNA == 'G')
                        {
                            if(g==0)
                            emST2 = cG;
                            else
                            emST2 = nG;
                        }
                        else if(predDNA == 'T')
                        {
                            if(g==0)
                            emST2 = cT;
                            else
                            emST2 = nT;
                        }



       return (emST2);
}
```

```
//************************ V I T E R B I A L G O R I T H M

public static double VH1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VL1(double E, double MAX, int i)
{
   double q = E + MAX;
   return q;
}
public static double VH2(double E, double Vl, double T, int i)
{
   double q = E + Vl + T;
   return q;
}
public static double VL2(double E, double Vh, double T, int i)
{
   double q = E + Vh + T;
   return q;
}

//************************* P R I N T T O F I L E Transition


public void writeLinesToFileT(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Transition:\n" + "A00: " + gc1);
          pw.println("Transition:\n" + "A01: " + gc2);
          pw.println("Transition:\n" + "A10: " + gc3);
          pw.println("Transition:\n" + "A11: " + gc4);
          arrays++;
          pw.println(" ");
          pw.flush();
}
```

```
//************************* P R I N T T O F I L E Emission


public void writeLinesToFileE(String filename, boolean appendToFile, double gc1, double gc2, double gc3, double gc4, double gc5, double gc6,
      double gc7, double gc8) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));

   }

          pw.println("Emiss CG: " + gc1 + " " + gc2 + " " + gc3 + " " + gc4);
          pw.println("Emiss NON: " + gc5 + " " + gc6 + " " + gc7 + " " + gc8);
          arrays++;
          pw.println(" ");
          pw.flush();
}



//************************* P R I N T T O F I L E


public void writeLinesToFile(String filename, boolean appendToFile, double gc1, int i) throws IOException
{

    PrintWriter pw = null;
    int arrays = 0;

   if (appendToFile)
   {

       //If the file already exists, start writing at the end of it.
       pw = new PrintWriter(new FileWriter(filename, true));
   }
   else
   {

       pw = new PrintWriter(new FileWriter(filename));
       //this is equal to:
       //pw = new PrintWriter(new FileWriter(filename, false));
```

```
  }

        pw.println(i + " " + gc1);
        arrays++;
        pw.println(" ");
        pw.flush();
}


}
```

Appendix E

Random number generator (R)

```
# * * * * * * * * * * S A M P L E
W1 = 0:19
rep = 5
s = matrix(c(0),10,rep)
testVal = matrix(c(0),10,rep)

for(i in 1:rep)
{
   sW1 = sort(sample(W1,10))
   s[,i] = sW1
   write.infile(sort(s[,i]), "Rprint.txt", append = TRUE)
   W2 = W1[-(s[,i]+1)]
   x = sample(W2,10)
   testVal[,i] = x
   write.infile(sort(testVal[,i]), "trialSeqNUMBER.txt", append = TRUE)
}
```

Curriculum Vitae

Education

University of Texas at El Paso, El Paso, Texas, USA
- M.S., Statistics, May 2011

University of Texas at El Paso, El Paso, Texas, USA
- B.S., Psychology, December, 2008

Robert Ortega was born on December 22, 1984 to Bobby and Kathy Ortega in El Paso, TX. He attended Maxine Silva Health Magnet Highschool for Healthcare Professions, and graduated in May, 2003. While studying for his M.S., Dr. Ming-Ying Leung, served as his thesis advisor.