Spring 5-23-2016

Multiple Sequence Alignment with Pro le Hidden Markov Models

Shubhangi Rakhonde
San Jose State University

Multiple Sequence Alignment with Profile Hidden Markov Models

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Shubhangi Rakhonde

May 2016

The Designated Project Committee Approves the Project Titled

Multiple Sequence Alignment with Profile Hidden Markov Models

by

Shubhangi Rakhonde

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2016

Dr. Sami Khuri	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Ms. Vidya Rangasayee	Department of Computer Science

ABSTRACT

Multiple Sequence Alignment with Profile Hidden Markov Models

by Shubhangi Rakhonde

The human genome consists of various patterns and sequences that are of biological significance. Capturing these patterns can help us in resolving various mysteries related to the genome, like how genomes evolve, how diseases occur due to genetic mutation, how viruses mutate to cause new disease and what is the cure for these diseases. All these applications are covered in the study of bioinformatics.

One of the very common tasks in bioinformatics involves simultaneous alignment of a number of biological sequences. In bioinformatics, this is widely known as Multiple Sequence Alignment. Multiple sequence alignments help in grouping together organisms with the same evolutionary history. They also help in learning properties of a new sequence by aligning it with previously studied homologous sequences.

This project covers probabilistic modeling method to perform multiple sequence alignment (MSA). Use of hidden Markov models in MSA significantly improves computational speed especially for sequences that contain overlapping regions. We used Baum-Welch expectation maximization algorithm to train hidden Markov models and Viterbi algorithm to align the sequences. Our results are comparable to the ones obtained by publicly available packages like ClustalW and Clustal Omega.

ACKNOWLEDGMENTS

I want to thank my project advisor Dr. Sami Khuri for his valuable guidance and continuous encouragement. I would like to also thank my committee members, Dr. Chris Pollett and Ms. Vidya Rangasayee for their time and support.

Also, I would like to thank my husband Harshay Buradkar for standing behind me all the time and providing support and encouragement.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction to Multiple Sequence Alignment (MSA)

With evolution, new biological sequences get derived from old sequences. Some residues get conserved during this process. With multiple sequence alignment, these conserved residues are aligned together in columns. With alignment, studying the sequences becomes easy. New sequences can be studied by aligning them with homologous sequences that are already known. Also, this alignment helps in grouping together organisms with a same evolutionary relationship.

Multiple sequence alignment initially may look similar to simple text alignment where same characters are aligned in columns. However, in biology, this process can be more complex. Aligning characters that are structurally and evolutionary similar but not identical, is more challenging than text alignment. These sequences also undergo some insertions and deletions during evolution. Also, these biological sequences and patterns are so long that aligning them manually is impossible. Methods to do such alignments have been an extensive area of research and various methods have been devised for e.g. dynamic programming, progressive alignment construction, consensus methods, probabilistic models, etc. These methods help in the alignment automatically and more accurately and more efficiently when compared to manual methods.

In this project, a probabilistic method of multiple alignment is studied and implemented. Profile hidden Markov models are used to produce the alignment. They give most likelihood MSA or set of possible MSAs by assigning likelihoods to all possible combinations of gaps, matches and mismatches. [2]

The remainder of the report is organized as follows. Chapter 2 and 3 discusses the basics of hidden Markov models and profile hidden Markov models. The main steps for this particular project are the initialization of the model, training it using Baum-Welch algorithm and alignment using Viterbi algorithm. The training step has the following sub-steps, forward algorithm, backward algorithm and re-estimation. Chapter 4 covers the forward and backward algorithm. Chapter 5 covers Viterbi algorithm and Chapter 6 has an example of computing the most likely path using the Viterbi algorithm. Chapter 7 has detailed explanation and implementation of Baum-Welch algorithm. Results obtained from this training are summarized in Chapter 8. The report is concluded by future work and enhancements in Chapter 9.

CHAPTER 2

Introduction to Hidden Markov Models (HMM)

Hidden Markov models(HMM) are statistical Markov models in which the system being modeled is assumed to be Markov models with hidden states. [2]

2.1 Toy HMM by Sean R. Eddy

The best way to see how HMM work is to consider an example. Consider the example of an HMM in Bioinformatics that Sean R Eddy provided in 'What is a Hidden Markov Model?' [3] Eddy's toy hidden Markov model is a good example to understand HMMs.

2.1.1 Toy HMM 5' Splice Site Recognition

Assume we have been given a DNA that starts with an exon and ends with an intron with one 5 splice site between them. The problem is to find the exact position of 5 splice site. These 3 labels show that we should have 3 states in our model. The 3 states are Exon (E) state, Intron (I) State and 5 splice site (5) state.

Number of states, N = 3.

Set of States, Q = {E, 5, I}.

Every label has nucleotide base A, T, G, or C; hence,

Number of symbols M = 4.

Set of Symbols V = {A, T, G, C}.

Based on the statistical properties of E, I and 5 labels, the emission probabilities

(the probability of emitting the respective symbol) are given in Table 1.

Table 1: Emission Probabilities of 3 states of Eddy Toy Model

Symbol(V)	Exon State(E)	5'SS State (5)	Intron State (I)
A	0.25	0.05	0.4
C	0.25	0	0.1
G	0.25	0.95	0.1
T	0.25	0	0.4

The transition probabilities (probabilities of moving from one state to a new state) are given in Table 2. These probabilities are based on a linear order in which we want these states to occur i.e. one or more E, one 5, and one or more I.

Table 2: Transition Probabilities of 3 states of Eddy Toy Model

From / To	Exon State(E)	5'SS State (5)	Intron State (I)
Exon State (E)	0.9	0.1	0
5'SS State (5)	0	0	1
Intron State (I)	0	0	0.9

The model(λ) is given in Figure 1. [3]

2.2 What is Hidden Markov Model

Consider the observed sequence O = CTTCATGTGAAAGCAGACGTAAGTCA of length 26. The question is to determine the sequence of states that might have produced this observed sequence. Potentially, there could be many state paths that generate the same sequence. The sequence of states in not known, it is 'hidden'. And as the state sequence is nothing but a Markov chain, we call the model as Hidden Markov Model.

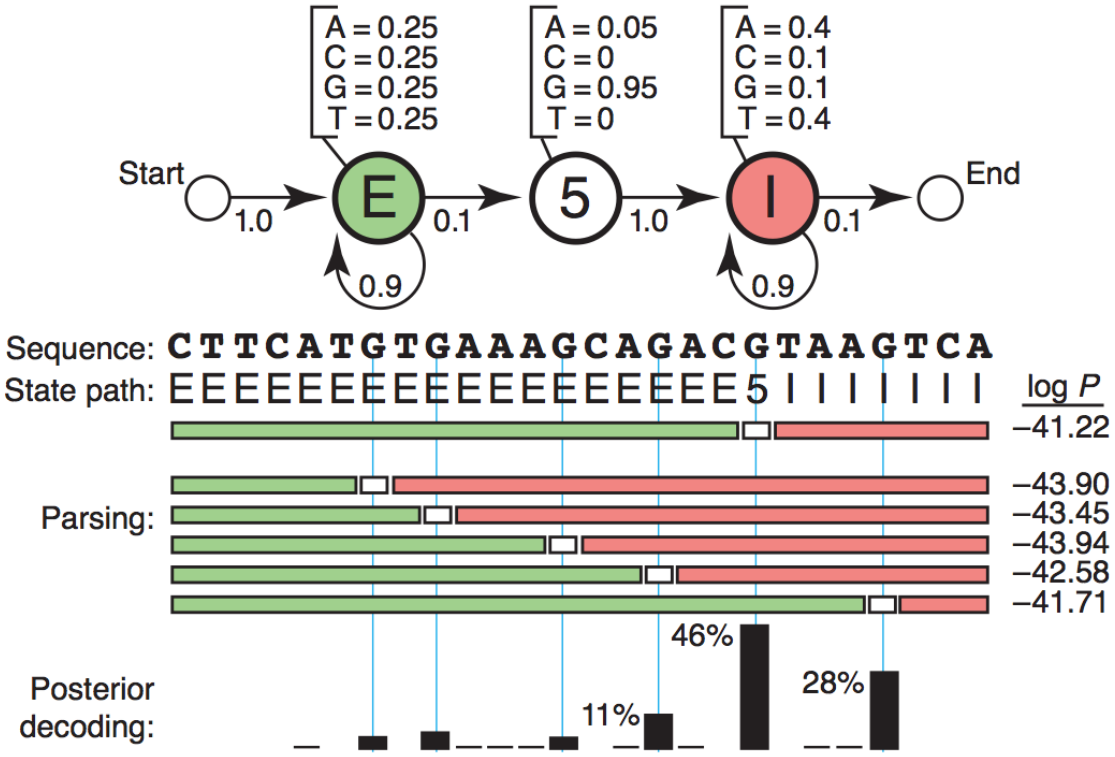

Figure 1: A Toy HMM of 5' Splice Site Recognition

2.3 Finding best State Path

As can be seen in Figure 1, state 5 emits only As and Gs. And the observed sequence O has a total of 14 As and Gs (do not count the last A, as it belongs to state I). From this, we can derive that there can be 14 possible state paths from the model, out of which only one will be the best and will have the highest probability. Let's calculate the probability of the first state path shown in Figure 1 under 'State

path'.

$$
\begin{aligned}
P(O|\lambda) &= P(CTTCATGTGAAAGCAGACGTAAGTCA) \\
&= P(EEEEEEEEEEEEEEEEEEE5IIIIIII) \\
&= 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 \\
&\quad * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 \\
&\quad * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 * 0.9 * 0.25 \\
&\quad * 0.9 * 0.25 * 0.1 * 0.95 * 1.0 * 0.4 * 0.9 * 0.4 * 0.9 * 0.4 * 0.9 * 0.1 \\
&\quad * 0.9 * 0.4 * 0.9 * 0.1 * 0.9 * 0.4 * 0.1 \\
&= 1.2546467e - 18 \\
Ln(P) &= -41.22.
\end{aligned} \tag{1}
$$

Similarly, we calculated all the probabilities of sequence across all paths. These probabilities are given in Table 3.

Table 3: Log Probabilities of 14 potential state path sequences

No.	State Path Sequences	Probability	Ln(P)	Posterior Decoding
1	EEEE5IIIIIIIIIIIIIIIIIII	2.90421E-21	-47.29	0.11
2	EEEEEE5IIIIIIIIIIIIIIIII	8.62187E-20	-43.90	3.20
3	EEEEEEEE5IIIIIIIIIIIIIII	1.34717E-19	-43.45	4.99
4	EEEEEEEEE5IIIIIIIIIIIIII	4.43147E-21	-46.87	0.16
5	EEEEEEEEEE5IIIIIIIIIIIII	2.76967E-21	-47.36	0.10
6	EEEEEEEEEEE5IIIIIIIIIIII	1.73104E-21	-47.81	0.06
7	EEEEEEEEEEEE5IIIIIIIIIII	8.22245E-20	-43.94	3.05
8	EEEEEEEEEEEEEE5IIIIIIIII	6.0857E-22	-48.85	0.02
9	EEEEEEEEEEEEEEE5IIIIIIII	2.89071E-20	-44.99	1.07
10	EEEEEEEEEEEEEEEE5IIIIIII	9.5089E-22	-48.40	0.04
11	EEEEEEEEEEEEEEEEEE5IIIIII	1.25465E-18	-41.22	46.50
12	EEEEEEEEEEEEEEEEEEEE5IIIII	2.57945E-20	-45.10	0.96
13	EEEEEEEEEEEEEEEEEEEEE5IIII	3.0631E-19	-42.63	11.35
14	EEEEEEEEEEEEEEEEEEEEEE5III	7.65776E-19	-41.71	28.38

As can be seen in Table 3, the 11th state path is the most likely path as it has the highest probability. However, the brute force approach shown in Table 3 is not practical for more complex problems. Hence, Viterbi algorithm is used to get the most likely state path.

Posterior decoding helps in finding the most likely path and it is computed as the ratio of the current path over all possible paths. Posterior decoding for path 11 is given by:

$$\frac{Probability\ of\ path\ 11}{Sum\ of\ probabilities\ of\ 14\ paths} * 100 = 46.50.$$

The higher the posterior decoding the better is the path. For many complex problems, it is impossible to enumerate and compute all state paths by using the brute force approach, hence, posterior decoding uses dynamic programming algorithms, namely, forward and backward algorithms.

2.4 Elements of HMM

The following are the elements of HMM [5]:

- Number of hidden states: N

  N represents a total number of states present in an HMM. N is 3 for HMM given in Figure 1.

- Set Q of N states, $Q = \{1,\ 2,\ 3, ..., N\}$

  States for Toy HMM of Figure 1 are represented in the same format.

  Q = {E, 5, I}.

- An observed sequence, $O = (O_1,\ O_2,\ O_3,\ ...,\ O_T)$ where O_T is the observation

at time T.

- An unobservable state sequence, $q = (q_1,\ q_2,\ q_3,\ ..., q_T)$

  where, q_T emits observed character O_T at time T.

- Number of symbols: M

  M is number of possible symbols that can be emitted by an HMM. For HMM in Figure 1, this value is 4 as this model is emitting nucleotide bases. In case of an HMM for protein sequences, this value will be 20 that is a total number of amino acids.

- Set of symbols, $V = \{1, 2, 3, ..., M\}$

  If an HMM for nucleic bases is created, then set of symbols is,

  $V = \{A,\ T,\ G,\ C\}$.

  If an HMM for proteins is created, then set of symbols is,

  $V = \{A,\ R,\ N,\ D,\ C,\ Q,\ E,\ G,\ H,\ I,\ L,\ K,\ M,\ F,\ P,\ S,\ T,\ W,\ Y,\ V\}$.

- State transition probability matrix: A

  $a_{ij}\ =\ P(q_{t+1}\ =\ j \mid q_t\ =\ i) \quad where\ 1 <=\ i;\ j\ <=\ N$

  a_{ij} is the probability for making transition from state i to state j.

- Emission probability distribution: B

  k is a symbol.

  $b_j(k)\ =\ P(O_t\ =\ k \mid q_t\ =\ j) \quad where\ 1\ <=\ i\ ;\ j\ <=\ M$

  $b_j(k)$ is the probability of emitting symbol k by state j at time t.

- The initial State Distribution: Π

  $\Pi_i\ =\ P(q_1\ =\ i)$

  This represents the probability of system starting from state i.

- The entire model $\lambda = (A,\ B,\ \Pi)$

  Three probability measures a_{ij}, $b_j(k)$ and Π_i represents the entire model λ.

2.5 Applications of HMMs

Here are some of the common and more popular application areas where HMM is used [5].

1. Generating Multiple Sequence Alignment

   Using profile HMMs, alignment of DNA or protein sequences can be performed.

2. Modeling protein families and DNA

   During cell reproduction, biological sequences get duplicated in daughter cells. Due to some errors, insertions or deletions during this process, they are not the exact replica of each other. However, their structure remains similar to some primary structure. Grouping of such sequences with structural similarities can be done using HMMs [1].

3. Gene Prediction

   A long sequence of DNA has an unknown number of genes in it. These genes can be identified by tracking various patterns that are commonly found. An HMM is trained for these patterns is used to predict the genetic regions from long DNA strand.

2.6 Three types of problem solved by HMM

2.6.1 Evaluation

Given a model $\lambda = (A,\ B,\ \Pi)$ and a sequence of observations $O = (O_1,\ O_2,\ O_3, ...,\ O_T)$, efficiently compute the probability that the observed sequence

is produced by the model. Hidden states in model complicate the evaluation process. In a case where we are dealing with several models for one pattern, it helps **to choose the best model** among a few competing models. This is helpful in scoring a sequence. The problem is solved using the forward or the backward algorithm.

2.6.2 Decoding

Given a model $\lambda = (A,\ B,\ \Pi)$ and a sequence of observations $O = ((O_1,\ O_2,\ O_3, ...,\ O_T))$, find the optimal state sequence $Q = (q_1,\ q_2, ...,\ q_T)$. Here the optimality criterion we considered is maximum likelihood. With this problem, we try **to find the most likely hidden path** of the model. Viterbi algorithm is used to find the maximum likelihood path.

2.6.3 Learning

Given the sequence of observations $O = (O_1,\ O_2,\ O_3, ...,\ O_T)$, **estimate the model parameters** $\lambda = (A,\ B,\ \Pi)$ to maximize $P(O|\lambda)$. The sequences used to adjust the model parameters are called training sequences. Training the model is a crucial process and this is achieved with Baum - Welch Algorithm that uses the expectation-maximization(EM) algorithm.

2.7 Evaluation explained in detail

Solution: The brute force solution to this problem is the summation of probabilities over all paths $q = (q_1,\ q_2, ...,\ q_T)$ that gives O. This solution has very high complexity which is equal to $(2T.N^T)$, where N is number of possible states and T is the length of the observed sequence. Even the small values of T and N make the computation infeasible. Hence, we use dynamic programming technique to efficiently

compute probability. We use the forward or backward algorithms as follows:

2.7.1 Forward Algorithm

Define the forward variable $\alpha_t(i)$ as:

$$\alpha_t(i) \ = \ P(O_1,\ O_2,\ O_3, ...,\ O_t,\ q_t = i \mid \lambda)$$

It is the probability of the partial observation sequence, $(O_1,\ O_2,\ O_3, ..., O_t)$(until time t) and state S_t at time t, given the model λ. $\alpha_t(i)$ can be solved inductively as given in equation (2):

$$
\begin{aligned}
&Initialization: \\
&\qquad \alpha_1(i) \ = \ \Pi_i b_i(O_1), \qquad\qquad 1 \ <= i <= N. \\
&Induction: \\
&\qquad \alpha_{t+1}(j) \ = \ \big[\sum_{i=1}^{N} \alpha_t(i).a_{ij}\big].b_j(O_{t+1}), \ \ 1 \ <= t <= T \ - \ 1, \ \ 1 \ <= \ j \ <= \ N. \\
&Termination: \\
&\qquad P(O|\lambda) \ = \ \sum_{i=1}^{N} \alpha_T(i)
\end{aligned}
\tag{2}
$$

The forward probabilities are initialized as the joint probabilities of every state S_i and the initial observation O_1. The induction function calculates the probability of the joint event that observed sequence $(o_1,\ o_2,\ o_3, ...,\ o_t)$ is observed and the state S_j is reached from a state S_i at time $t + 1$. This induction step is the key step of the forward algorithm. Finally, the termination step computes the total probability by summing up all forward variables of all states at time $t \ = \ T$.

2.7.2 Backward Algorithm

Similar to forward algorithm, here we consider the backward variable for state i as $\beta_t(i) = P(O_{t+1}O_{t+2}...O_T \mid q_t = S_i, \lambda)$ as the probability of the partial observed sequence from $t+1$ to the end T i.e. $(O_{t+1}, O_{t+2}, O_{t+3}, ..., O_T)$ knowing that we land in state i at time t.

$$
\begin{aligned}
&Initialization: \\
&\qquad \beta_T(i) = 1, \qquad\qquad 1 <= i <= N. \\
&Induction: \\
&\qquad \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \\
&\qquad\qquad\qquad\qquad t = T-1, T-2, ..., 1, \\
&\qquad\qquad\qquad\qquad 1 <= i <= N.
\end{aligned}
\tag{3}
$$

The initialization step initializes the value of the backward variable as 1 for all states S_i. Starting with the final observation, step 2 goes on calculating the probabilities backwards at each state by adding previous observations. And the termination step adds up the probabilities to get the final probability that the given model emits the sequence. Similar to the forward algorithm, the backward algorithm requires the order of N^2T calculations. Recall, either forward or backward algorithm can be used to solve problem 1.

2.8 Decoding explained in detail

The most likely path can be found using a dynamic programming technique - Viterbi algorithm. The Viterbi algorithm is a modified version of the forward al-

gorithm that was used for solving problem 1. Instead of adding probabilities of all possible paths, Viterbi algorithm calculates the probability across the best path and then traces back to get the most likely path.

2.8.1 Viterbi Algorithm

Define $\delta_t(i)$ as the highest probability among the best path at time t.

$$
\begin{aligned}
&Initialization: \\
&\qquad \delta_1(i) = \Pi_i b_i(o_1), \qquad 1 < i < N \\
&\qquad \psi_1(i) = 0 \\
&Recursion: \\
&\qquad \delta_t(j) = max_{1<=i<=N}[\delta_{t-1}(i)a_{ij}]b_j(O_t) \\
&\qquad \psi_t(j) = argmax_{1<=i<=N}[\delta_{t-1}(i)a_{ij}] \\
&\qquad\qquad\qquad 2 <= t <= T, \\
&\qquad\qquad\qquad 1 <= j <= N \\
&Termination: \\
&\qquad P_T^* = max_{1<=i<=N}[\delta_T(i)] \\
&\qquad q_T^* = argmax_{1<=i<=N}[\delta_T(i)] \\
&\qquad where \\
&\qquad P_T^* = P(q_1,\ q_2, ..., q_T|O, \lambda)
\end{aligned}
\tag{4}
$$

2.9 Learning explained in detail

2.9.1 Baum-Welch Algorithm

In problem 3, we have to estimate the model parameters $\lambda = (A,\ B,\ \Pi)$ with given sequence of observations $O = (O_1,\ O_2,\ O_3, ..., O_T)$, to maximize $P(O|\lambda)$. There is neither any analytical method nor any optimal solution present to estimate these parameters, hence, it is the most difficult among the 3 problems. However, we can use probabilistic parameter estimation to locally maximize the probability using either iterative or gradient technique. Baum-Welch Expectation Maximization is one of the iterative procedure to solve problem 3.

Baum-Welch Algorithm

- Initialization:

  Initialize model parameters.

- Recurrence:

  For each sequence :

  - Compute forward matrix
  - Compute backward matrix
  - Compute expectation counts for emission and transition

- Update the model parameters.
- Compute log likelihood of the new model.
- Terminate if stopping criterion is reached.

This concludes the overview of hidden Markov model and the next chapter will cover profile HMMs and Baum-Welch algorithm for profile HMM in detail.

CHAPTER 3

Profile Hidden Markov Models (PHMM)

An HMM architecture that has well represented the profiles of multiple alignments is profile Hidden Markov Model(PHMM). This model is a strongly linear model. It has three states as explained in the next section.

3.1 States in Profile HMM

The three states in profile HMM are match, insert and delete. Most of the columns of the multiple sequence alignment are assigned to match states. Each match state has an emission distribution that reflects the probability of observing a given character in that position. Each match state is also accompanied by two other states. The first one is called a delete state that emits nothing and which allows a column to be skipped. This deletion is relative to the consensus observed. And the second one is called an insert state that exists between each pair of consecutive match states. It has a state transition to itself, which allows one or more characters to be inserted at any point relative to the consensus observed. [12]

Figure 2 shows a sample profile HMM model with match, insert and delete states represented by M, I and D, respectively.

The three types of problems solved by HMM can also be tackled using profile HMM. The next few chapters cover these three problems in detail.

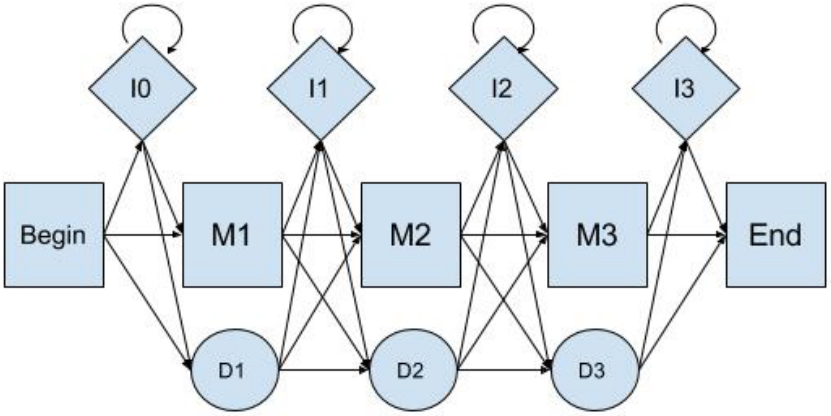

Figure 2: Full structure of Profile HMM

CHAPTER 4

Evaluation

Evaluation is efficiently computing the probability of observed sequence emitted by the model when an observed sequence and a model are given.

For example, consider the following aligned DNA sequences.

$$\begin{aligned} O_1 &= \texttt{G C A G} \\ O_2 &= \texttt{G - - G} \\ O_3 &= \texttt{G - A G} \\ O_4 &= \texttt{G C T G} \\ O_5 &= \texttt{A - A C} \\ O_6 &= \texttt{G - A C} \\ O_7 &= \texttt{G - G G} \\ O_8 &= \texttt{A - A C} \end{aligned} \tag{5}$$

Given observed sequence O = `GCCAG`, we want to compute the probability of O being emitted by the model. First, we will estimate the model parameters from given aligned sequences. One heuristic rule to consider here is that alignment column with a fraction of gap symbols below 0.5 corresponds to a match state of a profile HMM. Hence, considering columns 1, 3 and 4 as match states and column 2 as insert state, we get emission counts for match states and insert states as given in Table 4 and the transition counts as given in Table 5.

By calculating the emission and transition probabilities from the above counts, we get Table 6 and Table 7.

Table 4: Emission Count Matrix

i	O_i	M_0	M_1	M_2	M_3		I_0	I_1	I_2	I_3
0	A	-	2	5	0		0	0	0	0
1	C	-	0	0	3		0	2	0	0
2	G	-	6	1	5		0	0	0	0
3	T	-	0	1	0		0	0	0	0

Table 5: Transition Count Matrix

i	X-TO-X	0	1	2	3
0	M-M	8	5	7	8
1	M-D	0	1	0	-
2	M-I	0	2	0	0
3	I-M	0	2	0	0
4	I-D	0	0	0	-
5	I-I	0	0	0	0
0	D-M	-	0	1	0
1	D-D	-	0	0	-
2	D-I	-	0	0	0

Based on above probabilities, the state diagram for the given model is given in Figure 3.

When data is finite, the events that are not observed at all in the dataset, get zero probability. To avoid the problem of zero probabilities, various methods are used.

Table 6: Emission Probability Matrix

i	O_i	M_0	M_1	M_2	M_3		I_0	I_1	I_2	I_3
0	A	-	2/8	5/7	0		0	0	0	0
1	C	-	0	0	3/8		0	2/2	0	0
2	G	-	6/8	1/7	5/8		0	0	0	0
3	T	-	0	1/7	0		0	0	0	0

Table 7: Transition Probability Matrix

i	X-TO-X	0	1	2	3
0	M-M	8/8	5/8	7/7	8/8
1	M-D	0	1/8	0	-
2	M-I	0	2/8	0	0
3	I-M	0	2/2	0	0
4	I-D	0	0	0	-
5	I-I	0	0	0	0
0	D-M	-	0	1/1	0
1	D-D	-	0	0	-
2	D-I	-	0	0	0

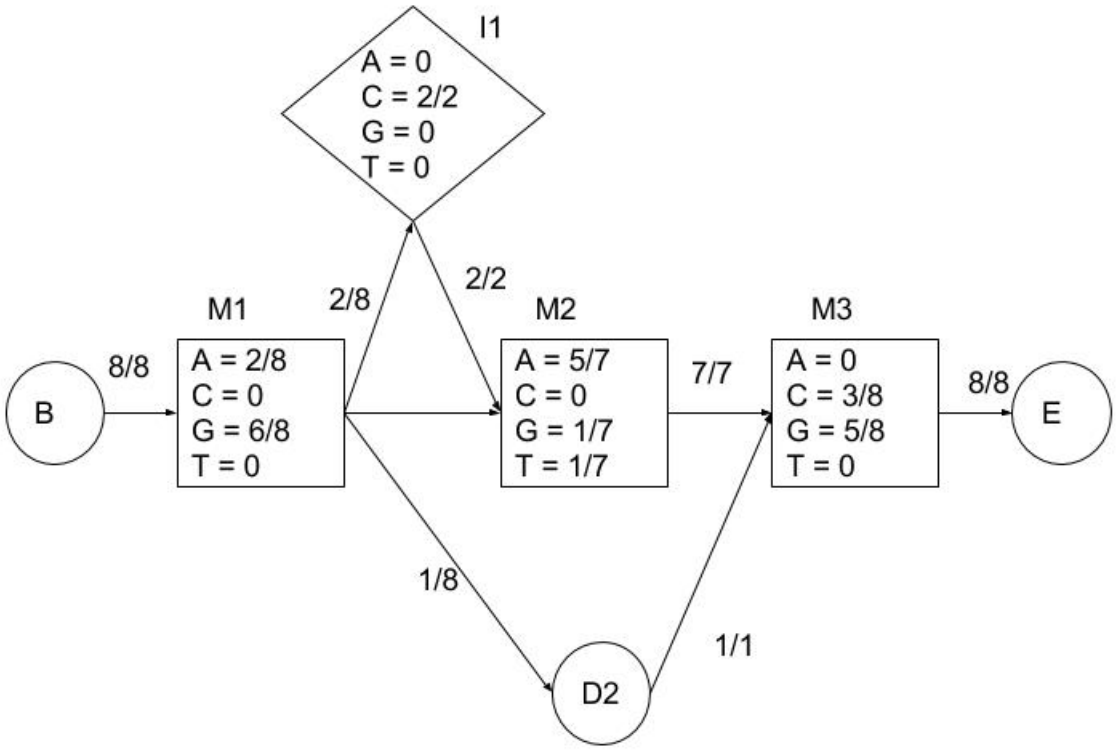

Figure 3: State Diagram for DNA sequence Evaluation Example

Adding a constant to all the counts is the simplest method among all. This constant is known as pseudocount. When this pseudocount is 1, it is known as "Laplace's rule". After adding the constant, the updated emission and transition probabilities are now given in Table 8 and Table 9.

Table 8: Emission Probability Matrix (With pseudocounts)

i	O_i	M_0	M_1	M_2	M_3		I_0	I_1	I_2	I_3
0	A	-	3/12	5/11	1/12		1/4	1/6	1/4	1/4
1	C	-	1/12	1/11	4/12		1/4	3/6	1/4	1/4
2	G	-	7/12	2/11	6/12		1/4	1/6	1/4	1/4
3	T	-	1/12	2/11	1/12		1/4	1/4	1/4	1/4

Table 9: Transition Probability Matrix (With pseudocounts)

i	X-TO-X	0	1	2	3
0	M-M	9/11	6/11	8/10	9/10
1	M-D	1/11	2/11	1/10	-
2	M-I	1/11	3/11	1/10	1/10
3	I-M	1/3	3/5	1/3	1/2
4	I-D	1/3	1/5	1/3	-
5	I-I	1/3	1/5	1/3	1/2
0	D-M	-	1/3	1/4	1/2
1	D-D	-	1/3	1/4	-
2	D-I	-	1/3	1/4	1/2

Hence, the final state diagram is given in Figure 4.

This represents our final model. In order to calculate the probability of the observed sequence O, we can use either of the following two algorithms.

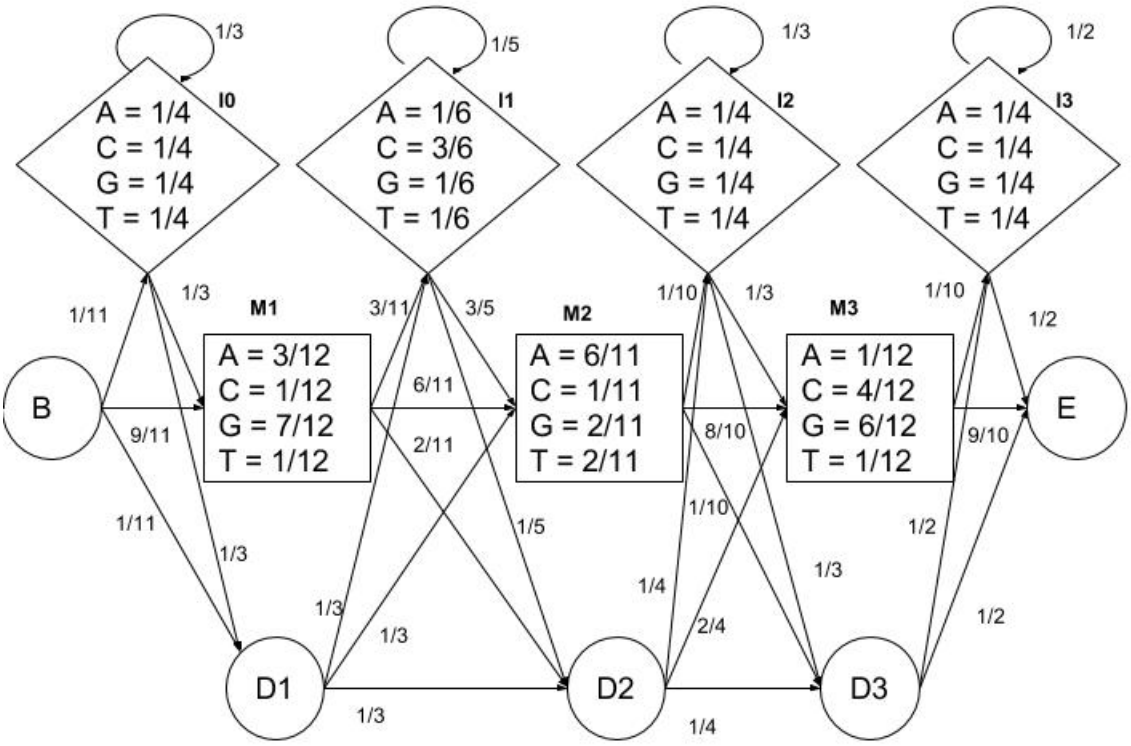

Figure 4: Final State Diagram

4.1 Forward Algorithm

The pseudo code for forward algorithm is given below [4].

Initialization :

$$f_0^M(0) \;=\; 1$$

Recursion :

$$\begin{aligned}
f_k^M(i) &= e_{M_k}(x_i).\left[f_{k-1}^M(i-1).a_{M_{k-1}M_k} \;+\; f_{k-1}^I(i-1).a_{I_{k-1}M_k} \;+\; f_{k-1}^D(i-1).a_{D_{k-1}M_k}\right] \\
f_k^I(i) &= e_{I_k}(x_i).\left[f_k^M(i-1).a_{M_kI_k} \;+\; f_k^I(i-1).a_{I_kI_k} \;+\; f_k^D(i-1).a_{DkI_k}\right] \\
f_k^D(i) &= \left[f_{k-1}^M(i).a_{M_{k-1}D_k} \;+\; f_{k-1}^I(i).a_{I_{k-1}D_k} \;+\; f_{k-1}^D(i).a_{D_{k-1}D_k}\right]
\end{aligned}$$

Termination :

$$f_{m+1}^M(L+1) \;=\; \left[f_m^M(L).a_{M_mM_{m+1}} \;+\; f_m^I(L).a_{I_mM_{m+1}} \;+\; f_m^D(L).a_{D_mM_{m+1}}\right] \tag{6}$$

I have implemented the forward algorithm, as explained in equations (6) and used it to compute the probability of the model for generating the sequence : `GCCAG`. This same forward algorithm implementation is later used in Baum-Welch algorithm which is used for training the model with input sequences. This is explained in detail in Chapter 7.

Applying this algorithm, we get a forward matrix as Table 10.

Table 10: Forward Matrix for `GCCAG`

State	-	G	C	C	A	G
M0	1.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
M1	0.00E+00	4.77E-01	6.31E-04	5.26E-05	1.32E-05	2.56E-06
M2	0.00E+00	5.51E-03	2.42E-02	3.70E-03	2.28E-03	2.72E-05
M3	0.00E+00	7.58E-03	1.67E-02	9.39E-03	3.59E-04	9.80E-04
I0	0.00E+00	2.27E-02	1.89E-03	1.58E-04	1.32E-05	1.10E-06
I1	0.00E+00	5.05E-03	6.69E-02	6.88E-03	2.35E-04	8.66E-06
I2	0.00E+00	1.89E-03	5.94E-03	1.96E-03	3.43E-04	8.86E-05
I3	0.00E+00	9.47E-04	3.28E-03	1.81E-03	6.32E-04	1.32E-04
D1	9.09E-02	7.58E-03	6.31E-04	5.26E-05	4.38E-06	3.65E-07
D2	3.03E-02	9.03E-02	1.37E-02	1.40E-03	5.08E-05	2.32E-06
D3	7.58E-03	2.38E-02	7.82E-03	1.37E-03	3.55E-04	3.28E-05

Using termination step given in the forward algorithm (6) the final probability of O emitted by given model is 0.00096.

This evaluation problem can also be solved using the backward algorithm instead of the forward algorithm.

4.2 Backward Algorithm

The pseudo code for backward algorithm is given below [4].

Initialization :

$$
\begin{aligned}
b^M_{M+1}(L+1) &= 1; \\
b^M_M(L) &= a_{M_M M_{M+1}}; \\
b^I_M(L) &= a_{I_M M_{M+1}}; \\
b^D_M(L) &= a_{D_M M_{M+1}};
\end{aligned}
$$

Recursion :

$$
\begin{aligned}
b^M_k(i) &= e_{M_{k+1}}(x_{i+1}).b^M_{k+1}(i+1).a_{M_k M_{k+1}} + e_{I_k}(x_{i+1}).b^I_k(i+1).a_{M_k I_k} + b^D_{k+1}(i).a_{M_k D_{k+1}} \\
b^I_k(i) &= e_{M_{k+1}}(x_{i+1}).b^M_{k+1}(i+1).a_{I_k M_{k+1}} + e_{I_k}(x_{i+1}).b^I_k(i+1).a_{I_k I_k} + b^D_{k+1}(i).a_{Ik D_{k+1}} \\
b^D_k(i) &= e_{M_{k+1}}(x_{i+1}).b^M_{k+1}(i+1).a_{D_k M_{k+1}} + e_{I_k}(x_{i+1}).b^I_k(i+1).a_{D_k I_k} + b^D_{k+1}(i).a_{D_k D_{k+1}}
\end{aligned}
\tag{7}
$$

Similar to the forward algorithm, I have implemented the backward algorithm as explained in equations (7), and used it to compute the probability of the model for generating the sequence : `GCCAG`. This same backward algorithm implementation is later used in Baum-Welch algorithm which is used for training the model with input sequences. This is explained in detail in Chapter 7.

On applying backward algorithm to the observed sequence O and the given model, we get backward matrix as shown in Table 11.

The forward algorithm and the backward algorithm that I have implemented will be used with the profile HMM that is discussed in Chapter 7.

From Table 11, the value of $b^M_0(0)$ which is given in first row of first column is 9.65E-04. It represents the probability of `GCCAG` being emitted from the given model.

Next chapter discusses the decoding problem in detail.

Table 11: Backward Matrix for `GCCAG`

State	-	G	C	C	A	G
M0	9.65E-04	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
M1	0.00E+00	1.89E-03	1.78E-02	1.15E-01	5.17E-02	2.27E-02
M2	0.00E+00	1.15E-04	9.73E-04	6.23E-03	3.70E-01	5.00E-02
M3	0.00E+00	2.44E-05	1.95E-04	1.56E-03	1.25E-02	9.00E-01
I0	0.00E+00	2.36E-03	1.31E-02	3.23E-02	3.49E-02	1.39E-02
I1	0.00E+00	1.42E-03	1.33E-02	1.26E-01	5.65E-02	2.50E-02
I2	0.00E+00	2.31E-04	2.03E-03	1.83E-02	1.85E-01	1.67E-01
I3	0.00E+00	1.22E-04	9.77E-04	7.81E-03	6.25E-02	5.00E-01
D1	9.38E-05	2.30E-03	2.17E-02	7.52E-02	8.81E-02	4.17E-02
D2	2.44E-05	1.90E-04	1.65E-03	1.40E-02	2.51E-01	1.25E-01
D3	1.53E-05	1.22E-04	9.77E-04	7.81E-03	6.25E-02	5.00E-01

CHAPTER 5

Decoding

Decoding is finding the most optimal path given a model and a sequence. The criteria for optimization are taken as most likely path. Viterbi algorithm is an efficient way to get the most likely state path. Viterbi algorithm uses a dynamic programming approach and is described in the next section.

5.1 Viterbi Algorithm

The pseudo code for Viterbi algorithm is given below in equation (8). [4].

$$
\begin{aligned}
&\textit{Initialization}: \\
&\quad v_0^M(0) = 1; \quad v_{k>0}^M(0) = 0; \quad v_0^M(i>0) = 0 \\
&\quad v_k^I(0) = 0; \quad\quad v_0^D(i) = 0; \\
&\textit{Recursion}: \\
&\quad v_k^M(i) = e_{M_k}(x_i).max \begin{cases} v_{k-1}^M(i-1).a_{M_{k-1}M_k} \\ v_{k-1}^I(i-1).a_{I_{k-1}M_k} \\ v_{k-1}^D(i-1).a_{D_{k-1}M_k} \end{cases} \\
&\quad v_k^I(i) = e_{I_k}(x_i).max \begin{cases} v_k^M(i-1).a_{M_kI_k} \\ v_k^I(i-1).a_{I_kI_k} \\ v_k^D(i-1).a_{D_kI_k} \end{cases} \\
&\quad v_k^D(i) = max \begin{cases} v_{k-1}^M(i).a_{M_{k-1}D_k} \\ v_{k-1}^I(i).a_{I_{k-1}Dk} \\ v_{k-1}^D(i).a_{D_{k-1}D_k} \end{cases} \\
&\textit{Termination}: \\
&\quad v = max\big[v_L^M(N), v_L^I(N), v_L^D(N)\big]
\end{aligned}
\tag{8}
$$

Here, the most probable state path for O = `GCCAG` is computed using Viterbi algorithm. The model used here is the same as from Chapter 4. I have implemented the Viterbi algorithm as explained in equations (8), and used it to compute the most likely path that generates the sequence : `GCCAG`. This same Viterbi algorithm implementation is later used in MSA. This is explained in detail in Chapter 7.

After applying Viterbi algorithm, we get a Viterbi matrix as shown in Table 12 and the state path is highlighted in Table 13 based on the argmax value.

Table 12: Viterbi Matrix for DNA sequence `GCCAG`

State	-	G	C	C	A	G
1.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
0.00E+00	4.77E-01	6.31E-04	5.26E-05	1.32E-05	2.56E-06	0.00E+00
0.00E+00	5.51E-03	2.37E-02	3.55E-03	2.13E-03	2.37E-05	0.00E+00
0.00E+00	7.58E-03	1.45E-02	6.31E-03	2.37E-04	8.52E-04	7.67E-04
0.00E+00	2.27E-02	1.89E-03	1.58E-04	1.32E-05	1.10E-06	0.00E+00
0.00E+00	5.05E-03	6.51E-02	6.51E-03	2.17E-04	7.23E-06	0.00E+00
0.00E+00	1.89E-03	5.42E-03	8.14E-04	8.87E-05	5.32E-05	0.00E+00
0.00E+00	9.47E-04	2.71E-03	4.07E-04	1.58E-04	2.66E-05	1.33E-05
9.09E-02	7.58E-03	6.31E-04	5.26E-05	4.38E-06	3.65E-07	0.00E+00
3.03E-02	8.68E-02	1.30E-02	1.30E-03	4.34E-05	1.45E-06	0.00E+00
7.58E-03	2.17E-02	3.25E-03	3.55E-04	2.13E-04	1.77E-05	8.87E-06

Table 13: Viterbi state path for DNA sequence `GCCAG`

State	-	G	C	C	A	G
M0	M0	M0	M0	M0	M0	-
M1	M0	I0	I0	I0	I0	-
M2	D1	M1	I1	I1	I1	-
M3	D2	D2	M2	M2	M2	M3
I0	M0	I0	I0	I0	I0	-
I1	D1	M1	I1	I1	I1	-
I2	D2	D2	D2	M2	M2	-
I3	D3	D3	D3	M3	D3	-
D1	I0	I0	I0	I0	I0	-
D2	M1	I1	I1	I1	I1	-
D3	D2	D2	M2	M2	I2	-

Hence, the most probable path for observed sequence `GCCAG` is B-M1-I1-I1-M2-M3-E.

In the previous example, a model with nucleic sequence was considered. Next chapter discusses the Viterbi algorithm implementation example of a protein model.

CHAPTER 6

Viterbi Algorithm Example of a Protein Model

In the example below, the most likely path for the sequence `ACCY` is to be computed.

In Figure 5, the insertion states are labeled from left to right by: I0, I1, I2 and I3. The matching states are labeled from left to right by M1, M2, and M3. The figure gives all the transition probabilities but not all the emission probabilities. Assume that the probabilities of emission of the three amino acids, A, C, and Y are given by the following Table 14 where Z represents some other amino acid. Each state can emit any of the 20 amino acids.

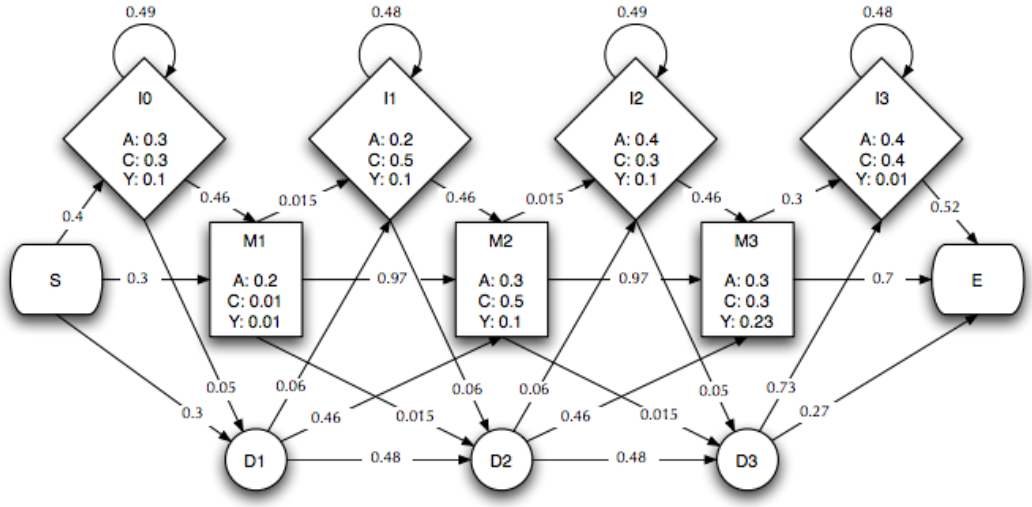

Figure 5: Profile HMM Example

From Figure 5, the emission and transition probabilities are observed as given in Table 14 and Table 15.

The Viterbi matrix for sequence `ACCY` is calculated in Table 16 using Viterbi

Table 14: Emission Probabilities for Match and Insert States

i	O_i	M_0	M_1	M_2	M_3		I_0	I_1	I_2	I_3
0	A	-	0.2	0.3	0.3		0.3	0.2	0.4	0.4
1	C	-	0.01	0.5	0.3		0.3	0.5	0.3	0.4
2	Y	-	0.01	0.1	0.23		0.1	0.1	0.1	0.01
3	Z	-	0.046	0.006	0.016		0.018	0.012	0.012	0.011

Table 15: Transition Probabilities

X-X	1	2	3	4
M-M	0.3	0.97	0.97	0.7
M-D	0.3	0.015	0.015	-
M-I	0.4	0.015	0.015	0.3
I-M	0.46	0.46	0.46	0.52
I-D	0.05	0.06	0.05	-
I-I	0.49	0.48	0.49	0.48
D-M	-	0.46	0.46	0.27
D-D	-	0.48	0.48	-
D-I	-	0.06	0.06	0.73

algorithm equations (8) of Chapter 5. I used my implementation of the Viterbi algorithm to compute the most likely path of the sequence `ACCY`.

The state path matrix is given in Table 17.

The maximum likelihood path for `ACCY` is given as "M0-I0-M1-M2-M3".

The next chapter discusses the learning problem for multiple alignment in detail.

Table 16: Viterbi Matrix for `ACCY`

X	Init	$O_1 = A$	$O_2 = C$	$O_3 = C$	$O_4 = Y$	Final
M0	1.0	0.0	0.0	0.0	0.0	0.0
M1	0.0	0.06	0.000552	0.000081144	0.000011928	0.0
M2	0.0	0.0414	0.0291	0.00026772	0.000009539	0.0
M3	0.0	0.019872	0.0120474	0.0084681	0.000059728	0.000041810
I0	0.0	0.12	0.01764	0.00259308	0.000127061	0.0
I1	0.0	0.0036	0.000864	0.00020736	0.000009953	0.0
I2	0.0	0.003456	0.000508032	0.00013095	0.000006417	0.0
I3	0.0	0.02018304	0.003875144	0.001445688	0.000025404	0.000013210
D1	0.3	0.006	0.000882	0.000129654	0.000006353	0.0
D2	0.144	0.00288	0.00042336	0.000062234	0.000003049	0.0
D3	0.069	0.0013824	0.0004365	0.000029872	0.000001464	0.000000395

Table 17: Maximum likelihood State path for `ACCY`

State	Init	$O_1 = A$	$O_2 = C$	$O_3 = C$	$O_4 = Y$	Final
M0	M0	M0	M0	M0	M0	-
M1	M0	M0	I0	I0	I0	-
M2	M0	D1	M1	M1	I1	-
M3	M0	D2	M2	M2	M2	MAX:M3
I0	M0	M0	I0	I0	I0	-
I1	M0	D1	I1	I1	I1	-
I2	M0	D2	I2	M2	I2	-
I3	M0	D3	I3	M3	M3	-
D1	M0	I0	I0	I0	I0	-
D2	D1	D1	D1	D1	D1	-
D3	D2	D2	M2	D2	D2	-

CHAPTER 7

Baum-Welch Training

In this chapter, the learning problem is covered which is relatively more complex than the other two problems that we have seen in previous chapters 4 and 5. This project mainly focuses on multiple sequence alignment and hence we will be taking the example of multiple sequence alignment training problem to explain this topic.

The steps to solve a multiple sequence alignment problem are as follows:

1. **Initialization:**

   Create an initial model by choosing a length and initial parameters

2. **Training:**

   Train the model using Baum-Welch algorithm

3. **Multiple Alignment:**

   Using Viterbi algorithm align all sequences to the trained model

The above steps are discussed in detail below.

7.1 Initialization

First step in initialization process is to decide the length (M) of the model. By length of the model we mean, number of match states present in the model and not the total number of states in the model. Hence, if M is the number of match states in the profile HMM, total number of states in our initial model will be $(3 * M + 3)$. One common rule to choose the length of the model is to set M as average length of the training sequences. We can also set M based on the prior knowledge of the sequence

family [4]. In this project, M was selected as maximum length of training sequences. In some cases it ends up stretching the resulting alignment by adding extra insert or delete columns in the final alignment but it is trivial to remove them as part of post processing phase. Due to choosing M as the maximum length, the final alignment after post processing was found to be better as compared to average length.

Second step is to initialize all the model parameters. Model parameters comprise of emission probabilities of characters of all states and transition probabilities from one state to another. Proper initialization can help in getting better alignment. Though there are various approaches to initialize these parameters, equi-probable emissions of all characters for all states were selected for simplicity. Optimization and tuning of initialization parameters were not part of the scope of this project. Transition probability for match to match state has been assigned higher value than transition probability for match to any other state.

7.2 Training

Baum-Welch algorithm is used for training the model. The algorithm works as follows:

1. Forward Algorithm

   For every sequence in training set, a forward matrix is computed using forward algorithm. This algorithm is discussed in detail in Chapter 4. To avoid the problem of underflow for larger sequences, log version of forward algorithm is used in this project. The forward algorithm equations in log domain are given in Appendix A.

2. Backward algorithm

   Similar to step 1, a backward matrix is computed for every sequence in training set using backward algorithm that is discussed in Chapter 4. Again, the backward algorithm equations in log domain are given in Appendix A.

3. Re-estimation of model parameters

   Model parameters are then re-estimated using forward and backward matrices calculated in the above steps. The expected emission counts are first calculated for all match and insert states using equations (9). Similarly transition counts are also calculated using equations (10).

   - Expected emission counts from sequence x

$$\begin{aligned} E_{M_k}(a) &= \frac{1}{P(x)} \sum_{i|x_i=a} f_{M_k}(i) b_{M_k}(i); \\ E_{I_k}(a) &= \frac{1}{P(x)} \sum_{i|x_i=a} f_{I_k}(i) b_{I_k}(i); \end{aligned} \tag{9}$$

   - Expected transition counts from sequence x

$$\begin{aligned} A_{X_k M_{k+1}} &= \frac{1}{P(x)} \sum_i f_{X_k}(i).a_{X_k M_{k+1}}.e_{M_{k+1}}(x_{i+1}).b_{M_{k+1}}(i+1) \\ A_{X_k I_k} &= \frac{1}{P(x)} \sum_i f_{X_k}(i).a_{X_k I_{k+1}}.e_{I_k}(x_{i+1}).b_{I_k}(i+1) \\ A_{X_k D_{k+1}} &= \frac{1}{P(x)} \sum_i f_{X_k}(i).a_{X_k D_{k+1}}.b_{D_{k+1}}(i) \end{aligned} \tag{10}$$

   - Probability estimation for all sequences

     After calculating expected counts from all sequences, final transition and emission probabilities are computed as follows:

$$
\begin{aligned}
a_{kl} &= \frac{A_{kl}}{\sum_{l'} A_{kl'}} \\
e_k(a) &= \frac{E_k(a)}{\sum_{a'} E_k(a')}
\end{aligned}
\tag{11}
$$

While converting to log space we have used the LSE function. The advantages and importance of LSE function are covered in Appendix B. Also, the problem of underflow is explained in Appendix A.

7.3 Multiple Alignment

After training the model, the maximum likelihood state paths are obtained for all sequences using Viterbi algorithm. Viterbi algorithm was explained in Chapter 5.

Once all state paths are obtained, alignment is performed. The length of the state path varies based on varying insertions in training sequences. Hence, first maximum number of insertion emissions are calculated for each insert state and then final alignment is done. This situation is explained with the following sample sequences.

```
FPHF-Dls.....HGSAQ          FPHF-Dls.......HGSAQ
FESFGDlstpdavMGNPK          FESFGDlstpdav..MGNPK
FDRFKHlkteaemKASED          FDRFKHlkteaem..KASED
FTQFAGkdlesi.KGTAP          FTQFAGkdlesi...KGTAP
FPTFKGlttadqlKKSAD          FPTFKGlttadql..KKSAD
FS-FLKgtsevp.QNNPE          FS-FLKgtsevp...QNNPE
FG-FSGas.....--DPG          FG-FSGas.......--DPG
                            FS-FLKngvdptaai--NPK
```

Alignment on the left is expanded as a new sequence with more insertions is added to the alignment. The resulting alignment after addition of the new sequence is displayed

on the right side [4].

7.4 Post Processing

Some Books and research papers [4] [6] have explained a process of model surgery, where insert states emitting too many characters are converted to match states and match states that are redundant get absorbed in insert states. In this project similar post processing is implemented. Poorly aligned blocks with multiple empty states are chosen for post processing and a block is picked out from the final alignment with fully conserved boundaries or if not found, the first and/or the last column. The block is then processed via a small phase of retraining and alignment. This helps in getting rid of unwanted empty states and makes the alignment more accurate and compact. This resulting alignment is definitely observed to be better than previous ones before post processing.

7.5 Toy Example of Baum-Welch training of Profile HMM

The whole process of getting alignment from unaligned sequences is explained with the help of a toy example here. Consider unaligned sequences given below:

$$
\begin{aligned}
O_1 &= \texttt{GCAG} \\
O_2 &= \texttt{GG} \\
O_3 &= \texttt{GAG} \\
O_4 &= \texttt{GCTG} \\
O_5 &= \texttt{AAC} \\
O_6 &= \texttt{GAC} \\
O_7 &= \texttt{GGG} \\
O_8 &= \texttt{AAC} \\
O_9 &= \texttt{GCCAG}
\end{aligned}
\tag{12}
$$

Largest sequence in this set is of length 5. Hence, a model with 5 match states is initialized. Hence total number of states in this model will be $(5 * 3 + 3)$ i.e. 18.

The initial emission and transition matrices in log space are given in Table 18, Table 19 and Table 20. The emission probability for all characters at all states is 0.25 and transition probability from M_i to M_{i+1} is 0.8, from M_i to I_i is 0.1 and from M_i to D_{i+1} is 0.1. Rest of the transition probabilities are equi-probable. These initial values are selected based on prior knowledge.
I have implemented the initialization function that generates these matrices for the model that is later trained for MSA with input sequences.

The forward and backward matrices for all sequences are calculated. The forward and backward matrices of sequence O = GCAG are shown in Table 21.

These forward and backward matrices are used to compute expectation counts for emission and transition for all sequences which then are used to update the model parameters. Transition matrix is given for reference in Table 23.

Table 18: Match State Emission Probability Matrix

i	O_i	M_0	M_1	M_2	M_3	M_4	M_5
0	A	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
1	C	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
2	G	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
3	T	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436

Table 19: Insert State Emission Probability Matrix

i	O_i	I_0	I_1	I_2	I_3	I_4	I_5
0	A	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
1	C	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
2	G	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436
3	T	$-\infty$	-1.38629436	-1.38629436	-1.38629436	-1.38629436	-1.38629436

Once the final model is created, all the sequences are passed through it to get state paths. These state paths and characters are then aligned to get final alignment as follows:

```
GC-AG
G---G
G--AG
GC-TG
--AAC
G--AC
G--GG
--AAC
GCCAG
```

Table 20: Transition Probability Matrix

i	X-TO-X	0	1	2	3	4	5
0	M-M	-2.23E-01	-2.23E-01	-2.23E-01	-2.23E-01	-2.23E-01	-1.63E-01
1	M-D	-2.30E+00	-2.30E+00	-2.30E+00	-2.30E+00	-2.30E+00	$-\infty$
2	M-I	-2.30E+00	-2.30E+00	-2.30E+00	-2.30E+00	-2.30E+00	-1.90E+00
3	I-M	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-6.93E-01
4	I-D	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	$-\infty$
5	I-I	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-6.93E-01
6	D-M	$-\infty$	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-6.93E-01
7	D-D	$-\infty$	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	$-\infty$
8	D-I	$-\infty$	-1.10E+00	-1.10E+00	-1.10E+00	-1.10E+00	-6.93E-01

Table 21: Forward Matrix for `GCAG` of Toy alignment example

State	-	G	C	A	G
M0	0.00E+00	$-\infty$	$-\infty$	$-\infty$	$-\infty$
M1	$-\infty$	-3.22E+00	-9.39E+00	-1.35E+01	-1.76E+01
M2	$-\infty$	-6.40E+00	-6.40E+00	-1.17E+01	-1.57E+01
M3	$-\infty$	-7.50E+00	-8.73E+00	-9.57E+00	-1.44E+01
M4	$-\infty$	-8.59E+00	-9.73E+00	-1.14E+01	-1.27E+01
M5	$-\infty$	-9.69E+00	-1.07E+01	-1.23E+01	-1.42E+01
I0	$-\infty$	-5.30E+00	-9.39E+00	-1.35E+01	-1.76E+01
I1	$-\infty$	-6.40E+00	-8.27E+00	-1.22E+01	-1.61E+01
I2	$-\infty$	-7.50E+00	-9.18E+00	-1.14E+01	-1.51E+01
I3	$-\infty$	-8.59E+00	-1.01E+01	-1.21E+01	-1.45E+01
I4	$-\infty$	-9.69E+00	-1.11E+01	-1.30E+01	-1.51E+01
I5	$-\infty$	-1.04E+01	-1.16E+01	-1.34E+01	-1.54E+01
D1	-2.30E+00	-6.40E+00	-1.05E+01	-1.46E+01	-1.87E+01
D2	-3.40E+00	-5.28E+00	-9.18E+00	-1.31E+01	-1.71E+01
D3	-4.50E+00	-6.19E+00	-8.36E+00	-1.21E+01	-1.60E+01
D4	-5.60E+00	-7.13E+00	-9.14E+00	-1.15E+01	-1.51E+01
D5	-6.70E+00	-8.09E+00	-9.97E+00	-1.21E+01	-1.46E+01

This approach is used to train model with various sequence sets and to get alignment. Results are discussed in next chapter.

Table 22: Backward Matrix for `GCAG` of Toy alignment example

State	-	G	C	A	G
M0	-1.39E+01	-1.18E+01	-9.98E+00	-8.42E+00	$-\infty$
M1	-1.28E+01	-1.10E+01	-9.06E+00	-7.41E+00	-6.29E+00
M2	-1.39E+01	-9.62E+00	-8.16E+00	-6.41E+00	-5.19E+00
M3	-1.46E+01	-1.10E+01	-6.48E+00	-5.43E+00	-4.09E+00
M4	-1.55E+01	-1.19E+01	-8.16E+00	-3.32E+00	-3.00E+00
M5	-1.67E+01	-1.30E+01	-9.27E+00	-5.59E+00	-1.63E-01
I0	-1.37E+01	-1.15E+01	-9.38E+00	-7.56E+00	-6.19E+00
I1	-1.32E+01	-1.07E+01	-8.54E+00	-6.59E+00	-5.09E+00
I2	-1.36E+01	-1.01E+01	-7.73E+00	-5.65E+00	-3.99E+00
I3	-1.43E+01	-1.06E+01	-6.99E+00	-4.73E+00	-2.89E+00
I4	-1.50E+01	-1.13E+01	-7.56E+00	-3.86E+00	-1.79E+00
I5	-1.54E+01	-1.18E+01	-8.07E+00	-4.38E+00	-6.93E-01
D1	-1.32E+01	-1.07E+01	-8.54E+00	-6.59E+00	-5.09E+00
D2	-1.36E+01	-1.01E+01	-7.73E+00	-5.65E+00	-3.99E+00
D3	-1.43E+01	-1.06E+01	-6.99E+00	-4.73E+00	-2.89E+00
D4	-1.50E+01	-1.13E+01	-7.56E+00	-3.86E+00	-1.79E+00
D5	-1.54E+01	-1.18E+01	-8.07E+00	-4.38E+00	-6.93E-01

Table 23: Expected Transition Counts Matrix for `GCAG`

i	X-TO-X	0	1	2	3	4	5
0	M-M	$-\infty$	-1.56E+00	-1.94E+00	-2.11E+00	-2.10E+00	$-\infty$
1	M-D	$-\infty$	-1.21E+00	-1.23E+00	-1.24E+00	$-\infty$	$-\infty$
2	M-I	$-\infty$	-3.64E+00	-4.02E+00	-4.19E+00	-4.18E+00	-3.44E+00
3	I-M	-4.75E+00	-4.33E+00	-4.18E+00	-4.18E+00	-4.28E+00	$-\infty$
4	I-D	-3.06E+00	-2.60E+00	-2.48E+00	-2.49E+00	$-\infty$	$-\infty$
5	I-I	$-\infty$	-4.33E+00	-4.18E+00	-4.18E+00	-4.28E+00	-3.69E+00
6	D-M	$-\infty$	-4.99E+00	-3.02E+00	-2.54E+00	-2.50E+00	$-\infty$
7	D-D	$-\infty$	-3.49E+00	-2.42E+00	-2.26E+00	$-\infty$	$-\infty$
8	D-I	$-\infty$	-4.99E+00	-3.02E+00	-2.54E+00	-2.50E+00	-2.36E+00

CHAPTER 8

Results

Program reads the test sequences from file. Sequences are given in FASTA format which is a text-based format for representing nucleotide sequences or protein sequences [13]. These test sequences are passed through the model obtained from Baum-Welch training for performing alignment. Same test sequences are input to a publicly known multiple sequence alignment program, ClustalW [11]. The results obtained from my implementation were manually compared with the alignments generated by ClustalW. Number of columns that are 100% conserved are almost similar in both alignments.

The results were also verified by generating phylogenetic trees [14] and comparing them. Phylogenetic tree from the alignment given by ClustalW and that of generated by my alignment were found to be the same in most of the cases.

Below are some test aligned sequences that were obtained. ClustalW results are also shown for comparison.

8.1 Sequence Set 1: Toy Sequence DNA

This is the toy sequence set used in Chapter 7.

Input Sequence Set	My Alignment	ClustalW alignment
GCAG	GC-AG	GCAG-
GG	G---G	G-G--
GAG	G--AG	G-AG-
GCTG	GC-TG	GCTG-
AAC	--AAC	AAC--
GAC	G--AC	GAC--
GGG	G--GG	G-GG-
AAC	--AAC	AAC--
GCCAG	GCCAG	GCCAG

8.2 Sequence Set 2 : Toy Sequence Protein

This toy sequence is used to test the alignment accuracy for protein models.

Input Sequence Set	My Alignment	ClustalW Alignment
GARFIELDTHELASTFATCAT	GARFIELDTHELASTFA-TCAT	GARFIELDTHELASTFAT-CAT
GARFIELDTHEFASTCAT	GARFIELDTHE----FASTCAT	GARFIELDTHEFASTCAT----
GARFIELDTHEVERYFASTCAT	GARFIELDTHEVERYFASTCAT	GARFIELDTHEVERYFASTCAT
THEFATCAT	--------THE----FA-TCAT	--------THEFAT-----CAT
	*** ** ****	***.

8.3 Sequence Set 3: HBB Sequences

Training dataset contained 7 sequences of average 147 length. Output displayed here is from column 0 to 59. Total time taken to align : 00:04:809(mm:ss:SSS).
My Alignment

```
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
M--LTAEEKAAVTGFWGKVKVDEVGAEALGRLLVVYPWTQRFFEHFGDLSNADAVMNNPK
M--LTAEEKAAVTGFWGKVKVDEVGAEALGRLLVVYPWTQRFFEHFGDLSNADAVMNNPK
-VHLSGEEKAAVTGLWGKVKVDEVGGEALGRLLVVYPWTQRFFDSFGDLSSASAVMGNPK
M--LTAEEKAAVTAFWGKVKVDEVGGEALGRLLVVYPWTQRFFETFGDLSTADAVMNNPK
-VHLTAEEKSAVTALWAKVNVEEVGGEALGRLLVVYPWTQRFFEAFGDLSTADAVMKNPK
M--LTAEEKAAVTAFWGKVHVDEVGGEALGRLLVVYPWTQRFFESFGDLSTADAVMNNPK
   *  *** ***  * ** * *** *****************  *****   *** ***
```

ClustalW Alignment

```
HBB_HUMAN          MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPK
HBB_SHEEP          --MLTAEEKAAVTGFWGKVKVDEVGAEALGRLLVVYPWTQRFFEHFGDLSNADAVMNNPK
HBB_SHEEP2         --MLTAEEKAAVTGFWGKVKVDEVGAEALGRLLVVYPWTQRFFEHFGDLSNADAVMNNPK
HBB_BAT            -VHLSGEEKAAVTGLWGKVKVDEVGGEALGRLLVVYPWTQRFFDSFGDLSSASAVMGNPK
HBB_BULL           --MLTAEEKAAVTAFWGKVKVDEVGGEALGRLLVVYPWTQRFFETFGDLSTADAVMNNPK
HBB_WHALE          -VHLTAEEKSAVTALWAKVNVEEVGGEALGRLLVVYPWTQRFFEAFGDLSTADAVMKNPK
HBB_WATER_BUFFALO --MLTAEEKAAVTAFWGKVHVDEVGGEALGRLLVVYPWTQRFFESFGDLSTADAVMNNPK
                     *: ***:***.:*.**:*:***.*****************: *****...*** ***
```

When phylogenetic trees are generated using both the output alignments, they were found to be similar as displayed in Figure 6 and Figure 7.

8.4 Sequence Set 4: BRCA1 Sequences

The sequences are of average length 1849. Below is a sample part of the alignment from column 1750 to 1800. Total time taken for alignment is 08:55:285(mm:ss:SSS).

My Alignment

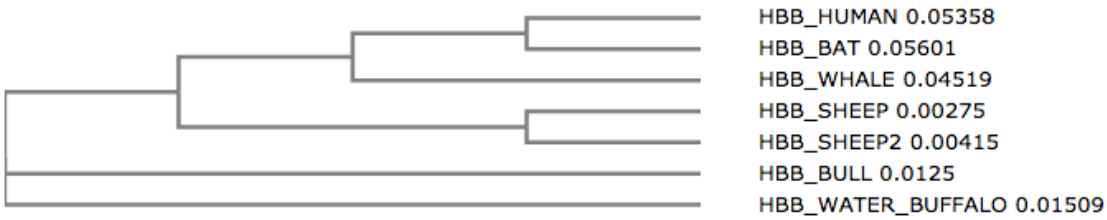

Figure 6: Phylogenetic Tree for HBB sequences aligned by my program

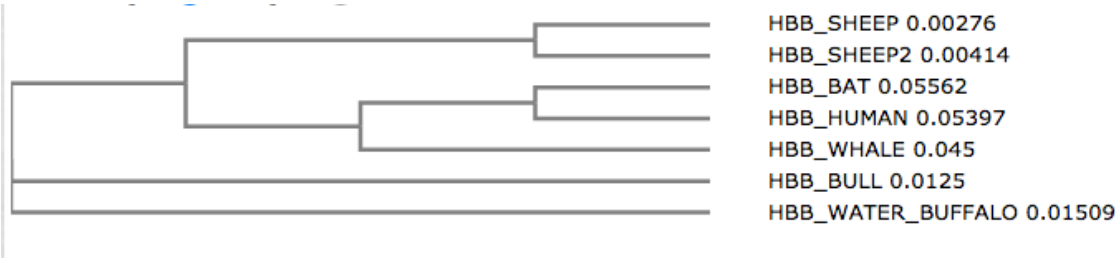

Figure 7: Phylogenetic Tree for HBB sequences aligned by ClustalW

```
NGRNHQGPKRARESQD-----RKIFRGLEICCYGPFTNMPTDQLEWMVQL

NGRNHQGPKRARESQD-----RKIFRGLEICCYGPFTNMPTDQLEWIVQL

TGRNHQGPRRSRES--RE----KLFKGLQVYCCEPFTNMPKDELERMLQL

NGRNHQGPKRARESQD-----RKIFRGLDICCYGPFTNMPTDQLEWMVQL

NGRNHQGPKRARESQDRESQDRKIFRGLEICCYGPFTNMPTDQLEWMVHL

TGSNHQGPRRSRESQ--E----KLFEGLQIYCCEPFTNMPKDELERMLQL

NGRNHQGPKRARES--R---DKKIFKGLEICCYGPFTNMPTDQLEWMVQL

 * ***** * ***        * * **   *  ****** * **    *
```

ClustalW Alignment

```
gi|555932|Homo           NGRNHQGPKRARESQDR-----KIFRGLEICCYGPFTNMPTDQLEWMVQL
gi|55976416|BRCA1_PONPY  NGRNHQGPKRARESQDR-----KIFRGLEICCYGPFTNMPTDQLEWIVQL
gi|4097808|Mus           TGRNHQGPRRSRESREK------LFKGLQVYCCEPFTNMPKDELERMLQL
gi|55976414|BRCA1_GORGO  NGRNHQGPKRARESQDR-----KIFRGLDICCYGPFTNMPTDQLEWMVQL
gi|61740517|Canis        NGRNHQGPKRARESQDRESQDRKIFRGLEICCYGPFTNMPTDQLEWMVHL
gi|2695691|Rattus        TGSNHQGPRRSRESQEK------LFEGLQIYCCEPFTNMPKDELERMLQL
gi|30466260|Bos          NGRNHQGPKRARESRDK-----KIFKGLEICCYGPFTNMPTDQLEWMVQL
                         .* *****:*:***:::      :*.**:: *  ******.*:** :::*
```

When above 2 alignments were passed through ClustalW2-Phylogeny, they produced the same phylogenetic tree as given in Figure 8 and Figure 9.

Figure 8: Phylogenetic Tree for BRCA1 sequences aligned by my program

Figure 9: Phylogenetic Tree for BRCA1 sequences aligned by ClustalW

8.5 Conclusion

In this project, I implemented Baum-Welch algorithm to train the model for alignment. I implemented the Viterbi algorithm to build the alignment. Among the various approaches for initializing the length of the model, I selected length of the model to be the maximum length of sequences. This approach provided more room for the algorithm to align the sequences. The post processing works in conjunction with initialization to remove unwanted gaps.

The results were compared with the alignments obtained from ClustalW by considering fully conserved regions and phylogenetic trees. The number of fully conserved columns were comparable. Also, the phylogenetic trees are similar too.

Thus, the profile HMM based multiple sequence alignment has been implemented with help of initialization and post processing optimizations. The main focus of the project was to concentrate on the actual alignment part of multiple sequence alignment, neglecting domain specific optimizations and fine tuning with respect to protein and nucleic sequences. For example, in protein sequences, it is okay if one character is aligned with another character of the same group. This could lead to simplified alignment process since its valid to have different characters in the same aligned column. Considering the above caveat the pure text character alignment was found to be almost as efficient or more accurate in some cases.

The next chapter talks about the scope of additional improvements and future enhancements that could be possible but are out of the scope of this project.

CHAPTER 9

Enhancements and Future Work

The initialization and post processing have various other approaches for getting better alignments. These each can be bigger areas of research with applications and use of various algorithms for each of them. Initialization and post processing were not primary focus of this project, it could be a future enhancement.

During evolution, certain substitutions are preferred over others. This gives rise to partially conserved residues. When a residue is substituted by another residue that belongs to the group having similar properties, the scoring of alignment increases. This substitution matrix can be implemented in future to get better scoring.

Currently only one final model is generated to get the alignment. This final alignment is manually compared to the reference alignment obtained from softwares like ClustalW and Clustal Omega. After implementing a scoring scheme, multiple models can be generated and using the scores of reference alignment as a reference score the best alignment model can be selected.

LIST OF REFERENCES

[1] Rachel Karchin,
Hidden Markov Models and Protein Sequence Analysis.
Retrieved December 3, 2015, from
http://www.cse.ucsc.edu/research/compbio/ismb99.handouts/KK185FP.html

[2] Multiple Sequence Alignment : Wikipedia.
Retrieved April 15, 2016, from
https://en.wikipedia.org/wiki/Multiple_sequence_alignment

[3] Sean R Eddy,
What is a Hidden Markov Model,
Nature, 2004.
Retrieved January 28, 2015, from
http://www.nature.com/nbt/journal/v22/n10/pdf/nbt1004-1315.pdf

[4] Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison,
Biological Sequence Analysis, Probabilistic models of Proteins and Nucleic Acids, 1998.

[5] Dr. Sami Khuri, Bioinformatics, Lecture 10, Hidden Markov Models,
Department of Computer Science, San Jose State University, 2015.

[6] Krogh et al.,
Hidden Markov models in computational biology. Applications to protein modeling. Retrieved August 5, 2015, from
http://www.ncbi.nlm.nih.gov/pubmed/8107089

[7] Machine Learning in Bioinformatics, Spring 2013,
HMM For Sequence Alignment, Profile HMM,
Retrieved April 16, 2016, from
http://mendel.informatics.indiana.edu/~yye/lab/index.php

[8] LogSumExp : Wikipedia.
Retrieved February 1, 2016, from
https://en.wikipedia.org/wiki/LogSumExp

[9] ClustalW and ClustalX version 2 (2007)
Larkin MA, Blackshields G, Brown NP, Chenna R, McGettigan PA, McWilliam H, Valentin F, Wallace IM, Wilm A, Lopez R, Thompson JD, Gibson TJ and Higgins DG Bioinformatics 2007 23(21): 2947-2948.

[10] Clustal - Omega.
Retrieved December 20, 2015, from
http://www.ebi.ac.uk/Tools/msa/clustalo/

[11] Multiple Sequence Alignment by CLUSTALW.
Retrieved April 15, 2016, from
http://www.genome.jp/tools/clustalw/

[12] Hidden Markov Models.
Retrieved May 4, 2016, from
http://www.biopred.net/eddy.html

[13] FASTA format : Wikipedia.
Retrieved May 1, 2016, from
https://en.wikipedia.org/wiki/FASTA_format

[14] ClustalW2 - Phylogeny.
Retrieved May 1, 2016, from
http://www.ebi.ac.uk/Tools/phylogeny/clustalw2_phylogeny/

APPENDIX A

Conversion from Natural to Log Domain

In order to avoid underflow, all calculations are performed in log domain. To make these calculations easier and to avoid unnecessary conversions of values from log to exponential domain, all initial parameters are stored in their log format. Hence, if $a_{M_{k-1}M_k}$ is transition probability from state (k-1) to (k),

$A_{M_{k-1}M_k} = \ln(a_{M_{k-1}M_k})$.

Similarly, $E_{M_k}(x_i) = \ln(e_{M_k}(x_i))$

and $F_k^M(i) = \ln(f_k^M(i))$

A.1 Forward Algorithm Logs Version

$$
\begin{aligned}
&\textit{Initialization}: \\
&\quad F_0^M(0) \;=\; 0 \\
&\textit{Recursion}: \\
&\quad F_k^M(i) \;=\; E_{M_k}(x_i) \;+\; \ln\big[\exp(F_{k-1}^M(i-1) \;+\; A_{M_{k-1}M_k}) \\
&\qquad\qquad +\; \exp(F_{k-1}^I(i-1) \;+\; A_{I_{k-1}M_k}) \\
&\qquad\qquad +\; \exp(F_{k-1}^D(i-1) \;+\; A_{D_{k-1}M_k})\big] \\
&\quad F_j^I(i) \;=\; E_{I_j}(x_i).\ln\big[\exp(F_j^M(i-1) \;+\; A_{M_jI_j}) \\
&\qquad\qquad +\; \exp(F_j^I(i-1) \;+\; A_{I_jI_j}) \\
&\qquad\qquad +\; \exp(F_j^D(i-1) \;+\; A_{DjI_j})\big] \\
&\quad F_j^D(i) \;=\; \ln[\exp(F_{j-1}^M(i) \;+\; A_{M_{j-1}D_j}) \\
&\qquad\qquad +\; \exp(F_{j-1}^I(i) \;+\; A_{I_{j-1}D_j}) \\
&\qquad\qquad +\; \exp(F_{j-1}^D(i) \;+\; A_{D_{j-1}D_j})\big] \\
&\textit{Termination}: \\
&\quad F_{m+1}^M(L+1) \;=\; \ln[\exp(F_m^M(L) \;+\; A_{M_mM_{m+1}}) \\
&\qquad\qquad +\; \exp(F_m^I(L) \;+\; A_{I_mM_{m+1}}) \\
&\qquad\qquad +\; \exp(F_m^D(L) \;+\; A_{D_mM_{m+1}})\big]
\end{aligned}
\tag{A.13}
$$

Similarly, backward, Viterbi and Baum-Welch re-estimation algorithms are given in next sections.

A.2 Backward Algorithm Logs Version

$$
\begin{aligned}
&\textit{Initialization}: \\
B^M_{M+1}(L+1) &= 0; \\
B^M_M(L) &= A_{M_M M_{M+1}}; \\
B^I_M(L) &= A_{I_M M_{M+1}}; \\
B^D_M(L) &= A_{D_M M_{M+1}}; \\
&\textit{Recursion}: \\
B^M_k(i) &= \ln\big[\exp(E_{M_{k+1}}(x_{i+1}) + B^M_{k+1}(i+1) + A_{M_k M_{k+1}}) \\
&\quad + \exp(E_{I_k}(x_{i+1}) + B^I_k(i+1) + A_{M_k I_k}) \\
&\quad + \exp(B^D_{k+1}(i) + A_{M_k D_{k+1}})\big] \\
B^I_k(i) &= \ln\big[\exp(E_{M_{k+1}}(x_{i+1}) + B^M_{k+1}(i+1) + A_{I_k M_{k+1}} \\
&\quad + \exp(E_{I_k}(x_{i+1}) + B^I_k(i+1) + A_{I_k I_k}) \\
&\quad + \exp(B^D_{k+1}(i) + A_{IkD_{k+1}})\big] \\
B^D_k(i) &= \ln\big[\exp(E_{M_{k+1}}(x_{i+1}) + B^M_{k+1}(i+1) + A_{D_k M_{k+1}} \\
&\quad + \exp(E_{I_k}(x_{i+1}) + B^I_k(i+1) + A_{D_k I_k}) \\
&\quad + \exp(B^D_{k+1}(i) + A_{D_k D_{k+1}})\big]
\end{aligned}
\tag{A.14}
$$

A.3 Viterbi Algorithm Logs Version

$Initialization:$

$$V_0^M(0) = 0; \quad V_{k>0}^M(0) = -Infinity; \quad V_0^M(i>0) = -Infinity$$

$$V_k^I(0) = -Infinity; \quad V_0^D(i) = -Infinity;$$

$Recursion:$

$$V_k^M(i) = E_{M_k}(x_i) \ + \ max \begin{cases} (V_{k-1}^M(i-1) \ + \ A_{M_{k-1}M_k}) \\ V_{k-1}^I(i-1) \ + \ A_{I_{k-1}M_k} \\ V_{k-1}^D(i-1) \ + \ A_{D_{k-1}M_k} \end{cases}$$

$$V_k^I(i) = E_{I_k}(x_i) \ + \ max \begin{cases} V_k^M(i-1) \ + \ A_{M_kI_k} \\ V_k^I(i-1) \ + \ A_{I_kI_k} \\ V_k^D(i-1) \ + \ A_{D_kI_k} \end{cases} \tag{A.15}$$

$$V_k^D(i) = max \begin{cases} V_{k-1}^M(i) \ + \ A_{M_{k-1}D_k} \\ V_{k-1}^I(i) \ + \ A_{I_{k-1}Dk} \\ V_{k-1}^D(i) \ + \ A_{D_{k-1}D_k} \end{cases}$$

$Termination:$

$$V = max\big[V_L^M(N), V_L^I(N), V_L^D(N)\big]$$

A.4 Baum-Welch Re-estimations Logs Version

Expected emission counts from sequence x:

$$\begin{aligned} E_{M_k}(a) &= -\ln(P(x)) \sum_{i|x_i=a} \exp(F_{M_k}(i) + B_{M_k}(i)); \\ E_{I_k}(a) &= -\ln(P(x)) \sum_{i|x_i=a} \exp(F_{I_k}(i) + B_{I_k}(i)); \end{aligned} \tag{A.16}$$

Expected transition counts from sequence x:

$$\begin{aligned} A_{X_k M_{k+1}} &= -\ln(P(x)) \sum_i \exp[F_{X_k}(i) + A_{X_k M_{k+1}} + E_{M_{k+1}}(x_{i+1}) + B_{M_{k+1}}(i+1)] \\ A_{X_k I_k} &= -\ln(P(x)) \sum_i \exp[F_{X_k}(i) + A_{X_k I_{k+1}} + E_{I_k}(x_{i+1}) + B_{I_k}(i+1)] \\ A_{X_k D_{k+1}} &= -\ln(P(x)) \sum_i \exp[F_{X_k}(i) + A_{X_k D_{k+1}} + B_{D_{k+1}}(i)] \end{aligned} \tag{A.17}$$

APPENDIX B

Log Sum Exponential (LSE)

In case of forward and backward algorithm, there is one problem with computation. Consider the following recursion equation (B.18) of forward algorithm in log space, which is a part of equation (A.13). Here, we cannot easily calculate the logarithm of term without calculating exponentiation of other terms. This may still lead to underflow. Hence, log sum exponential function is used to solve this problem. LSE is an approximation function to calculate log of sum of exponential in machine learning algorithms[8]. This function is given in equation (B.19)

$$\begin{aligned} F_k^M(i) \;=\; & E_{M_k}(x_i) \;+\; \ln\big[\exp(F_{k-1}^M(i-1) \;+\; A_{M_{k-1}M_k}) \\ & +\; \exp(F_{k-1}^I(i-1) \;+\; A_{I_{k-1}M_k}) \\ & +\; \exp(F_{k-1}^D(i-1) \;+\; A_{D_{k-1}M_k})\big] \end{aligned} \tag{B.18}$$

$$\begin{aligned} LSE(x_1,\; x_2, ...,\; x_n) \;&=\; \log(\exp(x_1) \;+\; \exp(x_2 \;+ ..... +\; \exp(x_n)) \\ LSE(x_1,\; x_2, ...,\; x_n) \;&=\; x^* \;+\; \log(\exp(x_1 - x^*) \;+\; \exp(x_2 - x^*) \;+ ..... +\; \exp(x_n - x^*)) \\ where\; x^* \;&=\; max(x_1,\; x_2, ...,\; x_n) \end{aligned} \tag{B.19}$$