Hidden Markov Methods. Algorithms and Implementation

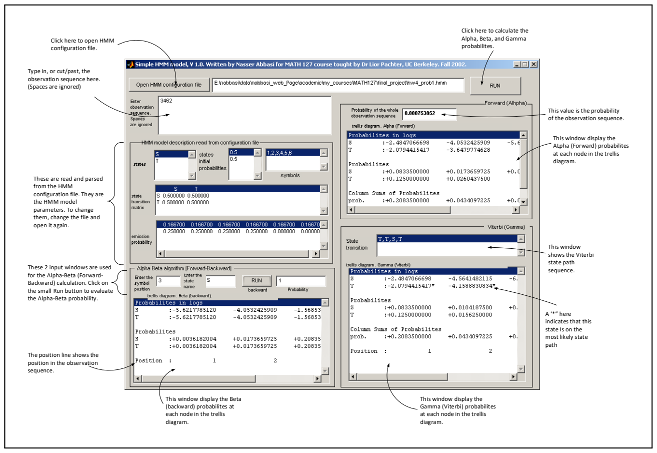
Final Project Report. MATH 127.

Nasser M. Abbasi

Course taken during Fall 2002 Compiled on September 9, 2023 at 10:45am

Hidden Markov Models (HMM) main algorithms (forward, backward, and Viterbi) are outlined, and a GUI based implementation (in MATLAB) of a basic HMM is included along with a user guide. The implementation uses a flexible plain text configuration file format for describing the HMM.

This is screen shot of the final Matlab program written for the implmenetation.


Simple HMM model, V 1.0. Written by Nasser Abbasi for MATH 127 course tought by Dr Lior Pachter, UC Berkeley. Fall 2002.
Open HMM configuration file
E:\nabbasi\data\nabbasi_web_Page\academic\my_courses\MATH127\final_project\hw4_prob1.hmm
RUN
Enter observation sequence. Spaces are ignored
HMM model description read from configuration file
states
states initial probabilities
symbols
state transition matrix
emission probability
Alpha Beta algorithm (Forward-Backward)
Enter the symbol position
Enter the state name
RUN
backward
Probability
trellis diagram. Beta (backward).
Forward (Alhpha)
Probability of the whole observation sequence
trellis diagram. Alpha (Forward)
Viterbi (Gamma)
State transition
trellis diagram. Gamma (Viterbi)

Click here to open HMM configuration file.

Type in, or cut/past, the observation sequence here. (Spaces are ignored)

These are read and parsed from the HMM configuration file. They are the HMM model parameters. To change them, change the file and open it again.

These 2 input windows are used for the Alpha-Beta (Forward-Backward) calculation. Click on the small Run button to evaluate the Alpha-Beta probability.

The position line shows the position in the observation sequence.

This window display the Beta (backward) probabilites at each node in the trellis diagram.

Click here to calculate the Alpha, Beta, and Gamma probabilites.

This value is the probability of the observation sequence.

This window display the ALpha (Forward) probabilites at each node in the trellis diagram.

This window shows the Viterbi state path sequence.

A "*" here indicates that this state Is on the most likely state path

This window display the Gamma (Viterbi) probabilites at each node in the trellis diagram.

Enter observation sequence: 3462

states: S, T

states initial probabilities: 0.5, 0.5

symbols: 1,2,3,4,5,6

```
   S        T
S 0.500000 0.500000
T 0.500000 0.500000
```

```
0.166700 0.166700 0.166700 0.166700 0.166700 0.16670
0.250000 0.250000 0.250000 0.250000 0.000000 0.00000
```

Enter the symbol position: 3; Enter the state name: S; Probability: 1

```
Probabilites in logs
S          :-5.6217785120     -4.0532425909     -1.56853
T          :-5.6217785120     -4.0532425909     -1.56853

Probabilites
S          :+0.0036182004     +0.0173659725     +0.20835
T          :+0.0036182004     +0.0173659725     +0.20835

Position  :            1                 2
```

Probability of the whole observation sequence: 0.000753852

```
Probabilites in logs
S          :-2.4847066698     -4.0532425909     -5.6
T          :-2.0794415417     -3.6479774628

Probabilites
S          :+0.0833500000     +0.0173659725     +0.0
T          :+0.1250000000     +0.0260437500

Column Sums of Probabilites
prob.      :+0.2083500000     +0.0434097225     +0.0
```

State transition: T,T,S,T

```
Probabilites in logs
S          :-2.4847066698     -4.5641482115     -6.
T          :-2.0794415417*    -4.1588830834*

Probabilites
S          :+0.0833500000     +0.0104187500     +0.
T          :+0.1250000000     +0.0156250000

Column Sums of Probabilites
prob.      :+0.2083500000     +0.0434097225     +0.

Position  :            1                 2
```

Figure 1: Matlab HMM Program GUI

Description and download information on the above program as in the appendix.

Contents

CHAPTER 1

INTRODUCTION

This report is part of my final term project for the course MATH 127, Mathematics department, Berkeley, CA, USA

I have taken this course as an extension student in the fall of 2002 to learn about computational methods in biology. The course is titled **Mathematical and Computational Methods in Molecular Biology** and given by Dr. Lior Pachter.

This report contains a detailed description of the three main HMM algorithms. An implementation (in MATLAB) was completed. See page 29 for detailed description of the implementation, including a user guide.

HMM are probabilitsic based methods for the analysis of signals and data. In the area of computational biology, HMMs are used in the following areas:

1. Pattern finding (Gene finding[3]).
2. multiple sequence alignments.
3. protein or RNA secondary structure prediction.
4. data mining and classification.

CHAPTER 2

DEFINITIONS AND NOTATIONS

To explain HMM, a simple example is used.

Assume we have a sequence. This could be a sequence of letters, such as DNA or protein sequence, or a sequence of numerical values (a general form of a signal) or a sequence of symbols of any other type. This sequence is called the observation sequence.

Now, we examine the system that generated this sequence. Each symbol in the observation sequence is generated when the system was in some specific state. The system will continuously flip-flop between all the different states it can be in, and each time it makes a state change, a new symbol in the sequence is emitted.

In most instances, we are more interested in analyzing the system states sequence that generated the observation sequence. To be able to do this, we model the system as a stochastic model which when run, would have generated this sequence. One such model is the hidden Markov model. To put this in genomic perspective, if we are given a DNA sequence, we would be interested in knowing the structure of the sequence in terms of the location of the genes, the location of the splice sites, and the location of the exons and intron among others.

What we know about this system is what states it can be in, the probability of changing from one state to another, and the probability of generating a specific type of element from each state. Hence, associated with each symbol in the observation sequence is the state the system was in when it generated the symbol.

When looking a sequence of letter (a string) such as "ACGT ", we do not know which state the system was in when it generated, say the letter 'C'. If the system have n states that it can be in, then the letter 'C' could have been generated from any one of those n states.

From the above brief description of the system, we see that there are two stochastic processes in progress at the same time. One stochastic process selects which state the system will switch to each time, and another stochastic process selects which type of element to be emitted when the system is in some specific state.

We can imagine a multi-faced coin being flipped to determine which state to switch to, and yet another such coin being flipped to determine which symbol to emit each time we are in a state.

In this report, the type of HMM that will be considered will be time-invariant. This means that the probabilities that describe the system do not change with time: When an HMM is in some state, say k, and there is a probability p to generate the symbol x from this state, then this probability do not change with time (or equivalent, the probability do not change with the length of the observation sequence).

In addition, we will only consider first-order HMM.[1]

An HMM is formally defined[2] as a five-tuple:

1. Set of states $\{k, l, \cdots, q\}$. These are the finite number of states the system can be in at any one time.

   It is possible for the system to be in what is referred to as *silent states* in which no symbol is generated. Other than the special start and the end silent states (called the 0 states), all the system states will be active states (meaning the system will generate a symbol each time it is in one of these states).

   When we want to refer to the state the system was in when it generated the element at position i in the observation sequence x, we write π_i. The initial state, which is the silent state the system was in initially, is written as π_0. When we want to refer to the name of the state the system was in when it generated the symbol at position i we write $\pi_i = k$.

2. P_{kl} is the probability of the system changing from state k to state l.

   For a system with n number of states, there will be a transition matrix of size $n \times n$ which gives the state transition probability P_{kl} between any 2 states.

3. $b_k(i)$ is the emission probability. It is the probability of emitting the symbol seen at position i in the observation sequence x when the system was in state k.

   As an example of the emission probability, assume that the observed sequence is $x = ATATTCGTC$ and assume that the system can be in any one of two states $\{k, l\}$. Then we write the probability of emitting the first symbol A when the system is in state k as $b_k(1)$.

   We could also write the above as $b_k(A)$, since the symbol A is located at the first position of the sequence. In the above observation sequence, since the letter A occurred in the first and third positions, then $b_k(1)$ will have the same value as $b_k(3)$, since in both cases, the same type of letter is being emitted, which is the symbol A.

[1] first-order HMM means that the current state of the system is dependent only on the previous state and not by any earlier states the system was in.

[2] In this report, I used the notations for describing the HMM algorithms as those used by Dr Lior Pachter, and not the notations used by the text book Durbin et all[1]

4. Set of initial probabilities P_{0k} for all states k. We imagine a special silent start state 0 from which the initial active state is selected according to the probability P_{0k}. see figure 2.1 on page 9 for illustration.

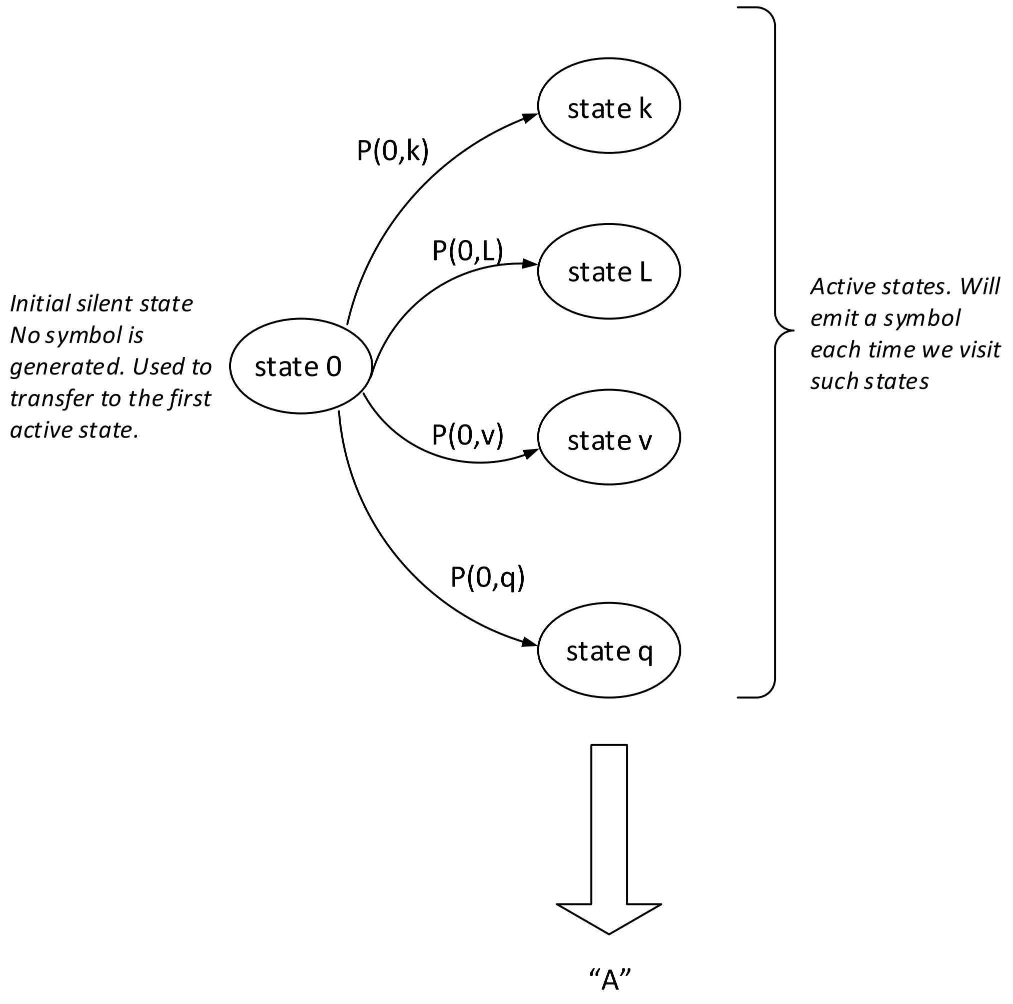

Initial silent state No symbol is generated. Used to transfer to the first active state.
state 0
P(0,k)
P(0,L)
P(0,v)
P(0,q)
state k
state L
state v
state q
Active states. Will emit a symbol each time we visit such states
"A"

Figure 2.1: Initial silent state and initial transition to one of the active states. silentState.pdf

For a system with n number of states, there will be n such $P_{0,k}$ probabilities. So for some state, say q, the probability of this state being the first state the system starts in is given by P_{0q}. Notice that another notation for the initial probability P_{0k} is π_k, and this is used in [1] and other publication.

5. The set of possible observation sequences x. We imagine the system running and generating an observation sequence each time according to the emission probability distribution and according to the state sequence associated with this observation sequence. The collection of possible observation sequences is the language of the system. The unique alphabets that make up the observation sequences x are the allowed alphabet of the language. As an example, an HMM used to model DNA

sequences would have $ACGT$ as its alphabets, while one used to model protein sequences would have the 20 amino acids as its alphabets.

So, to summarize how this model generates an observation sequence: The system starts in state 0. Then based the probability distribution P_{0k} called the initial state probability, a state k is selected. Then based on the emission probability distribution $b_k(i)$ for this state, the symbol at position i is selected and generated. Then a new state l is selected based on the transition probability distribution P_{kl} and the process repeats again as above. This process is continued until the last symbol is generated.

The sequence generated by the system is called the observation sequence x. The symbol at position i in the sequence is called $x[i]$ or x_i. We write $x[i] = A$ to mean that the symbol at position i is A.

In the implementation of HMM for this project, the length of each output symbol is limited to one character.

For a system with n states, and an observation sequence of length m, the number of possible state sequences that could have generated this observation sequence is n^m. Each one of these possible state sequences $\pi_{1 \cdots m}$ is of length m since the system will generate one symbol for each state transition as we are considering that there will not be any silent states other than the start and end states.

As an example, given a system with $n = 2$ with states $\{k, l\}$, and an observation sequence of 'GCT' of length $m = 3$, then there are $n^m = 2^3$ or 8 possible state sequences. They are

$$kkk, kkl, klk, kll, lkk, lkl, llk, lll$$

One of these 8 state sequences is the most likely one to have occurred.[3]

To summarize the above discussion, imagine we have a system with states $\{T, H\}$, and that the possible symbols or alphabets of the system it can generate are $1, 2, 3, 4$. Figure 2.2 on page 11 shows one such possible state sequence π and the associated observation sequence x.

[3]One of the HMM algorithms, called Viterbi, is used to find which one of these sequences is the most likely one.

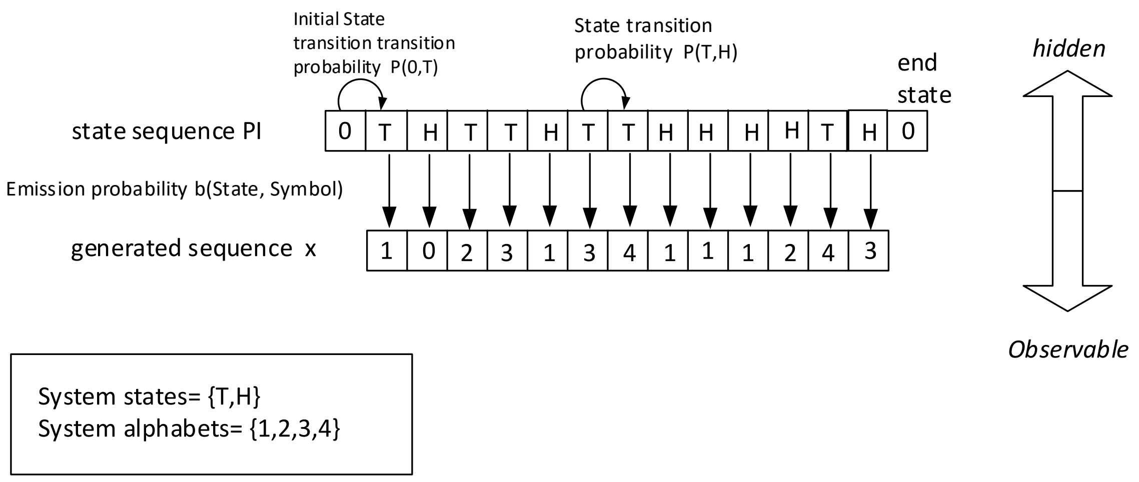

Initial State transition transition probability P(0,T)
State transition probability P(T,H)
end state
hidden
state sequence PI
0 T H T T H T T H H H H T H 0
Emission probability b(State, Symbol)
generated sequence x
1 0 2 3 1 3 4 1 1 1 2 4 3
Observable
System states= {T,H}
System alphabets= {1,2,3,4}

Figure 2.2: HMM parameters and how they are related.stateSeq.pdf

The questions one could ask about the above HMM system are:

1. Given a sequence of symbols x of length L that represents the output of the system (the observation sequence), determine the probability of the system generating this sequence up to and including $x[L]$. This is solved using the forward algorithm α.
2. Given a sequence of symbols x that represents the output of the system (the observations), determine the probability that some specific symbol in the observation sequence was generated when the system was in some specific state.

   This is solved using the forward-backward algorithm ($\alpha\beta$). As an example, we can ask about the probability that some DNA nucleotide from a large DNA sequence was part of an exon, where an exon is one of the possible system states.
3. Given a sequence of symbols x of length L that represents the output of the system (the observation sequence), determine the most likely state transition sequence the system would have undergone in order to generate this observation sequence.

   This is solved using the Viterbi algorithm γ. Note again that the sequence of states $\pi_{i \cdots j}$ will have the same length as the observation sequence $x_{i \cdots j}$.

The above problems are considered HMM analysis problems. There is another problem, which is the training or identification problem whose goal is to estimate the HMM parameters (the set of probabilities) we discussed on page 8 given some set of observation sequences taken from the type of data we wish to use the HMM to analyze. As an example, to train the HMM for gene finding for the mouse genome, we would train the HMM on a set of sequences taken from this genome.

One standard algorithm used for HMM parameter estimation (or HMM training) is called Baum-Welch, and is a specialized algorithm of the more general algorithm called EM (for expectation maximization). In the current MATLAB implementation, this algorithm is not implemented, but could be easily added later if time permits.

A small observation: In these notations, and in the algorithms described below, I use the value 1 as the first position (or index) of any sequence instead of the more common 0 used in languages such as C, C++ or Java. This is only for ease of implementation as I will be using MATLAB for the implementation of these algorithms, and MATLAB uses 1 and not 0 as the starting index of an array. This made translating the algorithms into MATALAB code much easier.

CHAPTER 3

GRAPHICAL REPRESENTATIONS OF AN HMM

There are two main methods to graphically represent an HMM.

One method shows the time axis (or the position in the observation sequence) and the symbols emitted at each time instance (called the trellis diagram). The other method is a more concise way of representing the HMM, but does not show which symbol is emitted at what time instance.

To describe each representation, I will use an example, and show how the example can be represented graphically in both methods.

3.1 Example HMM^^E^^L

Throughout this report, the following HMM example is used for illustration. The HMM configuration file for the example that can be used with the current implementation is shown. Each different HMM state description (HMM parameters) is written in a plain text configuration file and read by the HMM implementation. The format of the HMM configuration file is described on page 29.

Assume the observed sequence is $ATACC$. This means that A was the first symbol emitted, seen at time $t = 1$, and T is the second symbol seen at time $t = 2$, and so forth. Notice that we imagine that the system will write out each symbol to the end of the current observation sequence.

Next, Assume the system can be in any one of two internal states $\{S, T\}$.

Assume the state transition matrix P_{kl} to be as follows

$$P_{ST} = 0.3 \quad P_{SS} = 0.7$$
$$P_{TS} = 0.4 \quad P_{TT} = 0.6$$

And for the initial probabilities P_{0k}, assume the following

$$\pi_S = 0.4$$
$$\pi_T = 0.6$$

And finally for the emission probabilities $b_k(i)$, there are 3 unique symbols that are generated in this example (the alphabets of the language of the observation sequence), and they are A, T, C. So there will be 6 emission probabilities, 3 symbols for each state, and given we have 2 states, then we will have a total of 6 emission probabilities. Assume they are given by the following emission matrix:

$$\begin{array}{ccc} b_S(A) = 0.4 & b_S(C) = 0.4 & b_S(T) = 0.2 \\ b_T(A) = 0.25 & b_T(C) = 0.55 & b_T(T) = 0.2 \end{array}$$

The first (non-time dependent) graphical representation of the above example is shown in figure 3.1 on page 14

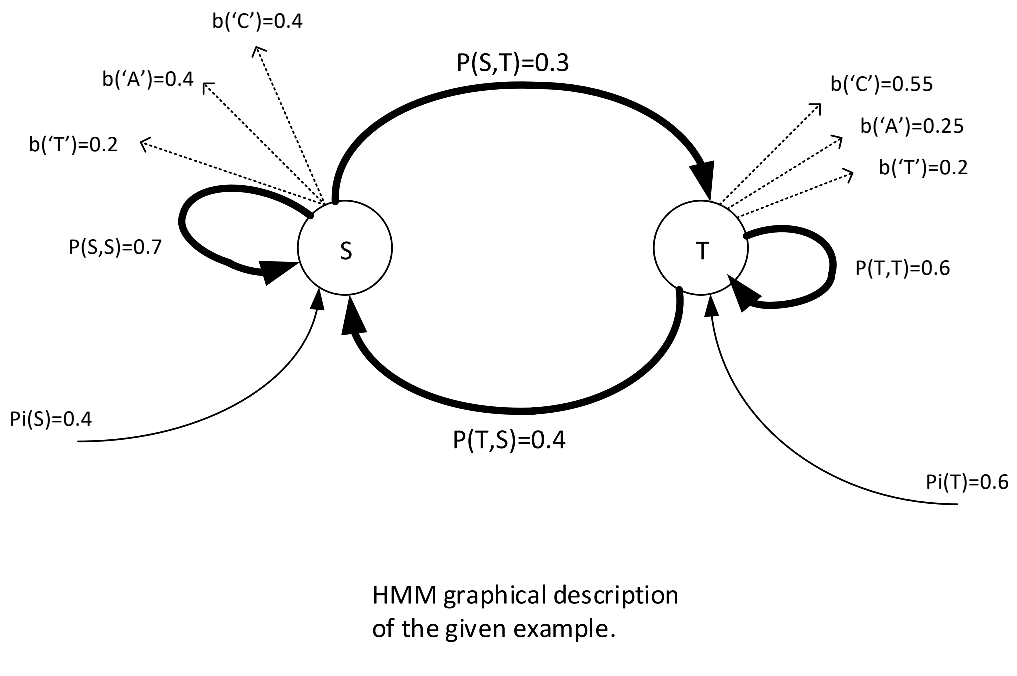


Figure 3.1: HMM representation using node based diagram

In this representation, each state is shown as a circle (node). State transitions are represented by a directed edge between the states. Emission of a symbol is shown as an edge leaving the state with the symbol at the end of the edge.

Initial probability $\pi_{0,k}$ are shown as edges entering into the state from state 0 (not shown). We imagine that there is a silent start state 0 which all states originate from. The system cannot transition to state 0, it can only transition out of it.

The second graphical representation is a more useful way to represent an HMM for the description of the algorithms. This is called the trellis diagram.

Draw a grid, where time flows from left to right. On top of the grid show the symbols being emitted at each time instance. Each horizontal line in this grid represents one state, with state transitions shown as sloping edges between the nodes in the grid.

See figure 3.2 on page 15 for the graphical representation for the same example above using the trellis representation.

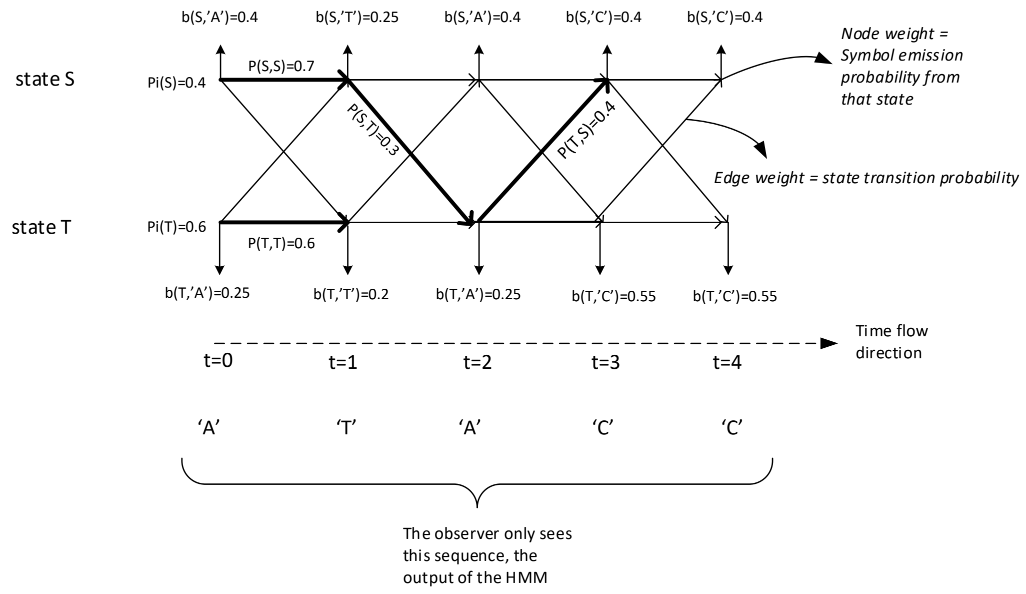


Figure 3.2: HMM representation using trellis diagram.

On each line segment in the grid, we draw an arrow from position i to $i + 1$ and label this arrow by the state transition probability from the state where the start of the arrow was to the state where the end of the arrow is. The weight of the edge is the state transition probability.

Notice that the edge can only be horizontal or slopped in direction. We can not have vertical edges, since we must advance by one time step to cause a state transition to occur. This means there will be as many internal states transitions as the number of time steps, or equivalently, as the number of symbols observed or the length of the observation sequence.

To use the above example in SHMM, we need to write down the state description. The format of writing down the state description is given on page 29. This below is the SHMM configuration file for the above example.

```
<states>
S
T
<init_prob>
```

```
0.4
0.6
<symbols>
A,C,T
<emit_prob>
#emit probability from S state
0.4,0.4,0.2
#emit probability from T state
0.25,0.55,0.2

<tran_prob>
0.7,   0.3
0.4,   0.6
```

CHAPTER 4

HMM ALGORITHMS

Given the above notations and definitions, we are ready to describe the HMM algorithms.

4.1 Forward α algorithm^^E^^L

Here we wish to find the probability p that some observation sequence x of length m was generated by the system which has n states. We want to find the probability of the sequence up to and including $x[m]$, in other words, the probability of the sequence $x[1] \cdots x[m]$.

The most direct an obvious method to find this probability, is to enumerate all the possible state transition sequences $q = \pi_{1 \cdots m}$ the system could have gone through. Recall that for an observation sequence of length m there will be n^m possible state transition sequences.

For each one of these state transition sequences q_i (that is, we fix the state transition sequence), we then find the probability of generating the observation sequence $x[1] \cdots x[m]$ for this one state transition sequence.

After finding the probability of generating the observation sequence for each possible state transition sequence, we sum all the resulting probabilities to obtain the probability of the observation sequence being generated by the system.

As an example, for a system with states $\{S, K, Z\}$ and observation sequence $AGGT$ a partial list of the possible state sequences we need to consider is shown in figure 4.1 on page 18.

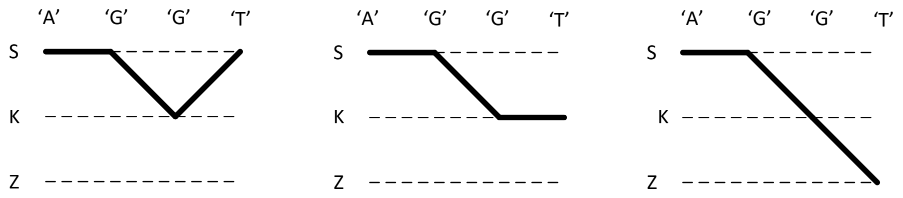


Figure 4.1: Finding the probability of a an observation sequence using brute search method.

We can perform the above algorithm as follows:

Find the probability of the system emitting the symbol $x[1]$ from π_1. This is equal to the probability of emitting the symbol $x[1]$ from state π_1 multiplied by the probability of being in state π_1.

Next, find the probability of generating the sequence $x[1] \cdots x[2]$. This is equal to the probability of emitting the sequence $x[1]$ (which we found from above), multiplied by the probability of moving from state π_1 to state π_2, multiplied by the probability of emitting the symbol $x[2]$ from state π_2.

Next, find the probability of generating the sequence $x[1] \cdots x[3]$. This is equal to the probability of emitting the sequence $x[1] \cdots x[2]$ (which we found from above), multiplied by the probability of moving from state π_2 to state π_3, multiplied by the probability of emitting the symbol $x[3]$ from state π_3.

We continue until we find the probability of emitting the full sequence $x[1] \cdots x[m]$

Obviously the above algorithm is not efficient. There are n^m possible state sequences, and for each one we have to perform $O(2m)$ multiplications, resulting in $O(m\ n^m)$ multiplications in total.

For an observation sequence of length $m = 500$ and a system with $n = 10$ states, we have to wait a very long time for this computation to complete.

A much improved method is the α algorithm. The idea of the α algorithm, is that instead of finding the probability for each different state sequence separately and then summing the resulting probabilities, we scan the sequence $x[1] \cdots x[i]$ from left to right, one position at a time, but for each position we calculate the probability of generating the sequence up to this position for all the possible states.

Let us call the probability of emitting the sequence $x[1] \cdots x[i]$ as $\alpha(i)$. And we call the probability of emitting $x[1] \cdots x[i]$ when the system was in state k when emitting $x[i]$ as $\alpha_k(i)$.

So, the probability of emitting the sequence $x[1] \cdots x[i]$ will be equal to the probability of emitting the same sequence but with $x[i]$ being emitted from one state say k, this is called $\alpha_k(i)$, plus the probability of emitting the same sequence but with $x[i]$ being emitted from another state, say l, this is called $\alpha_l(i)$, and so forth until we have covered all the possible states. In other words $\alpha(i)$ is the $\sum_k \alpha_k(i)$ for all states k.

Each $\alpha_k(i)$ is found by multiplying the $\alpha(i-1)$ for each of the states, by the probability of moving from each of those states to state k, multiplied by the probability of emitting $x[i]$ from state k. This step is illustrated in figure 4.2 on page 19.

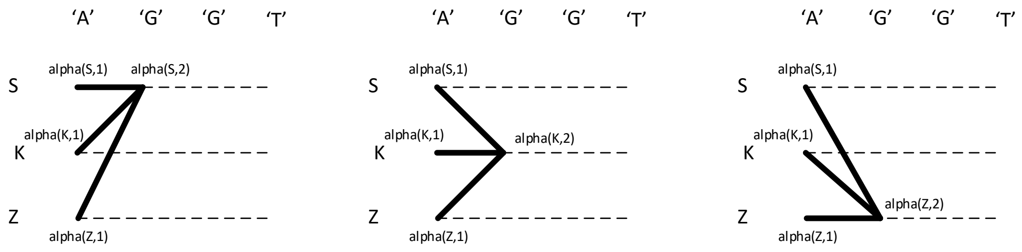


Figure 4.2: Finding the probability of a sequence using the α method

Each time we calculate $\alpha_k(i)$ for some state k, we store this value, and use it when calculating $\alpha(i+1)$. Hence We only need to store the $\alpha's$ for one position earlier since we are working with a first-order HMM here. This means we need space requirements for two columns each of length of the number of states n.

From above, we see that we have reduced the running time $O(mn^m)$ of the direct method, to $O(mn)$ with the α method.

The following is a more formal description of the algorithm to find α for one state, say state k.

Initialization step:

$$\alpha_k(1) = p_{0k} b_k(1)$$

What the above says is that α, initially (at position $i = 1$) and for state k , is equal to the probability of the HMM starting initially in state k multiplied by the probability of emitting the symbol found at $x[1]$ from state k.

Iterative step:

$$j = 2 \cdots i \qquad \alpha_k(j) = b_k(j) \sum_l \alpha_l(j-1) P_{lk}$$

What the above says, is that α at position $i = j$ when in state k, is the sum of all the $\alpha's$ from one observation earlier collected from all the states, multiplied by the probability of transition from each of those states to the state k, multiplied by the probability of emitting the symbol x_j when in state k.

I will illustrate this iterative step for one time step, using the same example from above, and using the grid representation of an HMM to find $\alpha(3)$. Since there are 2 states $\{S, T\}$ in this example, we need to calculate the iterative step for each state.

When the system is in state S at $j = 3$ we obtain the diagram shown in figure 4.3 on page 20.

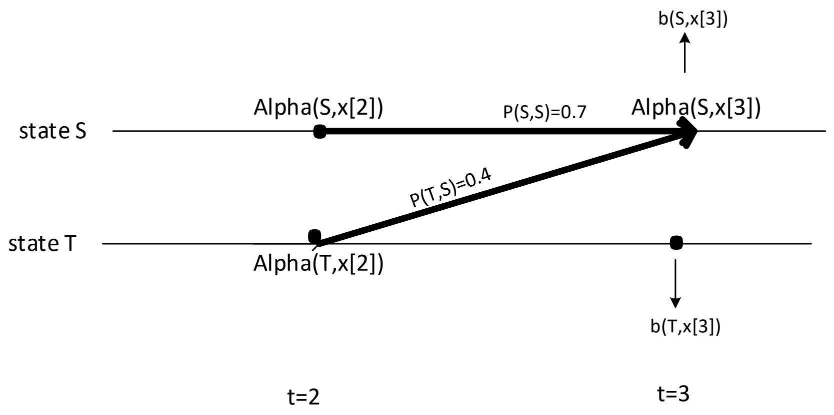


Figure 4.3: Recursive step for the α method for state S

In the above example, to find $\alpha_S(3)$, we obtain

$$\alpha_S(3) = b_S(3)[\alpha_S(2)P_{SS} + \alpha_T(2)P_{TS}]$$

$\alpha_T(3)$ is found similarly, and is shown in figure 4.4 on page 20.

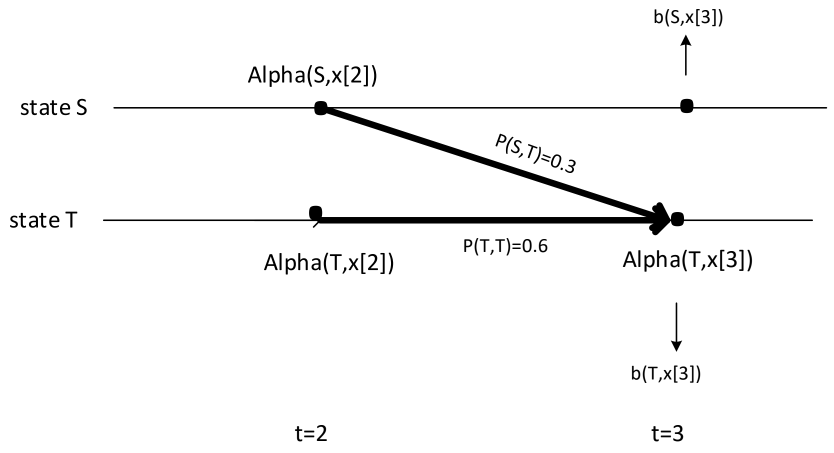


Figure 4.4: Recursive step for the α method for state T

In the above example, for $\alpha_T(3)$ we obtain

$$\alpha_T(3) = b_T(3)[\alpha_T(2)P_{TT} + \alpha_S(2)P_{ST}]$$

Hence, to find the probability of the sequence up to and including x_3, the final answer would be

$$\begin{aligned}\alpha(x_3) &= \alpha_T(3) + \alpha_S(3)\\ &= b_S(3)[\alpha_S(2)P_{SS} + \alpha_T(2)P_{TS}] + b_T(3)[\alpha_T(2)P_{TT} + \alpha_S(2)P_{ST}]\end{aligned}$$

4.1.1 Calculation of the alpha's using logarithms$^{\frown\frown}$E$^{\frown\frown}$L

Due to numerical issues (underflow) which will occur when multiplying probabilities (values less than 1.0) repeatedly as we calculate the α in the forward direction, the log of the probabilities will be used instead, and multiplications become summations. Any logarithm for a base greater than 1 can be used. In the MATLAB implementation, I used the natural logarithms e.

Initially, all the probabilities P (this includes the initial state probabilities $\pi_{(0,k)}$, the state transition probabilities matrix P_{xy}, and the emission probabilities matrix $b_S(i)$) are converted to $log(P)$ and stored in cache for later access, this speeds the implementation by not having to call the log function each time the program is run for different observation sequence but for the same HMM.

Assume the system has states $\{A, B, C, \ldots, Z\}$, and we wish to find the α for the observation sequence up to and including $x[i]$ when the state at i was T. We obtain

$$\alpha_T(i) = b_T(i)[\alpha_A(i-1)P_{AT} + \alpha_B(i-1)P_{BT} + \cdots + \alpha_Z(i-1)P_{ZT}]$$

Taking the log of the above results in

$$log(\alpha_T(i)) = log(b_T(i)) + log[\alpha_A(i-1)P_{AT} + \alpha_B(i-1)P_{BT} + \cdots + \alpha_Z(i-1)P_{ZT}]$$

Now, looking at the term $log[\alpha_A(i-1)P_{AT} + \cdots + \alpha_Z(i-1)P_{ZT}]$, this is the same as $log(x+y+\cdots+z)$. Here, we know the values of $log(x), log(y), \cdots, log(z)$, and not $x, y, \cdots, z$ (recall that we have converted all the probabilities to log initially for performance purposes). So we need to find $log(x+y)$ given $log(x)$ and $log(y)$.

This is done using the following transformation

$$\begin{aligned} log(x+y) &= log(x(1+\frac{y}{x})) \\ &= log(x(1+e^{log(\frac{y}{x})}) \\ &= log(x) + log(1+e^{log(y)-log(x)}) \end{aligned}$$

Applying the above transformation to many terms being summed, and not just two, is straightforward. Take x as the first term, and let y be the rest of the terms (the tail of the expression), we then apply the above transformation recursively to the tail to compute the log of the sum of the probabilities as long as the tail is of length greater than one element. As shown below.

$$log(a+b+c+\cdots+y+z) = log(a+H) = log(a) + log(1+e^{log(H)-log(a)})$$

Where

$$H = (b+c+\cdots+y+z)$$

Now repeat the above process to find $log(b+c+\cdots+y+z)$.

$$log(b+c+\cdots+y+z) = log(b+H) = log(b) + log(1+e^{log(H)-log(b)})$$

Where now $H = (c+\cdots+y+z)$.

Continue the above recursive process, until the expression H contains one term to obtain

$$log(y+z) = log(y) + log(1+e^{log(z)-log(y)})$$

This is a sketch of a function which accomplishes the above.

```
function Log_Of_Sums(A: array 1..n of probabilities) returns double IS
   begin
     IF (length(A)>2) THEN
         return( log(A[1]) + log( 1 + exp( Log_of_Sums(A[2..n])) - log( A[1] )));
     ELSE
          return( log(A[1] + log( 1 + exp( A[2] - log( A[1] )));
    END
end log_Of_Sums
```

The number of states is usually much less than the number of observations, so there is no problems (such as stack overflow issues) with using recursion in this case.

4.2 Backward β algorithm^^E^^L

Here we are interested in finding the probability that some part of the sequence was generated when the system was in some specific state.

For example, we might want to find, looking at a DNA sequence x, the probability that some nucleotide at some position, say $x[i]$ was generated from an exon, where an exon is one of the possible states the system can be in.

Let such state be S. Let the length of the sequence be L, then we need to obtain $P(\pi_i = S|x_{i\ldots L})$. In the following, when x is written without subscripts, it is understood to stand for the whole sequence $x_1\cdots x_L$. Also x_i is the same as $x[i]$.

Using Baye's probability rule we get

$$P(\pi_i = S|x) = \frac{P(\pi_i = S, x)}{P(x)}$$

But the joint probability $P(\pi_i = S, x)$ can be found from

$$P(\pi_i = S, x) = P(x_1\cdots x_i, \pi_i = S)P(x_{i+1}\cdots x_L|\pi_i = S)$$

Looking at the equation above, we see that $P(x_1\ldots x_i, \pi_i = S)$ is nothing but $\alpha_S(i)$, which is defined as the probability of the sequence $x_1\cdots x_i$ with $x[i]$ being generated from state S.

The second quantity $P(x_{i+1}\cdots x_L|\pi_i = S)$ is found using the backward β algorithm as described below.

Initialization step:

For each state k, let $\beta_k(L) = 1$

recursive step:

$$i = (L-1)\cdots 1 \qquad \beta_k(i) = \sum_l P_{kl} b_l(i+1)\beta_l(i+1)$$

Where the sum above is done over all states. Hence

$$P(\pi_i = S|x) = \frac{\alpha_S(i)\beta_S(i)}{P(x)}$$

Where $P(x)$ is found by running the forward algorithm to the end of the observed sequence x, so $P(x) = \alpha(L)$. Hence we obtain

$$P(\pi_i = S|x) = \frac{\alpha_S(i)\beta_S(i)}{\alpha(L)}$$

To illustrate this algorithm, Using the grid HMM diagram, I will use the example on page 13, to answer the question of what is the probability of the observation at position 3 (which is the letter 'A') coming from state S, given the observation ATACC.

In the above, we will run the forward α algorithm from position 1 to position 3, then run the backward algorithm from the end of the sequence back to position 3. (In practice, the forward $\alpha's$ and backward $\beta's$ are calculated once for all the states and all the positions (that is, for all the nodes in the trellis diagram) and stored for later lookup). See figure 4.5 on page 23 for an illustration.

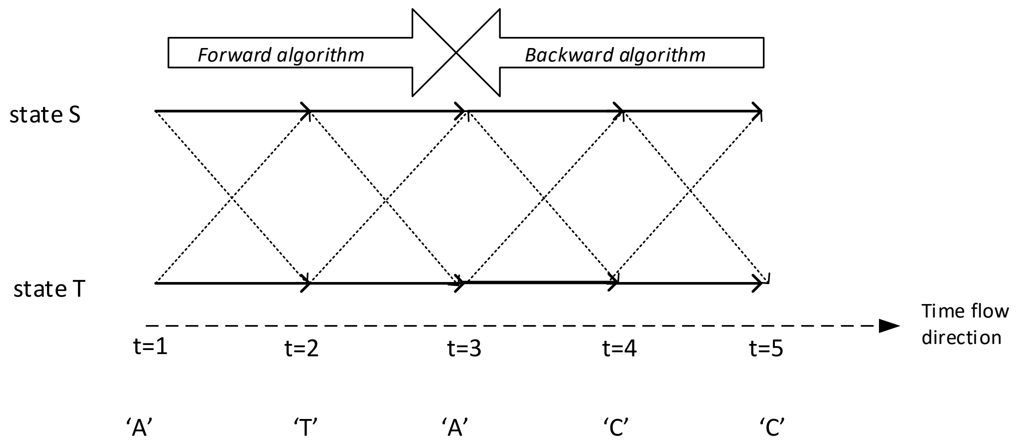


Figure 4.5: The forward-backward algorithm

Initialization step:

$$\beta_S(5) = 1$$
$$\beta_T(5) = 1$$

recursive step: $t = (5-1)\cdots 1$.

At time $t = 4$, we get

$$\beta_S(4) = P_{ST}b_T(5)\beta_T(5) + P_{SS}b_S(5)\beta_S(5)$$
$$\beta_T(4) = P_{TT}b_T(5)\beta_T(5) + P_{TS}b_S(5)\beta_S(5)$$

The above quantities are calculated using the parameters in this example, resulting in

$$\beta_S(4) = 0.3 \times 0.55 \times 1 + 0.7 \times 0.4 \times 1 = 0.445$$
$$\beta_T(4) = 0.6 \times 0.55 \times 1 + 0.4 \times 0.4 \times 1 = 0.49$$

at time $t = 3$ we get

$$\beta_S(3) = P_{ST}b_T(4)\beta_T(4) + P_{SS}b_S(4)\beta_S(4)$$
$$\beta_T(3) = P_{TT}b_T(4)\beta_T(4) + P_{TS}b_S(4)\beta_S(4)$$

The above quantities are calculated using the parameters in this example, resulting in

$$\beta_S(3) = 0.3 \times 0.55 \times 0.49 + 0.7 \times 0.4 \times 0.445 = 0.2054$$
$$\beta_T(3) = 0.6 \times 0.55 \times 0.49 + 0.4 \times 0.4 \times 0.445 = 0.2329$$

And so forth, until we reach $t = 1$

$$\beta_S(1) = P_{ST}b_T(2)\beta_T(2) + P_{SS}b_S(2)\beta_S(2)$$
$$\beta_T(1) = P_{TT}b_T(2)\beta_T(2) + P_{TS}b_S(2)\beta_S(t)$$

This completes the iterative step of the backward algorithm. From the above we see that $\beta_S(3) = 0.205449$ Now, apply the forward algorithm from position 1 to position 5, and find $\alpha_S(3)$.

I will not show this step here, since the forward algorithm is already described above. But if we calculate it, we will get $\alpha(S, t = 3) = 0.012488$

So

$$P(\pi_3 = S|ATACC) = \frac{\alpha_S(3)\beta_S(3)}{P(x)} = \frac{0.012488 \times 0.205449}{P(x)} = \frac{0.0025656}{P(x)}$$

Where $P(x)$ is the same as $\alpha(L)$, that is, the forward algorithm found at the end of the sequence x. $\alpha(L)$ can be found to be 0.0046025920, so

$$P(\pi_3 = S|ATACC) = \frac{0.0025656}{0.0046025920} = 0.557438$$

4.3 Viterbi algorithm^^E^^L

The Viterbi algorithm is used to find the most likely state transition sequence $q = \pi_1 \cdots \pi_i$ associated with some observation sequence $x = x_i \cdots x_i$.

Since one symbol is generated each time the system switches to a new state, the state sequence q will be of the same length as that of the observation sequence x.

In this algorithm, we need to calculate γ for each position, where $\gamma_k(i)$ means the probability of the most likely state path ending in state k when generating the symbol at position i in the observation sequence.

If we know $\gamma_k(i)$ for each state k, then we can easily find $\gamma_l(i+1)$ for each state l, since we just need to find the most probable state transition from each state k to each state l.

The following algorithm finds $\gamma_k(i)$ at position i for state k.

Initialization step:

Find γ at position $i = 1$ for all states. Assume we have n states, $a, b, c, \ldots, z$, then

$$\begin{aligned}
\gamma_a(1) &= P_{0a}\, b_a(1) \\
\gamma_b(1) &= P_{0b}\, b_b(1) \\
&\vdots \\
\gamma_z(1) &= P_{0z}\, b_z(1)
\end{aligned}$$

Now we need to calculate γ for the rest of the sequence.

Assume we have states $a, b, m, c, \ldots, z$, then to find γ for each of these states at each time step we do the following: (this below shows finding γ for some state m)

iterative step: $j = 2 \ldots i$

$$\gamma_m(j) = b_m(j) \;\; max \begin{cases} P_{am}\,\gamma_a(j-1) \\ P_{bm}\,\gamma_b(j-1) \\ P_{mm}\,\gamma_m(j-1) \\ \vdots \\ P_{zm}\,\gamma_z(j-1) \end{cases}$$

The above is illustrated in figure 4.6 on page 26.

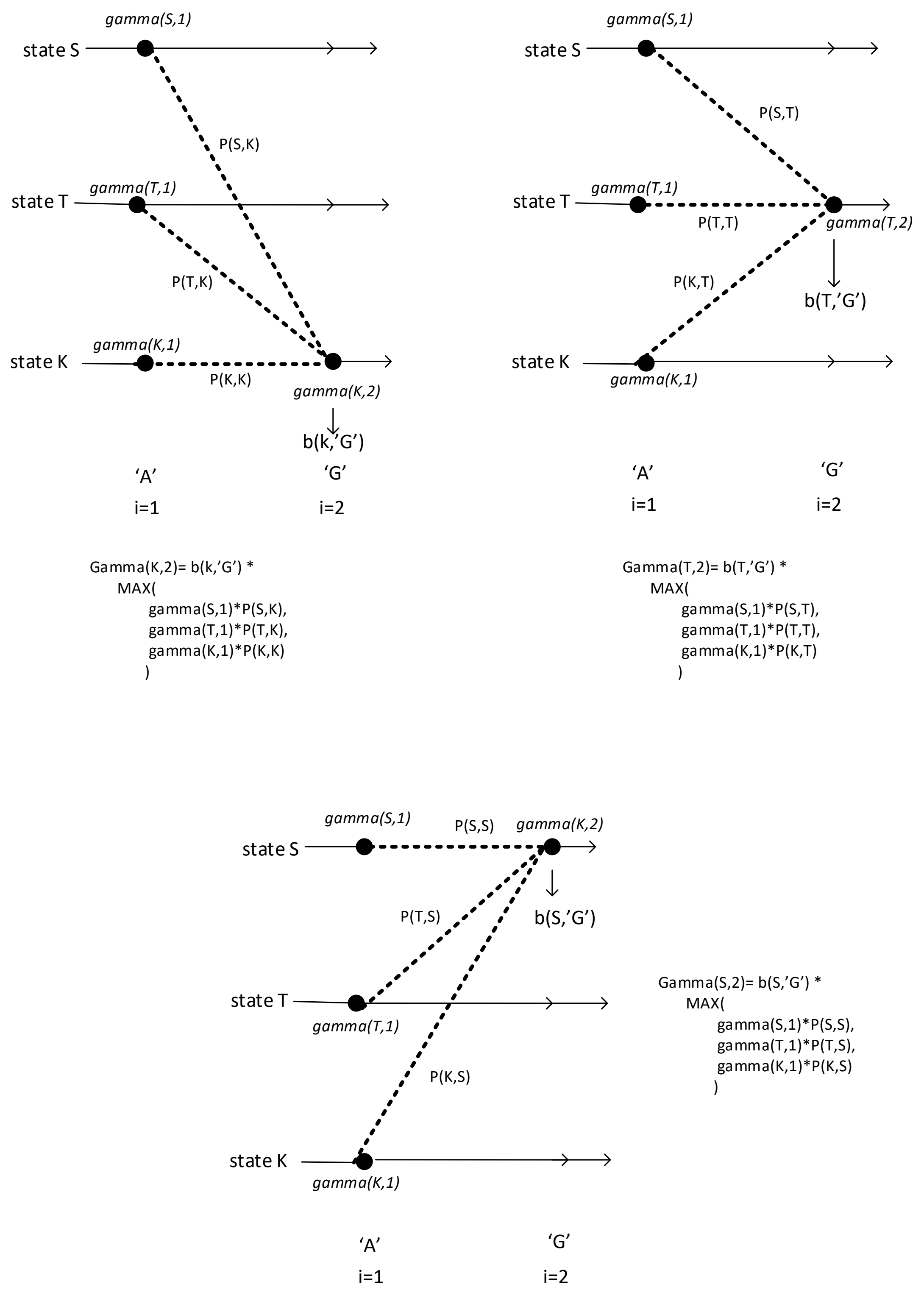


Figure 4.6: The Viterbi iterative step to find γ for each position viterbi.pdf

Now that we have found γ for each state at each position $1 \cdots i$, we can find the most likely state sequence up to any position $j <= i$ by finding which state had the largest γ at each position.

CHAPTER 5

DESIGN AND USER GUIDE OF SHMM

To help understand HMMs more, I have implemented, in MATLAB, a basic HMM with a GUI user interface for ease of use. This tool accepts an HMM description (model parameters) from a plain text configuration file. The tool (called SHMM, for simple HMM) implements the forward, backward, and Viterbi algorithms. This implementation is for a first order HMM model.

Due to lack of time to implement an HMM for an order greater than a first order, which is needed to enable this tool to be used for such problems as gene finding (to detect splice sites, start of exon, etc. . .). This implementation therefor is restricted to one character look-ahead on the observation sequence. However, even with this limitation, this tool can be useful for learning how HMM's work.

SHMM takes as input a plain text configuration file (default .hmm extension), which contains the description of the HMM (states, initial probabilities, emission probabilities, symbols and transition probabilities). And the user is asked to type in the observation sequence in a separate window for analysis.

5.1 Configuration file description^^E^^L

The configuration file is broken into 5 sections. The order of the sections must be as shown below. All 5 sections must exist. Each section starts with the name of the section, contained inside a left < and a right > brackets. No space is allowed after < or before >. The entries in the file are case sensitive. So a state called 'sunny' will be taken as different from a state called 'SUNNY'. The same applies to the characters allowed in the observation sequence: the letter 'g' is taken as different from 'G'.

The sections, and the order in which they appear in the configuration file, must be as this, from top to bottom:

1. `<states>`
2. `<init_prob>`
3. `<symbols>`
4. `<emit_prob>`
5. `<tran_prob>`

In the `<states>` sections, the names of the states are listed. One name per line. Names must not contain a space.

In the `<init_prob>` section, the initial probabilities of being in each state are listed. One per line. One number per line. The order of the states is assumed the same as the order in the `<states>` section. This means the initial probability of being in the first state listed in the `<states>` section will be assumed to be the first number in the `<init_prob>` section.

In the `<symbols>` section, the observation symbols are listed. Each symbol must be one character long, separated by a comma. Can use as many lines as needed to list all the symbols. The symbol must be an alphanumeric character.

In the `<emit_prob>` section, we list the probability of emitting each symbol from each state. The order of the states is assumed to be the same order of the states as in the <states> section.

The emission probability for each symbol is read. The order of the symbols is taken as the same order shown in the symbols section. The emission probabilities are separated by a comma. One line per state. See examples below.

In the `<tran_prob>`, we list the transition probability matrix. This is the probability of changing state from each state to another. One line represent one row in the matrix. Entries are separated by a comma. One row must occupy one line only. The order of the rows is the same as the order of the states shown in the `<states>` section.

For example, if we have 2 states S1,S2, then the first line will be the row that represents `S1->S1`, and `S1->S2` transition probabilities. Where `S1->S1` is in position 1,1 of matrix, and `S1->S2` will be in position 1,2 in the matrix. See example below for illustration.

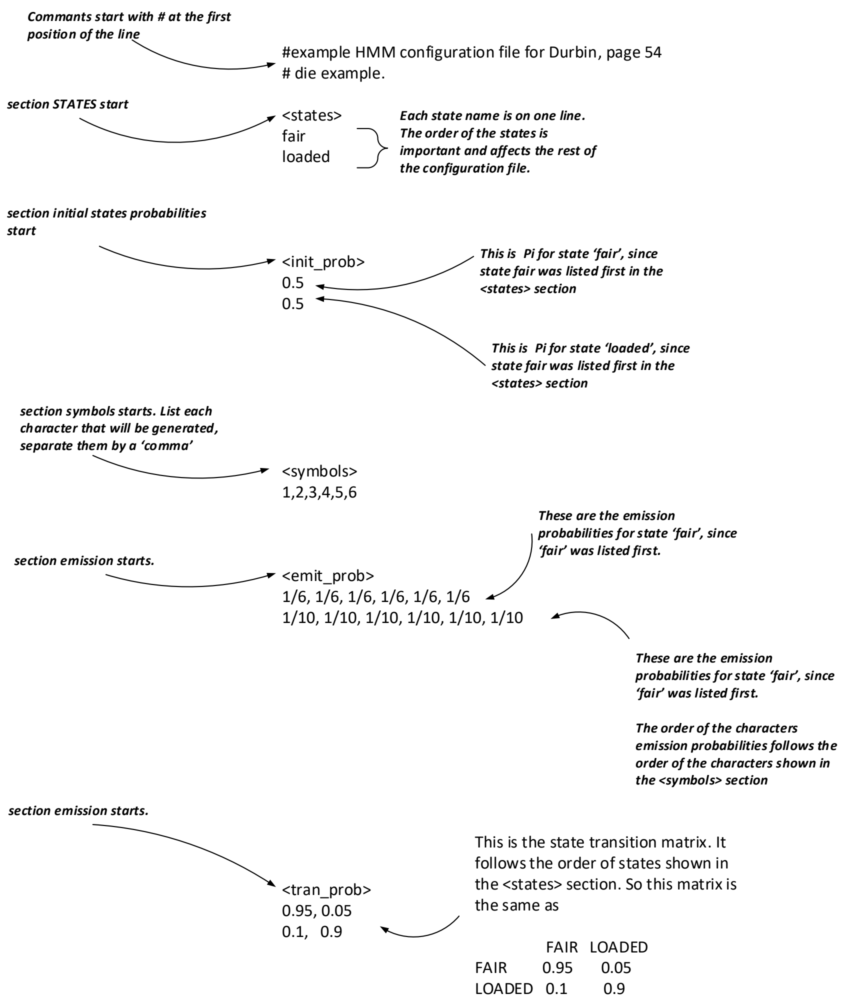
Comments in the configuration file are supported. A comment can start with the symbol `#`. Any line that starts with a `#` is ignored and not parsed. The comment symbol `#` must be in the first position of the line.

To help illustrate the format of the HMM configuration file, I will show an annotated configuration file for the casino example shown in the book 'Biological sequence analysis' by Durbin et. all[1], page 54. In this example, we have 2 dies, one is a fair die, and one is a loaded die. The following is the HMM configuration file.

```
# example HMM configuration file for Durbin, page 54
# die example.

<states>
fair
```

```
loaded

<init_prob>
#fair init prob
0.5

#loaded init prob
0.5

<symbols>
1,2,3,4,5,6

<emit_prob>
#emit probability from fair state
1/6, 1/6, 1/6, 1/6, 1/6, 1/6
#emit probability from fair state
1/10, 1/10, 1/10, 1/10, 1/10, 1/10

<tran_prob>
0.95, 0.05
0.1,   0.9
```

See figure 5.1 on page 32 for description of each field in the HMM configuration file.



Commants start with # at the first position of the line

section STATES start

Each state name is on one line. The order of the states is important and affects the rest of the configuration file.

section initial states probabilities start

This is Pi for state 'fair', since state fair was listed first in the <states> section

This is Pi for state 'loaded', since state fair was listed first in the <states> section

section symbols starts. List each character that will be generated, separate them by a 'comma'

section emission starts.

These are the emission probabilities for state 'fair', since 'fair' was listed first.

These are the emission probabilities for state 'fair', since 'fair' was listed first.

The order of the characters emission probabilities follows the order of the characters shown in the <symbols> section

section emission starts.

This is the state transition matrix. It follows the order of states shown in the <states> section. So this matrix is the same as

```
          FAIR   LOADED
FAIR      0.95     0.05
LOADED    0.1      0.9
```

```
#example HMM configuration file for Durbin, page 54
# die example.

<states>
fair
loaded

<init_prob>
0.5
0.5

<symbols>
1,2,3,4,5,6

<emit_prob>
1/6, 1/6, 1/6, 1/6, 1/6, 1/6
1/10, 1/10, 1/10, 1/10, 1/10, 1/10

<tran_prob>
0.95, 0.05
0.1,   0.9
```

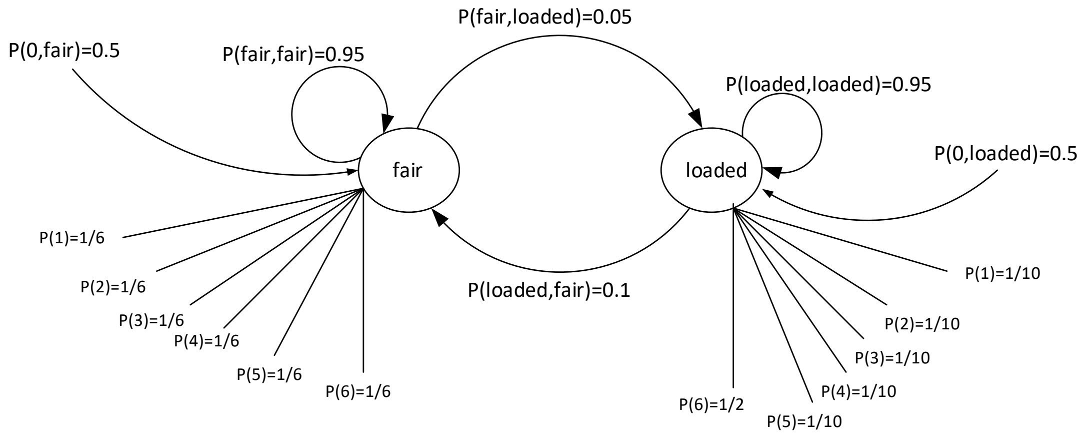
Figure 5.1: Description of fields in the HMM configuration file. The HMM parameters

5.2 Configuration file examples^^E^^L

To help better understand the format of the HMM configuration file, I will show below a number of HMM states and with the configuration file for each.

5.2.1 Example 1. Casino example^^E^^L

This example is taken from page 54, Durbin et. all book [1]. See figure 5.2 on page 33



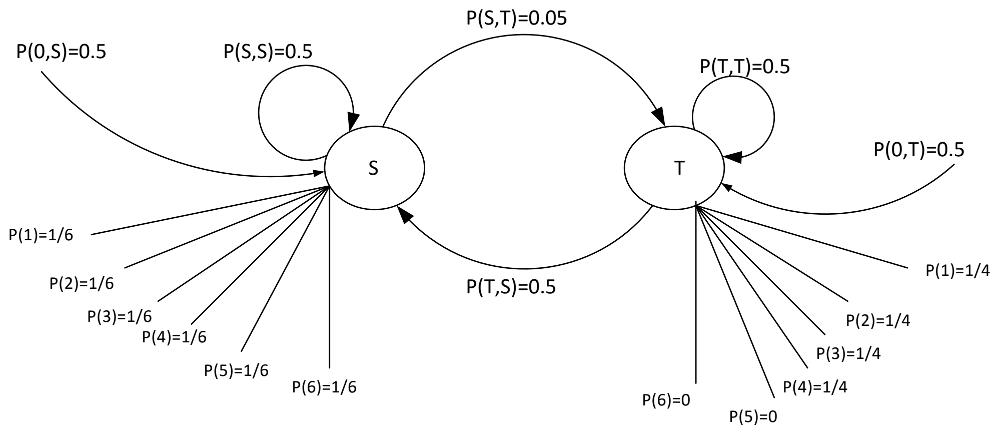
```
<states>
fair
loaded

<init_prob>
#fair init prob
0.5

#loaded init prob
0.5

<symbols>
1,2,3,4,5,6

<emit_prob>
#emit probability from fair state
 1/6, 1/6, 1/6, 1/6, 1/6, 1/6
#emit probability from fair state
1/10, 1/10, 1/10, 1/10, 1/10, 1/10

<tran_prob>
0.95, 0.05
0.1,   0.9
```

Figure 5.2: Example 1 HMM configuration file

5.2.2 Example 2. Homework problem 4.1^^E^^L

This example is taken from Math 127 course, homework problem 4.1, Fall 2002.

See figure 5.3 on page 34



```
<states>
S
T

<init_prob>
0.5
0.5

<symbols>
1, 2, 3,4,5,6

<emit_prob>
0.166666667, 0.166666667, 0.166666667, 0.166666667, 0.166666667, 0.166666667

0.25, 0.25, 0.25 , 0.25 , 0 , 0

<tran_prob>
0.5,   0.5
0.5,   0.5
```

Figure 5.3: Example 2 HMM configuration file

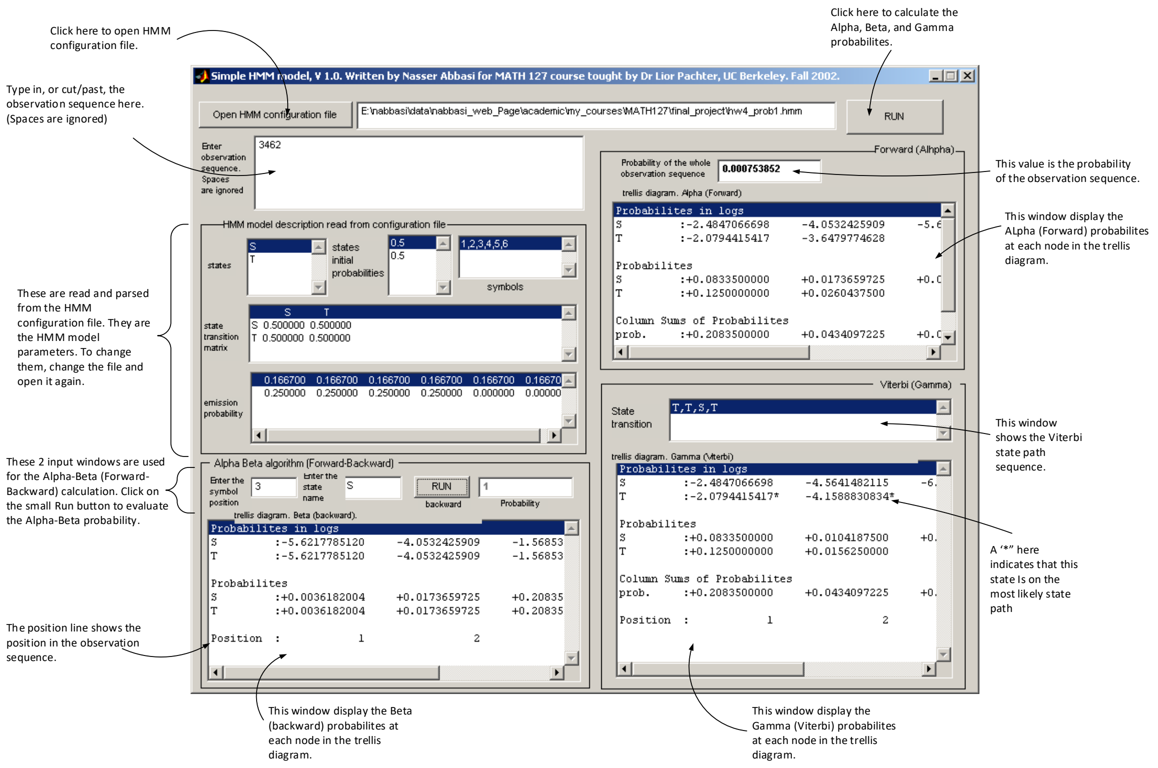
5.3 SHMM User interface^^E^^L

Figure 5.4 on page 35 shows the SHMM GUI. To start using SHMM, the first thing to do is to load the HMM configuration file which contains the HMM parameters. SHMM will parse the file and will display those parameters. Next, type into the observation sequence window the observation sequence itself. This sequence could be cut and pasted into the window as well. Spaces and new lines are ignored.

Next, hit the top right RUN button. This will cause SHMM to evaluate the α, β and γ for each node in the trellis diagram, and will display all these values. For the γ calculations, the nodes in the trellis diagram which are on the Viterbi path have a '*' next to them.

To invoke the (α, β) algorithm, enter the position in the observation and the state name and hit the small RUN button inside the window titled `Alpha Beta`.

To modify the HMM parameters and run SHMM again, simply use an outside editor to do the changes in the HMM configuration file, then load the file again into SHMM using the 'Open' button.


Simple HMM model, V 1.0. Written by Nasser Abbasi for MATH 127 course tought by Dr Lior Pachter, UC Berkeley. Fall 2002.
Open HMM configuration file
RUN
Enter observation sequence. Spaces are ignored
Forward (Alhpha)
Probability of the whole observation sequence
trellis diagram. Alpha (Forward)
HMM model description read from configuration file
states
states initial probabilities
symbols
state transition matrix
emission probability
Viterbi (Gamma)
State transition
trellis diagram. Gamma (Viterbi)
Alpha Beta algorithm (Forward-Backward)
Enter the symbol position
Enter the state name
RUN
backward
Probability
trellis diagram. Beta (backward).

Click here to open HMM configuration file.

Type in, or cut/past, the observation sequence here. (Spaces are ignored)

These are read and parsed from the HMM configuration file. They are the HMM model parameters. To change them, change the file and open it again.

These 2 input windows are used for the Alpha-Beta (Forward-Backward) calculation. Click on the small Run button to evaluate the Alpha-Beta probability.

The position line shows the position in the observation sequence.

Click here to calculate the Alpha, Beta, and Gamma probabilites.

This value is the probability of the observation sequence.

This window display the ALpha (Forward) probabilites at each node in the trellis diagram.

This window shows the Viterbi state path sequence.

A '*' here indicates that this state Is on the most likely state path

This window display the Beta (backward) probabilites at each node in the trellis diagram.

This window display the Gamma (Viterbi) probabilites at each node in the trellis diagram.

```
E:\nabbasi\data\nabbasi_web_Page\academic\my_courses\MATH127\final_project\hw4_prob1.hmm
```

```
3462
```

```
0.000753852
```

```
Probabilites in logs
S          :-2.4847066698      -4.0532425909      -5.6
T          :-2.0794415417      -3.6479774628

Probabilites
S          :+0.0833500000      +0.0173659725      +0.0
T          :+0.1250000000      +0.0260437500

Column Sums of Probabilites
prob.      :+0.2083500000      +0.0434097225      +0.0
```

```
S
T
```

```
0.5
0.5
```

```
1,2,3,4,5,6
```

```
     S        T
S  0.500000  0.500000
T  0.500000  0.500000
```

```
 0.166700  0.166700  0.166700  0.166700  0.166700  0.16670
 0.250000  0.250000  0.250000  0.250000  0.000000  0.00000
```

```
T,T,S,T
```

```
3        S        1
```

```
Probabilites in logs
S          :-5.6217785120      -4.0532425909      -1.56853
T          :-5.6217785120      -4.0532425909      -1.56853

Probabilites
S          :+0.0036182004      +0.0173659725      +0.20835
T          :+0.0036182004      +0.0173659725      +0.20835

Position   :            1                  2
```

```
Probabilites in logs
S          :-2.4847066698      -4.5641482115      -6.
T          :-2.0794415417*     -4.1588830834*

Probabilites
S          :+0.0833500000      +0.0104187500      +0.
T          :+0.1250000000      +0.0156250000

Column Sums of Probabilites
prob.      :+0.2083500000      +0.0434097225      +0.

Position   :            1                  2
```

Figure 5.4: SHMM MATLAB user interface

5.4 Source code description^^E^^L

The following is a list of all the MATLAB files that make up SHMM, with a brief description of each file. At the end of this report is the source code listing.

- nma_hmmGUI.fig: The binary description of the GUI itself. It is automatically generated by MATLAB using GUIDE.
- nma_hmmGUI.m: The MATLAB code that contains the properties of each component in the GUI.
- nma_hmm_main.m: This is the main function, which loads the GUI and initializes SHMM.
- nma_hmm_callbacks.m: The callbacks which are called when the user activate any control on the GUI.

- nma_readHMM.m:
- nma_hmmBackward.m: The function which calculates β.
- nma_hmmForward.m: The function which calculates α.
- nma_hmm_veterbi.m: The function which calculates γ.
- nma_hmmEmitProb.m: A utility function which returns the emission probability given a symbol and a state.
- nma_trim.m: A utility function which removes leading and trailing spaces from a string.
- getPosOfState.m: A utility function which returns position of state in an array given the state name.
- nma_logOfSums.m: A utility function which calculates the log of sum of logs for the α computation.
- nma_logOfSumsBeta.m: A utility function which calculates the log of sum of logs for the β computation.

5.5 Installation instructions^^E^^L

Included on the CDROM are 2 files: setup.exe and unix.tar. The file setup.exe is the installer for windows. On windows, simply double click the file setup.exe and follow the instructions. When installation is completed, an icon will be created on the desktop called *MATH127*. Double click on this icon to open. Then 2 icons will appear. One of them is titled HMM. Double click on that to start SHMM. A GUI will appear.

On Unix (Linux or Solaris), no installer is supplied. Simply untar the file math127.zip. This will create the same tree that the installer would create on windows. The tree contains a folder called *src*, which contains the source code needed to run SHMM from inside matlab. To run SHMM on Unix, cd to the folder where you untarred the file math127.tar, which will be called *MATH127* by default, and navigate to the src folder for the HMM project. Then run the file *nma_hmm_main* from inside MATLAB console to start SHMM.

This report is in PDF format.

Example HMM configuration files for use with SHMM are included in the *conf* folder.

CHAPTER 6

APPENDIX

6.1 User guide and download^^E^^L

This contains information how to download the windows executable and install it on your PC.

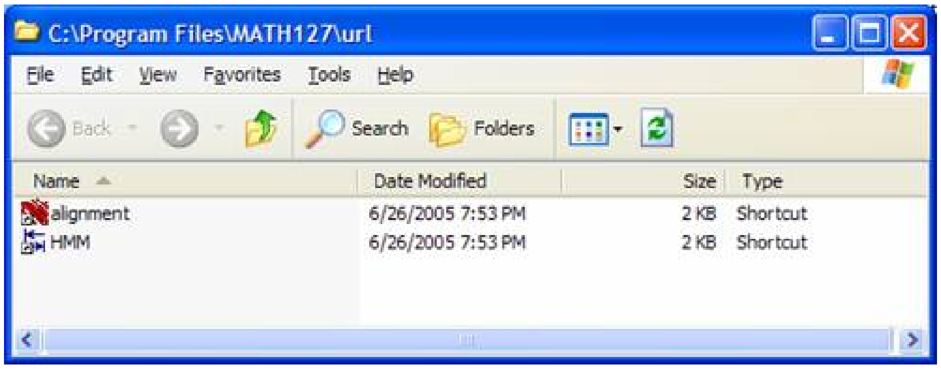
6.1.1 Multiple alignment applications installation^^E^^L

These instructions show how to install the HMM application on windows.

1. Download the file setup.exe (12 MB) to some folder on your PC.
2. Once setup.exe is downloaded, Double click on it to run it. This will bring up a standard installation. Click next on the questions being asked.
3. When the installation is completed, double click on icon created on your desktop, it will labeled Math127.
4. Now you will see 2 icons that represent 2 tools. One for the HMM application and another for a multiple alignment application.
5. To run either application, double click on the icon. This will bring up the application.

6.1.2 Instruction for running and using HMM˜˜E˜˜L

Double click on the HMM icon as per the above instruction.



Name	Date Modified	Size	Type
alignment	6/26/2005 7:53 PM	2 KB	Shortcut
HMM	6/26/2005 7:53 PM	2 KB	Shortcut

Figure 6.1: Screen shot

Now the main screen will come up

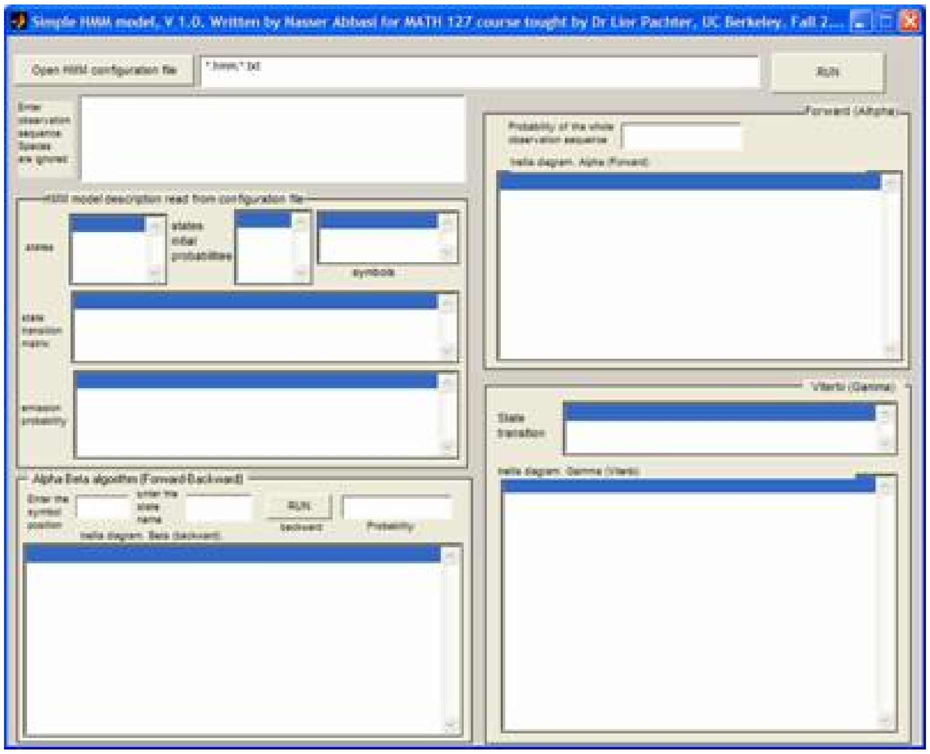
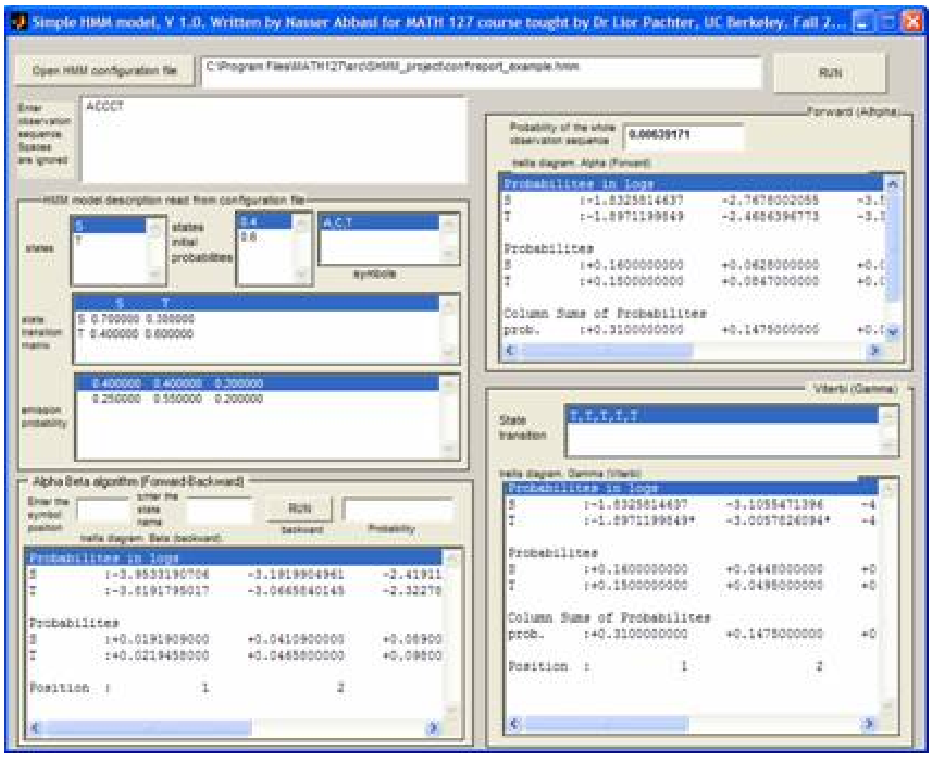

Simple HMM model, V 1.0. Written by Nasser Abbasi for MATH 127 course tought by Dr Lior Pachter, UC Berkeley. Fall 2...
Open HMM configuration file
.hmm;.txt
RUN
Enter observation sequence Spaces are ignored
HMM model description read from configuration file
states
states initial probabilities
symbols
state transition matrix
emission probability
Alpha Beta algorithm (Forward-Backward)
Enter the symbol position
enter the state name
RUN
backward
Probability
trellis diagram. Beta (backward)
Forward (Alpha)
Probability of the whole observation sequence
trellis diagram. Alpha (Forward)
Viterbi (Gamma)
State transition
trellis diagram. Gamma (Viterbi)

Figure 6.2: Main Screen

Now click the `Open HMM configuration file` to open the HMM file that describes the system being modeled. I include few files with this installation. Navigate as follows to find the files `Open->src->SHMM_project->conf` and select the file `report_example`.

Now enter the observation sequence in the window (i.e. type the sequence). The sequence must contain symbols that belong to the allowed set of symbols as shown in the `symbols` window. For the `report_example`, these are "A" and "C" and "T" only. So you can type in `ACCCCTTT` for an example.

Now hit the RUN botton on the upper right hand corner. This is the result


Simple HMM model, V 1.0, Written by Nasser Abbasi for MATH 127 course tought by Dr Lior Pachter, UC Berkeley. Fall 2...
Open HMM configuration file
RUN
Enter observation sequence Spaces are ignored
HMM model description read from configuration file
states
states initial probabilities
symbols
state transition matrix
emission probability
Alpha Beta algorithm (Forward-Backward)
Enter the symbol position
Enter the state name
RUN
backward
Probability
trellis diagram. Beta (backward)
Forward (Alpha)
Probability of the whole observation sequence
trellis diagram. Alpha (Forward)
Viterbi (Gamma)
State transition
trellis diagram. Gamma (Viterbi)

```
C:\Program Files\MATH127\src\SHMM_project\conf\report_example.hmm
```

```
ACCCT
```

```
S
T
```

```
0.4
0.6
```

```
A,C,T
```

```
    S        T
S 0.700000 0.300000
T 0.400000 0.600000
```

```
  0.400000   0.400000   0.200000
  0.250000   0.550000   0.200000
```

```
0.00639171
```

```
Probabilites in logs
S          :-1.8325814637     -2.7678002055     -3.5
T          :-1.8971199549     -2.4686396773     -3.3

Probabilites
S          :+0.1600000000     +0.0628000000     +0.0
T          :+0.1500000000     +0.0847000000     +0.0

Column Sums of Probabilites
prob.      :+0.3100000000     +0.1475000000     +0.1
```

```
T,T,T,T,T
```

```
Probabilites in logs
S          :-1.8325814637     -3.1055471396     -4
T          :-1.8971199549*    -3.0057826094*    -4

Probabilites
S          :+0.1600000000     +0.0448000000     +0
T          :+0.1500000000     +0.0495000000     +0

Column Sums of Probabilites
prob.      :+0.3100000000     +0.1475000000     +0

Position  :              1                 2
```

```
Probabilites in logs
S          :-3.9533190706     -3.1919904961     -2.41911
T          :-3.8191795017     -3.0665840145     -2.32270

Probabilites
S          :+0.0191909000     +0.0410900000     +0.08900
T          :+0.0219458000     +0.0465800000     +0.09800

Position  :              1                 2
```

Figure 6.3: Result of the above

6.2 Source code^^E^^L

Source code in zip file

The following is the listing

6.2.1 nma_hmm_main.m^^E^^L

```
function  nma_hmm_main()

%main line for SHMM. MATH 127
%Nasser Abbasi


h0= hgload('nma_hmmGUI');

nma_hmm_callbacks('init',h0);
```

6.2.2 nma_trim.m^^E^^L

```
function line=nma_trim(line)
%function line=nma_trim(line)
%
%removes leading and trailing spaces from the line
%
%Nasser Abbasi. Math 127.

N=length(line);
if(N==0)
   return;
end

i=1;
while(i<=N & (line(i)==' '|line(i)=='\t'))
   i=i+1;
end

if(i>N)
   line='';
   return;
end

line=line(i:end);

i=length(line);
```

```
while(i>=1 & (line(i)==' '|line(i)=='\t'))
   i=i-1;
end

line=line(1:i);
```

6.2.3 nma_readHMM.m^^E^^L

```
function [symbols, initProb, states, tranProb, emitProb, err]=nma_readHMM(fileName)
%function [symbols, initProb, states, tranProb, emissionProb]=nma_readHMM(fileName)
%
% This function read the HMM configuration file for my final project
% for MATH 127, UC Berkeley. Fall 2002.
%
% INPUT:
%   fileName: the full path of the configuration file.
%
% OUTPUT:
%   symbols: a CELL array of characters. The allowed observation symbols.
%
%   initProb: an array of numbers, there will be as many number as there
%             are states
%
%   states: a cell array of strings. The names of the states.
%
%   tranProb: a matrix, nxn, where n is the number of states. the order
%             of states is that indicated by the states array.
%
%   emissionProb: The probability of emission of a symbol from each state.
%                 an nxm matrix, where n is the number of states, and m
%                 is the number of symbols. The order of the states is as
%                 indicated by the states array, and the order of the
%
%                 symbols is as indicated by the symbols array.
%  err: a string, if empty, indicates no error in parsing the file, else
%                 the string shows the parsing error.
%

% by Nasser Abbasi

err='';
symbols={''};
initProb=[];
states={''};
tranProb=[];
emitProb=[];
```

```
[fid,err]=fopen(fileName,'rt');
if(fid==-1)
   return;
end

TRUE=1;
FALSE=0;

INITIAL=0;
STATES=1;
statesFound=FALSE;

INIT_PROB=2;
initProbFound=FALSE;

SYMBOLS=3;
symbolsFound=FALSE;

EMIT_PROB=4;
emitProbFound=FALSE;

TRAN_PROB=5;
tranProbFound=FALSE;

DONE=6;

nState=0;
nInitProb=0;
nSym=0;
nEmitProbStates=0;
nTranProbLines=0;

currentState=INITIAL;

while(currentState ~=DONE)
   switch(currentState)
      case DONE
         if(fid ~= -1)
            fclose(fid);
         end
         return;

      case INITIAL
         %fprintf('initial state\n');
         err=findToken('<states>',fid);
         if(isempty(err))
```

```
            currentState=STATES;
         else
            err=['failed to find states section' err];
            currentState=DONE;
         end


      case STATES
         % looking for list of names, one per line
         %fprintf('states state\n');
         line=fgetl(fid);
         while(line ~= -1)
            if(length(line)==0)
               line=fgetl(fid)
            else
               break;
            end
         end

         if(line==-1)
            err='Invalid HMM file. incomplete states section';
            currentState=DONE;
         else
            line=nma_trim(line);
            if(length(line)>0)
               if(line(1) ~='#')
                  if(isequal(line,'<init_prob>'))
                     if(nState==0)
                        err='No states found in the states section.';
                        currentState=DONE;
                     else
                        currentState=INIT_PROB;
                        statesFound=TRUE;
                     end
                  else
                     nState=nState+1;
                     states(nState)={line};
                  end
               end
            end
         end

      case INIT_PROB
         %looking for name=value, one per line. each name must be
         %allready found in the STATES.
         %fprintf('init_prob state\n');
         line=fgetl(fid);
```

```
        while(line ~= -1)
           if(length(line)==0)
              line=fgetl(fid)
           else
              break;
           end
        end

        if(line==-1)
           err='Invalid HMM file. incomplete init_prob section';
           currentState=DONE;
        else
           line=nma_trim(line);
           if(length(line)>0)
              if(line(1) ~='#')
                 if(isequal(line,'<symbols>'))
                    if(nInitProb==0)
                       err='No init_prob entries found in the init_prob section.';
                       if(fid ~= -1)
                          fclose(fid);
                       end
                       return;
                    end

                    if(nInitProb ~= nState)
                       err='Number of init_prob entries must equal number of states.';
                       if(fid ~= -1)
                          fclose(fid);
                       end
                       return;
                    end

                    currentState=SYMBOLS;
                    initProbFound=TRUE;
                 else
                    %parse value  from line. allready trimmed.
                    i=0;
                    thisValue=0;

                    thisValue=str2double(line);
                    if(thisValue==NaN)
                       err=sprintf('invalid numeric value for probability in init_prob sectio
                       if(fid ~= -1)
                          fclose(fid);
                       end
                       return;
                    end
```

```
               nInitProb=nInitProb+1;
               if(nInitProb>nState)
                  err='Too many entries in init_prob. should equal to number of states'
                  if(fid ~= -1)
                     fclose(fid);
                  end
                  return;
               end

               initProb(nInitProb)=thisValue;
            end
         end
      end
   end

case SYMBOLS
   %parse line with this a,b,c,d format
   %fprintf('symbols state\n');

   line=fgetl(fid);
   while(line ~= -1)
      if(length(line)==0)
         line=fgetl(fid)
      else
         break;
      end
   end

   if(line==-1)
      err='Invalid HMM file. incomplete symbols section';
      return;
   end

   line=nma_trim(line);
   if(length(line)>0)
      if(line(1) ~='#')
         if(isequal(line,'<emit_prob>'))
            if(nSym==0)
               err='No symbols found in the emit_prob section.';
               if(fid ~= -1)
                  fclose(fid);
               end

               return;
            end
            currentState=EMIT_PROB;
         else
            lookForComma=FALSE;
            %parse values from line. allready trimmed.
            if(length(line)==1)
               nSym=nSym+1;
               symbols(nSym)={line};
            else
               i=0;
               while(1)
                  i=i+1;
                  if(i>length(line))
                     break;
                  end

                  while(i<=length(line) & (line(i)==' ' | line(i)=='\t' ))
                     i=i+1;
                  end

                  if(i>length(line))
                     break;
                  end

                  if(lookForComma)
                     if(line(i)~=',')
                        err='expected a comma in symbols section, did not find it';
                        if(fid ~= -1)
                           fclose(fid);
                        end

                        return;
                     else
                        lookForComma=FALSE;
                     end
                  end

                  if(line(i) ~=',')
                     nSym=nSym+1;
                     symbols(nSym)={line(i)};
                     lookForComma=TRUE;
                  end
               end
            end
         end
      end
   end
```

```
      case EMIT_PROB
         %parse value,value,value
         %fprintf('emission_prob state\n');

         line=fgetl(fid);
         while(line ~= -1)
            if(length(line)==0)
               line=fgetl(fid)
            else
               break;
            end
         end

         if(line==-1)
            err='Invalid HMM file. incomplete emit_prob section';
            if(fid ~= -1)
               fclose(fid);
            end

            return;
         end

         line=nma_trim(line);
         if(length(line)>0)
            if(line(1) ~='#')
               if(isequal(line,'<tran_prob>'))
                  if(nEmitProbStates==0)
                     err='No emit_prob entries found in the emit_prob section.';
                     if(fid ~= -1)
                        fclose(fid);
                     end

                     return;
                  end

                  if(nEmitProbStates ~= nState)
                     err='number of lines in emit_prob section must equal to number of states
                     return;
                  end

                  currentState=TRAN_PROB;
               else
                  %parse values from line. allready trimmed.

                  lookForComma=FALSE;
                  nValuesFoundOnThisLine=0;
                  values=[];
                  i=1;
                  while(1)
                     if(line(i)~=',' & (line(i)==' ' | line(i)=='\t'))
                        i=i+1;
                        if(i>length(line))
                           break;
                        end
                     end

                     if(lookForComma)
                        if(line(i)~=',')
                           err='expected a comma in emit_prob section';
                           if(fid ~= -1)
                              fclose(fid);
                           end

                           return;
                        else
                           lookForComma=FALSE;
                           i=i+1;
                        end
                     end

                     if(i>length(line))
                        break;
                     end

                     while(line(i)==' ' | line(i)=='\t')
                        i=i+1;
                        if(i>length(line))
                           break;
                        end
                     end

                     startPos=i;
                     while(i<length(line) & line(i)~=' ' &line(i)~='\t' &line(i)~=',')
                        i=i+1;
                     end

                     if(i==startPos)
                        endPos=i;
                     else
                        if(line(i)==' ' | line(i)=='\t' | line(i)==',')
                           endPos=i-1;
                        else
                           endPos=i;
                        end
                     end

                     value=line(startPos:endPos);
                     value=str2double(value);
                     if(value==NaN)
                        err='Invalid numeric value for emit_prob found';
                        if(fid ~= -1)
                           fclose(fid);
                        end

                        return;
                     end

                     nValuesFoundOnThisLine=nValuesFoundOnThisLine+1;

                     if(nValuesFoundOnThisLine>nSym)
                        err='emit probability line contains too many entries, more than numbe
                        if(fid ~= -1)
                           fclose(fid);
                        end

                        return;
                     end

                     values(nValuesFoundOnThisLine)=value;
                     if(i==length(line))
                        if(nValuesFoundOnThisLine ~= nSym)
                           err='emit_prob section, missing emit_probability for some symbols'
                           if(fid ~= -1)
                              fclose(fid);
                           end

                           return;
                        end
                        nEmitProbStates=nEmitProbStates+1;
                        if(nEmitProbStates>nState)
                           err='number of lines in emit_prob section can not be more than the
                           if(fid ~= -1)
                              fclose(fid);
                           end

                           return;
                        end
                        emitProb(nEmitProbStates,1:nSym)=values(1:end);
                        break;
                     else
                        lookForComma=TRUE;
```

```
                  end
               end
            end
         end
      end

   case TRAN_PROB
      %parse matrix  v11,v12
      %              v21,v22
      %there will be n number of lines, where n is the number of
      %states

      %fprintf('tran_prob state\n');

      line=fgetl(fid);

      while(line ~= -1)
         if(length(line)==0)
            line=fgetl(fid)
         else
            break;
         end
      end

      if(line==-1)
         if(nTranProbLines < nState)
            err='Invalid HMM file. incomplete tran_prob section';
         end

         if(fid ~= -1)
            fclose(fid);
         end

         return;
      end

      line=nma_trim(line);
      if(length(line)>0)
         if(line(1) ~='#')
            if(nTranProbLines==nState)
               err='Too many lines in tran_prob section. can only have same lines as number
               if(fid ~= -1)
                  fclose(fid);
               end

               return;
```

```
          end

          lookForComma=FALSE;
          nValuesFoundOnThisLine=0;
          values=[];
          i=1;
          while(1)
             if(line(i)~=',' & (line(i)==' ' | line(i)=='\t'))
                i=i+1;
                if(i>length(line))
                   break;
                end
             end

             if(lookForComma)
                if(line(i)~=',')
                   err='expected a comma in tran_prob section';
                   if(fid ~= -1)
                      fclose(fid);
                   end

                   return;
                else
                   lookForComma=FALSE;
                   i=i+1;
                end
             end

             if(i>length(line))
                break;
             end

             while(line(i)==' ' | line(i)=='\t')
                i=i+1;
                if(i>length(line))
                   break;
                end
             end

             startPos=i;
             while(i<length(line) & line(i)~=' ' &line(i)~='\t' &line(i)~=',')
                i=i+1;
             end

             if(i==startPos)
                endPos=i;
             else
```

```
                     if(line(i)==' ' | line(i)=='\t' | line(i)==',')
                        endPos=i-1;
                     else
                        endPos=i;
                     end
                  end

                  value=line(startPos:endPos);
                  value=str2double(value);
                  if(value==NaN)
                     err='Invalid numeric value for tran_prob found';
                     if(fid ~= -1)
                        fclose(fid);
                     end

                     return;
                  end

                  nValuesFoundOnThisLine=nValuesFoundOnThisLine+1;

                  if(nValuesFoundOnThisLine>nState)
                     err='tran probability line contains too many entries, more than number of
                     if(fid ~= -1)
                        fclose(fid);
                     end

                     return;
                  end

                  values(nValuesFoundOnThisLine)=value;
                  if(i==length(line))
                     if(nValuesFoundOnThisLine ~= nState)
                        err='tran_prob section, missing tran_probability for some states';
                        if(fid ~= -1)
                           fclose(fid);
                        end

                        return;
                     end
                     nTranProbLines=nTranProbLines+1;
                     if(nTranProbLines>nState)
                        err='number of lines in tran_prob section can not be more than the num
                        if(fid ~= -1)
                           fclose(fid);
                        end

                        return;
                     end
                     tranProb(nTranProbLines,1:nState)=values(1:end);
                     break;
                  else
                     lookForComma=TRUE;
                  end
               end
            end
         end
   end
end



%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%
function err=findToken(token,fid)

err='';
found=0;

while(~found)

   line=fgetl(fid);
   while(line ~= -1)
      if(length(line)==0)
         line=fgetl(fid)
      else
         break;
      end
   end

   if(line==-1)
      err='Invalid HMM file. Expected <states> section, did not find it';
      return;
   end

   line=nma_trim(line);
   if(length(line)>0)
      if(line(1) ~='#')
         if(isequal(line,token))
            %fprintf('found %s\n',token);
            found=1;
         else
            err='Failed to find <states> section where it is expected';
         end
      end
   end
end
```

6.2.4 nma_logOfSumsBeta.m^^E^^L

```
function sum=nma_logOfSumsBeta(beta,tranProb,emitProb)
%function sum=logOfSumsBeta(alpha,tranProb,emit)
%
%INPUT:
%  beta: This is an array of the beta values (backward
%         algorithm)
%  tranProb: This is an array of transition probability
%         from all the states to the state we are calculating
%         beta at.
%  emitProb: the emission probability
%  x: the observation sequence

%By Nasser Abbasi
%Dec 2, 2002
%
% Final project , Math 127, computational biology.


x= emitProb(1) + tranProb(1) + beta(1);

if(length(tranProb)>2)
   y   = nma_logOfSumsBeta( beta(2:end), tranProb(2:end), emitProb(2:end) );
else
   y   = emitProb(2) + tranProb(2) + beta(2);
end

sum = x + log( 1 + exp(y - x ));
```

6.2.5 nma_logOfSums.m^^E^^L

```
function sum=nma_logOfSums(alpha,tranProb)
%function sum=logOfSums(alpha,tranProb)
%
%INPUT:
%  alpha: This is an array of the alpha values (forward
%         algorithm) at t=i-1
%  tranProb: This is an array of transition probability
%         from all the states to the state we are calculating
%         alpha at.

%By Nasser Abbasi
%Dec 2, 2002
%
% Final project , Math 127, computational biology.

x = tranProb(1) + alpha(1);

if(length(tranProb)>2)
   y   = nma_logOfSums(alpha(2:end),tranProb(2:end));
else
   y  = tranProb(2) + alpha(2);
end

sum = x + log( 1 + exp( y - x ));
```

6.2.6 nma_hmmForward.m^^E^^L

```
function [alpha,err]=nma_hmmForward(observation,symbols,initProb,emitProb,tranProb)
%function prob=nma_hmmForward(observation,h.symbols,h.initProb,h.emitProb,h.tranProb)
%
%calculate the HMM forward algorithm
%
%INPUT
%  observation: a char array of the observation sequence
%
%  symbols:     a char array. represents the alphabets allowed
%               in the observation. It is assumed the observation
%               sequence allready contains only valid alphabets
%
%  initProb:    An array of doubles. represents the LOG_e initial probability
%               of the states.
%
%  emitProb:    A matrix nxm. There are n rows for n states, and m columns for
%                m alphabets. The order of the alphabets is taken from the
%                symbols array. The emission probability, in Log_e
%
% tranProb:    The state transition probability matrix. In Log_e
%
% OUTPUT
%   alpha:     Alpha matrix. nxm
%   err:       empty if no errors.
%

err='';

nStates=size(emitProb,1);
alpha=zeros(nStates,length(observation));

for(state=1:nStates)
   [p,err]=nma_hmmEmitProb(state,observation(1),emitProb,symbols);
   if(~isempty(err))
      return;
   end

   alpha(state,1) = initProb(state) + p;
end

for(time=2:length(observation))
   x=observation(time);

   for(currentState=1:nStates)
      [emitProbAtCurrentState,err] = nma_hmmEmitProb(currentState,x,emitProb,symbols);
      if(~isempty(err))
         return;
      end

      sum=nma_logOfSums(alpha(:,time-1),tranProb(currentState,:));

      alpha(currentState,time)=sum + emitProbAtCurrentState;

   end
end
```

6.2.7 nma_hmmEmitProb.m^^E^^L

```
function [emit,err]=nma_hmmEmitProb(state,c,emitProb,symbols)
%function [emit,err]=nma_hmmEmitProb(state,c,emitProb,symbols)
%returns the emit probability of c from state.
%

%Nasser Abbasi, MATH 127.

err='';
emit=0;
nSyms=length(symbols);

for(i=1:nSyms)
   if(c==char(symbols(i)))
      emit=emitProb(state,i);
      return;
   end
end

err=sprintf('observation char %c not found in emit probability matrix.',c);
```

6.2.8 nma_hmmBackward.m^^E^^L

```
function [beta,err]=nma_hmmBackward(observation, symbols, initProb, emitProb, tranProb)
%
%calculate the HMM backward algorithm
%
%INPUT
%  observation: a char array of the observation sequence
%
%  symbols:     a char array. represents the alphabets allowed
%               in the observation. It is assumed the observation
%               sequence allready contains only valid alphabets
%
%  initProb:    An array of doubles. represents the initial probability
%               of the states.
%
%  emitProb:    A matrix nxm. There are n rows for n states, and m columns for
%               m alphabets. The order of the alphabets is taken from the
%               symbols array.
%
% tranProb:    The state transition probability matrix.
%
% OUTPUT
%   beta:      beta matrix. nxm
%   err:       empty if no errors.
%

err='';
beta=[];
nSeq=length(observation);

nStates=size(emitProb,1);
beta=zeros(nStates,nSeq);

for(state=1:nStates)
   beta(state,end) = log(1);
end

for(time=nSeq-1:-1:1)
   x=observation(time+1);
   currentEmitProb=[];
   for(k=1:length(symbols))
      if(x==char(symbols(k)))
         currentEmitProb=emitProb(:,k);
         break;
      end
   end

   for(currentState=1:nStates)
       beta(currentState,time)=nma_logOfSumsBeta(beta(:,time+1),tranProb(currentState,:),curren
   end
end
```

6.2.9 nma_hmm_veterbi.m^^E^^L

```
function [statePath,stateSeq,gamma]=nma_hmm_veterbi(observation,symbols,initProb,emitProb,tranP
%function gamma=nma_hmm_veterbi(observation,initProb,emitProb,tranProb)
%
%
%Find Viterbi Gamma values.
%
% INPUT
%    observation: the observation sequence. a Vector of characters.
%    symbols:     the allowed alphabets of the language, or observation
%    initProb:    a vector of the initial probabilities for each state
%                 in logs.
%    emitProb:    a matrix that represents the emission probability of each
%                 symbol from each state
%    tanProb:     a matrix that represents the state transition
%                 probability.
%
% OUTPUT
%   gamma:        the gamma for each state and each position. a matrix.
%   stateSeq:     the state sequence vector of most likely states.

%by Nasser Abbasi


nState=size(tranProb,1);
statePath=zeros(nState,length(observation));
gamma=zeros(nState,length(observation));

c=observation(1);
theMax=-Inf;

%initial
for(state=1:nState)
   [emit,err]=nma_hmmEmitProb(state,c,emitProb,symbols);
   if(~isempty(err))
      return;
   end

   gamma(state,1)= initProb(state) + emit;

   if(gamma(state,1)>theMax)
      theMax=gamma(state,1);
   end
end

for(time=2:length(observation))
   c=observation(time);

   for(currentState=1:nState)

      [emit,err]=nma_hmmEmitProb(currentState,c,emitProb,symbols);
      if(~isempty(err))
         return;
      end

      theMax=-Inf;

      for(k=1:nState)
         t= tranProb(k,currentState) + gamma(k,time-1);

         if(t>theMax)
            theMax=t;
         end
      end

      %gamma(currentState,time)=theMax*emit;
      gamma(currentState,time)=theMax+ emit;
   end
end


for(time=2:length(observation))
   for(to=1:nState)
      theMax=-Inf;
      for(from=1:nState)
         t= tranProb(from,to) +gamma(from,time-1);

         if(t>theMax)
            theMax=t;
            statePath(to,time)=from;
         end
      end
   end
end

stateSeq=zeros(length(observation),1);
[V,I]=max(gamma(:,end));
stateSeq(end)=I;
for(i=length(observation)-1:-1:1)
   stateSeq(i)=statePath(stateSeq(i+1),i+1);
end
```

6.2.10 nma_hmm_callbacks.m^^E^^L

```
function nma_hmm_callbacks(arg,h0)
%function nma_hmm_callbacks(arg,h0)
%

%
% Final project
% course MATH 127, UC Berkeley, FALL 2002 tought by Dr Lior Pachter.
%

%change history
%nabbasi-112802 started
%nabbasi-0201402 all algorithms implemented OK. Need more GUI work.

switch arg
```

```
    case 'init'
       % called by nma_alignment_main just after loading the GUI

       warning on;

       h = struct( ...
          'isHMMLoaded',0,...
          'fullHMMFileName','',...
          'hmmLastFolder','',...
          'fullLogFileName','',...
          'hmmFileName_tag',0,...
          'logFileName_tag',0,...
          'seq_tag',0,...
          'hmmFID',0,...
          'modelSeq_tag',0,...
          'modelStates_tag',0,...
          'modelStateTranMatrix_tag',0,...
          'modelEmitProbMatrix_tag',0,...
          'modelInitialProb_tag',0,...
          'forward_tag',0,...
          'forward_log_tag',0,...
          'veterbi_tag',0,...
          'grid_tag',0,...
          'forward_grid_tag',0,...
          'backwardPos_tag',0,...
          'backwardState_tag',0,...
          'backwardProb_tag',0,...
          'backwardGrid_tag',0,...
          'symbols',{''},...
          'emitProb',[],...
          'Log_Emit_Prob',[],...
          'states',{''},...
          'tranProb',[],...
          'Log_Tran_Prob',[],...
          'initProb',[],...
          'Log_Init_Prob',[],...
          'beta',[],...
          'alpha',[],...
          'gamma',[],...
          'observation','');

       zoom(h0,'on');

       %
       %  set version number to display and program name and update build date
       %
       set(h0,'Name','Simple HMM model, V 1.0. Written by Nasser Abbasi for MATH 127 course to
       set(h0,'NumberTitle','off');

       %
       % store all GUI handles in userData for quick access in the callbacks
       %
       h.hmmFileName_tag       = findobj(h0,'Tag','hmmFileName_tag');
       h.logFileName_tag       = findobj(h0,'Tag','logFileName_tag');
       h.seq_tag               = findobj(h0,'Tag','seq_tag');
       h.modelSeq_tag          = findobj(h0,'Tag','modelSeq_tag');
       h.modelStates_tag       = findobj(h0,'Tag','modelStates_tag');
       h.modelStateTranMatrix_tag = findobj(h0,'Tag','modelStateTranMatrix_tag');
       h.modelEmitProbMatrix_tag  = findobj(h0,'Tag','modelEmitProbMatrix_tag');
       h.modelInitialProb_tag     = findobj(h0,'Tag','modelInitialProb_tag');
       h.forward_tag            = findobj(h0,'Tag','forward_tag');
       h.forward_log_tag        = findobj(h0,'Tag','forward_log_tag');
       h.veterbi_tag            = findobj(h0,'Tag','veterbi_tag');
       h.grid_tag               = findobj(h0,'Tag','grid_tag');
       h.forward_grid_tag       = findobj(h0,'Tag','forward_grid_tag');

       h.backwardPos_tag      = findobj(h0,'Tag','backwardPos_tag');
       h.backwardState_tag    = findobj(h0,'Tag','backwardState_tag');
       h.backwardProb_tag     = findobj(h0,'Tag','backwardProb_tag');
       h.backwardGrid_tag     = findobj(h0,'Tag','backwardGrid_tag');


       set(h0,'UserData',h);

    case 'backward_callback'

       [o,FF]=gcbo;
       h=get(FF,'UserData');

       if(~h.isHMMLoaded)
          uiwait(errordlg(sprintf('Error. No HMM model loaded. Please load an HMM model first'))
          return;
       end

       observation= get(h.seq_tag,'String');
       observation=cleanSeq(observation);
       if(length(observation)==0)
          uiwait(errordlg(sprintf('Error. No observation sequence available. Please type in a se
          return;
       end

       theTime = get(h.backwardPos_tag,'String');
       theTime   = str2double(theTime);
       if( isnan(theTime))
```

```
        uiwait(errordlg(sprintf('Error in backward algorithm input. Invalid numeric value fo
        return;
    end

    if(theTime<1 | theTime>length(observation))
        if(theTime<1)
            uiwait(errordlg(sprintf('Error in backward algorithm input. can not have a negati
        else
            uiwait(errordlg(sprintf('Error in backward algorithm input. position given exceed
        end
        return;
    end

    inputState = nma_trim(get(h.backwardState_tag,'String'));
    if(isempty(inputState))
        uiwait(errordlg(sprintf('Error in backward algorithm input. Empty state name.')));
        return;
    end

    found=0;
    for(i=1:length(h.states))
        s=char(h.states(i));
        if(length(s)==length(inputState))
            if(s==inputState)
                found=1;
                break;
            end
        end
    end

    if(~found)
        uiwait(errordlg(sprintf('Error in backward algorithm input. Invalid state name. Unre
        return;
    end

    statePos=getPosOfState(h.states,inputState);
    prob=exp(h.beta(statePos,theTime))*exp(h.alpha(statePos,theTime))/fullProb(h);
    set(h.backwardProb_tag,'String',prob);

case 'run_callback'

    [o,FF]=gcbo;
    h=get(FF,'UserData');

    if(~h.isHMMLoaded)
        uiwait(errordlg(sprintf('Error. No HMM model loaded. Please load an HMM model first'
        return;
    end

    observation= get(h.seq_tag,'String');
    observation=cleanSeq(observation);
    if(length(observation)==0)
       uiwait(errordlg(sprintf('Error. No observation sequence available. Please type in a se
       return;
    end

    set(h.seq_tag,'String',observation);

    h.observation=observation;

    [h,err]=forward(h);
    if(~isempty(err))
       uiwait(errordlg(sprintf('%s',err)));
       return;
    end

    h=viterbi(h);

    [beta,err]=nma_hmmBackward(observation,h.symbols,h.Log_Init_Prob,h.Log_Emit_Prob,h.Log_Tr
    if(~isempty(err))
       uiwait(errordlg(sprintf('%s',err)));
       return;
    end
    h.beta=beta;

    Formatted_String = Format_Probability_Grid( h.beta, h.states , []);
    Position_String  = Format_Position_String( h.alpha );
    Final_String     = Append_Cell_Array( Formatted_String, Position_String);

    set( h.backwardGrid_tag, 'String', Final_String);

    set(FF,'UserData',h);
    return;


 case 'openHMMFile_callback'

    [o,FF]=gcbo;
    h=get(FF,'UserData');

    if(~isempty(h.hmmLastFolder))
       cd( h.hmmLastFolder);
    end
```

```
[fileName, pathName] = uigetfile(['*.hmm;*.txt'], 'Select HMM configuration file to ope
if(pathName==0)
   return;
end

fullFileName=[pathName fileName];

%          [fid,err] = fopen(fullFileName,'rt');
%          if(~isempty(err))
%              uiwait(errordlg(sprintf('Error opening HMM file[%s]. error=%s.',fullFileN
%              h.hmmFID = 0;
%              set(FF,'UserData',h);
%              return;
%          end
%
%          fclose(fid);
%
h.hmmLastFolder=pathName;
h.fullHMMFileName=fullFileName;

set(h.hmmFileName_tag,'String',fullFileName);

[symbols, initProb, states, tranProb, emitProb, err]=nma_readHMM(h.fullHMMFileName);
if(~isempty(err))
   uiwait(errordlg(sprintf('Error parsing HMM configuration file[%s]. error=%s.',fileNa
   return;
end

DELTA=0.00001;
LOW_THRESHOLD=1-DELTA;
HIGH_THRESHOLD=1+DELTA;

T=sum(initProb);
if(T<LOW_THRESHOLD | T>HIGH_THRESHOLD)
   uiwait(errordlg(sprintf('Initial probability sum is found to be %f. It must equal 1.
   return;
end

Number_Of_States = size(tranProb,1);
for(row=1:Number_Of_States)
   T=sum(tranProb(row,:));
   if(T<LOW_THRESHOLD | T>HIGH_THRESHOLD)
      uiwait(errordlg(sprintf('Probability transition matrix, row %d,  the sum must equ
      return;
   end
end
```

```
for(row=1:Number_Of_States)
   T=sum(emitProb(row,:));
   if(T<LOW_THRESHOLD | T>HIGH_THRESHOLD)
      uiwait(errordlg(sprintf('emit probability matrix, row %d,  the sum must equal 1.0,
      return;
   end
end

h.isHMMLoaded=1;

h.symbols=symbols;
h.initProb=initProb;
h.emitProb=emitProb;
h.states=states;
h.tranProb=tranProb;

% convert prob to log now for speed, so we do not have
% to do this every time.
Number_Of_States  = length(states);
Number_Of_Symbols = length(symbols);

for(i=1:Number_Of_States)
   if(initProb(i) == 0 )
      h.Log_Init_Prob(i) = -inf;
   else
      h.Log_Init_Prob(i) = log(initProb(i));
   end

   for(j=1:Number_Of_Symbols)
      if(emitProb(i,j)==0)
         h.Log_Emit_Prob(i,j) = -inf;
      else
         h.Log_Emit_Prob(i,j) = log(emitProb(i,j));
      end
   end

   for(j=1:Number_Of_States)
      if(tranProb(i,j)==0)
         h.Log_Tran_Prob(i,j) = -inf;
      else
         h.Log_Tran_Prob(i,j) = log(tranProb(i,j));
      end
   end
end

T='';
symbolsAsString=char(symbols);
```

```
    for(i=1:length(symbols))
        if(i==length(symbols))
            T=[T sprintf('%c',symbolsAsString(i))];
        else
            T=[T sprintf('%c,',symbolsAsString(i))];
        end
    end

    set(h.modelSeq_tag,'String',{T});

    set(h.modelStates_tag,'String',states);

    theText={''};
    nLine=0;
    T='';
    statesAsString=char(states);
    for(j=1:size(tranProb,2))
        T=[T sprintf('%10c  %s',' ',statesAsString(j,:))];
    end
    nLine=nLine+1;
    theText(nLine)={T};

    for(i=1:size(tranProb,1))
        T=sprintf('%s  ',statesAsString(i,:));
        for(j=1:size(tranProb,2))
            T=[T sprintf('%6.6f  ',tranProb(i,j))];
        end
        nLine=nLine+1;
        theText(nLine)={T};
    end

    set(h.modelStateTranMatrix_tag,'String',theText);

    theText={''};
    nLine=0;
    T='';
    for(i=1:size(emitProb,1))
        T='';
        for(j=1:size(emitProb,2))
            T=[T '     ' sprintf('%6.6f',emitProb(i,j))];
        end
        nLine=nLine+1;
        theText(nLine)={T};
    end

    set(h.modelEmitProbMatrix_tag,'String',theText);
    set(h.modelInitialProb_tag,'String',initProb);
```

```
      set(FF,'UserData',h);
      return;


   case 'openLogFile_callback'

      [o,FF]=gcbo;
      h=get(FF,'UserData');

      if(~isempty(h.hmmLastFolder))
         cd( h.hmmLastFolder);
      end

      [fileName, pathName] = uigetfile(['*.txt'], 'Select log file to open');
      if(pathName==0)
         return;
      end

      fullFileName=[pathName fileName];

      [fid,err] = fopen(fullFileName,'at');
      if(~isempty(err))
         uiwait(errordlg(sprintf('Error opening log file[%s]. error=%s.',fullFileName,err)));
         h.hmmFID = 0;
         set(FF,'UserData',h);
         return;
      end

      fclose(fid);

      h.hmmLastFolder=pathName;
      h.fullLogFileName=fullFileName;

      set(h.logFileName_tag,'String',fullFileName);

      set(FF,'UserData',h);
      return;
end

%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%
function sum=fullProb(h)
sum=0;
for(i=1:size(h.alpha,1))
```

```
   sum=sum+exp(h.alpha(i,end));
end

%%%%%%%%%%%%%%%%%%%%%%%%%
%
% removes spaces
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Output_Sequence=cleanSeq(seq)

Number_Of_Lines = size(seq,1);
Length_Of_Line  = size(seq,2);

Output_Sequence = zeros(Number_Of_Lines * Length_Of_Line , 1);

k=0;

for(i=1:Number_Of_Lines)
   for(j=1:Length_Of_Line)
      if(seq(i,j)==' ' |seq(i,j)=='\t')
         ;
      else
         k=k+1;
         Output_Sequence(k)=seq(i,j);
      end
   end
end

Output_Sequence = char(Output_Sequence(1:k))';
Output_Sequence = nma_trim(Output_Sequence);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%check if char in observation sequence is valid
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
function valid=validSymbol(c,allowed)

valid=0;

for(i=1:length(allowed))
   if(c == char(allowed(i)))
      valid=1;
      return;
   end
end


%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%
function [h,err]=forward(h)

err='';

for(i=1:length(h.observation))
   if(~validSymbol(h.observation(i),h.symbols))
      err=sprintf('Error. observation sequence contains an invalid character ''%c''',h.observa
      return;
   end
end

[alpha,err]=nma_hmmForward(h.observation,h.symbols,h.Log_Init_Prob,h.Log_Emit_Prob,h.Log_Tran_
if(~isempty(err))
   err=sprintf('Error. Failed in forward algorithm. err=%s',err);
   return;
end

h.alpha=alpha;
prob=0;
for(state=1:size(alpha,1))
   prob=prob+exp(alpha(state,end));
end

set(h.forward_tag,'String',sprintf('%g',prob));

Formatted_String = Format_Probability_Grid( h.alpha, h.states, [] );
Sum_Of_Probability_String = Format_Sum_Of_Probability_String(h.alpha);
Position_String  = Format_Position_String(h.alpha);

Final_String = Append_Cell_Array( Formatted_String , Sum_Of_Probability_String);
Final_String = Append_Cell_Array( Final_String , Position_String);

set(h.forward_grid_tag,'String',Final_String);

%%%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%
function h=viterbi(h)

[statePath,stateSeq,gamma]=nma_hmm_veterbi(h.observation,h.symbols,h.Log_Init_Prob,h.Log_Emit_
h.gamma=gamma;

final={''};
```

```
k=0;
for(i=1:length(stateSeq))
   if(i==1)
      T=char(h.states(stateSeq(i)));
   else
      T=[T ',' char(h.states(stateSeq(i)))];
   end
   if(0==mod(i,20))
      k=k+1;
      final(k)={T};
      T='';
   end
end
k=k+1;
final(k)={T};

set(h.veterbi_tag,'String',final);

Formatted_String = Format_Probability_Grid(h.gamma, h.states, stateSeq);
Sum_Of_Probability_String = Format_Sum_Of_Probability_String(h.alpha);
Position_String  = Format_Position_String(h.alpha);

Final_String = Append_Cell_Array( Formatted_String , Sum_Of_Probability_String);
Final_String = Append_Cell_Array( Final_String , Position_String);

set(h.grid_tag,'String',Final_String);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Formatted_String = Format_Probability_Grid(Data_Matrix, HMM_States,Viterbi_State_Seq

n=0;
Formatted_String={''};
T='Probabilites in logs';
n=n+1;
Formatted_String(n)={T};

for(state=1:size(Data_Matrix,1))
   name=char(HMM_States(state));
   L=length(name);
   T='';

   for(m=1:10)
      if(m>L)
         c=' ';
      else
         c=name(m);
      end
      T=[T c];
   end

   T=[T ':'];

   for(time=1:size(Data_Matrix,2))
      c=' ';
      if( (~isempty(Viterbi_State_Seq)) &  (Viterbi_State_Seq(time)==state) )
         c='*';
      end

      if(Data_Matrix(state,time)==-Inf)
         Z=sprintf('    -Inf     ');
      else
         Z=sprintf('%10.10f',Data_Matrix(state,time));
      end

      if(time==size(Data_Matrix,2))
         T=sprintf('%s%s%c',T,Z,c);
      else
         T=sprintf('%s%s%c    ',T,Z,c);
      end
   end

   n=n+1;
   Formatted_String(n)={T};
end

T=' ';
n=n+1;
Formatted_String(n)={T};

T='Probabilites';
n=n+1;
Formatted_String(n)={T};

for(state=1:size(Data_Matrix,1))
   name=char(HMM_States(state));
   L=length(name);
   T='';
   for(m=1:10)
      if(m>L)
         c=' ';
      else
```

```
         c=name(m);
      end
      T=[T c];
   end
   T=[T ':'];
   for(time=1:size(Data_Matrix,2))
      if(Data_Matrix(state,time)==-Inf)
         Z=sprintf('    0        ');
      else
         Z=sprintf('+%3.10f',exp(Data_Matrix(state,time)));
      end

      if(time==size(Data_Matrix,2))
         T=sprintf('%s%s%c',T,Z,' ');
      else
         T=sprintf('%s%s%c    ',T,Z,' ');
      end
   end
   n=n+1;
   Formatted_String(n)={T};
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Sum_Of_Probability_String =  Format_Sum_Of_Probability_String(Data_Matrix)

Line_Number     = 0;

Line=' ';
Line_Number  = Line_Number + 1;
Sum_Of_Probability_String(Line_Number) = {Line};


Line='Column Sums of Probabilites';
Line_Number  = Line_Number + 1;
Sum_Of_Probability_String(Line_Number) = {Line};


%Now also display the probability of partial sequences.
Line='prob.     :';
for(time=1:size(Data_Matrix,2))
   prob=0;
   for(state=1:size(Data_Matrix,1))
```

```
      prob=prob+exp(Data_Matrix(state,time));
   end

   Line=sprintf('%s+%10.10f     ',Line,prob);
end

Line_Number  = Line_Number + 1;
Sum_Of_Probability_String(Line_Number)={Line};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function Position_String = Format_Position_String(Data_Matrix)

Line_Number = 0;

Line=' ';
Line_Number  = Line_Number + 1;
Position_String(Line_Number) = {Line};

Line='Position  :';

for(time=1:size(Data_Matrix,2))
   Z=sprintf(' %12d',time);
   Line=sprintf('%s%s%c    ',Line,Z,' ');
end

Line_Number  = Line_Number + 1;
Position_String(Line_Number) = {Line};



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function   S=Append_Cell_Array(S1,S2)

S=S1;
N=length(S);
for(i=1:length(S2))
   S(N+i)=S2(i);
end
```

6.2.11 getPosOfState.m^^E^^L

```
function pos=getPosOfState(stateArray,stateName)
%function pos=getPosOfState(stateArray,stateName)
% returns the position of state name from state Array
%
% Nasser Abbasi
pos=-1;

stateName=nma_trim(stateName);
N=length(stateName);
for(i=1:length(stateArray))
     s=char(stateArray(i));
     M=length(s);
     if(N==M)
        if(s==stateName)
           pos=i;
           return;
        end
     end
end
```

BIBLIOGRAPHY

[1] R.Durbin,S.Eddy,A.Krogh,G.Mitchison. Biological Sequence Analysis.

[2] Dr Lior Pachter lecture notes. Math 127. Fall 2002. UC Berkeley.

[3] Identification of genes in human genomic DNA. By Christopher Burge, Stanford University, 1997.