Markov Chains and Their Application in Biological Sequence Analysis

By Eric W. Wasiolek

This paper will introduce the probability theory of Markov Chains and then discuss their myriad applications in biological sequence analysis (the analysis of proteomic and genomic sequences). There are many forms of Markov Chains, and the most relevant to biological sequence analysis are Hidden Markov Chains --- these will be discussed in detail. Markov chains have applications in a wide variety of fields including speech recognition and many engineering applications, but we will particularly be concerned with the use of Markov chains to do analysis of biological data in this paper.

Introduction

We can first understand Markov chains as a transition probability matrix where given a sequence, the probability of generating that sequence is given by multiplying successive values in the transition probability matrix.

Consider first an n x n matrix that consists of four values, the four nucleotides: A,C,G, and T, where n equals the number of nucleotides.

	A	C	G	T
A	0.18	0.274	0.426	0.12
C	0.171	0.368	0.274	0.188
G	0.161	0.339	0.375	0.125
T	0.079	0.355	0.384	0.182
	0.591	1.336	1.459	0.615

Table 1: Transition Probability Matrix for Four Nucleotides

The cells of the matrix indicate the probability of state s(i) transitioning to state s(i+1). Consider a sequence of length L, where L equals 5: "CGATA." The probability of a C being followed by a G is .274, which is different from the probability of a C being followed by an A, which is .171. The probabilities of the matrix are calculated by training sessions on actual DNA sequences and represent the proportion of times that letter or state s(i+1) follows s(i). Since there are only 4 states in this analysis, any given state s(i) can only transition to itself or three other states. The sum of the probabilities of each state that can be transitioned to, or the sum of the row, is equal to 1, of course. Combinatorics would state that each state would have a 1/n chance of being followed by another state, but the point here is that the transition values are not even but determined by analysis of actual DNA sequences, where, for example, it is more probable that a C

will be followed by a G than by an A. Underlying this point is the important point that the occurrence and sequences of nucleotides in DNA sequences is not strictly random.

To calculate the probability of a specific sequence occurring, the probabilities of each transition are simply multiplied. For example, the probability of the sequence "CGATA" occurring is given by 1 x .274 x .161 x .12 x .079 = .00041820 (call this p(s)). The probability is small and this is to be expected. In fact, under a combinatorial analysis, given equal probabilities, the probability of any particular sequence is given by 1/(n^L), where n is the number of letters or possible states and L is the length of the sequence. It is easy to see that as the sequence grows longer, the probability of a particular sequence becomes extremely small, so small that for computers to calculate a long sequence log(1/(n^L)) need be used, just as log of the product of the transition probabilities may need to be used in the Markov chain case. Note that the initial symbol in the sequence is given the probability of 1/n as there is a 1/n chance of any particular letter or symbol being the starting symbol. The initial symbol is also called the begin state of the Markov chain.

It can be shown that a Markov chains describe a proper probability distribution over the space of sequences because the sum of the probabilities (p(s)) of all possible sequences of length L, i.e., n^L possible sequences is equal to 1.

A Markov chain is so called because the probability of any state in the sequence depends only on the previous state, s(i) depends upon only s(i-1), and nothing else. Markov chains can also be represented by a state transition diagram, in the form of a graph, where the nodes represent states (e.g. C or G or A or T) and the edges represent transition probabilities. A sequence can then be described by the **path** in the state transition diagram.

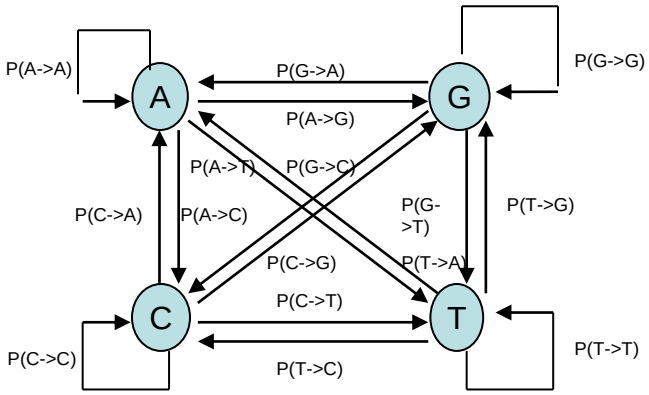


Figure 1: State Transition Diagram of Markov Chain that Generates Nucleotide Sequence

A particular sequence would then be described by the path or sequence of transitions from one state to the next, with corresponding transition probability, with L-1 transitions for a sequence of length L, and starting (transition) probability 1/n.

Part I – Markov Chains and CG Islands for Gene Identification

Now, before discussing Hidden Markov models, let's consider an application of Markov chains to help identify genes or exon coding regions within a DNA sequence. Such an application we will discuss in depth in this paper, as one of the greatest challenges facing modern biology is to identify genes for the myriad of organisms whose DNA has been sequenced but genes have not been identified. We will initially use the approach of using Markov chains to identify CG islands, i.e., CG subsequences of the DNA sequence which are indicators of promoter regions or start transcription regions of many genes. CG islands are typically a few hundred to a few thousand bases long. Some post-processing steps may be needed, as the Markov chain model will typically predict short CG subsequences where rules like concatenating predictions less than 500 bases apart and eliminate predictions shorter than 500 bases can be used to finish the job.

Suppose we did some more sampling from DNA sequences and generated a larger matrix that had the transition probabilities (proportions) in CG islands, which will will designate (C+, G+, A+, and T+), the transition probabilities in non-CG islands (which we will designate C-, G-, A-, and T-), and the lower probabilities of the transitions between CG island and non-CG-island bases. The matrix may look something like this:

	A+	C+	G+	T+	A-	C-	G-	T-	
A+	0.145	0.239	0.391	0.085	0.029	0.0478	0.0782	0.017	1.0
C+	0.136	0.333	0.239	0.153	0.0272	0.0666	0.0478	0.0306	1.0
G+	0.126	0.304	0.34	0.09	0.0252	0.0608	0.068	0.018	1.0
T+	0.044	0.32	0.349	0.147	0.0088	0.064	0.0698	0.0294	1.0
A-	0.02875	0.01925	0.02725	0.01975	0.2875	0.1925	0.2725	0.1975	1.0
C-	0.03095	0.02855	0.00655	0.02895	0.3095	0.2855	0.0655	0.2895	1.0
G-	0.02355	0.02335	0.02855	0.01955	0.2355	0.2335	0.2855	0.1955	1.0
T-	0.01645	0.02265	0.02795	0.02795	0.1645	0.2265	0.2795	0.2795	1.0

Table 2: Transition Probability Matrix Between CG Islands and Non-Islands

A few observations are in order. Transition probabilities to C's and G's are higher than transition probabilities to A's and T's in CG island sequences, whereas this is not true in non-CG-island sequences (where the transition probabilities are for the most part similar to all nucleotides). This keeps island sequences composed mainly of C's and G's. Transition probabilities BETWEEN island and non-island sequences are small, but it is more probable to transition from an island to a non-island nucleotide than from a non-island to an island nucleotide. In this way, the model spends most of its time in either island or non-island sequences but not flipping between them. Island to non-island probabilities are calculated as one fifth island to island probabilities, and non-island to island probabilities are calculated as one tenth of non-island to non-island probabilities.

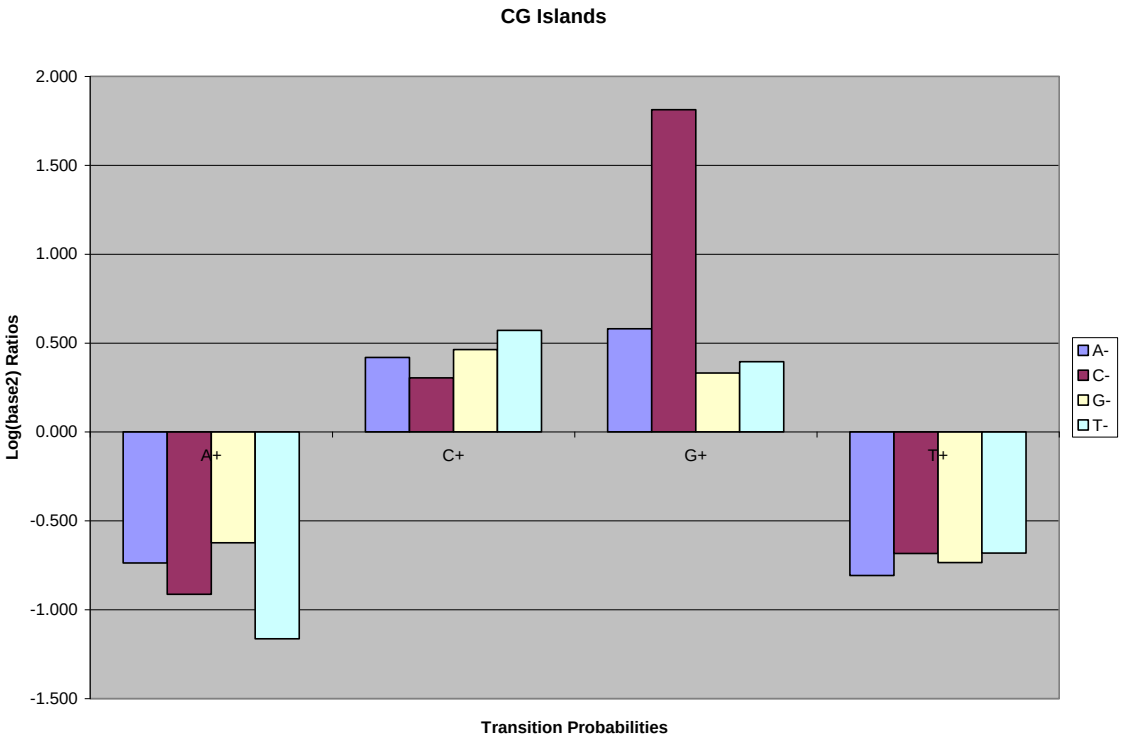
A note about mathematical notation here. We distinguish n, the number of possible symbols in our observed sequence, where n = 4 in our example (A,C,T, or G) from n(s) which is the number of STATES in our system where n(s) = 8 (namely: A+, C+, T+, G+, A-, C-, T-, or G-), where the number of state transitions and therefore number of state transition probabilities is n(s)^2 or 64. We will later further introduce the notation n(cs) in Hidden Markov Models to refer to a candidate state. I.e. if the symbol C is observed at position 3 (x(i) where i = 3) in the observed sequence, there are only 2 states that could emit C in our system, either C+ or C-, i.e., n(cs) = 2.

Before we refer to Table 2 to discuss how to identify the state sequence that accounts for a particular observed subsequence of DNA, consider how the probabilities of the islands of CGs stand out in this probability matrix from non-CG islands by looking at the logs ratio of these probabilities and their graph. The log ratio is calculated as log(base 2)(s+(i)->s+(i+1)/s-(i)->s-(i+1)), here where s+(i)->s+(i+1) refers to an island state transition probability and s-(i)->s-(i+1) refers to the corresponding non-island state transition probability. Hence the following nxn table:

Log Ratio	A+	C+	G+	T+
A-	-0.737	0.419	0.580	-0.807
C-	-0.913	0.304	1.813	-0.684
G-	-0.623	0.463	0.332	-0.735
T-	-1.164	0.571	0.395	-0.682

Table 3: Base Two Log Ratio of CG-Island Versus Non-Island Transition Probabilities

And the corresponding graph which indicates clear the C+G+ islands (C's and G's in CG islands as positive logs, with C's and G's in non-islands as negative logs).



Graph 1: Log ratio of CG island versus non-CG island transition probabilities

Hidden Markov Chains

Now we move to the important topic of Hidden Markov Chains or HMM's, critical to many aspects of biological sequence analysis. Briefly, in a HMM, we distinguish between the observed sequence of nucleotides (x(i) … x(L)) and the states that generate the observed sequence (s(i) to s(L)). In regular (non-hidden) Markov chains, there is a one to one correspondence between states and observed nucleotides. If C moves to state A, and A nucleotide is observed, if A then moves to a G, a G nucleotide is observed. I.e. as state S(i-1) moves to S(i), the observed nucleotide in the sequence at position i (x(i)) is simply determined by S(i). But in reality it is often not know what state sequence determines the observed nucleotide sequence. In the case where A's, C's, G's and T's may come from either CG island or non-island sequences, we do not know which exact state sequence (or path) accounts for the observed sequence. For example, the sequence CGCG may come either from state sequence (path) C+G+C+G+ or path C-G-C-G- or path C+G-C-G+. In fact there are n(cs)^L possible state sequences or paths that could account for the observed sequence CGCG, where n(cs) is the number of possible values for each position, e.g. a C must be either a C+ or C- state, and L is the length of the sequence. In this case, 16 (2^4) possible state sequences (paths) could account for the observed sequence. As the observed sequence length L grows, the number of possible state sequences that could account for it grow exponentially.

However, associated with each possible state sequence is a sequence probability, calculated by multiplying the transition probabilities for the succession of states for the particular state sequence. For example if C+G+C+G+ were the state sequence generated the observed sequence, the probability would be: P(begin->C+) is .125, P(s(C+->G+)) is .239 x P(s(G+->C+)) is .304 x P(s(C+->G+) is .239. Since we don't know the state before the initial C+, we don't know its transition probability, so by convention the initial transition probability P(s(begin->C+)) is simply 1/n(s) where n(s) is the number of possible states in the system, which is 8. Therefore, the probabilities of the following states are calculated as:

1. C+G+C+G+ is .125 x .239 x .304 x .239 which equals .00298
2. C-G-C-G- is .125 x .0655 x .2335 x .0655 which equals .000125
3. C+G-C+G- is .125 x .0478 x .02335 x .0478 which equals .000006669

These state sequence or path probabilities fit with our intuition and our model that a CGCG sequence in a CG island region is more probable than one in a non-island region, and that a CG sequence which flips between island and non-island regions is extremely improbable. By this same model, note that observed sequences involving A's and T's would be more likely to be in a non-CG island region.

Now that we have observed that different paths have different probabilities, it is simple to understand the notion of the most probable or optimal path (we will designate as p*). We here introduce an algorithm that determines the most probable path p*, not by the brute force method of comparing all of the probabilities of the n(cs)^L possible paths, which becomes exponentially intractable as L grows, but by an algorithm called the Viterbi algorithm.

The Viterbi Algorithm

The Viterbi algorithm simply says find the maximum probability of the candidate states that can account for a given observation x(i), do so for each observation, and multiply the maximum probabilities of these states together to generate the optimal path probability. At the same time, note the state that corresponds to each of these maximum probabilities, and backtrack to determine the optimal state sequence or optimal path.

Written as a pseudo-code algorithm:

1. Initialization. Let p(s(0)), the probability of the begin state = 1/n(s). Let p* = 1/n(cs).
2. For i through L (where L is the length of the observed sequence and the number of states in the path)
3. Let p(i) = max prob (s(n)(i)) where n is the number of candidate states for the observed symbol in the sequence x(i) and s(i) is the ith state.
4. Let path[i] = s(i) where s(i) is the candidate state with the highest probability of emitting observed symbol x(i)
5. Let p* = p* x p(i)

6. Increment i.
7. The optimal path probability is then p*. p* may need to be expressed as log(p*) to avoid underflow errors for a long sequence.
8. The optimal path is then: for j = 0; j <= L; ++j; output path[j]

The Viterbi algorithm is 91% accurate in identifying the state sequence for islands versus the state sequence for non-island regions. In the case of CGCG, the Viterbi algorithm would find the state sequence C+G+C+G+ with probability .00298.

The Forward Algorithm for Calculating the Probability of the Observed Sequence

Additionally, we may use something called the forward algorithm to calculate the probability of the observed sequence occurring at all, i.e., by any state sequence. Simply, the observed sequence x(i) where i = 0 to L is given by the forward algorithm. Again, we don't want to use the brute force method of determining the probabilities of every possible path that can yield the observed sequence and then summing these probabilities, as this would require the evaluation of n(cs)^L paths, an intractable number of paths where L is large. Instead we use the following algorithm to compute the probability of observed sequence x(i) where i = 0 to L:

1. Replace the maximization step 3 of the Viterbi algorithm with a summation of the transition probabilities of n(cs)^2 state transitions (here n(cs) = 2, C- or C+, or G- or G+) that are candidates to emit observed value x(i).

In this case the probability of observed sequence x occurring is given by (according to Table 2):

.25 x .35885 x .62165 x .35885 = .02 using combinatorics to calculate the initial value

Where the first symbol C is produced by x(0)->x(i) transition probability. Since x(0) or the state preceeding the state that emits the first symbol in the observed sequence is unknown, since the second state in the transition has to be either a C+ or a C- to emit a C, and since there are 8 possible transitions that will result in the second state s(i+1) from s(i) being a C+, each with probability .125 and 8 possible transition that will result in the second state s(i+1) from s(i) being a C-, each with probability .125, the probability of a C occurring in state s(i) in sequence x is .125 x 2 or .25, this is the same as 1/n, where n = 4. Alternatively, we can use the transition probability matrix to use the average of any state going to C+ and the average of any state going to C-, which would make the initial probability of emitting a C about .3.

For the second observed symbol in the sequence, G, there are only 4 states that could produce it, since the proceeding state is either a C+ or C-: (C+->G+; C+->G-; C- -> G+; or C- -> G-). The transition probabilities of each of these transitions is summed and the result is .35885. This is also the case for the fourth observed symbol. For the third observed symbol, the only possible state transitions that could have produced it are: (G+->C+, G+->C-; G- ->C+, or G- -> C-) the sum of the transition probabilities of which is

.62165. Multiplying the four sums of possible transition probabilities that could produce the observed sequence we get .02.

Note about computability:

Note that in general only (n(cs)^2 x (L-1)) + 1 number of transition probabilities need be calculated (summed and then multiplied). For a sequence of 4 where n(cs) is 2, (2^2 x 3) +1 is 13. For a sequence 100 long, we can see the value of this approach. Calculating all of the probabilities of state paths of a observed sequence 100 symbols long would involve 397 transition probabilities.

L-1 sums of n(cs)^2 transition probabilities and one product of L terms, in other words, n(cs)^2(L-1) + L calculations. I.e., 3 sums of 4 transition probabilities and one product of 4 terms, i.e, 16 calculations, where we are counting as a calculation each += and *= iteration.

For a sequence of 100 symbols long, the forward algorithm is computable whereas summing the probabilities of all possible paths for the observed sequence is not practically computable. Where L = 100, the forward algorithm would involve 2^2(99)+100 calculations or 496 calculations; whereas summing the probabilities of all possible paths of the observed sequence would involve one sum of n(cs)^L terms and n(cs)^L products of L terms each. This would involve n(cs)^L + (n(cs)^L x L) calculations or 2^100 + (2^100 x 100) calculations, which is about 1.28 x 10^32 calculations, practically infeasible, especially if you are calculating an observed sequence of 100,000 symbols! So the forward algorithm HAS to be used to calculate the probability of an observed sequence.

A Digression on Combinatorics

It is useful to compare the results of various calculations using the transition probability matrix to what we would expect to find through combinatorial analysis. This way we have some idea of whether the probabilities for paths and for observed sequences that we calculate are approximately correct.

Consider that for a sequence of length L, there are n^L possible sequences given n symbols that can occupy any position x(i) in the sequence. So, for a combination lock where there are three numbers, L = 3, and using the decimal system, there are ten symbols that can occupy any position in the sequence, 0 through 9, hence there are n^L or 10^3 or 1000 possible combinations. The same of course holds true in our analysis. There are 4 possible symbols (A,C,G,T) and a sequence 4 symbols long, then there are 4^4 or 256 possible sequences that can be generated. Given a random chance and equal transition probabilities there are 1/(n^L) or 1/256 or .0039 chance or probability of any particular sequence occurring. Another way of calculating this is, if there is an equal probability or any symbol transitioning to any other symbol then the transition probability of symbol x -> symbol y (x(i)-> x(i+1)) is 1/n or ¼ or .25. In a sequence of 4 symbols

long, the probability of that sequence occurring would be .25^4 or .0039 which is also 1/256. In general for matrices of equal transition probabilities in all cells, which is what a combinatorial analysis assumes, the probability of a particular sequence is (1/n)^L which is also 1/(n^L). In this case there is a one to one correspondence between state sequences and observed sequences, in a non-HMM Markov model, so the probability of a particular sequence is 1/256 or .0039 and this is also the probability of the unique state sequence that emits the sequence.

In an HMM the probability situation is different. For example, in an HMM, the probability of the occurrence of an observed sequence is NOT the same as the probability of the particular state sequence that emitted it. Consider in our case a sequence of length L, where L = 4. But consider n = 8 symbols (A+,C+,G+,T+, A-,C-,G-, and T-). Now there are n^L or 8^4 or 4096 possible state paths but still only 4^4 or 256 possible unique observable sequences. This is because there are 16 possible state paths or 4096/256 that emit each unique observed sequence. Any particular state path therefore occurs with the probability of 1/(n^L) or 1/4096 or with p = 2.44 x 10^-4., which is also (1/8)^4. The observed sequence x however occurs with probability 16 x (2.44 x 10-4) or .0039, which equals 1/256. We therefore expect:

1. That in our actual transition probability matrix, where transition probabilities are not equal, the forward probability (fp) of the observed sequence should be in the vicinity of .0039
2. And p*, the optimal probability of the most probable path to emit the sequence should be: 2.44 x 10-4 < p < .0039 or .000244 < p < .0039. But in practice, p* is an approximation of the forward probability and can be as much as .8 fp as many of the other paths that can emit the sequence are highly improbable.

One note about the 16 possible state paths that can account for an observed sequence of 4. Since under this analysis, only 2 of the eight symbols can emit each of the symbols in the observed sequence, e.g. for the sequence CGCG, only C+ or C- can account for C in the first position, only G+ or G- can account for G in the second position, etc… so that there are only 2 candidate states that can account for any given symbol in any position. Let us designate the candidate states a n(cs). Then for a sequence of length L, n(cs)^L or 2^4 gives us the number of possible state sequences that can generate the observed sequence. If the state sequence is visualized as a tree, then when n(cs) = 2, it is a binary tree, and in general n(cs) indicates the number of edges exiting from each node in the tree graph, where n(cs) needn't be binary.

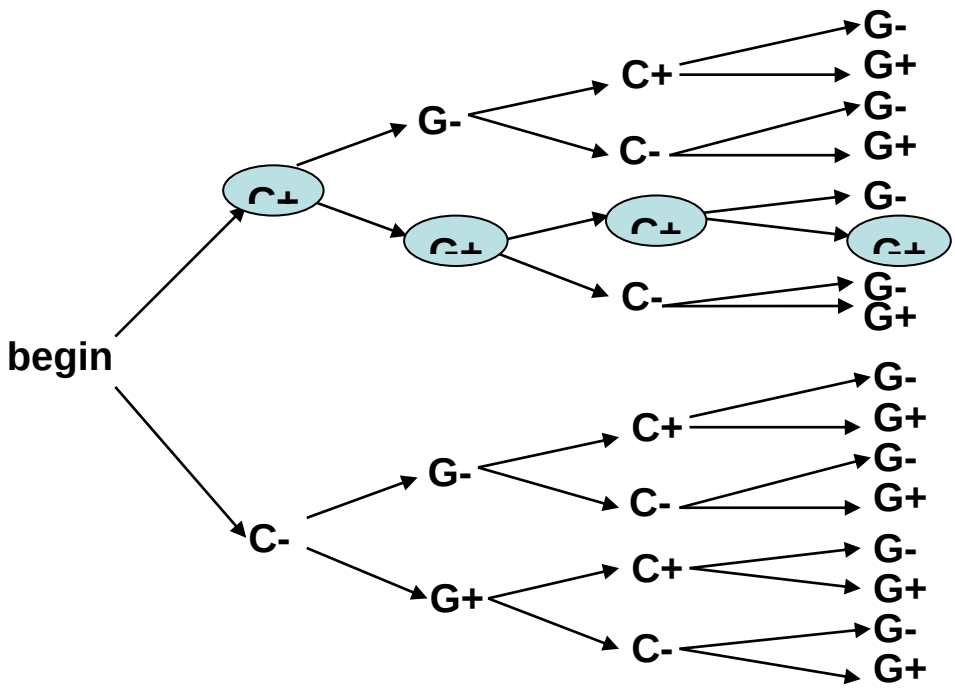


Figure 2: Binary Tree Representation of all possible paths emitting the Sequence CGCG, given S(cs) = 2.

Now let us compare our results to those we would expect were all transition probabilities equal as in a combinatorial analysis.

p* is calculated to be .00298. This is consistent with our combinatorial analysis were:

$.000244 < p < .0039$.

The forward probability for the observed sequence should be around .0039 with some variance, i.e., not all observed sequences are equally probable given a transition probability matrix where there is great variance between the transition probabilities. In this case the forward probability is .02, quite a bit larger than .0039, meaning that the sequence CGCG is much more probable in our model than any random sequence, which makes sense, as the model is used to find CGCG islands. Note that other sequences, such as TATA which has a probability of .021 of being observed is less than .0039. .0039 should just be the mean fp with a great deal of variance.

Also note that p*/fp is 14%, which is reasonable, as in our combinatorial analysis we expect $p^*/fp > 1/16^{th}$ or 6.25%. In other words, the optimal path is more than twice as probable as the average path that is a candidate to emit the observed sequence.

Hidden Markov "Models"

Thus far in this paper we have assumed a particular model to distinguish and identify CG-islands from non-CG islands in a DNA sequence. But, this model is by no means the only type of Hidden Markov Model. In fact there are a great variety of HMMs which are distinguished by three factors:

1. The interconnection model
2. The emission model
3. The selection of transition probabilities.

The model used thus far in this paper has the following characteristics:

1. It is a fully-interconnected model
2. The emission model is uniforum with exactly 2 states emitting one symbol
3. The transition probabilities are non-uniform (i.e., every state transition's probability is different).

Consider the following Hidden Markov Model:

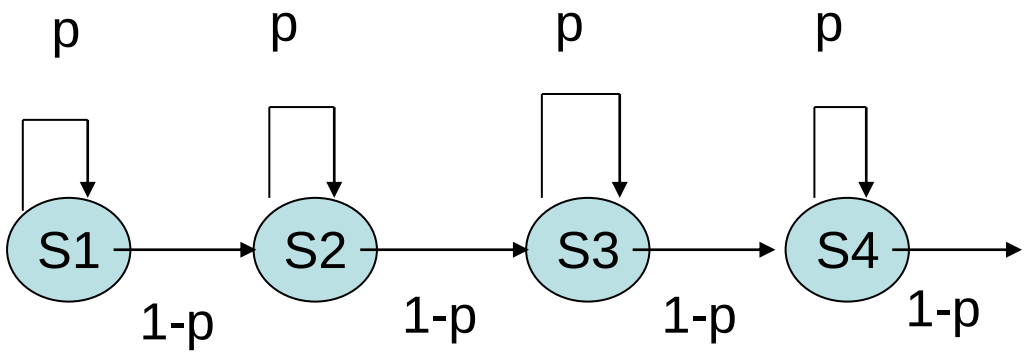


Figure 3: A non-fully interconnected HMM with uniform probabilities.

In this model, not all states connect to all other states, for example, s1 cannot transition to s4 directly, s2 cannot transition to s1, etc…, i.e., the model is not fully connected, as our original model. Secondly, the transition probabilities in this model are uniform, i.e., there are not different transition probabilities for each state transition, instead every state transition s(i)->s(i+1) has probability 1-p, and every state transition s(i)->s(i), i.e. every state transition to itself has probability p. This model has the additional condition that any path must traverse the states s1 through s4 and exit from s4, meaning that no path shorter than 4 is allowed. Because of the uniformity of transition probabilities, the

probability of any path can be given by a formula (which was not true of our original model, which required the separate arithmetic calculation of the probability of each path. The path probability formula for this HMM is:

P^(L-n(s)) * (1-p)^n(s), where L is the length of the path, n(s) is the number of states in the HMM, which is here 4, and p is a specific probability, say, .33. Then for each path of length L, say L is 10, the probability would be .33^6 * .67^4 = .00026. Furthermore, we can say in this case that there are L!/n!(L-n)! possible paths of length L. For L = 10 there would be 210 paths, and the total probability summed over all possible paths would be: (P^(L-n(s))) ((1-p)^n(s)) (L!/n!(L-n)!).

Note: sum of all probabilities is not 1, why?

Now, all Hidden Markov Models can be described by an n(s) x n(s) matrix, where the transition probability equals zero for every cell where there is no connect or transition from s(i) to s(i+1). Consider the following matrix to describe the Hidden Markov Model above:

	s1	**s2**	**s3**	**s4**	**end**
s1	p	1-p	0	0	0
s2	0	p	1-p	0	0
s3	0	0	p	1-p	0
s4	0	0	0	p	1-p

Table 4: Transition probability matrix of Hidden Markov Model in Figure 3.

So far we have discussed the state transition connection model and transition probability model for this HMM, but we have not discussed the emission model. Consider that in any HMM there is a many-to-one mapping between states and possible symbols in the observed sequence. In our original model there was a 2:1 mapping between + and – states and the observed sequence symbols (A,C,T,G). Namely, 2 states, C+ and C- both emitted symbol C, 2 states T+ and T- both emitted symbol T, etc… Because of this 2:1 mapping, we were able to determine the number of possible paths that could emit an observed sequence of length L. For example, there were 2^4 possible paths that could emit a sequence of length 4, or s(cs)^L possible paths.

In other models, things may not be so neat. A state transition model along with its emission mapping determines the possible observable sequences of length L that can be emitted.

	A	B	C
s1	1	0	0
s2	1	0	0
s3	0	1	0
s4	0	0	1

Table 5: Emission mapping between states and letters or symbols emitted for Figure 3.

Consider Table 5 which shows the mapping between states of Figure 3 and the letters or symbols they emit. Here two states, s1 and s2 emit A, s3 emits only B and s4 emits only C. Because there are more states than letters emitted this is a Hidden Markov model, though in some ways an unlikely one (most emission models will be a simple many-to-one relationship between states and emitted symbols, where n(cs) > 1 and n(cs) is an integer). The point here is that the state model together with the emission mapping determine which and how many observed sequences are possible. Consider an observed sequence of length 4. In the Figure 3 model, there is only ONE set of state transitions possible and therefore only one observable sequence possible: the state sequence is S1->s2->s3->s4, the emitted sequence is AABC, and the probability is (1-p)^n(s). Now consider an observed sequence of length 5.

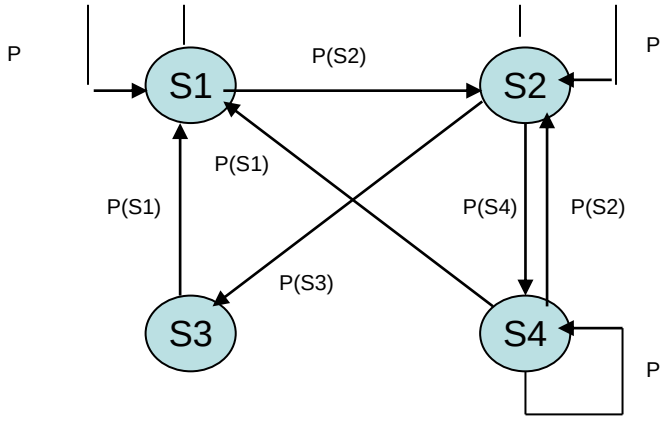


Figure 4: A possible Markov Chain State Transition Diagram with Semi-Uniform Transition Probability Assignments.

In Figure 4 we present an unorthodox Markov model, if only to show that such a model is possible, and that in fact there are as many models for n(s) states as there are subsets of an n(s) x n(s) connection matrix. I leave it as an exercise of combinatorics to calculate exactly how many connection models are possible given n(s) states.

Consider below the connection model and transition probability assignment of the state transition diagram presented in Figure 4:

	s1	**s2**	**s3**	**s4**	**# branches**
s1	p	p(s2)	0	0	2
s2	0	p	p(s3)	p(s4)	3
s3	p(s1)	0	0	0	1
s4	p(s1)	p(s2)	0	p	2

Table 6: Connection Model Matrix and Probability Assignments of the Unorthodox State Transition Diagram Presented in Figure 4.

In this unorthodox but possible Markov chain state transition model, there are a number of things to note. First, this model, as in any possible model, can be described by an n(s) x n(s) matrix. Note that the model is NOT binary, i.e., each state does not possibly transition to exactly two other states, as in our CG island model. The number of states to which any particular state may transition varies, as indicated by the # branches parameter. Here the probability assignments are not all unique (non-uniform) but not simple and consistent as in the Markov model in Figure 3 (uniform). Here, those states that transition to themselves all transition to themselves with probability p (note S3 does NOT transition to itself). Secondly, any state that transitions to a second state, transitions with a probability consistent with that second state. I.e., all states that transition to S1, other than S1, transition with probability p(s1), and all states that transition to S2, other than S2, transition with probability p(s2), and so on.

Now that we've discussed the connection model and transition probability model for our unorthodox (but possible) Markov model, let's consider the emission model (the mapping from states to observed symbols). For simplicity, let's use the same emission model as was used for Figure 3.

Now, if we consider an observed sequence of length L, we can determine the number of unique sequences of length L that the model can emit, we can determine which sequences are NOT possible, we can determine all the paths that can produce a particular observed sequence, and we can determine the probability of any particular observed sequence, as well as the optimal probability (probabilities) paths of any particular observed sequence.

To do this, we are interested here in formalizing general algorithms that should be able to take ANY connection model, with ANY transition probability assignments, and ANY emission models, and determine any of the matters discussed in the paragraph above. Note that as the models become more complex and less uniform, it becomes harder to

model them with a mathematical formula, but still quite conducent to modeling them with an algorithm that could be executed by a computer.

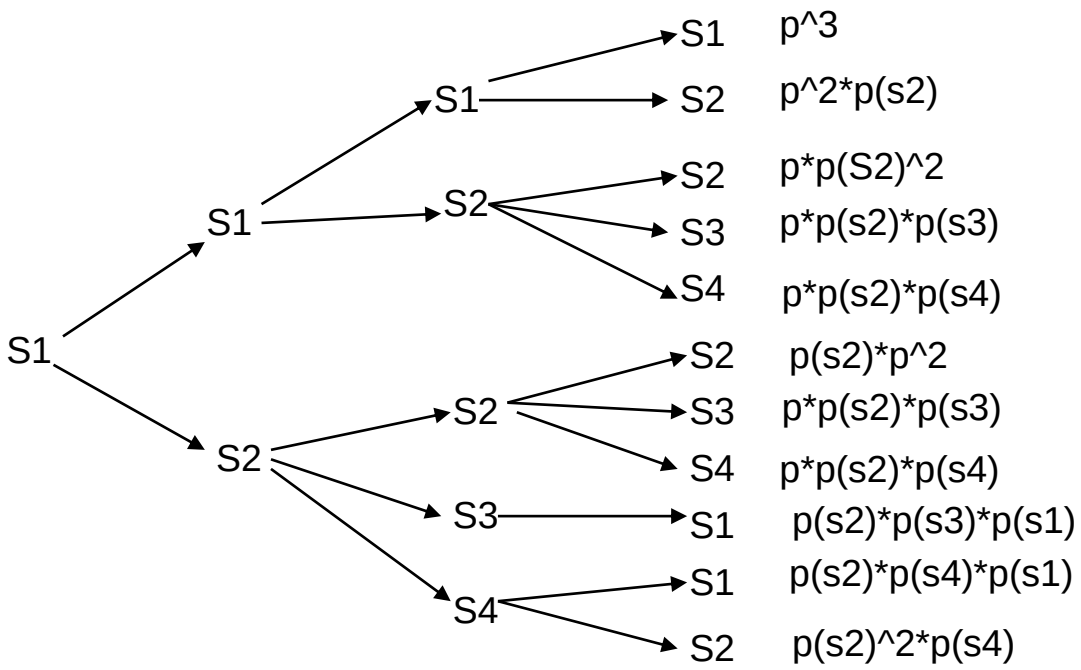


Figure 5: The 11 possible state paths of the Model in Figure 4, given start state S1 and L = 4 and their path probabilities.

Given start state S1, and our model and computational model will allow any state to be a start state, the paths and path probabilities are as stated in Figure 5. Note that the path probability calculation would include a probability for starting in state S1, either .25 or (p*p(s1)^2)/3, as only 2 states other than s1 can transition to S1 in this model (s3 and s4).

Now, with the emission model used in Table 5, we can state the observed sequences of length 4 that can be emitted by this model:

1. There would only be 5 unique observable sequences emitted: AAAA, AAAB, AAAC, AABA, and AACA.
2. Examples of impossible sequences would be: CAAA, AABC, or AACB.

Of the possible emitted sequences, we can determine all of the paths that could emit a specific possible observed sequence. Consider sequence ABAA. S2->s3->s1->s1 or s2->s3->s1->s2 are the only sequences that could have produced ABAA. The constraints on the model determine this: B can only be emitted by s3 and s3 can only be preceeded by s2, therefore the first two states in the sequence must be s2->s3. s3 only transitions to s1, therefore the third state must be s1, and since s1 can transition either to s1 or s2, two paths are possible.

But, what we want are general algorithms which can determine each of the parameters above.

Note to professor – I am working on these general algorithms and a program which can execute these general algorithms. A generalized version of the Viterbi and forward algorithms as well as some other algorithms should accomplish this. We could also use graph theory to create a model generator, where the user inputs n(s) states and fills in a connection matrix to determine which connections are possible (where the default connection or adjacency index is 0), and what the probabilities are for each connection. Along with an emission model, all of the parameters above are computable. I have written a program called graph creator which can be modified for this purpose.

Silent State HMMs

One other topic should be discussed in discussing the variety of Markov Models that can be used to model biological phenomena. This is "silent states." Simply, a "silent state" is a state which emits no symbols. It may be transitioned to or from like any other state, with a specified transition probability, but it itself emits no observable symbol. Consider the following model which includes silent states (represented as boxes rather than circles):

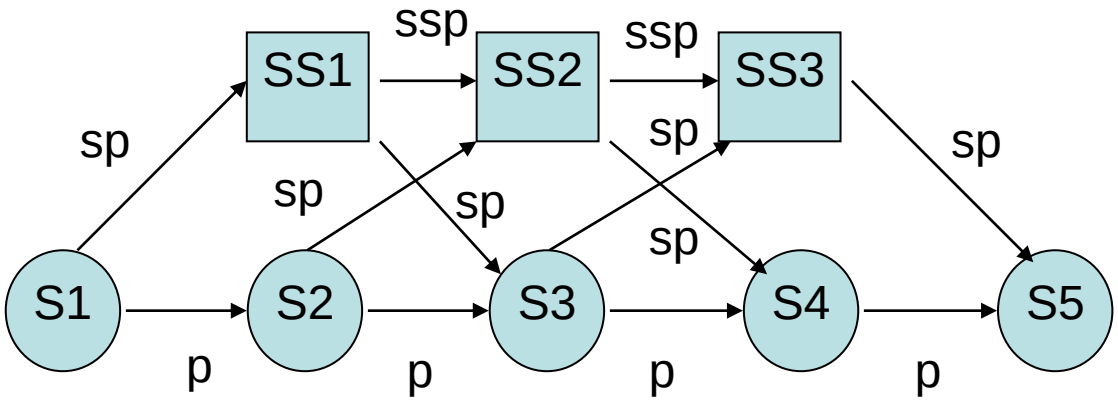


Figure 6: Example HMM with Silent States.

Here the HMM with silent states is a non-fully interconnected HMM with three uniform probabilities: one for transition between silent states (ssp), one for transition between real and silent states (sp), and one for transitions between real states. A 'real' state, simply, is one which emits a symbol. The following connection matrix describes the HMM with silent states in Figure 6:

	s1	s2	s3	s4	s5	ss1	ss2	ss3
s1	0	p	0	0	0	sp	0	0
s2	0	0	p	0	0	0	sp	0
s3	0	0	0	p	0	0	0	sp
s4	0	0	0	0	p	0	0	0
s5	0	0	0	0	0	0	0	0
ss1	0	0	sp	0	0	0	ssp	0
ss2	0	0	0	sp	0	0	0	ssp
ss3	0	0	0	0	sp	0	0	0

Table 7: Connection and Transition Probability Matrix for HMM in Figure 6 with Silent States.

Now, even if every real state emits one symbol, Figure 6 is still an HMM, because there is a many to one relationship between states sequences (paths) and observed symbols. I.e., because of the silent states many possible paths may result in the same sequence of observed symbols. Let's suppose real states 1 through 5 emit symbols A through E, respectively. Then, starting with state s1, and given a state path of 5 states, the following sequences could be emitted: ABCDE, ABCE, ABDE, ABE, ACDE, ACE, ACD, AC, AE, and A. I.e., there would be 10 paths, emitting 10 unique sequences. But note that since each path is 5 states long, and many emitted sequences are shorter than 5 symbols, silent states are involved in the emitted sequences being shorter than the number of state transitions. In general, models with silent states should avoid silent states which can transition to themselves, this could result in a large number of state transitions with no corresponding emitted symbols.

Higher Order Markov Models

So far the Markov Models we have been considering are of order one. Namely, the probability of the system being in a certain state at a certain time or a symbol occupying a certain position in a sequence depends only on the previous state. But, a Markov model could be developed where the probability of the system being in a certain state at a certain time is based upon the n previous states. Here n gives the order of the Markov model, i.e., n-order Markov models are possible. Thus far we have only considered order one Markov Models, and we may discuss higher order Markov Models elsewhere in this paper.

Selecting a Markov Model to Model a Biological Phenomenon

We have thus far introduced the breadth of Markov models, but we haven't discussed how to choose a Markov model among these many possible Markov models to model a certain biological phenomenon. To reiterate, three things must be chosen to use a Markov model to model a biological phenomenon: the connection model, the transition probability model, and the emission model. The connection model determines the finite state machine. If the biological phenomenon to be modeled can results in ANY possible combination of letters ACTG, then a fully connected model should be used as in our original example. The fully connected model can then be modified by the selection of

transition probabilities to influence which sequences are more or less likely. When determining the connection model, the number of states must first be selected. The states should each represent an actual state that the biological phenomenon could be in at any one time. The connection model should preclude sequences of states that are not possible in the observed phenomenon. Low transition probabilities should be selected for states that are unlikely to succeed eachother, and higher probabilities should be selected for states that are more likely to succeed eachother. The emission model should be a mapping of what possible states could result in the observed phenomenon or state of the biological system.

These are just guidelines as there is no science behind Markov Model design to model a biological phenomenon; it seems, rather, to be a bit of an art.

Section Summary.

In this section on Markov Models, I have attempted to show how MANY Markov models are possible. I have shown how ANY Markov model consists in three things: a connection model, a transition probability model, and an emission model. A uniform and an unorthodox model was presented to demonstrate the diversity of such possible models. There was also some discussion on how a model may be selected to model a biological phenomenon, which is more of an art than a science.

Part II - Markov Chains for Pairwise Alignment

Now we will turn our attention to another application of HMMs: pairwise alignment. By pairwise alignment, we mean aligning two sequences to determine whether they are related. These may be either nucleotide sequences or amino acid sequences, for example. In our example we will consider two amino acid sequences. By a 'pairwise alignment' more exactly we mean that for two sequences of lengths n and m respectively, the ith amino acid of each sequence is compared and scored. If the same amino acid is found at the ith position in the sequence, the score is high, if the two amino acids don't match at the ith position, the score may be positive or negative: if the score is positive, the second amino acid is likely to have been substituted for the first amino acid, i.e., there is still some evolutionary relation between the two amino acids; if the score is negative, the second amino acid is unlikely to have been substituted by the first amino acid.

The scoring system is used to identify sequences that are related that may not have many exact matches between amino acids (or nucleotides) based upon the theory that related sequences may look quite divergent over time based upon amino acid (or nucleotide) substitutions. Through evolution a sequence diverges create two or more related sequences, which may appear unrelated at a later time. These substitutions may include insertions, deletions, or mutations of the nucleotides in the DNA. Hence the scoring system is used to score these substitutions (insertions and deletions) as 'gaps.' A high score is accorded an exact nucleotide or amino acid match. If the two nucleotides or

amino acids at position i of both sequences do not match, a positive score may still be accorded if, it is determined by sampling a proteome or genome, that there is a likelihood of the amino acid (or nucleotide) of the second sequence would replace the amino acid (or nucleotide) of the first sequence. The likelihood may be based upon biochemical similarities, for example, between two amino acids. If two amino acids (or nucleotides) at position i of two sequences do not match, in most cases, a negative score will be recorded, indicating that it is unlikely that the amino acid in the second sequence, for example, would replace the amino acid at that position in the first sequence, i.e., it is more likely that the two sequences are not related.

The scoring system is based upon the log of the ratio of the likelihood that two nucleotides or amino acids would occur together over the likelihood that they would each occur separately. This can be expressed mathematically as:

$$s(a,b) = \log[\, p(a,b) / (q(a)q(b)) \,]$$

where a is amino acid or nucleotide one and b is amino acid or nucleotide two, and p(a,b) is the probability of the occurrence of the two nucleotides or amino acids at the same position i in a sequence, and q(a) is the probability of just the occurrence of amino acid or nucleotide a times q(b) the probability of just the occurrence of nucleotide or amino acid b at a position i in the sequence. The probabilities are derived from sampling the frequency of occurrences of nucleotides or amino acids or the frequency of their co-occurrence in a large number of sequences in genomic and proteomic databases, respectively.

Consider the following matrix, which is known as the BLOSUM50 substitution matrix, which gives log odds scores (log ratios) of the likelihood or unlikelihood of the co-occurrence of the twenty amino acids (hence a 20 by 20 matrix).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**5**	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	**7**	-1	-2	-4	**1**	0	-3	0	-4	-3	**3**	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	**7**	**2**	-2	0	0	0	**1**	-3	-4	0	-2	-4	-2	**1**	0	-4	-2	-3
D	-2	-2	**2**	**8**	-4	0	**2**	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	**13**	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	**1**	0	0	-3	**7**	**2**	-2	**1**	-3	-2	**2**	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	**2**	-3	**2**	**6**	-3	0	-4	-3	**1**	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	**8**	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	**1**	-1	-3	**1**	0	-2	**10**	-4	-3	0	-1	-1	-2	-1	-2	-3	**2**	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	**5**	**2**	-3	2	0	-3	-3	-1	-3	-1	**4**
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	**2**	**5**	-3	3	**1**	-4	-3	-1	-2	-1	**1**
K	-1	3	0	-1	-3	**2**	**1**	-2	0	-3	-3	**6**	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	**2**	3	-2	**7**	0	-3	-2	-1	-1	0	**1**
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	**1**	-4	0	**8**	-4	-3	-2	**1**	**4**	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	**10**	-1	-1	-4	-3	-3
S	**1**	-1	**1**	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	**5**	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	**2**	**5**	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	**1**	-4	-4	-3	**15**	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	**2**	-1	-1	-2	0	**4**	-3	-2	-2	**2**	**8**	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	**4**	**1**	-3	**1**	-1	-3	-2	0	-3	-1	**5**

Table 8: N x N table of log odds (log ratio) of 20 Amino Acid Matches and Substitutions.

Note that amino acids that match exactly, e.g. A:A, or K:K cells of the matrix, receive high scores. Elsewhere, although most of the scores are negative (logs of low probabilities), some non-exact matches are positive, e.g. amino acid M with amino acid L, which indicates the likelihood of a substitution of one amino acid for the other, probably due to biochemical similarities (more than exist between the already repeating amino group of all amino acids).

Digression on the Biochemical and Organic Chemical Basis of the Log Odds Amino Acid Match and Substitution Table.

It may be noted in the Amino Acid Match and Substitution Table that the highest scores are accorded to the pairing of one amino acid by the same amino acid. However where one amino acid is substituted by another amino acid (i.e., they don't match in the pairwise alignment), the score may be either positive (ranging from 1 to 4) or negative. Remember that the scores are derived from a statistical analysis of the probability of two amino acids being in the same position in two sequence (by frequency analysis of a large number of proteins) versus the random probability of the occurence of those two amino acids at that position in the sequence. Yet, the greater likelihood of two amino acids

being in the same position (e.g. Isoleucine (I) and Valine (V)) versus the unlikelihood of two amino acids occurring in the same position (e.g. Glycine after Leucine), is not random but due to the biochemical (or organic chemical) similarity and hence likelihood of substitution of one amino acid of another. The basic idea is that the less similar the organic chemical structures of two amino acids, the less likely they are to be substituted for eachother and inversely the more similar the organic chemical structures of two amino acids, the more likely they are to be substituted for another. Hence we turn briefly to a chemical analysis of the similarity of the paired amino acids to give the scientific basis for their low or high pairing scores.

Consider some unmatched amino acid pairs which receive high scores. Take two high scoring unmatched (i.e., substitutions have occurred) amino acid pairs: Valine (V) and IsoLeucine (I), and Pheylalanine (F) and Tyrosine (Y). These pairs are both biochemically similar. Of course analyzed here are only the R groups or the amino acid side chain that differs in each amino acid, not the repeating NH2-CH-COOH group that is the same for all amino acids.

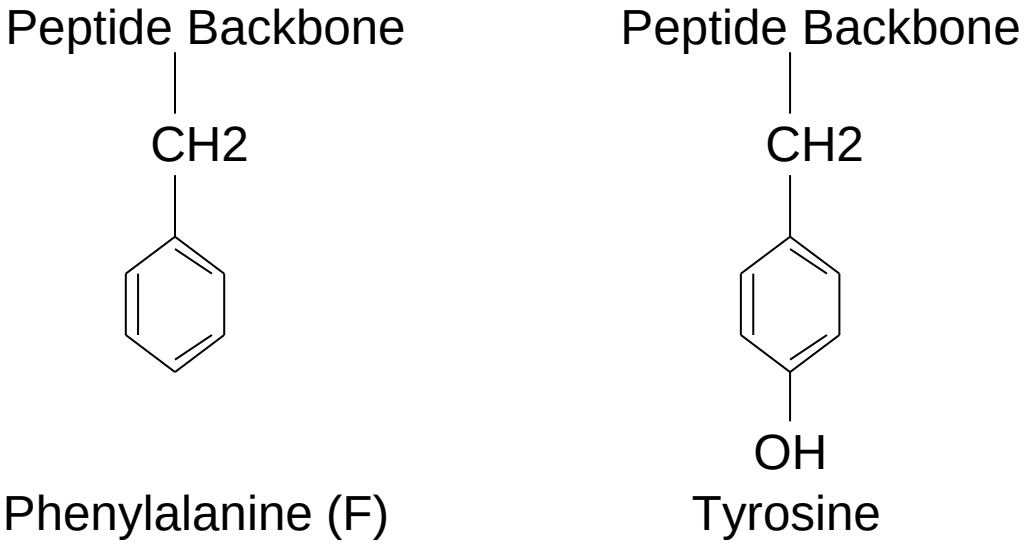
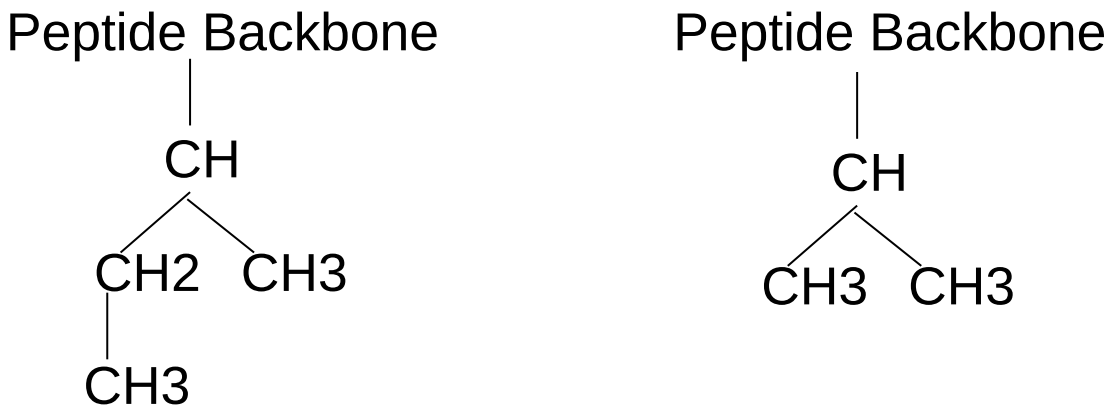


Figure 7: Phyenlalanine and Tyrosine differ only by a single hydroxyl group receiving score 4 in the subsitution matrix.



Isoleucine (I) Valine (V)

Figure 8: Isoleucine and Valine differ only by a single methyl group receiving score 4 in the subsitution matrix.

On the other hand, Leucine and Glycine are a pair that receive a negative four in the substitution matrix, because their chemical structures differ by four methyl groups and are thus less likely to be substituted:

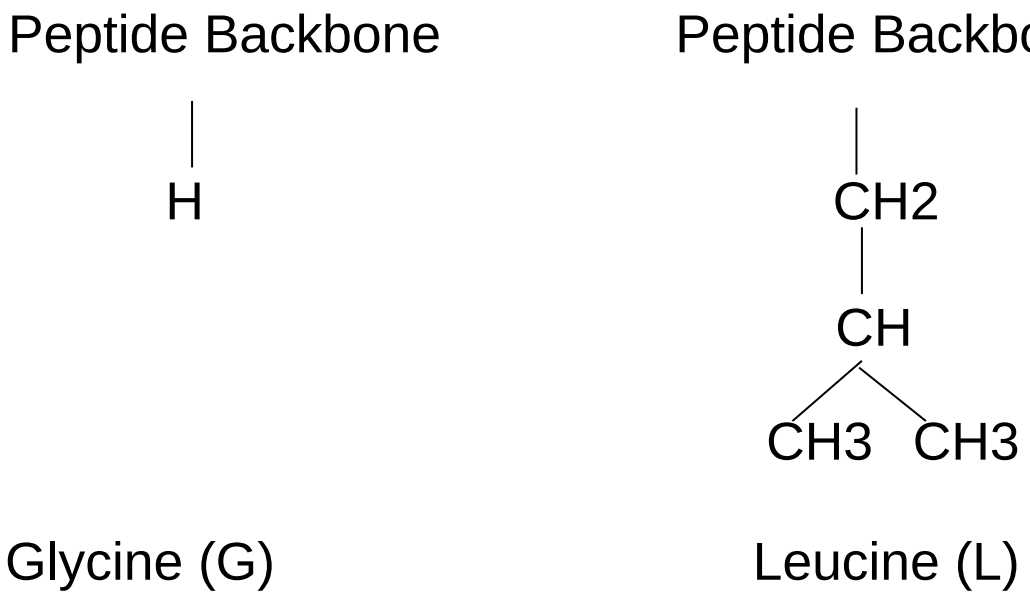


Figure 9: Leucine and Glycine differ by four methyl groups receiving score -4 in the subsitution matrix.

Let's look at one other unlikely substitution, scored -5 in the pairwise matrix: tryptophan (W) and aspartic acid (D). It will be seen that the chemical structures of these two molecules are so different that their substitution is highly unlikely:

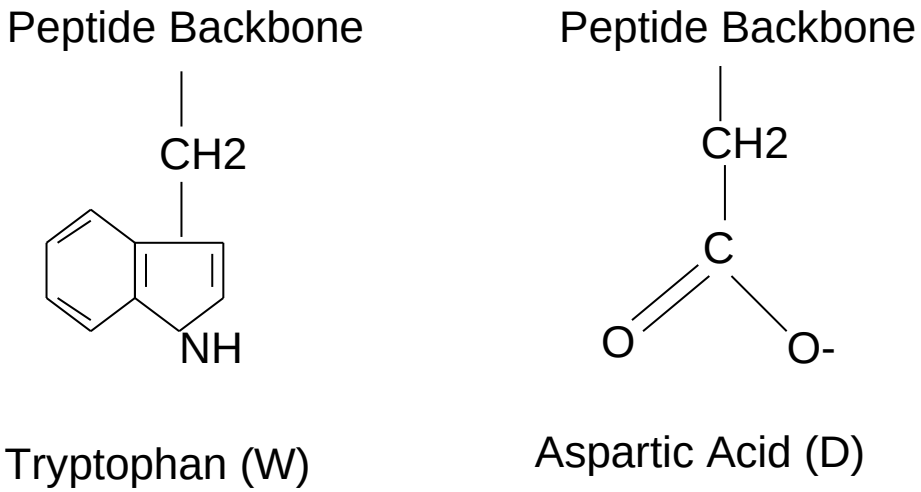


Figure 10:Tryptophan and Aspartic Acid differ so greatly in their chemical structure that they are extremely unlikely to be substituted for eachother, receiving a -5 in the subsitution matrix.

Next let's look at likely substitution that is nonetheless a bit more chemically involved than Valine to Isoleucine or Phenylalanine to Tryrosine, and receives therefore a score 3 rather than 4 (remember the actual scores are determine by a frequency analysis of proteins, this is simply a biochemical basis for the frequency of certain substitutions over others).

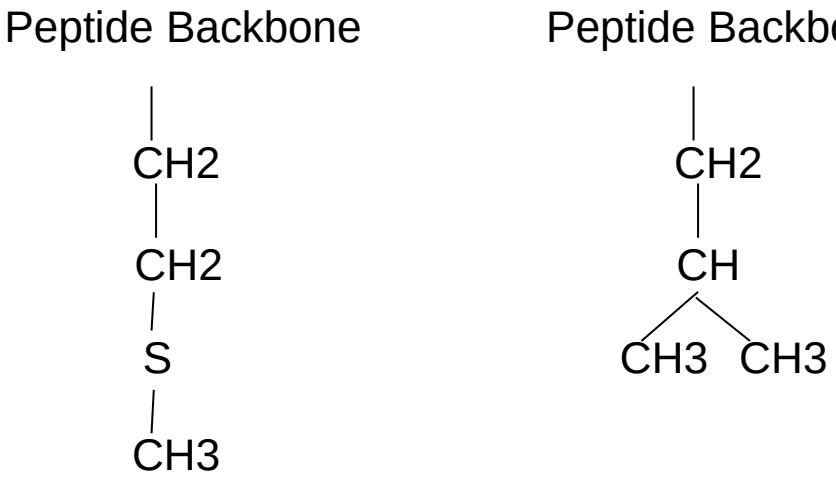


Figure 11: Leucine differs from Methionine by substituting a sulfur atom for a methyl group, a slightly more complicated rearrangement that receives a score 3 in the subsitution matrix.

The rearrangement involves substituting a sulfur atom from a methyl group, breaking the two bonds between the carbon and the two methyl groups and establishing a carbon-sulfur bond (for which the attached carbon receives one hydrogen, as carbon has a valence of four), and establishing a another carbon-sulfur bond between sulfur and the methyl group.

Let's look, just to round out our analysis, at the chemical basis of pairs that are scored 2 in the substitution matrix.

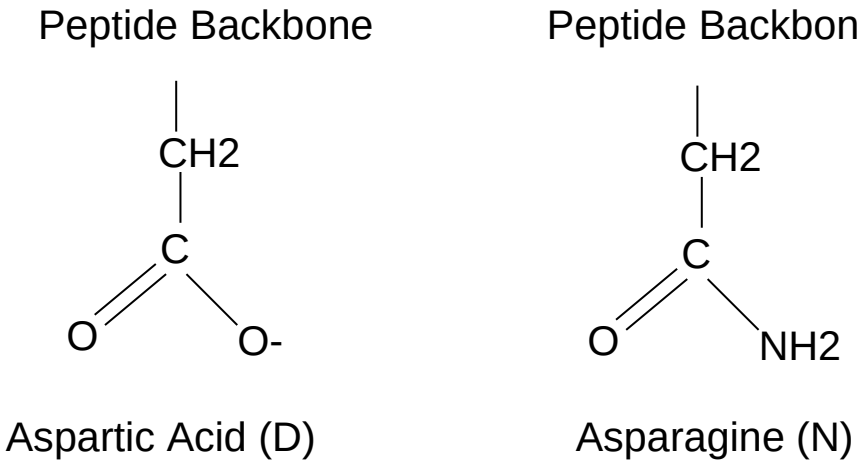


Figure 12: Aspartic acid can be substituted by Asparagine by substituting an amine (NH2) group for an ionic oxygen atom as part of a carboxyl group, changing an acid into a polar non-acid. The rearrangement receives a score 2 in the subsitution matrix.

Although there is only one substitution in this case, the substitution is less likely as a molecule that is acidic is being changed into a non-acid, which is nonetheless still polar. The oxygen ion is more likely, for example, to bond to a hydrogen atom than dissociate from carbon altogether and allow a nitrogen-carbon bond to be formed with an amine group.

Now that we have made our point that scores in the substitution matrix, which are determine by statistical analysis of the frequency of pairs in the same position in two sequences, ultimately are determined by the similarity (positive scores) or dissimilarity (negative scores) of the two amino acid molecules. The more similar the two amino acid's chemical structure the more likely they are to be substituted for eachother and thus paired; the more dissimilar the two amino acid's chemical structure, the less likely they are to be substituted with eachother and thus paired. Let us return to the pairwise alignment algorithm.

Pairwise Alignment Algorithms

Now we can turn our attention to the pairwise alignment algorithms. Consider the following n x m matrix comparing two amino acid sequences:

	H	E	A	G	A	W	G	H	E	E
P	-2	-1	-1	-2	**-1**	-4	-2	-2	-1	-1
A	-2	-1	5	0	**5**	-3	0	-2	-1	-1
W	-3	-3	-3	-3	-3	**15**	-3	-3	-3	-3
H	10	0	-2	-2	-2	-3	**-2**	**10**	-1	0
E	0	6	-1	-3	-1	-3	-3	-1	**6**	6
A	-2	-1	5	0	5	-3	0	-2	-1	**0**
E	0	6	-1	-3	-1	-3	-3	0	6	**6**

Table 9: Log Odds (Ratio) Table Comparing Two Amino Acid Sequences

The table is n x m because it is often the case that two sequences being compared that might be related are of unequal lengths. This is because through divergent evolution, the same sequence may diverge with deletions and insertions, which will make the divergent sequences of unequal lengths. Now, on the surface, PAWHEAE does not seem to be related to HEAGAWGHEE, but as we shall show, the two are related. By following the a simple algorithm which navigates the cells in the matrix, we shall show the sequences are related. Consider the top sequence to be numbered i through n and the second sequence on the side to be numbered j through m. Then, any cell is C(i,j).

1. Start with the ends of the two sequences, i.e., in the lower right-most cell.
2. If any of the adjacent cells: the cell to the left (C(i-1,j)), the cell on top (C(i,j-1)), or the cell diagonally opposed (C(i-1,j-1)) are the value of the current cell minus some value d, here 6, move to that cell. Only cells to the top, i.e., cell C(i,j-1) will be C(i.j)-d.
3. If any of the adjacent cells equals the value of the current cell minus e, here e equals 2d or 12, move to that cell. That cell will always be to the left, namely C(i-1,j).
4. If neither 3. nor 4. then go diagonally to cell C(i-1,j-1).
5. Stop when the current cell C(i,j) is either the left-most cell (i = 1), or the top-most cell (j=1).

Following this algorithm generates the path through the matrix visible in red. Now the sequences can be aligned, with gaps. The algorithm for this is:

1. Start with the lower right-most cell C(n,m). Move backward through the path through the matrix until either i=1 or j=1.
2. Record that letter for both sequences at C(n,m).
3. If you move up to cell C(i,j-1), record a blank or gap in the i through n sequence, and the letter at position j for the second j through m sequence.
4. If you move left to cell C(i-1,j), record a blank or gap in the j through m sequence, and the letter at position i for the i through n sequence.

5. Else, if you move diagonally to cell C(i-1, j-1), record both the letter at i for the i through n sequence, and the letter at j for the j through m sequence.

The result will be an alignment that looks like this:

_AWGHE_E
PAW_HEAE

You can see from this alignment that there are three substitutions at amino acids P, G, and A, and that there are five amino acid matches at positions 2,3,5,6, and 8 (-AW-HE-E).

The path through the matrix can be scored by adding (as these are logs) the values along the path. Here the value of the path is: 30, and this represents the highest value path through the matrix.

Now that we have introduced the algorithm that aligns and scores the sequence, we are ready to demonstrate how a Hidden Markov Model can be used in alignment. Consider that in the algorithm there are only three possible moves that can be made from any cell: to the left (to cell C(i-1,j)), to the cell on top (to cell C(I,j-1)), or diagonally (to cell C(i-1,j-1)). Since there are finite moves that can be made to traverse the matrix, the path can be described by a finite state machine or finite state automaton (FSA). Here's the finite state machine:

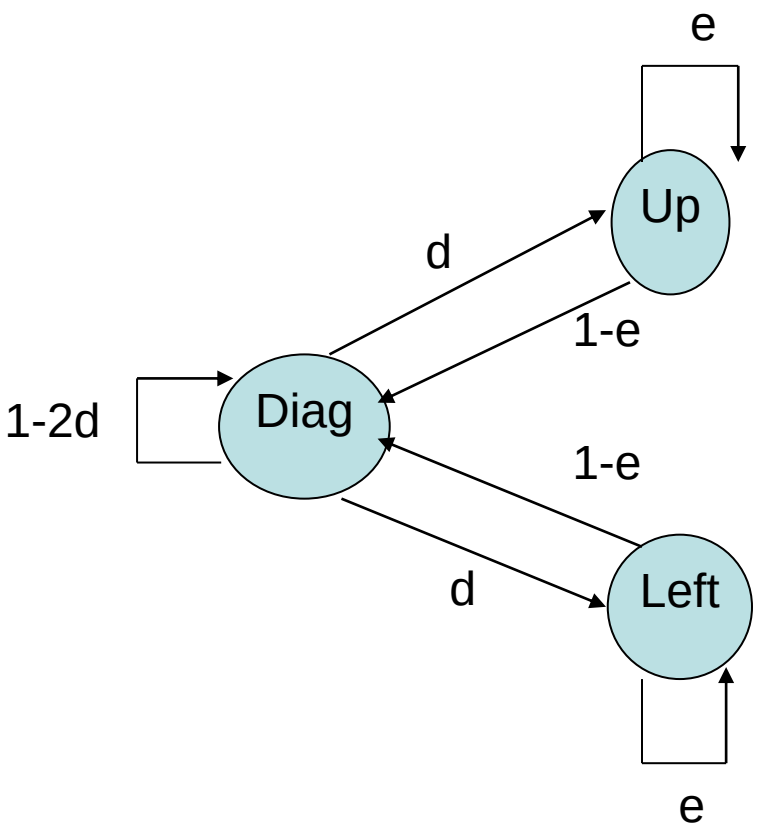


Figure 13: HMM Model of Transition Through Log Odds Matrix

In a 'Pair HMM' a pair of sequences are emitted by the HMM instead of a single sequence. Specifically, if in Diag state, the Pair HMM will emit a matched pair of amino acids, with emission probability p(a,b,), if in Up state, the model will emit a single amino acid paired with a gap, with probability Q(a), as the Left state will also emit a single amino acid paired with a gap, with probability Q(b). The transition probabilities are d for a Diag state to change to a gap state (Up or Left), and e for a gap state (Up or Left) to transition to another gap (Up or Left state, respectively) state. The transition probability 1-2d for a Diag state to transition to itself (i.e., to continue to traverse the log odds matrix in a diagonal manner) is derived from the fact that all states leaving the Diag state must sum to one, therefore 1-2d + 2d = 1. Likewise the 1-e probability of transitioning from an Up or Left state to a Diag state is derived from the fact that all transition probabilities leaving an Up or Left state must sum to one: 1-e + e = 1.

Hence it can be seen how, using the algorithms below (Viterbi and Forward algorithms), this Pair HMM can emit two sequences with matches and gaps that best align two sequences, by selecting the best "path" through the Log Odds matrix.

The Viterbi Algorithm for Pair HMMs

Here we will present a modification of the already presented Viterbi Algorithm specifically for Pair HMMs. The Viterbi algorithm will find the optimal path through the log odds matrix to find the best possible alignment of two sequences, e.g. two amino acid sequences.

0. Let the path probability = p = 0.
1. For i = 1 to n; For j = 1 to m:
2. Let p = p + [P(a,b)*max(3.) || Q(a) * max(4.) || Q(b)* max(5.)]
3. P(a,b) * Find max of:
    a. (1-2d)V^m(i-1,j-1) Diagonal to Diagonal state
    b. (1-e)V^x(i-1,j-1) Up state to Diagonal state
    c. (1-e)V^x(i-1,j-1) Left state to Diagonal state
4. Q(a) * Find max of:
    a. d * V^x(i-1,j) Diagonal to Left State
    b. e * V^y(i, j-1) Left to Left State
5. Q(b) * Find max of:
    a. d * V^m(I, j-1) Diagonal to Up state
    b. e * V^y(I, j-1) Up to Up state
6. Termination: Find max of (V^m(n,m), V^x(n,m), V^Y(n,m)
7. Print p. Traceback optimal path (i through n, j through m).

Note that going from an Up to Left or Left to Up state is not a state transition supported by the finite state machine.

A Pair HMM version of the forward algorithm can also be used to find the likelihood of ANY path emitting the observed amino acid sequence.

1. For i = 1 to n; For j = 1 to m:
2. F^m(I,j) = p(a,b)[1-2d)f^M(i-1,j-1)+(1-e)(f^x(i-1,j-1)) + (f^Y(i-1,j-1))]
3. F^x(I,j) = Q(a)[d*f^M(i-1,j)+(e*f^X(i-1,j)]
4. F^y(I,j) = Q(b)[d*f^M(I,j-1)+(e*f^Y(I,j-1))]
5. Termination: f^E(n,m) = f^M(n,m) + f^X(n,m) + f^y(n,m).

Part III - Markov Chains and Sequence Families

Let us now turn our attention to the final application of Markov chains that this paper will address: the construction of phylogenetic trees. Now that we have discussed pairwise alignment, consider that multiple (more than two) amino acid sequences may be compared, i.e., a multiple alignment may be performed, which searches for similarities between multiple sequences. Where similarities exist between multiple sequences, it is probable that they form a phylogenetic tree, i.e., a family tree of sequences, e.g. proteins, that are evolutionarily related. These proteins may exist in different organisms, such as yeast, man, and mouse, but all have a common ancestor, which diverged over time, to make proteins that are evolutionarily related but have slightly different amino acid sequences, due to substitutions, deletions, and insertions of amino acids.

In doing multiple alignments we use something called a Profile HMM (Hidden Markov Model), which is the most common use of an HMM in biology.

Note that in multiple alignments, protein gaps tend to line up with eachother indicating either insertions or deletions at this point in the amino acid sequence. Although a more sophisticated technique is generally used, we can use the rule that if

Consider the seven proteins below (the matrix representing an aligned section of each protein, where amino acid one on the NH2 end of the peptide bond are aligned for all proteins.

HBA-HUMAN	L	S	P	A	D	K	T	N	V	K	A	A	W	G	K
HBB-HUMAN	L	T	P	E	E	K	S	A	V	T	A	L	W	G	K
MYG-PHYCA	L	S	E	G	E	W	Q	L	V	L	H	V	W	A	K
GLB3-CHITP	L	S	A	D	Q	I	S	T	V	Q	A	S	F	D	K
GLB5-PETMA	L	S	A	A	E	K	T	K	I	R	S	A	W	A	P
LGB2-LUPLU	L	T	E	S	Q	A	A	L	V	K	S	S	W	E	E
GLB-BLYDI	L	S	A	A	Q	R	Q	V	I	A	A	T	W	K	D
CONSENSUS:	**L**	**s**	**.**	**.**	**.**	**.**			**v**		**a**		**W**		**k**

Table 10 – Consensus Model of a Protein Family

The simplest analysis of a multiple alignment involves a consensus, i.e., matching amino acids are tallied at each position along the sequence: here 6/7 matches designed by a capital letter for the amino acid, 4/7 to 5/7 matches indicated by a lower case letter for the most prevalent amino acid, and a dot representing 3/7 matches of an amino acid. In this section of protein, there are 10 out of 15 matches, 2 6/7 matches, 4 4/7 or 5/7 matches, and 4 3/7 matches.

The above represents an ungapped scoring method for proteins or protein segments. We have to move beyond this however and come up with a scoring method that allows gaps (insertions or deletions) in amino acid sequences being compared.

A Digression on Combinatorics for Proteins.

In the biological world the number of actual proteins used in biological systems are a small fraction of the number of 'possible' proteins. If you consider that the average protein is about 1000 amino acids long (proteins range from 30 amino acids to 10,000, with the vast majority being between 50 and 2000 amino acids long), then, since any one position along the amino acid sequence can be occupied by 20 possible amino acids, there are 20^1000 possible unique proteins 1000 amino acids long. This is an unfathomably large number. On the other hand, only a small fraction of the possible proteins that could be generated by the genetic code are ever generated in biological systems. There are (need a good number here) 28685 known proteins (from the Swiss Protein Bank and TrEMBL). Although more proteins are being discovered all of the time, the number is a ridiculously small fraction of the theoretically large number of possible proteins. The reason for this seems to be that only certain proteins form stable tertiary structure conformations which allow them to carry out consistent biological tasks (the tertiary structure ultimately being a function of the amino acid sequence, i.e., a function of the hydrogen bonds, ionic bonds, and Van der Waals forces formed between sequences of amino acid side chains that may either be polar, non-polar, acidic, or basic). Moreover, proteins which successfully serve a certain biological function in a cell tend to be reused again and again throughout evolution in different organisms, with perhaps slight variations; i.e., this small number of proteins actually used in biological systems is highly redundant, with a large number of proteins similar in amino acid sequence and hence tertiary structure and function across a wide range of organisms. The reason for this is that cells, the basic and definitive building block of all life, are formed, duplicated, structured, and carry out cell metabolic tasks in similar fashion. Compare the protein makeup of five organisms below:

Table 11. Estimated Total Number of Proteins and Number Predicted to Function in Certain Cell Processes in Microorganisms with Sequenced Genomes*

	M. genitalium	*M. jannaschii*	*H. influenzae*	*E. coli*	*S. cerevisiae*
Total genome length (kb)	580	1660	1830	4640	12,050
Total number of proteins†	470	1700	1700	4300	6200
Number of proteins with predicted functions in‡					
Metabolism	50	230	325	650	650
Energy production and storage	40	130	140	240	175
Transporters	40	60	150	280	250
DNA replication, repair, and recombination	40	90	110	120	175
Transcription	12	20	30	230	400
Translation	100	110	125	180	350
Intracellular protein targeting and secretion	20	25	35	35	430
Cell structure	10	40	110	180	250

* *From left to right, the organisms are* Mycoplasma genitalium, Methanococcus jannaschii, Haemophilus influenzae, Escherichia coli, *and* Saccharomyces cerevisiae. M. jannaschii *is an archaen;* M. genitalium, H. influenzae, *and* E. coli *are bacteria; and* S. cerevisiae *is a single-celled eukaryote.*

Values are the approximate total number of proteins encoded in each genome. They are based on the number of long open reading frames, which can encode proteins containing 100 or more amino acids, plus shorter genes encoding characterized proteins.

For each organism, only some of the proteins with predicted functions fall into the categories included in the table. The percentages of the estimated total number of proteins that have been assigned predicted functions vary among these organisms as follows: M. genitalium, *87%;* M. jannaschii, *38%;* H. influenzae, *83%;* E. coli, *62%; and* S. cerevisiae, *65%.*

SOURCES: *M. genitalium,* C. M. Fraser et al., 1995, *Science* **270:**397; *M. jannaschii,* C. J. Bult et al., 1996, *Science* **273:**1058; *H. influenzae*.R. D. Fleischmann et al., 1995, *Science* **269:**496; *E. coli,* F. R. Blattner et al., 1997, *Science* **277:**1453; *S. cerevisiae,* A. Goffeau et al., 1996, *Science* **274:**546. See also E. V. Koonin et al., 1997, *Mol Microbiol.* **25:**619.

By the above information, only 2% to 6% of protein in these organisms determine the cell's structure. I.e., most protein, which is similar in nature across organisms, is involved in common cell tasks common to all organisms, like transcription and translation, or cell metabolism, and only 2-6% of the protein accounts for the structural differences that we know as different types of cells that make up the different types of tissues and features of different organisms (for example the difference between a nerve cell and a myocyst, a muscle cell, is primarily although not exclusively a difference in structural proteins). I.e., in stem cell research, stimulating cells to differentiate into specific cell types, e.g. a neuron versus a myocyst, is a matter of using transcription factors to turn on genes to (through translation) produce different structural proteins.

So there are two points from a bioinformatics point of view. One, the number of proteins used in living organisms is a minute fraction of the combinatorially possible proteins that amino acids could produce, and that even the relatively small number of proteins in living systems are highly redundant across organisms.

Profile HMMs: Hidden Markov Models for Performing Multiple Alignment of a Set of Sequences

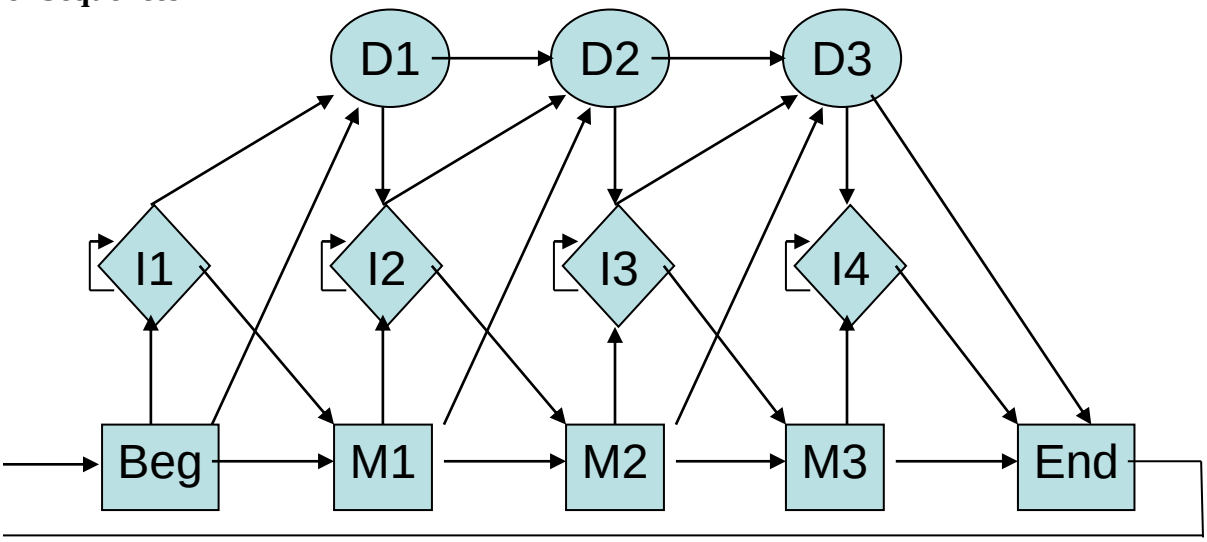


Figure 14 – A Hidden Markov Model (HMM) or Profile HMM for multiple alignment of amino acid or Nucleotide sequences to identify sequence families.

The above figure is a Hidden Markov Model or Profile HMM to perform multiple alignments (alignment of more than two sequences) of either nucleotides or amino acids, to identify gene families or protein families. The model consists of three states: M or Match states, where more than half of the sequences nucleotides or amino acids match at

the same position in the sequence; D or Delete states where in match states a match may be missing at that position (Delete states must number less than half of the number of sequences being aligned); and I or Insert states, where there is no Match state among the sequences at a specific position and the number of gaps (or mismatches) are more than half the number of sequences being aligned. Both Match states and Insert states can emit letters (amino acids or nucleotides), but Delete states can't. Transitions proceed from the first match state to the last (in this model 3 match states are allowed), corresponding to match positions in the sequence, with delete and insert states able to intercede. The model is a forward model where match states cannot go back to previous match states, previous insert states or previous delete states, insert states cannot go back to previous match states or previous delete states but can loop back upon themselves, and delete states cannot go back to previous match states or previous insert states. Once the model iterates once through the n match states (here n = 3), it is restarted for the next iteration as indicated by the loop back from the end state to the begin state. In this way, nucleotide sequences of length L are iterated through repeatedly by restarting the model, n match states at a time, requiring L/n iterations through a multiple alignment of a set of sequences.

To illustrate how the finite state machine Profile HMM works, let's start with a simple example involving the multiple alignment of nucleotide sequences, and then we'll proceed to an example involving amino acid sequences.

BAT	A	G	-	-	-	C
RAT	A	-	A	G	-	C
CAT	A	G	-	A	A	-
GNAT	-	-	A	A	A	C
GOAT	A	G	-	-	-	C
POS:	1	2	.	.	.	3

Table 12 – Pseudo-example of multiple alignment of a family of nucleotide sequences

Consider the pseudo-example above, representing one iteration through the Profile HMM. In these five sequences, only in positions designated 1, 2, and 3 are there match states. In position 2, there are two delete states, and in position 3 there is one delete state. The alignments between match states 2 and 3 represent insert states, where in some cases, as in the BAT sequence, there are no nucleotides emitted from the insert state, and in others, e.g. the GNAT sequence, there are nucleotides emitted.

If we were to track the state transitions emitting the above family of sequences, it could be described by the table below:

		Match Position			
		0	**1**	**2**	**3**
M					
Emission	**A**	-	4	0	0
	C	-	0	0	4
	G	-	0	3	0
	T	-	0	0	0
I					
Emission	**A**	0	0	6	0
	C	0	0	0	0
	G	0	0	1	0
	T	0	0	0	0
	M->M	4	3	0	0
	M->D	1	1	0	0
	M->I	0	0	3	0
	I->M	0	0	0	4
	I->D	0	0	0	1
	I->I	0	0	10	0
	D->M	-	0	0	1
	D->D	-	1	0	0
	D->I	-	0	2	0

Table 13 – The number of emissions and transitions of the Profile HMM in Figure 12 to generate the nucleotide sequence in Table 12.

Let Position 0 refer to the begin state. Then the 4 A's and one delete gap in the first match state are enabled by 4 M->M transitions and one M->D transition from the begin state. Likewise the second match position is enabled by the following state transitions from the first match state: 3 M->M transitions, 1 D->D transition, and 1 M-D transition. The next alignment is NOT a match state, as fewer than half the nucleotides are the same; this is true for the next 3 alignments. Hence the finite state machine transitions into insert states. The three match states in position two transition into insert states, while the two delete states transition into insert states that emit A's. In the next multiple alignment, the two insert states that emitted symbols transition into another insert state that emits G for RAT and A for GNAT, whereas the two of the three insert states that did not emit symbols loop back to themselves (another emissionless insert state) and one transitions to an insert state that emits an A. Now the insert sequence –GAA- transitions to - - AA- by to emissionless insert states again transitioning to themselves, one emissionful insert state transitioning to an emissionless insert state and two emissionful insert states transitioning to two more insert states with emissions. Finally in generating the third match state: CC-CC, three emissionless insert states transition to match states, and one emissionful insert state transitions to a delete state and the other to a match state. If this verbiage is confusing, studying Table N should clarify.

The diagram below may elucidate these transitions somewhat:

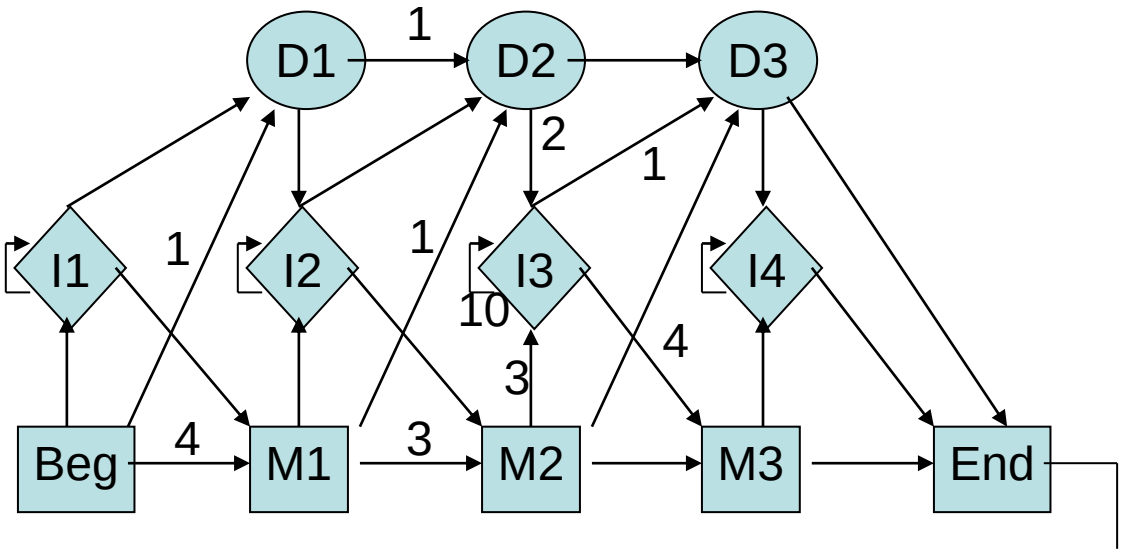


Figure 15 – A model indicating the number of transitions between each state in the Profile HMM to generate the sequence in Table 12.

Note that the number of transitions in the diagram equals the number of sequences times the number of alignments, here, 30. It is clear from this diagram that all of the insert states take place between the second and third match states. What is not indicated here, however, is which of the insert states emit symbols and which do not.

Now let the symbols A, C, and G represent amino acids, say Alanine, Cysteine, and Glycine. The exact same profile HMM could generate the same alignment between amino acids. I.e., the finite state machine is applicable to multiple alignment between any family of sequences regardless of what their symbols represent.

Probabilities, Profile HMMs and the Viterbi and Forward Algorithms

A Profile HMM like any other HMM is ultimately a probabilistic finite state machine. So let's consider the probabilistic aspects of our Profile HMM.

	Transition States:		
	M	**D**	**I**
M	M(j-1)Mj	M(j-1)Dj	MjIj
D	D(j-1)Mj	D(j-1)Dj	DjIj
I	I(j-1)Mj	I(j-1)Dj	IjIj

Table 14: Transition States in Profile HMM

Consider that the Profile HMM has three states: M or Match, D or Delete, and I or Insert. There are then 9 possible transitions in the Profile HMM: a Match to Match state, Match to Delete, Match to Insert, Delete to Match, Delete to Delete, Delete to Insert, Insert to Match, Insert to Delete, or Insert to Insert (an Insert self-loop). State transitions other than those involving Inserts go to the next Match position in the HMM. Insert states, for example, may iterate in the same match position, the number of iterations indicating the length of the insert gap. Here j refers to the Match position in the matrix, hence a transition like M(j-1)Mj indicates a match in position j-1 moving to position j.

In addition, there are emission probabilities which are considerations in the match states and insert states. All match states emit symbols, but insert states may be silent, where the log of the emission probability equals zero, or may emit symbols, where the log of the emission probability is greater than zero.

From these considerations we may now discuss the Viterbi and Forward algorithms for Profile HMMs. Remember that the Viterbi algorithm finds the most probable path through a set of sequences, and the Forward algorithm finds the probability of the set of observed sequences being emitted. The Viterbi algorithm for a Profile HMM would then be:

Letting j = 0 through L or the length of the sequences being multiply aligned:

$$V(j)M = \log e(Mj)(xi)/q(xi) + \max \{ \log a\, M(j-1)Mj \}$$
$$\text{Log a } D(j-1)Mj$$
$$\text{Log a } I(j-1)Mj \}$$

$$V(j)D = \max \{ \log a\, M(j-1)Dj$$
$$\text{Log a } D(j-1)Dj$$
$$\text{Log a } I(j-1)Dj \}$$

$$V(j)I = \log e(Ij)(xi)/q(xi) + \max \{ \log MjIj$$
$$\text{Log } DjIj$$
$$\text{Log } IjIj \}$$

Where e is the emission probability of the Match or Insert state, for the symbol emitted at position I, over the probability of that symbol randomly occurring at that position. Where the two are the same, the log reduces to zero, as in the case of silent Insert states. The max of the transition probabilities to the Match states are found, for each position, as are the max of the transitions probabilities to the Deletes states and the max of the transition probabilities to the Insert states, to find the optimal probability through the HMM, and hence the optimal path.

In the Forward algorithm, to calculate the probability of the observed set of sequences being observed at all, we replace maximums with sums:

$$F(j)M = \log e(Mj)(xi)/q(xi) + \log[M(j-1)Mj + I(j-1)Mj + D)j-1)Mj]$$

$$F(j)D = \log[M(j-1)Dj + I(j-1)Dj + D(j-1)Dj]$$

$$F(j)I = \log e(Ij)(xi)/q(xi) = \log[MjIj + IjIj + DjIj]$$

Summary

In this paper we have presented the notion of a Markov Chain and a Hidden Markov Model, and demonstrated the use of Hidden Markov Models in three applications that involve analyzing biological sequence data: identifying probable gene locations by identifying CG islands, pairwise alignment, i.e., aligning two sequences to see whether they are related, and finally multiple alignment, aligning multiple sequences to identify sequence families such as common proteins or genes. The diversity of HMMs was discussed, as well as their use.

Probabilistic models were used because sequence data in biological entities, such as nucleotide sequences in DNA or amino acid sequences in protein are NOT random sequences. Rather existing sequences are constrained by biochemical considerations, as we have illustrated in the case of pairwise alignment of amino acids, by the machinery of DNA, as we have indicated by the higher probabilities of C's and G's occurring in CG

islands, and by evolutionary divergence, as we have indicated by related sequence families, such as protein families, where members in the family have a common ancestor that has diverged by substitutions, deletions, and insertions of amino acids. Again, we have shown some of the biochemical considerations in which substitutions of amino acids are most likely.

Since we are dealing with not random sequences in biological data, but more or less likely sequences, probabilistic models are appropriate. And since we are dealing with large 'sequences,' of a small number of repeating elements, finite state machines, such as HMMs are the appropriate mechanism to generate (or emit) those sequences. The promise for using mathematics, such as finite state machines and probability to analyze biological data is great, as in living systems we are dealing with a small number of chemicals combined in probabilistically related sequences to give rise to the tremendous diversity of life.

```
// ========================================================================
// OOMarkov2.cpp  Created by Eric Wasiolek  May 30, 2005
//
// This program inputs a DNA sequences and automatically calculates the Markov
// chain probability of that sequence according to a specified probability
// transition matrix.  This is the object-oriented version.
//
// Added:  The user can define their own transition probability matrix.
// The probability matrix is read from a file as is the DNA sequence.
// The program automatically checks to see if the transition probability matrix
// is valid:  it must have a) 16 values, b) all values must be between 0 and 1,
// and c) all rows of values (every 4 values) must sum to 1.0, otherwise the
// program prints and error message and exits.
//
// Added:  The program verifies that the DNA sequence is a valid sequence
// file, i.e., that it ONLY contains A,C,G, or T's, otherwise the program
// prints and error message and re-requests input.
// The next version will be GUI (Windows interface) based.
// ========================================================================
/*
#include <iostream>
#include <string>
#include <iomanip>
#include <cmath>
#include <fstream>
#include <cctype>

using namespace std;

#define MULT_IDENT 1
#define NUM_NUCLEOTIDES 4
#define MAXLINES 100
#define MATRIX_FILE_ERROR 1
#define INIT_PROB_ERROR 1
#define ERROR_TERM .001

typedef enum Nucleotide{ A, C, G, T };
ifstream InFile;


class Markov {

// Declare private class variables
private:


        float tm[4][4];
```

```
        string Nucs;

        float InitProb;

// private methods, called by the public methods
private:
   Nucleotide char2enum(char);
   bool CheckNucleotides(string);
   ValidateTransProbMatrix(ifstream&);

// public methods, provide the user's interface to this class
public:
        string InputSeq();
        InputTransProbMatrix();
        float CalcTransProb(string);
        Markov();  // Constructor
        DefineMatrix(ifstream&);
        OutputTransProbMatrix();
        OutputSeqProb(float);
        SetInitProb(float);
        InputInitProb();

};

int main()
{

        Markov DNASeq;

        DNASeq.InputTransProbMatrix();

        cout << endl << endl;

        DNASeq.InputInitProb();

        string seq = DNASeq.InputSeq();

        cout << endl << "The sequence input is:  " << seq;

        cout << endl << endl;

        DNASeq.OutputSeqProb(DNASeq.CalcTransProb(seq));


return 0;

}

Markov::Markov()
{


        for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
                for(int j=0; j <NUM_NUCLEOTIDES; ++j)
                        tm[i][j] = 0;

// Absolutely cannot define a static array in the constructor!!!!

        Nucs  = "ACGT";

        InitProb = MULT_IDENT;

}

Markov::DefineMatrix(ifstream& InputFile)
{
        if(!InputFile)
                cout << "Can't find the file! " << endl;
        else
        {
                ValidateTransProbMatrix(InputFile);
```

```
                InputFile.clear();  // Yes!  Must rewind file!
                InputFile.seekg(0);
                for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
                        for(int j=0; j <NUM_NUCLEOTIDES; ++j)
                        {
                                InputFile >> tm[i][j];
                                cout << "Diagnostic:  " << tm[i][j] << endl;
                        }
        }
}

Markov::OutputTransProbMatrix()
{
        cout << "The transition probability matrix is: " << endl;
        cout << "\tA\tC\tG\tT" << endl;
        for(int i = 0; i < NUM_NUCLEOTIDES; ++i)
        {
                cout << Nucs[i], ":  ";
                for(int j = 0; j < NUM_NUCLEOTIDES; ++j)
                        cout << fixed << setprecision(3) << "\t" << tm[i][j];
                cout << endl;
        }
}

Markov::OutputSeqProb(float path_prob)
{
        cout << "The Markov probability of this seq of nucleotides is:  ";
        cout << fixed << setprecision(10) << path_prob;
        cout << endl << endl;
        cout << "The Markov probability expressed as a log is:  ";
        cout << fixed << setprecision(2) << log10(path_prob);
        cout << endl << endl;
}

Nucleotide Markov::char2enum(char c)
{

        switch(c)
        {
        case 'A' : return A;
        case 'C' : return C;
        case 'G' : return G;
        case 'T' : return T;
        default  : cout << "Invalid nucleotide\n";
        }

}



float Markov::CalcTransProb(string seq)
{

        float transprob = InitProb;

        for (int k = 0; k < seq.length()-1; k++)
        {
                transprob *= tm[char2enum(seq[k])][char2enum(seq[k+1])];
        }

        return transprob;

}

string Markov::InputSeq()
{

        string Seq = "";
        string SubSeq[MAXLINES];
        char Source;
        string filename;
```

```
        int count = 0;

        cout << "Would you like to input a sequence from the keyboard (K) or ";
        cout << "from a File (F)? ";
        cin.get(Source);
        switch(toupper(Source))
        {
        case 'K' :  {
                                        do{
                                        cout << endl << endl << "What's the sequence? ";
                                        cin >> Seq;
                                        }
                                        while(false == CheckNucleotides(Seq))
                                        ;
                                        break;
                                }

        case 'F' :  {
                                        cout << endl << "What's the file name? ";
                                        cin >> filename;
                                        InFile.open(filename.c_str());
                                        if(!InFile)
                                                {
                                                        cout << endl << "Cannot find or open
file!" << endl;
                                                        cout << "Re-input the sequence or a
valid filename!";
                                                        cout << endl << endl;
                                                        Seq = InputSeq();
                                                }
                                        else
                                                {
                                                        cout << endl << "File \"" << filename
<< "\" opened successfully!" << endl;
                                                        while(!InFile.eof())
                                                                {
                                                                InFile >> SubSeq[count];
                                                            Seq += SubSeq[count];
                                                                if(false ==
CheckNucleotides(Seq))
                                                                {
                                                                        cout << "Re-input the
file with a correct DNA Sequence";
                                                                        cout << endl <<
"composed only of A,C,G, or T! ";
                                                                        cout << endl << endl;
                                                                        Seq = InputSeq();
                                                                }
                                                            cout << "SubSeq = " <<
SubSeq[count] << endl;
                                                            ++count;
                                                                }
                                                }
                                        break;
                                }

        default :   {
                                        cout << endl << "You must input either a 'K' or an
'F' ";
                                        cout << endl << endl;
                                        Seq = InputSeq();
                                }
        }  // end switch

InFile.close();
return Seq;

} // end InputSeq() function

bool Markov::CheckNucleotides(string Sequence)
{
```

```
        int count = 0;
        for(int i = 0; i < Sequence.length(); ++i)
        {
                if(        Sequence[i] == 'A'
                    || Sequence[i] == 'C'
                       || Sequence[i] == 'G'
                       || Sequence[i] == 'T'
                  )
                ++count;
                ;
        }

                if(count == Sequence.length())
                        return true;
                else if (count < Sequence.length())
                {
                        cout << "The sequence may only contain 'A', 'C', 'G', or 'T' ";
                        cout << endl << "Re-input the sequence! " << endl << endl;
                        return false;
                }
}

Markov::ValidateTransProbMatrix(ifstream& MatrixFile)
{

        int count = 0;
        float sum = 0;
        bool problem = false;
        float value = 0;

        while(!MatrixFile.eof())
        {
                MatrixFile >> value;
                ++count;
                sum += value;
                if(!(value > 0 && value < 1))
                {
                        cout << "There is a transition probability is not between 0 and 1!"
<< endl << endl;
                        problem = true;
                }

                if(count % NUM_NUCLEOTIDES == 0)
                {
                        cout << "Sum at this point is:  " << fixed << setprecision(10) <<
sum << endl;
                        if((fabs(1-sum) >  ERROR_TERM) && (sum != 1))
                        {
                                cout << "Each row must sum to 1! Re-input transition
probability ";
                            cout << "values for this row!" << endl << endl;
                               problem = true;
                        }

                        sum = 0;
                }
        }

        if((count - 1) != (NUM_NUCLEOTIDES*NUM_NUCLEOTIDES))
        {
                cout << "You must have exactly " << NUM_NUCLEOTIDES*NUM_NUCLEOTIDES << "
values ";
                cout << "in your transition probability matrix!" << endl << endl;
                problem = true;
        }

                if(problem == true)
                {
                        cout << "Your transition probability input matrix is not valid." <<
endl;
                        cout << endl << "Please fix it and re-run the program." << endl <<
```

```
endl;
                        cout << "Exiting!" << endl << endl;
                        exit(MATRIX_FILE_ERROR);
                }

}

Markov::InputTransProbMatrix()
{

        string FileName;
        cout << "What is the name of your Transition Probability Matrix File? ";
        cin >> FileName;

        ifstream TransMatrix;
        TransMatrix.open(FileName.c_str());

        // Just call these methods directly, they are part of the same class
        // as is this method, don't have to attach them to an object
        DefineMatrix(TransMatrix);
        OutputTransProbMatrix();

        cout << endl << endl;

}

Markov::SetInitProb(float InitialProb)
{
        if((InitialProb > 0) && (InitialProb < 1))
                InitProb = InitialProb;
        else
                {
                cout << "Initial probability must be between 0 and 1!";
                cout << endl << "Re-run program with a valid initial probability." <<
endl;
                cout << "Exiting!";
                exit(INIT_PROB_ERROR);
                }
}

Markov::InputInitProb()
{
        string Input;
        float InitialProb;
        float DefaultInitProb = 1;

        cout << "Would you like to define an initial probability?  (Y)es or (N)o :  ";

        cin >> Input;

        switch(toupper(Input[0]))
        {
                case 'Y' : {
                                                cout << "What is the initial probability? ";
                                                cin >> InitialProb;
                                                SetInitProb(InitialProb);
                                                cout << endl << endl;
                                                break;
                                        }

                case 'N' : {
                                                cout << "The initial probability will be set
to the default: 1";
                                                cout << endl << endl;
                                                SetInitProb(DefaultInitProb);
                                                break;
                                        }

                default :  {
                                                cout << "Your answer must be either (Y)es or
(N)o.";
```

```
                                                cout << endl << endl;
                                                InputInitProb();
                                                break;
                                        }
                }

        }
        */
```