Algorithms for Molecular Biology **Fall Semester, 2001**

Lecture 5: December 13, 2001

Lecturer: Ron Shamir *Scribe: Roi Yehoshua and Oren Danewitz* [1]

5.1 Hidden Markov Models

5.1.1 Preface: CpG islands

CpG is a pair of nucleotides C and G, appearing successively, in this order, along one DNA strand. It is known that due to biochemical considerations *CpG* is relatively rare in most DNA sequences [4]. However, in particular short subsequences, which are several hundreds of nucleotides long, the couple CpG is more frequent. These subsequences, called *CpG islands*, are known to appear in biologically more significant parts of the genome, such as around the promoters or 'start' regions of many genes. The ability to identify these CpG islands in the DNA will therefore help us spot the more significant regions of interest along the genome.

We will consider two problems involving CpG islands : First, given a short genome sequence, decide if it comes from a CpG island or not. Second, given a long DNA sequence, locate all the CpG islands in it.

5.1.2 Reminder: Markov chains

Definition A *Markov chain* is a triplet $(Q, \{p(x_1 = s)\}, A)$, where:

- Q is a finite set of states. Each state corresponds to a symbol in the alphabet Σ.
- p is the initial state probabilities.
- A is the state transition probabilities, denoted by a_{st} for each $s, t \in Q$.

For each $s, t \in Q$ the transition probability is:

$$a_{st} \equiv P(x_i = t | x_{i-1} = s) \tag{5.1}$$

[1] This scribe is partially based on Ophir Gvirtzer's and Zohar Ganon's scribe from Fall Semester 2000.

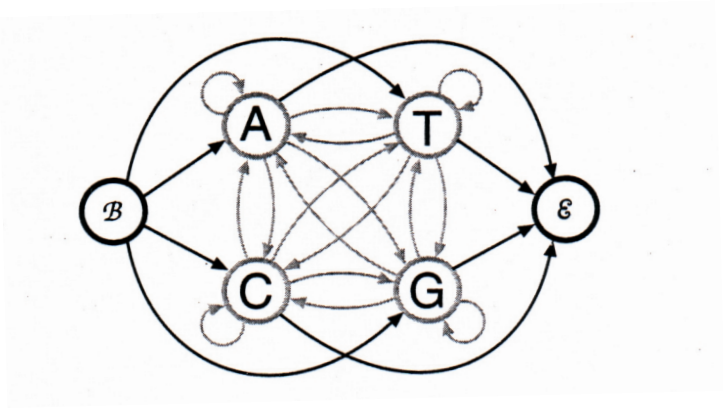

Figure 5.1: Source: [4]. A Markov chain for modeling a DNA sequence. B and ε are the begin and end states, respectively.

We assume that $X = (x_1, \ldots, x_L)$ is a random process with a memory of length 1, i.e., the value of the random variable x_i depends only on its predecessor x_{i-1}. Formally we can write:

$$\begin{aligned} \forall s_1, \ldots, s_i \in \Sigma \quad P(x_i = s_i | x_1 = s_1, \ldots, x_{i-1} = s_{i-1}) = \\ = P(x_i = s_i | x_{i-1} = s_{i-1}) = a_{s_{i-1}, s_i} \end{aligned} \tag{5.2}$$

The probability of the whole sequence X will therefore be:

$$P(X) = p(x_1) \cdot \prod_{i=2}^{L} a_{x_{i-1}, x_i} \tag{5.3}$$

We can simplify this formula by transforming the initial probability $p(x_1)$ into a transition probability. We can add fictitious *begin* and *end* states together with corresponding symbols x_0 and x_{L+1}. Then we can define $\forall_{s \in \Sigma}\, a_{0,s} \equiv p(s)$, where $p(s)$ is the initial probability of the symbol s. Hence:

$$P(X) = \prod_{i=1}^{L} a_{x_{i-1}, x_i} \tag{5.4}$$

Figure 5.1 shows a simple Markov chain for modeling DNA sequences.

+	A	C	G	T	-	A	C	G	T
A	0.180	0.274	0.426	0.120	A	0.300	0.205	0.285	0.210
C	0.171	0.368	0.274	0.188	C	0.322	0.298	0.078	0.302
G	0.161	0.339	0.375	0.125	G	0.248	0.246	0.298	0.208
T	0.079	0.355	0.384	0.182	T	0.177	0.239	0.292	0.292

Table 5.1: Source: [4]. Transition probabilities inside/outside a CpG island.

Problem 5.1 Identifying a CpG island.
INPUT: A short DNA sequence $X = (x_1, \ldots, x_L) \in \Sigma^*$ (where $\Sigma = \{\text{A,C,G,T}\}$).
QUESTION: Decide whether X is a CpG island.

We can use two Markov chain models to solve this problem : one for dealing with CpG islands (the '+' model) and the other for dealing with non CpG island (the '-' model).

Let a^+_{st} denote the transition probability of $s, t \in \Sigma$ inside a CpG island and let a^-_{st} denote the transition probability outside a CpG island (see table 5.1 for the values of these probabilities).

Therefore we can compute a logarithmic likelihood score for a sequence X by:

$$Score(X) = \log \frac{P(X|\text{CpG island})}{P(X|\text{non CpG island})} = \sum_{i=1}^{L} \log \frac{a^+_{x_{i-1},x_i}}{a^-_{x_{i-1},x_i}} \tag{5.5}$$

The higher this score, the more likely it is that X is a CpG Island.

Figure 5.2 shows the distribution of such scores, normalized by dividing by their length. We can see a reasonable discrimination between regions labelled CpG island and other regions.

Problem 5.2 Locating CpG islands in a DNA sequence.
INPUT: A long DNA sequence $X = (x_1, \ldots, x_L) \in \Sigma^*$.
QUESTION: Locate the CpG islands along X.

A naive approach for solving this problem will be to extract a sliding window $X^k = (x_{k+1}, \ldots, x_{k+\ell})$ of a given length ℓ (where $\ell \ll L$, usually several hundred bases long, and $1 \leq k \leq L - \ell$) from the sequence and calculate $Score(X^k)$ for each one of the resulting subsequences. Subsequences that receive positive scores are potential CpG islands.

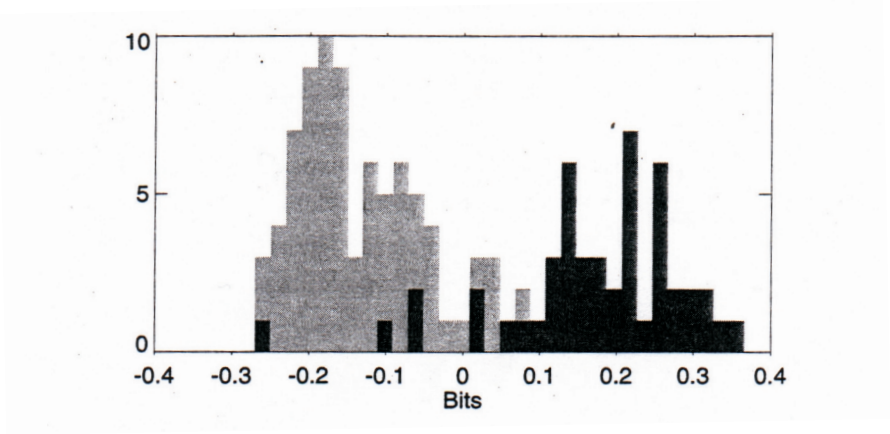

Figure 5.2: Source: [4]. The histogram of length-normalized scores for given sequences. CpG islands are shown with dark grey and non-CpG with light grey.

The main disadvantage in this algorithm is that we have no information about the lengths of the islands, while the algorithm suggested above assumes that those islands are at least ℓ nucleotides long. Should we use a value of ℓ which is too large, the CpG islands would be short substrings of our windows, and the score we give those windows may not be high enough. On the other hand, windows that are too small might not provide enough information to determine whether their bases are distributed like those of an island or not.

A better solution to this problem is to combine the two Markov chains of the last section into a unified model, with a small probability of switching from one chain to the other at each transition point. However, this introduces the complication that we now have two states corresponding to each nucleotide symbol. Therefore we need to use another model for this problem. This model will be described in the following section.

5.1.3 Hidden Markov Models

Definition A *Hidden Markov Model (HMM)* is a triplet $\mathcal{M} = (\Sigma, Q, \Theta)$, where:

- Σ is an alphabet of symbols.
- Q is a finite set of states, capable of emitting symbols from the alphabet Σ.
- Θ is a set of probabilities, comprised of:
  - *State transition probabilities*, denoted by a_{kl} for each $k, l \in Q$.
  - *Emission probabilities*, denoted by $e_k(b)$ for each $k \in Q$ and $b \in \Sigma$.

A *path* $\Pi = (\pi_1, \ldots, \pi_L)$ in the model $\mathcal{M}$ is a sequence of states. The path itself follows a simple Markov chain, so the probability of moving to a given state depends only on the previous state. As in the Markov chain model, we can define the *state transition probabilities* in terms of Π :

$$a_{kl} = P(\pi_i = l | \pi_{i-1} = k) \tag{5.6}$$

However, in a hidden Markov model there isn't a one-to-one correspondence between the states and the symbols. Therefore, in a HMM we introduce a new set of parameters, $e_k(b)$, called the emission probabilities. Given a sequence $X = (x_1, \ldots, x_L) \in \Sigma^*$ we define :

$$e_k(b) = P(x_i = b | \pi_i = k) \tag{5.7}$$

$e_k(b)$ is the probability that symbol b is seen when we are in state k.

The probability that the sequence X was generated by the model $\mathcal{M}$ given the path Π is therefore:

$$P(X, \Pi) = a_{\pi_0, \pi_1} \cdot \prod_{i=1}^{L} e_{\pi_i}(x_i) \cdot a_{\pi_i, \pi_{i+1}} \tag{5.8}$$

Where for our convenience we denote $\pi_0 = \textit{begin}$ and $\pi_{L+1} = \textit{end}$.

Example 5.1.3a An HMM for detecting CpG islands in a long DNA sequence

The model contains eight states corresponding to the four symbols of the alphabet Σ ={A,C,G,T}:

State:	A^+	C^+	G^+	T^+	A^-	C^-	G^-	T^-
Emitted Symbol:	A	C	G	T	A	C	G	T

$\pi_i \backslash \pi_{i+1}$	A^+	C^+	G^+	T^+	A^-	C^-	G^-	T^-
A^+	$0.180p$	$0.274p$	$0.426p$	$0.120p$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$
C^+	$0.171p$	$0.368p$	$0.274p$	$0.188p$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$
G^+	$0.161p$	$0.339p$	$0.375p$	$0.125p$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$
T^+	$0.079p$	$0.355p$	$0.384p$	$0.182p$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$	$\frac{1-p}{4}$
A^-	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$0.300q$	$0.205q$	$0.285q$	$0.210q$
C^-	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$0.322q$	$0.298q$	$0.078q$	$0.302q$
G^-	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$0.248q$	$0.246q$	$0.298q$	$0.208q$
T^-	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$\frac{1-q}{4}$	$0.177q$	$0.239q$	$0.292q$	$0.292q$

Table 5.2: Transition probabilities $a_{\pi_i,\pi_{i+1}}$ in the CpG island HMM.

If the probability for staying in a CpG island is p and the probability of staying outside it is q, then the transition probabilities will be as described in table 5.2 (derived from the transition probabilities given in table 5.1 under the assumption that we lose memory when moving from/into a CpG island, and that we ignore background probabilities).

In this special case the emission probability of each state X^+ or X^- is exactly 1 for the symbol X and 0 for any other symbol.

Let us consider another example, where the emission probabilities will not be degenerate.

Example 5.1.3b An HMM for modeling a dishonest casino dealer

Suppose a dealer in a casino rolls a die. The dealer use a fair die most of the time, but occasionally he switches to a loaded die. The loaded die has probability 0.5 of a six and probability 0.1 for the numbers 1 to 5. We also know that the dealer does not tend to change dies - he switches from a fair to a loaded die with probability 0.05, and the probability of switching back is 0.1. Given a sequence of die rolls we wish to determine when did the dealer use a fair die and when did he use a loaded die.

The corresponding HMM is:

- The states are $Q = \{F, L\}$, where F stands for "fair" and L for "loaded".
- The alphabet is $\Sigma = \{1, 2, 3, 4, 5, 6\}$.

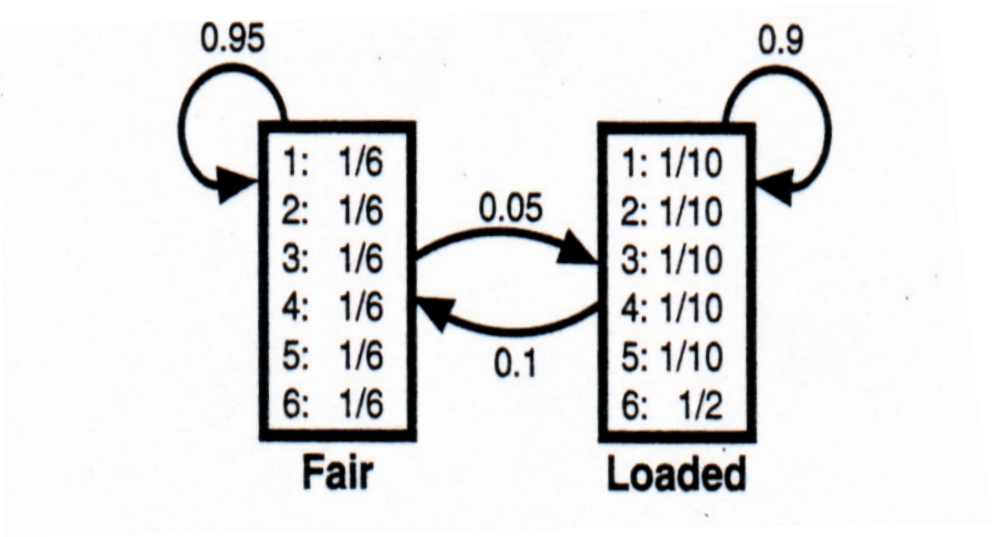

Figure 5.3: Source: [4]. HMM for the dishonest casino problem.

- The probabilities are:

$$a_{FF} = 0.95 \quad a_{FL} = 0.05 \tag{5.9}$$
$$a_{LL} = 0.9 \quad a_{LF} = 0.1 \tag{5.10}$$

$$\forall_{1 \le i \le 6}\, e_F(i) = \frac{1}{6} \tag{5.11}$$
$$\forall_{1 \le i \le 5}\, e_L(i) = 0.1 \qquad e_L(6) = 0.5 \tag{5.12}$$

Figure 5.3 gives a full description of the model.

Returning to the general case, we have defined the probability $P(X, \Pi)$ for a given sequence X and a given path Π. However, we do not know the actual sequence of states $(\pi_1, \ldots, \pi_L)$ that emitted $(x_1, \ldots, x_L)$. We therefore say that the generating path of X is *hidden*.

Problem 5.3 The decoding problem.
INPUT: A hidden Markov model $\mathcal{M} = (\Sigma, Q, \Theta)$ and a sequence $X \in \Sigma^*$, for which the generating path $\Pi = (\pi_1, \ldots, \pi_L)$ is unknown.
QUESTION: Find the most probable generating path Π^* for X, i.e., a path such that $P(X, \Pi^*)$ is maximized. We denote this also by:

$$\Pi^* = \arg\max_{\Pi}\{P(X, \Pi)\}$$

In the CpG islands case (problem 5.2), the optimal path can help us find the location of the islands. Had we known Π^*, we could have traversed it determining that all the parts that pass through the "+" states are CpG islands.

Similarly, in the dishonest casino case (example 5.1.3b), the parts of Π^* that pass through the L (loaded) state are suspected rolls of the loaded die.

A solution for the most probable path is described in the following section.

5.1.4 Viterbi Algorithm

We can calculate the most probable path in a hidden Markov model using a dynamic programming algorithm. This algorithm is widely known as *Viterbi Algorithm*. Viterbi [10] devised this algorithm for the decoding problem, even though its more general description was originally given by Bellman [3].

Let X be a sequence of length L. For $k \in Q$ and $0 \leq i \leq L$, we consider a path Π ending at k, and the probability of Π generating the prefix $(x_1, \ldots, x_i)$ of X. Denote by $v_k(i)$ the probability of the most probable path for the prefix $(x_1, \ldots, x_i)$ that ends in state k.

$$v_k(i) = \max_{\{\Pi | \Pi_i = k\}} P(x_1, \ldots, x_i, \Pi) \tag{5.13}$$

1. Initialize:

$$v_{begin}(0) = 1 \tag{5.14}$$
$$\forall_{k \neq begin} \quad v_k(0) = 0 \tag{5.15}$$

2. For each $i = 0, \ldots, L-1$ and for each $l \in Q$ recursively calculate:

$$v_l(i+1) = e_l(x_{i+1}) \cdot \max_{k \in Q} \{v_k(i) \cdot a_{kl}\} \tag{5.16}$$

3. Finally, the value of $P(X, \Pi^*)$ is:

$$P(X, \Pi^*) = \max_{k \in Q} \{v_k(L) \cdot a_{k,end}\} \tag{5.17}$$

We can reconstruct the path Π^* itself by keeping back pointers during the recursive stage and tracing them.

Complexity: We calculate the values of $O(|Q| \cdot L)$ cells of the matrix V, spending $O(|Q|)$ operations per cell. The overall time complexity is therefore $O(L \cdot |Q|^2)$ and the space complexity is $O(L \cdot |Q|)$.

Since we are dealing with probabilities, the extensive multiplication operations we perform may result in an underflow. This can be avoided if we choose to work with logarithmic scores, which will make the products become sums and the numbers stay reasonable. We can therefore define $v_k(i)$ to be the logarithmic score of the most probable path for the prefix $(x_1, \ldots, x_i)$ that ends in the state k.

We shall initialize:

$$v_{begin}(0) = 0 \tag{5.18}$$

$$\forall_{k \neq begin} \quad v_k(0) = -\infty \tag{5.19}$$

The recursion will look like:

$$v_l(i+1) = \log e_l(x_{i+1}) + \max_{k \in Q} \{v_k(i) + \log(a_{kl})\} \tag{5.20}$$

Finally, the score for the best path Π^* is:

$$Score(X, \Pi^*) = \max_{k \in Q} \{v_k(L) + \log(a_{k,end})\} \tag{5.21}$$

Figure 5.4 shows the results of running the Viterbi algorithm on the dishonest casino example.

5.1.5 Posterior Decoding

Problem 5.4 The posterior decoding problem.
INPUT: A hidden Markov model $\mathcal{M} = (\Sigma, Q, \Theta)$ and a sequence $X \in \Sigma^*$, for which the generating path $\Pi = (\pi_1, \ldots, \pi_L)$ is unknown.
QUESTION: For each $1 \leq i \leq L$ and $k \in Q$, compute the probability $P(\pi_i = k|X)$.

For this we shall need some extra definitions.

Forward algorithm: Given a sequence $X = (x_1, \ldots, x_L)$ let us denote by $f_k(i)$ the probability of emitting the prefix $(x_1, \ldots, x_i)$ and eventually reaching state $\pi_i = k$:

$$f_k(i) = P(x_1, \ldots, x_i, \pi_i = k) \tag{5.22}$$

Figure 5.4: Source: [4]. The numbers show 300 rolls of a die. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that, the prediction by the Viterbi algorithm is shown.

We use the same initial values for $f_k(0)$ as was done in the Viterbi algorithm:

$$f_{begin}(0) = 1 \tag{5.23}$$
$$\forall_{k \neq begin} \quad f_k(0) = 0 \tag{5.24}$$

In analogy to 5.16 we can use the recursive formula:

$$f_l(i+1) = e_l(x_{i+1}) \cdot \sum_{k \in Q} f_k(i) \cdot a_{kl} \tag{5.25}$$

We terminate the process by calculating:

$$P(X) = \sum_{k \in Q} f_k(L) \cdot a_{k,end} \tag{5.26}$$

Backward algorithm: In a complementary manner we denote by $b_k(i)$ the probability of the suffix $(x_{i+1}, \ldots, x_L)$ given $\pi_i = k$:

$$b_k(i) = P(x_{i+1}, \ldots, x_L, \pi_i = k) \tag{5.27}$$

In this case, we initialize:

$$\forall_{k \in Q} \quad b_k(L) = a_{k,end} \tag{5.28}$$

The recursive formula is:

$$b_k(i) = \sum_{l \in Q} a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1) \tag{5.29}$$

We terminate the process by calculating:

$$P(X) = \sum_{l \in Q} a_{begin,l} \cdot e_l(x_1) \cdot b_l(1) \tag{5.30}$$

Complexity: All the values of $f_k(i)$ and $b_k(i)$ can be calculated in $O(L \cdot |Q|^2)$ time and stored in $O(L \cdot |Q|)$ space, as it is the case with Viterbi algorithm.

Unlike in the Viterbi algorithm, here we cannot trivially use the logarithmic weights, since we do not perform only multiplication of probabilities, but we also sum probabilities. We can solve this problem by using the exponent function. We can calculate a logarithm of

a sum of probabilities from the logarithms of the probabilities in the following way :

$$\log(p+q) = \log(\exp(\log p) + \exp(\log q)) \tag{5.31}$$

Formally, let's define $f_k(i)$ as the logarithm of the probability of emitting the prefix $(x_1, \ldots, x_i)$ given $\pi_i = k$.

We shall initialize:

$$f_{begin}(0) = 0 \tag{5.32}$$
$$\forall_{k \neq begin} \quad f_k(0) = -\infty \tag{5.33}$$

The recursion is :

$$f_l(i+1) = \log e_l(x_{i+1}) + \log \sum_{k \in Q} [a_{kl} \cdot \exp(f_k(i))] \tag{5.34}$$

And we terminate the process by calculating:

$$P(X) = \log \sum_{k \in Q} [a_{k,end} \cdot \exp(f_k(L))] \tag{5.35}$$

Using the forward and backward probabilities we can compute the value of $P(\pi_i = k|X)$. Since the process X has memory of only length 1, there is a dependency only on the last state, so we can write:

$$\begin{aligned} P(X, \pi_i = k) &= P(x_1, \ldots, x_i, \pi_i = k) \cdot P(x_{i+1}, \ldots, x_L | x_1, \ldots, x_i, \pi_i = k) = \\ &= P(x_1, \ldots, x_i, \pi_i = k) \cdot P(x_{i+1}, \ldots, x_L | \pi_i = k) = \\ &= f_k(i) \cdot b_k(i) \end{aligned} \tag{5.36}$$

Using the definition of conditional probability, we obtain the solution to the posterior decoding problem:

$$P(\pi_i = k|X) = \frac{P(X, \pi_i = k)}{P(X)} = \frac{f_k(i) \cdot b_k(i)}{P(X)} \tag{5.37}$$

where P(x) is the result of the forward or backward calculation. For example, using the backward calculation we get :

$$P(X) = \sum_{l \in Q} a_{begin,l} \cdot e_l(x_1) \cdot b_l(1) \tag{5.38}$$

Figure 5.5 shows the posterior probability of being in the state corresponding to the fair die in the dishonest casino example.

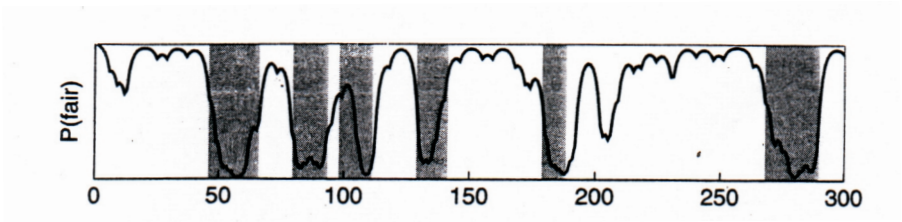

Figure 5.5: Source: [4]. The posterior probability for the die being fair in the dishonest casino example. The x axis shows the number of the roll. The shaded areas show when the roll was generated by the loaded die.

Uses for Posterior Decoding

A major use of the posterior probabilities $P(\pi_i = k|X)$ is for two alternative forms of decoding in addition to the Viterbi decoding we introduced in the previous section. These are particularly useful when many different paths have almost the same probability as the most probable one, because then we may want to consider other possible paths as well.

The first approach is to define an alternative path Π^{**}, comprised of the state sequence $\{k_i\}$ in which each state k_i has the highest probability to emit x_i at step i. We denote this by :

$$\Pi^{**} = \arg\max_k \{P(\Pi_i = k|X)\}$$

We should notice that this may not be a legitimate path, if some transitions are not permitted.

The second decoding approach arises when we're not interested in the state sequence itself, but in some other property derived from it. We can define a function $g(k)$ on the states, and then look at the value :

$$G(i|X) = \sum_k \{P(\Pi_i = k|X)g(k)\}$$

In the special case where $g(k)$ takes the value 1 for a subset S of the states and 0 for the rest, $G(i|X)$ is the posterior probability that a state in S emitted the symbol x_i. For example in the CpG island model, we can use this approach to calculate the posterior probability of each nucleotide to be in a CpG island.

5.1.6 Parameter Estimation for HMMs

In examples 5.1.3a and 5.1.3b we constructed hidden Markov models knowing the transition and emission probabilities for the problems we had to solve. In real life, this may not be the

case. We may be given n example sequences $X^{(1)}, \ldots, X^{(n)} \in \Sigma^*$ of length $L^{(1)}, \ldots, L^{(n)}$, respectively, which were all generated from the HMM $\mathcal{M} = (\Sigma, Q, \Theta)$. The values of the probabilities in Θ, however, are unknown a-priori.

In order to construct the HMM that will best characterize $X^{(1)}, \ldots, X^{(n)}$, we need to assign values to Θ that will maximize the probabilities of our sequences according to the model. Since all sequences are assumed to be generated independently, we can write:

$$P(X^{(1)}, \ldots, X^{(n)}|\Theta) = \prod_{j=1}^{n} P(X^{(i)}|\Theta) \tag{5.39}$$

Using the logarithmic score, our goal is to find Θ^* such that

$$\Theta^* = \arg\max_{\Theta}\{Score(X^{(1)}, \ldots, X^{(n)}|\Theta)\} \tag{5.40}$$

where:

$$Score(X^{(1)}, \ldots, X^{(n)}|\Theta) = \log P(X^{(1)}, \ldots, X^{(n)}|\Theta) = \sum_{j=1}^{n} \log(P(X^{(i)}|\Theta)) \tag{5.41}$$

The sequences $X^{(1)}, \ldots, X^{(n)}$ are usually called the *training sequences*.

We shall examine two cases for parameter estimation :
(1) Estimation when the state sequence is known.
(2) Estimation when the state sequence is unknown.

Estimation when the state sequence is known

In this case we know the state sequences $\Pi^{(1)}, \ldots, \Pi^{(n)}$ corresponding to $X^{(1)}, \ldots, X^{(n)}$, respectively. For example, in our CpG island model, this could be the case if we were given a set of genomic sequences in which the CpG islands were already labelled, based on experimental data.

In this case, we can scan the given sequences and compute:

- A_{kl} - the number of transitions from the state k to l.
- $E_k(b)$ - the number of times that an emission of the symbol b occurred in state k.

The maximum likelihood estimators will be:

$$a_{kl} = \frac{A_{kl}}{\sum_{q \in Q} A_{kq}} \tag{5.42}$$

$$e_k(b) = \frac{E_k(b)}{\sum_{\sigma \in \Sigma} E_k(\sigma)} \tag{5.43}$$

To avoid zero probabilities, when working with a small amount of samples, it is recommended to work with A'_{kl} and $E'_k(b)$, where:

$$A'_{kl} = A_{kl} + r_{kl} \tag{5.44}$$
$$E'_k(b) = E_k(b) + r_k(b) \tag{5.45}$$

Usually the *Laplace correction*, where all r_{kl} and $r_k(b)$ values equal 1, is applied, having an intuitive interpretation of a-priori assumed uniform distribution. However, it may be beneficial in some cases to use other values for the correction (e.g. when having some prior information about the transition or emission probabilities).

Estimation when state sequence is unknown : Baum-Welch training

Usually, the state sequences $\Pi^{(1)}, \ldots, \Pi^{(n)}$ are not known. In this case, the problem of finding the optimal set of parameters Θ^* is known to be NP-complete. The *Baum-Welch algorithm* [2], which is a special case of the *EM technique (Expectation and Maximization)*, can be used for heuristically finding a solution to the problem.

1. *Initialization*: Assign arbitrary values to Θ.
2. *Expectation*:
   (a) Compute the expected number of state transitions from state k to state l. Using the same arguments we used for computing $P(X, \pi_i = k)$ (see 5.36), we get:

$$P(\pi_i = k, \pi_{i+1} = l | X, \Theta) = \frac{f_k(i) \cdot a_{kl} \cdot e_l(x_{i+1}) \cdot b_l(i+1)}{P(X)} \tag{5.46}$$

   Let $\{f_k^{(j)}(i), b_k^{(j)}(i)\}_{k \in Q, i \leq L(j)}$ denote the forward and backward probabilities of the string $X^{(j)}$. Then the expectation is :

$$A_{kl} = \sum_{j=1}^{n} \frac{1}{P(X^{(j)})} \cdot \sum_{i=1}^{L(j)} f_k^{(j)}(i) \cdot a_{kl} \cdot e_l(x_{i+1}^{(j)}) \cdot b_l^{(j)}(i+1) \tag{5.47}$$

(b) Compute the expected number of emissions of the symbol b that occurred at the state k (using the value of $P(\pi_i = k|X)$ as calculated in 5.37):

$$E_k(b) = \sum_{j=1}^{n} \frac{1}{P(X^{(j)})} \cdot \sum_{\{i|x_i^{(j)}=b\}} f_k^{(j)}(i) \cdot b_k^{(j)}(i) \tag{5.48}$$

3. *Maximization*: Re-compute the new values for Θ from A_{kl} and $E_k(b)$, as explained above (in case 1).

4. Repeat steps 2 and 3 until the improvement of $Score(X^{(1)}, \ldots, X^{(n)}|\Theta)$ is less then a given parameter ϵ.

The EM-algorithm guarantees that the target function values $Score(X^{(1)}, \ldots, X^{(n)}|\Theta)$ are monotonically increasing, and as logarithms of probabilities are certainly bounded by 0, the algorithm is guaranteed to converge. It is important to notice that the convergence is of the target function and not in the Θ space: the values of Θ may change drastically even for almost equal values of the target function, which may imply that the obtained solution is not stable.

The main problem with the Baum-Welch algorithm is that there may exist several local maxima of the target function and it is not guaranteed that we reach the global maximum: the convergence may lead to a local maximum. A useful way to circumvent this pitfall is to run the algorithm several times, each time with different initial values for Θ. If we reach the same maximum most of the times, it is highly probable that this is indeed the global maximum. Another way is to start with Θ values that are meaningful, like in the case of the CpG islands we might start from Θ values obtained from a real case statistics.

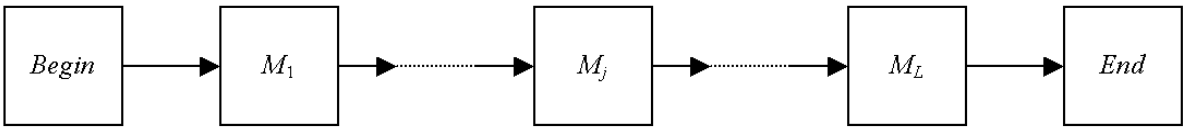

Figure 5.6: Match states in a profile HMM.

5.2 Profile Alignment

5.2.1 Profile HMMs

HMMs can be used for aligning a string versus a given profile, thus helping us to solve the multiple alignment problem.

Ungapped Profile Alignment

We define a profile $\mathcal{P}$ of length L, as a set of probabilities, consisting of, for each $b \in \Sigma$ and $1 \leq i \leq L$, the probability $e_i(b)$ of observing the symbol b at the i^{th} position. In such a case the probability of a string $X = (x_1, \ldots, x_L)$ given the profile $\mathcal{P}$ will be:

$$P(X|\mathcal{P}) = \prod_{i=1}^{L} e_i(x_i) \tag{5.49}$$

We can calculate a likelihood score for the ungapped alignment of X against the profile $\mathcal{P}$:

$$Score(X|\mathcal{P}) = \sum_{i=1}^{L} \log \frac{e_i(x_i)}{p(x_i)} \tag{5.50}$$

where $p(b)$ is the background frequency of occurrences of the symbol b.

This leads to a definition of the following HMM: all the states are *match states* $M_1, \ldots, M_L$ which correspond to matches of the string's symbols with the profile positions. All these states are sequentially linked (i.e., each match state M_j is linked to its successor M_{j+1}) as shown in Figure 5.6. The emission probability of the symbol b from the state M_j is of course $e_j(b)$, and the transition probability between two match states is 1. Alignment to this profile HMM is trivial, because there is no choice of transitions.

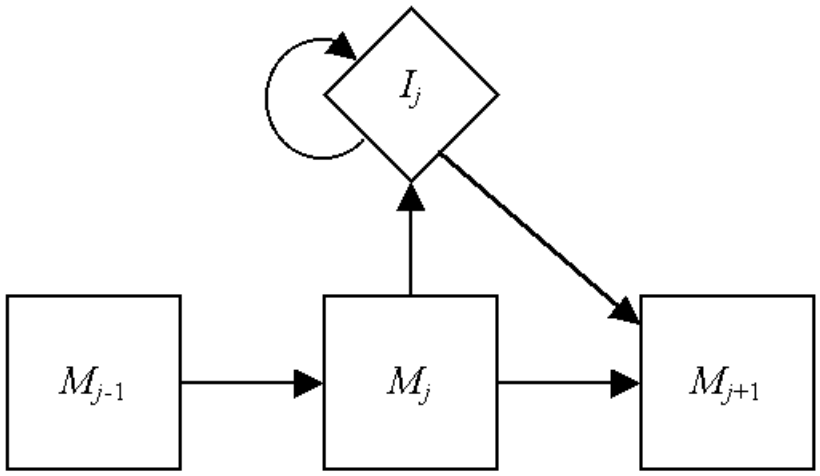

Figure 5.7: A profile HMM with an insertion state (and some match states).

Adding insert and delete states

To allow insertions, we will add also *insertion states* $I_0, \ldots, I_L$ to the model. We shall assume that:

$$\forall_{b \in \Sigma} \quad e_{I_j}(b) = p(b)$$

Each insertion state I_j has a link entering from the corresponding match state M_j, a leaving link towards the next match state M_{j+1} and also has a self-loop (see figure 5.7). Assigning the appropriate probabilities for those transitions corresponds to the application of affine gap penalties, since the overall contribution of a gap of length h to the logarithmic likelihood score is:

$$\underbrace{\log(a_{M_j I_j}) + \log(a_{I_j M_{j+1}})}_{\text{gap creation}} + \underbrace{(h-1) \cdot \log(a_{I_j I_j})}_{\text{gap extension}}$$

To allow deletions as well, we add the *deletion states* $D_1, \ldots, D_L$. These states cannot emit any symbol and are therefore called *silent* (Note that the *begin*/*end* states are silent as well). The deletion states are sequentially linked, in a similar manner to the match states and they are also interleaved with the match states (see figure 5.8).

The cost of a deletion is analogous to the cost of an insertion, although the path through the model looks different. Theoritically, it is possible that the D $\rightarrow$ D transitions for one

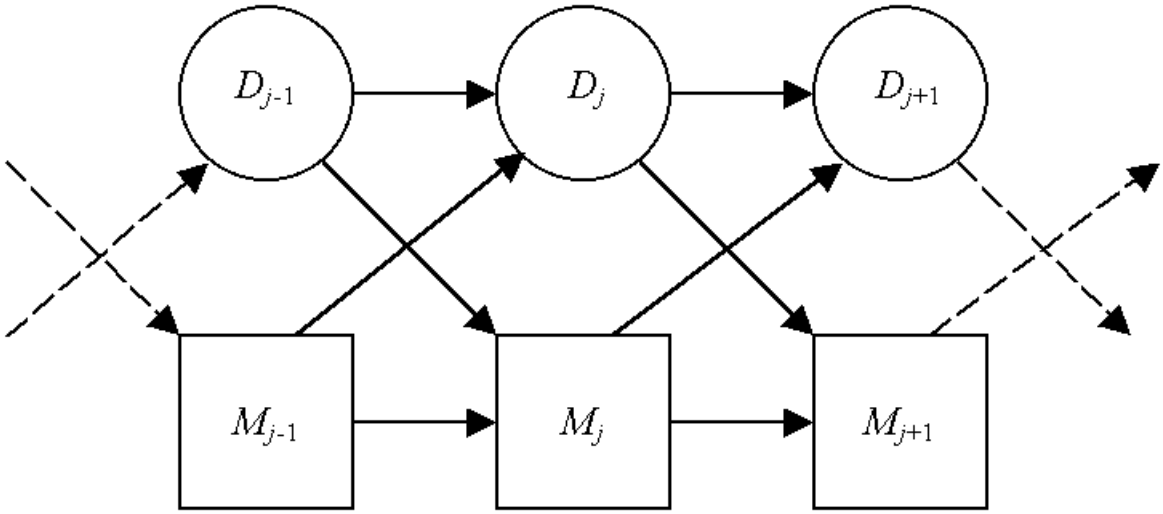

Figure 5.8: Profile HMM with deletion and match states.

deletion will have different probabilities, and thus contribute differently to the score, whereas all the I $\rightarrow$ I transitions for one insertion involve the same state, and thus have the same cost.

The full HMM for modeling the profile $\mathcal{P}$ of length L is comprised of L layers, each layer has three states M_j, I_j and D_j. To complete the model, we add *begin* and *end* states, connected to the layers as shown in figure 5.9. This model, which we will call a profile HMM, was first introduced by Haussler et al [5].

We have added transitions between insertion and deletion states, as in the original model, although these are usually very improbable. Other models, that don't contain these transitions, are equivalent to the model described above.

5.2.2 Deriving Profile HMMs From Multiple Alignments

Before we can use the profile HMM, we need to estimate the emission and transition probabilities of the model. In case the HMM is derived from a given multiple alignment of sequences, we can regard the alignment as providing a set of independent samples of alignments of sequences to our HMM. Since the alignments are given, we can estimate the model parameters using the following equations from section 5.1.6:

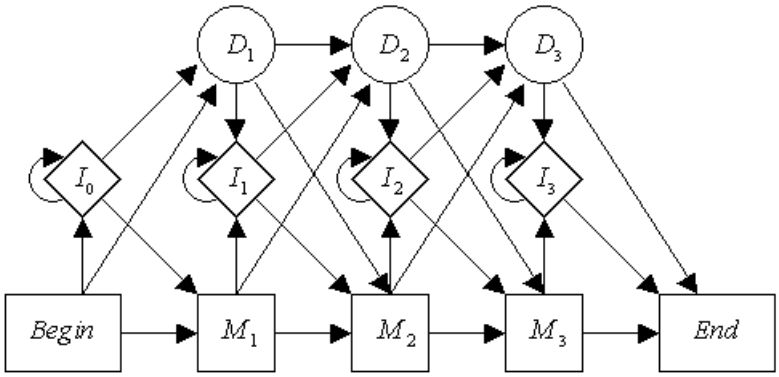

Figure 5.9: Source: [4]. Profile HMM for global alignment.

$$a_{kl} = \frac{A_{kl}}{\sum_{q \in Q} A_{kq}} \tag{5.51}$$

$$e_k(b) = \frac{E_k(b)}{\sum_{\sigma \in \Sigma} E_k(\sigma)} \tag{5.52}$$

If we have a large number of sequences in our training alignment, this will give us an accurate and consistent estimate of the parameters. However, it has problems when there are only a few sequences. A major difficulty is that some transitions or emissions may not be seen in the training alignment, so we would acquire zero probability. As before, we can use Laplace's correction (adding one for each frequency) or other approaches to avoid zero probabilities.

Figure 5.11 shows an example for estimating the probability parameters in a profile HMM based on the multiple alignment given in Figure 5.10, using Laplace's rule. For example, in column 1 of the alignment there are six transitions from match to match, one transition to a delete state and no insertions. Thus we obtain $a_{M_1M_2} = \frac{7}{10}, a_{M_1D_2} = \frac{2}{10}, a_{M_1I_1} = \frac{1}{10}$, and the estimation for the emission probabilities for M_1 are : $e_{M_1}(V) = \frac{6}{27}, e_{M_1}(F) = \frac{2}{27}$, and $e_{M_1}(a) = \frac{1}{27}$ for each a other than V, I, F.

```
HBA_HUMAN    ...VGA--HAGEY...
HBB_HUMAN    ...V----NVDEV...
MYG_PHYCA    ...VEA--DVAGH...
GLB3_CHITP   ...VKG------D...
GLB5_PETMA   ...VYS--TYETS...
LGB2_LUPLU   ...FNA--NIPKH...
GLB1_GLYDI   ...IAGADNGAGV...
                  ***   *****
```

Figure 5.10: Source: [4]. Ten columns from the multiple alignment of seven globin protein sequences. The starred columns are ones that will be treated as 'matches' in the profile HMM.

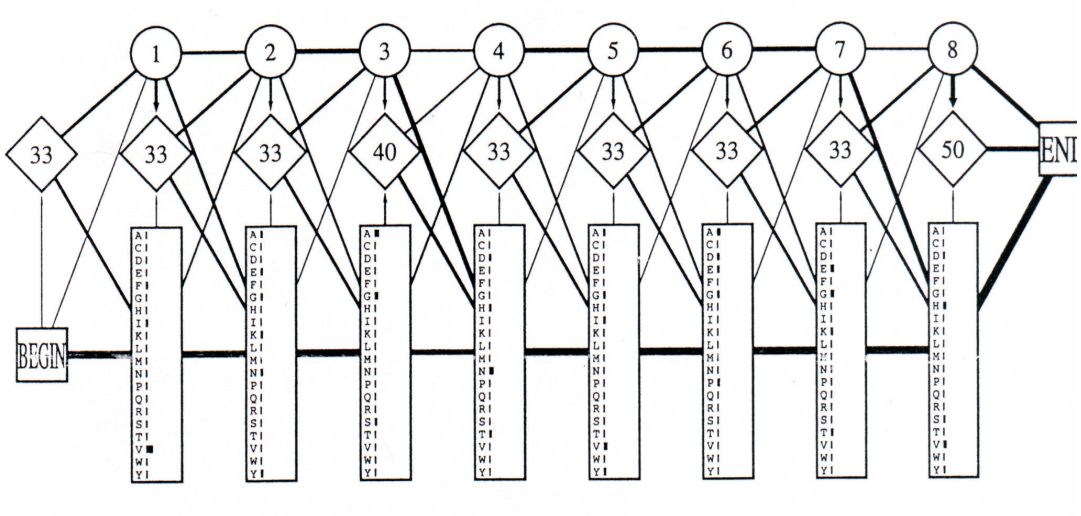

Figure 5.11: Source: [4]. A hidden Markov model derived from the small alignment shown in the above figure using Laplace's rule. Emission probabilities are shown as bars opposite the different amino acids for each match state, and transition probabilities are indicated by the thickness of the lines. The $I \rightarrow I$ transition probabilities are shown as percentages in the insert states.

5.2.3 Aligning Sequences to a Profile HMM

To align the string $X = (x_1, \ldots, x_m)$ against a profile $\mathcal{P}$ of length L, we will use a variant of the Viterbi algorithm. For each $1 \leq j \leq L$ and $1 \leq i \leq m$ we use the following definitions:

- Let $v_j^M(i)$ be the logarithmic likelihood score of the best path for matching $X = (x_1, \ldots, x_i)$ to the profile HMM $\mathcal{P}$, ending with x_i emitted by the state M_j.
- Let $v_j^I(i)$ be the logarithmic likelihood score of the best path for matching $X = (x_1, \ldots, x_i)$ to the profile HMM $\mathcal{P}$, ending with x_i emitted by the state I_j.
- Let $v_j^D(i)$ be the logarithmic likelihood score of the best path for matching $X = (x_1, \ldots, x_i)$ to the profile HMM $\mathcal{P}$, ending with the state D_j (without emitting any symbol).

The initial value of the special *begin* state is:

$$v_{begin}(0) = 0 \tag{5.53}$$

To calculate the values of $v_j^M(i)$, $v_j^I(i)$ and $v_j^D(i)$ we use the same technique as in the Viterbi algorithm. There are however two major differences:

- Each state in the model has at most three entering links (see figure 5.9).
- The deletion states are silent - they cannot emit any symbol.

The three predecessors of the match state M_j are the three states of the previous layer, $j-1$:

$$v_j^M(i) = \log \frac{e_{M_j}(x_i)}{p(x_i)} + \max \begin{cases} v_{j-1}^M(i-1) + \log(a_{M_{j-1},M_j}) \\ v_{j-1}^I(i-1) + \log(a_{I_{j-1},M_j}) \\ v_{j-1}^D(i-1) + \log(a_{D_{j-1},M_j}) \end{cases} \tag{5.54}$$

The three predecessors of the insertion state I_j are the three states of the same layer, j:

$$v_j^I(i) = \log \frac{e_{I_j}(x_i)}{p(x_i)} + \max \begin{cases} v_j^M(i-1) + \log(a_{M_j,I_j}) \\ v_j^I(i-1) + \log(a_{I_j,I_j}) \\ v_j^D(i-1) + \log(a_{D_j,I_j}) \end{cases} \tag{5.55}$$

The three predecessors of the deletion state D_j are the three states of the layer $j-1$. Since D_j is a silent state, we should not consider the emission likelihood score for x_i in this case:

$$v_j^D(i) = \max \begin{cases} v_{j-1}^M(i) + \log(a_{M_{j-1},D_j}) \\ v_{j-1}^I(i) + \log(a_{I_{j-1},D_j}) \\ v_{j-1}^D(i) + \log(a_{D_{j-1},D_j}) \end{cases} \tag{5.56}$$

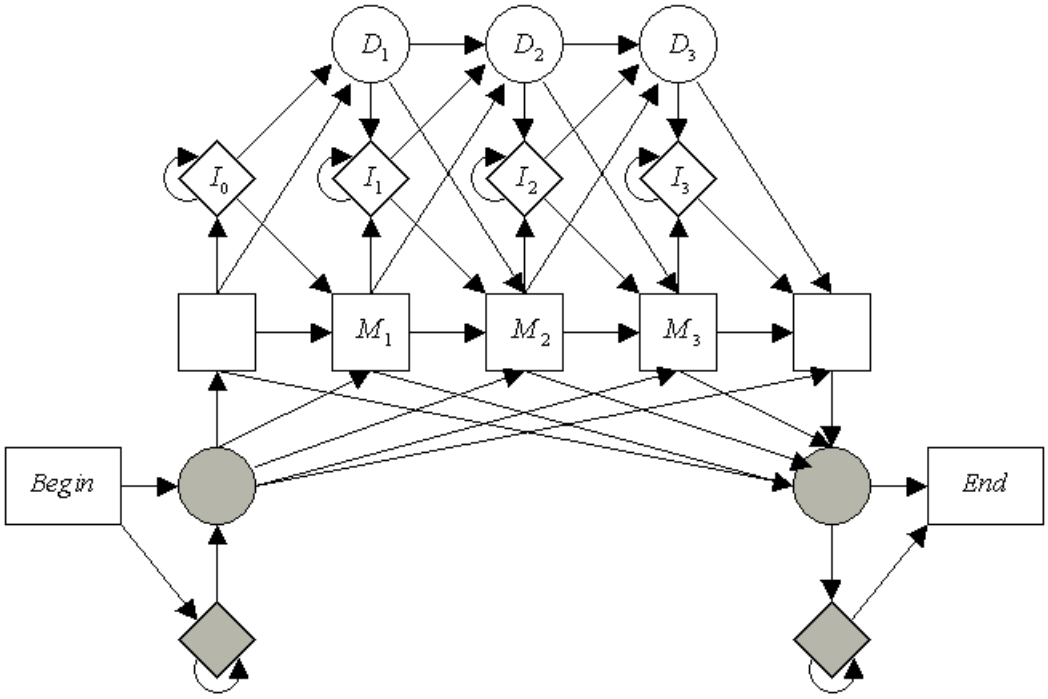

Figure 5.12: Source: [4]. Profile HMM for local alignment.

We conclude by calculating the optimal score:

$$Score(X|\Pi^*) = \max \begin{cases} v_L^M(m) + \log(a_{M_L,end}) \\ v_L^I(m) + \log(a_{I_L,end}) \\ v_L^D(m) + \log(a_{D_L,end}) \end{cases} \tag{5.57}$$

Complexity: We have to calculate $O(L \cdot m)$ values, each taking $O(1)$ operations (since we only need to consider the scores of at most three predecessors). We therefore need $O(L \cdot m)$ time and $O(L \cdot m)$ space.

We can use a similar approach for the problem of local alignment of a sequence versus a profile HMM. This is achieved by adding four additional states (the lightly shaded states in Figure 5.12) corresponding to the alignment of a subsequence of X to a part of the profile.

5.2.4 Forward and Backward Probabilities for a Profile HMM

In the previous section we modelled the problem of aligning a string to a profile. As with general HMMs, the main problem is to assign meaningful values to the transition and emis-

sion probabilities to a profile HMM. It is possible to use the Baum-Welch algorithm for training the model probabilities, but it first has to be shown how to compute the forward and backward probabilities needed for the algorithm.

Given a string $X = (x_1, \ldots, x_m)$ we define:

- The forward probabilities:

$$f_j^M(i) = P(x_1, \ldots, x_i \text{ ending at } M_j) \tag{5.58}$$
$$f_j^I(i) = P(x_1, \ldots, x_i \text{ ending at } I_j) \tag{5.59}$$
$$f_j^D(i) = P(x_1, \ldots, x_i \text{ ending at } D_j) \tag{5.60}$$

- The backward probabilities:

$$b_j^M(i) = P(x_{i+1}, \ldots, x_m \text{ beginning from } M_j) \tag{5.61}$$
$$b_j^I(i) = P(x_{i+1}, \ldots, x_m \text{ beginning from } I_j) \tag{5.62}$$
$$b_j^D(i) = P(x_{i+1}, \ldots, x_m \text{ beginning from } D_j) \tag{5.63}$$

Computing the Forward Probabilities:

1. Initialization:

$$f_{begin}(0) = 1 \tag{5.64}$$

2. Recursion:

$$\begin{aligned} f_j^M(i) = e_{M_j}(x_i) \cdot [&f_{j-1}^M(i-1) \cdot a_{M_{j-1},M_j} + \\ &f_{j-1}^I(i-1) \cdot a_{I_{j-1},M_j} + \\ &f_{j-1}^D(i-1) \cdot a_{D_{j-1},M_j}] \end{aligned} \tag{5.65}$$

$$\begin{aligned} f_j^I(i) = e_{I_j}(x_i) \cdot [&f_j^M(i-1) \cdot a_{M_j,I_j} + \\ &f_j^I(i-1) \cdot a_{I_j,I_j} + \\ &f_j^D(i-1) \cdot a_{D_j,I_j}] \end{aligned} \tag{5.66}$$

$$\begin{aligned} f_j^D(i) = &f_{j-1}^M(i) \cdot a_{M_{j-1},D_j} + \\ &f_{j-1}^I(i) \cdot a_{I_{j-1},D_j} + \\ &f_{j-1}^D(i) \cdot a_{D_{j-1},D_j} \end{aligned} \tag{5.67}$$

Computing the Backward Probabilities:

1. Initialization:

$$b_L^M(m) = a_{M_L,end} \tag{5.68}$$

$$b_L^I(m) = a_{I_L,end} \tag{5.69}$$

$$b_L^D(m) = a_{D_L,end} \tag{5.70}$$

2. Recursion:

$$\begin{aligned} b_j^M(i) = {} & b_{j+1}^M(i+1) \cdot a_{M_j,M_{j+1}} \cdot e_{M_{j+1}}(x_{i+1}) + \\ & b_j^I(i+1) \cdot a_{M_j,I_j} \cdot e_{I_j}(x_{i+1}) + \\ & b_{j+1}^D(i) \cdot a_{M_j,D_{j+1}} \end{aligned} \tag{5.71}$$

$$\begin{aligned} b_j^I(i) = {} & b_{j+1}^M(i+1) \cdot a_{I_j,M_{j+1}} \cdot e_{M_{j+1}}(x_{i+1}) + \\ & b_j^I(i+1) \cdot a_{I_j,I_j} \cdot e_{I_j}(x_{i+1}) + \\ & b_{j+1}^D(i) \cdot a_{I_j,D_{j+1}} \end{aligned} \tag{5.72}$$

$$\begin{aligned} b_j^D(i) = {} & b_{j+1}^M(i+1) \cdot a_{D_j,M_{j+1}} \cdot e_{M_{j+1}}(x_{i+1}) + \\ & b_j^I(i+1) \cdot a_{D_j,I_j} \cdot e_{I_j}(x_{i+1}) + \\ & b_{j+1}^D(i) \cdot a_{D_j,D_{j+1}} \end{aligned} \tag{5.73}$$

Again, if we want to transform to the log space, we can do it in the following way :

Computing the Forward Logarithmic Probabilities:

1. Initialization:

$$f_{begin}(0) = 0 \tag{5.74}$$

2. Recursion:

$$\begin{aligned} f_j^M(i) = log(e_{M_j}(X_i)) + log[& a_{M_{j-1},M_j} \cdot exp(f_{j-1}^M(i-1)) + \\ & a_{I_{j-1},M_j} \cdot exp(f_{j-1}^I(i-1)) + \\ & a_{D_{j-1},M_j} \cdot exp(f_{j-1}^D(i-1))] \end{aligned} \tag{5.75}$$

$$f_j^I(i) = log(e_{I_j}(X_i)) + log[a_{M_{j-1},I_j} \cdot exp(f_{j-1}^M(i-1)) + a_{I_{j-1},I_j} \cdot exp(f_{j-1}^I(i-1)) + a_{D_{j-1},I_j} \cdot exp(f_{j-1}^D(i-1))] \tag{5.76}$$

$$f_j^D(i) = log[a_{M_{j-1},D_j} \cdot exp(f_{j-1}^M(i-1)) + a_{I_{j-1},D_j} \cdot exp(f_{j-1}^I(i-1)) + a_{D_{j-1},D_j} \cdot exp(f_{j-1}^D(i-1))] \tag{5.77}$$

We can transform the backward probabilities in the same way.

The forward and backward variables can then be combined to re-estimate emission and transition probability parameters as follows:

Baum-Welch re-estimation equations for profile HMMs:

1. Expected emission counts from sequence X:

$$E_{M_k}(a) = \frac{1}{P(X)} \sum_{i|x_i=a} f_k^M(i) b_k^M(i) \tag{5.78}$$

$$E_{I_k}(a) = \frac{1}{P(X)} \sum_{i|x_i=a} f_k^I(i) b_k^I(i) \tag{5.79}$$

2. Expected transition counts from sequence X (X_k is anyone of M_k, I_k, D_k):

$$A_{X_k M_{k+1}} = \frac{1}{P(X)} \sum_i f_k^X(i) a_{X_k M_{k+1}} e_{M_{k+1}}(x_{i+1}) b_{k+1}^M(i+1) \tag{5.80}$$

$$A_{X_k I_k} = \frac{1}{P(X)} \sum_i f_k^X(i) a_{X_k I_k} e_{I_k}(x_{i+1}) b_k^I(i+1) \tag{5.81}$$

$$A_{X_k D_{k+1}} = \frac{1}{P(X)} \sum_i f_k^X(i) a_{X_k D_{k+1}} b_{k+1}^D(i) \tag{5.82}$$

5.2.5 Multiple Alignment with Profile HMMs

Profile HMMs can help us to obtain an approximate solution to the multiple alignment problem. Given n sequences $S^{(1)}, \ldots, S^{(n)}$, consider the following cases:

1. If the profile HMM $\mathcal{P}$ is known, the following procedure can be applied:
   - Align each sequence $S^{(i)}$ to the profile separately.
   - Accumulate the obtained alignments to a multiple alignment.
   - Insert runs are not aligned, i.e. the choice of how to put the letters in the insert regions is arbitrary (Most profile HMM implementations simply left-justify insert regions, as in the following example).
2. If the profile HMM $\mathcal{P}$ is not known, one can use the following technique in order to obtain an HMM profile from the given sequences:
   - Choose a length L for the profile HMM and initialize the transition and emission probabilities.
   - Train the model using the Baum-Welch algorithm, on all the training sequences.
   - Obtain the multiple alignment from the resulting profile HMM, as in the previous case.

Figure 5.14 shows an example for obtaining a multiple alignment from a profile HMM. First, we derive the profile HMM shown in Figure 5.13 from a given multiple alignment of seven sequences. The shaded columns were arbitrarily defined to be insertions for the purposes of this example, and the other ten columns correspond to ten profile HMM match states. Second, the same seven sequences were realigned to the model, and their alignment is shown in Figure 5.14, left, where lower-case letters were assigned to an insert state and upper-case letters were assigned to a match state. We should notice that the original alignment and the new alignment are the same alignment. A profile HMM does not attempt to align the lower-case letters assigned to insert states (as was described in the algorithm). This is a biologically realistic view of multiple alignment, since the insert regions usually represent parts of the sequences which are unconserved, and not meaningfully alignable. Third, we align a new sequence to the same model, shown in Figure 5.14, right. This sequence has more inserted positions than any of the other seven sequences, so the alignment of the other seven sequences must be adjusted to allow space for these positions.

One can use an extension of the above approach to identify similar patterns in a given set of sequences by using the profile HMM for local alignment (see Figure 5.12). Next, we present another approach to this problem, which tackles the problem from a totally different perspective.

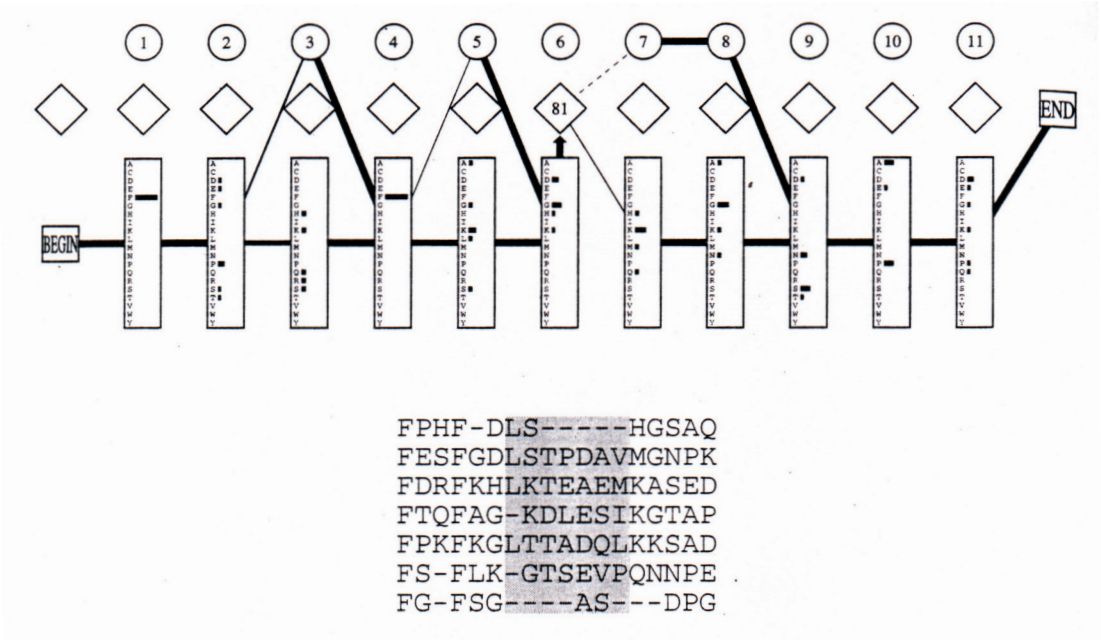

Figure 5.13: Source: [4]. A model (top) estimated from an alignment (bottom). The columns in the shaded area of the alignment were treated as inserts.

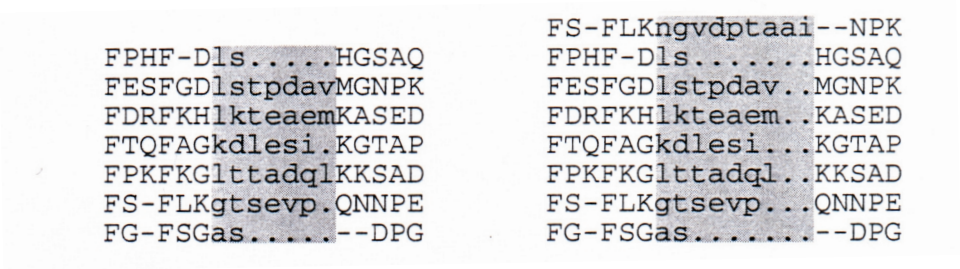

Figure 5.14: Source: [4]. Left: the alignment of seven sequences is shown with lower-case letters meaning inserts. The dots are just space-filling characters to make the matches line up correctly. Right: the alignment is shown after a new sequence was added to the set. The new sequence is shown at the top, and because it has more inserts, more space-filling dots were added.

5.3 Gibbs Sampling

Problem 5.5 Locating a common pattern.
INPUT: A set of sequences $\mathcal{S} = S^{(1)}, \ldots, S^{(n)}$ and an integer w.
QUESTION: For each string $S^{(i)}$, find a substring (pattern) of length at most w, so that the similarity between the n substrings is maximized.

Let $a^{(1)}, \ldots, a^{(n)}$ be the starting indices of the chosen substrings in $S^{(1)}, \ldots, S^{(n)}$, respectively. We introduce the following notations:

- Let c_{ij} be the number of occurrences of the symbol $j \in \Sigma$ among the i^{th} positions of the n substrings: $\{s^{(1)}_{a^{(1)}+i-1}, \ldots, s^{(n)}_{a^{(n)}+i-1}\}$.
- Let q_{ij} denote the probability of the symbol j to occur at the i^{th} position of the pattern.
- Let p_j denote the frequency of the symbol j in all the sequences in $\mathcal{S}$.

We therefore wish to maximize the logarithmic likelihood score:

$$Score = \sum_{i=1}^{w} \sum_{j \in \Sigma} c_{ij} \cdot \log \frac{q_{ij}}{p_j} \tag{5.83}$$

When the number of occurences of symbol j in the i^{th} position of the pattern increases, its frequency in that position q_{ij}, relative to its background frequency p_j, increases (and vice versa). In the above expression, each symbol j in the i^{th} position of the pattern is given a weight, represented by c_{ij}, which is proportional to its frequency in that position. Thus, when the similarity between the n substrings increases, we have more occurences of symbols j with high frequency in each position i, hence the given score increases.

To accomplish this task, we perform the following iterative procedure:

1. Initialization: Randomly choose $a^{(1)}, \ldots, a^{(n)}$.
2. Randomly choose $1 \leq z \leq n$ and calculate the c_{ij}, q_{ij} and p_j values for the strings in $\mathcal{S} \setminus S^{(z)}$.
3. Find the best substring of $S^{(z)}$ according to the model, and determine the new value of $a^{(z)}$. This is done by applying the algorithm for local alignment of $S^{(z)}$ against the profile of the current pattern.
4. Repeat steps 2 and 3 until the improvement of the score is less then ϵ.

Unlike the profile HMM technique, the Gibbs sampling algorithm (due to Lawrence et al. [8]) does not rely on any substantial theoretic basis. However, this method is known to work in specific cases.

Known problems:

- *Phase shift* - The algorithm may converge on an offset of the best pattern.
- The value of w is usually unknown. Choosing different values for w may significantly change the results.
- The strings may contain more than a single common pattern.
- As is the case with the Baum-Welch algorithm, the process may converge to a local maximum.

References

- Hidden Markov models were originally invented and are commonly used for speech recognition. For additional material on this subject, see [9].
- For a more comprehensive overview on the EM technique, see [1].
- For a detailed discussion about the adaptation of HMMs to computational biology algorithms see [4] (chapters 3-6).
- Applications of profile HMMs for multiple alignment of proteins can be found in [5] and for finding genes in the DNA can be found in [7, 6].

Bibliography

[1] N. M. Laird A. P. Dempster and D. B. Rubin. Maximum likelihood estimation from incomplete data. *Journal of the Royal Statistics Society*, 39:1–38, 1977.

[2] L. E. Baum. An inequality and associated maximization technique occuring in the statistical analysis of probabilistic functions of markov chains. *Inequalities*, 3:1–8, 1972.

[3] R. Bellman. *Dynamic Programming*. Princeton University Press, Boston, 1957.

[4] R. Durbin, S. Eddy, A. Krough, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[5] A. Krogh, M. Brown, S. Mian, M. Sjölander, and D. Haussler. Hidden markov models in computational biology. applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 4 February 1994.

[6] Anders Krogh, I. Saira Mian, and David Haussler. A hidden markov model that finds genes in E. coli DNA. *Nucleic Acids Research*, 22:4768–4778, 1994.

[7] D. Kulp, D. Haussler, M.G. Reese, and F.H. Eeckman. A generalized hidden Markov model for the recognition of human genes in DNA. In D. J. States, P. Agarwal, T. Gaasterland, L. Hunter, and R. Smith, editors, *Proc. Conf. On Intelligent Systems in Molecular Biology '96*, pages 134–142. AAAI/MIT Press, 1996. St. Louis, Mo.

[8] Charles E. Lawrence, Stephen F. Altschul, Mark S. Boguski, Jun S. Liu, Andrew F. Neuwald, and John C. Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262:208–214, 8 October 1993.

[9] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[10] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Information Theory*, IT-13:260–269, 1967.