For convenience, we have introduced π_0 and π_{n+1} as the fictitious initial and terminal states $begin$ and end.

This model defines the probability $P(x|\pi)$ for a given sequence x and a given path π. Since only the dealer knows the real sequence of states π that emitted x, we say that π is *hidden* and attempt to solve the following Decoding problem:

Decoding Problem:
Find an optimal hidden path of states given observations.

Input: Sequence of observations $x = x_1 \ldots x_n$ generated by an HMM $\mathcal{M}(\sum, Q, A, E)$.

Output: A path that maximizes $P(x|\pi)$ over all possible paths π.

The Decoding problem is an improved formulation of the ill-defined Fair Bet Casino problem.

11.3 Decoding Algorithm

In 1967 Andrew Viterbi used an HMM-inspired analog of the Manhattan grid for the Decoding problem, and described an efficient dynamic programming algorithm for its solution. Viterbi's Manhattan is shown in figure 11.2 with every choice of $\pi_1, \ldots, \pi_n$ corresponding to a path in this graph. One can set the edge weights in this graph so that the product of the edge weights for path $\pi{=}\pi_1 \ldots \pi_n$ equals $P(x|\pi)$. There are $|Q|^2(n-1)$ edges in this graph with the weight of an edge from (k, i) to $(l, i+1)$ given by $e_l(x_{i+1}) \cdot a_{kl}$. Unlike the alignment approaches covered in chapter 6 where the set of valid directions was restricted to south, east, and southeast edges, the Manhattan built to solve the decoding problem only forces the tourists to move in any eastward direction (e.g., northeast, east, southeast, etc.), and places no additional restrictions (fig. 11.3). To see why the length of the edge between the vertices (k, i) and $(l, i+1)$ in the corresponding graph is given by $e_l(x_{i+1}) \cdot a_{kl}$, one should compare $p_{k,i}$ [the probability of a path ending in vertex (k, i)] with

the probability

$$\begin{aligned} p_{l,i+1} &= \prod_{j=1}^{i+1} e_{\pi_j}(x_j) \cdot a_{\pi_{j-1},\pi_j} \\ &= \left(\prod_{j=1}^{i} e_{\pi_j}(x_j) \cdot a_{\pi_{j-1},\pi_j} \right) \cdot (e_{\pi_{i+1}}(x_{i+1}) \cdot a_{\pi_i,\pi_{i+1}}) \\ &= p_{k,i} \cdot e_l(x_{i+1}) \cdot a_{kl} \\ &= p_{k,i} \cdot \text{weight of edge from } (k,i) \text{ to } (l,i+1) \end{aligned}$$

Therefore, the decoding problem is reduced to finding a longest path in the directed acyclic graph (DAG) shown in figure 11.2, which poses no problems to the algorithm presented in chapter 6. We remark that, in this case, the length of the path is defined as the product of its edges' weights, rather than the sum of weights used in previous examples of dynamic programming algorithms, but the application of logarithms makes the problems the same.

The idea behind the Viterbi algorithm is that the optimal path for the $(i+1)$-prefix $x_1 \dots x_{i+1}$ of x uses a path for $x_1\, x_2 \cdots x_i$ that is optimal among the paths ending in some unknown state π_i. Let k be some state from Q, and let i be between 1 and n. Define $s_{k,i}$ to be the probability of the most likely path for the prefix $x_1 \dots x_i$ that ends at state k. Then, for any state l,

$$\begin{aligned} s_{l,i+1} &= \max_{k \in Q} \{ s_{k,i} \cdot \text{ weight of edge between } (k,i) \text{ and } (l,i+1) \} \\ &= \max_{k \in Q} \{ s_{k,i} \cdot a_{kl} \cdot e_l(x_{i+1}) \} \\ &= e_l(x_{i+1}) \cdot \max_{k \in Q} \{ s_{k,i} \cdot a_{kl} \} \end{aligned}$$

We initialize $s_{begin,0} = 1$ and $s_{k,0} = 0$ for $k \neq begin$. If π^* is an optimal path, then the value of $P(x|\pi^*)$ is

$$P(x|\pi^*) = \max_{k \in Q} \{ s_{k,n} \cdot a_{k,end} \}$$

As in chapter 6, these recurrence relations and the initial conditions determine the entire dynamic programming algorithm, so we do not provide pseudocode here.

The Viterbi algorithm runs in $O(n|Q|^2)$ time. The computations in the Viterbi algorithm are usually done using logarithmic scores $S_{k,i} = \log s_{k,i}$

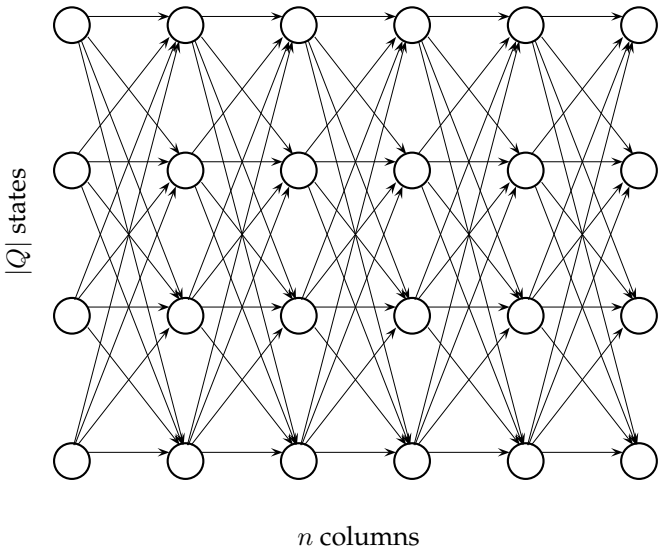

Figure 11.2 Manhattan, according to Viterbi, consists of $|Q|$ rows, n columns, and $|Q|^2$ edges per layer ($|Q| = 4$ and $n = 6$ in the example above).

to avoid overflow[6]:

$$S_{l,i+1} = \log e_l(x_{i+1}) + \max_{k \in Q}\{S_{k,i} + \log(a_{kl})\}.$$

As we showed above, every path π through the graph in figure 11.2 has probability $P(x|\pi)$. The Viterbi algorithm is essentially a search through the space of all possible paths in that graph for the one that maximizes the value of $P(x|\pi)$.

We can also ask a slightly different question: given x and the HMM, what is the probability $P(\pi_i = k|x)$ that the HMM was in state k at time i? In the casino analogy, we are given a sequence of coin tosses and are interested in the probability that the dealer was using a biased coin at a particular time.

We define $P(x) = \sum_\pi P(x|\pi)$ as the sum of probabilities of all paths and $P(x, \pi_i = k) = \sum_{\text{all } \pi \text{ with } \pi_i = k} P(x|\pi)$ as the sum of probabilities of all

6. Overflow occurs in real computers because there are only a finite number of bits (binary digits) in which to hold a number.

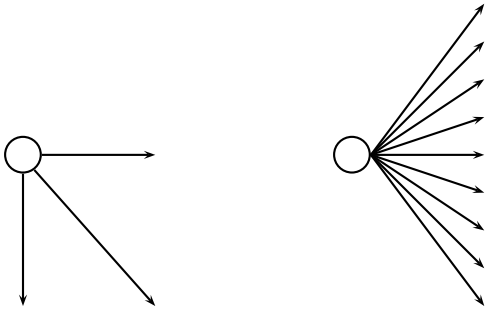

Figure 11.3 The set of valid directions in the alignment problem (a) is usually limited to south, east, and southeast edges, while the set of valid directions in the decoding problem (b) includes any eastbound edge.

paths with $\pi_i = k$. The ratio $\frac{P(x,\pi_i=k)}{P(x)}$ defines the probability $P(\pi_i = k|x)$ that we are trying to compute.

A simple variation of the Viterbi algorithm allows us to compute the probability $P(x, \pi_i = k)$. Let $f_{k,i}$ be the probability of emitting the prefix $x_1 \ldots x_i$ and reaching the state $\pi_i = k$. It can be expressed as follows.

$$f_{k,i} = e_k(x_i) \cdot \sum_{l \in Q} f_{l,i-1} \cdot a_{lk}$$

The only difference between the *forward algorithm* that calculates $f_{k,i}$ and the Viterbi algorithm is that the "$\max$" sign in the Viterbi algorithm changes into a "$\sum$" sign in the forward algorithm.

However, forward probability $f_{k,i}$ is not the only factor affecting $P(\pi_i = k|x)$. The sequence of transitions and emissions that the HMM undergoes between π_{i+1} and π_n also affects $P(\pi_i = k|x)$. The backward probability $b_{k,i}$ is defined as the probability of being at state $\pi_i = k$ and emitting the *suffix*

$x_{i+1} \dots x_n$. The *backward algorithm* uses a similar recurrence:

$$b_{k,i} = \sum_{l \in Q} e_l(x_{i+1}) \cdot b_{l,i+1} \cdot a_{kl}$$

Finally, the probability that the dealer had a biased coin at moment i is given by

$$P(\pi_i = k|x) = \frac{P(x, \pi_i = k)}{P(x)} = \frac{f_k(i) \cdot b_k(i)}{P(x)}.$$

11.4 HMM Parameter Estimation

The preceding analysis assumed that we know the state transition and emission probabilities of the HMM. Given these parameters, it is easy for an intelligent gambler to figure out that the dealer in the Fair Bet Casino is using a biased coin, simply by noticing that 0 and 1 have different expected frequencies ($\frac{3}{8}$ vs $\frac{5}{8}$). If the ratio of zeros to ones in a daylong sequence of tosses is suspiciously low, then it is likely that the dealer is using a biased coin. Unfortunately, the most difficult problem in the application of HMMs is that the HMM parameters are usually unknown and therefore need to be estimated from data. It is more difficult to estimate the transition and emission probabilities of an HMM than it is to reconstruct the most probable sequence of states it went through when you do know the probabilities. In this case, we are given the set of states, Q, but we do not know with what probability the HMM moves from one state to another, or with what probability it emits any particular symbol.

Let Θ be a vector combining the unknown transition and emission probabilities of the HMM $\mathcal{M}$. Given an observed symbol string x that the HMM emitted, define $P(x|\Theta)$ as the maximum probability of x given the assignment of parameters Θ. Our goal is to find

$$\max_{\Theta} P(x|\Theta).$$

Usually, instead of a single string x, we can obtain a sample of *training sequences* $x^1, \dots, x^m$, so a natural goal is to find

$$\max_{\Theta} \prod_{j=1}^{m} P(x^j|\Theta).$$

This results in a difficult optimization problem in the multidimensional parameter space Θ. Commonly used algorithms for this type of parameter optimization are heuristics that use local improvement strategies. If we know the path $\pi_1 \dots \pi_n$ corresponding to the observed states $x_1 \dots x_n$, then we can scan the sequences and compute empirical estimates for transition and emission probabilities. If A_{kl} is the number of transitions from state k to l and $E_k(b)$ is the number of times b is emitted from state k, then the reasonable estimators are

$$a_{kl} = \frac{A_{kl}}{\sum_{q \in Q} A_{kq}} \qquad e_k(b) = \frac{E_k(b)}{\sum_{\sigma \in \sum} E_k(\sigma)}.$$

However, we do not usually know the state sequence $\pi = \pi_1 \dots \pi_n$, and in this case we can start from a wild guess for $\pi_1 \dots \pi_n$, compute empirical estimates for transition and emission probabilities using this guess, and solve the decoding problem to find a new, hopefully less wild, estimate for π. The commonly used iterative local improvement strategy, called the *Baum-Welch* algorithm, uses a similar approach to estimate HMM parameters.

11.5 Profile HMM Alignment

Given a family of functionally related biological sequences, one can search for new members of the family from a database using pairwise alignments between family members and sequences from the database. However, this approach may fail to identify distantly related sequences because distant cousins may have weak similarities that do not pass the statistical significance test. However, if the sequence has weak similarities with many family members, it is likely to belong to the family. The problem then is to somehow align a sequence to *all* members of the family at once, using the whole set of functionally related sequences in the search.

The simplest representation of a family of related proteins is given by their multiple alignment and the corresponding profile.[7] As with sequences, profiles can also be compared and aligned against each other since the dynamic programming algorithm for aligning two sequences works if both of the input sequences are profiles.

7. While in chapter 4 we defined profile element p_{ij} as a *count* of the nucleotide i in the jth column of alignment matrix, biologists usually define p_{ij} as *frequency* of the nucleotide i in the jth column of the alignment matrix, that is, they divide all of the counts by t (see figure 11.4). In order to avoid columns that contain one or more letters with probabilities of 0, small numbers