25

Exact Marginalization in Trellises

In this chapter we will discuss a few exact methods that are used in probabilistic modelling. As an example we will discuss the task of decoding a linear error-correcting code. We will see that inferences can be conducted most efficiently by *message-passing algorithms*, which take advantage of the graphical structure of the problem to avoid unnecessary duplication of computations (see Chapter 16).

▶ 25.1 Decoding problems

A codeword $\mathbf{t}$ is selected from a linear (N, K) code $\mathcal{C}$, and it is transmitted over a noisy channel; the received signal is $\mathbf{y}$. In this chapter we will assume that the channel is a memoryless channel such as a Gaussian channel. Given an assumed channel model $P(\mathbf{y} \mid \mathbf{t})$, there are two decoding problems.

The codeword decoding problem is the task of inferring which codeword $\mathbf{t}$ was transmitted given the received signal.

The bitwise decoding problem is the task of inferring for each transmitted bit t_n how likely it is that that bit was a one rather than a zero.

As a concrete example, take the $(7, 4)$ Hamming code. In Chapter 1, we discussed the codeword decoding problem for that code, assuming a binary symmetric channel. We didn't discuss the bitwise decoding problem and we didn't discuss how to handle more general channel models such as a Gaussian channel.

Solving the codeword decoding problem

By Bayes' theorem, the posterior probability of the codeword $\mathbf{t}$ is

$$P(\mathbf{t} \mid \mathbf{y}) = \frac{P(\mathbf{y} \mid \mathbf{t}) P(\mathbf{t})}{P(\mathbf{y})}. \tag{25.1}$$

Likelihood function. The first factor in the numerator, $P(\mathbf{y} \mid \mathbf{t})$, is the *likelihood* of the codeword, which, for any memoryless channel, is a separable function,

$$P(\mathbf{y} \mid \mathbf{t}) = \prod_{n=1}^{N} P(y_n \mid t_n). \tag{25.2}$$

For example, if the channel is a Gaussian channel with transmissions $\pm x$ and additive noise of standard deviation σ, then the probability density

of the received signal y_n in the two cases $t_n = 0, 1$ is

$$P(y_n \,|\, t_n = 1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - x)^2}{2\sigma^2}\right) \tag{25.3}$$

$$P(y_n \,|\, t_n = 0) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n + x)^2}{2\sigma^2}\right). \tag{25.4}$$

From the point of view of decoding, all that matters is the *likelihood ratio*, which for the case of the Gaussian channel is

$$\frac{P(y_n \,|\, t_n = 1)}{P(y_n \,|\, t_n = 0)} = \exp\left(\frac{2xy_n}{\sigma^2}\right). \tag{25.5}$$

Exercise 25.1.[2] Show that from the point of view of decoding, a Gaussian channel is equivalent to a time-varying binary symmetric channel with a known noise level f_n which depends on n.

Prior. The second factor in the numerator is the *prior* probability of the codeword, $P(\mathbf{t})$, which is usually assumed to be uniform over all valid codewords.

The denominator in (25.1) is the normalizing constant

$$P(\mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{y} \,|\, \mathbf{t}) P(\mathbf{t}). \tag{25.6}$$

The complete solution to the codeword decoding problem is a list of all codewords and their probabilities as given by equation (25.1). Since the number of codewords in a linear code, 2^K, is often very large, and since we are not interested in knowing the detailed probabilities of all the codewords, we often restrict attention to a simplified version of the codeword decoding problem.

The MAP codeword decoding problem is the task of identifying *the most probable codeword* $\mathbf{t}$ given the received signal.

If the prior probability over codewords is uniform then this task is identical to the problem of *maximum likelihood decoding*, that is, identifying the codeword that maximizes $P(\mathbf{y} \,|\, \mathbf{t})$.

Example: In Chapter 1, for the $(7, 4)$ Hamming code and a binary symmetric channel we discussed a method for deducing the most probable codeword from the syndrome of the received signal, thus solving the MAP codeword decoding problem for that case. We would like a more general solution.

The MAP codeword decoding problem can be solved in exponential time (of order 2^K) by searching through all codewords for the one that maximizes $P(\mathbf{y} \,|\, \mathbf{t})P(\mathbf{t})$. But we are interested in methods that are more efficient than this. In section 25.3, we will discuss an exact method known as the *min–sum algorithm* which may be able to solve the codeword decoding problem more efficiently; how much more efficiently depends on the properties of the code.

It is worth emphasizing that MAP codeword decoding for a *general* linear code is known to be NP-complete (which means in layman's terms that MAP codeword decoding has a complexity that scales exponentially with the block length, unless there is a revolution in computer science). So restricting attention to the MAP decoding problem hasn't necessarily made the task much less challenging; it simply makes the answer briefer to report.

Solving the bitwise decoding problem

Formally, the exact solution of the bitwise decoding problem is obtained from equation (25.1) by *marginalizing* over the other bits.

$$P(t_n \,|\, \mathbf{y}) = \sum_{\{t_{n'} : n' \neq n\}} P(\mathbf{t} \,|\, \mathbf{y}). \tag{25.7}$$

We can also write this marginal with the aid of a truth function $\mathbb{1}[S]$ that is one if the proposition S is true and zero otherwise.

$$P(t_n = 1 \,|\, \mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t} \,|\, \mathbf{y}) \, \mathbb{1}[t_n = 1] \tag{25.8}$$

$$P(t_n = 0 \,|\, \mathbf{y}) = \sum_{\mathbf{t}} P(\mathbf{t} \,|\, \mathbf{y}) \, \mathbb{1}[t_n = 0]. \tag{25.9}$$

Computing these marginal probabilities by an explicit sum over all codewords $\mathbf{t}$ takes exponential time. But, for certain codes, the bitwise decoding problem can be solved much more efficiently using the *forward–backward algorithm*. We will describe this algorithm, which is an example of the *sum–product algorithm*, in a moment. Both the min–sum algorithm and the sum–product algorithm have widespread importance, and have been invented many times in many fields.

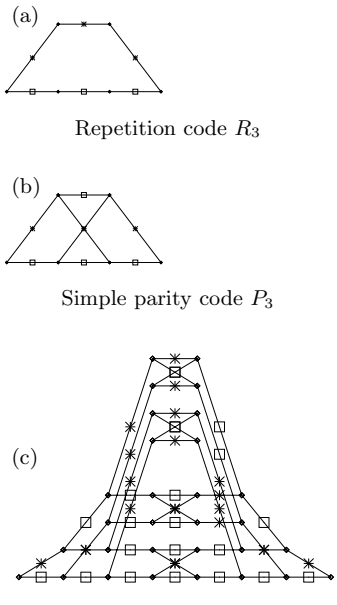

Figure 25.1. Examples of trellises. Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

▶ 25.2 Codes and trellises

In Chapters 1 and 11, we represented linear (N, K) codes in terms of their generator matrices and their parity-check matrices. In the case of a *systematic* block code, the first K transmitted bits in each block of size N are the source bits, and the remaining $M = N - K$ bits are the parity-check bits. This means that the generator matrix of the code can be written

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_K \\ \mathbf{P} \end{bmatrix}, \tag{25.10}$$

and the parity-check matrix can be written

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_M \end{bmatrix}, \tag{25.11}$$

where $\mathbf{P}$ is an $M \times K$ matrix.

In this section we will now study another representation of a linear code called a trellis. The codes that these trellises represent will not in general be systematic codes, but they can be mapped onto systematic codes if desired by a reordering of the bits in a block.

Definition of a trellis

Our definition will be quite narrow. For a more comprehensive view of trellises, the reader should consult Kschischang and Sorokine (1995).

A trellis is a *graph* consisting of *nodes* (also known as states or vertices) and *edges*. The nodes are grouped into vertical slices called *times*, and the times are ordered such that each edge connects a node in one time to a node in a neighbouring time. Every edge is labelled with a *symbol*. The leftmost and rightmost states contain only one node. Apart from these two extreme nodes, all nodes in the trellis have at least one edge connecting leftwards and at least one connecting rightwards.

A trellis with $N+1$ times defines a code of block length N as follows: a codeword is obtained by taking a path that crosses the trellis from left to right and reading out the symbols on the edges that are traversed. Each valid path through the trellis defines a codeword. We will number the leftmost time 'time 0' and the rightmost 'time N'. We will number the leftmost state 'state 0' and the rightmost 'state I', where I is the total number of states (vertices) in the trellis. The nth bit of the codeword is emitted as we move from time $n-1$ to time n.

The *width* of the trellis at a given time is the number of nodes in that time. The *maximal width* of a trellis is what it sounds like.

A trellis is called a *linear trellis* if the code it defines is a linear code. We will solely be concerned with linear trellises from now on, as nonlinear trellises are much more complex beasts. For brevity, we will only discuss binary trellises, that is, trellises whose edges are labelled with zeroes and ones. It is not hard to generalize the methods that follow to q-ary trellises.

Figures 25.1(a–c) show the trellises corresponding to the repetition code R_3 which has $(N, K) = (3, 1)$; the parity code P_3 with $(N, K) = (3, 2)$; and the $(7, 4)$ Hamming code.

▷ Exercise 25.2.[2] Confirm that the sixteen codewords listed in table 1.14 are generated by the trellis shown in figure 25.1c.

Observations about linear trellises

For any linear code the *minimal trellis* is the one that has the smallest number of nodes. In a minimal trellis, each node has at most two edges entering it and at most two edges leaving it. All nodes in a time have the same left degree as each other and they have the same right degree as each other. The width is always a power of two.

A minimal trellis for a linear (N, K) code cannot have a width greater than 2^K since every node has at least one valid codeword through it, and there are only 2^K codewords. Furthermore, if we define $M = N - K$, the minimal trellis's width is everywhere less than 2^M. This will be proved in section 25.4.

Notice that for the linear trellises in figure 25.1, all of which are minimal trellises, K is the number of times a binary branch point is encountered as the trellis is traversed from left to right or from right to left.

We will discuss the construction of trellises more in section 25.4. But we now know enough to discuss the decoding problem.

▶ 25.3 Solving the decoding problems on a trellis

We can view the trellis of a linear code as giving a causal description of the probabilistic process that gives rise to a codeword, with time flowing from left to right. Each time a divergence is encountered, a random source (the source of information bits for communication) determines which way we go.

At the receiving end, we receive a noisy version of the sequence of edge-labels, and wish to infer which path was taken, or to be precise, (a) we want to identify the most probable path in order to solve the codeword decoding problem; and (b) we want to find the probability that the transmitted symbol at time n was a zero or a one, to solve the bitwise decoding problem.

Example 25.3. Consider the case of a single transmission from the Hamming $(7, 4)$ trellis shown in figure 25.1c.

t	Likelihood	Posterior probability
0000000	0.0275562	0.25
0001011	0.0001458	0.0013
0010111	0.0013122	0.012
0011100	0.0030618	0.027
0100110	0.0002268	0.0020
0101101	0.0000972	0.0009
0110001	0.0708588	0.63
0111010	0.0020412	0.018
1000101	0.0001458	0.0013
1001110	0.0000042	0.0000
1010010	0.0030618	0.027
1011001	0.0013122	0.012
1100011	0.0000972	0.0009
1101000	0.0002268	0.0020
1110100	0.0020412	0.018
1111111	0.0000108	0.0001

Figure 25.2. Posterior probabilities over the sixteen codewords when the received vector $\mathbf{y}$ has normalized likelihoods $(0.1, 0.4, 0.9, 0.1, 0.1, 0.1, 0.3)$.

Let the normalized likelihoods be: $(0.1, 0.4, 0.9, 0.1, 0.1, 0.1, 0.3)$. That is, the ratios of the likelihoods are

$$\frac{P(y_1 \,|\, x_1 = 1)}{P(y_1 \,|\, x_1 = 0)} = \frac{0.1}{0.9}, \quad \frac{P(y_2 \,|\, x_2 = 1)}{P(y_2 \,|\, x_2 = 0)} = \frac{0.4}{0.6}, \quad \text{etc.} \tag{25.12}$$

How should this received signal be decoded?

1. If we threshold the likelihoods at 0.5 to turn the signal into a binary received vector, we have $\mathbf{r} = (0, 0, 1, 0, 0, 0, 0)$, which decodes, using the decoder for the binary symmetric channel (Chapter 1), into $\hat{\mathbf{t}} = (0, 0, 0, 0, 0, 0, 0)$.

   This is not the optimal decoding procedure. Optimal inferences are always obtained by using Bayes' theorem.

2. We can find the posterior probability over codewords by explicit enumeration of all sixteen codewords. This posterior distribution is shown in figure 25.2. Of course, we aren't really interested in such brute-force solutions, and the aim of this chapter is to understand algorithms for getting the same information out in less than 2^K computer time.

   Examining the posterior probabilities, we notice that the most probable codeword is actually the string $\mathbf{t} =$ 0110001. This is more than twice as probable as the answer found by thresholding, 0000000.

   Using the posterior probabilities shown in figure 25.2, we can also compute the posterior marginal distributions of each of the bits. The result is shown in figure 25.3. Notice that bits 1, 4, 5 and 6 are all quite confidently inferred to be zero. The strengths of the posterior probabilities for bits 2, 3, and 7 are not so great. □

In the above example, the MAP codeword is in agreement with the bitwise decoding that is obtained by selecting the most probable state for each bit using the posterior marginal distributions. But this is not always the case, as the following exercise shows.

n	Likelihood		Posterior marginals	
	$P(y_n \mid t_n=1)$	$P(y_n \mid t_n=0)$	$P(t_n=1 \mid \mathbf{y})$	$P(t_n=0 \mid \mathbf{y})$
1	0.1	0.9	0.061	0.939
2	0.4	0.6	0.674	0.326
3	0.9	0.1	0.746	0.254
4	0.1	0.9	0.061	0.939
5	0.1	0.9	0.061	0.939
6	0.1	0.9	0.061	0.939
7	0.3	0.7	0.659	0.341

Figure 25.3. Marginal posterior probabilities for the 7 bits under the posterior distribution of figure 25.2.

Exercise 25.4.[2, p.335] Find the most probable codeword in the case where the normalized likelihood is $(0.2, 0.2, 0.9, 0.2, 0.2, 0.2, 0.2)$. Also find or estimate the marginal posterior probability for each of the seven bits, and give the bit-by-bit decoding.

[Hint: concentrate on the few codewords that have the largest probability.]

We now discuss how to use message passing on a code's trellis to solve the decoding problems.

The min–sum algorithm

The MAP codeword decoding problem can be solved using the min–sum algorithm that was introduced in section 16.3. Each codeword of the code corresponds to a path across the trellis. Just as the cost of a journey is the sum of the costs of its constituent steps, the log likelihood of a codeword is the sum of the bitwise log likelihoods. By convention, we flip the sign of the log likelihood (which we would like to maximize) and talk in terms of a cost, which we would like to minimize.

We associate with each edge a cost $-\log P(y_n \mid t_n)$, where t_n is the transmitted bit associated with that edge, and y_n is the received symbol. The min–sum algorithm presented in section 16.3 can then identify the most probable codeword in a number of computer operations equal to the number of edges in the trellis. This algorithm is also known as the Viterbi algorithm (Viterbi, 1967).

The sum–product algorithm

To solve the bitwise decoding problem, we can make a small modification to the min–sum algorithm, so that the messages passed through the trellis define 'the probability of the data up to the current point' instead of 'the cost of the best route to this point'. We replace the costs on the edges, $-\log P(y_n \mid t_n)$, by the likelihoods themselves, $P(y_n \mid t_n)$. We replace the min and sum operations of the min–sum algorithm by a sum and product respectively.

Let i run over nodes/states, $i = 0$ be the label for the start state, $\mathcal{P}(i)$ denote the set of states that are parents of state i, and w_{ij} be the likelihood associated with the edge from node j to node i. We define the forward-pass messages α_i by

$$\alpha_0 = 1$$

$$\alpha_i = \sum_{j \in \mathcal{P}(i)} w_{ij} \alpha_j. \tag{25.13}$$

These messages can be computed sequentially from left to right.

▷ Exercise 25.5.[2] Show that for a node i whose time-coordinate is n, α_i is proportional to the joint probability that the codeword's path passed through node i and that the first n received symbols were $y_1, \ldots, y_n$.

The message α_i computed at the end node of the trellis is proportional to the marginal probability of the data.

▷ Exercise 25.6.[2] What is the constant of proportionality? [Answer: 2^K]

We define a second set of backward-pass messages β_i in a similar manner. Let node I be the end node.

$$\begin{aligned} \beta_I &= 1 \\ \beta_j &= \sum_{i: j \in \mathcal{P}(i)} w_{ij} \beta_i. \end{aligned} \tag{25.14}$$

These messages can be computed sequentially in a backward pass from right to left.

▷ Exercise 25.7.[2] Show that for a node i whose time-coordinate is n, β_i is proportional to the conditional probability, *given* that the codeword's path passed through node i, that the subsequent n received symbols were $y_{n+1} \cdots y_N$.

Finally, to find the probability that the nth bit was a 1 or 0, we do two summations of products of the forward and backward messages. Let i run over nodes at time n and j run over nodes at time $n-1$, and let t_{ij} be the value of t_n associated with the trellis edge from node j to node i. For each value of $t = 0/1$, we compute

$$r_n^{(t)} = \sum_{i,j:\, j \in \mathcal{P}(i),\, t_{ij}=t} \alpha_j w_{ij} \beta_i. \tag{25.15}$$

Then the posterior probability that t_n was $t = 0/1$ is

$$P(t_n = t \,|\, \mathbf{y}) = \frac{1}{Z} r_n^{(t)}, \tag{25.16}$$

where the normalizing constant $Z = r_n^{(0)} + r_n^{(1)}$ should be identical to the final forward message α_I that was computed earlier.

Exercise 25.8.[2] Confirm that the above sum–product algorithm does compute $P(t_n = t \,|\, \mathbf{y})$.

Other names for the sum–product algorithm presented here are 'the forward–backward algorithm', 'the BCJR algorithm', and 'belief propagation'.

▷ Exercise 25.9.[2, p.335] A codeword of the simple parity code P_3 is transmitted, and the received signal $\mathbf{y}$ has associated likelihoods shown in table 25.4. Use the min–sum algorithm and the sum–product algorithm in the trellis (figure 25.1) to solve the MAP codeword decoding problem and the bitwise decoding problem. Confirm your answers by enumeration of all codewords (000, 011, 110, 101). [Hint: use logs to base 2 and do the min–sum computations by hand. When working the sum–product algorithm by hand, you may find it helpful to use three colours of pen, one for the αs, one for the ws, and one for the βs.]

n	$P(y_n \,\|\, t_n)$	
	$t_n = 0$	$t_n = 1$
1	1/4	1/2
2	1/2	1/4
3	1/8	1/2

Table 25.4. Bitwise likelihoods for a codeword of P_3.

▶ 25.4 More on trellises

We now discuss various ways of making the trellis of a code. You may safely jump over this section.

The *span* of a codeword is the set of bits contained between the first bit in the codeword that is non-zero, and the last bit that is non-zero, inclusive. We can indicate the span of a codeword by a binary vector as shown in table 25.5.

Codeword	`0000000`	`0001011`	`0100110`	`1100011`	`0101101`
Span	`0000000`	`0001111`	`0111110`	`1111111`	`0111111`

Table 25.5. Some codewords and their spans.

A generator matrix is in *trellis-oriented form* if the spans of the rows of the generator matrix all start in different columns and the spans all end in different columns.

How to make a trellis from a generator matrix

First, put the generator matrix into trellis-oriented form by row-manipulations similar to Gaussian elimination. For example, our $(7, 4)$ Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{25.17}$$

but this matrix is not in trellis-oriented form – for example, rows 1, 3 and 4 all have spans that end in the same column. By subtracting lower rows from upper rows, we can obtain an equivalent generator matrix (that is, one that generates the same set of codewords) as follows:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{25.18}$$

Now, each row of the generator matrix can be thought of as defining an $(N, 1)$ subcode of the (N, K) code, that is, in this case, a code with two codewords of length $N = 7$. For the first row, the code consists of the two codewords `1101000` and `0000000`. The subcode defined by the second row consists of `0100110` and `0000000`. It is easy to construct the minimal trellises of these subcodes; they are shown in the left column of figure 25.6.

We build the trellis incrementally as shown in figure 25.6. We start with the trellis corresponding to the subcode given by the first row of the generator matrix. Then we add in one subcode at a time. The vertices within the span of the new subcode are all duplicated. The edge symbols in the original trellis are left unchanged and the edge symbols in the second part of the trellis are flipped wherever the new subcode has a `1` and otherwise left alone.

Another $(7, 4)$ Hamming code can be generated by

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{25.19}$$

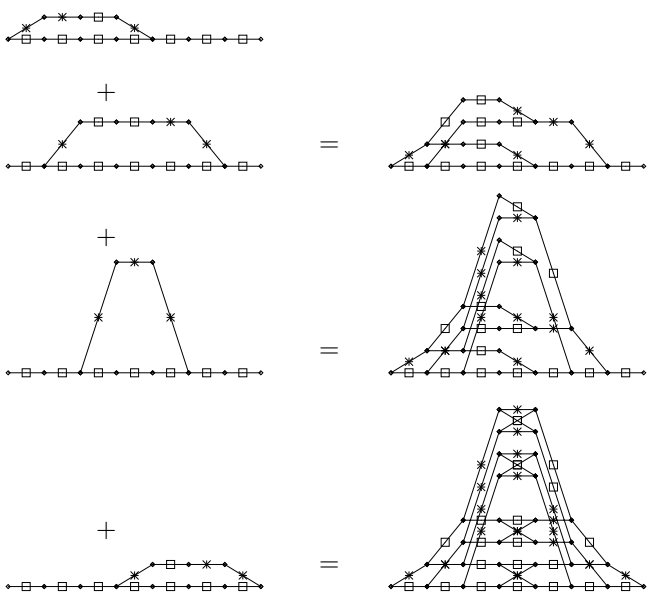

Figure 25.6. Trellises for four subcodes of the $(7, 4)$ Hamming code (left column), and the sequence of trellises that are made when constructing the trellis for the $(7, 4)$ Hamming code (right column).
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

The $(7, 4)$ Hamming code generated by this matrix differs by a permutation of its bits from the code generated by the systematic matrix used in Chapter 1 and above. The parity-check matrix corresponding to this permutation is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{25.20}$$

The trellis obtained from the permuted matrix $\mathbf{G}$ given in equation (25.19) is shown in figure 25.7a. Notice that the number of nodes in this trellis is smaller than the number of nodes in the previous trellis for the Hamming $(7, 4)$ code in figure 25.1c. We thus observe that *rearranging the order of the codeword bits can sometimes lead to smaller, simpler trellises.*

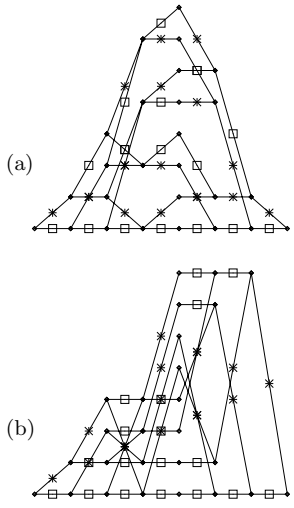

Figure 25.7. Trellises for the permuted $(7, 4)$ Hamming code generated from (a) the generator matrix by the method of figure 25.6; (b) the parity-check matrix by the method on page 334.
Each edge in a trellis is labelled by a zero (shown by a square) or a one (shown by a cross).

Trellises from parity-check matrices

Another way of viewing the trellis is in terms of the syndrome. The syndrome of a vector $\mathbf{r}$ is defined to be $\mathbf{Hr}$, where $\mathbf{H}$ is the parity-check matrix. A vector is only a codeword if its syndrome is zero. As we generate a codeword we can describe the current state by the *partial syndrome*, that is, the product of $\mathbf{H}$ with the codeword bits thus far generated. Each state in the trellis is a partial syndrome at one time coordinate. The starting and ending states are both constrained to be the zero syndrome. Each node in a state represents a different possible value for the partial syndrome. Since $\mathbf{H}$ is an $M \times N$ matrix, where $M = N - K$, the syndrome is at most an M-bit vector. So we need at most 2^M nodes in each state. We can construct the trellis of a code from its parity-check matrix by walking from each end, generating two trees of possible syndrome sequences. The intersection of these two trees defines the trellis of the code.

In the pictures we obtain from this construction, we can let the vertical coordinate represent the syndrome. Then any horizontal edge is necessarily associated with a zero bit (since only a non-zero bit changes the syndrome)

and any non-horizontal edge is associated with a one bit. (Thus in this representation we no longer need to label the edges in the trellis.) Figure 25.7b shows the trellis corresponding to the parity-check matrix of equation (25.20).

▶ 25.5 Solutions

Table 25.8. The posterior probability over codewords for exercise 25.4.

t	Likelihood	Posterior probability
0000000	0.026	0.3006
0001011	0.00041	0.0047
0010111	0.0037	0.0423
0011100	0.015	0.1691
0100110	0.00041	0.0047
0101101	0.00010	0.0012
0110001	0.015	0.1691
0111010	0.0037	0.0423
1000101	0.00041	0.0047
1001110	0.00010	0.0012
1010010	0.015	0.1691
1011001	0.0037	0.0423
1100011	0.00010	0.0012
1101000	0.00041	0.0047
1110100	0.0037	0.0423
1111111	0.000058	0.0007

Solution to exercise 25.4 (p.331). The posterior probability over codewords is shown in table 25.8. The most probable codeword is 0000000. The marginal posterior probabilities of all seven bits are:

n	Likelihood		Posterior marginals	
	$P(y_n \mid t_n = 1)$	$P(y_n \mid t_n = 0)$	$P(t_n = 1 \mid \mathbf{y})$	$P(t_n = 0 \mid \mathbf{y})$
1	0.2	0.8	0.266	0.734
2	0.2	0.8	0.266	0.734
3	0.9	0.1	0.677	0.323
4	0.2	0.8	0.266	0.734
5	0.2	0.8	0.266	0.734
6	0.2	0.8	0.266	0.734
7	0.2	0.8	0.266	0.734

So the bitwise decoding is 0010000, which is not actually a codeword.

Solution to exercise 25.9 (p.332). The MAP codeword is 101, and its likelihood is 1/8. The normalizing constant of the sum–product algorithm is $Z = \alpha_I = 3/16$. The intermediate α_i are (from left to right) $1/2$, $1/4$, $5/16$, $4/16$; the intermediate β_i are (from right to left), $1/2$, $1/8$, $9/32$, $3/16$. The bitwise decoding is: $P(t_1 = 1 \mid \mathbf{y}) = 3/4$; $P(t_1 = 1 \mid \mathbf{y}) = 1/4$; $P(t_1 = 1 \mid \mathbf{y}) = 5/6$. The codewords's probabilities are $1/12$, $2/12$, $1/12$, $8/12$ for 000, 011, 110, 101.