CHAPTER 12

Optimum Decoding of Convolutional Codes

In this chapter we show that convolutional encoders have a natural trellis structure, and we present two decoding algorithms based on this structure. Both of these algorithms are optimal according to a certain criterion. In 1967 Viterbi [1] introduced a decoding algorithm for convolutional codes that has since become known as the *Viterbi algorithm*. Later, Omura [2] showed that the Viterbi algorithm was equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. Then, Forney [3, 4] recognized that it was in fact a maximum likelihood (ML) decoding algorithm for convolutional codes; that is, the decoder output selected is always the codeword that maximizes the conditional probability of the received sequence. The *Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm* [5] was introduced in 1974 as a maximum a posteriori probability (MAP) decoding method for convolutional codes or block codes with a trellis structure. The optimality condition for the BCJR algorithm is slightly different than for the Viterbi algorithm: in MAP decoding, the probability of information bit error is minimized, whereas in ML decoding the probability of codeword error is minimized (alternative versions of MAP decoding were introduced by McAdam, Welch, and Weber [6], Lee [7], and Hartmann and Rudolph [8]); but the performance of these two algorithms in decoding convolutional codes is essentially identical. Because the Viterbi algorithm is simpler to implement, it is usually preferred in practice; however, for iterative decoding applications, such as the Shannon limit–approaching turbo codes to be discussed in Chapter 16, the BCJR algorithm is usually preferred. We also present a version of the Viterbi algorithm that produces soft outputs, the soft-output Viterbi algorithm (SOVA) introduced in 1989 by Hagenauer and Hoeher [9], for iterative decoding applications.

Union bounds on the performance of convolutional codes with ML decoding are developed in this chapter, and tables of optimal codes based on these bounds are given. The bounds make use of the weight-enumerating functions, or transfer functions, derived in Chapter 11. The application of transfer function bounds to the analysis of convolutional codes was also introduced by Viterbi [10]. This work laid the foundation for the evaluation of upper bounds on the performance of any convolutionally encoded system. The technique is used again to develop performance bounds in Chapter 16 on turbo coding and in Chapter 18 on trellis-coded modulation. (Tightened versions of the transfer function bound are given in [11, 12, 13].) Finally, the practically important code modification techniques of tail-biting and puncturing are introduced.

12.1 THE VITERBI ALGORITHM

To understand the Viterbi decoding algorithm, it is convenient to expand the state diagram of the encoder in time, that is, to represent each time unit with a separate state diagram. The resulting structure is a trellis diagram, first introduced for linear block codes in Chapter 9. An example is shown in Figure 12.1 for the (3, 1, 2) nonsystematic feedforward encoder with

$$\mathbf{G}(D) = [1 + D \quad 1 + D^2 \quad 1 + D + D^2] \tag{12.1}$$

and an information sequence of length $h = 5$. The trellis diagram contains $h+m+1 = 8$ time units or levels, and these are labeled from 0 to $h + m = 7$ in Figure 12.1. For a terminated code, assuming that the encoder always starts in state S_0 and returns to state S_0, the first $m = 2$ time units correspond to the encoder's departure from state S_0, and the last $m = 2$ time units correspond to the encoder's return to state S_0. It follows that not all states can be reached in the first m or the last m time units; however, in the center portion of the trellis, all states are possible, and each time unit contains a replica of the state diagram. There are $2^k = 2$ branches leaving and entering each state. The upper branch leaving each state at time unit i represents the input bit $u_i = 1$, and the lower branch represents $u_i = 0$. Each branch is labeled with the $n = 3$ corresponding outputs $\mathbf{v}_i$, and each of the $2^h = 32$ codewords of length $N = n(h + m) = 21$ is represented by a unique path through the trellis. For example, the codeword corresponding to the information sequence $\mathbf{u} = (1\ 1\ 1\ 0\ 1)$

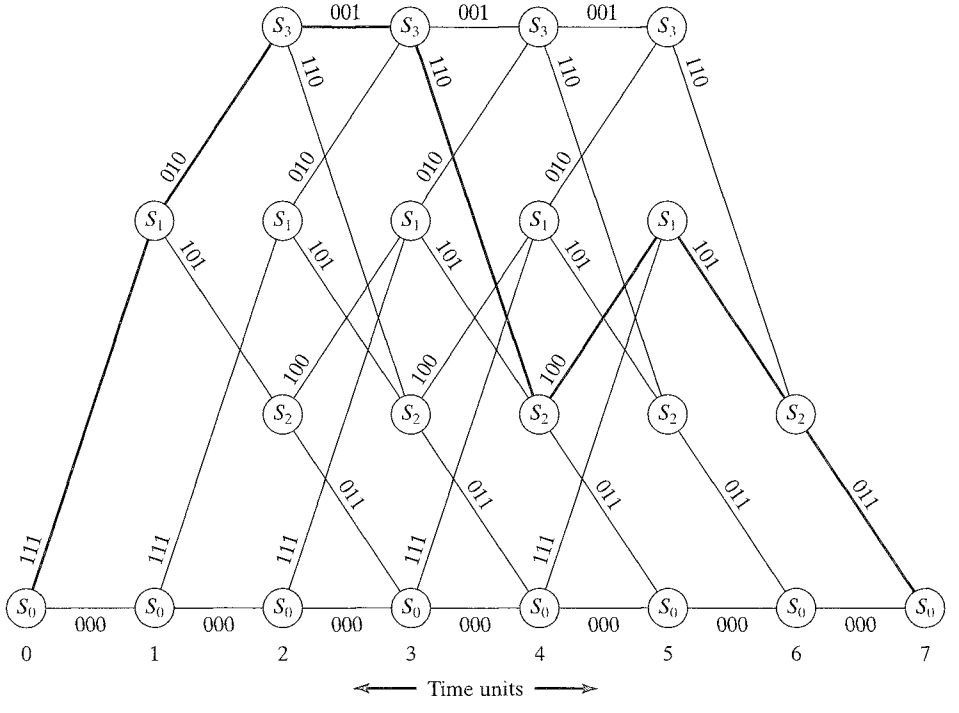


FIGURE 12.1: Trellis diagram for a (3, 1, 2) encoder with $h = 5$.

is shown highlighted in Figure 12.1. In the general case of an (n, k, ν) encoder and an information sequence of length $K^* = kh$, there are 2^k branches leaving and entering each state, and 2^{K^*} distinct paths through the trellis corresponding to the 2^{K^*} codewords.

Now, assume that an information sequence $\mathbb{u} = (\mathbb{u}_0, \cdots, \mathbb{u}_{h-1})$ of length $K^* = kh$ is encoded into a codeword $\mathbb{v} = (\mathbb{v}_0, \mathbb{v}_1, \cdots, \mathbb{v}_{h+m-1})$ of length $N = n(h + m)$ and that a Q-ary sequence $\mathbb{r} = (\mathbb{r}_0, \mathbb{r}_1, \cdots, \mathbb{r}_{h+m-1})$ is received over a binary-input, Q-ary output *discrete memoryless channel* (DMC). Alternatively, we can write these sequences as $\mathbb{u} = (u_0, u_1, \cdots, u_{K^*-1})$, $\mathbb{v} = (v_0, v_1, \cdots, v_{N-1})$, and $\mathbb{r} = (r_0, r_1, \cdots, r_{N-1})$, where the subscripts now simply represent the ordering of the symbols in each sequence. As discussed in Section 1.4, the decoder must produce an estimate $\hat{\mathbb{v}}$ of the codeword $\mathbb{v}$ based on the received sequence $\mathbb{r}$. A *maximum likelihood* (ML) *decoder* for a DMC chooses $\hat{\mathbb{v}}$ as the codeword $\mathbb{v}$ that maximizes the log-likelihood function $\log P(\mathbb{r}|\mathbb{v})$. Because for a DMC

$$P(\mathbb{r}|\mathbb{v}) = \prod_{l=0}^{h+m-1} P(\mathbb{r}_l|\mathbb{v}_l) = \prod_{l=0}^{N-1} P(r_l|v_l), \tag{12.2}$$

it follows that

$$\log P(\mathbb{r}|\mathbb{v}) = \sum_{l=0}^{h+m-1} \log P(\mathbb{r}_l|\mathbb{v}_l) = \sum_{l=0}^{N-1} \log P(r_l|v_l), \tag{12.3}$$

where $P(r_l|v_l)$ is a channel transition probability. This is a minimum error probability decoding rule when all codewords are equally likely.

The log-likelihood function $\log(\mathbb{r}|\mathbb{v})$, denoted by $M(\mathbb{r}|\mathbb{v})$, is called the *metric* associated with the path (codeword) $\mathbb{v}$. The terms $\log P(\mathbb{r}_l|\mathbb{v}_l)$ in the sum of (12.3) are called *branch metrics* and are denoted by $M(\mathbb{r}_l|\mathbb{v}_l)$, whereas the terms $\log P(r_l|v_l)$ are called *bit metrics* and are denoted by $M(r_l|v_l)$. Hence, we can write the path metric $M(\mathbb{r}|\mathbb{v})$ as

$$M(\mathbb{r}|\mathbb{v}) = \sum_{l=0}^{h+m-1} M(\mathbb{r}_l|\mathbb{v}_l) = \sum_{l=0}^{h+m-1} \log P(\mathbb{r}_l|\mathbb{v}_l) = \sum_{l=0}^{N-1} M(r_l|v_l) = \sum_{l=0}^{N-1} \log P(r_l|v_l). \tag{12.4}$$

We can now express a partial path metric for the first t branches of a path as

$$M([\mathbb{r}|\mathbb{v}]_t) = \sum_{l=0}^{t-1} M(\mathbb{r}_l|\mathbb{v}_l) = \sum_{l=0}^{t-1} \log P(\mathbb{r}_l|\mathbb{v}_l) = \sum_{l=0}^{nt-1} M(r_l|v_l) = \sum_{l=0}^{nt-1} \log P(r_l|v_l). \tag{12.5}$$

The following algorithm, when applied to the received sequence $\mathbb{r}$ from a DMC, finds the path through the trellis with the largest metric, that is, the *maximum likelihood path (codeword)*. The algorithm processes $\mathbb{r}$ in a recursive manner. At each time unit it adds 2^k branch metrics to each previously stored path metric

(the *add* operation), it compares the metrics of all 2^k paths entering each state (the *compare* operation), and it selects the path with the largest metric, called the *survivor* (the *select* operation). The survivor at each state is then stored along with its metric.

The Viterbi Algorithm

Step 1. Beginning at time unit $t = m$, compute the partial metric for the single path entering each state. Store the path (the survivor) and its metric for each state.

Step 2. Increase t by 1. Compute the partial metric for all 2^k paths entering a state by adding the branch metric entering that state to the metric of the connecting survivor at the previous time unit. For each state, compare the metrics of all 2^k paths entering that state, select the path with the largest metric (the survivor), store it along with its metric, and eliminate all other paths.

Step 3. If $t < h + m$, repeat step 2; otherwise, stop.

The basic computation performed by the Viterbi algorithm is the add, compare, select (ACS) operation of step 2. As a practical matter, the information sequence corresponding to the surviving path at each state, rather than the surviving path itself, is stored in steps 1 and 2, thus eliminating the need to invert the estimated codeword $\hat{\mathbf{v}}$ to recover the estimated information sequence $\hat{\mathbf{u}}$ when the algorithm finishes.

There are 2^ν survivors from time unit m through the time unit h, one for each of the 2^ν states. After time unit h there are fewer survivors, since there are fewer states while the encoder is returning to the all-zero state. Finally, at time unit $h + m$, there is only one state, the all-zero state, and hence only one survivor, and the algorithm terminates. We now prove that this final survivor is the maximum likelihood path.

Theorem 12.1 The final survivor $\hat{\mathbf{v}}$ in the Viterbi algorithm is the maximum likelihood path; that is,

$$M(\mathbf{r}|\hat{\mathbf{v}}) \geq M(\mathbf{r}|\mathbf{v}), \text{ all } \mathbf{v} \neq \hat{\mathbf{v}}. \tag{12.6}$$

Proof. Assume that the maximum likelihood path is eliminated by the algorithm at time unit t, as illustrated in Figure 12.2. This implies that the partial path metric of the survivor exceeds that of the maximum likelihood path at this point. Now, if the remaining portion of the maximum likelihood path is appended onto the survivor at time unit t, the total metric of this path will exceed the total metric of the maximum likelihood path; but this contradicts the definition of the maximum likelihood path as the path with the largest metric. Hence, the maximum likelihood path cannot be eliminated by the algorithm; that is, it must be the final survivor. **Q.E.D.**

Theorem 12.1 shows that the Viterbi algorithm is an optimum decoding algorithm in the sense that it always finds the maximum likelihood path through the trellis.

From an implementation point of view, it is more convenient to use positive integers as metrics rather than the actual bit metrics. The bit metric $M(r_l|v_l) = \log P(r_l|v_l)$ can be replaced by $c_2[\log P(r_l|v_l) + c_1]$, where c_1 is any real number

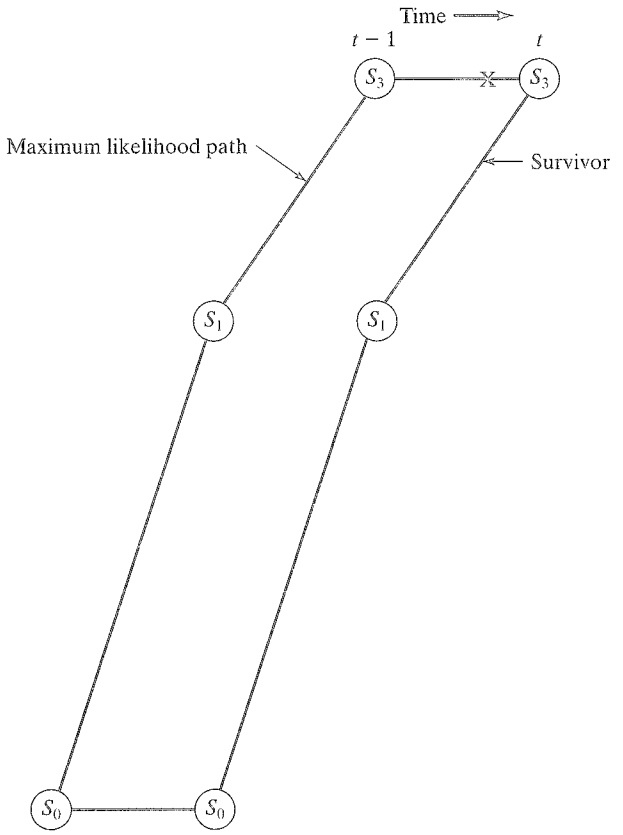


FIGURE 12.2: Elimination of the maximum likelihood path.

and c_2 is any positive real number. It can be shown (see Problem 12.2) that a path $\mathbf{v}$ that maximizes $M(\mathbf{r}|\mathbf{v}) = \sum_{l=0}^{N-1} M(r_l|v_l) = \sum_{l=0}^{N=1} \log P(r_l|v_l)$ also maximizes $\sum_{l=0}^{N-1} c_2[\log P(r_l|v_l) + c_1]$, and hence the modified metrics can be used without affecting the performance of the Viterbi algorithm. If c_1 is chosen to make the smallest metric 0, c_2 can then be chosen so that all metrics can be approximated by integers. There are many sets of integer metrics possible for a given DMC, depending on the choice of c_2 (see Problem 12.3). The performance of the Viterbi algorithm is now slightly suboptimum owing to the metric approximation by integers, but c_1 and c_2 can always be chosen such that the degradation is very slight. We now give two examples illustrating the operation of the Viterbi algorithm.

EXAMPLE 12.1 The Viterbi Algorithm for a Binary-Input, Quaternary-Output DMC

Consider the binary-input, quaternary-output ($Q = 4$) DMC shown in Figure 12.3. Using logarithms to the base 10, we display the bit metrics for this channel in a *metric table* in Figure 12.4(a). Choosing $c_1 = 1$ and $c_2 = 17.3$, we obtain the *integer metric table* shown in Figure 12.4(b). Now, assume that a codeword from the trellis

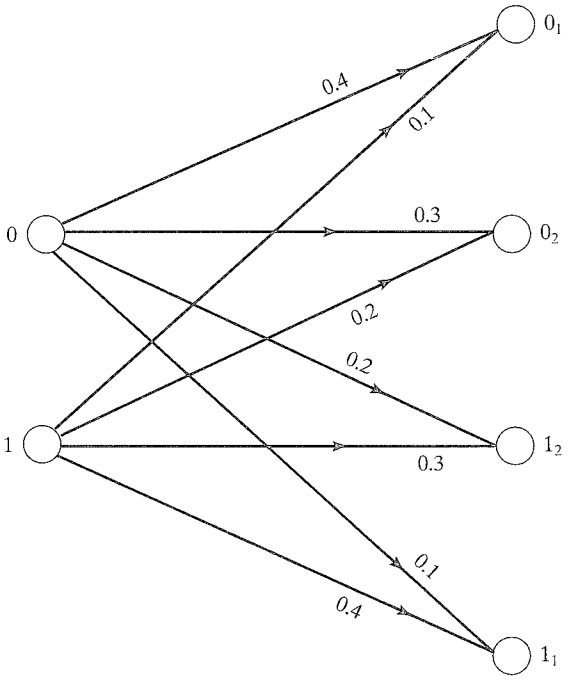


FIGURE 12.3: A binary-input, quaternary-output DMC.

v_l \ r_l	0_1	0_2	1_2	1_1
0	−0.4	−0.52	−0.7	−1.0
1	−1.0	−0.7	−0.52	−0.4

(a)

v_l \ r_l	0_1	0_2	1_2	1_1
0	10	8	5	0
1	0	5	8	10

(b)

FIGURE 12.4: Metric tables for the channel of Figure 12.3.

diagram of Figure 12.1 is transmitted over the DMC of Figure 12.3 and that the quaternary received sequence is given by

$$\mathbf{r} = (1_1 1_2 0_1,\ 1_1 1_1 0_2,\ 1_1 1_1 0_1,\ 1_1 1_1 1_1,\ 0_1 1_2 0_1,\ 1_2 0_2 1_1,\ 1_2 0_1 1_1). \tag{12.7}$$

The application of the Viterbi algorithm to this received sequence is shown in Figure 12.5. The numbers above each state represent the metric of the survivor for that state, and the paths eliminated at each state are shown crossed out on the trellis diagram. The final survivor,

$$\hat{\mathbf{v}} = (111,\ 010,\ 110,\ 011,\ 000,\ 000,\ 000), \tag{12.8}$$

is shown as the highlighted path. This surviving path corresponds to the decoded information sequence $\hat{\mathbf{u}} = (1\,1\,0\,0\,0)$. Note that the final $m = 2$ branches in any trellis

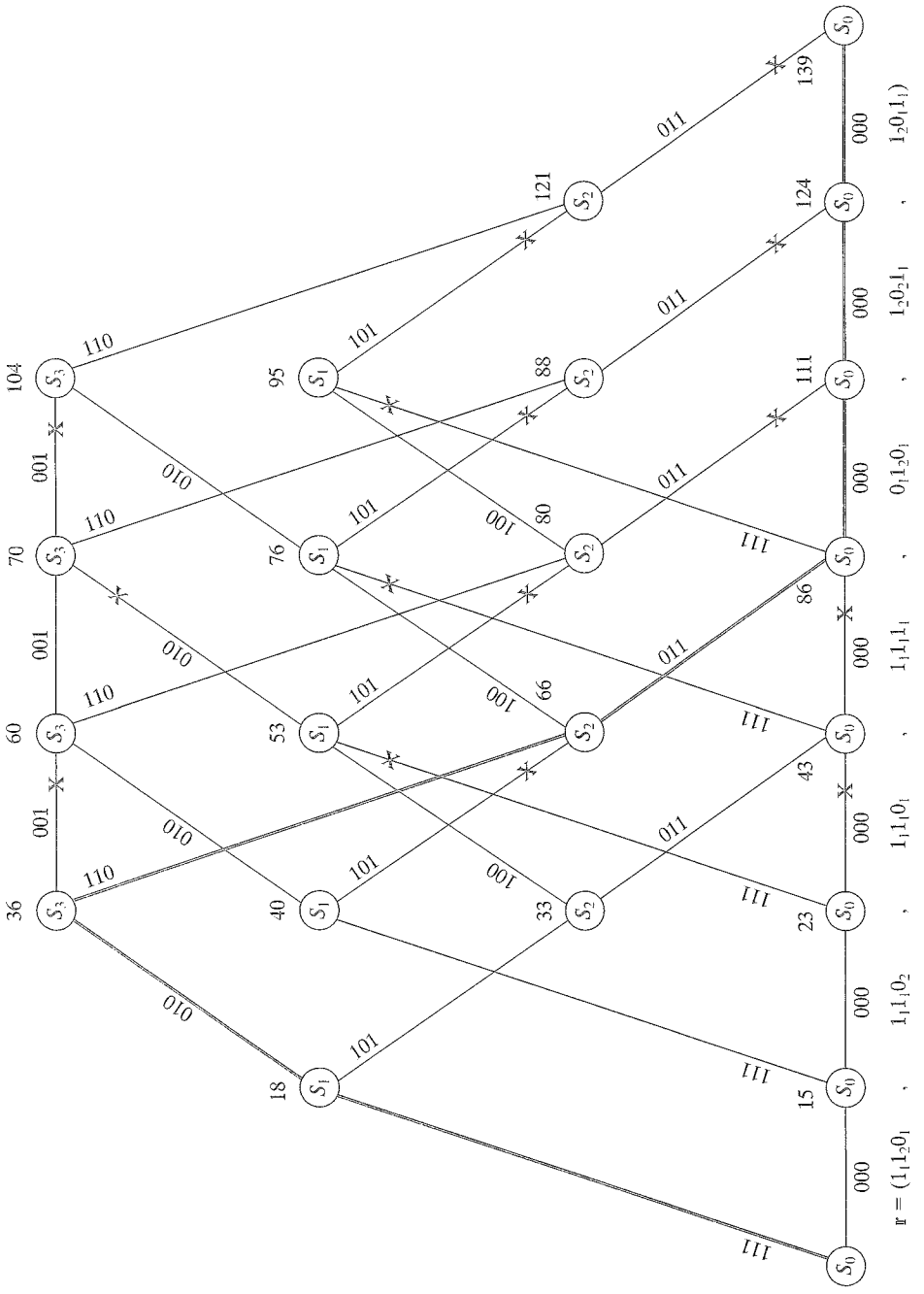


FIGURE 12.5: The Viterbi algorithm for a DMC.

path correspond to 0 inputs and hence are not considered part of the information sequence.

In the special case of a *binary symmetric channel* (BSC) with transition probability $p < 1/2$,[1] the received sequence $\mathbf{r}$ is binary ($Q = 2$) and the log-likelihood function becomes (see (1.11))

$$\log P(\mathbf{r}|\mathbf{v}) = d(\mathbf{r}, \mathbf{v}) \log \frac{p}{1-p} + N \log(1-p), \tag{12.9}$$

where $d(\mathbf{r}, \mathbf{v})$ is the Hamming distance between $\mathbf{r}$ and $\mathbf{v}$. Because $\log \frac{p}{1-p} < 0$ and $N \log(1-p)$ is a constant for all $\mathbf{v}$, an MLD for a BSC chooses $\mathbf{v}$ as the codeword $\hat{\mathbf{v}}$ that minimizes the Hamming distance

$$d(\mathbf{r}, \mathbf{v}) = \sum_{l=0}^{h+m-1} d(\mathbf{r}_l, \mathbf{v}_l) = \sum_{l=0}^{N-1} d(r_l, v_l). \tag{12.10}$$

Hence, when we apply the Viterbi algorithm to the BSC, $d(\mathbf{r}_l, \mathbf{v}_l)$ becomes the branch metric, $d(r_l, v_l)$ becomes the bit metric, and the algorithm must find the path through the trellis with the smallest metric, that is, the path closest to $\mathbf{r}$ in Hamming distance. The operation of the algorithm is exactly the same, except that the Hamming distance replaces the log-likelihood function as the metric, and the survivor at each state is the path with the *smallest* metric.

EXAMPLE 12.2 The Viterbi Algorithm for a BSC

An example of the application of the Viterbi algorithm to a BSC is shown in Figure 12.6. Assume that a codeword from the trellis diagram of Figure 12.1 is transmitted over a BSC and that the received sequence is given by

$$\mathbf{r} = (110, 110, 110, 111, 010, 101, 101). \tag{12.11}$$

The final survivor,

$$\hat{\mathbf{v}} = (111, 010, 110, 011, 111, 101, 011), \tag{12.12}$$

is shown as the highlighted path in the figure, and the decoded information sequence is $\hat{\mathbf{u}} = (11001)$. That the final survivor has a metric of 7 means that no other path through the trellis differs from $\mathbf{r}$ in fewer than seven positions. Note that at some states neither path is crossed out. This indicates a tie in the metric values of the two paths entering that state. If the final survivor goes through any of these states, then there is more than one maximum likelihood path, that is, more than one path whose distance from $\mathbf{r}$ is minimum. From an implementation point of view, whenever a tie in metric values occurs, one path is arbitrarily selected as the survivor, owing to the impracticality of storing a variable number of paths. This arbitrary resolution of ties has no effect on the decoding error probability.

[1] A BSC with $p > 1/2$ can always be converted to a BSC with $p < 1/2$ simply by reversing the output labels.

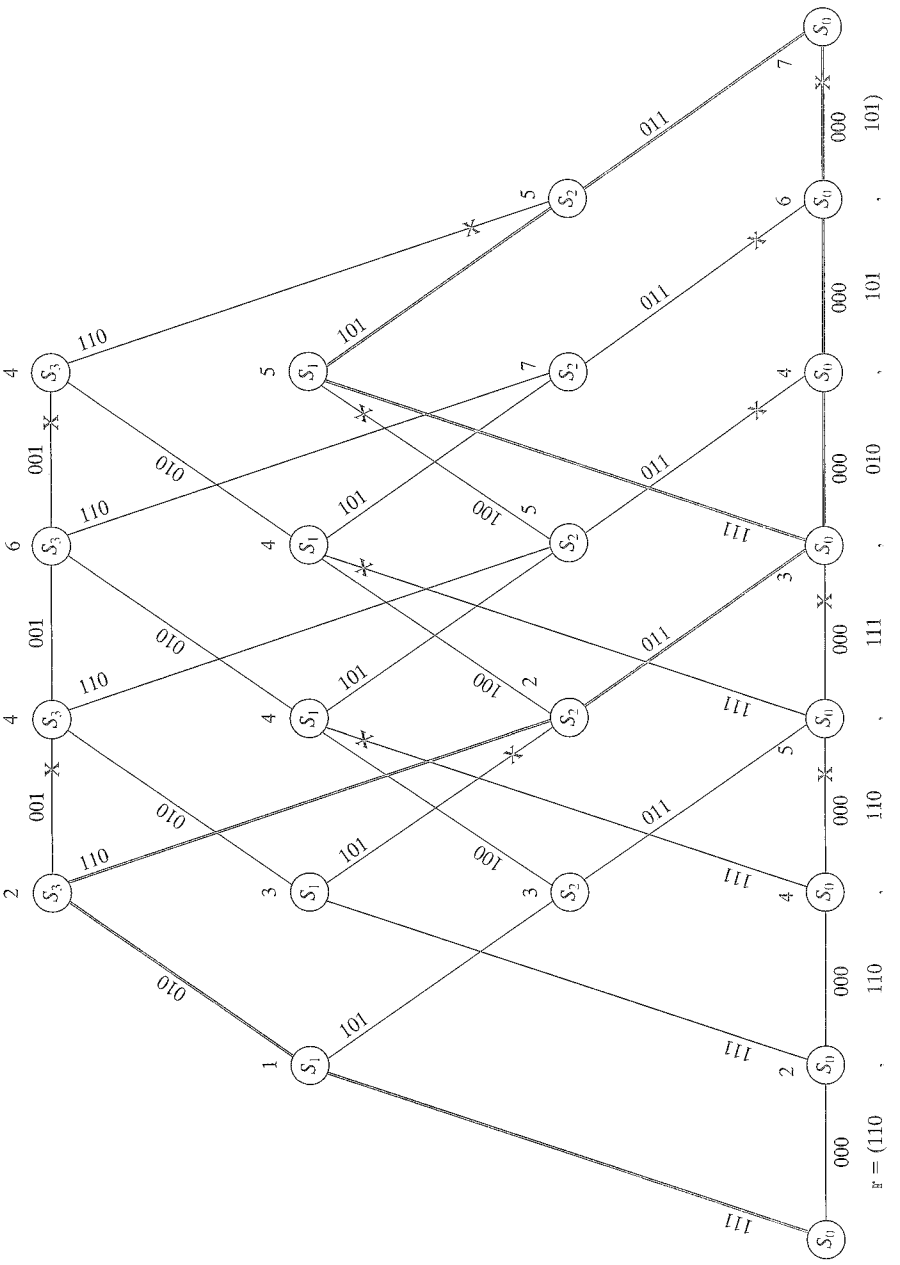


FIGURE 12.6: The Viterbi algorithm for a BSC.

Now, consider a binary-input, additive white Gaussian noise (AWGN) channel with no demodulator output quantization, that is, a binary-input, continuous-output channel. Assume the channel inputs 0 and 1 are represented by the BPSK signals (see (1.1)) $\pm\sqrt{\frac{2E_s}{T}}\cos(2\pi f_0 t)$, where we use the mapping $1 \rightarrow +\sqrt{E_s}$ and $0 \rightarrow -\sqrt{E_s}$.

Normalizing by $\sqrt{E_s}$, we consider the codeword $\mathbf{v} = (v_0, v_1, \cdots, v_{N-1})$ to take on values ± 1 according to the mapping $1 \rightarrow +1$ and $0 \rightarrow -1$, and the (normalized) received sequence $\mathbf{r} = (r_0, r_1, \cdots, r_{N-1})$ to be real-valued (unquantized). The conditional probability density function (pdf) of the (normalized) received symbol r_l given the transmitted bit v_l is

$$p(r_l|v_l) = \sqrt{\frac{E_s}{\pi N_0}} e^{\left[-\frac{(r_l\sqrt{E_s}-v_l\sqrt{E_s})^2}{N_0}\right]} = \sqrt{\frac{E_s}{\pi N_0}} e^{-(E_s/N_0)(r_l-v_l)^2}, \tag{12.13}$$

where N_0/E_s is the (normalized) one-sided noise power spectral density. If the channel is *memoryless*, the log-likelihood function of the received sequence $\mathbf{r}$ given the transmitted codeword $\mathbf{v}$ is

$$\begin{aligned} M(\mathbf{r}|\mathbf{v}) &= \ln p(\mathbf{r}|\mathbf{v}) = \ln \prod_{l=0}^{N-1} p(r_l|v_l) = \sum_{l=0}^{N-1} \ln p(r_l|v_l) \\ &= -\frac{E_s}{N_0} \sum_{l=0}^{N-1} (r_l - v_l)^2 + \frac{N}{2} \ln \frac{E_s}{\pi N_0} \\ &= -\frac{E_s}{N_0} \sum_{l=0}^{N-1} (r_l^2 - 2r_l v_l + 1) + \frac{N}{2} \ln \frac{E_s}{\pi N_0} \\ &= (2\frac{E_s}{N_0}) \sum_{l=0}^{N-1} (r_l v_l) - \frac{E_s}{N_0}(|\mathbf{r}|^2 + N) + \frac{N}{2} \ln \frac{E_s}{\pi N_0} \\ &= C_1(\mathbf{r}\cdot\mathbf{v}) + C_2, \end{aligned} \tag{12.14}$$

where $C_1 = (2E_s/N_0)$ and $C_2 = -[(E_s/N_0)(|\mathbf{r}|^2 + N) - (N/2)\ln(E_s/\pi N_0)]$ are constants independent of the codeword $\mathbf{v}$ and $\mathbf{r}\cdot\mathbf{v}$ represents the inner product (correlation) of the received vector $\mathbf{r}$ and the codeword $\mathbf{v}$. Because C_1 is positive, the trellis path (codeword) that maximizes the correlation $\mathbf{r}\cdot\mathbf{v}$ also maximizes the log-likelihood function $\ln p(\mathbf{r}|\mathbf{v})$. It follows that the path metric corresponding to the codeword $\mathbf{v}$ is given by $M(\mathbf{r}|\mathbf{v}) = \mathbf{r}\cdot\mathbf{v}$, the branch metrics are $M(\mathbf{r}_l|\mathbf{v}_l) = \mathbf{r}_l\cdot\mathbf{v}_l$, $l = 0, 1, \cdots, h + m - 1$, the bit metrics are $M(r_l|v_l) = r_l v_l$, $l = 0, 1, \cdots, N - 1$, and the Viterbi algorithm finds the path through the trellis that maximizes the correlation with the received sequence. (It is important to remember in this case that the (real-valued) received sequence $\mathbf{r}$ is correlated with trellis paths (codewords) $\mathbf{v}$ labeled according to the mapping $1 \rightarrow +1$ and $0 \rightarrow -1$.)

An interesting analogy with the binary symmetric channel is obtained by representing the (real-valued) received sequence and the codewords as vectors in N-dimensional Euclidean space. (In this case, the N-bit codewords with components ± 1 all lie on the vertices of an N-dimensional hypersphere.) Then, for a continuous-output AWGN channel, maximizing the log-likelihood function is equivalent to finding the codeword $\mathbf{v}$ that is closest to the received sequence $\mathbf{r}$ in Euclidean distance (see, e.g., [12] or [14]). In the BSC case, on the other hand, maximizing the log-likelihood function is equivalent to finding the (binary) codeword $\mathbf{v}$ that is closest to the (binary) received sequence $\mathbf{r}$ in Hamming distance (see (12.9)). An

example of the application of the Viterbi algorithm to a continuous-output AWGN channel is given in Problem 12.7.

In Section 1.3 we noted that making soft demodulator decisions ($Q > 2$) results in a performance advantage over making hard decisions ($Q = 2$). The preceding two examples of the application of the Viterbi algorithm serve to illustrate this point. If the quaternary outputs 0_1 and 0_2 are converted into a single output 0, and 1_1 and 1_2 are converted into a single output 1, the soft-decision DMC is converted to a hard-decision BSC with transition probability $p = 0.3$. In the preceding examples the sequence $\mathbb{r}$ in the hard-decision case is the same as in the soft-decision case with 0_1 and 0_2 converted to 0 and 1_1 and 1_2 converted to 1; but the Viterbi algorithm yields different results in the two cases. In the soft-decision case ($Q = 4$), the information sequence $\mathbb{u} = (1\ 1\ 0\ 0\ 0)$ produces the maximum likelihood path, which has a final metric of 139. In the hard-decision case ($Q = 2$), however, the maximum likelihood path is $\mathbb{u} = (1\ 1\ 0\ 0\ 1)$. The metric of this path on the quaternary output channel is 135, and so it is not the maximum likelihood path in the soft-decision case; however, since hard decisions mask the distinction between certain soft-decision outputs—for example, outputs 0_1 and 0_2 are treated as equivalent outputs in the hard-decision case—the hard-decision decoder makes an estimate that would not be made if more information about the channel were available, that is, if soft decisions were made. (For another example of the difference between soft-decision and hard-decision decoding, see Problems 12.4–12.6.)

As a final comment, both of the preceding channels can be classified as "very noisy" channels. The code rate $R = 1/2$ exceeds the channel capacity C in both cases. Hence, we would not expect the performance of this code to be very good with either channel, as reflected by the relatively low value (139) of the final metric of the maximum likelihood path in the DMC case, as compared with a maximum possible metric of 210 for a path that "agrees" completely with $\mathbb{r}$. Also, in the BSC case, the final Hamming distance of 7 for the maximum likelihood path is large for paths only 21 bits long. Lower code rates would be needed to achieve good performance over these channels. The performance of convolutional codes with Viterbi decoding is the subject of the next section.

12.2 PERFORMANCE BOUNDS FOR CONVOLUTIONAL CODES

We begin this section by analyzing the performance of maximum likelihood (Viterbi) decoding for a specific code on a BSC. We will discuss more general channel models later. First, assume, without loss of generality, that the all-zero codeword $\mathbb{v} = \mathbb{0}$ is transmitted from the (3, 1, 2) encoder of (12.1). The IOWEF of this encoder (see Problem 11.19) is given by

$$\begin{aligned} A(W, X, L) &= \frac{X^7WL^3}{1 - XWL(1 + X^2L)} \\ &= X^7WL^3[1 + XWL(1 + X^2L) + X^2W^2L^2(1 + X^2L^2) + \cdots] \\ &= X^7WL^3 + X^8W^2L^4 + X^9W^3L^5 + X^{10}(W^2L^5 + W^4L^6) + \cdots; \end{aligned} \tag{12.15}$$

that is, the code contains one codeword of weight 7 and length 3 branches generated by an information sequence of weight 1, one codeword of weight 8 and length 4 branches generated by an information sequence of weight 2, and so on.

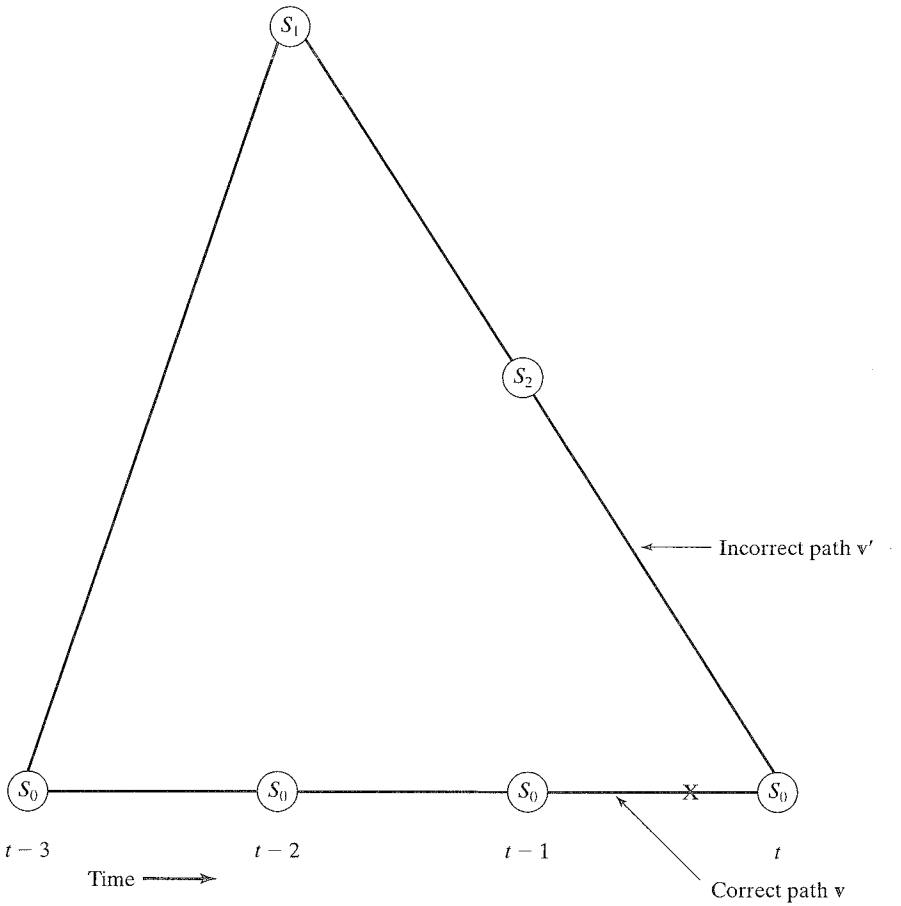


FIGURE 12.7: A first event error at time unit t.

We say that a *first event error* is made at an arbitrary time unit t if the all-zero path (the *correct path*) is eliminated *for the first time* at time unit t in favor of a competitor (the *incorrect path*). This situation is illustrated in Figure 12.7, where the correct path $\mathbf{v}$ is eliminated by the incorrect path $\mathbf{v}'$ at time unit t. The incorrect path must be some path that previously diverged from the all-zero state and is now remerging for the first time at time t; that is, it must be one of the paths enumerated by the codeword WEF A(X) of the code. Assuming it is the weight-7 path, a first event error will be made if, in the seven positions in which the correct and incorrect paths differ, the binary received sequence $\mathbf{r}$ agrees with the incorrect path in four or more of these positions, that is, if $\mathbf{r}$ contains four or more 1's in these seven positions. If the BSC transition probability is p, this probability is

$$\begin{aligned} P_7 &= P[\text{four or more 1's in seven positions}] \\ &= \sum_{e=4}^{7} \binom{7}{3} p^e (1-p)^{7-e}. \end{aligned} \tag{12.16}$$

Assuming that the weight-8 path is the incorrect path, a first event error is made with probability

$$P_8 = \frac{1}{2}\binom{8}{4}p^4(1-p)^4 + \sum_{e=5}^{8}\binom{8}{e}p^e(1-p)^{8-e}, \tag{12.17}$$

since if the metrics of the correct and incorrect paths are tied, an error is made with probability 1/2. In general, assuming the incorrect path has weight d, a first event error is made with probability

$$P_d = \begin{cases} \displaystyle\sum_{e=\frac{d+1}{2}}^{d}\binom{d}{e}p^e(1-p)^{d-e}, & d \text{ odd} \\ \displaystyle\frac{1}{2}\binom{d}{d/2}p^{d/2}(1-p)^{d/2} + \sum_{e=\frac{d}{2}+1}^{d}\binom{d}{e}p^e(1-p)^{d-e}, & d \text{ even.} \end{cases} \tag{12.18}$$

Because all incorrect paths of length t branches or less can cause a first event error at time unit t, the *first event error probability at time unit t*, $P_f(E,t)$, can be overbounded, using a union bound, by the sum of the error probabilities of each of these paths. If all incorrect paths of length greater than t branches are also included, $P_f(E,t)$ is overbounded by

$$P_f(E,t) < \sum_{d=d_{free}}^{\infty} A_d P_d, \tag{12.19}$$

where A_d is the number of codewords of weight d (i.e., it is the coefficient of the weight-d term in the codeword WEF A(X) of the code). Because this bound is independent of t, it holds for all time units, and the *first event error probability* at any time unit, $P_f(E)$, is bounded by

$$P_f(E) < \sum_{d=d_{free}}^{\infty} A_d P_d. \tag{12.20}$$

The bound of (12.20) can be further simplified by noting that for d odd,

$$\begin{aligned} P_d &= \sum_{e=\frac{d+1}{2}}^{d}\binom{d}{e}p^e(1-p)^{d-e} \\ &< \sum_{e=\frac{d+1}{2}}^{d}\binom{d}{e}p^{d/2}(1-p)^{d/2} \\ &= p^{d/2}(1-p)^{d/2}\sum_{e=\frac{d+1}{2}}^{d}\binom{d}{e} \\ &< p^{d/2}(1-p)^{d/2}\sum_{e=0}^{d}\binom{d}{e} \\ &= 2^d p^{d/2}(1-p)^{d/2}. \end{aligned} \tag{12.21}$$

It can also be shown (see Problem 12.9) that (12.21) is an upper bound on P_d for d even. Hence,

$$P_f(E) < \sum_{d=d_{free}}^{\infty} A_d \left[2\sqrt{p(1-p)}\right]^d, \tag{12.22}$$

and for any convolutional code with codeword WEF $A(X) = \sum_{d=d_{free}}^{\infty} A_d X^d$, it follows by comparing (12.22) with the expression for $A(X)$ that

$$P_f(E) < A(X)|_{X=2\sqrt{p(1-p)}}. \tag{12.23}$$

The final decoded path can diverge from and remerge with the correct path any number of times; that is, it can contain any number of event errors, as illustrated in Figure 12.8. After one or more event errors have occurred, the two paths compared at the all-zero state will both be incorrect paths, one of which contains at least one previous event error. This situation is illustrated in Figure 12.9, where it is assumed

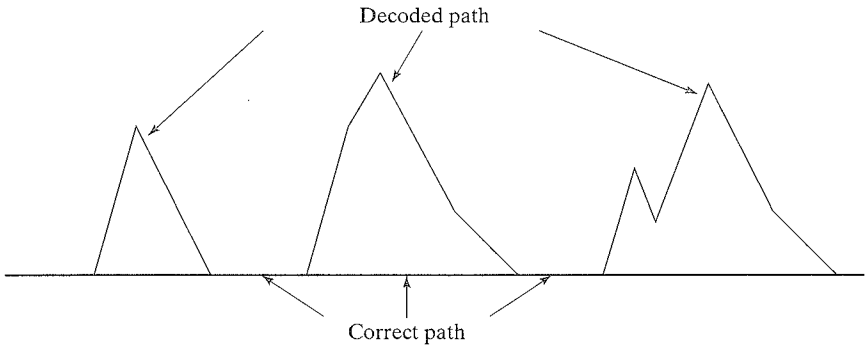


FIGURE 12.8: Multiple error events.

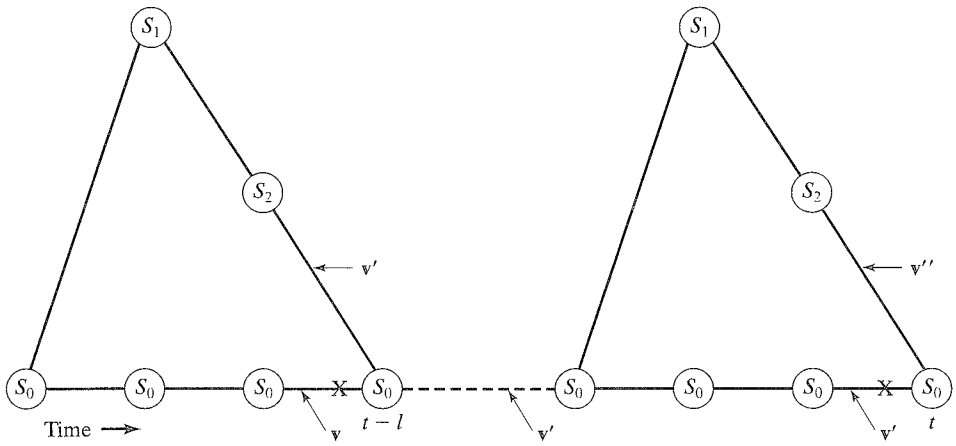


FIGURE 12.9: Comparison of two incorrect paths.

that the correct path $\mathbf{v}$ has been eliminated for the first time at time unit $t - l$ by incorrect path $\mathbf{v}'$, and that at time unit t incorrect paths $\mathbf{v}'$ and $\mathbf{v}''$ are compared. If the partial metric for $\mathbf{v}''$ exceeds the partial metric for $\mathbf{v}'$ at time unit t, then it must also exceed the partial metric for $\mathbf{v}$, since $\mathbf{v}'$ has a better metric than $\mathbf{v}$ at time t owing to the first event error at time $t - l$. Hence, if $\mathbf{v}''$ were compared with $\mathbf{v}$ at time unit t, a first event error would be made. We say that an event error occurs at time unit t if $\mathbf{v}''$ survives over $\mathbf{v}'$, and the *event-error probability at time unit t*, $P(E, t)$, is bounded by

$$P(E, t) \leq P_f(E, t), \tag{12.24}$$

since if $\mathbf{v}''$ survives over $\mathbf{v}'$ at time unit t, then it would also survive if compared with $\mathbf{v}$. In other words, the event that $\mathbf{v}''$ has a better metric than $\mathbf{v}'$ at time t (with probability $P(E, t)$) is contained in the event that $\mathbf{v}''$ has a better metric than $\mathbf{v}$ at time t (with probability $P_f(E, t)$).

The situation illustrated in Figure 12.9 is not the only way in which an event error at time unit t can follow a first event error made earlier. Two other possibilities are shown in Figure 12.10. In these cases, the event error made at time unit t either totally or partially replaces the event error made previously. Using the same arguments, it follows that (12.24) holds for these cases also, and hence it is a valid bound for any event error occurring at time unit t.[2]

The bound of (12.19) can be applied to (12.24). Because it is independent of t, it holds for all time units, and the *event-error probability* at any time unit, $P(E)$, is bounded by

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d P_d < \sum_{d=d_{free}}^{\infty} A_d \left[2\sqrt{p(1-p)}\right]^d = A(X)|_{X=2\sqrt{p(1-p)}}, \tag{12.25}$$

just as in (12.23). For small p, the bound is dominated by its first term, that is, the free distance term, and the event-error probability can be approximated as

$$P(E) \approx A_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}} \approx A_{d_{free}} 2^{d_{free}} p^{d_{free}/2}. \tag{12.26}$$

EXAMPLE 12.3 Evaluating the Event-Error Probability

For the (3, 1, 2) encoder of (12.1), $d_{free} = 7$ and $A_{d_{free}} = 1$. Thus for $p = 10^{-2}$, we obtain

$$P(E) \approx 2^7 p^{7/2} = 1.28 \times 10^{-5}. \tag{12.27}$$

The event-error probability bound of (12.25) can be modified to provide a bound on the *bit-error probability*, $P_b(E)$, that is, the expected number of information bit errors per decoded information bit. Each event error causes a number of information bit errors equal to the number of nonzero information bits

[2] In the two cases shown in Figure 12.10, the event error at time unit t replaces at least a portion of a previous error event. The net effect may be a decrease in the total number of decoding errors; that is, the number of positions in which the decoded path differs from the correct path. Hence, using the first event error probability as a bound may be conservative in some cases.

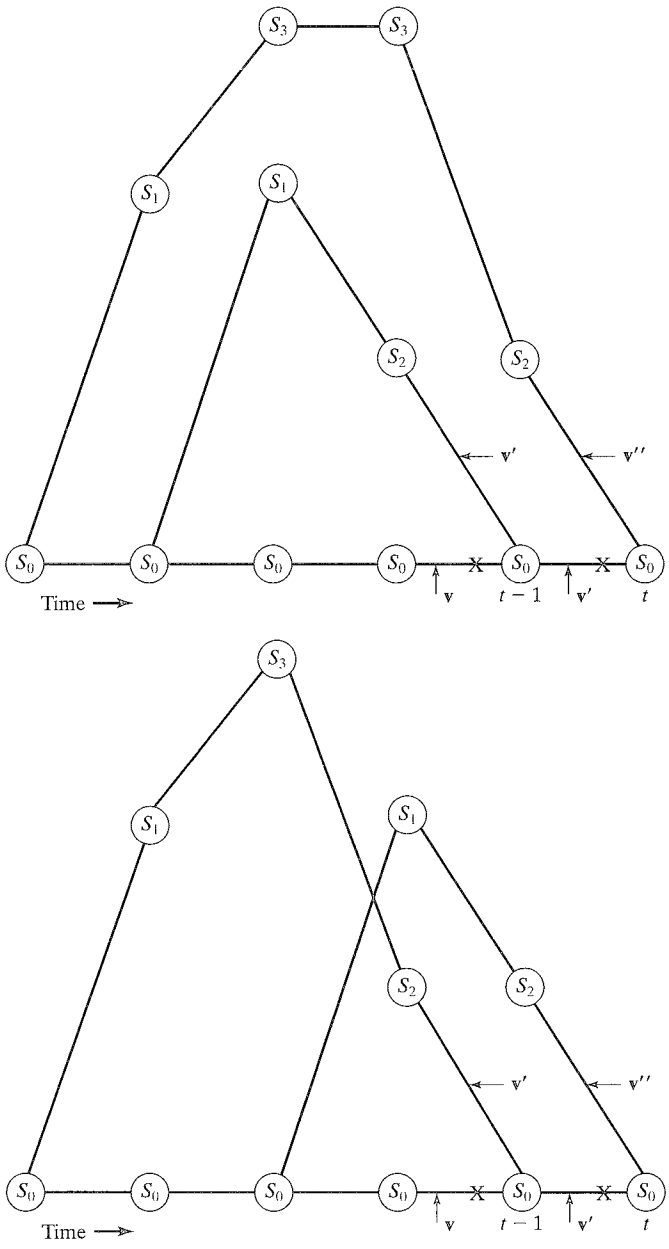


FIGURE 12.10: Other error event configurations.

on the incorrect path. Hence, if each event error probability term P_d is weighted by the number of nonzero information bits on the weight-d path, or if there is more than one weight-d path, by the total number of nonzero information bits on all weight-d paths, a bound on the expected number of information bit errors made at any time unit results. This bound can then be divided by k, the number of information bits

per unit time, to obtain a bound on $P_b(E)$. In other words, the bit-error probability is bounded by

$$P_b(E) < \sum_{d=d_{free}}^{\infty} B_d P_d, \tag{12.28}$$

where B_d is the total number of nonzero information bits on all weight-d paths, divided by the number of information bits k per unit time (i.e., it is the coefficient of the weight-d term in the bit WEF $B(X) = \sum_{d=d_{free}}^{\infty} B_d X^d$ of the encoder). Now, using (12.21) in (12.28) and comparing that bound with the expression for $B(X)$, we see that

$$P_b(E) < \sum_{d=d_{free}}^{\infty} B_d P_d < \sum_{d=d_{free}}^{\infty} B_d \left[2\sqrt{p(1-p)}\right]^d = B(X)|_{X=2\sqrt{p(1-p)}} \tag{12.29}$$

for any convolutional encoder with bit WEF $B(X)$. (We note here that the bound of (12.25) on the event-error probability $P(E)$ depends only on the code, whereas the bound of (12.29) on the bit-error probability $P_b(E)$ depends on the encoder.) For small p, the bound of (12.29) is dominated by its first term, so that

$$P_b(E) \approx B_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}} \approx B_{d_{free}} 2^{d_{free}} p^{d_{free}/2}. \tag{12.30}$$

EXAMPLE 12.4 Evaluating the Bit-Error Probability

For the (3, 1, 2) encoder of (12.1), $B_{d_{free}} = 1$, and $d_{free} = 7$. Thus, for $p = 10^{-2}$, we obtain

$$P_b(E) \approx 2^7 p^{7/2} = 1.28 \times 10^{-5}, \tag{12.31}$$

the same as for the event-error probability. In other words, when p is small, the most likely error event is that the weight-7 path is decoded instead of the all-zero path, thereby causing one information bit error. Typically, then, each event error causes one bit error, which is reflected in the approximate expressions for $P(E)$ and $P_b(E)$.

Slightly tighter versions of the bounds on $P(E)$ and $P_b(E)$ given in (12.25) and (12.29), respectively, have been derived in [11] and [13].

If the BSC is derived from an AWGN channel with BPSK modulation, optimum coherent detection, and binary-output quantization (hard decisions), then from (1.4) we have

$$p = Q\left(\sqrt{\frac{2E_s}{N_0}}\right). \tag{12.32}$$

Using the bound of (1.5) as an approximation yields

$$p \approx \tfrac{1}{2} e^{-E_s/N_0}, \tag{12.33}$$

and for a convolutional code with free distance d_{free},

$$P_b(E) \approx B_{d_{free}} 2^{d_{free}/2} e^{-(d_{free}/2)(E_s/N_0)} \tag{12.34}$$

when p is small, that is, when the channel SNR E_s/N_0 is large. Defining the *energy per information bit*, E_b, as

$$E_b \triangleq \frac{E_s}{R}, \tag{12.35}$$

and noting that $\frac{1}{R}$ is the number of transmitted symbols per information bit, we can write

$$P_b(E) \approx B_{d_{free}} 2^{d_{free}/2} e^{-(Rd_{free}/2)(E_b/N_0)} \quad \text{(with coding)} \tag{12.36}$$

for large bit SNR E_b/N_0.

On the other hand, if no coding is used, that is, $R = 1$ and $E_s = E_b$, the BSC transition probability p is the bit-error probability $P_b(E)$ and

$$P_b(E) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \approx \frac{1}{2} e^{-E_b/N_0} \quad \text{(without coding)}. \tag{12.37}$$

Comparing (12.36) and (12.37), we see that for a fixed E_b/N_0 ratio, the (negative) exponent with coding is larger by a factor of $Rd_{free}/2$ than the exponent without coding. Because the exponential term dominates the error probability expressions for large E_b/N_0, the factor $Rd_{free}/2$, in decibels, is called the *asymptotic coding gain* γ:

$$\gamma \triangleq 10 \log_{10}\left(\frac{Rd_{free}}{2}\right) \text{ dB} \tag{12.38}$$

in the hard-decision case. It is worth noting that coding gains become smaller as E_b/N_0 becomes smaller. In fact, if E_b/N_0 is reduced to the point at which the code rate R is greater than the channel capacity C, reliable communication with coding is no longer possible, and an uncoded system will outperform a coded system. This point is illustrated in Problem 12.12.

Bounds similar to (12.25) and (12.29) can also be obtained for more general channel models than the BSC. For a binary-input AWGN channel with finite (Q-ary) output quantization (a DMC), the bounds become

$$P(E) < A(X)|_{X=D_0} \tag{12.39a}$$

and

$$P_b(E) < B(X)|_{X=D_0}, \tag{12.39b}$$

where $D_0 \triangleq \sum_{(0 \le j \le Q-1)} \sqrt{P(j|0)P(j|1)}$ is a function of the channel transition probabilities called the *Bhattacharyya parameter*. Note that when $Q = 2$, a BSC results, and $D_0 = 2\sqrt{p(1-p)}$. Complete derivations of slightly tighter versions of these bounds can be found in [12].

In the case of a binary-input AWGN channel with no output quantization, that is, a continuous-output AWGN channel, we again assume that the all-zero codeword $\mathbf{v}$ is transmitted. In other words, $\mathbf{v} = (-1, -1, \cdots, -1)$ according to the mapping $0 \to -1$ and $1 \to +1$. Now, we calculate the probability P_d that the correct path $\mathbf{v}$ is eliminated for the first time at time unit t by an incorrect path $\mathbf{v}'$ that differs from $\mathbf{v}$ in d positions. Because the Viterbi algorithm always selects the path with

the largest value of the log-likelihood function $\log p(\mathbb{r}|\mathbb{v})$ (see (12.14)), a first event error at time unit t is made with probability

$$\begin{aligned} P_d &= Pr\{M([\mathbb{r}|\mathbb{v}']_t) > M([\mathbb{r}|\mathbb{v}]_t)\} = Pr\{[\mathbb{r}\cdot\mathbb{v}']_t > [\mathbb{r}\cdot\mathbb{v}]_t\} \\ &= Pr\{[\mathbb{r}\cdot\mathbb{v}']_t - [\mathbb{r}\cdot\mathbb{v}]_t > 0\} = Pr\left\{\sum_{l=0}^{t-1} \mathbb{r}_l\cdot\mathbb{v}'_l - \sum_{l=0}^{t-1} \mathbb{r}_l\cdot\mathbb{v}_l > 0\right\} \\ &= Pr\left\{\sum_{l=0}^{nt-1} r_l v'_l - \sum_{l=0}^{nt-1} r_l v_l > 0\right\}, \end{aligned} \tag{12.40}$$

where $[\mathbb{r}\cdot\mathbb{v}]_t$ represents the inner product of $\mathbb{r}$ and $\mathbb{v}$ over their first t branches. (Note that since we are now dealing with real numbers, the event $M([\mathbb{r}|\mathbb{v}']_t) = M([\mathbb{r}|\mathbb{v}]_t)$ has probability zero.) Clearly, the difference in the sums in (12.40) is nonzero only in the d positions where $v'_l \neq v_l$. Without loss of generality, let $l = 1, 2, \cdots, d$ represent those d positions. Then, recalling that in the d positions where $v'_l \neq v_l$, $v'_l = +1$ and $v_l = -1$, we can express (12.40) as

$$\begin{aligned} P_d &= Pr\left\{\sum_{l=1}^{d}(+r_l) - \sum_{l=1}^{d}(-r_l) > 0\right\} = Pr\left\{2\sum_{l=1}^{d} r_l > 0\right\} \\ &= Pr\left\{\sum_{l=1}^{d} r_l > 0\right\}. \end{aligned} \tag{12.41}$$

Equation (12.41) can be interpreted to mean that a first event error at time unit t occurs if the sum of the received symbol values in the d positions where $v'_l \neq v_l$ is positive. In other words, if the received sequence $\mathbb{r}$ in these d positions looks more like the incorrect path $\mathbb{v}'$ than the correct path $\mathbb{v}$, a first event error is made.

Because the channel is memoryless, and the transmitted codeword is assumed to be $\mathbb{v} = (-1, -1, \cdots, -1)$, $\rho \triangleq \sum_{l=1}^{d} r_l$ is a sum of d independent Gaussian random variables, each with mean -1 and variance $N_0/2E_s$ (see (12.13)); that is, ρ is a Gaussian random variable with mean $-d$ and variance $dN_0/2E_s$ (see, e.g., [11] or [14]). Thus, we can write (12.41) as

$$P_d = Pr\{\rho > 0\} = \left(\sqrt{\frac{E_s}{\pi d N_0}}\right) \int_0^\infty e^{\frac{-E_s(\rho+d)^2}{dN_0}} d\rho, \tag{12.42}$$

and with the substitution $y = (\rho + d)\sqrt{2E_s/dN_0}$, (12.42) becomes

$$\begin{aligned} P_d &= \frac{1}{\sqrt{2\pi}} \int_{+d\sqrt{2E_s/dN_0}}^{\infty} e^{-\frac{y^2}{2}} dy \\ &= \frac{1}{\sqrt{2\pi}} \int_{+\sqrt{2dE_s/N_0}}^{\infty} e^{-\frac{y^2}{2}} dy \\ &= Q\left(\sqrt{\frac{2dE_s}{N_0}}\right) = Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right), \end{aligned} \tag{12.43}$$

where $Q(x)$ is the familiar complementary error function of Gaussian statistics. Now, substituting (12.43) into the bounds of (12.25) and (12.29), we obtain the following expressions for the event- and bit-error probabilities of a binary-input, continuous-output AWGN channel:

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right) \tag{12.44a}$$

$$P_b(E) < \sum_{d=d_{free}}^{\infty} B_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right). \tag{12.44b}$$

Using the (slightly weaker than (1.5)) bound $Q(x) < exp(-x^2/2)$ in (12.44), we obtain the expressions

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d e^{-\frac{dRE_b}{N_0}} = A(X)|_{X=\exp(-RE_b/N_0)} \tag{12.45a}$$

$$P_b(E) < \sum_{d=d_{free}}^{\infty} B_d e^{-\frac{dRE_b}{N_0}} = B(X)|_{X=\exp(-RE_b/N_0)}. \tag{12.45b}$$

Comparing (12.45) with (12.39), we see that in the case of a binary-input AWGN channel with no output quantization, that is, a continuous-output AWGN channel, the Bhattacharyya parameter is given by $D_0 = \exp(-RE_b/N_0)$.

It is instructive to compare the approximate expression for $P_b(E)$ given in (12.36) for a BSC with a similar expression obtained for a binary-input, continuous-output AWGN channel from (12.45b). For large E_b/N_0, the first term in the bit WEF dominates the bound of (12.45b), and we can approximate $P_b(E)$ as

$$P_b(E) \approx B_{d_{free}}(e^{-RE_b/N_0})^{d_{free}} = B_{d_{free}} e^{-Rd_{free}E_b/N_0}. \tag{12.46}$$

Comparing the exponent of (12.46) with that of (12.36), we see that the exponent of (12.46) is larger by a factor of 2. This difference is equivalent to a 3-dB energy (or power) advantage for the continuous-output AWGN channel over the BSC, since to achieve the same error probability on the BSC, the transmitter must generate an additional 3 dB of signal energy (or power). This energy advantage illustrates the benefits of allowing soft decisions, that is, an unquantized demodulator output, instead of making hard decisions. The asymptotic coding gain in the soft-decision case is given by

$$\gamma \triangleq 10 \log_{10}(Rd_{free}) \text{ dB}, \tag{12.47}$$

an increase of 3 dB over the hard-decision case. The decoder complexity increases, however, owing to the need to accept real-valued inputs.

The foregoing analysis is based on performance bounds for specific codes and is valid only for large values of E_b/N_0. A similar comparison of soft decisions with finite-output quantization ($Q > 2$) and hard decisions can be made by computing the approximate expression for (12.39b) for a particular binary-input DMC and comparing with (12.30) for a BSC. Generally, it is found that $Q = 8$ allows one to achieve a performance within about 0.25 dB of the optimum performance achievable

with an unquantized demodulator output while avoiding the need for a decoder that accepts real-valued inputs.

A random coding analysis has also been used to demonstrate the advantages of soft decisions over hard decisions [12, 14]. For small values of E_b/N_0, this analysis shows that there is about a 2-dB penalty in signal power attached to the use of hard decisions; that is, to achieve the same error probability 2 dB more signal power must be generated at the transmitter when the demodulator output is hard quantized rather than unquantized. Over the entire range of E_b/N_0 ratios, the decibel loss associated with hard decisions is between 2 dB and 3 dB. Hence, the use of soft decisions is preferred in many applications as a means of regaining the 2–3-dB loss owing to hard quantization, at a cost of some increase in decoding complexity.

Somewhat tighter versions of the bounds in (12.45) can be found using the inequality

$$Q(\sqrt{y+z}) \le Q(\sqrt{y})e^{-\frac{z}{2}} \quad (y > 0, z \ge 0). \tag{12.48}$$

Setting $y = 2d_{free}RE_b/N_0$ and $z = 2(d - d_{free})RE_b/N_0$, we can use (12.48) to write (12.43) as

$$\begin{aligned} P_d &= Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right) = Q\left(\sqrt{y+z}\right) \\ &\le Q\left(\sqrt{\frac{2d_{free}RE_b}{N_0}}\right) e^{-\frac{(d-d_{free})RE_b}{N_0}} \\ &= Q\left(\sqrt{\frac{2d_{free}RE_b}{N_0}}\right) e^{\frac{d_{free}RE_b}{N_0}} e^{-\frac{dRE_b}{N_0}}. \end{aligned} \tag{12.49}$$

Now, defining the function

$$f(x) = Q(\sqrt{2x})e^{x}, \tag{12.50}$$

we can write

$$P_d \le f\left(\frac{d_{free}RE_b}{N_0}\right) e^{-\frac{dRE_b}{N_0}}. \tag{12.51}$$

Finally, substituting (12.51) into the bounds of (12.25) and (12.29), we obtain the following expressions for the event- and bit-error probabilities of a binary-input, continuous-output AWGN channel:

$$\begin{aligned} P(E) &< \sum_{d=d_{free}}^{\infty} A_d f\left(\frac{d_{free}RE_b}{N_0}\right) e^{-\frac{dRE_b}{N_0}} \\ &= f\left(\frac{d_{free}RE_b}{N_0}\right) A(X)|_{X=\exp(-RE_b/N_0)} \end{aligned} \tag{12.52a}$$

$$\begin{aligned} P_b(E) &< \sum_{d=d_{free}}^{\infty} B_d f\left(\frac{d_{free}RE_b}{N_0}\right) e^{-\frac{dRE_b}{N_0}} \\ &= f\left(\frac{d_{free}RE_b}{N_0}\right) B(X)|_{X=\exp(-RE_b/N_0)}. \end{aligned} \tag{12.52b}$$

We note that the tightened bounds of (12.52) differ from the bounds of (12.45) by the scaling factor $f(d_{free}RE_b/N_0)$, which depends only on the free distance of the code. We now give an example illustrating the application of the bit-error probability bounds in (12.45b) and (12.52b).

EXAMPLE 12.5 Bit-Error Probability Bounds for an AWGN Channel

Consider the (3, 1, 2) encoder of (12.1) whose IOWEF $A(W, X, L)$ is given in (12.15). The bit WEF of this encoder is given by

$$\begin{aligned} B(X) &= (1/k) \left.\frac{\partial A(W, X)}{\partial W}\right|_{W=1} \\ &= \left.\frac{\partial[X^7W/(1 - XW - X^3W)]}{\partial W}\right|_{W=1} \\ &= \frac{X^7}{(1 - 2X + X^2 - 2X^3 + 2X^4 + X^6)} \\ &= X^7 + 2X^8 + 3X^9 + 6X^{10} + \cdots . \end{aligned} \tag{12.53}$$

The free distance of this code is $d_{free} = 7$, and we can use (12.53) directly to evaluate the bounds of (12.45b) and (12.52b). In Figure 12.11 we plot these two bounds as functions of the bit SNR E_b/N_0. Also plotted for comparison is the performance of uncoded BPSK and the result of a computer simulation showing the Viterbi decoding performance of this code on a continuous-output AWGN channel. Note that the tightened bound of (12.52b) agrees almost exactly with the simulation for SNRs higher than about 4 dB, whereas (12.45b) tracks the simulation closely but is not as tight. For lower SNRs, however, both bounds diverge from the simulation result. This is typical behavior for union bounds; namely, at SNRs near capacity (about −0.5 dB for rate $R = 1/3$ and BPSK modulation), they do not give good estimates of performance.

The (soft-decision) asymptotic coding gain of this code is given by

$$\gamma = 10\log_{10}(Rd_{free}) = 10\log_{10}(7/3) = 3.68 \text{ dB}. \tag{12.54}$$

This is the coding gain, compared with uncoded BPSK, that is achieved in the limit of high SNR. We see from Figure 12.11, however, that the *real coding gain* at a bit-error probability of $P_b(E) = 10^{-4}$ is only about 3.3 dB, illustrating that real coding gains are always somewhat less than asymptotic coding gains. In general, larger real coding gains are achieved by codes with fewer nearest-neighbor codewords, that is, smaller values of $A_{d_{free}}$ and $B_{d_{free}}$.

The performance bounds derived in this section are valid for *unterminated* convolutional encoders; that is, they are independent of the length N of the codeword. The event-error probability $P(E)$ is the probability that at any given time unit the Viterbi algorithm will select a path that introduces errors into the

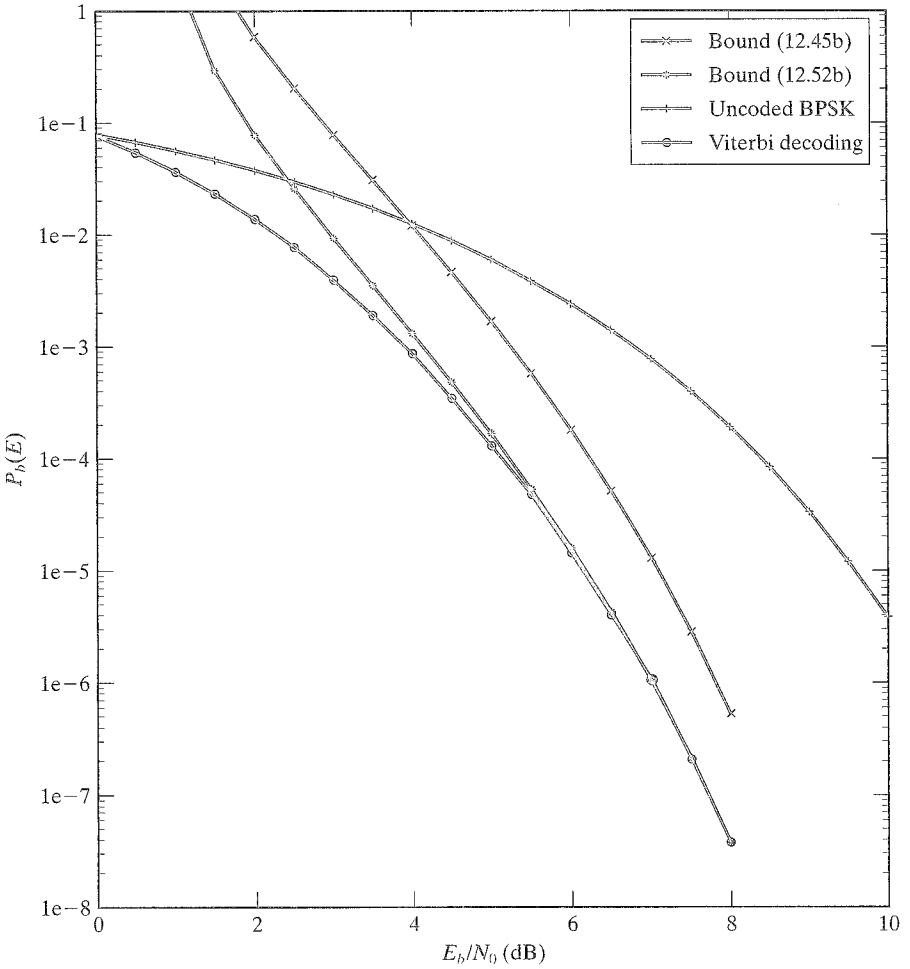


FIGURE 12.11: Performance bounds for convolutional codes.

decoded sequence, whereas the bit-error probability $P_b(E)$ is the average number of bit errors per unit time. Thus the bit-error probability $P_b(E)$ represents the bit-error rate (BER) of an unterminated encoder. The word-error probability, or *word-error rate* (WER), of an unterminated encoder, on the other hand, is essentially unity, since if a long enough sequence is encoded, at least one error event must occur.

To determine bounds on the WER and BER of *terminated* encoders, we must modify the codeword and bit WEFs to include delayed versions of codewords and codewords that diverge and remerge with the all-zero state more than once, as noted previously in Section 11.2. In other words, terminated convolutional codes are block codes, and the WEFs must account for all possible codewords. Techniques for finding these modified WEFs and evaluating the performance of terminated convolutional codes are presented in Chapter 16 on turbo coding.

12.3 CONSTRUCTION OF GOOD CONVOLUTIONAL CODES

We can now address the problem of constructing good codes for use with maximum likelihood (Viterbi) decoding. Once a desired code rate has been selected, the performance bounds presented in the previous section can be used as guidance in the construction of good codes. For example, the bounds of (12.25), (12.29), (12.39), and (12.45) all indicate that the most significant term for both $P(E)$ and $P_b(E)$, that is, the free distance term, decays exponentially with increasing d_{free} and increases linearly with increasing $A_{d_{free}}$ and $B_{d_{free}}$. This relationship suggests that the most important criterion should be maximizing the free distance d_{free}. Then, as secondary criteria, $A_{d_{free}}$, the number of (nearest-neighbor) codewords with weight d_{free}, and $B_{d_{free}}$, the total information sequence weight of all weight-d_{free} codewords, divided by k, should be minimized. (Because of the close relationship between $A_{d_{free}}$ and $B_{d_{free}}$ (see (11.129)), it is sufficient to simply minimize $A_{d_{free}}$.) Generally speaking, d_{free} is of primary importance in determining performance at high SNRs, but as the SNR decreases, the influence of the number of nearest-neighbors $A_{d_{free}}$ increases, and for very low SNRs the entire weight spectrum plays a role. Finally, the use of catastrophic encoders should be avoided under all circumstances.

Most code constructions for convolutional codes have been done by computer search. Algebraic structures that guarantee good distance properties, similar to the BCH construction for block codes, have proved difficult to find for convolutional codes. This has prevented the construction of good long codes, since most computer search techniques are time-consuming and limited to relatively short constraint lengths. An efficient search procedure for finding d_{free} and $A_{d_{free}}$ based on the Viterbi algorithm has been developed by Bahl, Cullum, Frazer, and Jelinek [15] and modified by Larsen [16]. The algorithm assumes the received sequence is all zeros, confines the search through the trellis to only those paths starting with a nonzero information block, uses a metric of 0 for an agreement and +1 for a disagreement, and searches for the path with the minimum metric. As soon as the metric of the survivor at the all-zero state is less than or equal to the metric of the other survivors, the algorithm can be terminated. The metric at the all-zero state then equals d_{free}, since none of the other survivors can ever remerge to the all-zero state with a smaller metric. (A straightforward modification of this procedure can also be used to compute $A_{d_{free}}$.) A trellis depth of several constraint lengths is typically required to find d_{free} for a noncatastrophic encoder. For a catastrophic encoder, the survivor at the all-zero state may never achieve the minimum metric, owing to the zero loop in the state diagram. (This can be used as an indicator of a catastrophic encoder.) This algorithm is capable of computing d_{free} and $A_{d_{free}}$ for values of ν up to about 20. For larger values of ν, the number of storage locations required by the algorithm, 2^{ν}, becomes unacceptably large and other means of finding d_{free} must be tried. No general solution to the problem of finding d_{free} for large values of ν has yet been discovered. We shall see in Section 13.4, however, that some sequential decoding algorithms for convolutional codes can, with proper modification, be used to compute the free distance of codes for values of ν up to about 30.

Lists of optimum codes for rates $R = 1/4, 1/3, 1/2, 2/3$, and $3/4$ are given in Table 12.1, where the optimality criterion first maximizes d_{free} and then minimizes $A_{d_{free}}$. In the table, we list the overall constraint length ν, the free distance d_{free}, the number of nearest-neighbor codewords $A_{d_{free}}$, and the soft-decision asymptotic

coding gain γ, given by (12.47), of each code. In Tables 12.1(a), 12.1(b), and 12.1(c), for the (low-rate) codes with $k = 1$, we list the coefficients of the polynomials in the generator matrix $\mathbb{G}(D) = [\mathbb{g}^{(0)}(D)\ \mathbb{g}^{(1)}(D) \cdots \mathbb{g}^{(n-1)}(D)]$ (see (11.80a)), that is, the generator sequences, in octal form, for the minimal controller canonical form encoder realization. In Tables 12.1(d) and 12.1(e), for the (high-rate) codes with $k > 1$, we list the coefficients of the polynomials in the parity-check matrix $\mathbb{H}(D) = [\mathbb{h}^{(n-1)}(D) \cdots \mathbb{h}^{(1)}(D)\ \mathbb{h}^{(0)}(D)]$ (see (11.82b)), that is, the parity-check sequences, in octal form, for the minimal observer canonical form encoder realization. To be consistent in the octal representations of the generator and parity-check sequences for convolutional encoders listed throughout this text, we adopt the following convention. We first write a binary polynomial $\mathbb{f}(D)$ of degree ν from highest order to lowest order as follows:

$$\mathbb{f}(D) = f_\nu D^\nu + f_{\nu-1} D^{\nu-1} + \ldots + f_1 D + f_0. \tag{12.55}$$

TABLE 12.1(a)[3]: Optimum rate $R = 1/4$ convolutional codes.

ν	$\mathbb{g}^{(0)}$	$\mathbb{g}^{(1)}$	$\mathbb{g}^{(2)}$	$\mathbb{g}^{(3)}$	d_{free}	$A_{d_{free}}$	γ (dB)
1	1	1	3	3	6	1	1.76
2	5	5	7	7	10	1	3.98
3	13	13	15	17	13	2	5.12
4	25	27	33	37	16	4	6.02
5	45	53	67	77	18	3	6.53
6	117	127	155	171	20	2	6.99
7	257	311	337	355	22	1	7.40
8	533	575	647	711	24	1	7.78
9	1173	1325	1467	1751	27	3	8.29

TABLE 12.1(b): Optimum rate $R = 1/3$ convolutional codes.

ν	$\mathbb{g}^{(0)}$	$\mathbb{g}^{(1)}$	$\mathbb{g}^{(2)}$	d_{free}	$A_{d_{free}}$	γ (dB)
1	1	3	3	5	1	2.22
2	5	7	7	8	2	4.26
3	13	15	17	10	3	5.22
4	25	33	37	12	5	6.02
5	47	53	75	13	1	6.36
6	117	127	155	15	3	6.99
7	225	331	367	16	1	7.27
8	575	623	727	18	1	7.78
9	1167	1375	1545	20	3	8.23
10	2325	2731	3747	22	7	8.65
11	5745	6471	7553	24	13	9.03
12	2371	13725	14733	24	5	9.03

[3]Tables 12.1a–e adapted from [17].

TABLE 12.1(c): Optimum rate $R = 1/2$ convolutional codes.

ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{free}	$A_{d_{free}}$	γ (dB)
1	3	1	3	1	1.76
2	5	7	5	1	3.98
3	13	17	6	1	4.77
4	27	31	7	2	5.44
5	53	75	8	1	6.02
6	117	155	10	11	6.99
7	247	371	10	1	6.99
8	561	753	12	11	7.78
9	1131	1537	12	1	7.78
10	2473	3217	14	14	8.45
11	4325	6747	15	14	8.75
12	10627	16765	16	14	9.03
13	27251	37363	16	1	9.03

TABLE 12.1(d): Optimum rate $R = 2/3$ convolutional codes.

ν	$\mathbf{h}^{(2)}$	$\mathbf{h}^{(1)}$	$\mathbf{h}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (dB)
2	3	5	7	3	1	3.01
3	17	15	13	4	1	4.26
4	23	31	27	5	3	5.23
5	71	57	73	6	7	6.02
6	123	147	121	7	17	6.69
7	313	227	241	8	43	7.27
8	555	631	477	8	6	7.27
9	1051	1423	1327	9	17	7.78
10	2621	2137	3013	10	69	8.24

TABLE 12.1(e): Optimum rate $R = 3/4$ convolutional codes.

ν	$\mathbf{h}^{(3)}$	$\mathbf{h}^{(2)}$	$\mathbf{h}^{(1)}$	$\mathbf{h}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (dB)
2	2	5	7	6	3	6	3.52
3	11	13	15	12	4	10	4.77
4	33	25	37	31	4	2	4.77
5	47	73	57	75	5	7	5.74
6	107	135	133	141	6	27	6.53
7	211	341	315	267	6	5	6.53
8	535	757	733	661	7	27	7.20
9	1475	1723	1157	1371	8	136	7.78

Then, starting with the rightmost bit f_0, we group the coefficients in threes and represent them as octal digits, with 0's appended on the left to make the total number of bits a multiple of 3. We then list the octal digits from right (lowest-order terms) to left (highest-order terms) in the tables. For example, the best (3, 1, 4) encoder in Table 12.1(b) has

$$\begin{aligned}\mathbb{G}(D) &= [\mathbb{g}^{(0)}(D)\ \ \mathbb{g}^{(1)}(D)\ \ \mathbb{g}^{(2)}(D)] \\ &= [1 + D^2 + D^4 \quad 1 + D + D^3 + D^4 \quad 1 + D + D^2 + D^3 + D^4],\end{aligned} \tag{12.56}$$

and its generator sequences are listed as $\mathbb{g}^{(0)} = (25)$, $\mathbb{g}^{(1)} = (33)$, and $\mathbb{g}^{(2)} = (37)$. Similarly, the best (3, 2, 3) encoder in Table 12.1(d) has

$$\begin{aligned}\mathbb{H}(D) &= [\mathbb{h}^{(2)}(D)\ \ \mathbb{h}^{(1)}(D)\ \ \mathbb{h}^{(0)}(D)] \\ &= [D^3 + D^2 + D + 1 \quad D^3 + D^2 + 1 \quad D^3 + D + 1],\end{aligned} \tag{12.57}$$

and its parity-check sequences are listed as $\mathbb{h}^{(2)} = (17)$, $\mathbb{h}^{(1)} = (15)$, and $\mathbb{h}^{(0)} = (13)$. (We note that the controller canonical form polynomials in (12.56) must be written in reverse order prior to forming the octal representation, whereas the observer canonical form polynomials in (12.57) are already written in reverse order.) Because the search procedures employed are essentially exhaustive, optimum codes have been obtained only for relatively short constraint lengths. Nevertheless, the soft-decision asymptotic coding gains of some of the codes are quite large. For example, the (2, 1, 18) code with $d_{free} = 22$ achieves an asymptotic coding gain of 10.41 dB with soft-decision decoding; however, since the number of nearest neighbors for this code is a rather large $A_{22} = 65$, the real coding gain will exceed 10 dB only at very low BERs (very high SNRs).

The optimum codes listed in Table 12.1 are generated by either a nonsystematic feedforward encoder or an equivalent systematic feedback encoder. This is because for a given rate and constraint length, more free distance is available with nonsystematic feedforward encoders than with systematic feedforward encoders, as was pointed out in Chapter 11; however, systematic encoders have the advantage that no inverting circuit is needed to recover a noisy version of the information sequence from the codeword without decoding. This property allows the user to take a "quick look" at the received information sequence without having to invert a "nonsystematic codeword." This feature can be important in systems where decoding is done off-line, or where the decoder is subject to temporary failures, or where the channel is known to be "noiseless" during certain time intervals and decoding becomes unnecessary. In these cases, the systematic feedback form of encoder realization is preferred.

In some applications a feedforward encoder realization may be preferred. For example, for feedforward encoders, m blocks of all-zero input bits can always be used to terminate the encoder rather than having the termination bits depend on the information sequence. In this case it is desirable to try to combine the quick-look property of systematic feedforward encoders with the superior free distance available with nonsystematic feedforward encoders. To this end, Massey and Costello [18] developed a class of rate $R = 1/2$ nonsystematic feedforward encoders called

quick-look-in encoders that have a property similar to the quick-look capability of systematic encoders. They are defined by

$$\mathbb{g}^{(0)}(D) + \mathbb{g}^{(1)}(D) = D \tag{12.58}$$

and $g_0^{(0)} = g_0^{(1)} = g_m^{(0)} = g_m^{(1)} = 1$; that is, the two generator sequences differ in only a single position. These encoders are always noncatastrophic (see Problem 12.15), and their feedforward inverse has the trivial transfer function matrix

$$\mathbb{G}^{-1}(D) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{12.59}$$

Because

$$\mathbb{G}(D)\mathbb{G}^{-1}(D) = [\mathbb{g}^{(0)}(D) \quad D + \mathbb{g}^{(0)}(D)]\begin{bmatrix} 1 \\ 1 \end{bmatrix} = D, \tag{12.60}$$

the information sequence $\mathbb{u}(D)$ can be recovered from the codeword $\mathbb{V}(D)$ with a one time unit delay. The recovery equation is

$$\mathbb{V}(D)\mathbb{G}^{-1}(D) = \mathbb{v}^{(0)}(D) + \mathbb{v}^{(1)}(D) = D\mathbb{u}(D), \tag{12.61}$$

and we see that if p is the probability that a bit in the codeword $\mathbb{V}(D)$ is in error, then the probability of a bit error in recovering $\mathbb{u}(D)$ is roughly $2p$,[4] because an error in recovering u_l can be caused by an error in either $v_{i+1}^{(0)}$ or $v_{i+1}^{(1)}$.

For any noncatastrophic rate $R = 1/2$ encoder with feedforward inverse

$$\mathbb{G}^{-1}(D) = \begin{bmatrix} \mathbb{g}_0^{-1}(D) \\ \mathbb{g}_1^{-1}(D) \end{bmatrix}, \tag{12.62}$$

the recovery equation is

$$\mathbb{v}^{(0)}(D)\mathbb{g}_0^{-1}(D) + \mathbb{v}^{(1)}(D)\mathbb{g}_1^{-1}(D) = D^l\mathbb{u}(D), \tag{12.63}$$

for some l, and an error in recovering u_l can be caused by an error in any of $w[\mathbb{g}_0^{-1}(D)]$ positions in $\mathbb{v}^{(0)}(D)$ or any of $w[\mathbb{g}_1^{-1}(D)]$ positions in $\mathbb{v}^{(1)}(D)$. Hence, the probability of a bit error in recovering $\mathbb{u}(D)$ is $A \triangleq w[\mathbb{g}_0^{-1}(D)] + w[\mathbb{g}_1^{-1}(D)]$ times the probability of a bit error in the codeword. A is called the *error probability amplification factor* of the encoder. $A = 2$ for quick-look-in encoders, and this is the minimum value of A for any $R = 1/2$ nonsystematic encoder. For $R = 1/2$ systematic (feedforward or feedback) encoders,

$$\mathbb{G}^{-1}(D) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{12.64}$$

and $A = 1$ for systematic encoders. Hence, quick-look-in encoders are "almost systematic," in the sense that they have the minimum value of A for any nonsystematic encoder. Catastrophic encoders, at the other extreme, have no feedforward inverse, and their error probability amplification factor is infinite.

[4] We are ignoring here the unlikely event that two errors in $\mathbb{V}(D)$ will cancel, causing no error in recovering $\mathbb{u}(D)$.

TABLE 12.2: Optimum rate $R = 1/2$ quick-look-in convolutional codes.

ν	$\mathbf{g}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (dB)
2	5	5	1	3.98
3	15	6	1	4.77
4	31	7	2	5.44
5	55	8	2	6.02
6	151	9	4	6.53
7	215	9	1	6.53
8	455	10	1	6.99
9	1335	11	3	7.40
10	3055	12	3	7.78
11	6055	13	8	8.13
12	14135	14	10	8.45
13	34731	14	3	8.45
14	60545	15	6	8.75
15	171045	16	11	9.03
16	341225	16	2	9.03
17	613151	17	5	9.29
18	1422255	18	6	9.54
19	3007451	18	2	9.54
20	6153605	19	4	9.78
21	14565371	20	7	10.00
22	32720445	20	1	10.00
23	63347465	21	3	10.21
24	147373045	22	7	10.41

The capability of quick-look-in encoders to provide an immediate estimate (prior to decoding) of the information sequence from a noisy version of the codeword with an error probability amplification factor of only 2 makes them desirable in some applications. The free distances and number of nearest neighbors for the optimum rate $R = 1/2$ codes generated by quick-look-in encoders are listed in Table 12.2. Note that d_{free} for the best quick-look-in $R = 1/2$ codes is a little less than d_{free} for the best overall $R = 1/2$ codes listed in Table 12.1(c). Thus, the "almost systematic" property of quick-look-in codes results in a small performance penalty compared with the best codes (see Problem 12.16). Their free distances are superior, however, to what can be achieved with the codes generated by systematic feedforward encoders (see Problem 12.17).

As noted previously, there are few algebraic constructions available for convolutional codes. One exception is the construction of orthogonal codes for use with majority-logic decoding. These constructions are covered in Chapter 13. Another approach, initiated by Massey, Costello, and Justesen [19], uses the minimum distance properties of a cyclic block code to provide a lower bound on the free distance of an associated convolutional code. If $\mathbb{g}(X)$ is the generator polynomial of any (n, k) cyclic code of odd length n with minimum distance d_g, and $\mathbb{h}(X) = (X^n - 1)/\mathbb{g}(X)$ is

the generator polynomial of the $(n, n-k)$ dual code with minimum distance d_h, the following construction for rate $R = 1/2$ codes results.

Construction 12.1 The rate $R = 1/2$ convolutional encoder with composite generator polynomial $\mathbf{g}(D)$ is noncatastrophic and has $d_{free} \geq \min(d_g, 2d_h)$.

The cyclic codes should be selected so that $d_g \approx 2d_h$. This suggests trying cyclic codes with rates in the range $1/3 \leq R \leq 1/2$ (see Problem 12.18).

EXAMPLE 12.6 Constructing Convolutional Codes from Block Codes

Consider the (15, 5) BCH code with generator polynomial $\mathbf{g}(X) = 1+X+X^2+X^4+X^5+X^8+X^{10}$. This code has minimum distance $d_g = 7$. The generator polynomial of the dual code is $\mathbf{h}(X) = (X^{15}-1)/\mathbf{g}(X) = X^5 + X^3 + X + 1$, and $d_h = 4$. The rate $R = 1/2$ convolutional encoder with composite generator polynomial $\mathbf{g}(D) = 1 + D + D^2 + D^4 + D^5 + D^8 + D^{10}$ then has $d_{free} \geq \min(7, 8) = 7$. The polynomial generator matrix is given by

$$\mathbf{G}(D) = [1 + D + D^2 + D^4 + D^5 \quad 1 + D^2], \tag{12.65}$$

and we see that the encoder is noncatastrophic and has constraint length $\nu = 5$. If $\mathbf{u}(D) = 1$, the codeword

$$\mathbf{v}(D) = \mathbf{u}(D^2)\mathbf{g}(D) = 1 + D + D^2 + D^4 + D^5 + D^8 + D^{10} \tag{12.66}$$

has weight 7, and hence $d_{free} = 7$ for this code.

A similar construction can be used to produce codes with rate $R = 1/4$.

Construction 12.2 The rate $R = 1/4$ convolutional code with composite generator polynomial $\mathbf{g}(D^2) + D\mathbf{h}(D^2)$ is noncatastrophic and has $d_{free} \geq \min(d_g + d_h, 3d_g, 3d_h)$.

The cyclic codes should be selected so that $d_g \approx d_h$. This suggests trying cyclic codes with rates near $R = 1/2$ (see Problem 12.18).

Two difficulties prevent these constructions from yielding good long convolutional codes. The first is the problem of finding long cyclic codes with large minimum distances. The second is the dependence of the bound on the minimum distance of the duals of cyclic codes. The second difficulty was circumvented in a subsequent paper by Justesen [20]. Justesen's construction yields the bound $d_{free} \geq d_g$, but it involves a rather complicated condition on the roots of $\mathbf{g}(X)$ and, in the binary case, can be used only to construct convolutional codes with odd values of n. Another paper by Tanner [21] broadens the class of block codes to include quasi-cyclic codes. Tanner's construction yields the bound $d_{free} \geq d_{min}$, where d_{min} is the minimum distance of an associated quasi-cyclic code, and it provides an interesting link between the theories of block and convolutional codes.

12.4 IMPLEMENTATION AND PERFORMANCE OF THE VITERBI ALGORITHM

The basic operation of the Viterbi algorithm was presented in Section 12.1. In a practical implementation of the algorithm, several additional factors must be

considered. In this section we discuss some of these factors and how they affect decoder performance.

Decoder Memory. Because there are 2^ν states in the state diagram of the encoder, the decoder must reserve 2^ν words of storage for the survivors. Each word must be capable of storing the surviving path along with its metric. Since the storage requirements increase exponentially with the constraint length ν, in practice it is not feasible to use codes with large ν (although special-purpose Viterbi decoders with constraint length as high as $\nu = 14$ have been implemented [22]). This limits the available free distance, and soft-decision coding gains of around 7 dB are the practical limit of the Viterbi algorithm in most cases. The exact error probabilities achieved depend on the code, its rate, its free distance, the available channel SNR, and the demodulator output quantization, as well as other factors.

Path Memory. We noted in Section 11.1 that convolutional codes are most efficient when the length of the information sequence is large. The difficulty this causes is that each of the 2^ν words of storage must be capable of storing a $K^* = kh$-bit path plus its metric. For very large h, this is clearly impossible, and some compromises must be made. The approach that is usually taken is to truncate the path memory of the decoder by storing only the most recent τ blocks of information bits for each survivor, where $\tau << h$. Hence, after the first τ blocks of the received sequence have been processed by the decoder, the decoder memory is full. After the next block is processed, a decoding decision must be made on the first block of k information bits, since it can no longer be stored in the decoder's memory.

There are several possible strategies for making this decision. Among these are the following:

1. Choose an arbitrary survivor, and select the first information block on this path.
2. Select from among the 2^k possible first information blocks the one that appears most often in the 2^ν survivors.
3. Choose the survivor with the best metric, and select the first information block on this path.

After the first decoding decision is made, additional decoding decisions are made in the same way for each new received block processed. Hence, the decoding decisions always lag the progress of the decoder by an amount equal to the path memory, that is, τ blocks. At the end of a terminated trellis, there remain $\tau - m$ information blocks to decode. These are simply selected as the last $\tau - m$ information blocks on the final surviving path.

The decoding decisions made in this way are no longer maximum likelihood, but can be almost as good as maximum likelihood if τ is not too small. Experience and analysis have shown that if τ is on the order of 5 times the encoder memory order or more, with probability approaching 1 all 2^ν survivors stem from the same information block τ time units back; thus, there is no ambiguity in making the decoding decision. This situation is illustrated in Figure 12.12. In addition, this must be the maximum likelihood decision, since no matter which survivor eventually becomes the maximum likelihood path, it must contain this decoded information block. Hence, if τ is large enough, almost all decoding decisions will be maximum

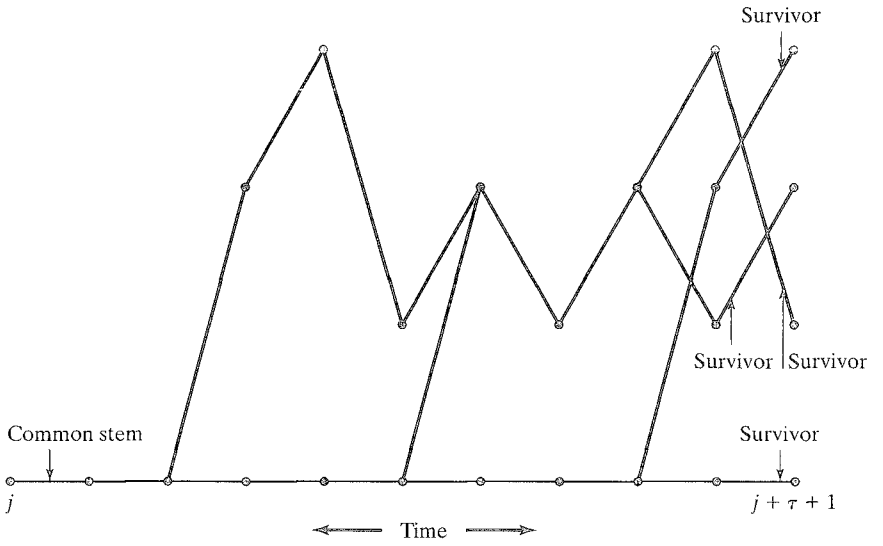


FIGURE 12.12: Decoding decisions with a finite path memory.

likelihood, and the final decoded path will be close to the maximum likelihood path. This point is illustrated in Problem 12.19.

There are two ways in which errors can occur in a truncated decoder. Assume that a branch decision at time unit t is made by selecting the survivor at time unit $t + \tau + 1$ with the best metric and then decoding the information bits on that path at time unit t. If a decoder with unlimited path memory contains a decoding error (i.e., the maximum likelihood path diverges from the correct path) at time unit t, it is reasonable to assume that the maximum likelihood path is the best survivor at time unit $t + \tau + 1$, and hence a decoder with finite path memory will make the same error. An additional source of error with a truncated decoder occurs when some incorrect path that is unmerged with the correct path from time unit t through time unit $t + \tau + 1$ is the best survivor at time unit $t + \tau + 1$. In this case a decoding error may be made at time unit t, even though this incorrect path may be eliminated when it later remerges with the correct path and thus will not cause an error in a decoder with unlimited path memory. Decoding errors of this type are called *decoding errors due to truncation*. The subset of incorrect paths that can cause a decoding error due to truncation is shown in Figure 12.13. Note that it includes all unmerged paths of length greater than τ that diverge from the correct path at time unit j or earlier.

For a convolutional encoder with codeword WEF $A(W, X, L)$, the event-error probability on a BSC of a truncated decoder is bounded by

$$P(E) < \left[A(W, X, L) + \sum_{i=1}^{2^{\nu}-1} A_i^{\tau}(W, X, L) \right]_{X=2\sqrt{p(1-p)}, W=1, L=1}, \qquad (12.67)$$

where $\sum_{i=1}^{2^{\nu}-1} A_i^{\tau}(W, X, L)$ is the codeword WEF for the subset of incorrect paths that can cause decoding errors due to truncation [23]. In other words, $A_i^{\tau}(W, X, L)$ is

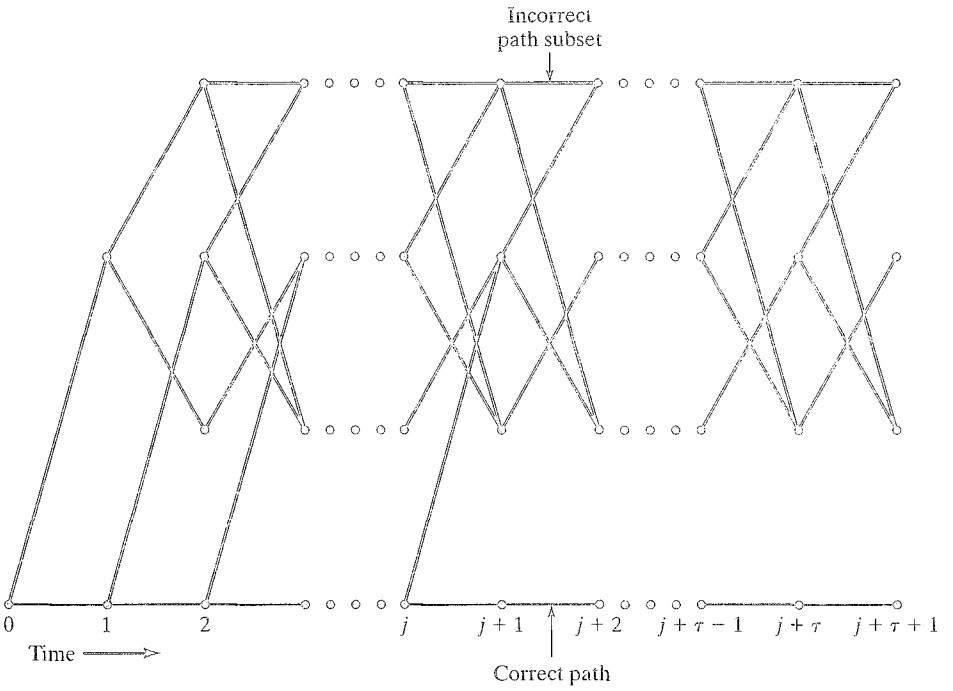


FIGURE 12.13: Incorrect path subset for a truncated decoder.

the codeword WEF for the set of all unmerged paths of length more than τ branches in the augmented modified encoder state diagram that connect the all-zero state with the ith state. The first term in (12.67) represents the decoding errors made by a decoder with unlimited path memory, whereas the second term represents the decoding errors due to truncation. Equation (12.67) can be generalized to other DMCs and the unquantized AWGN channel by letting $X = D_o$ and $X = e^{-RE_b/N_0}$, respectively. Also, expressions similar to (12.67) can be obtained for the bit-error probability $P_b(E)$ by starting with the bit WEF $B(W, X, L)$ of the encoder.

When p is small (if the BSC is derived from a hard-quantized AWGN channel, this means large E_b/N_0), (12.67) can be approximated as

$$P(E) \approx A_{d_{free}} \left(2\sqrt{p(1-p)}\right)^{d_{free}} + A_{d(\tau)} \left(2\sqrt{p(1-p)}\right)^{d(\tau)}, \tag{12.68}$$

where $d(\tau)$ is the smallest power of D, and $A_{d(\tau)}$ is the number of terms with weight $d(\tau)$ in the unmerged codeword WEF $\sum_{i=1}^{2^{\nu}-1} A_i^{\tau}(W, X, L)$. Further simplification of (12.68) yields

$$P(E) \approx A_{d_{free}} 2^{d_{free}} p^{d_{free}/2} + A_{d(\tau)} 2^{d(\tau)} p^{d(\tau)/2}. \tag{12.69}$$

From (12.69) it is clear that for small values of p, if $d(\tau) > d_{free}$, the second term is negligible compared with the first term, and the additional error probability due to truncation can be ignored. Hence, the path memory τ should be chosen large enough so that $d(\tau) > d_{free}$ in a truncated decoder. The minimum value of τ for

TABLE 12.3: Minimum truncation lengths for rate $R = 1/2$ optimum free distance codes.

m	d_{free}	τ_{min}	$d(\tau_{min})$
2	5	7	6
3	6	9	7
4	7	14	8
5	8	18	9
6	10	26	11
7	10	27	11

which $d(\tau) > d_{free}$ is called the *minimum truncation length* τ_{min} of the encoder. The minimum truncation lengths for some of the rate $R = 1/2$ optimum free distance codes listed in Table 12.1(c) are given in Table 12.3. Note that $\tau_{min} \approx 4m$ in most cases. A random coding analysis by Forney [4] shows that in general τ_{min} depends on R, but that typically a small multiple of m is sufficient to ensure that the additional error probability due to truncation is negligible. Extensive simulations and actual experience have verified that 4 to 5 times the memory order of the encoder is usually an acceptable truncation length to use in practice.

EXAMPLE 12.7 Minimum Truncation Length of a (2, 1, 2) Encoder

Consider the (2, 1, 2) nonsystematic feedforward encoder with $\mathbb{G}(D) = [1 + D + D^2 \quad 1 + D^2]$. The augmented modified state diagram for this encoder is shown in Figure 12.14, and the codeword WEF is given by

$$A(W, X, L) = \frac{X^5 W L^3}{1 - XWL(1 + L)}. \tag{12.70}$$

Letting $A_i(W, X, L)$ be the WEF for all paths connecting the all-zero state (S_0) with the ith state (S_i) we find that

$$A_1(W, X, L) = \frac{X^2 WL(1 - XWL)}{1 - XWL(1 + L)}, \tag{12.71a}$$

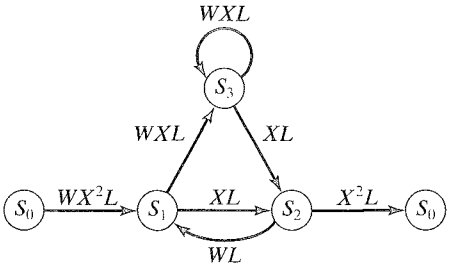


FIGURE 12.14: Augmented modified state diagram for a (2, 1, 2) encoder.

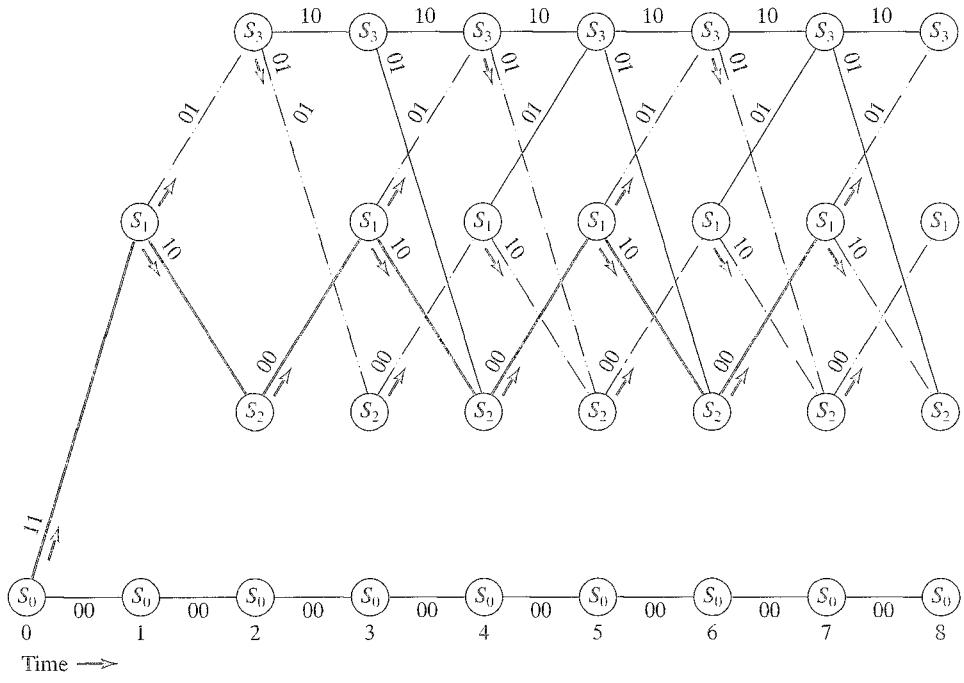


FIGURE 12.15: Determining the truncation distance of a (2, 1, 2) encoder.

$$A_2(W, X, L) = \frac{X^3 W L^2}{1 - XWL(1 + L)}, \tag{12.71b}$$

$$A_3(W, X, L) = \frac{X^3 W^2 L^2}{1 - XWL(1 + L)}. \tag{12.71c}$$

If we now expurgate each of these generating functions to include only paths of length more than τ branches, $d(\tau)$ is the smallest power of X in any of the three expurgated functions. For example, $d(0) = 2$, $d(1) = 3$, $d(2) = 3$, and so on. Because $d_{free} = 5$ for this code, τ_{min} is the minimum value of τ for which $d(\tau) = d_{free} + 1 = 6$. $A_1(W, X, L)$ contains a term $X^5 W^4 L^7$, and hence $d(6) \leq 5$ and τ_{min} must be at least 7. The particular path yielding this term is shown darkened on the trellis diagram of Figure 12.15. A careful inspection of the trellis diagram shows that there is no path of length 8 branches that terminates on S_1, S_2, or S_3 and has weight less than 6. There are five such paths, however, that have weight 6, and these are shown dotted in Figure 12.15. Hence, $d(7) = 6$, $A_{d(7)} = 5$, and $\tau_{min} = 7$, which is 3.5 times the memory order of the encoder in this case. Hence, a decoder with a path memory of 7 should be sufficient to ensure a negligible increase in event-error probability over a decoder with unlimited path memory for this code.

Finally, it is important to point out the distinction between the column distance d_l defined in Section 11.3 and the *truncation distance* $d(\tau)$ defined here. d_l is the

minimum weight of any codeword of length $l + 1$ branches. Because this includes codewords that have remerged with the all-zero state and hence whose weight has stopped increasing beyond a certain point, d_l reaches a maximum value of d_{free} as l increases. On the other hand, $d(\tau)$ is the minimum weight of any codeword of length more than τ branches that has not yet remerged with the all-zero state. Because remergers are not allowed, $d(\tau)$ will continue to increase without bound as τ increases. For example, for the encoder state diagram of Figure 12.14, $d(9) = 7, d(19) = 12, d(29) = 17, \cdots$, and, in general, $d(\tau) = \frac{\tau+1}{2} + 2$ for odd τ.

Catastrophic encoders are the only exception to this rule. In a catastrophic encoder, the zero-weight loop in the state diagram prevents $d(\tau)$ from increasing as τ increases. Hence, catastrophic encoders contain very long codewords with low weight, which makes them susceptible to high error probabilities when used with Viterbi decoding, whether truncated or not. For example, the catastrophic encoder of Figure 11.14 has $d_{free} = 4$ but contains an unmerged codeword of infinite length with weight 3, and $d(\tau) = 3$ for all $\tau \geq 1$. Hence, the additional error probability due to truncation will dominate the error probability expression of (12.69), no matter what truncation length is chosen, and the code will not perform as well as a code generated by a noncatastrophic encoder with $d_{free} = 4$. This performance difference between catastrophic and noncatastrophic encoders is discussed further in [24].

Decoder Synchronization. In practice, decoding does not always commence with the first branch transmitted after the encoder is set to the all-zero state but may begin with the encoder in an unknown state, in midstream, so to speak. In this case, all state metrics are initially set to zero, and decoding starts in the middle of the trellis. If path memory truncation is used, the initial decisions taken from the survivor with the best metric are unreliable, causing some decoding errors. But Forney [4] has shown, using random coding arguments, that after about $5m$ branches are decoded, the effect of the initial lack of branch synchronization becomes negligible. Hence, in practice the decoding decisions over the first $5m$ branches are usually discarded, and all later decisions are then treated as reliable.

Bit (or symbol) synchronization is also required by the decoder; that is, the decoder must know which of n consecutive received symbols is the first one on a branch. In attempting to synchronize symbols, the decoder makes an initial assumption. If this assumption is incorrect, the survivor metrics typically remain relatively closely bunched. This is usually indicative of a stretch of noisy received data, since if the received sequence is noise-free, the metric of the correct path typically dominates the metrics of the other survivors. This point is illustrated in Problem 12.21. If this condition persists over a long enough span, it is indicative of incorrect symbol synchronization, since long stretches of noise are very unlikely. In this case, the symbol synchronization assumption is changed until correct synchronization is achieved. Note that at most n attempts are needed to acquire correct symbol synchronization.

Receiver Quantization. For a binary-input AWGN channel with finite-output quantization, we gave performance bounds in (12.39) that depended on the Bhattacharyya parameter

$$D_0 = \sum_{j=0}^{Q-1} \sqrt{P(j|0)P(j|1)}, \tag{12.72}$$

where $P(j|0)$ and $P(j|1)$ are channel transition probabilities, and Q is the number of output quantization symbols. In particular, smaller values of D_0, which depends only on the binary-input, Q-ary output DMC model, result in lower error probabilities. Because we are free to design the quantizer, that is, to select the quantization thresholds, we should do this in such a way that D_0 is minimized. In general, for Q-ary output channels, there are $Q-1$ thresholds to select, and D_0 should be minimized with respect to these $Q-1$ parameters.

Because the transmitted symbols $\pm\sqrt{E_s}$ are symmetric with respect to zero, the (Gaussian) distributions $p(r|0)$ and $p(r|1)$ associated with an (unquantized) received value r are symmetric with respect to $\pm\sqrt{E_s}$, and since Q is normally a power of 2, that is, even, one threshold should be selected at $T_0 = 0$, and the other thresholds should be selected at values $\pm T_1, \pm T_2, \cdots, \pm T_{(Q/2)-1}$. Thus, in this case, the quantizer design problem reduces to minimizing D_0 with respect to the $(Q/2)-1$ parameters $T_1, T_2, \cdots, T_{(Q/2)-1}$. In Figure 12.16 we illustrate the calculation of the channel transition probabilities $P(\pm q_i|0)$ and $P(\pm q_i|1)$ based on a particular set of thresholds, where the notation $\pm q_i$ denotes the quantization symbol whose interval is bounded by $\pm T_i$ and $\pm T_{i-1}$, $i = 1, 2, \cdots, Q/2$, and $T_{Q/2} \equiv \infty$. Using this notation and the mapping $1 \rightarrow +\sqrt{E_s}$ and $0 \rightarrow -\sqrt{E_s}$, we can rewrite (12.72) as

$$D_0 = \sum_{i=1}^{Q/2} \left[\sqrt{P(q_i|0)P(q_i|1)} + \sqrt{P(-q_i|0)P(-q_i|1)} \right], \tag{12.73}$$

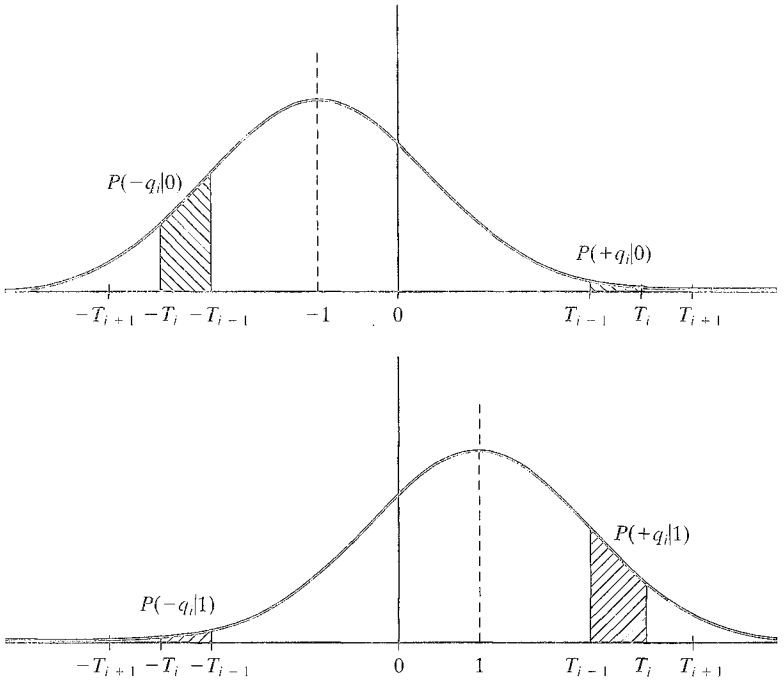


FIGURE 12.16: Calculating channel transition probabilities for a binary-input, Q-ary output DMC.

and we can calculate the transition probability $P(q_i|0)$ using (12.13) as follows:

$$P(q_i|0) = \int_{T_{i-1}}^{T_i} p(r|0)\,dr = \int_{T_{i-1}}^{T_i} \sqrt{\frac{E_s}{\pi N_0}} e^{-\frac{E_s}{N_0}(r+1)^2} dr. \tag{12.74}$$

We now minimize (12.72) with respect to T_i by solving the following equation for T_i:

$$\frac{\partial D_0}{\partial T_i} = \frac{\partial \left\{ \sum_{j=1}^{Q/2} \left[\sqrt{P(q_j|0)P(q_j|1)} + \sqrt{P(-q_j|0)P(-q_j|1)} \right] \right\}}{\partial T_i} = 0. \tag{12.75}$$

The solution of (12.75) involves terms of the form

$$\frac{\partial P(q_j|v)}{\partial T_i} = \frac{\partial \left[\int_{T_{i-1}}^{T_i} p(r|v)\,dr \right]}{\partial T_i}, \quad v = 0 \text{ or } 1. \tag{12.76}$$

Using the relations

$$\frac{\partial \left[\int_{T_{i-1}}^{T_i} p(r|v)\,dr \right]}{\partial T_i} = p(T_i|v) \tag{12.77a}$$

and

$$\frac{\partial \left[\int_{T_i}^{T_i+1} p(r|v)\,dr \right]}{\partial T_i} = -p(T_i|v), \tag{12.77b}$$

we can write

$$\frac{\partial P(q_j|v)}{\partial T_i} = \begin{cases} p(T_i|v), & j = i \\ -p(T_i|v), & j = i+1 \\ 0, & j \neq i,\ j \neq i+1. \end{cases} \tag{12.78}$$

(From Figure 12.16 we can see that only the transition probabilities $P(q_i|0)$, $P(q_i|1)$, $P(q_{i+1}|0)$, and $P(q_{i+1}|1)$ depend on the selection of the threshold T_i.) Now, using (12.78) in (12.75) we obtain

$$\begin{aligned} \frac{\partial D_0}{\partial T_i} &= \frac{1}{2}\sqrt{\frac{P(q_i|0)}{P(q_i|1)}}p(T_i|1) + \frac{1}{2}\sqrt{\frac{P(q_i|1)}{P(q_i|0)}}p(T_i|0) \\ &\quad - \frac{1}{2}\sqrt{\frac{P(q_{i+1}|0)}{P(q_{i+1}|1)}}p(T_i|1) - \frac{1}{2}\sqrt{\frac{P(q_{i+1}|1)}{P(q_{i+1}|0)}}p(T_i|0) \\ &= 0. \end{aligned} \tag{12.79}$$

We can also write (12.79) as

$$\sqrt{\frac{P(q_i|0)}{P(q_i|1)}}p(T_i|1) + \sqrt{\frac{P(q_i|1)}{P(q_i|0)}}p(T_i|0) = \sqrt{\frac{P(q_{i+1}|0)}{P(q_{i+1}|1)}}p(T_i|1) + \sqrt{\frac{P(q_{i+1}|1)}{P(q_{i+1}|0)}}p(T_i|0) \tag{12.80}$$

or

$$p(T_i|0)\left[\sqrt{\frac{P(q_i|1)}{P(q_i|0)}}-\sqrt{\frac{P(q_{i+1}|1)}{P(q_{i+1}|0)}}\right]=p(T_i|1)\left[\sqrt{\frac{P(q_{i+1}|0)}{P(q_{i+1}|1)}}-\sqrt{\frac{P(q_i|0)}{P(q_i|1)}}\right] \tag{12.81}$$

or

$$\begin{aligned}\frac{p(T_i|1)}{p(T_i|0)}&=\frac{\left[\sqrt{P(q_i|1)/P(q_i|0)}-\sqrt{P(q_{i+1}|1)/P(q_{i+1}|0)}\right]}{\left[\sqrt{P(q_{i+1}|0)/P(q_{i+1}|1)}-\sqrt{P(q_i|0)/P(q_i|1)}\right]}\\&=\frac{\sqrt{P(q_{i+1}|1)P(q_i|1)}}{\sqrt{P(q_i|0)P(q_{i+1}|0)}}\\&=\sqrt{\frac{P(q_i|1)}{P(q_i|0)}}\sqrt{\frac{P(q_{i+1}|1)}{P(q_{i+1}|0)}}.\end{aligned} \tag{12.82}$$

Now, we define the *likelihood ratio* of a received value r at the output of an unquantized binary-input channel as

$$\lambda(r)=\frac{p(r|1)}{p(r|0)}. \tag{12.83a}$$

Similarly, we define the likelihood ratio of a received symbol q_i at the output of a binary-input, Q-ary output DMC as

$$\lambda(q_i)=\frac{P(q_i|1)}{P(q_i|0)}. \tag{12.83b}$$

Using (12.83), we can now express the condition of (12.82) on the threshold T_i that minimizes D_0 as

$$\lambda(T_i)=\sqrt{\lambda(q_i)\lambda(q_{i+1})},\quad i=1,2,\cdots,Q/2. \tag{12.84}$$

Condition (12.84) provides a necessary condition for a set of thresholds $T_i, i=1,2,\cdots,Q/2$ to minimize the Bhattacharyya parameter D_0 and hence also to minimize the bounds of (12.39) on error probability. In words, the likelihood ratio of each quantizer threshold value T_i must equal the geometric mean of the likelihood ratios of the two quantization symbols q_i and q_{i+1} that border T_i. Because (12.84) does not have a closed-form solution, the optimum set of threshold values must be determined using a trial-and-error approach. In Problem 12.22, quantization thresholds are calculated for several values of Q using the optimality condition of (12.84). It is demonstrated that the required SNR E_s/N_0 needed to achieve a given value of D_0 is only slightly larger when $Q=8$ than when no output quantization is used. In other words, an 8-level quantizer involves very little performance loss compared with a continuous-output channel.

Computational Complexity. The Viterbi algorithm must perform 2^ν ACS operations per unit time, one for each state, and each ACS operation involves 2^k additions, one for each branch entering a state, and 2^k-1 binary comparisons. Hence, the computational complexity (decoding time) of the Viterbi algorithm is

proportional to the *branch complexity* $2^k 2^\nu = 2^{k+\nu}$ of the decoding trellis. Thus, as noted previously for decoder memory, the exponential dependence of decoding time on the constraint length ν limits the practical application of the Viterbi algorithm to relatively small values of ν. In addition, since the branch complexity increases exponentially with k, codes with high rates take more time to decode. This computational disadvantage of high-rate codes can be eliminated using a technique called *puncturing* that is discussed in Section 12.7.

High-speed decoding can be achieved with the Viterbi algorithm by employing *parallel processing*. Because the 2^ν ACS operations that must be performed at each time unit are identical, 2^ν identical processors can be used to do the operations in parallel rather than having a single processor do all 2^ν operations serially. Thus, a parallel implementation of the Viterbi algorithm has a factor of 2^ν advantage in speed compared with a serial decoder, but it requires 2^ν times as much hardware. Decoding speed can be further improved by a factor of about 1/3 for a large subclass of nonsystematic feedforward encoders by using a compare–select–add (CSA) operation instead of the usual ACS operation. Details of this *differential Viterbi algorithm* can be found in [25].

Code Performance. Computer simulation results illustrating the performance of the Viterbi algorithm are presented in Figure 12.17. The bit-error probability $P_b(E)$ of the optimum rate $R = 1/2$ codes with constraint lengths $\nu = 2$ through $\nu = 7$ listed in Table 12.1(c) is plotted as a function of the bit SNR E_b/N_0 (in decibels) for a continuous-output AWGN channel in Figure 12.17(a). These simulations are repeated for a BSC, that is, a hard-quantized ($Q = 2$) channel output, in Figure 12.17(b). In both cases the path memory was $\tau = 32$. Note that there is about a 2-dB improvement in the performance of soft decisions (unquantized channel outputs) compared with hard decisions ($Q = 2$). This improvement is illustrated again in Figure 12.17(c), where the performance of the optimum constraint length $\nu = 4$, rate $R = 1/2$ code with $Q = 2, 4, 8$, and ∞ (unquantized outputs) and path memory $\tau = 32$ is shown. Also shown in Figure 12.17(c) is the uncoded curve of (12.37). Comparing this curve with the coding curves shows real coding gains of about 2.4 dB in the hard-decision case ($Q = 2$), 4.4 dB in the quantized ($Q = 8$) soft-decision case, and 4.6 dB in the unquantized ($Q = \infty$) soft-decision case at a bit-error probability of 10^{-5}. Also, we note that there is only about 0.2-dB difference between the $Q = 8$ quantized channel performance and the unquantized ($Q = \infty$) channel performance, suggesting that there is not much to gain by using more than 8 channel-output quantization levels. In Figure 12.17(d), the performance of this same code is shown for path memories $\tau = 8, 16, 32$, and ∞ (no truncation) for a BSC ($Q = 2$) and an unquantized ($Q = \infty$) channel output. Note that in both cases a path memory of $\tau = 8 = 2\nu$ degrades the performance by about 1.25 dB, that $\tau = 16 = 4\nu$ is almost as good as $\tau = 32 = 8\nu$, and that $\tau = 32 = 8\nu$ performs the same as no truncation. (Recall that $\nu = m$ for rate $R = 1/2$ codes.) The performance of the optimum rate $R = 1/3$ codes with constraint lengths $\nu = 3, 5$, and 7 listed in Table 12.1(b) is shown in Figure 12.17(e) for both a continuous-output AWGN channel and a BSC. Note that these codes do better than the corresponding rate $R = 1/2$ codes of Figures 12.17(a) and (b) by between 0.25 dB and 0.5 dB. This is because the coding gain, that is, the product of R and d_{free} in decibels, is larger at

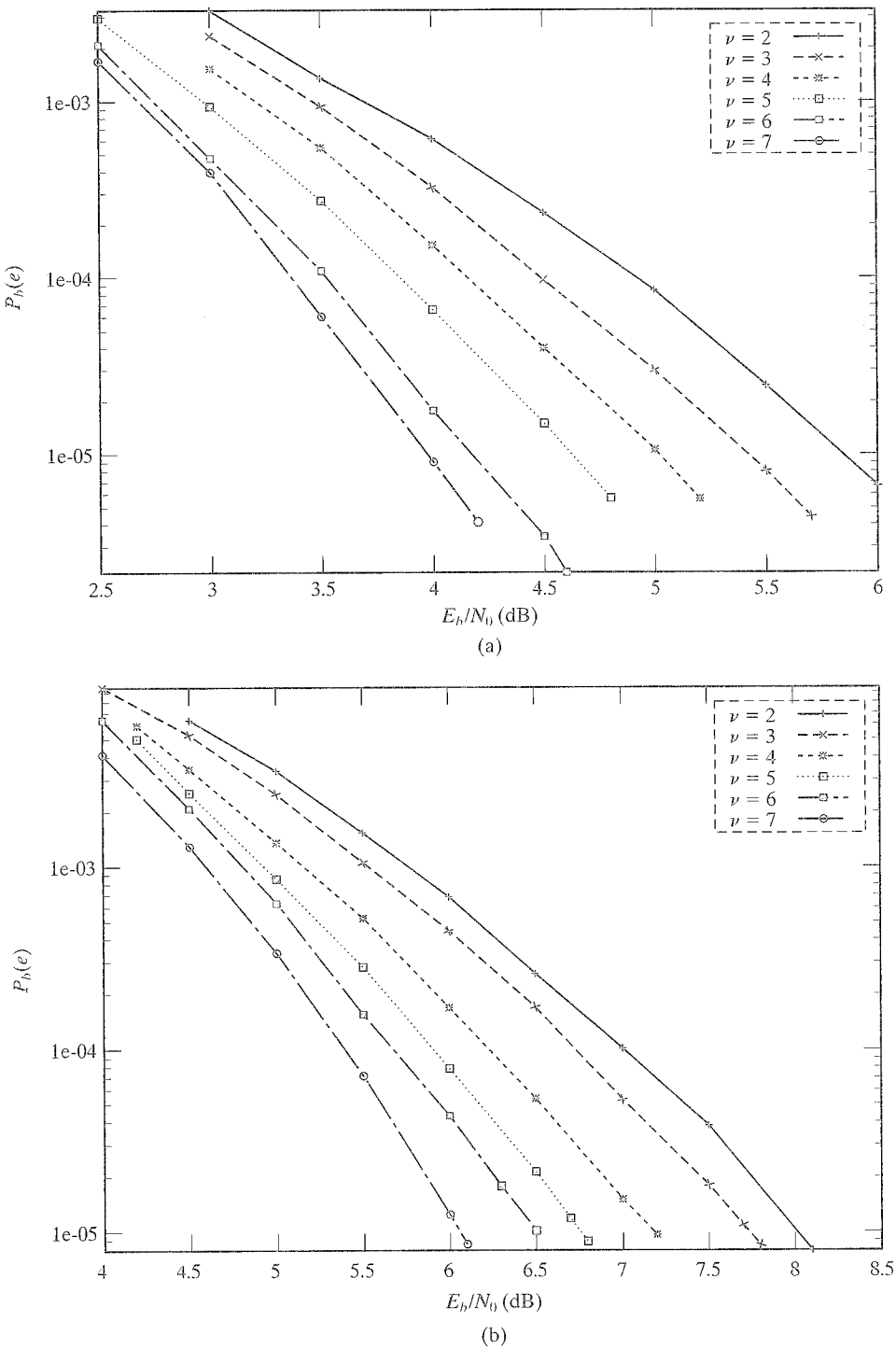
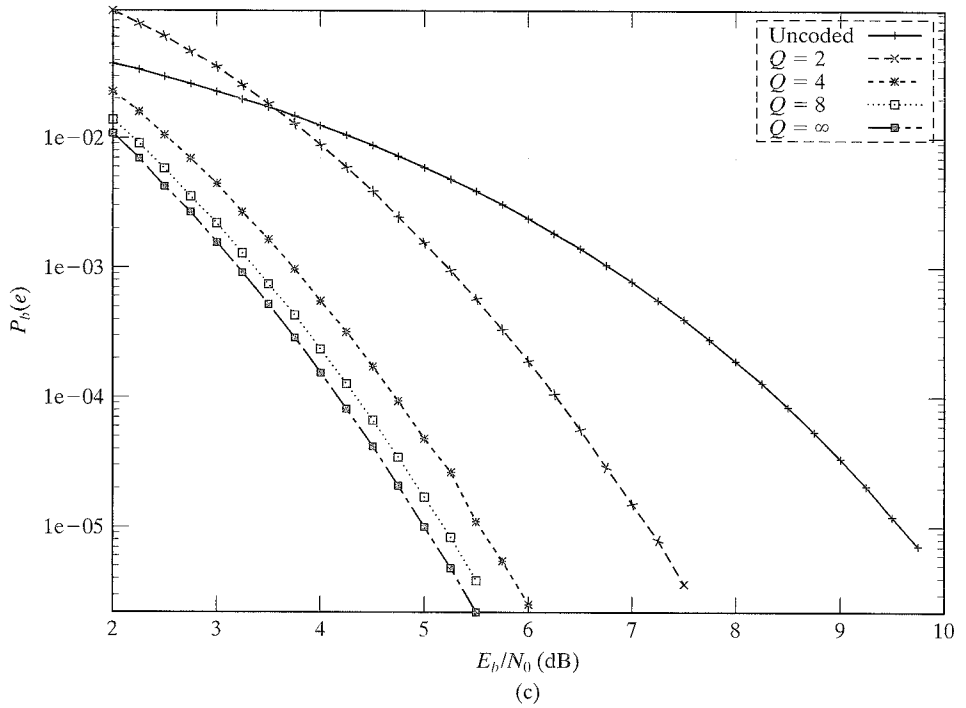
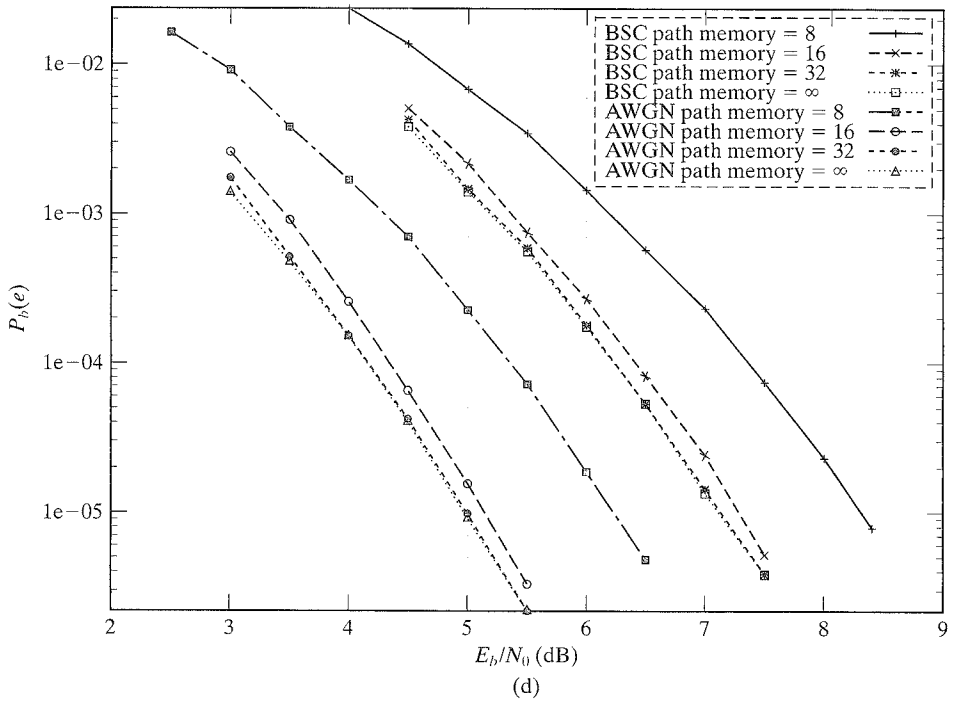


FIGURE 12.17: Simulation results for the Viterbi algorithm.

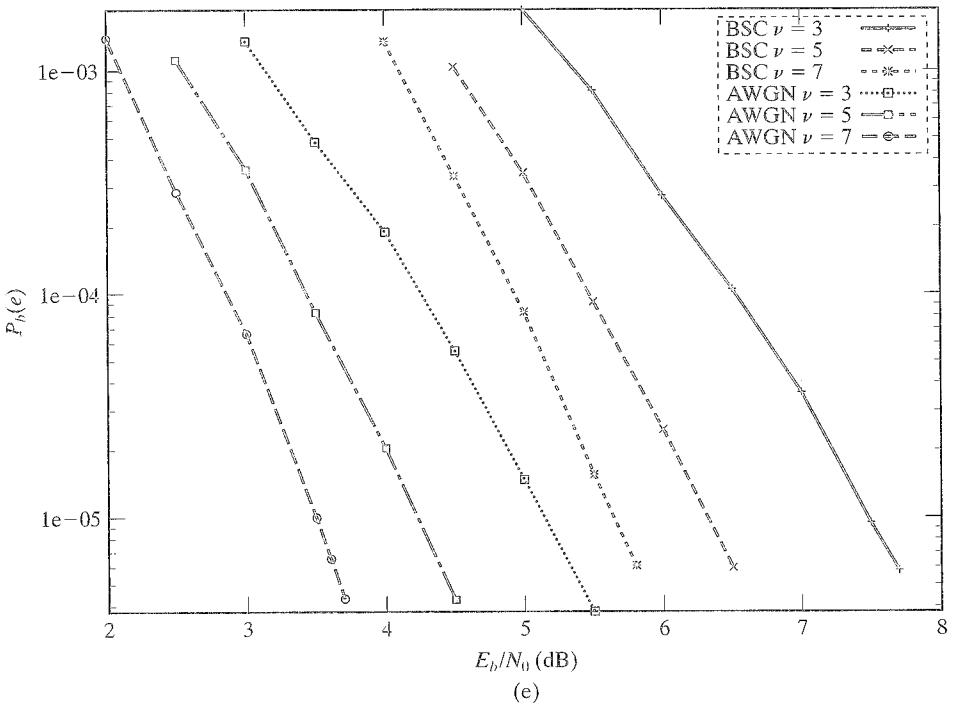


(c)

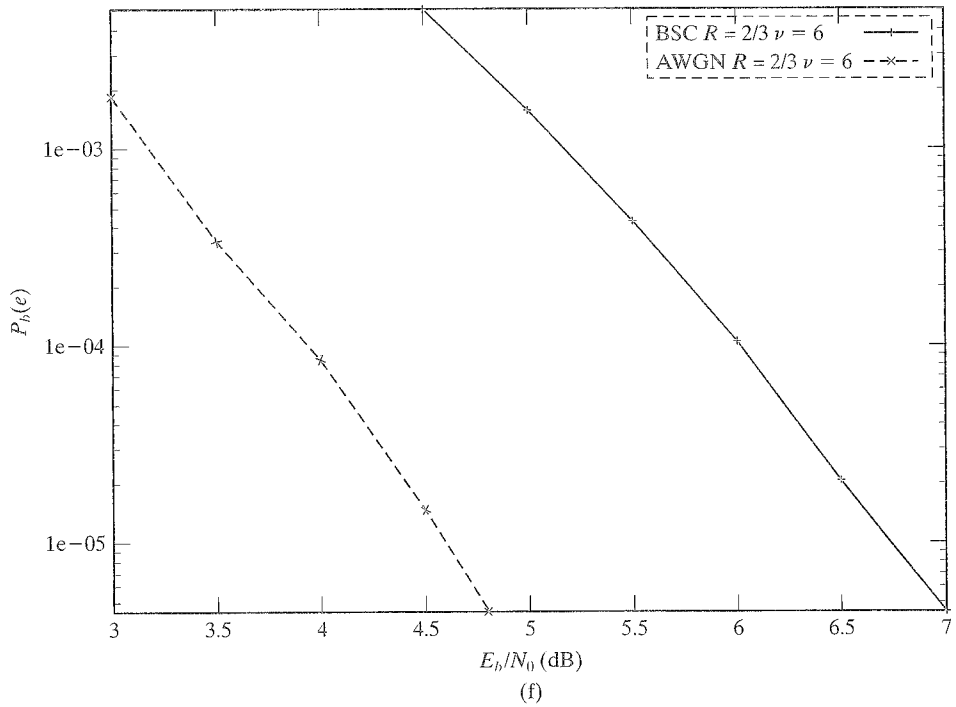


(d)

FIGURE 12.17: (*continued*)



(e)



(f)

FIGURE 12.17: (*continued*)

$R = 1/3$ than at $R = 1/2$ for the same constraint length.[5] Finally, Figure 12.17(f) shows the performance of the optimum rate $R = 2/3$ code with $\nu = 6$ listed in Table 12.1(d) for both a continuous-output AWGN channel and a BSC. Again, note that the corresponding rate $R = 1/2$ code with $\nu = 6$ in Figures 12.17(a) and (b) does better than the rate $R = 2/3$ code by about 0.4 dB and 0.25 dB, respectively. All these observations are consistent with the performance analysis presented earlier in this chapter.

12.5 THE SOFT-OUTPUT VITERBI ALGORITHM (SOVA)

The subject of code concatenation, in which two or more encoders (and decoders) are connected in series or in parallel, will be discussed in Chapters 15 and 16. In a concatenated decoding system, it is common for one decoder to pass reliability (confidence) information about its decoded outputs, so-called soft outputs, to a second decoder. This allows the second decoder to use soft-decision decoding, as opposed to simply processing the hard decisions made by the first decoder. Decoders that accept soft-input values from the channel (or from another decoder) and deliver soft-output values to another decoder are referred to as *soft-in, soft-out* (SISO) decoders.

The Soft-Output Viterbi Algorithm (SOVA) was first introduced in 1989 in a paper by Hagenauer and Hoeher [9]. We describe the SOVA here for the case of rate $R = 1/n$ convolutional codes used on a binary-input, continuous-output AWGN channel; that is, we describe a SISO version of the Viterbi algorithm. In our presentation of the SOVA we deviate from the usual assumption that the a priori probabilities of the information bits, $P(u_l)$, $l = 0, 1, \cdots, h - 1$, are equally likely by allowing the possibility of non-equally likely a priori probabilities. This generality is necessary to apply a SISO decoding algorithm to an iterative decoding procedure, such as those to be discussed in Chapter 16.

The basic operation of the SOVA is identical to the Viterbi algorithm. The only difference is that a reliability indicator is attached to the hard-decision output for each information bit. The combination of the hard-decision output and the reliability indicator is called a *soft output.* At time unit $l = t$, the partial path metric that must be maximized by the Viterbi algorithm for a binary-input, continuous-output AWGN channel given the partial received sequence $[\mathbf{r}]_t = (\mathbf{r}_0, \mathbf{r}_1, \cdots, \mathbf{r}_{t-1}) = (r_0^{(0)}, r_0^{(1)}, \cdots, r_0^{(n-1)}; r_1^{(0)}, r_1^{(1)}, \cdots, r_1^{(n-1)}; \cdots; r_{t-1}^{(0)}, r_{t-1}^{(1)}, \cdots, r_{t-1}^{(n-1)})$ can be written as

$$M([\mathbf{r}|\mathbf{v}]_t) = \ln\{p([\mathbf{r}|\mathbf{v}]_t)P([\mathbf{v}]_t)\}. \tag{12.85}$$

Equation (12.85) differs from (12.14) only in the inclusion of the a priori path probability $P([\mathbf{v}]_t)$, since these will, in general, not be equally likely when the a priori probabilities of the information bits are not equally likely. Noting that the a priori path probability $P([\mathbf{v}]_t)$ is simply the a priori probability of the associated information sequence $[\mathbf{u}]_t$, we can separate the contribution to the metric before

[5] Recall from Chapter 1, however, that a rate $R = 1/3$ code requires more channel bandwidth expansion than a rate $R = 1/2$ code.

time t from the contribution at time t as follows:

$$\begin{aligned} M([\mathbb{r}|\mathbb{v}]_t) &= \ln\left\{\left[\prod_{l=0}^{t-1} p(\mathbb{r}_l|\mathbb{v}_l)P(u_l)\right] p(\mathbb{r}_t|\mathbb{v}_t)P(u_t)\right\} \\ &= \ln\left\{\prod_{l=0}^{t-1} p(\mathbb{r}_l|\mathbb{v}_l)P(u_l)\right\} + \ln\left\{\left[\prod_{j=0}^{n-1} p(r_t^{(j)}|v_t^{(j)})\right] P(u_t)\right\} \qquad (12.86) \\ &= \ln\left\{\prod_{l=0}^{t-1} p(\mathbb{r}_l|\mathbb{v}_l)P(u_l)\right\} + \left\{\sum_{j=0}^{n-1} \ln\left[p(r_t^{(j)}|v_t^{(j)})\right] + \ln[P(u_t)]\right\}. \end{aligned}$$

We now modify the time t term in (12.86) by multiplying each term in the sum by 2 and introducing constants $C_r^{(j)}$ and C_u as follows:

$$\left\{\sum_{j=0}^{n-1}\left[2\ln\left[p(r_t^{(j)}|v_t^{(j)})\right] - C_r^{(j)}\right] + [2\ln[P(u_t)] - C_u]\right\}, \qquad (12.87)$$

where the constants

$$C_r^{(j)} \equiv \ln\left[p(r_t^{(j)}|v_t^{(j)} = +1)\right] + \ln\left[p(r_t^{(j)}|v_t^{(j)} = -1)\right], \quad j = 0, 1, \cdots, n-1 \qquad (12.88a)$$

$$C_u \equiv \ln[P(u_t = +1)] + \ln[P(u_t = -1)] \qquad (12.88b)$$

are independent of the path $[\mathbb{v}]_t$, and we assume the mapping $1 \to +1$ and $0 \to -1$. Similarly modifying each of the terms in (12.86) before time t and noting that the modifications do not affect the path $[\mathbb{v}]_t$ that maximizes (12.86), we can express the modified metric as (see Problem 12.23)

$$\begin{aligned} M^*([\mathbb{r}|\mathbb{v}]_t) &= M^*([\mathbb{r}|\mathbb{v}]_{t-1}) + \sum_{j=0}^{n-1}\left\{2\ln\left[p(r_t^{(j)}|v_t^{(j)})\right] - C_r^{(j)}\right\} + \left\{2\ln[P(u_t)] - C_u\right\} \\ &= M^*([\mathbb{r}|\mathbb{v}]_{t-1}) + \sum_{j=0}^{n-1} v_t^{(j)} \ln\left[\frac{p(r_t^{(j)}|v_t^{(j)} = +1)}{p(r_t^{(j)}|v_t^{(j)} = -1)}\right] + u_t \ln\left[\frac{p(u_t = +1)}{p(u_t = -1)}\right]. \end{aligned} \qquad (12.89)$$

We can simplify the expression for the modified metric in (12.89) by defining the *log-likelihood ratio*, or *L-value*, of a received symbol r at the output of an unquantized channel with binary inputs $v = \pm 1$ as

$$L(r) = \ln\left[\frac{p(r|v = +1)}{p(r|v = -1)}\right]. \qquad (12.90)$$

(Note that $L(r) = \ln[\lambda(r)]$, defined in (12.83a).) Similarly, the L-value of an information bit u is defined as

$$L(u) = \ln\left[\frac{p(u = +1)}{p(u = -1)}\right]. \qquad (12.91)$$

An L-value can be interpreted as a measure of reliability for a binary random variable. For example, assuming that the a priori (before transmission) probabilities of the code bits v are equally likely, that is, $P(v = +1) = P(v = -1) = 1/2$, an assumption that always holds when the information bits are equally likely and the code is linear, we can rewrite (12.90) using Bayes' rule as follows:

$$L(r) = \ln\left[\frac{p(r|v=+1)}{p(r|v=-1)}\right] = \ln\left[\frac{p(v=+1|r)}{p(v=-1|r)}\right]. \tag{12.92}$$

From (12.92) we see that given the received symbol r, a large positive value of $L(r)$ indicates a high reliability that $v = +1$, a large negative value of $L(r)$ indicates a high reliability that $v = -1$, and a value of $L(r)$ near 0 indicates that a decision about the value of v based only on r is very unreliable. Similarly, a large positive value of $L(u)$ indicates a high reliability that $u = +1$.

For an AWGN channel with received symbol r', binary input symbols $v' = \pm\sqrt{E_s}$, and SNR E_s/N_0, we obtain (see Problem 12.24)

$$L(r') = (4\sqrt{E_s}/N_0)r'. \tag{12.93a}$$

We can rewrite (12.93a) as

$$L(r) = (4E_s/N_0)r, \tag{12.93b}$$

where the normalized value $r \equiv r'/\sqrt{E_s}$ corresponds to the received symbol with binary-input symbols $v = \pm 1$. Equation (12.93b) illustrates that the reliability (L-value) associated with a received symbol r, assuming a transmitted symbol $v = \pm 1$, increases linearly with r and that the constant of proportionality increases linearly with the channel SNR E_s/N_0, an intuitively appealing result. Defining $L_c \equiv 4E_s/N_0$ as the *channel reliability factor*, we can now express the modified metric for SOVA decoding as

$$M^*([\mathbb{r}|\mathbb{v}]_t) = M^*([\mathbb{r}|\mathbb{v}]_{t-1}) + \sum_{j=0}^{n-1} L_c v_t^{(j)} r_t^{(j)} + u_t L(u_t), \tag{12.94}$$

where the summation term in (12.94), along with the corresponding terms in $M^*([\mathbb{r}|\mathbb{v}]_{t-1})$, represents the correlation metric first noted in (12.14). In other words, the constants C_1 and C_2 in the metric of (12.14) are absorbed into the term L_c in the modified metric of (12.94), and the term $u_t L(u_t)$ in (12.94) does not appear in (12.14), since the a priori probabilities of the information bits were assumed to be equally likely in that case.

As noted previously, the SOVA operates exactly like the Viterbi algorithm, except that it appends a reliability measure to each hard-decision output. Assume that a comparison is being made at state $S_i, i = 0, 1, \cdots, 2^\nu - 1$, between the maximum likelihood (ML) path $[\mathbb{v}]_t$ and an incorrect path $[\mathbb{v}']_t$ at time $l = t$. We define the metric difference as

$$\Delta_{t-1}(S_i) = \frac{1}{2}\{M^*([\mathbb{r}|\mathbb{v}]_t) - M^*([\mathbb{r}|\mathbb{v}']_t)\}. \tag{12.95}$$

The probability $P(C)$ that the ML path is correctly selected at time t is given by

$$P(C) = \frac{P([\mathbb{v}|\mathbb{r}]_t)}{\{P([\mathbb{v}|\mathbb{r}]_t) + P([\mathbb{v}'|\mathbb{r}]_t)\}}. \tag{12.96}$$

We note that

$$P([\mathbb{v}|\mathbb{r}]_t) = \frac{p([\mathbb{r}|\mathbb{v}]_t)P([\mathbb{v}]_t)}{p(\mathbb{r})} = \frac{e^{M([\mathbb{r}|\mathbb{v}]_t)}}{p(\mathbb{r})} \tag{12.97a}$$

and

$$P([\mathbb{v}'|\mathbb{r}]_t) = \frac{p([\mathbb{r}|\mathbb{v}']_t)P([\mathbb{v}']_t)}{p(\mathbb{r})} = \frac{e^{M([\mathbb{r}|\mathbb{v}']_t)}}{p(\mathbb{r})}. \tag{12.97b}$$

Using the modifications to the metric made in (12.87), we can write

$$M^*([\mathbb{r}|\mathbb{v}]_t) = 2M([\mathbb{r}|\mathbb{v}]_t) - c \tag{12.98a}$$

and

$$M^*([\mathbb{r}|\mathbb{v}']_t) = 2M([\mathbb{r}|\mathbb{v}']_t) - c, \tag{12.98b}$$

where c is a constant that does not depend on $[\mathbb{v}]_t$ or $[\mathbb{v}']_t$ (see Problem 12.25). Now, we can rewrite (12.96) using (12.97) and (12.98) as

$$\begin{aligned} P(C) &= \frac{[e^{\{M^*([\mathbb{r}|\mathbb{v}]_t)/2\}+c}/p(\mathbb{r})]}{[e^{\{M^*([\mathbb{r}|\mathbb{v}]_t)/2\}+c}/p(\mathbb{r})] + [e^{\{M^*([\mathbb{r}|\mathbb{v}']_t)/2\}+c}/p(\mathbb{r})]} \\ &= \frac{e^{M^*([\mathbb{r}|\mathbb{v}]_t)/2}}{e^{M^*([\mathbb{r}|\mathbb{v}]_t)/2} + e^{M^*([\mathbb{r}|\mathbb{v}']_t)/2}} \\ &= \frac{e^{\Delta_{t-1}(S_i)}}{1 + e^{\Delta_{t-1}(S_i)}}. \end{aligned} \tag{12.99}$$

Finally, the log-likelihood ratio, or *reliability*, of this path decision is given by

$$\ln\left\{\frac{P(C)}{[1 - P(C)]}\right\} = \Delta_{t-1}(S_i). \tag{12.100}$$

We now show how the reliability of a path decision is associated with the hard-decision outputs of a Viterbi decoder. First, consider the path decisions made at time $l = m + 1$, that is, the initial set of 2^ν path decisions. The path decision at state S_i has reliability $\Delta_m(S_i)$, as shown in (12.100). The modulo-2 sum of the two information sequences being compared, that is, $[\mathbb{u}]_m$ corresponding to the codeword $[\mathbb{v}]_m$ and $[\mathbb{u}']_m$ corresponding to the codeword $[\mathbb{v}']_m$, indicates the *error positions*, namely, the bit positions in which the two information sequences differ. These are the information bits that are affected by the reliability of this path decision; that is, when these bits are decoded, the reliability of these bit decisions will depend on the reliability of this path decision. On the other hand, the reliability of the decisions made for those bit positions where $[\mathbb{u}]_m$ and $[\mathbb{u}']_m$ are the same will not depend on the reliability of this path decision. Thus, we form a *reliability vector* at time $l = m + 1$ for state S_i as follows:

$$\mathbb{L}_{m+1}(S_i) = [L_0(S_i), L_1(S_i), \cdots, L_m(S_i)], \tag{12.101}$$

where

$$L_l(S_i) \equiv \begin{cases} \Delta_m(S_i) & \text{if } u_l \neq u'_l \\ \infty & \text{if } u_l = u'_l \end{cases}, \quad l = 0, 1, \cdots, m. \tag{12.102}$$

In other words, the reliability of a bit position is either ∞, if it is not affected by the path decision, or $\Delta_m(S_i)$, the reliability of the path decision, if it is affected by the path decision. In the same way, we can obtain a reliability vector $\mathbb{L}_{m+1}(S_i)$ for each state S_i, $i = 0, 1, \cdots, 2^\nu - 1$, at time $l = m + 1$.

At time $l = m + 2$, the path decisions result in reliabilities $\Delta_{m+1}(S_i)$, and the reliability vectors are given by $\mathbb{L}_{m+2}(S_i) = [L_0(S_i), L_1(S_i), \cdots, L_{m+1}(S_i)]$, $i = 0, 1, \cdots, 2^\nu - 1$. The reliability vectors are updated by first determining $L_{m+1}(S_i)$ as in (12.102) and then, for the remaining entries, taking the minimum of $\Delta_{m+1}(S_i)$ and the previous entry in the reliability vector for the error positions. The previous entries in the reliability vector for the bit positions in which $[\mathfrak{u}]_{m+1}$ and $[\mathfrak{u}']_{m+1}$ are the same are left unchanged. This update procedure is repeated after each path decision, and for each state, a reliability vector whose length is equal to the surviving path must be stored along with the surviving path and its metric. For example, at time $l = t$ and for state S_i, we update the reliability vector as follows:

$$\mathbb{L}_t(S_i) = [L_0(S_i), L_1(S_i), \cdots, L_{t-1}(S_i)], \tag{12.103}$$

where

$$L_l(S_i) \rightarrow \begin{cases} \min[\Delta_{t-1}(S_i),\ L_l(S_i)] & \text{if } u_l \neq u'_l \\ L_l(S_i) & \text{if } u_l = u'_l \end{cases}, \quad l = 0, 1, \cdots, t-1. \tag{12.104}$$

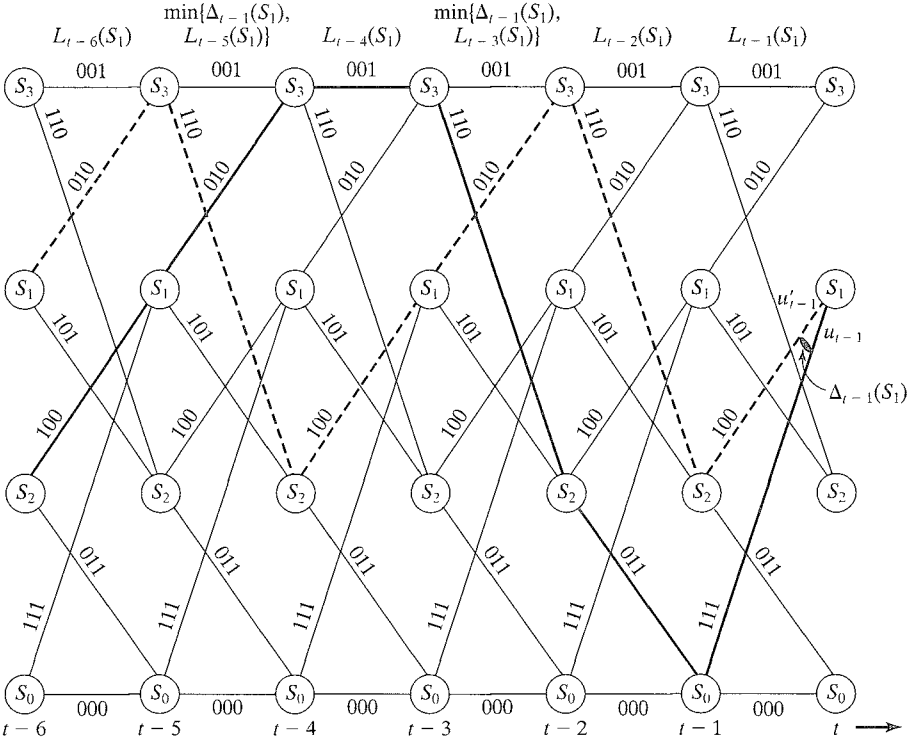


FIGURE 12.18: Updating the reliability vector following a path decision at time t.

In Figure 12.18 we illustrate the reliability vector update procedure that follows a path decision at time unit t. (The illustration assumes that the "up" branch leaving each state represents an input 1, whereas the "down" branch represents an input 0.)

Typically, updating a reliability vector involves changing only a few of the entries. The determination of the error positions requires no additional storage, since the information sequences are stored by the algorithm as surviving paths. (We note here that for feedforward encoders, $[\mathbb{u}]_{t-1}$ and $[\mathbb{u}']_{t-1}$ always agree in the m most recent bit positions, but that this is not true in general for feedback encoders.) The reliability vector associated with a state keeps track of the reliabilities of all the information bits along the surviving path for that state. When the end of the decoding trellis is reached, the reliability vector $\mathbb{L}_{m+h}(S_0)$ associated with the final survivor provides the soft outputs.

The operation of the SOVA with finite path memory τ is essentially the same as described previously (see Problem 12.26). In this case, the reliability vectors store only the reliabilities of the τ most recent bits on each surviving path. Thus, the reliability vectors have the same memory as the paths. Bits are decoded exactly as in the truncated Viterbi algorithm, that is, at each time unit the survivor with the best metric is selected, and the bit τ time units back on that path is decoded. In the SOVA the reliability associated with that bit becomes the soft output.

The storage complexity of the SOVA is greater than that of the Viterbi algorithm because the SOVA must store 2^ν reliability vectors, one for each state. Computational complexity is also increased, because of the need to update a reliability vector after each ACS operation; however, the overall increase in complexity is modest, and the SOVA always decodes the ML path and, in addition, provides soft outputs for concatenated SISO decoding applications. In the next section, we introduce MAP decoding of convolutional codes, a decoding algorithm that maximizes the probability of correct decoding for each information bit and also provides soft outputs.

12.6 THE BCJR ALGORITHM

Given a received sequence $\mathbb{r}$, the Viterbi algorithm finds the codeword $\mathbb{v}$ that maximizes the log-likelihood function. For a BSC, this is equivalent to finding the (binary) codeword $\mathbb{v}$ that is closest to the (binary) received sequence $\mathbb{r}$ in Hamming distance. For the more general continuous-output AWGN channel, this is equivalent to finding the (binary) codeword $\mathbb{v}$ that is closest to the (real-valued) received sequence $\mathbb{r}$ in Euclidean distance. Once the ML codeword $\mathbb{v}$ is determined, its corresponding information sequence $\mathbb{u}$ becomes the decoded output.

Because the Viterbi algorithm finds the most likely codeword, it minimizes the WER $P_w(E)$, that is, the probability $P(\hat{\mathbb{v}} \neq \mathbb{v}|\mathbb{r})$ that the decoded (ML) codeword $\hat{\mathbb{v}}$ is not equal to the transmitted codeword $\mathbb{v}$, given the received sequence $\mathbb{r}$. In many cases, however, we are interested in minimizing the BER $P_b(E)$, that is, the probability $P(\hat{u}_l \neq u_l|\mathbb{r})$ that the decoded information bit $\hat{u}_l$ at time l is not equal to the transmitted information bit u_l at time l, $l = 0, 1, \ldots, K^* - 1$, rather than the WER $P_w(E)$. To minimize the BER, the a posteriori probability $P(\hat{u}_l = u_l|\mathbb{r})$ that an information bit u_l is correctly decoded must be maximized. An algorithm that maximizes $P(\hat{u}_l = u_l|\mathbb{r})$ is called a maximum a posteriori probability (MAP) decoder. When the information bits are equally likely, the (ML) Viterbi algorithm

maximizes $P(\hat{\mathbf{v}} = \mathbf{v}|\mathbf{r})$. Although this does not guarantee that $P(\hat{u}_l = u_l|\mathbf{r})$ is also maximized, $P(\hat{\mathbf{v}} = \mathbf{v}|\mathbf{r})$ is closely related to $P(\hat{u}_l = u_l|\mathbf{r})$, as shown next, and the Viterbi algorithm results in near-optimum BER performance.

The Viterbi algorithm maximizes $P(\hat{\mathbf{v}} = \mathbf{v}|\mathbf{r}) = P(\hat{\mathbf{u}} = \mathbf{u}|\mathbf{r})$, where $\hat{\mathbf{u}}$ and $\mathbf{u}$ are the information sequences corresponding to the codewords $\hat{\mathbf{v}}$ and $\mathbf{v}$, respectively. This is equivalent to minimizing $P(\hat{\mathbf{u}} \neq \mathbf{u}|\mathbf{r})$. The BER is given by

$$P(\hat{u}_l \neq u_l|\mathbf{r}) = \left[\frac{d(\hat{\mathbf{u}}, \mathbf{u})}{K^*}\right] P(\hat{\mathbf{u}} \neq \mathbf{u}|\mathbf{r}), \quad l = 0, 1, \cdots, K^* - 1. \tag{12.105}$$

We see from (12.105) that the BER depends on the Hamming distance between $\hat{\mathbf{u}}$ and $\mathbf{u}$ as well as on the WER $P(\hat{\mathbf{u}} \neq \mathbf{u}|\mathbf{r})$. Thus, minimizing $P(\hat{u}_l \neq u_l|\mathbf{r})$ also involves selecting an encoder with the property that low-weight codewords correspond to low-weight information sequences. This choice ensures that the most likely codeword errors, i.e., those with low weight, result in a small number of information bit errors, so that the BER is also minimized. We note that the class of systematic encoders satisfies this property, whereas the class of catastrophic encoders does not.

In 1974 Bahl, Cocke, Jelinek, and Raviv [5] introduced a MAP decoder, called the *BCJR algorithm*, that can be applied to any linear code, block or convolutional. The computational complexity of the BCJR algorithm is greater than that of the Viterbi algorithm, and thus Viterbi decoding is preferred in the case of equally likely information bits. When the information bits are not equally likely, however, better performance is achieved with MAP decoding. Also, when iterative decoding is employed (see, for example, Chapter 16), and the a priori probabilities of the information bits change from iteration to iteration, a MAP decoder gives the best performance.

In this section we describe the BCJR algorithm for the case of rate $R = 1/n$ convolutional codes used on a binary-input, continuous-output AWGN channel and on a DMC. (The application of the BCJR algorithm to block codes is covered in Chapter 14.) Our presentation is based on the log-likelihood ratios, or L-values, introduced in the previous section. The decoder inputs are the received sequence $\mathbf{r}$ and the a priori L-values of the information bits $L_a(u_l)$, $l = 0, 1, \cdots, h - 1$. As in the case of the SOVA, we do *not* assume that the information bits are equally likely. The algorithm calculates the a posteriori L-values

$$L(u_l) \equiv \ln\left[\frac{P(u_l = +1|\mathbf{r})}{P(u_l = -1|\mathbf{r})}\right], \tag{12.106}$$

called the *APP L-values*, of each information bit, and the decoder output is given by

$$\hat{u}_l = \begin{cases} +1 \text{ if } L(u_l) > 0 \\ -1 \text{ if } L(u_l) < 0 \end{cases}, \quad l = 0, 1, \cdots, h - 1. \tag{12.107}$$

In iterative decoding, the APP L-values can be taken as the decoder outputs, resulting in a SISO decoding algorithm.

We begin our development of the BCJR algorithm by rewriting the APP value $P(u_l = +1|\mathbf{r})$ as follows:

$$P(u_l = +1|\mathbf{r}) = \frac{p(u_l = +1, \mathbf{r})}{P(\mathbf{r})} = \frac{\sum_{\mathbf{u} \in \mathbb{U}_l^+} p(\mathbf{r}|\mathbf{v}) P(\mathbf{u})}{\sum_{\mathbf{u}} p(\mathbf{r}|\mathbf{v}) P(\mathbf{u})}, \tag{12.108}$$

where $\mathbb{U}_l^+$ is the set of all information sequences $\mathbb{u}$ such that $u_l = +1$, $\mathbb{v}$ is the transmitted codeword corresponding to the information sequence $\mathbb{u}$, and $p(\mathbb{r}|\mathbb{v})$ is the pdf of the received sequence $\mathbb{r}$ given $\mathbb{v}$. Rewriting $P(u_l = -1|\mathbb{r})$ in the same way, we can write the expression in (12.106) for the APP L-value as

$$L(u_l) = \ln\left[\frac{\sum_{\mathbb{u}\in\mathbb{U}_l^+} p(\mathbb{r}|\mathbb{v})P(\mathbb{u})}{\sum_{\mathbb{u}\in\mathbb{U}_l^-} p(\mathbb{r}|\mathbb{v})P(\mathbb{u})}\right], \tag{12.109}$$

where $\mathbb{U}_l^-$ is the set of all information sequences $\mathbb{u}$ such that $u_l = -1$. MAP decoding can be achieved by computing the APP L-values $L(u_l), l = 0, 1, \cdots, h-1$ directly from (12.109) and then applying (12.107); however, except for very short block lengths h, the amount of computation required is prohibitive. For codes with a trellis structure and a reasonable number of states, such as short constraint length convolutional codes and some block codes, employing a recursive computational procedure based on the trellis structure of the code considerably simplifies the process.

First, making use of the trellis structure of the code, we can reformulate (12.108) as follows:

$$P(u_l = +1|\mathbb{r}) = \frac{p(u_l = +1, \mathbb{r})}{P(\mathbb{r})} = \frac{\sum_{(s',s)\in\Sigma_l^+} p(s_l = s', s_{l+1} = s, \mathbb{r})}{P(\mathbb{r})}, \tag{12.110}$$

where Σ_l^+ is the set of all state pairs $s_l = s'$ and $s_{l+1} = s$ that correspond to the input bit $u_l = +1$ at time l. Reformulating the expression $P(u_l = -1|\mathbb{r})$ in the same way, we can now write (12.106) for the APP L-value as

$$L(u_l) = \ln\left\{\frac{\sum_{(s',s)\in\Sigma_l^+} p(s_l = s', s_{l+1} = s, \mathbb{r})}{\sum_{(s',s)\in\Sigma_l^-} p(s_l = s', s_{l+1} = s, \mathbb{r})}\right\}, \tag{12.111}$$

where Σ_l^- is the set of all state pairs $s_l = s'$ and $s_{l+1} = s$ that correspond to the input bit $u_l = -1$ at time l. Equations (12.109) and(12.111) are equivalent expressions for the APP L-value $L(u_l)$, but whereas the summations in (12.109) extend over a set of 2^{h-1} information sequences, the summations in (12.111) extend only over a set of 2^ν state pairs. Hence, for large block lengths h, (12.111) is considerably simpler to evaluate. Note that every branch in the trellis connecting a state at time l to a state at time $l+1$ is included in one of the summations in (12.111).

We now show how the joint pdf's $p(s', s, \mathbb{r})$ in (12.111) can be evaluated recursively. We begin by writing

$$p(s', s, \mathbb{r}) = p(s', s, \mathbb{r}_{t<l}, \mathbb{r}_l, \mathbb{r}_{t>l}), \tag{12.112}$$

where $\mathbb{r}_{t<l}$ represents the portion of the received sequence $\mathbb{r}$ before time l, and $\mathbb{r}_{t>l}$ represents the portion of the received sequence $\mathbb{r}$ after time l. Now, application of Bayes' rule yields

$$\begin{aligned} p(s', s, \mathbb{r}) &= p(\mathbb{r}_{t>l}|s', s, \mathbb{r}_{t<l}, \mathbb{r}_l)p(s', s, \mathbb{r}_{t<l}, \mathbb{r}_l) \\ &= p(\mathbb{r}_{t>l}|s', s, \mathbb{r}_{t<l}, \mathbb{r}_l)p(s, \mathbb{r}_l|s', \mathbb{r}_{t<l})p(s', \mathbb{r}_{t<l}) \\ &= p(\mathbb{r}_{t>l}|s)p(s, \mathbb{r}_l|s')p(s', \mathbb{r}_{t<l}), \end{aligned} \tag{12.113}$$

where the last equality follows from the fact that the probability of the received branch at time l depends only on the state and input bit at time l. Defining

$$\alpha_l(s') \equiv p(s', \mathbb{r}_{t<l}) \tag{12.114a}$$

$$\gamma_l(s', s) \equiv p(s, \mathbb{r}_l | s') \tag{12.114b}$$

$$\beta_{l+1}(s) \equiv p(\mathbb{r}_{t>l} | s), \tag{12.114c}$$

we can write (12.113) as

$$p(s', s, \mathbb{r}) = \beta_{l+1}(s)\gamma_l(s', s)\alpha_l(s'). \tag{12.115}$$

We can now rewrite the expression for the probability $\alpha_{l+1}(s)$ as

$$\begin{aligned}
\alpha_{l+1}(s) = p(s, \mathbb{r}_{t<l+1}) &= \sum_{s' \in \sigma_l} p(s', s, \mathbb{r}_{t<l+1}) \\
&= \sum_{s' \in \sigma_l} p(s, \mathbb{r}_l | s', \mathbb{r}_{t<l}) p(s', \mathbb{r}_{t<l}) \\
&= \sum_{s' \in \sigma_l} p(s, \mathbb{r}_l | s') p(s', \mathbb{r}_{t<l}) \\
&= \sum_{s' \in \sigma_l} \gamma_l(s', s)\alpha_l(s'),
\end{aligned} \tag{12.116}$$

where σ_l is the set of all states at time l. Thus, we can compute a *forward metric* $\alpha_{l+1}(s)$ for each state s at time $l+1$ using the *forward recursion* (12.116). Similarly, we can write the expression for the probability $\beta_l(s')$ as (see Problem 12.27)

$$\beta_l(s') = \sum_{s \in \sigma_{l+1}} \gamma_l(s', s)\beta_{l+1}(s), \tag{12.117}$$

where σ_{l+1} is the set of all states at time $l+1$, and we can compute a *backward metric* $\beta_l(s')$ for each state s' at time l using the *backward recursion* (12.117). The forward recursion begins at time $l = 0$ with the initial condition

$$\alpha_0(s) = \begin{cases} 1, & s = \mathbb{0} \\ 0, & s \neq \mathbb{0} \end{cases}, \tag{12.118}$$

since the encoder starts in the all-zero state $S_0 = \mathbb{0}$, and we use (12.116) to recursively compute $\alpha_{l+1}(s)$, $l = 0, 1, \cdots, K-1$, where $K = h + m$ is the length of the input sequence. Similarly, the backward recursion begins at time $l = K$ with the initial condition

$$\beta_K(s) = \begin{cases} 1, & s = \mathbb{0} \\ 0, & s \neq \mathbb{0} \end{cases}, \tag{12.119}$$

since the encoder also ends in the all-zero state $S_0 = \mathbb{0}$, and we use (12.117) to recursively compute $\beta_l(s)$, $l = K-1, K-2, \cdots, 0$.

We can write the *branch metric* $\gamma_l(s', s)$ as

$$\begin{aligned}\gamma_l(s', s) &= p(s, \mathbb{r}_l|s') = \frac{p(s', s, \mathbb{r}_l)}{P(s')} \\ &= \left[\frac{P(s', s)}{P(s')}\right]\left[\frac{p(s', s, \mathbb{r}_l)}{P(s', s)}\right] \\ &= P(s|s')p(\mathbb{r}_l|s', s) = P(u_l)p(\mathbb{r}_l|\mathbb{v}_l),\end{aligned} \tag{12.120}$$

where u_l is the input bit and $\mathbb{v}_l$ the output bits corresponding to the state transition $s' \to s$ at time l. For a continuous-output AWGN channel, if $s' \to s$ is a valid state transition,

$$\gamma_l(s', s) = P(u_l)p(\mathbb{r}_l|\mathbb{v}_l) = P(u_l)\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n e^{-\frac{E_s}{N_0}||\mathbb{r}_l - \mathbb{v}_l||^2}, \tag{12.121}$$

where $||\mathbb{r}_l - \mathbb{v}_l||^2$ is the squared Euclidean distance between the (normalized by $\sqrt{E_s}$) received branch $\mathbb{r}_l$ and the transmitted branch $\mathbb{v}_l$ at time l; however, if $s' \to s$ is not a valid state transition, $P(s|s')$ and $\gamma_l(s', s)$ are both zero. The algorithm that computes the APP L-values $L(u_l)$ using (12.111), (12.115), and the metrics defined in (12.116)–(12.119) and (12.121) is called the MAP algorithm.

We now introduce some modifications to the algorithm just outlined that result in greater computational efficiency. First, we note from (12.115)–(12.117) and (12.121) that the constant term $\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n$ always appears raised to the power h in the expression for the pdf $p(s', s, \mathbb{r})$. Thus, $\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^{nh}$ will be a factor of every term in the numerator and denominator summations of (12.111), and its effect will cancel. Hence, for simplicity, we can drop the constant term, which results in the modified branch metric

$$\gamma_l(s', s) = P(u_l)e^{-E_s/N_0||\mathbb{r}_l - \mathbb{v}_l||^2}. \tag{12.122}$$

Next, we express the a priori probabilities $P(u_l = \pm 1)$ as exponential terms by writing

$$\begin{aligned}P(u_l = \pm 1) &= \frac{[P(u_l = +1)/P(u_l = -1)]^{\pm 1}}{\{1 + [P(u_l = +1)/P(u_l = -1)]^{\pm 1}\}} \\ &= \frac{e^{\pm L_a(u_l)}}{\{1 + e^{\pm L_a(u_l)}\}} \\ &= \frac{e^{-L_a(u_l)/2}}{\{1 + e^{-L_a(u_l)}\}} e^{u_l L_a(u_l)/2} \\ &= A_l e^{u_l L_a(u_l)/2},\end{aligned} \tag{12.123}$$

where, since L-values do not depend on the value of their argument, the parameter A_l is independent of the actual value of u_l (see Problem 12.28). We then use (12.123) to replace $P(u_l)$ in (12.122) for $l = 0, 1, \cdots, h-1$, that is, for each information bit. For the termination bits u_l, $l = h, h+1, \cdots, h+m-1 = K-1$, however, where

$P(u_l) = 1$ and $L_a(u_l) = \pm\infty$ for each valid state transition, we simply use (12.122) directly. Thus, we can write (12.122) as

$$\begin{aligned}\gamma_l(s', s) &= A_l e^{u_l L_a(u_l)/2} e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2}, \\ &= A_l e^{u_l L_a(u_l)/2} e^{(2E_s/N_0)(\mathbf{r}_l \cdot \mathbf{v}_l) - ||\mathbf{r}_l||^2 - ||\mathbf{v}_l||^2} \\ &= A_l e^{-(||\mathbf{r}_l||^2 + n)} e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)} \\ &= A_l B_l e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = 0, 1, \cdots, h-1, \end{aligned} \tag{12.124a}$$

$$\begin{aligned}\gamma_l(s', s) &= P(u_l) e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2} \\ &= e^{-(E_s/N_0)||\mathbf{r}_l - \mathbf{v}_l||^2}, \\ &= B_l e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = h, h+1, \cdots, K-1, \end{aligned} \tag{12.124b}$$

where $B_l \equiv ||\mathbf{r}_l||^2 + n$ is a constant independent of the codeword $\mathbf{v}_l$, and $L_c = 4E_s/N_0$ is the channel reliability factor.

We now note from (12.115)–(12.119) and (12.124) that the pdf $p(s', s, \mathbf{r})$ contains the factors $\prod_{l=0}^{h-1} A_l$ and $\prod_{l=0}^{K-1} B_l$. Thus, $\prod_{l=0}^{h-1} A_l$ and $\prod_{l=0}^{K-1} B_l$ will be factors of every term in the numerator and denominator summations of (12.111), and their effect will cancel. Hence, we can drop these factors and use the exponential function

$$\gamma_l(s', s) = e^{u_l L_a(u_l)/2} e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = 0, 1, \cdots, h-1, \tag{12.125a}$$

$$\gamma_l(s', s) = e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = h, h+1, \cdots, K-1, \tag{12.125b}$$

as a simplified branch metric. Note that when the input bits are equally likely, $L_a(u_l) = 0$, and the simplified branch metric is given by

$$\gamma_l(s', s) = e^{(L_c/2)(\mathbf{r}_l \cdot \mathbf{v}_l)}, \quad l = 0, 1, \cdots, K-1. \tag{12.126}$$

Finally, we see from (12.116), (12.117), and (12.125) that the forward and backward metrics are sums of $2^k = 2$ exponential terms, one corresponding to each valid state transition in the trellis. This suggests simplifying the calculations by making use of the identity (see Problem 12.29)

$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|}) \tag{12.127}$$

to replace the (computationally more difficult) operation $\ln(e^x + e^y)$ with a max function plus a lookup table for evaluating $\ln(1 + e^{-|x-y|})$. To benefit from this simplification, we must first introduce the following log-domain metrics:

$$\gamma_l^*(s', s) \equiv \ln \gamma_l(s', s) = \begin{cases} \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2} \mathbf{r}_l \cdot \mathbf{v}_l, & l = 0, 1, \cdots, h-1, \\ \frac{L_c}{2} \mathbf{r}_l \cdot \mathbf{v}_l, & l = h, h+1, \cdots, K-1, \end{cases} \tag{12.128a}$$

$$\alpha_{l+1}^*(s) \equiv \ln\, \alpha_{l+1}(s) = \ln \sum_{s' \in \sigma_l} \gamma_l(s', s)\alpha_l(s')$$

$$= \ln \sum_{s' \in \sigma_l} e^{[\gamma_l^*(s', s) + \alpha_l^*(s')]} \tag{12.128b}$$

$$= \max{}^*_{s' \in \sigma_l}[\gamma_l^*(s', s) + \alpha_l^*(s')], \quad l = 0, 1, \cdots, K-1,$$

$$\alpha_0^*(s) \equiv \ln\, \alpha_0(s) = \begin{cases} 0, & s = \mathbb{0} \\ -\infty, & s \neq \mathbb{0}, \end{cases} \tag{12.128c}$$

$$\beta_l^*(s') \equiv \ln\, \beta_l(s') = \ln \sum_{s \in \sigma_{l+1}} \gamma_l(s', s)\beta_{l+1}(s)$$

$$= \ln \sum_{s \in \sigma_{l+1}} e^{[\gamma_l^*(s', s) + \beta_{l+1}^*(s)]} \tag{12.128d}$$

$$= \max{}^*_{s \in \sigma_{l+1}}[\gamma_l^*(s', s) + \beta_{l+1}^*(s)], \quad l = K-1, K-2, \cdots, 0,$$

$$\beta_K^*(s) \equiv \ln\, \beta_K(s) = \begin{cases} 0, & s = \mathbb{0} \\ -\infty, & s \neq \mathbb{0}. \end{cases} \tag{12.128e}$$

The use of the max* function in (12.128b) and (12.128d) follows from the fact that these recursive equations involve calculating the log of a sum of two exponential functions, one corresponding to each valid state transition. Further, we can now write the expressions for the pdf $p(s', s, \mathbb{r})$ in (12.115) and the APP L-value $L(u_l)$ in (12.111) as

$$p(s', s, \mathbb{r}) = e^{\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')} \tag{12.129}$$

and

$$L(u_l) = \ln \left\{ \sum_{(s', s) \in \Sigma_l^+} e^{\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')} \right\}$$

$$- \ln \left\{ \sum_{(s', s) \in \Sigma_l^-} e^{\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')} \right\}, \tag{12.130}$$

and we see that each of the two terms in (12.130) involves calculating the log of a sum of 2^ν exponential terms, one corresponding to each state in the trellis. Now, we note that we can apply the max* function defined in (12.127) to sums of more than two exponential terms by making use of the result (see Problem 12.29)

$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z], \tag{12.131}$$

in other words, we can evaluate a 3-variable max* function by applying the 2-variable max* function twice. Finally, using (12.131) in (12.130) allows us to express the APP L-values as

$$L(u_l) = \max{}^*_{(s', s) \in \Sigma_l^+}[\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')]$$

$$- \max{}^*_{(s', s) \in \Sigma_l^-}[\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')]. \tag{12.132}$$

The algorithm that computes the APP L-values $L(u_l)$ using (12.132), the log-domain metrics defined in (12.128), and the max* functions defined in (12.127) and (12.131) is called the *log-MAP algorithm*, or the *log-domain BCJR algorithm*. The log-MAP algorithm, because it uses just a max(•) function and a lookup table, is considerably simpler to implement and provides greater numerical stability than the MAP algorithm.

The (Log-Domain) BCJR Algorithm

Step 1. Initialize the forward and backward metrics $\alpha_0^*(s)$ and $\beta_K^*(s)$ using (12.128c) and (12.128e).

Step 2. Compute the branch metrics $\gamma_l^*(s', s)$, $l = 0, 1, \cdots, K-1$, using (12.128a).

Step 3. Compute the forward metrics $\alpha_{l+1}^*(s)$, $l = 0, 1, \cdots, K-1$, using (12.128b).

Step 4. Compute the backward metrics $\beta_l^*(s')$, $l = K-1, K-2, \cdots, 0$, using (12.128d).

Step 5. Compute the APP L-values $L(u_l)$, $l = 0, 1, \cdots, h-1$, using (12.132).

Step 6. (Optional) Compute the hard decisions $\hat{u}_l$, $l = 0, 1, \cdots, h-1$, using (12.107).

The calculations in each of steps 3, 4, and 5 involve essentially the same amount of computation as the Viterbi algorithm, and thus the BCJR algorithm is roughly three times as complex as the Viterbi algorithm. Also, we see from (12.128a) that the BCJR algorithm requires knowledge of the channel SNR E_s/N_0 ($L_c = 4E_s/N_0$) to calculate its branch metrics. The branch metrics for the Viterbi algorithm, on the other hand, are just the correlations $\mathbf{r}_l \cdot \mathbf{v}_l$, and no knowledge of the channel SNR is needed. (Note that the constant $C_1 (= L_c/2)$ can be dropped from the Viterbi algorithm metric in (12.14) because it does not affect the decoding result, but the constant $L_c/2$ cannot be dropped from the BCJR algorithm metric in (12.128a), because its relation to the a priori factor $u_l L_a(u_l)/2$ does affect the decoding result.) When the BCJR algorithm is used for iterative decoding (see Chapter 16), the computation of the hard decisions in step 6 takes place only at the final iteration. At earlier iterations, the APP L-values from step 5 are the (soft) decoder outputs. The basic operations of the log-domain BCJR algorithm are illustrated in Figure 12.19.

An even simpler algorithm results if we ignore the *correction* term $\ln(1 + e^{-|x-y|})$ in the max* function and simply use the approximation

$$\max^*(x, y) \approx \max(x, y) \tag{12.133}$$

instead. Because the correction term is bounded by

$$0 < \ln(1 + e^{-|x-y|}) \leq \ln(2) = 0.693, \tag{12.134}$$

the approximation is reasonably good whenever $|\max(x, y)| \geq 7$. The algorithm that computes the APP L-values $L(u_l)$ using the function max instead of max* is called the *Max-log-MAP algorithm*. Because the max function plays the same role as the compare-and-select operations in the Viterbi algorithm, the forward recursion in the Max-log-MAP algorithm is equivalent to a forward Viterbi algorithm, and

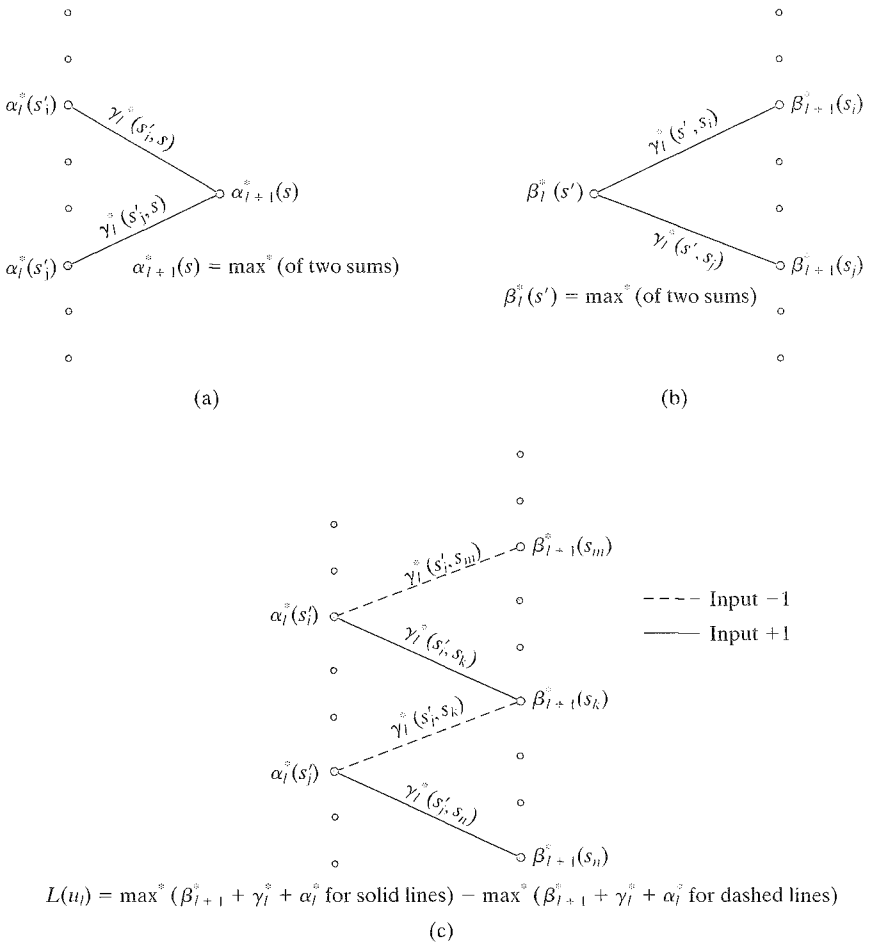


FIGURE 12.19: Trellis diagrams representing (a) the forward recursion of (12.128b), (b) the backward recursion of (12.128d), and (c) the APP L-value computation of (12.132) (adapted from [26]).

the backward recursion in the Max-log-MAP algorithm is equivalent to a backward Viterbi algorithm. The difference between the BCJR and the Viterbi algorithms, then, is in the correction term $\ln(1 + e^{-|x-y|})$, which accounts for the optimality of the BCJR algorithm with respect to BER. The Viterbi algorithm, which is optimized for WER, typically results in a slight BER performance degradation compared with the BCJR algorithm.

All three algorithms presented in this section: MAP, log-MAP, and Max-log-MAP, are examples of *forward–backward algorithms*, since they involve both a forward and a backward recursion. They are also examples of SISO decoders, since we can take the (real) APP L-values $L(u_l)$ as *soft* decoder outputs instead of applying (12.107) to obtain *hard* decoder outputs.

We now give an example of the operation of the log-domain BCJR algorithm.

EXAMPLE 12.8 BCJR Decoding of a (2, 1, 1) Systematic Recursive Convolutional Code on an AWGN Channel

Consider the (2, 1, 1) SRCC with generator matrix

$$\mathbb{G}(D) = [1 \;\; 1/(1+D)] \tag{12.135}$$

whose encoder diagram is shown in Figure 12.20(a). The 2-state trellis diagram, with branches labeled according to the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$, corresponding to an input sequence of length 4, is shown in Figure 12.20(b).[6] Let $\mathbb{u} = (u_0, u_1, u_2, u_3)$ denote the input vector of length $K = h + m = 4$, and $\mathbb{v} = (\mathbb{v}_0, \mathbb{v}_1, \mathbb{v}_2, \mathbb{v}_3)$ denote the codeword of length $N = nK = 8$. We assume a channel SNR of $E_s/N_0 = 1/4$ (−6.02 dB) and a (normalized by $\sqrt{E_s}$) received vector

$$\begin{aligned}\mathbb{r} &= (\mathbb{r}_0, \mathbb{r}_1, \mathbb{r}_2, \mathbb{r}_3) = (r_0^{(0)}, r_0^{(1)}; r_1^{(0)}, r_1^{(1)}; r_2^{(0)}, r_2^{(1)}; r_3^{(0)}, r_3^{(1)}) \\ &= (+0.8, +0.1; +1.0, -0.5; -1.8, +1.1; +1.6, -1.6).\end{aligned} \tag{12.136}$$

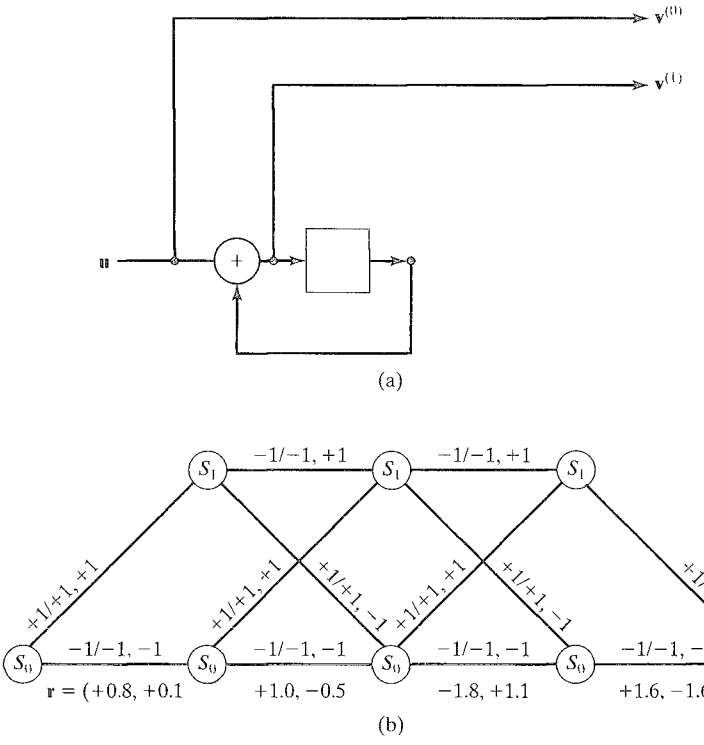


FIGURE 12.20: (a) A (2, 1, 1) systematic feedback encoder and (b) the decoding trellis for the (2, 1, 1) SRCC with $K = 4$ and $N = 8$.

[6] In Figure 12.20(b), the branch labels $u_l/v_l^{(0)}v_l^{(1)}$ indicate both the input u_l and the outputs $v_l^{(0)}$ and $v_l^{(1)}$. This is because, for feedback encoder trellises, the upper branch leaving each state does not necessarily correspond to a 1 (+1) input, and the lower branch to a 0 (−1) input. Also note, since the encoder is systematic, that the first output label on each branch equals the input label.

(Note that, since the rate of the terminated code is $R = h/N = 3/8$, an $E_s/N_0 = 1/4$ (-6.02 dB) corresponds to an $E_b/N_0 = E_s/RN_0 = 2/3$ (-1.76 dB).) The received vector $\mathbb{r}$ is also shown in Figure 12.20(b).

Assuming that the a priori probabilities of the information bits are equally likely, $L_a(u_l) = 0$, $l = 0, 1, 2$, and we compute the log-domain branch metrics using (12.128a) (note that $L_c = 4E_s/N_0 = 1$) as follows:

$$\gamma_0^*(S_0, S_0) = \frac{-1}{2}L_a(u_0) + \frac{1}{2}\mathbb{r}_0 \cdot \mathbb{v}_0$$

$$= \frac{1}{2}(-0.8 - 0.1) = -0.45$$

$$\gamma_0^*(S_0, S_1) = \frac{+1}{2}L_a(u_0) + \frac{1}{2}\mathbb{r}_0 \cdot \mathbb{v}_0$$

$$= \frac{1}{2}(0.8 + 0.1) = 0.45$$

$$\gamma_1^*(S_0, S_0) = \frac{-1}{2}L_a(u_1) + \frac{1}{2}\mathbb{r}_1 \cdot \mathbb{v}_1$$

$$= \frac{1}{2}(-1.0 + 0.5) = -0.25$$

$$\gamma_1^*(S_0, S_1) = \frac{+1}{2}L_a(u_1) + \frac{1}{2}\mathbb{r}_1 \cdot \mathbb{v}_1$$

$$= \frac{1}{2}(1.0 - 0.5) = 0.25$$

$$\gamma_1^*(S_1, S_1) = \frac{-1}{2}L_a(u_1) + \frac{1}{2}\mathbb{r}_1 \cdot \mathbb{v}_1$$

$$= \frac{1}{2}(-1.0 - 0.5) = -0.75$$

$$\gamma_1^*(S_1, S_0) = \frac{+1}{2}L_a(u_1) + \frac{1}{2}\mathbb{r}_1 \cdot \mathbb{v}_1$$

$$= \frac{1}{2}(1.0 + 0.5) = 0.75$$

$$\gamma_2^*(S_0, S_0) = \frac{-1}{2}L_a(u_2) + \frac{1}{2}\mathbb{r}_2 \cdot \mathbb{v}_2$$

$$= \frac{1}{2}(1.8 - 1.1) = 0.35$$

$$\gamma_2^*(S_0, S_1) = \frac{+1}{2}L_a(u_2) + \frac{1}{2}\mathbb{r}_2 \cdot \mathbb{v}_2$$

$$= \frac{1}{2}(-1.8 + 1.1) = -0.35$$

$$\gamma_2^*(S_1, S_1) = \frac{-1}{2}L_a(u_2) + \frac{1}{2}\mathbb{r}_2 \cdot \mathbb{v}_2$$

$$
\begin{aligned}
&= \frac{1}{2}(1.8 + 1.1) = 1.45 \\
\gamma_2^*(S_1, S_0) &= \frac{+1}{2} L_a(u_2) + \frac{1}{2}\mathbf{r}_2 \cdot \mathbf{v}_2 \\
&= \frac{1}{2}(-1.8 - 1.1) = -1.45 \\
\gamma_3^*(S_0, S_0) &= \frac{+1}{2}\mathbf{r}_3 \cdot \mathbf{v}_3 \\
&= \frac{1}{2}(-1.6 + 1.6) = 0 \\
\gamma_3^*(S_1, S_0) &= \frac{+1}{2}\mathbf{r}_3 \cdot \mathbf{v}_3 \\
&= \frac{1}{2}(1.6 + 1.6) = 1.60.
\end{aligned} \tag{12.137}
$$

We then compute the log-domain forward metrics using (12.128b) as follows:

$$
\begin{aligned}
\alpha_1^*(S_0) &= [\gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] = -0.45 + 0 = -0.45 \\
\alpha_1^*(S_1) &= [\gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] = 0.45 + 0 = 0.45 \\
\alpha_2^*(S_0) &= \max^*\{[\gamma_1^*(S_0, S_0) + \alpha_1^*(S_0)], [\gamma_1^*(S_1, S_0) + \alpha_1^*(S_1)]\} \\
&= \max^*\{[(-0.25) + (-0.45)], [(0.75) + (0.45)]\} \\
&= \max^*(-0.70, +1.20) = 1.20 + \ln(1 + e^{-|-1.9|}) = 1.34 \\
\alpha_2^*(S_1) &= \max^*\{[\gamma_1^*(S_0, S_1) + \alpha_1^*(S_0)], [\gamma_1^*(S_1, S_1) + \alpha_1^*(S_1)]\} \\
&= \max^*(-0.20, -0.30) = -0.20 + \ln(1 + e^{-|0.1|}) = 0.44.
\end{aligned} \tag{12.138}
$$

Similarly, we compute the log-domain backward metrics using (12.128d) as follows:

$$
\begin{aligned}
\beta_3^*(S_0) &= [\gamma_3^*(S_0, S_0) + \beta_4^*(S_0)] = 0 + 0 = 0 \\
\beta_3^*(S_1) &= [\gamma_3^*(S_1, S_0) + \beta_4^*(S_0)] = 1.60 + 0 = 1.60 \\
\beta_2^*(S_0) &= \max^*\{[\gamma_2^*(S_0, S_0) + \beta_3^*(S_0)], [\gamma_2^*(S_0, S_1) + \beta_3^*(S_1)]\} \\
&= \max^*\{[(0.35) + (0)], [(-0.35) + (1.60)]\} \\
&= \max^*(0.35, 1.25) = 1.25 + \ln(1 + e^{-|-0.90|}) = 1.59 \\
\beta_2^*(S_1) &= \max^*\{[\gamma_2^*(S_1, S_0) + \beta_3^*(S_0)], [\gamma_2^*(S_1, S_1) + \beta_3^*(S_1)]\} \\
&= \max^*(-1.45, 3.05) = 3.05 + \ln(1 + e^{-|-4.5|}) = 3.06 \\
\beta_1^*(S_0) &= \max^*\{[\gamma_1^*(S_0, S_0) + \beta_2^*(S_0)], [\gamma_1^*(S_0, S_1) + \beta_2^*(S_1)]\} \\
&= \max^*(1.34, 3.31) = 3.44 \\
\beta_1^*(S_1) &= \max^*\{[\gamma_1^*(S_1, S_0) + \beta_2^*(S_0)], [\gamma_1^*(S_1, S_1) + \beta_2^*(S_1)]\} \\
&= \max^*(2.34, 2.31) = 3.02.
\end{aligned} \tag{12.139}
$$

Finally, we compute the APP L-values for the three information bits using (12.132) as follows:

$$\begin{aligned}
L(u_0) &= [\beta_1^*(S_1) + \gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] - [\beta_1^*(S_0) + \gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] \\
&= (3.47) - (2.99) = +0.48 \\
L(u_1) &= \max^*\{[\beta_2^*(S_0) + \gamma_1^*(S_1, S_0) + \alpha_1^*(S_1)], [\beta_2^*(S_1) + \gamma_1^*(S_0, S_1) + \alpha_1^*(S_0)]\} \\
&\quad - \max^*\{[\beta_2^*(S_0) + \gamma_1^*(S_0, S_0) + \alpha_1^*(S_0)], [\beta_2^*(S_1) + \gamma_1^*(S_1, S_1) + \alpha_1^*(S_1)]\} \\
&= \max^*[(2.79), (2.86)] - \max^*[(0.89), (2.76)] \\
&= (3.52) - (2.90) = +0.62 \\
L(u_2) &= \max^*\{[\beta_3^*(S_0) + \gamma_2^*(S_1, S_0) + \alpha_2^*(S_1)], [\beta_3^*(S_1) + \gamma_2^*(S_0, S_1) + \alpha_2^*(S_0)]\} \\
&\quad - \max^*\{[\beta_3^*(S_0) + \gamma_2^*(S_0, S_0) + \alpha_2^*(S_0)], [\beta_3^*(S_1) + \gamma_2^*(S_1, S_1) + \alpha_2^*(S_1)]\} \\
&= \max^*[(-1.01), (2.59)] - \max^*[(1.69), (3.49)] \\
&= (2.62) - (3.64) = -1.02.
\end{aligned} \tag{12.140}$$

Using (12.107), we obtain the hard-decision outputs of the BCJR decoder for the three information bits:

$$\hat{\mathbf{u}} = (+1, +1, -1). \tag{12.141}$$

Even though termination bit u_3 is not an information bit, it is also possible to use the same procedure to find its APP L-value. As will be seen in Chapter 16, this is necessary in iterative decoding, where "soft-output" APP L-values are passed as a priori inputs to a second decoder.

In the preceding example, the decoder is operating at an SNR of $E_b/N_0 = -1.76$ dB, well below the capacity of $E_b/N_0 = -0.33$ dB for rate $R = 3/8$ codes (see Table 1.2). Although this is not normally the case for BCJR decoding of a convolutional code, it is not at all unusual in iterative decoding of turbo codes, where decoders typically operate at SNRs below capacity (see Chapter 16).

We now revisit Example 12.8 to see the effect of applying the Max-log-MAP algorithm.

EXAMPLE 12.8 (Continued)

In the Max-log-MAP algorithm, the branch metrics remain as computed in (12.137). The approximation $\max^*(x, y) \approx \max(x, y)$ of (12.133) affects the computation of the forward and backward metrics (see (12.138) and (12.139), respectively), as follows:

$$\alpha_2^*(S_0) = \max(-0.70, +1.20) = 1.20$$

$$\alpha_2^*(S_1) = \max(-0.20, -0.30) = -0.20$$

$$\beta_2^*(S_0) = \max(0.35, 1.25) = 1.25$$

$$\beta_2^*(S_1) = \max(-1.45, 3.05) = 3.05$$

$$\beta_1^*(S_0) = \max(1.34, 3.31) = 3.31$$
$$\beta_1^*(S_1) = \max(2.34, 2.31) = 2.34. \tag{12.142}$$

Then, we compute the APP L-values for the three information bits using (12.132), (12.133), (12.137), and (12.142) as follows:

$$\begin{aligned}
L(u_0) &= [\beta_1^*(S_1) + \gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] - [\beta_1^*(S_0) + \gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] \\
&= (2.79) - (2.86) = -0.07 \\
L(u_1) &= \max\{[\beta_2^*(S_0) + \gamma_1^*(S_1, S_0) + \alpha_1^*(S_1)], [\beta_2^*(S_1) + \gamma_1^*(S_0, S_1) + \alpha_1^*(S_0)]\} \\
&\quad - \max\{[\beta_2^*(S_0) + \gamma_1^*(S_0, S_0) + \alpha_1^*(S_0)], [\beta_2^*(S_1) + \gamma_1^*(S_1, S_1) + \alpha_1^*(S_1)]\} \\
&= \max[(2.79), (2.86)] - \max[(0.89), (2.76)] \\
&= (2.86) - (2.76) = +0.1000 \\
L(u_2) &= \max\{[\beta_3^*(S_0) + \gamma_2^*(S_1, S_0) + \alpha_2^*(S_1)], [\beta_3^*(S_1) + \gamma_2^*(S_0, S_1) + \alpha_2^*(S_0)]\} \\
&\quad - \max\{[\beta_3^*(S_0) + \gamma_2^*(S_0, S_0) + \alpha_2^*(S_0)], [\beta_3^*(S_1) + \gamma_2^*(S_1, S_1) + \alpha_2^*(S_1)]\} \\
&= \max[(-1.65), (2.45)] - \max[(1.55), (2.85)] \\
&= (2.45) - (2.85) = -0.4000.
\end{aligned} \tag{12.143}$$

Using (12.107), we obtain the hard-decision outputs of the Max-log-MAP decoder:

$$\hat{\mathbf{u}} = (-1, +1, -1), \tag{12.144}$$

and we see that the Max-log-MAP algorithm does not give the same result as the log-MAP algorithm in this case.

The log-MAP and Max-log-MAP algorithms do not give the same decoding result because the decoders are operating at an SNR well below capacity. Typically, when these algorithms are used to decode convolutional codes at SNRs above capacity, they will give the same result, although the performance of the Max-log-MAP algorithm is somewhat degraded at very low SNRs. Also, as will be discussed in Chapter 16, the performance loss of the Max-log-MAP algorithm is more pronounced when it is used for iterative decoding, since the approximation error accumulates as additional iterations are performed.

We now give an example illustrating the application of the BCJR algorithm to decoding on a DMC. In this case it is convenient to compute the APP L-values $L(u_l)$ using the probability-domain version of the algorithm given by expressions (12.111) and (12.115)–(12.120), where in (12.120) we compute the branch metrics $\gamma_l(s', s)$ by substituting the (discrete) channel transition probabilities

$$P(\mathbf{r}_l|\mathbf{v}_l) = \prod_{j=0}^{n-1} P(r_l^{(j)}|v_l^{(j)}), \quad l = 0, 1, \cdots, K-1 \tag{12.145}$$

for the (continuous) channel transition pdf's $p(\mathbf{r}_l|\mathbf{v}_l)$. We then use the branch metrics in (12.115) to calculate the (discrete) joint probabilities $P(s', s, \mathbf{r})$, which we

in turn use to calculate the APP L-values using (12.111). Also, a straightforward computation of the forward and backward metrics $\alpha_l(s)$ and $\beta_l(s')$ typically results in exceedingly small values, which can lead to numerical precision problems. Hence, we introduce normalization constants a_l and b_l and normalized forward and backward metrics $A_l(s)$ and $B_l(s')$, respectively, which are defined as follows:

$$\sum_{s\in\sigma_l} \alpha_l(s) \equiv a_l, \quad l = 1, 2, \cdots, K, \tag{12.146a}$$

$$A_l(s) \equiv \alpha_l(s)/a_l, \quad l = 1, 2, \cdots, K, \quad \text{all } s \in \sigma_l, \tag{12.146b}$$

$$\sum_{s'\in\sigma_l} \beta_l(s') \equiv b_l, \quad l = K-1, K-2, \cdots, 0, \tag{12.146c}$$

$$B_l(s') \equiv \beta_l(s')/b_l, \quad l = K-1, K-2, \cdots, 0, \quad \text{all } s' \in \sigma_l. \tag{12.146d}$$

From (12.146) we can see that the normalized forward and backward metrics sum to 1 at each time unit l; that is

$$\sum_{s\in\sigma_l} A_l(s) = 1, \quad l = 1, 2, \cdots, K, \tag{12.147a}$$

$$\sum_{s'\in\sigma_l} B_l(s') = 1, \quad l = K-1, K-2, \cdots, 0. \tag{12.147b}$$

Using the normalized forward and backward metrics $A_l(s)$ and $B_l(s')$, computed using (12.116), (12.117), and (12.146), instead of $\alpha_l(s)$ and $\beta_l(s')$, respectively, to evaluate the joint probabilities in (12.115) avoids numerical precision problems and has no effect on the calculation of the final APP L-values in (12.111) (see Problem 12.32).

EXAMPLE 12.9 BCJR Decoding of a (2, 1, 2) Nonsystematic Convolutional Code on a DMC

Assume a binary-input, 8-ary output DMC with transition probabilities $P(r_l^{(j)}|v_l^{(j)})$ given by the following table:

$v_l^{(j)}$ \ $r_l^{(j)}$	$\mathbb{0}_1$	$\mathbb{0}_2$	$\mathbb{0}_3$	$\mathbb{0}_4$	$\mathbb{1}_4$	$\mathbb{1}_3$	$\mathbb{1}_2$	$\mathbb{1}_1$
0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Now, consider the (2, 1, 2) nonsystematic convolutional code with generator matrix

$$\mathbb{G}(D) = [1 + D + D^2 \;\; 1 + D^2] \tag{12.148}$$

whose encoder diagram is shown in Figure 12.21(a). The 4-state trellis diagram corresponding to an input sequence of length 6 is shown in Figure 12.21(b). Let $\mathbb{u} = (u_0, u_1, u_2, u_3, u_4, u_5)$ denote the input vector of length $K = h + m = 6$,

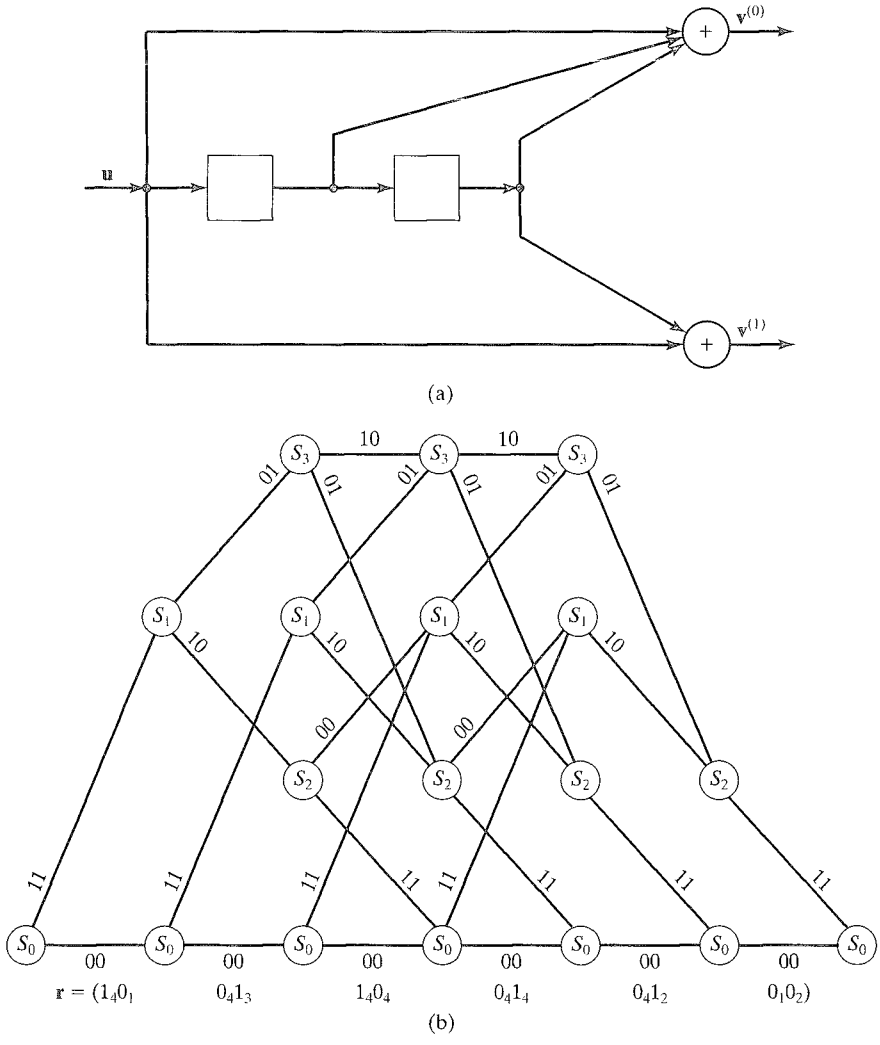


FIGURE 12.21: (a) A (2, 1, 2) nonsystematic feedforward encoder and (b) the decoding trellis for the (2, 1, 2) code with $K = 6$ and $N = 12$.

and $\mathbf{v} = (v_0, v_1, v_2, v_3, v_4, v_5)$ denote the codeword of length $N = nK = 12$. We assume that

$$P(u_l = 0) = \begin{cases} 2/3, & l = 0, 1, 2, 3 \quad \text{(information bits)} \\ 1, & l = 4, 5 \quad \text{(termination bits)}, \end{cases} \tag{12.149}$$

that is, the information bits are not equally likely, and that the received vector is given by

$$\mathbf{r} = (1_4 0_1, 0_4 1_3, 1_4 0_4, 0_4 1_4, 0_4 1_2, 0_1 0_2). \tag{12.150}$$

The received vector $\mathbb{r}$ is also shown in Figure 12.21(b). (Note that, since the encoder is feedforward, the upper branch leaving each state corresponds to a 1 input and the lower branch to a 0 input, and the termination bits must be 0's.)

We begin the algorithm by computing the (probability-domain) branch metrics using (12.120) as follows:

$$\begin{aligned}\gamma_0(S_0, S_0) &= P(u_0 = 0)P(1_4 0_1|00) = (2/3)P(1_4|0)P(0_1|0)\\ &= (2/3)(0.058)(.434) = 0.01678,\\ \gamma_0(S_0, S_1) &= P(u_0 = 1)P(1_4 0_1|11) = (1/3)P(1_4|1)P(0_1|1)\\ &= (1/3)(0.111)(.002) = 0.000074,\end{aligned} \tag{12.151}$$

and so on. The complete set of branch metrics for the trellis is shown in Figure 12.22(a).

Next, we can compute the (probability-domain) normalized forward and backward metrics using (12.116)–(12.119), (12.146), and (12.147) as follows:

$$\begin{aligned}\alpha_1(S_0) &= \gamma_0(S_0, S_0)\alpha_0(S_0) = (0.01678)(1) = 0.01678,\\ \alpha_1(S_1) &= \gamma_0(S_0, S_1)\alpha_0(S_0) = (0.000074)(1) = 0.000074,\\ a_1 &= \alpha_1(S_0) + \alpha_1(S_1) = 0.01678 + 0.000074 = 0.016854,\\ A_1(S_0) &= \alpha_1(S_0)/a_1 = (0.01678)/(0.016854) = 0.9956,\\ A_1(S_1) &= 1 - A_1(S_0) = 1 - 0.9956 = 0.0044,\\ \beta_5(S_0) &= \gamma_5(S_0, S_0)\beta_6(S_0) = (0.0855)(1) = 0.0855,\\ \beta_5(S_2) &= \gamma_5(S_2, S_0)\beta_6(S_0) = (0.000016)(1) = 0.000016,\\ b_5 &= \beta_5(S_0) + \beta_5(S_2) = 0.0855 + 0.000016 = 0.085516,\\ B_5(S_0) &= \beta_5(S_0)/b_5 = (0.0855)/(0.085516) = 0.9998,\\ B_5(S_2) &= 1 - B_5(S_0) = 1 - 0.9998 = 0.0002,\end{aligned} \tag{12.152}$$

and so on. The complete set of normalized forward and backward metrics for the trellis is shown in Figures 12.22(b) and (12.22c), respectively.

Finally, using the normalized forward and backward metrics in (12.115) to compute the joint probabilities $P(s', s, \mathbb{r})$, which we then use to calculate the APP L-values in (12.111), and recalling the mapping $0 \to -1$ and $1 \to +1$, we obtain

$$\begin{aligned}L(u_0) &= \ln\left\{\frac{P(s_0 = S_0, s_1 = S_1, \mathbb{r})}{P(s_0 = S_0, s_1 = S_0, \mathbb{r})}\right\}\\ &= \ln\left\{\frac{B_1(S_1)\gamma_0(S_0, S_1)A_0(S_0)}{B_1(S_0)\gamma_0(S_0, S_0)A_0(S_0)}\right\}\\ &= \ln\left\{\frac{(0.8162)(0.000074)(1)}{(0.1838)(0.01678)(1)}\right\} = -3.933,\end{aligned} \tag{12.153a}$$

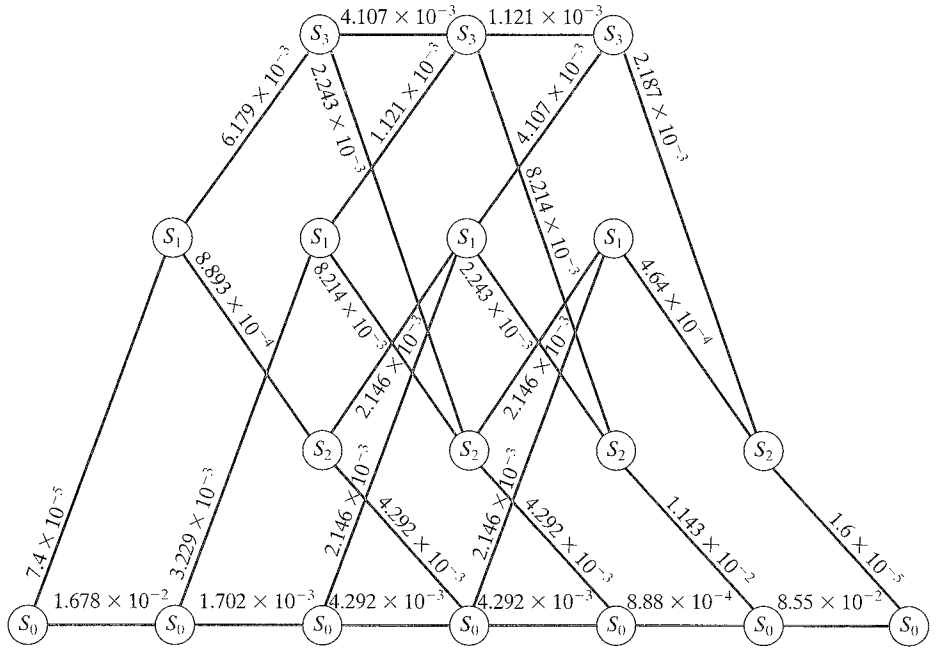
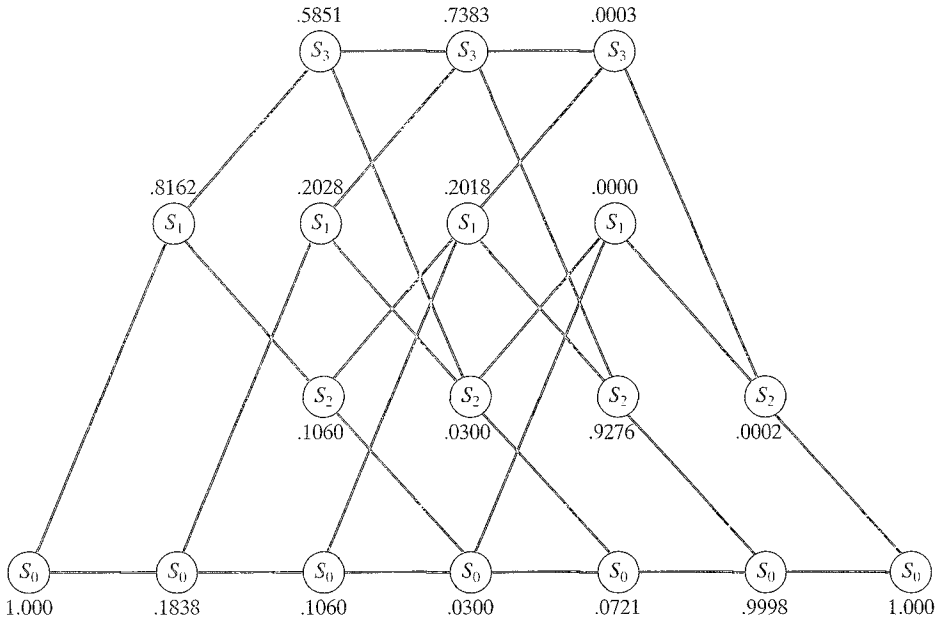


FIGURE 12.22: (a) The branch metric values $\gamma_l(s', s)$, (b) the normalized forward metric values $A_l(s)$, and (c) the normalized backward metric values $B_l(s')$ for Example 12.9.



(c)

FIGURE 12.22: (*continued*)

$$
\begin{aligned}
L(u_1) &= \ln \frac{P(s_1 = S_0, s_2 = S_1, \mathbb{r}) + P(s_1 = S_1, s_2 = S_3, \mathbb{r})}{P(s_1 = S_0, s_2 = S_0, \mathbb{r}) + P(s_1 = S_1, s_2 = S_2, \mathbb{r})} \\
&= \ln \frac{B_2(S_1)\gamma_1(S_0, S_1)A_1(S_0) + B_2(S_3)\gamma_1(S_1, S_3)A_1(S_1)}{B_2(S_0)\gamma_1(S_0, S_0)A_1(S_0) + B_2(S_2)\gamma_1(S_1, S_2)A_1(S_1)} \qquad (12.153b) \\
&= \ln \frac{(0.2028)(0.003229)(0.9956) + (0.5851)(0.006179)(0.0044)}{(0.1060)(0.001702)(0.9956) + (0.1060)(0.0008893)(0.0044)} \\
&= +1.311,
\end{aligned}
$$

and, following the same procedure (see Problem 12.33), we have

$$L(u_2) = +1.234, \qquad (12.153c)$$

and

$$L(u_3) = -8.817. \qquad (12.153d)$$

Using (12.107), and again recalling the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$, we see that the hard-decision outputs of the BCJR decoder for the four information bits are given by

$$\hat{\mathbb{u}} = (\hat{u}_0, \hat{u}_1, \hat{u}_2, \hat{u}_3) = (0, 1, 1, 0). \qquad (12.154)$$

In this case, since the encoder is feedforward and the termination bits are known to be 0's, it is not necessary to find the APP L-values of u_4 and u_5, even as part of an iterative decoding procedure.

Although it is convenient to use the probability-domain BCJR algorithm for DMCs, the log-domain algorithm can also be used. Similarly, the probability-domain algorithm can be used for AWGN channels.

12.7 PUNCTURED AND TAIL-BITING CONVOLUTIONAL CODES

The trellis-based decoding procedures for convolutional codes described previously in this chapter are well suited for medium-to-low encoder rates $R = k/n$ and moderate-to-large information sequence block lengths $K^* = hk$. For high-rate codes and short block lengths, however, modified encoding and decoding procedures are sometimes useful. In particular, since the branch complexity of the decoding trellis increases exponentially with k for a standard convolutional code, a modified form of high-rate code, called a *punctured convolutional code*, which has only two branches leaving each state, was introduced in a paper by Cain, Clark, and Geist [27]. Also, since the fractional rate loss of a terminated convolutional code is large when h is short, a modified form of terminated code, called a *tail-biting convolutional code*, which has no termination bits and therefore no rate loss, was introduced in papers by Solomon and van Tilborg [28] and Ma and Wolf [29].

A rate $R = (n-1)/n$ punctured convolutional code is obtained by periodically deleting or puncturing certain bits from the codewords of a rate $R = 1/2$ *mother code*, where $n \geq 3$. These codes can then be decoded using the Viterbi algorithm (or the BCJR algorithm) with roughly the same decoding complexity as the rate $R = 1/2$ mother code. This is because, in the rate $R = (n-1)/n$ punctured code, only two branches enter each state, and thus only one binary comparison is performed at each state, rather than the $2^{(n-1)} - 1$ binary comparisons required to decode a standard rate $R = (n-1)/n$ code. We illustrate the technique with an example.

EXAMPLE 12.10 A Punctured Convolutional Code

Consider the 4-state, rate $R = 1/2$ mother code generated by the $(2, 1, 2)$ nonsystematic feedforward convolutional encoder with generator matrix

$$\mathbb{G}(D) = [1 + D^2 \quad 1 + D + D^2]. \tag{12.155}$$

This is the optimum free distance code from Table 12.1(c) with $d_{free} = 5$. Now, consider forming a rate $R = 2/3$ punctured code by deleting the first bit on every other branch of the trellis. Six sections of the trellis diagram for this punctured code, with the deleted bits indicated by an x, are shown in Figure 12.23(a). In the punctured code, only one symbol is transmitted on those trellis branches where a bit has been deleted. Thus, the trellis diagram for the punctured code is time-varying, with a period of two branches.

A Viterbi or BCJR decoder for this rate $R = 2/3$ punctured code would operate exactly like the decoder for the mother code, except that no metrics would be computed for the deleted symbols. Thus, the metric computation would involve two symbols on half the branches, and only one symbol on the other half. In each two-branch section of the decoding trellis for the punctured code, a total of two binary comparisons per state are required to decode two information bits. By contrast, in each section of the decoding trellis for a standard rate $R = 2/3$ code, a total of three binary comparisons per state are required to decode two information bits. Hence,

decoding the rate $R = 2/3$ punctured code, based on the simpler structure of the rate $R = 1/2$ mother code, is less complex than decoding a standard rate $R = 2/3$ code.

A rate $R = 3/4$ punctured code can be obtained from the same rate $R = 1/2$ mother code by deleting two bits from each set of three branches, as shown in

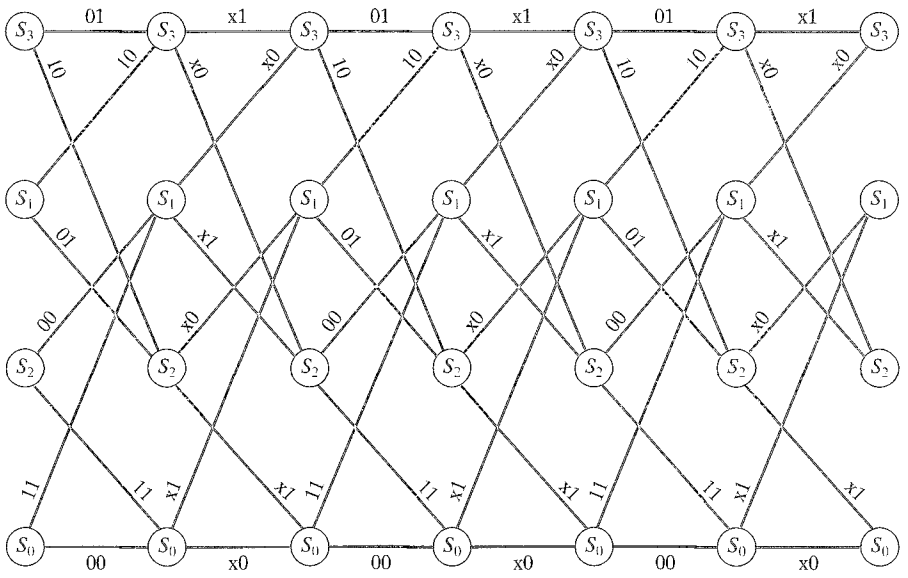


FIGURE 12.23: Trellis diagrams for (a) a rate $R = 2/3$ punctured convolutional code and (b) a rate $R = 3/4$ punctured convolutional code.

Figure 12.23(b). In this case, in each three-branch section of the trellis for the punctured code, a total of three binary comparisons per state are required to decode three information bits. In contrast, in each section of the trellis for a standard rate $R = 3/4$ code, a total of seven binary comparisons per state are required to decode three information bits. Again, decoding the rate $R = 3/4$ punctured code, based on the simpler structure of the rate $R = 1/2$ mother code, is less complex than decoding a standard rate $R = 3/4$ code.

By carefully examining the trellises in Figure 12.23, we see that the minimum free distance of each punctured code is $d_{free} = 3$. In other words, puncturing a rate $R = 1/2$ mother code to a higher rate, as expected, reduces the free distance. These free distances, however, are equal to the free distances of the best standard 4-state rate $R = 2/3$ and $3/4$ codes listed in Tables 12.1(d) & (e), respectively. In other words, no penalty in free distance is paid for the reduced decoding complexity of the punctured codes in these cases. In general, however, this is not so, since rate $R = (n-1)/n$ codes obtained by puncturing a rate $R = 1/2$ mother code form just a subclass of all possible rate $R = (n-1)/n$ convolutional codes.

In Table 12.4 (a) and (b), we list the best rate $R = 2/3$ and $3/4$ codes, respectively, that can be obtained by puncturing a rate $R = 1/2$ mother code. In each case, the puncturing pattern is indicated using a $2 \times T$ binary matrix $\mathbb{P}$, where T is the puncturing period, the first row of $\mathbb{P}$ indicates the bits to be deleted from the first encoded sequence, and the second row of $\mathbb{P}$ indicates the bits to be deleted from the second encoded sequence. (In the matrix $\mathbb{P}$, a 0 indicates a bit to be deleted, and a 1 indicates a bit to be transmitted.) For example, in Figure 12.23, the puncturing patterns are given by

$$\mathbb{P} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \text{(Figure 12.23(a))} \tag{12.156a}$$

and

$$\mathbb{P} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}. \quad \text{(Figure 12.23(b))} \tag{12.156b}$$

In applications where it is necessary to support two or more different code rates, it is sometimes convenient to make use of *rate-compatible punctured convolutional* (RCPC) *codes* [31]. An RCPC code is a set of two or more convolutional codes punctured from the same mother code in such a way that the codewords of a higher-rate code can be obtained from the codewords of a lower-rate code simply by deleting additional bits. In other words, the set of puncturing patterns must be such that the $\mathbb{P}$ matrix of a higher-rate code is obtained from the $\mathbb{P}$ matrix of a lower-rate code by simply changing some of the 1's to 0's. (This property implies using puncturing matrices with the same period T for all code rates and does not apply to the two puncturing matrices in (12.156).) An RCPC code then has the property that all the codes in the set have the same encoder and decoder. The higher-rate codes simply have additional bits deleted. This property is particularly convenient in two-way communication systems involving retransmission requests, where the initial transmission uses a high-rate punctured code and then, if the transmission is unsuccessful, punctured bits are sent during later transmissions, resulting in a more

powerful lower-rate code for decoding. (See Chapter 22 for more details on two-way communication systems.)

TABLE 12.4: Optimum (a) rate $R = 2/3$ and (b) rate $R = 3/4$ convolutional codes obtained by puncturing a rate $R = 1/2$ mother code. Adapted from [30].

Mother Code			Punctured Code		
ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbb{P}$	d_{free}	$A_{d_{free}}$
2	5	7	$\begin{matrix}1 & 0\\ 1 & 1\end{matrix}$	3	1
3	13	17	$\begin{matrix}1 & 1\\ 0 & 1\end{matrix}$	4	3
4	31	27	$\begin{matrix}1 & 1\\ 0 & 1\end{matrix}$	4	1
5	65	57	$\begin{matrix}1 & 0\\ 1 & 1\end{matrix}$	6	19
6	155	117	$\begin{matrix}1 & 1\\ 1 & 0\end{matrix}$	6	1

(a)

Mother Code			Punctured Code		
ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbb{P}$	d_{free}	$A_{d_{free}}$
2	5	7	$\begin{matrix}1 & 0 & 1\\ 1 & 1 & 0\end{matrix}$	3	6
3	13	17	$\begin{matrix}1 & 1 & 0\\ 1 & 0 & 1\end{matrix}$	4	29
4	31	27	$\begin{matrix}1 & 0 & 1\\ 1 & 1 & 0\end{matrix}$	3	1
5	65	57	$\begin{matrix}1 & 0 & 0\\ 1 & 1 & 1\end{matrix}$	4	1
6	155	117	$\begin{matrix}1 & 1 & 0\\ 1 & 0 & 1\end{matrix}$	5	8

(b)

A tail-biting convolutional code is obtained by terminating the encoder output sequence after the last information block in the input sequence. In other words, no "tail" of input blocks is used to force the encoder back to the all-zero state. In this case, the input sequence has length $K^* = hk$, the output sequence has length $N^* = hn$, and the rate of the resulting tail-biting convolutional code is

$$R_{tb} = \frac{K^*}{N^*} = \frac{hk}{hn} = \frac{k}{n}. \tag{12.157}$$

We see from (12.157) that the rate R_{tb} of the tail-biting code is the same as the rate R of its convolutional encoder; that is, there is no rate loss associated with this trellis termination technique.

It is not difficult to show that with any of the standard trellis-based decoding techniques, the lack of a tail causes the information bits near the end of the trellis to receive less protection against the effects of channel noise than the bits earlier in the trellis. (In fact, the reason for having a tail is to provide the same level of protection for all the information bits.) In other words, the bits near the end of the trellis will have a higher decoded error probability than the bits near the beginning of the trellis, leading to an *unequal error protection* property. Although this may actually be desirable in some applications, in most cases we want a uniform level of error protection throughout the entire information sequence.

To provide a uniform level of error protection for the entire information sequence and to maintain the zero-rate-loss property, we can modify the tail-biting technique to require that valid codewords start and end in the same state. Because codewords without a tail can end in any state, they must also be allowed to start in any state. In other words, rather than requiring the encoder to start in the all-zero state S_0, we require the encoder to start in the same state as its ending state, which can be determined from the information sequence. This restriction results in a set of $2^{K^*} = 2^{hk}$ codewords, of which a subset of $2^{hk-\nu}$ codewords starts and ends in each of 2^ν states. (If $hk \leq \nu$, then a fraction $2^{hk-\nu}$ of the states contain one valid codeword each.) A tail-biting convolutional code can thus be viewed as a set of 2^ν subcodes, each of which contains $2^{hk-\nu}$ codewords. Maximum likelihood decoding is then performed by applying the Viterbi algorithm separately to each of the 2^ν subcodes, resulting in 2^ν candidate codewords, and then choosing the codeword with the best overall metric from among the 2^ν candidates. Similarly, MAP decoding can be performed by applying the BCJR algorithm to each of the 2^ν subcodes and then determining the APP L-values of each information bit by combining the contributions from each subcode. When the number of states is too large for ML decoding, a suboptimum "two-cycle" decoding method that starts decoding in an arbitrary state, say the all-zero state S_0, finds the ending state, say state S_i, with the best survivor metric, and then repeats the Viterbi algorithm for the subcode that starts and ends in state S_i, achieves reasonably good performance [29]. Similar suboptimum versions of MAP decoding for tail-biting codes are presented in [32].

We now illustrate the tail-biting technique with an example.

EXAMPLE 12.11 Two Tail-biting Convolutional Codes

Consider the $(2, 1, 2)$ nonsystematic feedforward convolutional encoder of Example 12.9. The 4-state trellis diagram shown in Figure 12.21(b) represents an information sequence of length $K^* = h = 4$, an input sequence of length $K = h + m = 6$, and an output sequence of length $N = (h + m)n = 12$, resulting in a terminated convolutional code of rate

$$R_t = \frac{K^*}{N} = \frac{4}{12} = \frac{1}{3}. \tag{12.158}$$

Because the information sequence length h is short, the terminated code rate $R_t = 1/3$ is much less than the encoder rate $R = 1/2$; that is, a 33% rate loss occurs in this case. The minimum (free) distance of this $(12, 4)$ block (terminated

convolutional) code is $d_{\min} = 5$, and the number of minimum weight codewords is $A_5 = 4$.

Now, consider the 4-state tail-biting trellis shown in Figure 12.24(a) corresponding to an information sequence $\mathbf{u} = (u_0, u_1, u_2, u_3, u_4, u_5)$ of length $K^* = h = 6$, where we have shown the input labels as well as the output labels on each trellis branch. There are $2^{K^*} = 2^6 = 64$ codewords $\mathbf{v} = (v_0^{(0)}v_0^{(1)}, v_1^{(0)}v_1^{(1)}, v_2^{(0)}v_2^{(1)}, v_3^{(0)}v_3^{(1)}, v_4^{(0)}v_4^{(1)}, v_5^{(0)}v_5^{(1)})$ of length $N^* = hn = 12$ in this code, resulting in a tail-biting convolutional code of rate

$$R_{tb} = \frac{K^*}{N^*} = \frac{6}{12} = \frac{1}{2}, \tag{12.159}$$

the same as the encoder rate R. The 64 codewords are divided into four subcodes containing 16 codewords each. The subcode corresponding to state S_i, $i = 0, 1, 2, 3$, is formed by the set of 16 information sequences whose last two bits (u_4 and u_5) result in the ending state S_i. (Recall that for a feedforward encoder, the last ν bits of the input sequence determine the binary representation of the ending state.) Because all valid codewords must start and end in the same state, the 16 codewords in the subcode corresponding to state S_i are given by the 16 paths that connect the starting state S_i to the ending state S_i. The four subcodes corresponding to the tail-biting trellis of Figure 12.24(a) are shown in Table 12.5(a). Note that the subcodes for states S_1, S_2, and S_3 are cosets of the subcode for state S_0 (see Problem 12.36). We see from Table 12.5(a) that the minimum (free) distance of this (12, 6) block (tail-biting convolutional) code is $d_{\min} = 3$, and the number of minimum-weight codewords is $A_3 = 2$ (boldface codewords in Table 12.5(a)). The lower minimum distance of this code compared with the (12, 4) terminated convolutional code is reasonable, since the rate of the (12, 6) tail-biting code is 50% higher.

In Example 12.11 it is straightforward to find the subcodes in the tail-biting code corresponding to each state, because, as noted earlier, for feedforward encoders the ending state of a fixed-length input sequence is determined by the last ν input bits. Thus, the starting state of each valid tail-biting codeword is also determined by the last ν bits of the input sequence. This is not the case for feedback encoders, however, where the ending state depends on the entire input sequence. Hence, in order to form tail-biting convolutional codes using encoders with feedback, we must find, for each information sequence $\mathbf{u}$, a starting state S_i such that $\mathbf{u}$ also ends in state S_i.

EXAMPLE 12.11 (Continued)

Consider the (2, 1, 2) systematic feedback convolutional encoder with generator matrix

$$\mathbb{G}(D) = [1 \quad (1 + D^2)/(1 + D + D^2)]. \tag{12.160}$$

This systematic feedback encoder is equivalent to the foregoing nonsystematic feedforward encoder; that is, they both produce the same set of codewords. The 4-state tail-biting trellis corresponding to this feedback encoder is shown in Figure 12.24(b) for an information sequence $\mathbf{u}$ of length $K^* = h = 6$, where we have again shown

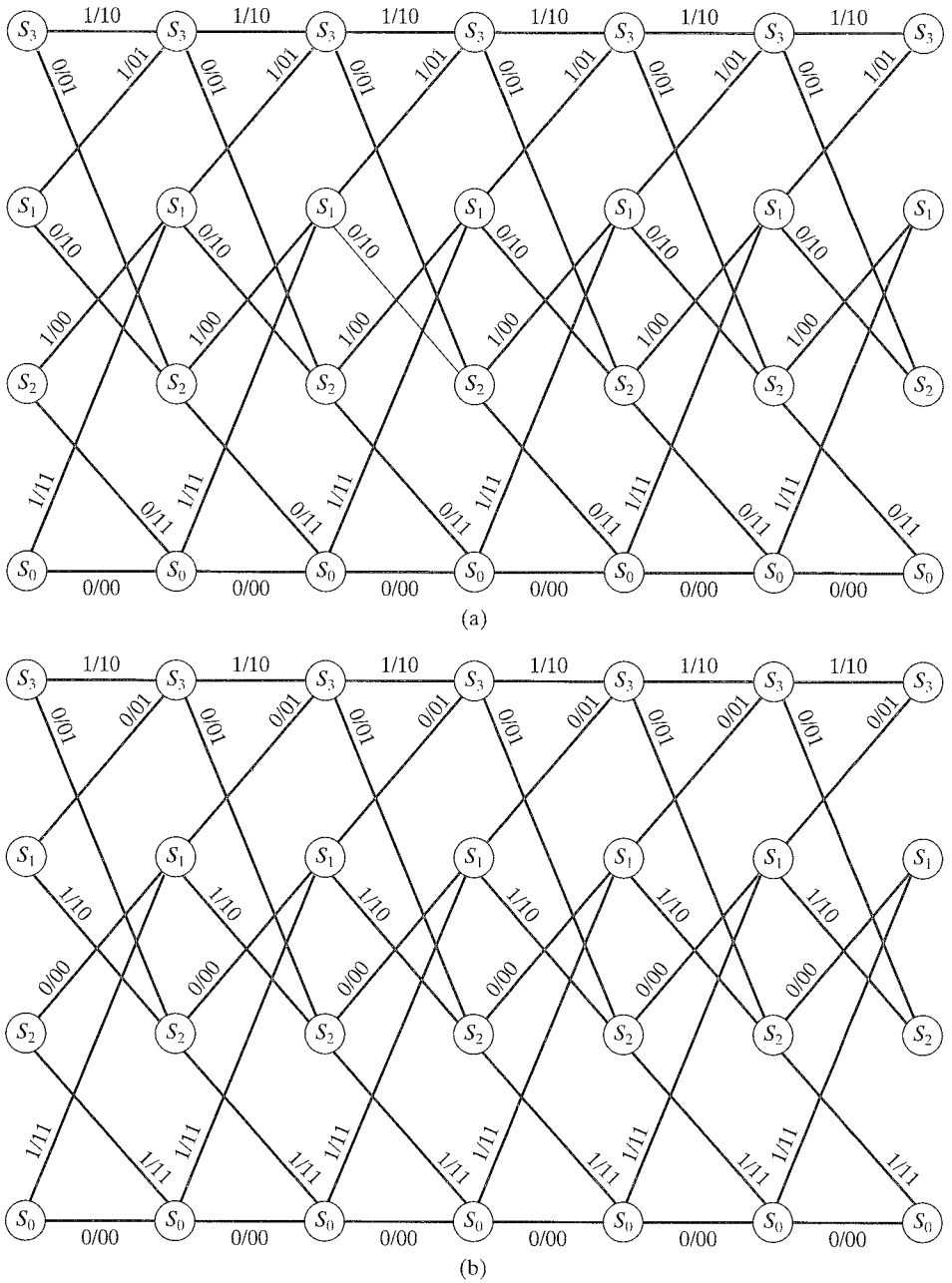


FIGURE 12.24: A 4-state tail-biting trellis with an information sequence of length 6 corresponding to (a) a feedforward encoder and (b) a feedback encoder.

TABLE 12.5: The four subcodes corresponding to (a) the feedforward tail-biting trellis of Figure 12.24(a) and (b) the feedback tail-biting trellis of Figure 12.24(b).

S_0	S_1	S_2	S_3
000000000000	101100000011	110000001110	011100001101
000000111011	101100111000	110000110101	011100110110
000011101100	101111101111	110011100010	011111100001
000011010111	101111010100	110011011001	011111011010
001110110000	100010110011	111110111110	010010111101
001110001011	100010001000	111110000101	010010000110
001101011100	100001011111	111101010010	010001010001
001101100111	100001100100	111101101001	010001101010
111011000000	010111000011	001011001110	100111001101
111011111011	010111111000	001011110101	100111110110
111000101100	010100101111	001000100010	100100100001
111000010111	010100010100	001000011001	100100011010
110101110000	011001110011	000101111110	101001111101
110101001011	011001001000	000101000101	101001000110
110110011100	011010011111	000110010010	101010010001
110110100111	011010100100	000110101001	101010101010

(a)

S_0	S_1	S_2	S_3
0000000000	1000010100	0010000101	0100100001
1101011100	0101001000	0001010010	1010010001
0011010111	0110100100	1100110101	1001000110
1110110000	0101110011	0001101001	0111001101
1101100111	1011000011	1111011001	0100011010
0011101100	0110011111	1100001110	1001111101
0000111011	1011111000	1111100010	0111110110
1110001011	1000101111	0010111110	1010101010

(b)

the input labels as well as the output labels on each trellis branch. Note that the output labels are exactly the same as for the trellis of Figure 12.24(a) corresponding to the equivalent feedforward encoder, but that the input labels are different. In particular, the ending state is no longer determined by the previous two input bits.

To determine the valid codewords in a tail-biting code, we must find, for each information sequence $\mathbf{u}$, a starting state such that $\mathbf{u}$ ends in the same state after $K^* = 6$ time units. Proceeding in a straightforward fashion, we see that the information sequence $\mathbf{u} = (000000)$, when starting in state S_0, ends in state S_0 after six time units and generates the codeword $\mathbf{v} = (00, 00, 00, 00, 00, 00)$, which is thus a valid tail-biting codeword. Now, consider the information sequence $\mathbf{u}' = (100000)$. If S_0 is the starting state, the ending state is S_2, so the associated

codeword is not a valid codeword in a tail-biting code. Trying the other possible starting states, we see that if $\mathbf{u}'$ starts in state S_2, the ending state is S_0; if the starting state is S_1, the ending state is S_3; and the starting state S_3 results in the ending state S_1. In other words, for a block length of $K^* = 6$, there is no valid tail-biting codeword corresponding to the information sequence $\mathbf{u}' = (100000)$. In addition, we note that the information sequence $\mathbf{u} = (000000)$, which generates the valid tail-biting codeword $\mathbf{v} = (00, 00, 00, 00, 00, 00)$ with S_0 as its starting state, generates (different) valid tail-biting codewords with any of the other three states as its starting state. Hence, we find that some information sequences, like $\mathbf{u}$, have several corresponding valid tail-biting codewords, resulting in an ambiguous encoding rule, whereas other information sequences, like $\mathbf{u}'$, have no corresponding valid tail-biting codeword. In other words, in this case, no valid tail-biting code with information block length $K^* = 6$ exists.

Now, consider an information block length of $K^* = 5$ by ignoring the last stage of the trellis in Figure 12.24(b). In this case, each information sequence $\mathbf{u}$ of length $K^* = 5$ corresponds to exactly one valid tail-biting codeword, and a (10, 5) block (tail-biting convolutional) code exists. The code has four subcodes, each containing eight codewords, with one subcode corresponding to each of the four trellis states. The four subcodes are shown in Table 12.5(b). We again note that the subcodes for states S_1, S_2, and S_3 are cosets of the subcode for state S_0. We see from Table 12.5(b) that the minimum (free) distance of this (10, 5) code is $d_{\min} = 3$, and the number of minimum-weight codewords is $A_3 = 5$ (boldface codewords in Table 12.5(b)).

Example 12.11 illustrates an interesting property of tail-biting convolutional codes. For feedforward encoders it is possible to form tail-biting codes of any length, and the determination of the proper starting and ending state for each codeword is straightforward. For feedback encoders, on the other hand, a length restriction exists; that is, it is not possible to form tail-biting codes of certain lengths. Also, the determination of the proper starting and ending state is more complex. (A method for finding the proper starting and ending state, based on a state variable representation of the encoding equations (see Example 11.14), is presented in [33].) Because the tail-biting codes generated by equivalent feedback and feedforward encoders are the same, the feedforward realization is normally preferred for the stated reasons, however, for turbo coding applications (see Chapter 16), feedback encoders are required.

One of the interesting structural properties of tail-biting convolutional codes is the symmetry of the trellis representation. Unlike terminated convolutional codes, where the initial and final m levels of the trellis exhibit an asymmetry (see, e.g., Figure 12.21(b)), the trellis representation of a tail-biting code (see Figure 12.24) is perfectly symmetric. This symmetry allows one to view a tail-biting code using a *circular trellis*, in which one *wraps around* each final state and attaches it to its corresponding initial state, thus forming a circular structure. For example, the circular trellis representation of the tail-biting trellis of Figure 12.24 is shown in Figure 12.25. (It is this circular trellis structure that initially suggested the terminology *tail-biting*.) A tail-biting codeword can then be viewed as starting at an arbitrary state on the circular trellis and then wrapping around it exactly once so as to reach the same ending state. (Note that choosing a different starting point on the

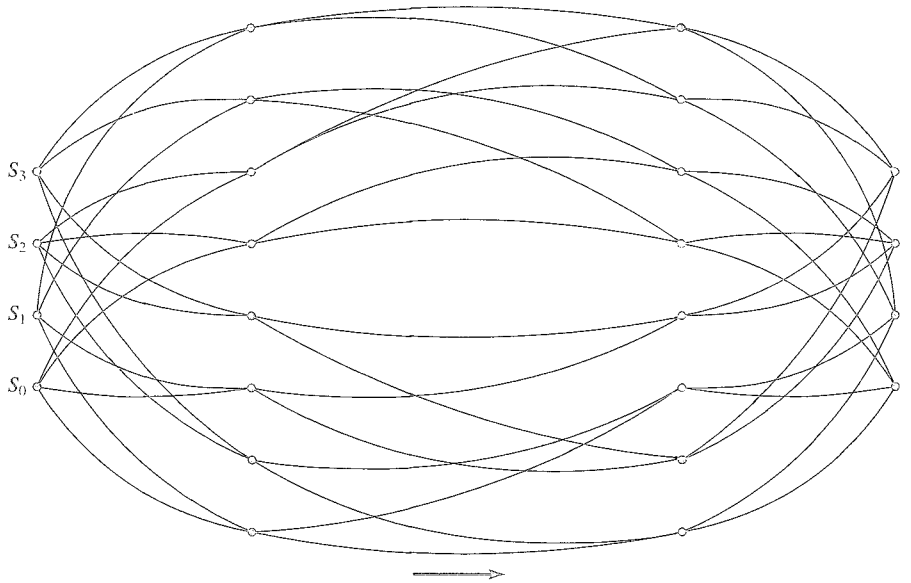


FIGURE 12.25: A 4-state tail-biting circular trellis corresponding to an information sequence of length 6.

trellis simply corresponds to a circular shift of the set of codewords, which does not affect the properties of the associated tail-biting code.)

The technique described for constructing a rate R_{tb} block (tail-biting convolutional) code from a rate $R = R_{tb}$ convolutional encoder suggests that it should be possible to determine the generator matrix of the block code from the generator matrix of the associated convolutional encoder. We illustrate the procedure for feedforward convolutional encoders. (For feedback encoders, the procedure is more complex.) First, we consider the semi-infinite (time-domain) generator matrix $\mathbb{G}_c$ of the unterminated convolutional code given in (11.22) as

$$\mathbb{G}_c = \begin{bmatrix} \mathbb{G}_0 & \mathbb{G}_1 & \mathbb{G}_2 & \cdots & \mathbb{G}_m & & \\ & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_{m-1} & \mathbb{G}_m & \\ & & \mathbb{G}_0 & \cdots & \mathbb{G}_{m-2} & \mathbb{G}_{m-1} & \mathbb{G}_m \\ & & \ddots & & & & \end{bmatrix}, \tag{12.161}$$

Then, we form the finite $K^* \times N$ matrix $\mathbb{G}_b^t$ corresponding to the block (terminated convolutional) code with K^* information bits. $\mathbb{G}_b^t$ is given by

$$\mathbb{G}_b^t = \begin{bmatrix} \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_m & & & & & & \\ & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_m & & & & & \\ & & \ddots & \ddots & \vdots & \ddots & & & & \\ & & & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_m & & & \\ & & & & \mathbb{G}_0 & \cdots & \mathbb{G}_{m-1} & \mathbb{G}_m & & \\ & & & & & \ddots & \vdots & & \ddots & \\ & & & & & & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_m \end{bmatrix}. \tag{12.162}$$

Finally, we obtain the $K^* \times N^*$ matrix $\mathbb{G}_b^{tb}$ corresponding to the block (tail-biting) convolutional code by adding the last nm columns of $\mathbb{G}_b^t$ to the first nm columns and then deleting the last nm columns (recall that $N = N^* + nm$). Thus, $\mathbb{G}_b^{tb}$ is given by

$$\mathbb{G}_b^{tb} = \begin{bmatrix} \mathbb{G}_0 & \mathbb{G}_1 & \cdots & & \mathbb{G}_m & & & \\ & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & & \mathbb{G}_m & & \\ & & \ddots & \ddots & \vdots & & \ddots & \\ & & & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & & \mathbb{G}_m \\ \mathbb{G}_m & & & & \mathbb{G}_0 & \mathbb{G}_1 & \cdots & \mathbb{G}_{m-1} \\ \mathbb{G}_{m-1} & \mathbb{G}_m & & & & \ddots & \ddots & \vdots \\ \vdots & & \ddots & & & & \ddots & \mathbb{G}_1 \\ \mathbb{G}_1 & \mathbb{G}_2 & \cdots & \mathbb{G}_m & & & & \mathbb{G}_0 \end{bmatrix}. \tag{12.163}$$

Comparing (12.162) with (12.163), we see that the effect of tail-biting is to wrap around the last nm columns of $\mathbb{G}_b^t$ to its first nm columns. The 2^{K^*} codewords in the block (tail-biting) convolutional code then correspond to the row space of $\mathbb{G}_b^{tb}$. It is interesting to note that the procedure is reversible; that is, if a rate R_{tb} block code exists with generator matrix $\mathbb{G}$ in the form of (12.163), $\mathbb{G}$ can be unwrapped to form a convolutional encoder with rate $R = R_{tb}$.

EXAMPLE 12.11 (Continued)

Applying the wrap around procedure to the generator matrix of the feedforward convolutional encoder in this example for an information block size of $K^* = 6$, we obtain the generator matrix

$$\mathbb{G}_b^{tb} = \begin{bmatrix} 11\ 10\ 11\ 00\ 00\ 00 \\ 00\ 11\ 10\ 11\ 00\ 00 \\ 00\ 00\ 11\ 10\ 11\ 00 \\ 00\ 00\ 00\ 11\ 10\ 11 \\ 11\ 00\ 00\ 00\ 11\ 10 \\ 10\ 11\ 00\ 00\ 00\ 11 \end{bmatrix}. \tag{12.164}$$

It is easy to verify (see Problem 12.38) that the row space of $\mathbb{G}_b^{tb}$ in (12.164) is the (12, 6) tail-biting code of Table 12.5(a).

In Table 12.6 we list, for constraint lengths $\nu = 1, 2, \cdots, 6$ and rates $R_{tb} = K^*/N^* = 1/3, 1/2$, and $2/3$, the block (tail-biting convolutional) codes with the largest minimum distance d_{min} and the smallest number of nearest neighbors $A_{d_{min}}$. In each case the best encoder is listed for all information block lengths satisfying $h = K^*/k \geq 2$. (Recall that for rates $R_{tb} = 1/3$ and $1/2$, $K^* = h$ and $\nu = m$, whereas for rate $R_{tb} = 2/3$, $K^* = 2h$ and $\nu = 2m$.)

TABLE 12.6: Generator matrices for tail-biting convolutional codes with constraint lengths $\nu = 1, 2, \ldots 6$ and rates (a) $R_{tb} = 1/3$, (b) $R_{tb} = 1/2$, and (c) $R_{tb} = 2/3$.

ν	1					2					3				
K^*	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$
2	1	1	3	4	3	1	3	7	4	3	15	3	7	4	3
3	1	1	3	4	3	1	3	7	4	3	15	7	17	4	3
4	1	3	3	4	1	1	3	7	6	12	1	3	13	6	12
5	1	3	3	5	6	1	3	7	6	10	1	3	7	7	15
6	1	3	3	5	6	5	3	7	6	1	15	3	7	8	45
7					K^*	5	3	7	7	8	15	7	17	8	28
8						5	3	7	7	8	11	13	17	8	13
9						5	3	7	7	9	11	13	17	8	9
10						5	7	7	8	45	11	13	17	9	20
11						5	7	7	8	33	15	13	17	10	44
12						5	7	7	8	27	15	13	17	10	36
13						5	7	7	8	26					$3K^*$
14										$2K^*$					
15															
16															
17															
18															
19															
20															
21															
22															
23															

Adapted from [13].

(continued overleaf)

TABLE 12.6: (*continued*)

v	4					5					6				
K^*	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	$g^{(2)}$	d_{min}	$A_{d_{min}}$
2	31	35	23	4	3	15	3	57	4	3	61	173	67	4	3
3	31	35	23	4	3	21	73	57	4	3	61	173	67	4	3
4	25	3	7	6	12	51	43	57	6	12	75	3	127	6	12
5	1	23	13	7	15	61	75	73	7	15	61	133	117	7	15
6	31	3	7	8	45	31	43	67	8	45	61	173	67	8	45
7	31	25	7	8	21	31	73	57	8	21	111	165	73	8	21
8	5	27	37	8	1	71	3	67	8	1	11	13	137	8	1
9	31	35	23	10	99	15	3	57	10	99	61	371	67	10	99
10	25	27	17	10	46	55	23	27	10	21	145	63	67	10	21
11	25	27	17	10	11	31	15	73	11	55	121	105	177	11	33
12	35	23	27	10	6	71	75	13	12	200	65	147	177	12	178
13	35	33	37	12	182	71	55	73	12	39	165	173	17	12	39
14	25	33	37	12	126	35	53	57	12	14	43	127	117	13	126
15	31	25	37	11	15	71	65	73	13	75	45	133	137	14	390
16	25	33	37	12	80	51	73	57	13	32	151	135	77	14	168
17	25	33	37	12	85	45	73	67	13	34	75	153	117	14	51
18	25	33	37	12	93	71	65	57	13	18	131	161	167	15	240
19	25	33	37	12	95					K^*	151	175	153	15	114
20					$5K^*$						135	123	157	15	60
21											165	133	117	15	63
22											135	123	157	15	66
23															$3K^*$

(a)

TABLE 12.6: (*continued*)

ν	1				2				3				4				5				6			
K^*	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$	$g^{(0)}$	$g^{(1)}$	d_{min}	$A_{d_{min}}$
2	1	3	2	1	3	7	2	1	15	3	2	1	11	37	2	1	65	7	2	1	161	67	2	1
3	1	3	3	4	1	5	3	4	15	17	3	4	31	6	3	4	5	73	3	4	161	67	3	4
4	1	3	3	4	1	7	4	14	1	13	4	14	15	23	4	14	31	57	4	14	111	43	4	14
5				K^*	3	7	4	10	5	13	4	10	31	3	4	10	45	33	4	10	161	67	4	10
6					3	7	4	9	15	3	4	6	31	3	4	6	35	57	4	6	41	133	4	6
7					5	7	4	7	15	17	4	7	11	37	4	7	65	7	4	7	161	67	4	7
8					5	7	4	2	15	3	5	24	31	3	5	24	65	7	5	24	65	153	5	24
9					5	7	5	18	15	7	6	102	25	13	6	102	61	67	6	102	135	177	6	102
10					5	7	5	12	15	7	6	90	15	23	6	90	75	3	6	40	155	157	6	40
11					5	7	5	11	15	17	6	44	35	37	6	44	65	7	7	176	111	123	7	176
12								K^*	15	17	6	48	23	27	6	30	5	73	6	20	141	67	8	759
13									15	17	6	13	35	37	6	13	35	73	7	117	111	143	7	117
14												K^*	31	33	7	72	11	67	7	56	153	37	8	546
15													35	23	7	75	15	57	8	450	141	127	8	450
16													35	23	7	64	15	57	8	348	135	17	8	238
17													31	33	7	34	65	37	8	170	165	63	8	153

(*continued overleaf*)

TABLE 12.6: (*continued*)

ν	1				2				3				4				5				6			
K^*	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{min}	$A_{d_{min}}$
18													35	23	7	54	75	63	8	117	135	113	8	72
19													35	23	7	38	35	57	8	76	55	163	8	19
20																$2K^*$	65	57	8	60	131	117	9	280
21																	65	57	8	42	161	155	9	189
22																	51	77	8	44	135	163	10	1364
23																	55	57	8	46	135	163	10	989
24																	65	57	8	24	151	163	9	96
25																				K^*	155	117	10	575
26																					135	163	10	494
27																					155	117	10	378
28																					135	163	10	364
29																					155	117	10	319
30																					135	163	10	390
31																					155	117	10	341
32																					135	163	10	384
33																					155	117	10	363
34																								$11K^*$

(b)

TABLE 12.6: (*continued*)

ν	2					4					6				
K^*	$\mathbb{h}^{(2)}$	$\mathbb{h}^{(1)}$	$\mathbb{h}^{(0)}$	d_{min}	$A_{d_{min}}$	$\mathbb{h}^{(2)}$	$\mathbb{h}^{(1)}$	$\mathbb{h}^{(0)}$	d_{min}	$A_{d_{min}}$	$\mathbb{h}^{(2)}$	$\mathbb{h}^{(1)}$	$\mathbb{h}^{(0)}$	d_{min}	$A_{d_{min}}$
4	7	3	1	2	3	25	7	3	2	3	127	45	123	2	3
6	7	3	1	2	3	5	23	27	2	3	173	45	71	2	3
8	7	3	1	3	16	25	7	3	3	16	127	45	123	3	16
10	7	3	1	3	15	7	15	1	4	105	133	117	141	4	105
12	7	5	3	3	8	35	7	3	4	78	161	27	135	4	78
14	7	5	3	3	7	31	7	3	4	56	155	107	1	4	56
16					$K^*/2$	27	25	33	4	26	177	35	33	4	26
18						17	31	15	4	18	57	41	173	4	18
20						17	31	15	5	172	145	67	63	5	162
22						5	23	27	5	132	105	177	121	6	1276
24						5	23	27	5	96	51	43	177	6	1036
26						23	37	35	5	52	165	67	163	6	767
28						27	25	33	5	42	153	21	135	6	329
30						27	35	23	5	45	137	31	151	6	195
32						25	31	27	5	32	163	145	165	6	104
34										K^*	165	133	137	6	51
36											135	103	131	6	36
38											77	113	121	6	38
40											137	163	51	6	20
42											121	123	147	7	528
44											121	123	147	7	418
46											121	123	147	7	414
48											121	123	147	7	432
50											121	123	147	7	425

(c)

We see from Table 12.6 that, in general, for a given rate R_{tb} and constraint length ν, the minimum distance d_{min} of the best block (tail-biting convolutional) code increases, or the number of nearest neighbors decreases, as the information block length K^* increases. Once K^* reaches a certain value, though, the minimum distance d_{min} of the best block (tail-biting convolutional) code is limited by the free distance d_{free} of the best terminated convolutional code with constraint length ν, and no further increase in d_{min} is possible; however, the number of nearest neighbors $A_{d_{min}}$ continues to grow linearly with K^*. Once this limit is reached, the generator sequences $\mathbf{g}^{(j)}$ (parity-check sequences $\mathbf{h}^{(j)}$ in the rate $R_{tb} = 2/3$ case) and the minimum distance d_{min} stay the same, and in Table 12.6 we simply list the growth rate of $A_{d_{min}}$. In other words, for a given R_{tb} and ν, block (tail-biting convolutional) codes improve as K^* increases up to a point, and then the codes get worse. Similarly, we can see from Table 12.6 that for a given R_{tb} and K^*, block (tail-biting convolutional) codes improve as ν increases up to a point, and then d_{min} and $A_{d_{min}}$ remain the same. Thus, the best block (tail-biting convolutional) codes are obtained by choosing the length K^* or the constraint length ν only as large as is needed to achieve the desired combination of d_{min} and $A_{d_{min}}$. It is worth noting that many of the best binary block codes can be represented as tail-biting convolutional codes, and thus they can be decoded using the ML (Viterbi) or MAP (BCJR) soft-decision decoding algorithms (see Problem 12.39).

PROBLEMS

12.1 Draw the trellis diagram for the (3, 2, 2) encoder in Example 11.2 and an information sequence of length $h = 3$ blocks. Find the codeword corresponding to the information sequence $\mathbf{u} = (11, 01, 10)$. Compare the result with (11.16) in Example 11.2.

12.2 Show that the path $\mathbf{v}$ that maximizes $\sum_{l=0}^{N-1} \log P(r_l|v_l)$ also maximizes $\sum_{l=0}^{N-1} c_2[\log P(r_l|v_l) + c_1]$, where c_1 is any real number and c_2 is any positive real number.

12.3 Find the integer metric table for the DMC of Figure 12.3 when $c_1 = 1$ and $c_2 = 10$. Use the Viterbi algorithm to decode the received sequence $\mathbf{r}$ of Example 12.1 with this integer metric table and the trellis diagram of Figure 12.1. Compare your answer with the result of Example 12.1.

12.4 Consider a binary-input, 8-ary output DMC with transition probabilities $P(r_l|v_l)$ given by the following table:

$v_l^{(j)}$ \ $r_l^{(j)}$	$\mathbb{0}_1$	$\mathbb{0}_2$	$\mathbb{0}_3$	$\mathbb{0}_4$	$\mathbb{1}_4$	$\mathbb{1}_3$	$\mathbb{1}_2$	$\mathbb{1}_1$
0	0.434	0.197	0.167	0.111	0.058	0.023	0.008	0.002
1	0.002	0.008	0.023	0.058	0.111	0.167	0.197	0.434

Find the metric table and an integer metric table for this channel.

12.5 Consider the (2, 1, 3) encoder of Figure 11.1 with

$$\mathbf{G}(D) = [1 + D^2 + D^3 \quad 1 + D + D^2 + D^3]$$

a. Draw the trellis diagram for an information sequence of length $h = 4$.

b. Assume a codeword is transmitted over the DMC of Problem 12.4. Use the Viterbi algorithm to decode the received sequence $\mathbf{r} = (1_2 1_1, 1_2 0_1, 0_3 0_1, 0_1 1_3, 1_2 0_2, 0_3 1_1, 0_3 0_2)$.

12.6 The DMC of Problem 12.4 is converted to a BSC by combining the soft-decision outputs 0_1, 0_2, 0_3, and 0_4 into a single hard-decision output 0, and the soft-decision outputs 1_1, 1_2, 1_3, and 1_4 into a single hard-decision output 1. A codeword from the code of Problem 12.5 is transmitted over this channel. Use the Viterbi algorithm to decode the hard-decision version of the received sequence in Problem 12.5 and compare the result with Problem 12.5.

12.7 A codeword from the code of Problem 12.5 is transmitted over a continuous-output AWGN channel. Use the Viterbi algorithm to decode the (normalized by $\sqrt{E_S}$) received sequence $\mathbf{r} = (+1.72, +0.93, +2.34, -3.42, -0.14, -2.84, -1.92, +0.23, +0.78, -0.63, -0.05, +2.95, -0.11, -0.55)$.

12.8 Consider a binary-input, continuous-output AWGN channel with signal-to-noise ratio $E_s/N_0 = 0$ dB.

a. Sketch the conditional pdf's of the (normalized by $\sqrt{E_s}$) received signal r_l given the transmitted bits $v_l = \pm 1$.

b. Convert this channel into a binary-input, 4-ary output symmetric DMC by placing quantization thresholds at the values $r_l = -1, 0$, and $+1$, and compute the transition probabilities for the resulting DMC.

c. Find the metric table and an integer metric table for this DMC.

d. Repeat parts (b) and (c) using quantization thresholds $r_l = -2, 0$, and $+2$.

12.9 Show that (12.21) is an upper bound on P_d for d even.

12.10 Consider the (2, 1, 3) encoder of Problem 12.5. Evaluate the upper bounds on event-error probability (12.25) and bit-error probability (12.29) for a BSC with transition probability

a. $p = 0.1$,

b. $p = 0.01$.

(*Hint*: Use the WEFs derived for this encoder in Example 11.12.)

12.11 Repeat Problem 12.10 using the approximate expressions for $P(E)$ and $P_b(E)$ given by (12.26) and (12.30).

12.12 Consider the (3, 1, 2) encoder of (12.1). Plot the approximate expression (12.36) for bit-error probability $P_b(E)$ on a BSC as a function of E_b/N_0 in decibels. Also plot on the same set of axes the approximate expression (12.37) for $P_b(E)$ without coding. The *coding gain* (in decibels) is defined as the difference between the E_b/N_0 ratio needed to achieve a given bit-error probability with coding and without coding. Plot the coding gain as a function of $P_b(E)$. Find the value of E_b/N_0 for which the coding gain is 0 dB, that is, the *coding threshold*.

12.13 Repeat Problem 12.12 for an AWGN channel with unquantized demodulator outputs, that is, a continuous-output AWGN channel, using the approximate expression for $P_b(E)$ given in (12.46).

12.14 Consider using the (3, 1, 2) encoder of (12.1) on the DMC of Problem 12.4. Calculate an approximate value for the bit-error probability $P_b(E)$ based on the bound of (12.39b). Now, convert the DMC to a BSC, as described in Problem 12.6, compute an approximate value for $P_b(E)$ on this BSC using (12.29), and compare the two results.

12.15 Prove that the rate $R = 1/2$ quick-look-in encoders defined by (12.58) are noncatastrophic.

12.16 Consider the following two nonsystematic feedforward encoders: (1) the encoder for the (2, 1, 7) optimum code listed in Table 12.1(c) and (2) the encoder for the (2, 1, 7) quick-look-in code listed in Table 12.2. For each of these codes find

a. the soft-decision asymptotic coding gain γ;

b. the approximate event-error probability on a BSC with $p = 10^{-2}$;

c. the approximate bit-error probability on a BSC with $p = 10^{-2}$;

d. the error probability amplification factor A.

12.17 Using trial-and-error methods, construct a (2, 1, 7) systematic feedforward encoder with maximum d_{free}. Repeat Problem 12.16 for this code.

12.18 Consider the (15,7) and (31,16) cyclic BCH codes. For each of these codes find

a. the polynomial generator matrix and a lower bound on d_{free} for the rate $R = 1/2$ convolutional code derived from the cyclic code using Construction 12.1;

b. the polynomial generator matrix and a lower bound on d_{free} for the rate $R = 1/4$ convolutional code derived from the cyclic code using Construction 12.2.

(*Hint:* d_h is at least one more than the maximum number of consecutive powers of α that are roots of $\mathbf{h}(X)$.)

12.19 Consider the (2, 1, 1) systematic feedforward encoder with $\mathbf{G}(D) = [1 \quad 1 + D]$.

a. For a continuous-output AWGN channel and a truncated Viterbi decoder with path memory $\tau = 2$, decode the received sequence $\mathbf{r} = (+1.5339, +0.6390, -0.6747, -3.0183, +1.5096, +0.7664, -0.4019, +0.3185, +2.7121, -0.7304, +1.4169, -2.0341, +0.8971, -0.3951, +1.6254, -1.1768, +2.6954, -1.0575)$ corresponding to an information sequence of length $h = 8$. Assume that at each level the survivor with the best metric is selected and that the information bit τ time units back on this path is decoded.

b. Repeat (a) for a truncated Viterbi decoder with path memory $\tau = 4$.

c. Repeat (a) for a Viterbi decoder without truncation.

d. Are the final decoded paths the same in all cases ? Explain.

12.20 Consider the (3, 1, 2) encoder of Problem 11.19.

a. Find $A_1(W, X, L)$, $A_2(W, X, L)$, and $A_3(W, X, L)$.

b. Find τ_{min}.

c. Find $d(\tau)$ and $A_{d(\tau)}$ for $\tau = 0, 1, 2, \cdots, \tau_{min}$.

d. Find an expression for $\lim_{\tau \to \infty} d(\tau)$.

12.21 A codeword from the trellis diagram of Figure 12.1 is transmitted over a BSC. To determine correct symbol synchronization, each of the three 21-bit subsequences of the sequence

$$\mathbf{r} = 0\,1\,1\,1\,0\,0\,1\,1\,0\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,1\,1$$

must be decoded, where the two extra bits in $\mathbf{r}$ are assumed to be part of a preceding and/or a succeeding codeword. Decode each of these subsequences and determine which one is most likely to be the correctly synchronized received sequence.

12.22 Consider the binary-input, continuous-output AWGN channel of Problem 12.8.

a. Using the optimality condition of (12.84), calculate quantization thresholds for DMCs with $Q = 2, 4$, and 8 output symbols. Compare the thresholds obtained for $Q = 4$ with the values used in Problem 12.8.

b. Find the value of the Bhattacharyya parameter D_0 for each of these channels and for a continuous-output AWGN channel.

c. Fixing the signal energy $\sqrt{E_s} = 1$ and allowing the channel SNR E_s/N_0 to vary, determine the increase in the SNR required for each of the DMCs to achieve

the same value of D_0 as the continuous-output channel. This SNR difference is called the *decibel loss* associated with receiver quantization. (*Note:* Changing the SNR also changes the quantization thresholds.)

(*Hint:* You will need to write a computer program to solve this problem.)

12.23 Verify that the two expressions given in (12.89) for the modified metric used in the SOVA algorithm are equivalent.

12.24 Define $L(r) = \ln \lambda(r)$ as the log-likelihood ratio, or L-value, of a received symbol r at the output of an unquantized binary input channel. Show that the L-value of an AWGN channel with binary inputs $\pm\sqrt{E_s}$ and SNR E_s/N_0 is given by

$$L(r) = (4\sqrt{E_s}/N_0)r.$$

12.25 Verify that the expressions given in (12.98) are correct, and find the constant c.

12.26 Consider the encoder, channel, and received sequence of Problem 12.19.

a. Use the SOVA with full path memory to produce a soft output value for each decoded information bit.

b. Repeat (a) for the SOVA with path memory $\tau = 4$.

12.27 Derive the expression for the backward metric given in (12.117).

12.28 Verify the derivation of (12.123) and show that $A_l = e^{-\frac{L_a(u_l)/2}{1+e^{-L_a(u_l)}}}$ is independent of the actual value of u_l.

12.29 Derive the expressions for the $\max^*(x, y)$ and $\max^*(x, y, z)$ functions given in (12.127) and (12.131), respectively.

12.30 Consider the encoder and received sequence of Problem 12.19.

a. For an AWGN channel with $E_s/N_0 = 1/2$ (−3 dB), use the log-MAP version of the BCJR algorithm to produce a soft output value for each decoded information bit. Find the decoded information sequence $\hat{\mathbf{u}}$.

b. Repeat (a) using the Max-log-MAP algorithm.

12.31 Repeat Problem 12.5 using the probability-domain version of the BCJR algorithm.

12.32 Show that using the normalized forward and backward metrics $A_l(s)$ and $B_l(s')$ instead of $\alpha_l(s)$ and $\beta_l(s')$, respectively, to evaluate the joint pdf's in (12.115) has no effect on the APP L-values computed using (12.111).

12.33 Verify all the computations leading to the determination of the final APP L-values in Example 12.9.

12.34 Repeat Example 12.9 for the case when the DMC is converted to a BSC, as described in Problem 12.6, and the received sequence $\mathbf{r}$ is replaced by its hard-decision version. Compare the final APP L-values in the two cases.

12.35 Consider an 8-state rate $R = 1/2$ mother code with generator matrix

$$\mathbb{G}(D) = [1 + D + D^3 \quad 1 + D^2 + D^3].$$

Find puncturing matrices $\mathbb{P}$ for the rate $R = 2/3$ and $R = 3/4$ punctured codes that give the best free distances. Compare your results with the free distances obtained using the 8-state mother code in Table 12.4.

12.36 Prove that the subcode corresponding to any nonzero state S_i, $i \neq 0$, in a tail-biting convolutional code is a coset of the subcode corresponding to the all-zero state S_0.

12.37 For the rate $R = 1/2$ feedback encoder tail-biting trellis in Figure 12.24(b), determine the parameters d_{min} and $A_{d_{min}}$ for information block lengths $K^* = 7, 8$, and 9. Is it possible to form a tail-biting code in each of these cases?

12.38 Verify that the row space of the tail-biting generator matrix in (12.164) is identical to the tail-biting code of Table 12.5(a).

12.39 Consider the rate $R = 4/8$, constraint length $\nu = 4$ feedforward convolutional encoder with generator matrix

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & 0 & 1 & 0 & 1+D & 1 & 1 & 1 \\ 0 & 1+D & 1 & 1 & D & 1+D & 1 & 0 \\ D & D & 1+D & 0 & 0 & D & 1+D & 1 \\ 0 & D & 0 & 1+D & D & D & D & 1+D \end{bmatrix}$$

a. Draw the controller canonical form encoder diagram.
b. Draw an $h = 3$ ($K^* = 12$), 16-state tail-biting trellis for this encoder.
c. Find the tail-biting generator matrix $\mathbf{G}_b^{tb}$ for the resulting (24, 12) tail-biting code.
d. Show that this code has $d_{min} = 8$ and is equivalent to the (24, 12) extended Golay code.
(*Note:* The convolutional code generated by $\mathbf{G}(D)$ is called the *Golay convolutional code.*)

BIBLIOGRAPHY

1. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Inform. Theory*, IT-13: 260–69, April 1967.

2. J. K. Omura, "On the Viterbi Decoding Algorithm," *IEEE Trans. Inform. Theory*, IT-15: 177–79, January 1969.

3. G. D. Forney, Jr., "The Viterbi Algorithm," *Proc. IEEE*, 61: 268–78, March 1973.

4. G. D. Forney, Jr., "Convolutional Codes II: Maximum Likelihood Decoding," *Inform. Control*, 25: 222–66, July 1974.

5. L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Inform. Theory*, IT-20: 284–87, March 1974.

6. P. L. McAdam, L. R. Welch, and C. L. Weber, "M.A.P. Bit Decoding of Convolutional Codes," in *Book of Abstracts IEEE International Symposium on Information Theory*, p. 91, Asilomar, Calif., February 1972.

7. L. N. Lee, "Real-Time Minimal-Bit-Error Probability Decoding of Convolutional Codes," *IEEE Trans. Commun.*, COM-22: 146–51, February 1974.

8. C. R. P. Hartmann and L. D. Rudolph, "An Optimum Symbol-by-Symbol Decoding Rule for Linear Codes," *IEEE Trans. Inform. Theory*, IT-22: 514–17, September 1976.

9. J. Hagenauer and P. Hoeher, "A Viterbi Decoding Algorithm with Soft-Decision Outputs and Its Applications," *Proc. IEEE Global Conference on Communications*, pp. 1680–86, Dallas, Tex., November 1989.

10. A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," *IEEE Trans. Commun. Technol.*, COM-19: 751–72, October 1971.

11. L. Van de Meeberg, "A Tightened Upper Bound on the Error Probability of Binary Convolutional Codes with Viterbi Decoding," *IEEE Trans. Inform. Theory*, IT-20: 389–91, May 1974.

12. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*. McGraw-Hill, New York, 1979.

13. R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. IEEE Press, Piscataway, N.J., 1999.

14. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*. John Wiley, New York, 1965.

15. L. R. Bahl, C. D. Cullum, W. D. Frazer, and F. Jelinek, "An Efficient Algorithm for Computing Free Distance," *IEEE Trans. Inform. Theory*, IT-18: 437–39, May 1972.

16. K. J. Larsen, "Comments on 'An Efficient Algorithm for Computing Free Distance'," *IEEE Trans. Inform. Theory*, IT-19: 577–79, July 1973.

17. J. J. Chang, D. J. Hwang, and M. C. Lin, "Some Extended Results on the Search for Good Convolutional Codes," *IEEE Trans. Inform. Theory*, IT-43: 1682–97, September 1997.

18. J. L. Massey and D. J. Costello, Jr., "Nonsystematic Convolutional Codes for Sequential Decoding in Space Applications," *IEEE Trans. Commun. Technol.*, COM-19: 806–13, October 1971.

19. J. L. Massey, D. J. Costello, Jr., and J. Justesen, "Polynomial Weights and Code Constructions," *IEEE Trans. Inform. Theory*, IT-19: 101–10, January 1973.

20. J. Justesen, "New Convolutional Code Constructions and a Class of Asymptotically Good Time-Varying Codes," *IEEE Trans. Inform. Theory*, IT-19: 220–25, March 1973.

21. R. M. Tanner, "Convolutional Codes from Quasi-Cyclic Codes: A Link between the Theories of Block and Convolutional Codes," Technical Report, Computer Science Research Laboratory, UC Santa Cruz, November 1987.

22. O. M. Collins, "The Subtleties and Intricacies of Building a Constraint Length 15 Convolutional Decoder," *IEEE Trans. Commun.*, COM-40: 1810–19, December 1992.

23. F. Hemmati and D. J. Costello, Jr., "Truncation Error Probability in Viterbi Decoding," *IEEE Trans. Commun.* COM-25: 530–32, May 1977.

24. F. Hemmati and D. J. Costello, Jr., "Asymptotically Catastrophic Convolutional Codes," *IEEE Trans. Inform. Theory*, IT-26: 298–304, May 1980.

25. M. P. C. Fossorier and S. Lin, "Differential Trellis Decoding of Convolutional Codes," *IEEE Trans. Inform. Theory*, IT-46, 1046–53, May 2000.

26. W. E. Ryan, "Concatenated Convolutional Codes and Iterative Decoding," in *Wiley Encyclopedia of Telecommunications*, edited by J. G. Proakis, John Wiley, New York, 2002.

27. J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured Convolutional Codes of Rate $(n-1)/n$ and Simplified Maximum Likelihood Decoding," *IEEE Trans. Inform. Theory*, IT-25: 97–100, January 1979.

28. G. Solomon and H. C. A. van Tilborg, "A Connection between Block and Convolutional Codes," *SIAM J. Appl. Math.*, 37(2): 358–69, October 1979.

29. H. H. Ma and J. K. Wolf, "On Tail-Biting Convolutional Codes," *IEEE Trans. Commun.*, COM-34: 104–11, February 1986.

30. L. H. C. Lee, *Convolutional Coding: Fundamentals and Applications*, Artech House, Boston, Mass., 1997.

31. J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and Their Applications," *IEEE Trans. Commun.*, COM-36: 389–400, April 1988.

32. J. B. Anderson and M. Hladik, "Tailbiting MAP Decoders," *IEEE J. Selected Areas Commun.*, SAC-16(2): 297–302, February 1998.

33. C. Weiss, C. Bettstetter, and S. Riedel, "Code Construction and Decoding of Parallel Concatenated Tail-Biting codes," *IEEE Trans. Information Theory*, IT-47: 366–86, January 2001.