CHAPTER 16

Turbo Coding

As noted in Chapter 1, Shannon's noisy channel coding theorem implies that arbitrarily low decoding error probabilities can be achieved at any transmission rate R less than the channel capacity C by using sufficiently long block (or constraint) lengths. In particular, Shannon showed that randomly chosen codes, along with maximum likelihood decoding (MLD), can provide capacity-achieving performance. He did this by proving that the *average* performance of a randomly chosen ensemble of codes results in an exponentially decreasing decoding error probability with increasing block (or constraint) length; however, he gave no guidance about how to actually construct good codes, that is, codes that perform at least as well as the average, or to implement MLD for such codes.

In the ensuing years after the publication of Shannon's paper in 1948 [1], a large amount of research was conducted into the construction of specific codes with good error-correcting capabilities and the development of efficient decoding algorithms for these codes. Much of this research has been described in the previous chapters of this book. Typically, the best code designs contain a large amount of structure, either algebraic, as is the case with most block codes, such as RM and BCH codes, or topological, as is the case with most convolutional codes, which can be represented using trellis or tree diagrams. The structure is a key component of the code design, since it can be used to guarantee good minimum distance properties for the code, as in the BCH bound, and since particular decoding methods, such as the Berlekamp–Massey algorithm for BCH codes and the Viterbi algorithm for convolutional codes, are based on this structure. In fact, one can say generally that the more structure a code contains, the easier it is to decode; however, structure does not always result in the best distance properties for a code, and most highly structured code designs usually fall far short of achieving the performance promised by Shannon.

Primarily because of the need to provide structure to develop easily implementable decoding algorithms, little attention was paid to the design of codes with randomlike properties, as originally envisioned by Shannon. Random code designs, because they lacked structure, were thought to be too difficult to decode. In this chapter we discuss a relatively new coding technique, dubbed *turbo coding* by its inventors [2], that succeeds in achieving a randomlike code design with just enough structure to allow for an efficient iterative decoding method. Because of this feature, these codes have exceptionally good performance, particularly at moderate BERs and for large block lengths. In fact, for essentially any code rate and information block lengths greater than about 10^4 bits, turbo codes with iterative decoding can achieve BERs as low as 10^{-5} at SNRs within 1 dB of the Shannon limit, that is, the value of E_b/N_0 for which the code rate equals channel capacity. This typically exceeds the performance of previously known codes of comparable length and decoding complexity by several decibels, although because decoding is iterative, larger decoding delays result.

Another important feature of turbo codes is that they are composed of two or more simple constituent codes, arranged in a variation of the concatenation scheme introduced in Chapter 15, along with a pseudorandom interleaver. Because the interleaver is part of the code design, a complete maximum likelihood decoder for the entire code would be prohibitively complex; however, because more than one code is used, it is possible to employ simple soft-in soft-out (SISO) decoders for each constituent code in an iterative fashion, in which the soft-output values of one decoder are passed to the other, and vice versa, until the final decoding estimate is obtained. Extensive experience with turbo decoders has shown that this simple, suboptimum iterative decoding approach almost always converges to the maximum likelihood solution.

In summary, turbo coding consists of two fundamental ideas: a code design that produces a code with randomlike properties, and a decoder design that makes use of soft-output values and iterative decoding. The basic features of turbo code design are developed in Sections 16.1 to 16.4, and the principles of iterative turbo decoding are presented in Section 16.5.

The original concept of turbo coding was introduced in a paper by Berrou, Glavieux, and Thitimajshima [2] delivered at the IEEE International Conference on Communications held in Geneva, Switzerland, in May 1993, and it was further elaborated in [3, 4]. Many of the earlier ideas of Batill and Hagenauer on randomlike codes and iterative decoding were influential in leading to the turbo coding concept [5–8]. Much of the research community was originally skeptical of the performance claims, but by the end of 1994 the basic results had been confirmed by several other research groups [9, 10]. Two papers by Benedetto and Montorsi [11, 12] provided the first theoretical justification for the exceptional performance of the codes, and further insights were presented in [13, 14]. The research group of Hagenauer was primarily responsible for illuminating the benefits of iterative decoding [15, 16], and additional understanding was achieved in [17, 18]. The quarterly progress reports, beginning in 1995, of Divsalar, Pollara, Dolinar, and colleagues at the Jet Propulsion Laboratory provide a detailed look at many aspects of turbo coding [19–22] particularly as they affect deep-space communication issues. Numerous variations of turbo coding have also appeared in the literature, such as serial concatenation, hybrid parallel and serial concatenation, and self-concatenation [23–25]. A comprehensive overview of the first five years of turbo coding is given in the book by Heegard and Wicker [26], which also contains a thorough discussion of iterative decoding from the point of view of the theory of belief propagation [27]. Another book covering roughly the same material was written by Vucetic [28], and a readable summary of the main aspects of turbo coding, with an emphasis on iterative decoding, was published by Ryan [29].

16.1 INTRODUCTION TO TURBO CODING

A block diagram of the basic turbo encoding structure is illustrated in Figure 16.1(a), and a specific example is shown in Figure 16.1(b). The basic system consists of an information (input) sequence, two $(2, 1, \nu)$ systematic feedback (recursive) convolutional encoders, and an interleaver, denoted by π. We will assume that the information sequence contains K^* information bits plus ν termination bits to return the first encoder to the all-zero state $S_0 = 0$, where ν is the constraint length of the

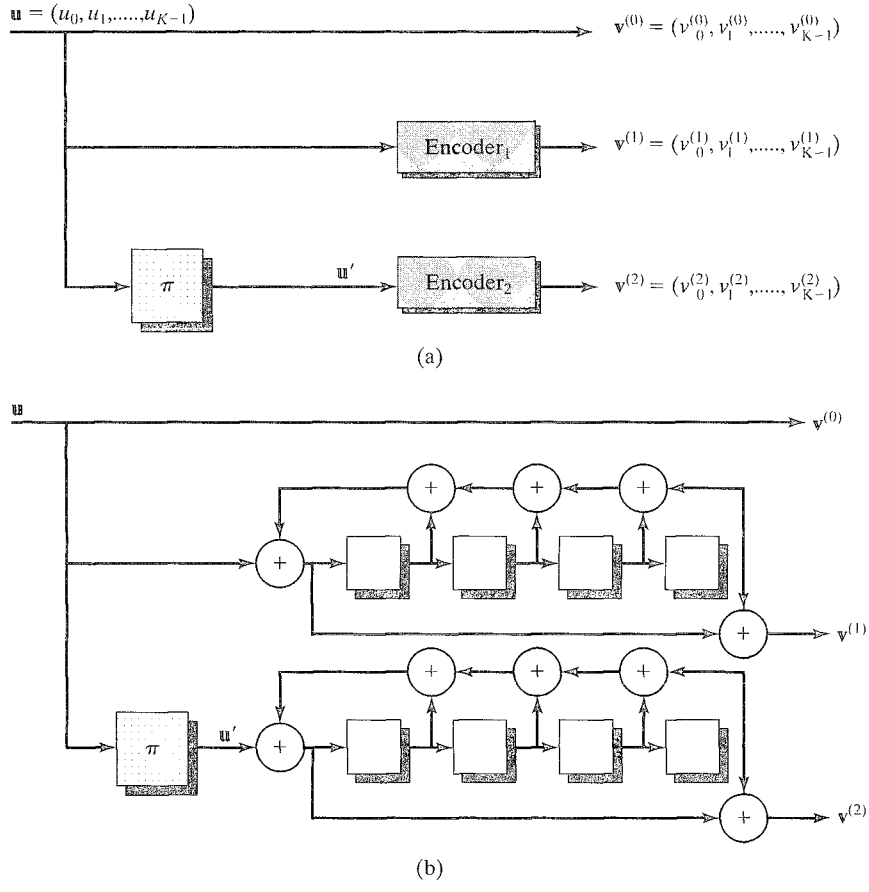


FIGURE 16.1: The basic turbo encoding structure.

first encoder. The information sequence (including termination bits) is considered to be a block of length $K = K^* + \nu$ and is represented by the vector

$$\mathbb{u} = \left(u_0, u_1, \cdots, u_{K-1}\right). \tag{16.1}$$

Because encoding is systematic, the information sequence $\mathbb{u}$ is the first transmitted sequence; that is

$$\mathbb{u} = \mathbb{v}^{(0)} = \left(v_0^{(0)}, v_1^{(0)}, \cdots, v_{K-1}^{(0)}\right). \tag{16.2a}$$

The first encoder generates the parity sequence

$$\mathbb{v}^{(1)} = \left(v_0^{(1)}, v_1^{(1)}, \cdots, v_{K-1}^{(1)}\right). \tag{16.2b}$$

The interleaver reorders or permutes the K bits in the information block so that the second encoder receives a permuted information sequence $\mathbb{u}'$ different from the first. (Note that the second encoder may or may not be terminated.) The parity

sequence generated by the second encoder is represented as

$$\mathbb{v}^{(2)} = \left(v_0^{(2)}, v_1^{(2)}, \cdots, v_{K-1}^{(2)}\right), \tag{16.2c}$$

and the final transmitted sequence (codeword) is given by the vector

$$\mathbb{v} = \left(v_0^{(0)} v_0^{(1)} v_0^{(2)},\ v_1^{(0)} v_1^{(1)} v_1^{(2)},\ \cdots,\ v_{K-1}^{(0)} v_{K-1}^{(1)} v_{K-1}^{(2)}\right), \tag{16.3}$$

so that the overall (terminated) code has length $N = 3K$ and rate $R_t = K^*/N = (K - \nu)/3K \approx 1/3$ for large K.

The information sequence $\mathbb{u}$ along with the first parity sequence $\mathbb{v}^{(1)}$ is referred to as the *first constituent code*, and the permuted information sequence $\mathbb{u}'$ (which is not transmitted) along with the second parity sequence $\mathbb{v}^{(2)}$ is referred to as the *second constituent code*. In Figure 16.1(b) both constituent codes are generated by the same (2, 1, 4) systematic feedback encoder whose generator matrix is given by

$$\mathbb{G}(D) = \begin{bmatrix} 1 & (1 + D^4)/(1 + D + D^2 + D^3 + D^4) \end{bmatrix}. \tag{16.4}$$

The following remarks are related to the typical operation of turbo codes. Explanations will be given in later sections of this chapter.

- To achieve performance close to the Shannon limit, the information block length (interleaver size) K is chosen to be very large, usually at least several thousand bits.
- The best performance at moderate BERs down to about 10^{-5} is achieved with short constraint length constituent encoders, typically $\nu = 4$ or less.
- The constituent codes are normally generated by the same encoder, as in Figure 16.1(b), but this is not necessary for good performance. In fact, some *asymmetric* code designs have been shown to give very good performance [30].
- Recursive constituent codes, generated by systematic feedback encoders, give much better performance than nonrecursive constituent codes, that is, feedforward encoders.
- Bits can be punctured from the parity sequences to produce higher code rates. For example, puncturing alternate bits from $\mathbb{v}^{(1)}$ and $\mathbb{v}^{(2)}$ produces a systematic rate $R = 1/2$ code.
- Bits also can be punctured from the information sequence to produce partially systematic or nonsystematic turbo codes [31].
- Additional constituent codes and interleavers can be used to produce lower-rate codes. For example, rate $R = 1/4$ can be achieved with three constituent codes and two interleavers, as shown in Figure 16.2. This configuration is called a *multiple turbo code* [19].
- The best interleavers reorder the bits in a pseudorandom manner. Conventional block (row–column) interleavers do not perform well in turbo codes, except at relatively short block lengths.

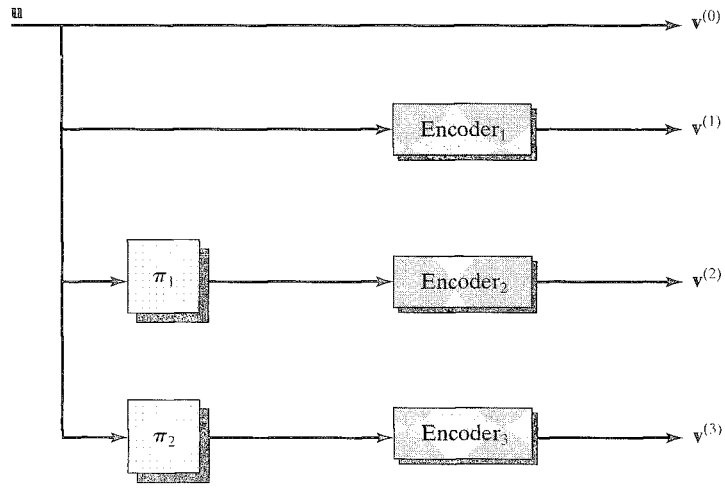


FIGURE 16.2: A rate $R = 1/4$ multiple turbo code.

- Because only the ordering of the bits is changed by the interleaver, the sequence $\mathbf{u}'$ that enters the second encoder has the same *weight* as the sequence $\mathbf{u}$ that enters the first encoder.
- The interleaver is an integral part of the overall encoder, and thus the state complexity of turbo codes is extremely large, making trellis-based ML or MAP decoding impossible.
- Suboptimum iterative decoding, which employs individual SISO decoders for each of the constituent codes in an iterative manner, typically achieves performance within a few tenths of a decibel of overall ML or MAP decoding. The best performance is obtained when the BCJR, or MAP, algorithm is used as the SISO decoder for each constituent code.[1]
- Because the MAP decoder uses a forward–backward algorithm, the information is arranged in blocks. Thus, the first constituent encoder is terminated by appending ν bits to return it to the $\mathbf{0}$ state. Because the interleaver reorders the input sequence, the second encoder will not normally return to the $\mathbf{0}$ state, but this has little effect on performance for large block lengths. If desired, though, modifications can be made to ensure termination of both encoders.
- Block codes also can be used as constituent codes in turbo encoders.
- Decoding can be stopped, and a final decoding estimate declared, after some fixed number of iterations (usually on the order of 10–20) or based on a stopping criterion that is designed to detect when the estimate is reliable with very high probability.

[1] It is important to note that iterative SISO decoding, using individual MAP decoders for each constituent code, is not the same as overall ML or MAP decoding.

The basic encoding scheme illustrated in Figures 16.1 and 16.2 is referred to as *parallel concatenation*, because of its similarity to Forney's original concatenation scheme. Compared with conventional serial concatenation, in parallel concatenation the input to the encoder, rather than its output, enters the second encoder, although it is first permuted by the interleaver. In other words, the two encoders operate in *parallel* on different versions of the information sequence.

The example code shown in Figure 16.1(b), punctured to rate 1/2, is capable of achieving a 10^{-5} BER at an SNR of $E_b/N_0 = 0.7$ dB with an information block length of $K = 2^{16} = 65536$ bits after 18 iterations of a SISO MAP decoder. By comparison, the NASA standard $(2, 1, 6)$ convolutional code with ML decoding requires an E_b/N_0 of 4.2 dB to achieve the same BER. The performance comparison of these two codes is shown in Figure 16.3. Thus, the rate $R = 1/2$ turbo code achieves a 3.5-dB coding gain compared with the $(2, 1, 6)$ convolutional code at a 10^{-5} BER. The decoding complexity of the two codes is roughly equivalent, since a 16-state MAP decoder has about the same complexity as a 64-state Viterbi decoder. This advantage of turbo codes over conventional methods of coding is fairly typical over the entire range of possible code rates; that is, several decibels of coding gain can be achieved at moderate BERs with long turbo codes of the same rate and roughly the same decoding complexity as conventional codes. In addition, the performance of turbo codes at moderate BERs is within 1 dB of capacity. For the example shown in Figure 16.3, the BER performance is only 0.7 dB away from the (unconstrained) capacity and only 0.5 dB away from the capacity for binary input channels.

Turbo codes suffer from two disadvantages: a large decoding delay, owing to the large block lengths and many iterations of decoding required for near-capacity

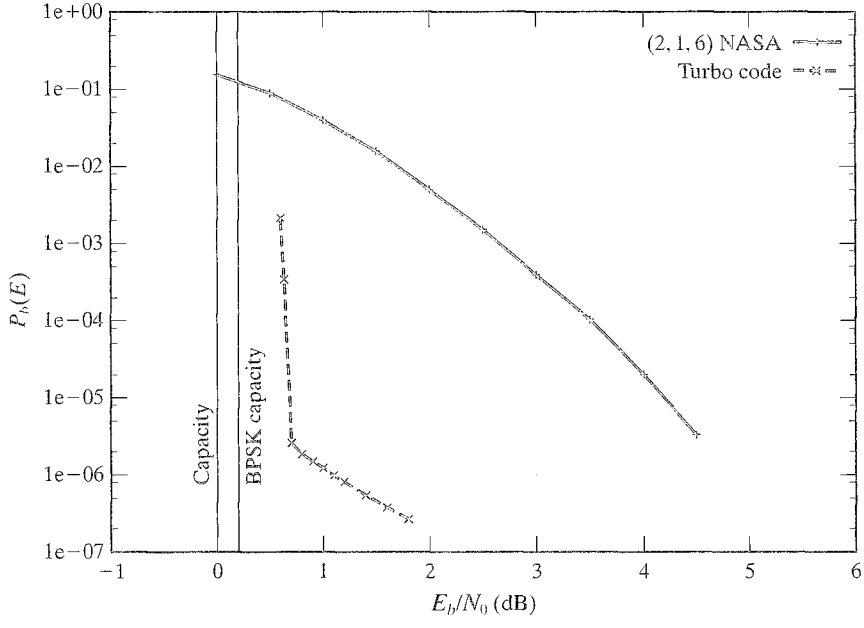


FIGURE 16.3: Performance comparison of convolutional codes and turbo codes.

performance, and significantly weakened performance at BERs below 10^{-5} owing to the fact that the codes have a relatively poor minimum distance, which manifests itself at very low BERs. The large delay appears to make turbo codes unsuitable for real-time applications such as voice transmission and packet communication in high-speed networks. It is possible, though, to trade delay for performance in such a way that turbo codes may be useful in some real-time applications involving block lengths on the order of a few thousand, or even a few hundred, bits. The fact that turbo codes typically do not have large minimum distances causes the performance curve to flatten out at BERs below 10^{-5}, as shown in Figure 16.3. This phenomenon, sometimes called an *error floor*, is due to the unusual weight distribution of turbo codes, which will be discussed in the next section. Because of the error floor, turbo codes may not be suitable for applications requiring extremely low BERs, such as some scientific or command-and-control applications; however, measures can be taken to mitigate this problem. Interleavers can be designed to improve the minimum distance of the code, thus lowering the error floor. Also, an outer code, or a second layer of concatenation, can be used with a turbo code to correct many of the errors caused by the small minimum distance, at a cost of a small decrease in overall rate. Both techniques will be discussed later in this chapter.

The fundamental property of turbo codes that underlies their excellent performance at moderate BERs is the randomlike weight spectrum of the codewords produced by the pseudorandom interleaver when systematic feedback encoders are used. To understand this feature we consider a series of examples.

EXAMPLE 16.1 Weight Spectrum of a Terminated Convolutional Code

Consider the conventional (2, 1, 4) convolutional code with nonsystematic feedforward generator matrix

$$\mathbb{G}_{ff}(D) = \begin{bmatrix} 1 + D + D^2 + D^3 + D^4 & 1 + D^4 \end{bmatrix}. \tag{16.5}$$

The minimum free distance of this code is 6, obtained from the information sequence $\mathbb{u} = (1\,1\,0\,\cdots)$. If the encoder is converted to systematic feedback form, the generator matrix is then given by

$$\mathbb{G}_{fb}(D) = \begin{bmatrix} 1 & (1 + D^4)/(1 + D + D^2 + D^3 + D^4) \end{bmatrix}. \tag{16.6}$$

Because the code is exactly the same, the free distance is still 6, but in this case the minimum-weight codeword is obtained from the information sequence $\mathbb{u} = [1\,0\,0\,0\,0\,1\,0\,0\,\cdots]$; that is, $\mathbb{u}(D) = 1 + D^5$. The two different encoders result in identical codes, but with different mappings between information sequences and codewords. Now, consider that each encoder is terminated after an information block of length $K^* = K - 4$ by appending 4 bits to return the encoders to the $\mathbb{0}$ state. (Note that for the feedback encoder, the K^* termination bits depend on the information block and are in general nonzero.) In this case we obtain an $(N, K^*) = (2K, K - 4)$ block code with rate $R_t = (K - 4)/2K \approx 1/2$ for large K. This block code contains exactly $K - 5$ weight-6 codewords, because the information sequence that produces the weight-6 codeword can begin at any of the first $K - 5$ information positions and generate the same codeword. A similar analysis reveals

TABLE 16.1: Weight spectra of two (32, 12) codes.

(a) Terminated convolutional		(b) Parallel concatenated	
Weight	Multiplicity	Weight	Multiplicity
0	1	0	1
1	0	1	0
2	0	2	0
3	0	3	0
4	0	4	0
5	0	5	1
6	11	6	4
7	12	7	8
8	23	8	16
9	38	9	30
10	61	10	73
11	126	11	144
12	200	12	210
13	332	13	308
14	425	14	404
15	502	15	496
16	545	16	571
17	520	17	558
18	491	18	478
19	346	19	352
20	212	20	222
21	132	21	123
22	68	22	64
23	38	23	24
24	11	24	4
25	2	25	4
26	0	26	1
27	0	27	0
28	0	28	0
29	0	29	0
30	0	30	0
31	0	31	0
32	0	32	0

that for weight-7 and other low weights, the number of codewords is also large, on the order of K or larger. In Table 16.1(a), we give the complete weight spectrum of the (32, 12) code that results from choosing $K = 16$. Observe that the number of codewords at each weight grows rapidly until it reaches a peak at length 16, half the block length. In other words, the weight spectrum of the code is dense at the low end, and this results in a relatively high probability of error at low SNRs, even with ML decoding.

In general, if an unterminated convolutional code has A_d codewords of weight d caused by a set of information sequences $\{\mathbf{u}(D)\}$ whose first one occurs at time unit $l = 0$, then it also has A_d codewords of weight d caused by the set of information sequences $\{D\mathbf{u}(D)\}$, and so on. Terminated convolutional codes have essentially the same property. In other words, convolutional encoders are *time-invariant*, and it is this property that accounts for the relatively large numbers of low-weight codewords in terminated convolutional codes.

Next, we look at an example in which a pseudorandom interleaver is used to produce a parallel concatenation of two identical systematic feedback convolutional encoders.

EXAMPLE 16.2 Weight Spectrum of a Parallel Concatenated Code

Consider the systematic feedback convolutional encoder of (16.6), a length $K = 16$ input sequence, and the size-16 interleaving pattern given by the permutation

$$\prod_{16} = [0, 8, 15, 9, 4, 7, 11, 5, 1, 3, 14, 6, 13, 12, 10, 2]. \tag{16.7}$$

The input sequence is first encoded by the parity generator $(1 + D^4)/(1 + D + D^2 + D^3 + D^4)$, producing the parity sequence $\mathbf{v}^{(1)}(D)$. Then, the interleaver takes the 12 information bits plus the 4 termination bits and reorders them such that

$$u'_0 = u_0, \quad u'_1 = u_8, \quad u'_2 = u_{15}, \quad \cdots, \quad u'_{15} = u_2. \tag{16.8}$$

This permuted input sequence is then reencoded using the same parity generator $(1 + D^4)/(1 + D + D^2 + D^3 + D^4)$, thus producing another version of the parity sequence. To compare with the code of Example 16.1, we now puncture alternate bits from the two versions of the parity sequence using the period $T = 2$ puncturing matrix:

$$P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{16.9}$$

The result is a parallel concatenated code with the same dimensions as the code in Example 16.1, that is, a (32, 12) code. The weight spectrum for this parallel concatenated code is given in Table 16.1(b). We see that there is a noticeable difference between this weight spectrum and the one for the terminated convolutional code shown in Table 16.1(a), even though the code generators are the same. This altered weight spectrum is a direct result of the interleaver that permutes the bits for reencoding. Note that the free distance has decreased, from 6 to 5, but that there is only one weight-5 codeword. More importantly, the multiplicities of the weight-6 through weight-9 codewords are less for the parallel concatenated code than for the terminated convolutional code. In other words, in the parallel concatenated case, there has been a shift from lower-weight codewords to higher-weight codewords relative to the convolutional code. This shifting of low-weight codewords toward higher weights in the parallel concatenation of feedback convolutional encoders has

been termed *spectral thinning* [13] and results when interleaving causes almost all the low-weight parity sequences in the first constituent code to be matched with high-weight parity sequences in the second constituent code. For example, consider the weight-2 input sequence $\mathbf{u} = (1000010\cdots0)$ that causes the low-weight parity sequence $\mathbf{v}^{(1)} = (11001100\cdots0)$. Thus, without the interleaver, the terminated convolutional code produces a codeword of weight 6. The interleaved input sequence is given by $\mathbf{u}' = (100000010\cdots0)$ and produces the high-weight parity sequence $\mathbf{v}^{(2)} = (1100101111000110)$. Combining $\mathbf{v}^{(1)}$ and $\mathbf{v}^{(2)}$ and then puncturing alternate bits using the period $T = 2$ puncturing matrix given in (16.9) produces the parity sequence (1100100101000100). Thus, the same weight-2 input sequence produces a codeword of weight 8 in the parallel concatenated code. This behavior is typical of parallel concatenation; that is, when feedback constituent encoders are used, most low-weight codewords are shifted to higher weights. In the next example we see that this spectral thinning becomes more dramatic for larger block lengths.

EXAMPLE 16.3 Spectral Thinning

Consider the same systematic feedback convolutional encoder as in Examples 16.1 and 16.2 but with a block length of $K = 32$, including the $\nu = 4$ termination bits. The weight spectrum of the terminated (64, 28) convolutional code is shown in Table 16.2(a). Now, consider the same encoder in a parallel concatenation scheme, using the size-32 interleaving pattern given by

$$\prod_{32} = [0, 16, 7, 17, 12, 28, 19, 2, 8, 11, 22, 9, 4, 20, 18, 26, 1, 3, 14, 6, 13, 31, 10, 29, 25, 24, 15, 30, 5, 23, 27, 21] \tag{16.10}$$

and the same period $T = 2$ puncturing matrix as in Example 16.2. The result is the (64, 28) parallel concatenated code whose weight spectrum is given in Table 16.2(b). We note that there is now a more pronounced difference between this weight spectrum and the one for the terminated convolutional code given in Table 16.2(a) than in the $K = 16$ case. The free distance of both codes is 6; that is, there is no change in d_{free}, but the multiplicities of the low-weight codewords are greatly reduced in the parallel concatenated code. This result can be seen more clearly in the plots of the two weight spectra shown in Figure 16.4. In other words, the effect of spectral thinning becomes more dramatic as the block length (interleaver size) K increases. In fact, for even larger values of K, the codeword and bit multiplicities of the low-weight codewords in the turbo code weight spectrum are reduced by roughly a factor of K, the interleaver size, compared with the terminated convolutional code. This reduction by a factor of K in the low-weight multiplicities is referred to as the *interleaver gain* [11] and will be verified analytically in the next section.

Several remarks can be made regarding Examples 16.1–16.3:

- Different interleavers and puncturing matrices would produce different results, but the behavior observed is typical in most cases.

TABLE 16.2: Weight spectra of two (64, 28) codes.

(a) Terminated convolutional				(b) Parallel concatenated			
Weight	Multiplicity	Weight	Multiplicity	Weight	Multiplicity	Weight	Multiplicity
0	1	33	25431436	0	1	33	25716960
1	0	34	23509909	1	0	34	23669905
2	0	35	20436392	2	0	35	20464409
3	0	36	16674749	3	0	36	16623260
4	0	37	12757248	4	0	37	12662360
5	0	38	9168248	5	0	38	9024333
6	27	39	6179244	6	6	39	6012086
7	28	40	3888210	7	9	40	3729485
8	71	41	2271250	8	15	41	2156481
9	118	42	1226350	9	9	42	1160459
10	253	43	615942	10	80	43	573214
11	558	44	287487	11	119	44	262676
12	1372	45	124728	12	484	45	110369
13	3028	46	50466	13	1027	46	42264
14	6573	47	19092	14	3007	47	15269
15	14036	48	6888	15	6852	48	4556
16	29171	49	2172	16	17408	49	1416
17	60664	50	642	17	40616	50	335
18	122093	51	140	18	90244	51	103
19	240636	52	35	19	193196	52	20
20	457660	53	6	20	390392	53	5
21	838810	54	2	21	754819	54	1
22	1476615	55	0	22	1368864	55	0
23	2484952	56	0	23	2367949	56	0
24	3991923	57	0	24	3874836	57	0
25	6098296	58	0	25	5988326	58	0
26	8845265	59	0	26	8778945	59	0
27	12167068	60	0	27	12149055	60	0
28	15844169	61	0	28	15907872	61	0
29	19504724	62	0	29	19684668	62	0
30	22702421	63	0	30	22978613	63	0
31	24967160	64	0	31	25318411	64	0
32	25927128			32	26289667		

- Spectral thinning has little effect on the minimum free distance, but it greatly reduces the multiplicities of the low-weight codewords.
- As the block length and corresponding interleaver size K increase, the weight spectrum of parallel concatenated convolutional codes begins to approximate a randomlike distribution, that is, the distribution that would result if each bit in every codeword were selected randomly from an independent and identically distributed probability distribution.
- There is only a small spectral thinning effect if feedforward constituent encoders are used, as will be seen in the next section.
- One can explain the superiority of feedback encoders in parallel concatenation as a consequence of their being IIR, rather than FIR, filters, that is, their response to single input 1's is not localized to the constraint length of the code but extends over the entire block length. This property of feedback encoders is exploited by a pseudorandom interleaver to produce the spectral thinning effect.

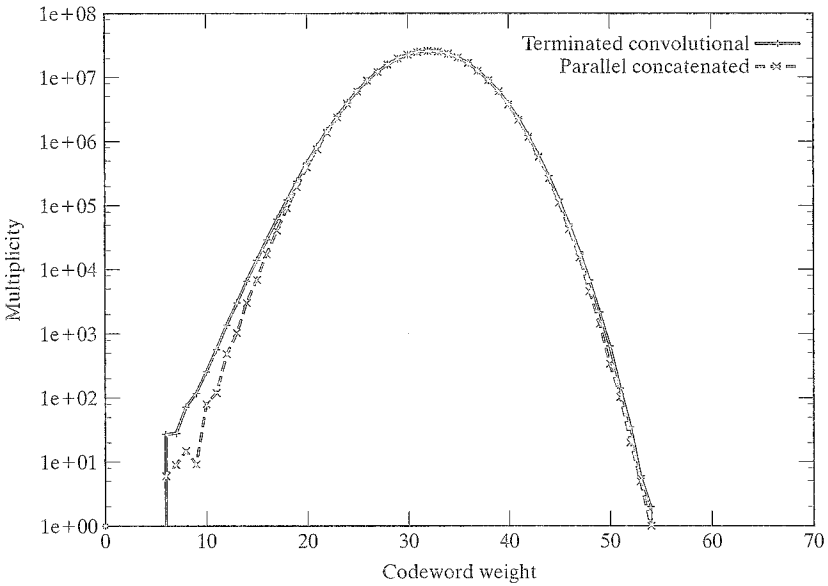


FIGURE 16.4: An illustration of spectral thinning.

It is also worth noting that parallel concatenated codes are no longer time-invariant. This can easily be seen by considering the effect when the input sequence in Example 16.2 is delayed by one time unit; that is, consider the input sequence $\mathbb{u} = [0100001000000000]$. The first parity sequence $\mathbb{v}^{(1)} = [0110011000000000]$ is also delayed by one time unit, but the interleaved input sequence is now $\mathbb{u}' = [0000000010010000]$, which produces the second parity sequence $\mathbb{v}^{(2)} = [0000000011010011]$. This is clearly not a delayed version of the $\mathbb{v}^{(2)}$ in Example 16.2. In other words, the interleaver has broken the time-invariant property of the code, resulting in a *time-varying* code. To summarize, to achieve the spectral thinning effect of parallel concatenation, it is necessary both to generate a time-varying code (via interleaving) and to employ feedback, that is, IIR, encoders.

It is clear from the preceding examples that the interleaver plays a key role in turbo coding. As we shall now briefly discuss, it is important that the interleaver has pseudorandom properties. Traditional block or convolutional interleavers do not work well in turbo coding, particularly when the block length is large. What is important is that the low-weight parity sequences from the first encoder get matched with high-weight parity sequences from the second encoder almost all the time. This requires that the interleaver break the patterns in the input sequences that produce low-weight parity sequences. Interleavers with structure, such as block or convolutional interleavers, tend to preserve too many of these "bad" input patterns, resulting in poor matching properties and limited spectral thinning. Pseudorandom interleavers, on the other hand, break up almost all the bad patterns and thus achieve the full effect of spectral thinning. In Example 16.2, the 11 input sequences

$$\mathbb{u}(D) = D^l\left(1 + D^5\right), \quad l = 0, 1, \cdots, 10, \tag{16.11}$$

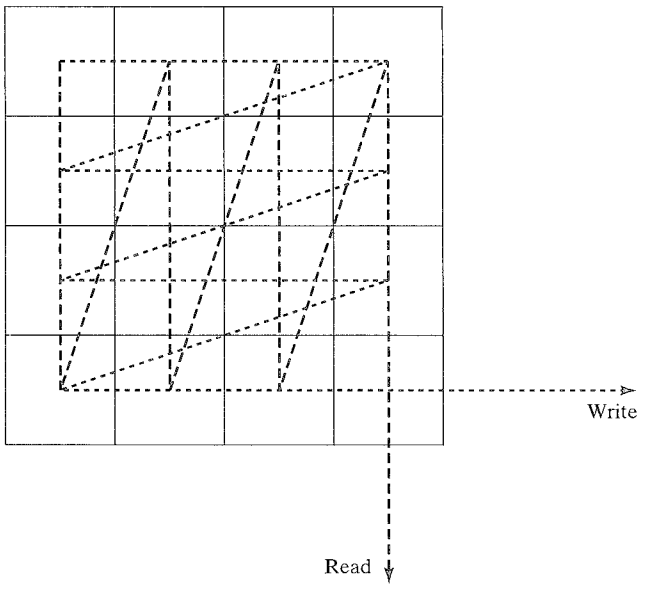


FIGURE 16.5: A (4×4) block interleaver for a $(32, 12)$ turbo code.

are bad because they generate a low-weight (4 in this case) parity sequence. As can be seen from a careful examination of Figure 16.5, if a 4×4 block (row–column) interleaver is employed, 9 of these 11 sequences will maintain the same bad pattern after interleaving, resulting in a large multiplicity of low-weight codewords. The weight spectrum of the code in Example 16.2 with this block interleaver, shown in Table 16.3, is clearly inferior to the weight spectrum shown in Table 16.1(b) obtained using the interleaver of (16.7). Pseudorandom interleavers, such as those in (16.7) and (16.10), generate a weight spectrum that has many of the same characteristics as the binomial distribution, which is equivalent to the weight spectrum assumed by Shannon in his random-coding proof of the noisy-channel coding theorem. In other words, codes with random (binomial) weight distributions can achieve the performance guaranteed by Shannon's bound. Turbo coding with pseudorandom interleaving results in a way of constructing codes with weight spectra similar to a binomial distribution, and a simple, near-optimal iterative decoding method exists.

Pseudorandom interleaving patterns can be generated in many ways, for example, by using a primitive polynomial to generate a maximum-length shift-register sequence whose cycle structure determines the permutation. Another method uses a computationally simple algorithm based on the quadratic congruence

$$c_m \equiv \frac{km(m+1)}{2} \pmod{K}, \quad 0 \leq m < K, \tag{16.12}$$

to generate an *index mapping function* $c_m \rightarrow c_{m+1} \pmod{K}$, where K is the interleaver size, and k is an odd integer. For example, for $K = 16$ and $k = 1$, we obtain

$$(c_0, c_1, \cdots, c_{15}) = (0, 1, 3, 6, 10, 15, 5, 12, 4, 13, 7, 2, 14, 11, 9, 8), \tag{16.13}$$

TABLE 16.3: The weight spectrum of a block interleaved (32, 12) turbo code.

Weight	Multiplicity
0	1
1	0
2	0
3	0
4	0
5	0
6	21
7	6
8	13
9	40
10	67
11	154
12	190
13	308
14	411
15	486
16	555
17	532
18	493
19	350
20	230
21	140
22	64
23	28
24	3
25	4
26	0
27	0
28	0
29	0
30	0
31	0
32	0

which implies that index 0 (input bit u'_0) in the interleaved sequence $\mathbf{u}'$ is mapped into index 1 in the original sequence $\mathbf{u}$ (i.e., $u'_0 = u_1$), index 1 in $\mathbf{u}'$ is mapped into index 3 in $\mathbf{u}$ ($u'_1 = u_3$), and so on, resulting in the permutation

$$\prod_{16} = [1, 3, 14, 6, 13, 12, 10, 2, 0, 8, 15, 9, 4, 7, 11, 5]. \tag{16.14}$$

If this interleaving pattern is shifted cyclically to the right $r = 8$ times, we obtain the interleaving pattern of (16.7). For K a power of 2, it can be shown that these *quadratic interleavers* have statistical properties similar to those of randomly chosen interleavers, and thus they give good performance when used in turbo coding [32]. Other good interleaving patterns can be generated by varying k and r, and the special case $r = K/2$ (used to obtain (16.7)) results in an interleaver that simply interchanges pairs of indices (see Problem 16.2). This special case is particularly interesting in terms of implementation, since the interleaving and deinterleaving functions (both used in decoding) are identical. Finally, when K is not a power of 2, the foregoing algorithm can be modified to generate similar permutations with good statistical properties.

The basic structure of an iterative turbo decoder is shown in Figure 16.6. (We assume here a rate $R = 1/3$ parallel concatenated code without puncturing.) It employs two SISO decoders using the MAP algorithm presented earlier in Chapter 12. At each time unit l, three output values are received from the channel, one for the information bit $u_l = v_l^{(0)}$, denoted by $r_l^{(0)}$, and two for the parity bits $v_l^{(1)}$ and $v_l^{(2)}$, denoted by $r_l^{(1)}$ and $r_l^{(2)}$, and the $3K$-dimensional received vector is denoted by

$$\mathbb{r} = \left(r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \cdots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)}\right). \tag{16.15}$$

Now, let each transmitted bit be represented using the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$. Then, for an AWGN channel with unquantized (soft) outputs, we define the *log-likelihood ratio* (L-value) $L\left(v_l^{(0)} \mid r_l^{(0)}\right) = L\left(u_l \mid r_l^{(0)}\right)$ (before decoding) of a transmitted information bit u_l given the received value $r_l^{(0)}$ as

$$\begin{aligned} L(u_l \mid r_l^{(0)}) &= \ln \frac{P\left(u_l = +1 \mid r_l^{(0)}\right)}{P\left(u_l = -1 \mid r_l^{(0)}\right)} \\ &= \ln \frac{P\left(r_l^{(0)} \mid u_l = +1\right) P\left(u_l = +1\right)}{P\left(r_l^{(0)} \mid u_l = -1\right) P\left(u_l = -1\right)} \end{aligned}$$

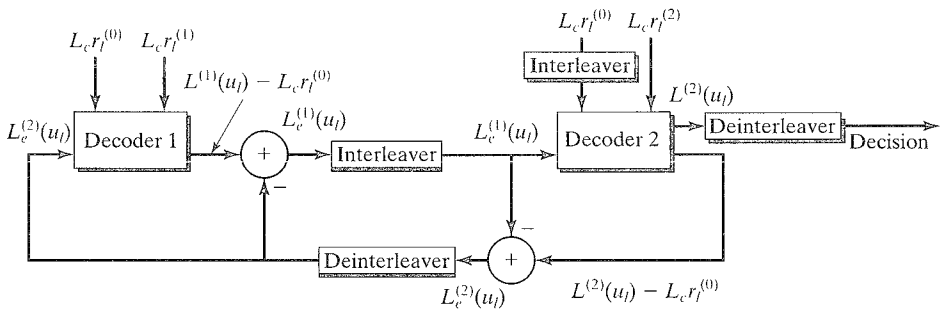


FIGURE 16.6: Basic structure of an iterative turbo decoder.

$$= \ln \frac{P\left(r_l^{(0)} \mid u_l = +1\right)}{P\left(r_l^{(0)} \mid u_l = -1\right)} + \ln \frac{P(u_l = +1)}{P(u_l = -1)}$$

$$= \ln \frac{e^{-(E_s/N_0)\left(r_l^{(0)}-1\right)^2}}{e^{-(E_s/N_0)\left(r_l^{(0)}+1\right)^2}} + \ln \frac{P(u_l = +1)}{P(u_l = -1)}, \tag{16.16}$$

where E_s/N_0 is the channel SNR, and u_l and $r_l^{(0)}$ have both been normalized by a factor of $\sqrt{E_s}$. This equation simplifies to

$$\begin{aligned} L\left(u_l \mid r_l^{(0)}\right) &= -\frac{E_s}{N_0}\left\{\left(r_l^{(0)} - 1\right)^2 - \left(r_l^{(0)} + 1\right)^2\right\} + \ln \frac{P(u_l = +1)}{P(u_l = -1)} \\ &= 4\frac{E_s}{N_0} r_l^{(0)} + \ln \frac{P(u_l = +1)}{P(u_l = -1)} \\ &= L_c r_l^{(0)} + L_a(u_l), \end{aligned} \tag{16.17}$$

where $L_c = 4(E_s/N_0)$ is the *channel reliability factor* (see Section 12.5), and $L_a(u_l)$ is the a priori *L-value* of the bit u_l. In the case of a transmitted parity bit $v_l^{(j)}$, given the received value $r_l^{(j)}$, $j = 1, 2$, the L-value (before decoding) is given by

$$L\left(v_l^{(j)} \mid r_l^{(j)}\right) = L_c r_l^{(j)} + L_a\left(v_l^{(j)}\right) = L_c r_l^{(j)}, \quad j = 1, 2, \tag{16.18}$$

since in a linear code with equally likely information bits, the parity bits are also equally likely to be $+1$ or -1, and thus the a priori L-values of the parity bits are 0; that is,

$$L_a\left(v_l^{(j)}\right) = \ln \frac{P\left(v_l^{(j)} = +1\right)}{P\left(v_l^{(j)} = -1\right)} = 0, \quad j = 1, 2. \tag{16.19}$$

(We note here that $L_a(u_l)$ also equals 0 for the first iteration of decoder 1 but that thereafter the a priori L-values of the information bits are replaced by *extrinsic* L-values from the other decoder, as will now be explained.)

The received soft channel L-values $L_c r_l^{(0)}$ for u_l and $L_c r_l^{(1)}$ for $v_l^{(1)}$ enter decoder 1, and the (properly interleaved) received soft channel L-values $L_c r_l^{(0)}$ for u_l and the received soft channel L-values $L_c r_l^{(2)}$ for $v_l^{(2)}$ enter decoder 2. The output of decoder 1 contains two terms:

1. $L^{(1)}(u_l) = \ln\left[P\left(u_l = +1 \,/\, \mathbb{r}_1, \mathbb{L}_a^{(1)}\right) \Big/ P\left(u_l = -1 \mid \mathbb{r}_1, \mathbb{L}_a^{(1)}\right)\right]$, the a posteriori L-value (after decoding) of each information bit produced by decoder 1 given the (partial) received vector $\mathbb{r}_1 \triangleq \left[r_0^{(0)} r_0^{(1)}, r_1^{(0)} r_1^{(1)}, \cdots, r_{K-1}^{(0)} r_{K-1}^{(1)}\right]$ and the a priori input vector $\mathbb{L}_a^{(1)} \triangleq \left[L_a^{(1)}(u_0), L_a^{(1)}(u_1), \cdots, L_a^{(1)}(u_{K-1})\right]$ for decoder 1, and
2. $L_e^{(1)}(u_l) = L^{(1)}(u_l) - \left[L_c r_l^{(0)} + L_e^{(2)}(u_l)\right]$, the extrinsic a posteriori L-value (after decoding) associated with each information bit produced by decoder 1,

which, after interleaving, is passed to the input of decoder 2 as the a priori value $L_a^{(2)}(u_l)$.

Subtracting the term in brackets, namely, $L_c r_l^{(0)} + L_e^{(2)}(u_l)$, removes the effect of the current information bit u_l from $L^{(1)}(u_l)$, leaving only the effect of the parity constraints, thus providing an independent estimate of the information bit u_l to decoder 2 in addition to the received soft channel L-values at time l. Similarly, the output of decoder 2 contains two terms:

1. $L^{(2)}(u_l) = \ln\left[P\left(u_l = +1 \,/\, \mathbb{r}_2, \mathbb{L}_a^{(2)}\right) \Big/ P\left(u_l = -1 \mid \mathbb{r}_2, \mathbb{L}_a^{(2)}\right)\right]$, where $\mathbb{r}_2$ is the (partial) received vector and $\mathbb{L}_a^{(2)}$ the a priori input vector for decoder 2, and
2. $L_e^{(2)}(u_l) = L^{(2)}(u_l) - \left[L_c r_l^{(0)} + L_e^{(1)}(u_l)\right]$, and the extrinsic a posteriori L-values $L_e^{(2)}(u_l)$ produced by decoder 2, after deinterleaving, are passed back to the input of decoder 1 as the a priori values $L_a^{(1)}(u_l)$.

Thus, the input to each decoder contains three terms, the soft channel L-values $L_c r_l^{(0)}$ and $L_c r_l^{(1)}$ (or $L_c r_l^{(2)}$) and the extrinsic a posteriori L-values $L_e^{(2)}(u_l) = L_a^{(1)}(u_l)$ (or $L_e^{(1)}(u_l) = L_a^{(2)}(u_l)$) passed from the other decoder. (In the initial iteration of decoder 1, the extrinsic a posteriori L-values $L_e^{(2)}(u_l) = L_a^{(1)}(u_l)$ are just the original a priori L-values $L_a(u_l)$, which, as noted before, are all equal to 0 for equally likely information bits. Thus the extrinsic L-values passed from one decoder to the other during the iterative decoding process are treated like new sets of a priori probabilities by the MAP algorithm.) Decoding then proceeds iteratively, with each decoder passing its respective extrinsic L-values back to the other decoder. This results in a *turbo* or *bootstrapping* effect in which each estimate becomes successively more reliable. After a sufficient number of iterations, the decoded information bits are determined from the a posteriori L-values $L^{(2)}(u_l)$, $l = 0, 1, \cdots, K - \nu - 1$, at the output of decoder 2. Because the decoded output is taken only after the final iteration, it is more accurate to refer to the SISO constituent decoders as a posteriori probability (APP) estimators rather than MAP decoders, since their outputs are extrinsic a posteriori L-values that are passed to their companion decoder for more processing. A more complete discussion of iterative turbo decoding is given in Section 16.5.

Although it is true, as stated previously, that the extrinsic a posteriori L-values $L_e(u_l)$ passed between decoders during the first iteration of decoding are independent of u_l, this is not so for subsequent iterations. Thus, the extrinsic information becomes less helpful in obtaining successively more reliable estimates of the information bits as the iterations continue. Eventually, a point is reached at which no further improvement is possible, the iterations are stopped, and the final decoding estimate is produced. Methods for determining when to stop the iterations, known as *stopping rules*, are discussed in Section 16.5.

It is worth pointing out here that the term *turbo* in turbo coding has more to do with decoding than encoding. Indeed, it is the successive feedback of extrinsic information from the SISO decoders in the iterative decoding process that mimics the feedback of exhaust gases in a turbocharged engine.

Finally, before moving on to a more detailed discussion of turbo coding, we note that many of its features are similar to those observed for low-density parity-check (LDPC) codes, to be discussed in Chapter 17. Both encoding schemes produce codes with randomlike weight distributions and a thin weight spectrum. Both decoding methods make use of APP likelihoods in an iterative process, and they both employ the concept of extrinsic information. In fact, it was the discovery of turbo coding in 1993 that led to a rediscovery of the merits of LDPC codes, which had been largely neglected by the research community for more than 30 years.

16.2 DISTANCE PROPERTIES OF TURBO CODES

As illustrated in Examples 16.1 through 16.3, the fundamental property of turbo codes that allows them to achieve such excellent performance is the randomlike weight spectrum, or spectral thinning, produced by the pseudorandom interleaver. In this section we examine the weight spectrum of turbo codes in more detail. In particular, we consider a series of examples for parallel concatenated codes (PCCs), including both parallel concatenated block codes (PCBCs) and parallel concatenated convolutional codes (PCCCs).

As noted in the remarks following Example 16.3, the exact weight spectrum of a turbo code depends on the particular interleaver chosen. Thus, in order to avoid exhaustively searching all possible interleavers for the best weight spectrum for a specific PCC, we introduce the concept of a *uniform interleaver* [11].

DEFINITION 16.1 A uniform interleaver of length K is a probabilistic device that maps a given input block of weight w into all its distinct $\binom{K}{w}$ permutations with equal probability $1 \Big/ \binom{K}{w}$.

Using the notion of a uniform interleaver allows us to calculate the average (over all possible interleavers) weight spectrum of a PCC. This average weight spectrum is typical of the weight spectrum obtained for a randomly chosen interleaver.

EXAMPLE 16.4 A Parallel Concatenated Block Code

Consider the (7, 4, 3) Hamming code in systematic form. The weight enumerating function (WEF) for this code is given by

$$A(X) = 7X^3 + 7X^4 + X^7; \tag{16.20}$$

that is, in addition to the all-zero codeword, the code contains 7 codewords of weight 3, 7 codewords of weight 4, and the all-one codeword of weight 7. The complete list of 16 codewords is shown in Table 16.4. Splitting the contributions of the information and parity bits gives the input redundancy weight enumerating function (IRWEF)

$$A(W, Z) = W(3Z^2 + Z^3) + W^2(3Z + 3Z^2) + W^3(1 + 3Z) + W^4Z^3. \tag{16.21}$$

In other words, there are 3 codewords with information weight 1 and parity weight 2, 1 codeword with information weight 1 and parity weight 3, 3 codewords with

TABLE 16.4: Codeword list for the (7, 4, 3) Hamming code.

Information				Parity		
0	0	0	0	0	0	0
1	0	0	0	1	0	1
0	1	0	0	1	1	0
1	1	0	0	0	1	1
0	0	1	0	0	1	1
1	0	1	0	1	1	0
0	1	1	0	1	0	1
1	1	1	0	0	0	0
0	0	0	1	1	1	1
1	0	0	1	0	1	0
0	1	0	1	0	0	1
1	1	0	1	1	0	0
0	0	1	1	1	0	0
1	0	1	1	0	0	1
0	1	1	1	0	1	0
1	1	1	1	1	1	1

information weight 2 and parity weight 1, and so on. Finally, the conditional weight enumerating function (CWEF) for each input weight is given by

$$\begin{aligned} A_1(Z) &= 3Z^2 + Z^3, \\ A_2(Z) &= 3Z + 3Z^2, \\ A_3(Z) &= 1 + 3Z, \\ A_4(Z) &= Z^3. \end{aligned} \tag{16.22}$$

Now, we examine how the uniform interleaver concept can be used to compute the average IRWEF for PCCs. First, consider the general case shown in Figure 16.7 of a PCBC with two different (n, k) systematic block constituent codes, C_1 and C_2, with CWEFs given by $A_w^{C_1}(Z)$ and $A_w^{C_2}(Z)$, $w = 1, \cdots, k$, connected by a uniform interleaver of size $K = k$. The original information block and both parity blocks are transmitted, resulting in a $(2n - k, k)$ PCBC. Assume that a particular input block of weight w enters the first encoder, thereby generating one of the parity weights in $A_w^{C_1}(Z)$. From the definition of the uniform interleaver, it follows that there is an equal probability that the second encoder will match that parity weight with any of the parity weights in $A_w^{C_2}(Z)$. Thus, the average CWEF of the PCBC is given by

$$A_w^{PC}(Z) = \frac{A_w^{C_1}(Z) A_w^{C_2}(Z)}{\binom{K}{w}}, \tag{16.23}$$

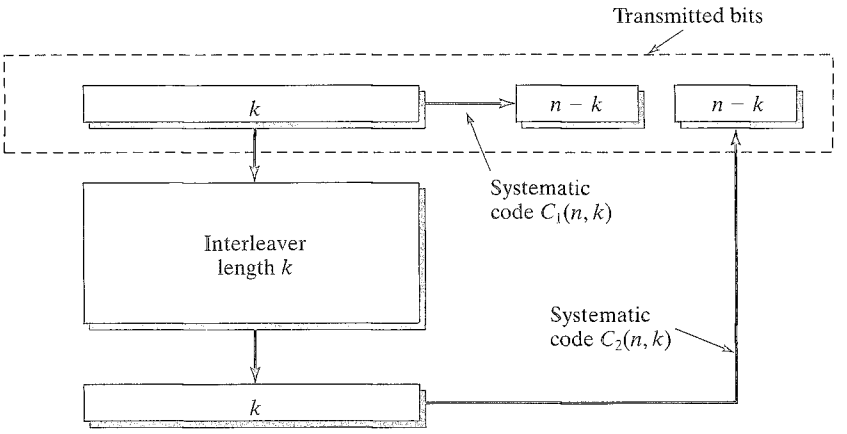


FIGURE 16.7: A parallel concatenated block code.

and the average IRWEF is given by

$$A^{PC}(W, Z) = \sum_{w,z} A_{w,z} W^w Z^z = \sum_{(1 \le w \le K)} W^w A_w^{PC}(Z). \tag{16.24}$$

Also, we can express the *bit* CWEF and the *bit* IRWEF as follows:

$$B_w^{PC}(Z) = \frac{w}{K} A_w^{PC}(Z) \tag{16.25a}$$

and

$$B^{PC}(W, Z) = \sum_{w,z} B_{w,z} W^w Z^z = \sum_{(1 \le w \le K)} W^w B_w^{PC}(Z), \tag{16.25b}$$

where $B_{w,z} = (w/K) A_{w,z}$. Finally, the average codeword and bit WEFs are given by

$$\begin{aligned} A^{PC}(X) &= \sum_{(d_{min} \le d)} A_d X^d = A^{PC}(W, Z)\Big|_{W=Z=X} \\ &= \sum_{(1 \le w \le K)} W^w A_w^{PC}(Z)\Big|_{W=Z=X} \end{aligned} \tag{16.26a}$$

and

$$\begin{aligned} B^{PC}(X) &= \sum_{(d_{min} \le d)} B_d X^d = B^{PC}(W, Z)\Big|_{W=Z=X} \\ &= \sum_{(1 \le w \le K)} W^w B_w^{PC}(Z)\Big|_{W=Z=X}. \end{aligned} \tag{16.26b}$$

The following remarks apply to the use of the uniform interleaver concept to calculate WEFs for PCBCs.

- The codeword CWEF $A_w^{PC}(Z)$, IRWEF $A^{PC}(W, Z)$, and WEF $A^{PC}(X)$ of the PCBC are all average quantities over the entire class of uniform interleavers, and thus their coefficients may not be integers, unlike the case in which these quantities are computed for the constituent codes individually.

- Equations (16.26) represents two different ways of expressing the codeword and bit WEFs of a PCBC, one as a sum over the codeword weights d that is valid for both systematic and nonsystematic codes and the other as a sum over input weights w that applies only to systematic codes.

- The codeword and bit WEF expressions in (16.26a) and (16.26b) are general and apply to all systematic codes. Expressions similar to the sums over input weights w that are valid for nonsystematic codes are given by

$$A(X) = A(W, X)|_{W=1} = \sum_{(1 \le w \le K)} W^w A_w(X)\big|_{W=1} \tag{16.26c}$$

and

$$B(X) = B(W, X)|_{W=1} = \sum_{(1 \le w \le K)} W^w B_w(X)\big|_{W=1}, \tag{16.26d}$$

where $A(W, X)$ and $B(W, X)$ are the input–output weight enumerating functions (IOWEFs), and $A_w(X)$ and $B_w(X)$ are the conditional weight enumerating functions (CWEFs) of the nonsystematic code.

- A more general class of PCBCs results if the two constituent codes, C_1 and C_2, have different block lengths, n_1 and n_2.

EXAMPLE 16.4 (Continued)

For the (10, 4) PCBC with two identical (7, 4, 3) Hamming codes as constituent codes, the CWEFs are

$$\begin{aligned}
A_1^{PC}(Z) &= \frac{(3Z^2 + Z^3)^2}{4} = 2.25Z^4 + 1.5Z^5 + 0.25Z^6, \\
A_2^{PC}(Z) &= \frac{(3Z + 3Z^2)^2}{6} = 1.5Z^2 + 3Z^3 + 1.5Z^4, \\
A_3^{PC}(Z) &= \frac{(1 + 3Z)^2}{4} = 0.25 + 1.5Z + 2.25Z^2, \\
A_4^{PC}(Z) &= \frac{(Z^3)^2}{1} = Z^6,
\end{aligned} \tag{16.27}$$

the IRWEFs are

$$\begin{aligned}
A^{PC}(W, Z) = W\left(2.25Z^4 + 1.5Z^5 + 0.25Z^6\right) + W^2\left(1.5Z^2 + 3Z^3 + 1.5Z^4\right) + \\
W^3\left(0.25 + 1.5Z + 2.25Z^2\right) + W^4Z^6
\end{aligned} \tag{16.28a}$$

and

$$B^{PC}(W, Z) = W\left(0.56Z^4 + 0.37Z^5 + 0.06Z^6\right) + W^2\left(0.75Z^2 + 1.5Z^3 + 0.75Z^4\right) + W^3\left(0.19 + 1.12Z + 1.69Z^2\right) + W^4Z^6; \quad (16.28b)$$

and the WEFs are

$$A^{PC}(X) = 0.25X^3 + 3X^4 + 7.5X^5 + 3X^6 + 0.25X^7 + X^{10} \quad (16.28c)$$

and

$$B^{PC}(X) = 0.19X^3 + 1.87X^4 + 3.75X^5 + 1.12X^6 + 0.06X^7 + X^{10}. \quad (16.28d)$$

We now make the following observations regarding Example 16.4:

- The coefficients of the codeword WEF's for the PCBC are fractional, owing to the effect of averaging.
- The sum of all the coefficients in $A^{PC}(X)$ equals 15, the total number of nonzero codewords in the PCBC.
- The minimum distance of the (10, 4) PCBC is almost certainly either 3 or 4, depending on the particular interleaver chosen.
- In this example, a detailed analysis (see [11]) of the $4! = 24$ possible interleaving patterns reveals that exactly 6 result in a minimum distance of 3, and the other 18 result in a minimum distance of 4.
- The *average codeword multiplicities* of the low-weight codewords are given by $A_3 = 0.25$ and $A_4 = 3.0$.
- The *average bit multiplicities* of the low-weight codewords are given by $B_3 = 0.19$ and $B_4 = 1.87$.

Now, we examine the more general case, illustrated in Figure 16.8, of forming the constituent codes in a PCBC by concatenating h codewords of a basic (n, k) systematic code C to form an (hn, hk) *h-repeated* code C^h and using an interleaver size of $K = hk$. Again, the original information block and both parity blocks are transmitted, resulting in a $(2hn - hk, hk)$ PCBC. In this case, the IRWEF of the code C^h is given by

$$A^{C^h}(W, Z) = \left[1 + A^C(W, Z)\right]^h - 1, \quad (16.29)$$

where we have included the 1's in (16.29) to account for the fact that a combination of all-zero codewords and nonzero codewords in the basic code C can be used to form a nonzero codeword in the h-repeated code C^h. If $A_w^{C_1^h}(Z)$ and $A_w^{C_2^h}(Z)$

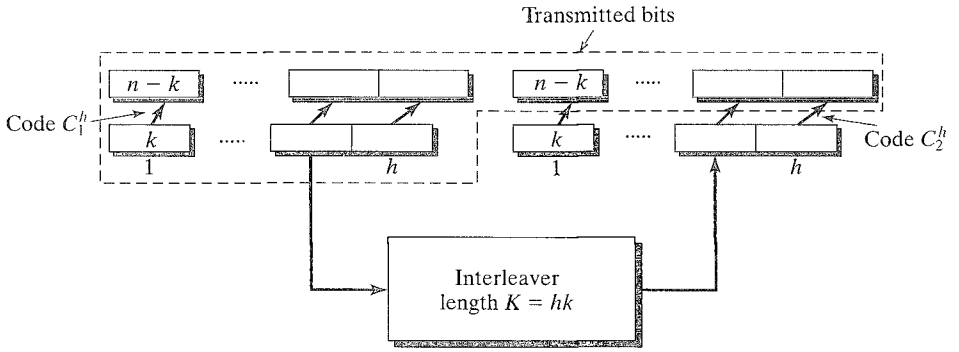


FIGURE 16.8: A PCBC with repeated block constituent codes.

represent the CWEFs of two h-repeated constituent codes C_1^h and C_2^h, then the average CWEF of the PCBC is given by

$$A_w^{PC}(Z) = \frac{A_w^{C_1^h}(Z) A_w^{C_2^h}(Z)}{\binom{K}{w}}, \tag{16.30}$$

the average IRWEFs are given by

$$A^{PC}(W, Z) = \sum_{(1 \le w \le hk)} W^w A_w^{PC}(Z) \tag{16.31a}$$

and

$$B^{PC}(W, Z) = \sum_{(1 \le w \le hk)} \frac{w}{hk} W^w A_w^{PC}(Z), \tag{16.31b}$$

and the average WEFs can be computed using (16.26).

EXAMPLE 16.5 A PCBC with Repeated Block Constituent Codes

Consider the $(20, 8)$ PCBC formed by using $h = 2$ codewords from the $(7, 4, 3)$ Hamming code, that is, a $(14, 8)$ 2-repeated Hamming code, as both constituent codes, along with a uniform interleaver of size $K = hk = 8$. The IRWEF of the $(14, 8)$ constituent code is

$$\begin{aligned} A(W, Z) &= \left[1 + W(3Z^2 + Z^3) + W^2(3Z + 3Z^2) + W^3(1 + 3Z) + W^4 Z^3\right]^2 - 1 \\ &= W(6Z^2 + 2Z^3) + W^2(6Z + 6Z^2 + 9Z^4 + 6Z^5 + Z^6) + \\ &\quad W^3(2 + 6Z + 18Z^3 + 24Z^4 + 6Z^5) + W^4(15Z^2 + 40Z^3 + 15Z^4) + \\ &\quad W^5(6Z + 24Z^2 + 18Z^3 + 6Z^5 + 2Z^6) + \\ &\quad W^6(1 + 6Z + 9Z^2 + 6Z^4 + 6Z^5) + W^7(2Z^3 + 6Z^4) + W^8 Z^6, \end{aligned} \tag{16.32}$$

and its CWEFs for each input weight are given by

$$A_1(Z) = 6Z^2 + 2Z^3, \qquad A_2(Z) = 6Z + 6Z^2 + 9Z^4 + 6Z^5 + Z^6,$$

$$A_3(Z) = 2 + 6Z + 18Z^3 + 24Z^4 + 6Z^5, \qquad A_4(Z) = 15Z^2 + 40Z^3 + 15Z^4,$$

$$A_5(Z) = 6Z + 24Z^2 + 18Z^3 + 6Z^5 + 2Z^6, \qquad A_6(Z) = 1 + 6Z + 9Z^2 + 6Z^4 + 6Z^5,$$

$$A_7(Z) = 2Z^3 + 6Z^4, \qquad A_8(Z) = Z^6. \tag{16.33}$$

Note that this code still has a minimum distance of 3; that is, it is a (14, 8, 3) code, since each of the seven weight-3 codewords in one code can be paired with the all-zero codeword in the other code, resulting in a total of 14 weight-3 codewords in the 2-repeated code. Thus, by itself, this (14, 8, 3) code would not be interesting, since it is longer (more complex) than the (7, 4, 3) code but does not have better distance properties; however, when it is used as a constituent code in a (20, 8) PCBC, the resulting code has better distance properties than the (10, 4) PCBC formed from a single Hamming code.

EXAMPLE 16.5 (Continued)

Using (16.30) we can compute the CWEFs of the (20, 8) PCBC with two identical (14, 8, 3) 2-repeated Hamming codes as constituent codes as follows:

$$A_1^{PC}(Z) = \frac{\left(6Z^2 + 2Z^3\right)^2}{8} = 4.5Z^4 + 3Z^5 + 0.5Z^6,$$

$$A_2^{PC}(Z) = \frac{\left(6Z + 6Z^2 + 9Z^4 + 6Z^5 + Z^6\right)^2}{28}$$

$$= 1.29Z^2 + 2.57Z^3 + 1.29Z^4 + 3.86Z^5 + 6.43Z^6 + 3Z^7 +$$

$$3.32Z^8 + 3.86Z^9 + 1.93Z^{10} + 0.43Z^{11} + 0.04Z^{12},$$

$$A_3^{PC}(Z) = \frac{\left(2 + 6Z + 18Z^3 + 24Z^4 + 6Z^5\right)^2}{56}$$

$$= 0.07 + 0.43Z + 0.64Z^2 + 1.29Z^3 + 5.57Z^4 + 5.57Z^5 +$$

$$7.07Z^6 + 15.43Z^7 + 14.14Z^8 + 5.14Z^9 + 0.64Z^{10},$$

$$A_4^{PC}(Z) = \frac{\left(15Z^2 + 40Z^3 + 15Z^4\right)^2}{70}$$

$$= 3.21Z^4 + 17.14Z^5 + 29.29Z^6 + 17.14Z^7 + 3.21Z^8,$$

$$A_5^{PC}(Z) = \frac{\left(6Z + 24Z^2 + 18Z^3 + 6Z^5 + 2Z^6\right)^2}{56}$$

$$= 0.64Z^2 + 5.14Z^3 + 14.14Z^4 + 15.43Z^5 + 7.07Z^6 + 5.57Z^7 +$$

$$5.57Z^8 + 1.29Z^9 + 0.64Z^{10} + 0.43Z^{11} + 0.07Z^{12},$$

$$A_6^{PC}(Z) = \frac{\left(1 + 6Z + 9Z^2 + 6Z^4 + 6Z^5\right)^2}{28}$$
$$= 0.04 + 0.43Z + 1.93Z^2 + 3.86Z^3 + 3.32Z^4 + 3Z^5 +$$
$$6.43Z^6 + 3.86Z^7 + 1.29Z^8 + 2.57Z^9 + 1.29Z^{10},$$
$$A_7^{PC}(Z) = \frac{\left(2Z^3 + 6Z^4\right)^2}{8} = 0.5Z^6 + 3Z^7 + 4.5Z^8,$$
$$A_8^{PC}(Z) = \frac{\left(Z^6\right)^2}{1} = Z^{12}. \tag{16.34}$$

Then, we can compute the IRWEFs $A^{PC}(W, Z)$ and $B^{PC}(W, Z)$ and the WEFs $A^{PC}(X)$ and $B^{PC}(X)$ using (16.31) and (16.26), respectively (see Problem 16.4).

We now conclude our discussion of PCBCs with a few observations regarding Example 16.5.

- The minimum distance of the (20, 8) PCBC is almost certainly either 3 or 4, depending on the particular interleaver chosen, the same as for the (10, 4) PCBC.
- However, the *average codeword multiplicities* of the low-weight codewords have decreased from $A_3 = 0.25$ to $A_3 = 0.07$ and from $A_4 = 3.0$ to $A_4 = 1.72$, respectively, despite the fact that the (20, 8) PCBC contains 16 times as many codewords as the (10, 4) PCBC.
- Also, the *average bit multiplicities* of the low-weight codewords have decreased from $B_3 = 0.19$ to $B_3 = 0.03$ and from $B_4 = 1.87$ to $B_4 = 0.48$, respectively.
- This is an example of spectral thinning; that is, the multiplicities of the low-weight codewords in a PCBC are decreased by increasing the length of the constituent code and the interleaver size.
- Increasing the code length and interleaver size by further increasing the repeat factor h leads to additional spectral thinning, which results in improved performance at low SNRs, but for block constituent codes the beneficial effect of increasing h diminishes for large h.
- A better approach would be to increase the interleaver size by using longer block constituent codes, but efficient SISO decoding of the constituent codes is more difficult for large block sizes.

Now, we consider the case of PCCCs, in which the constituent codes are generated by convolutional encoders, as illustrated in Figure 16.1. An exact analysis, similar to the foregoing examples for PCBCs, is possible but is complicated by issues involving termination of the constituent encoders. Hence, we make the simplifying assumption that both constituent encoders are terminated to the all-zero state. (As noted previously, this is normally the case for the first encoder but not for the second encoder, because the required termination bits are generated by the interleaver only

with probability $1/2^{\nu}$, where ν is the constraint length of the encoder.) The resulting analysis is nearly exact whenever the interleaver size K is at least an order of magnitude larger than the constraint length ν of the constituent encoders. Because turbo codes are most effective for short constraint lengths and large interleaver sizes, this condition always holds in practice.

We begin by illustrating the procedure for computing the CWEFs $A_w(Z)$ of the equivalent block code produced by a convolutional encoder that starts in the all-zero state $S_0 = \mathbb{0}$ and returns to the all-zero state after an input sequence of length K, including termination bits. For an $(n, 1, \nu)$ convolutional encoder, the equivalent block code has dimensions $(nK, K - \nu)$. The situation here is somewhat different from that presented in Chapter 11, where we were interested in computing the WEF of all codewords that diverged from the all-zero state at a particular time and remerged only once. This WEF was the appropriate one to consider for evaluating the event- and bit-error probabilities per unit time of an encoder driven by semi-infinite (unterminated) input sequences. To evaluate the block- and bit-error probabilities of PCCCs, however, the WEF must include the effect of multiple-error events, that is, error events that diverge from and remerge with the all-zero state more than once. This is because the encoders are driven by finite-length (terminated) input sequences, resulting in an equivalent block code, and the error probability analysis must consider the entire block, rather than just a particular time unit. Thus, we will modify the *single-error event WEF* from Chapter 11 to obtain a *multiple-error event WEF* appropriate for PCCCs.

From Figure 16.9 we can see that any multiple-error event in a codeword belonging to a terminated convolutional code can be viewed as a succession of single-error events separated by sequences of 0's. We begin by considering all codewords that can be constructed from a single-error event of length $\lambda \leq K$; the error event is surrounded by $(K - \lambda)$ 0's. Because the $(K - \lambda)$ 0's are divided into two groups, one preceding and one following the error event, the number of single-error events is the number of ways of summing two nonnegative integers that add to $K - \lambda$. Thus, the multiplicity of block codewords for single-error events is given by

$$c[\lambda, 1] = \begin{pmatrix} K - \lambda + 1 \\ 1 \end{pmatrix} = K - \lambda + 1. \tag{16.35}$$

Next, consider a pair of error events with total length $\lambda \leq K$; that is, the two error events have a total of $(K - \lambda)$ 0's appearing before, after, or between them. In this case, the $(K - \lambda)$ 0's are divided into three groups, and the number of double-error events is the number of ways of summing three nonnegative integers that add to

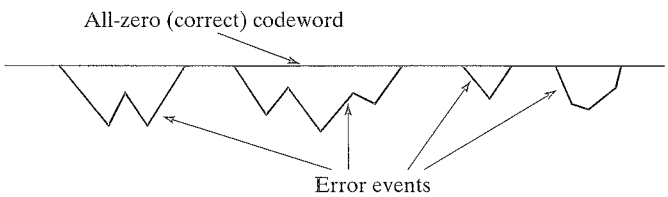


FIGURE 16.9: Multiple-error events in a terminated convolutional code.

$K - \lambda$. Thus, the multiplicity of block codewords for double-error events is given by

$$c[\lambda, 2] = \begin{pmatrix} K - \lambda + 2 \\ 2 \end{pmatrix} = \frac{(K - \lambda + 2)(K - \lambda + 1)}{2}. \tag{16.36}$$

In general, the number of h-error events with total length $\lambda \leq K$ is the number of ways of summing $h + 1$ nonnegative integers that add to $K - \lambda$, and the multiplicity of block codewords for h-error events is given by (see Problem 16.7)

$$c[\lambda, h] = \begin{pmatrix} K - \lambda + h \\ h \end{pmatrix}. \tag{16.37}$$

To proceed with the computation of the CWEFs $A_w(Z)$ of the equivalent block code produced by a convolutional encoder that terminates after an input sequence of length K, we must know the number $A^{(1)}_{w,z,\lambda}$ of single-error events of length $\lambda \leq K$ with input weight w and parity weight z, the number $A^{(2)}_{w,z,\lambda}$ of double-error events of total length $\lambda \leq K$ with input weight w and parity weight z, and so on. We can obtain the *single-error event enumerators* $A^{(1)}_{w,\lambda}(Z) = \sum_{(z)} A^{(1)}_{w,z,\lambda} Z^z$, $\lambda \leq K$, directly from the IRWEF $A(W, Z, L)$ of the unterminated convolutional code by simply dropping all terms of order larger than L^K and then collecting together all terms with input weight w and length λ. To determine the *double-error event enumerators* $A^{(2)}_{w,\lambda}(Z) = \sum_{(z)} A^{(2)}_{w,z,\lambda} Z^z$, $\lambda \leq K$, we must examine all pairs of terms in the IRWEF $A(W, Z, L)$ for which the total length of the two error events is $\lambda \leq K$. We can do this by defining the *double-error event IRWEF* $A^{(2)}(W, Z, L)$ of the unterminated convolutional code as follows:

$$A^{(2)}(W, Z, L) = [A(W, Z, L)]^2. \tag{16.38}$$

We can now obtain the double-error event enumerators $A^{(2)}_{w,\lambda}(Z)$, $\lambda \leq K$, from the double-error event IRWEF $A^{(2)}(W, Z, L)$ by dropping all terms of order larger than L^K and then collecting together all terms with input weight w and length λ. We can find higher-order error event enumerators in a similar way. Then, we can compute the CWEFs as

$$A_w(Z) = \sum_{(\lambda \leq K,\ 1 \leq h \leq h_{max})} c[\lambda, h] A^{(h)}_{w,\lambda}(Z), \tag{16.39}$$

where h_{max}, the largest possible number of error events associated with a weight w input sequence, depends on w and the code parameters.

EXAMPLE 16.6 Computing WEFs for a Terminated Convolutional Code and a PCCC

For the $(2, 1, 2)$ systematic feedback convolutional encoder whose generator matrix is given by

$$\mathbf{G}_{fb}(D) = \begin{bmatrix} 1 & (1 + D^2)/(1 + D + D^2) \end{bmatrix} \tag{16.40}$$

and whose encoder block diagram and state diagram are shown in Figure 16.10, consider the $(18, 7)$ block code obtained from a length $K = 9$ input sequence,

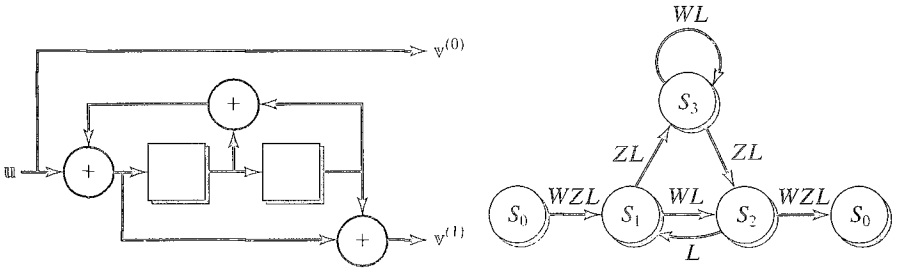


FIGURE 16.10: Encoder diagram and state diagram for Example 16.6.

including $\nu = 2$ termination bits. The IRWEF of this encoder is given by (see Example 11.14 in Chapter 11)

$$\begin{aligned} A(W, Z, L) = {} & L^3W^3Z^2 + L^4W^2Z^4 + L^5(W^3Z^4 + W^4Z^2) + \\ & L^6(2W^3Z^4 + W^4Z^4) + L^7(W^2Z^6 + 2W^4Z^4 + W^5Z^2 + W^5Z^4) + \\ & L^8(2W^3Z^6 + 3W^4Z^4 + 2W^5Z^4 + W^6Z^4) + L^9(3W^3Z^6 + 3W^4Z^6 + \\ & 3W^5Z^4 + W^6Z^2 + 2W^6Z^4 + W^7Z^4) + \cdots . \end{aligned} \tag{16.41}$$

After dropping all terms of order larger than L^9, we obtain the single-error event enumerators as follows:

$$\begin{aligned} & A_{2,4}^{(1)}(Z) = Z^4, \qquad A_{2,7}^{(1)}(Z) = Z^6 \\ & A_{3,3}^{(1)}(Z) = Z^2, \qquad A_{3,5}^{(1)}(Z) = Z^4, \qquad A_{3,6}^{(1)}(Z) = 2Z^4, \ A_{3,8}^{(1)}(Z) = 2Z^6, \ A_{3,9}^{(1)}(Z) = 3Z^6 \\ & A_{4,5}^{(1)}(Z) = Z^2, \qquad A_{4,6}^{(1)}(Z) = Z^4, \qquad A_{4,7}^{(1)}(Z) = 2Z^4, \ A_{4,8}^{(1)}(Z) = 3Z^4, \ A_{4,9}^{(1)}(Z) = 3Z^6 \\ & A_{5,7}^{(1)}(Z) = Z^2 + Z^4, \ A_{5,8}^{(1)}(Z) = 2Z^4, \qquad A_{5,9}^{(1)}(Z) = 3Z^4 \\ & A_{6,8}^{(1)}(Z) = Z^4, \qquad A_{6,9}^{(1)}(Z) = Z^2 + 2Z^4 \\ & A_{7,9}^{(1)}(Z) = Z^4. \end{aligned} \tag{16.42}$$

Note that there are no weight-1 input sequences for this code, since a second 1 is needed to terminate the encoder. To determine the double-error event enumerators $A_{w,\lambda}^{(2)}(Z)$, we first form the double-error event IRWEF $A^{(2)}(W, Z, L)$ as follows:

$$\begin{aligned} A^{(2)}(W, Z, L) &= [A(W, Z, L)]^2 \\ &= L^6W^6Z^4 + 2L^7W^5Z^6 + L^8(W^4Z^8 + 2W^6Z^6 + 2W^7Z^4) + \\ &\quad L^9(2W^5Z^8 + 6W^6Z^6 + 2W^7Z^6) + \cdots . \end{aligned} \tag{16.43}$$

Dropping all terms of order greater than L^9 gives us the double-error event enumerators

$$\begin{aligned} A_{4,8}^{(2)}(Z) &= Z^8 \\ A_{5,7}^{(2)}(Z) &= 2Z^6, \quad A_{5,9}^{(2)}(Z) = 2Z^8 \\ A_{6,6}^{(2)}(Z) &= Z^4, \quad A_{6,8}^{(2)}(Z) = 2Z^6, \quad A_{6,9}^{(2)}(Z) = 6Z^6 \\ A_{7,8}^{(2)}(Z) &= 2Z^4, \quad A_{7,9}^{(2)}(Z) = 2Z^6. \end{aligned} \tag{16.44}$$

Following the same procedure, we obtain the *triple-error event IRWEF*,

$$A^{(3)}(W, Z, L) = [A(W, Z, L)]^3 = L^9 W^9 Z^6 + \cdots, \tag{16.45}$$

and we see that there is only one triple-error event of total length $\lambda \leq 9$. The triple-error event enumerator is given by

$$A_{9,9}^{(3)}(Z) = Z^6, \tag{16.46}$$

and it is clear that there are no higher-order error events of total length $\lambda \leq 9$.

Before using (16.39) to compute the CWEFs, we must determine h_{max} for each input weight w. From the double-error event IRWEF in (16.43) we see that there are no double-error events with input weight $w < 4$; that is, $h_{max} = 1$ for input weights 2 and 3. Similarly, from (16.45) we see that there are no triple-error events with input weight $w < 9$; that is, $h_{max} = 2$ for input weights 4 through 7. Finally, it is clear that $h_{max} = 3$ for input weight 9. Now, using (16.37) and (16.39), we can compute the CWEFs for the (18, 7) block code as follows:

$$\begin{aligned} A_2(Z) &= c[4,1]A_{2,4}^{(1)}(Z) + c[7,1]A_{2,7}^{(1)}(Z) = 6Z^4 + 3Z^6 \\ A_3(Z) &= c[3,1]A_{3,3}^{(1)}(Z) + c[5,1]A_{3,5}^{(1)}(Z) + c[6,1]A_{3,6}^{(1)}(Z) + \\ &\quad c[8,1]A_{3,8}^{(1)}(Z) + c[9,1]A_{3,9}^{(1)}(Z) \\ &= 7Z^2 + 13Z^4 + 7Z^6 \\ A_4(Z) &= c[5,1]A_{4,5}^{(1)}(Z) + c[6,1]A_{4,6}^{(1)}(Z) + c[7,1]A_{4,7}^{(1)}(Z) + c[8,1]A_{4,8}^{(1)}(Z) + \\ &\quad c[9,1]A_{4,9}^{(1)}(Z) + c[8,2]A_{4,8}^{(2)}(Z) \\ &= 5Z^2 + 16Z^4 + 3Z^6 + 3Z^8 \\ A_5(Z) &= c[7,1]A_{5,7}^{(1)}(Z) + c[8,1]A_{5,8}^{(1)}(Z) + c[9,1]A_{5,9}^{(1)}(Z) + \\ &\quad c[7,2]A_{5,7}^{(2)}(Z) + c[9,2]A_{5,9}^{(2)}(Z) \\ &= 3Z^2 + 10Z^4 + 12Z^6 + 2Z^8 \end{aligned}$$

$$\begin{aligned}
A_6(Z) &= c[8,1]A_{6,8}^{(1)}(Z) + c[9,1]A_{6,9}^{(1)}(Z) + c[6,2]A_{6,6}^{(2)}(Z) + \\
&\quad c[8,2]A_{6,8}^{(2)}(Z) + c[9,2]A_{6,9}^{(2)}(Z) \\
&= Z^2 + 14Z^4 + 12Z^6 \\
A_7(Z) &= c[9,1]A_{7,9}^{(1)}(Z) + c[8,2]A_{7,8}^{(2)}(Z) + c[9,2]A_{7,9}^{(2)}(Z) = 7Z^4 + 2Z^6 \\
A_9(Z) &= c[9,3]A_{9,9}^{(3)}(Z) = Z^6. \qquad (16.47)
\end{aligned}$$

As a check, we note that the CWEFs include a total of 127 nonzero codewords, the correct number for an (18, 7) code. Finally, the IRWEF and WEF are given by

$$\begin{aligned}
A(W,Z) &= W^2\left(6Z^4 + 3Z^6\right) + W^3\left(7Z^2 + 13Z^4 + 7Z^6\right) + \\
&\quad W^4\left(5Z^2 + 16Z^4 + 3Z^6 + 3Z^8\right) + W^5\left(3Z^2 + 10Z^4 + 12Z^6 + 2Z^8\right) + \\
&\quad W^6\left(Z^2 + 14Z^4 + 12Z^6\right) + W^7\left(7Z^4 + 2Z^6\right) + W^9Z^6, \qquad (16.48a)
\end{aligned}$$

and

$$A(X) = 7X^5 + 11X^6 + 16X^7 + 20X^8 + 17X^9 + 17X^{10} + 19X^{11} + 15X^{12} + 4X^{13} + X^{15}. \qquad (16.48b)$$

Note that this terminated convolutional code has free distance 5, the same as the free distance of the unterminated code; that is, it is an (18, 7, 5) code. We can also see that it has codeword multiplicities of $A_5 = 7$, $A_6 = 11$, $A_7 = 16, \cdots$; that is, it has a dense weight spectrum. We next calculate its average-weight spectrum when it is used, along with a size $K = 9$ interleaver, as a constituent code in a $(3K, K - \nu) = (27, 7)$ PCCC.

Before proceeding with this calculation, however, we must modify the uniform interleaver concept to take into account that the termination bits of a convolutional encoder are not information bits. In other words, for an input sequence of length K, the number of information bits is only $N - \nu$, where ν is the encoder constraint length, and we must modify the denominator in the uniform interleaver averaging expression given in (16.23) so that we divide only by the number of valid input sequences of each weight and not by all length-K sequences of a given weight; that is, because the ν termination bits are fixed for a given information sequence of length $K - \nu$, only a fraction $1/2^\nu$ of all length-K input sequences are valid. This technical detail complicates our example somewhat, but, as we will see shortly, for large K and small ν, that is, normal turbo code parameter values, we can use an approximate analysis that makes this modification unnecessary.

EXAMPLE 16.6 (Continued)

We begin by noting from (16.47) that there is 1 valid input sequence for weights 0 and 9, there are no valid input sequences for weights 1 and 8, there are 9 valid input

sequences for weights 2 and 7, and there are 27 valid input sequences for weights 3, 4, 5, and 6. Then, using the modified version of (16.23), we can compute the average CWEFs of the (27, 7) PCCC as follows:

$$A_2^{PC}(Z) = \frac{\left(6Z^4 + 3Z^6\right)^2}{9} = 4Z^8 + 4Z^{10} + Z^{12},$$

$$A_3^{PC}(Z) = \frac{\left(7Z^2 + 13Z^4 + 7Z^6\right)^2}{27}$$

$$= 1.81Z^4 + 6.74Z^6 + 9.89Z^8 + 6.74Z^{10} + 1.81Z^{12},$$

$$A_4^{PC}(Z) = \frac{\left(5Z^2 + 16Z^4 + 3Z^6 + 3Z^8\right)^2}{27}$$

$$= 0.93Z^4 + 5.93Z^6 + 10.59Z^8 + 4.67Z^{10} + 3.89Z^{12} + 0.67Z^{14} + 0.33Z^{16},$$

$$A_5^{PC}(Z) = \frac{\left(3Z^2 + 10Z^4 + 12Z^6 + 2Z^8\right)^2}{27}$$

$$= 0.33Z^4 + 2.22Z^6 + 6.37Z^8 + 9.33Z^{10} + 6.81Z^{12} + 1.78Z^{14} + 0.15Z^{16},$$

$$A_6^{PC}(Z) = \frac{\left(Z^2 + 14Z^4 + 12Z^6\right)^2}{27}$$

$$= 0.04Z^4 + 1.04Z^6 + 8.15Z^8 + 12.44Z^{10} + 5.33Z^{12},$$

$$A_7^{PC}(Z) = \frac{\left(7Z^4 + 2Z^6\right)^2}{9} = 5.44Z^8 + 3.11Z^{10} + 0.44Z^{12},$$

$$A_9^{PC}(Z) = \frac{\left(Z^6\right)^2}{1} = Z^{12}. \tag{16.49}$$

Finally, we can compute the average IRWEF $A^{PC}(W, Z)$ and the average WEF $A^{PC}(X)$ using (16.24) and (16.26a), respectively (see Problem 16.8).

We now make a few observations regarding Example 16.6.

- The free distance of the (27, 7) PCCC is 7, an increase of 2 compared with the (18, 7) constituent code. An increase in free distance is expected, though, since the (27, 7) code has a lower rate than the (18, 7) code. Note that in this case the minimum-weight codeword is produced by an input sequence of weight 3.
- The average multiplicities of the low-weight codewords, namely, $A_7 = 1.81$, $A_8 = 0.93$, and $A_9 = 7.07$, are small compared with the low-weight multiplicities of the (18, 7) code. This is another example of spectral thinning.
- Increasing the code length and interleaver size K leads to additional spectral thinning, which results in improved performance at low SNRs. In fact, we will see shortly that for large K, the multiplicities of the low-weight codewords in a PCCC are reduced by a factor of K compared with the constituent codes. This multiplicity reduction factor is called the *interleaver gain*.

- The calculation of the average bit IRWEFs $B(W, Z)$ and $B^{PC}(W, Z)$ for this example is also complicated by the fact that the termination bits are not information bits, and the factor (w/K) in (16.25) must be adjusted to consider only the information bits (see Problem 16.9).

- The observations that turbo coding produces (1) a normal increase in free distance owing to the reduction in code rate and (2) a large decrease in codeword and bit multiplicities owing to the interleaver gain illustrate that, unlike conventional code design, turbo codes are designed to reduce the low-weight multiplicities rather than to increase the free distance. This results in much better performance than conventional codes achieve at low and moderate SNRs but somewhat weaker performance at high SNRs.

- By including the termination bits for the first encoder in the interleaver, we cannot guarantee that the second encoder terminates (in fact, as noted earlier, it will do so only with probability $1/2^{\nu}$), but this has little effect on code performance for large K.

Before extending the analysis to larger block lengths and interleaver sizes, we note that the uniform interleaver averaging represented by (16.23) results in dividing the product of the constituent code CWEFs by the factor $\binom{K}{w}$. Because there are no weight-1 input sequences to a terminated systematic feedback encoder (because of the termination bits), the nonzero input weight with the smallest division factor in (16.23) is the $w = 2$ term, for which the division factor is $\binom{K}{2}$. For $w = 3$, the division factor $\binom{K}{3}$ is larger by roughly a factor of K. In other words, compared with their relative influence in the individual constituent codes, weight-3 input sequences in PCCCs are less important than weight-2 input sequences, since they are associated with lower average codeword and bit multiplicities. Thus, particularly for large block lengths K, the codewords associated with weight-2 input sequences become the most important contributors to the low-order terms in the weight spectrum of PCCCs.

To develop an approximate weight spectrum analysis of PCCCs for large block lengths, we begin by simplifying the expression given in (16.37) for the number of codewords containing h error events of total length λ in a terminated convolutional code with block length K. Because, for the low-weight codewords of primary interest, both the number of error events h and the total length λ of error events cannot be large (or else the weight would also be large), for large K it is reasonable to approximate the number of codewords containing h error events in a block as

$$c[h] \approx \binom{K}{h}, \tag{16.50}$$

independent of the total length λ of the error events. Then, the CWEFs of the equivalent $(nK, K - \nu)$ block code are given by

$$A_w(Z) = \sum_{(1 \le h \le h_{max})} c[h] A_w^{(h)}(Z), \tag{16.51}$$

where $A_w^{(h)}(Z)$ is the *h-error event enumerator* for input weight w. (We note here that the h-error event enumerators of (16.39) and (16.51), $A_{w,\lambda}^{(h)}(Z)$ and $A_w^{(h)}(Z)$, respectively, are related by the expression $A_w^{(h)}(Z) = \sum_{(\lambda \le K)} A_{w,\lambda}^{(h)}(Z)$; that is, $A_w^{(h)}(Z)$ counts h-error events of all lengths.)

From (16.23), (16.50), and (16.51), the average CWEFs of the PCCC with the same $(nK, K - \nu)$ terminated convolutional code as constituent codes are given by

$$A_w^{PC}(Z) = \sum_{(1 \le h_1 \le h_{max})} \sum_{(1 \le h_2 \le h_{max})} \frac{c[h_1]\, c[h_2]}{\binom{K}{w}} A_w^{(h_1)}(Z) A_w^{(h_2)}(Z). \tag{16.52}$$

For $K \gg h$, we can use the approximation

$$\binom{K}{h} \approx \frac{K^h}{h!} \tag{16.53}$$

to obtain

$$A_w^{PC}(Z) \approx \sum_{(1 \le h_1 \le h_{max})} \sum_{(1 \le h_2 \le h_{max})} \frac{w!}{h_1! \cdot h_2!} K^{(h_1 + h_2 - w)} A_w^{(h_1)}(Z) A_w^{(h_2)}(Z). \tag{16.54}$$

We can further approximate by saving only the (most significant) term in the double summation of (16.54) with the highest power of K, that is, the term corresponding to $h_1 = h_2 = h_{max}$, which gives us the approximate codeword CWEF

$$A_w^{PC}(Z) \approx \frac{w!}{(h_{max}!)^2} K^{(2h_{max} - w)} \left[A_w^{(h_{max})}(Z) \right]^2 \tag{16.55a}$$

and the approximate bit CWEF

$$\begin{aligned} B_w^{PC}(Z) &\approx \frac{w}{K} \cdot \frac{w!}{(h_{max}!)^2} K^{(2h_{max} - w)} \left[A_w^{(h_{max})}(Z) \right]^2 \\ &= \frac{w \cdot w!}{(h_{max}!)^2} K^{(2h_{max} - w - 1)} \left[A_w^{(h_{max})}(Z) \right]^2. \end{aligned} \tag{16.55b}$$

(It is worth noting that taking only the term corresponding to $h_1 = h_2 = h_{max}$ in (16.54) is equivalent to saying that for a *fixed input weight* w, the lowest-weight codeword is likely to contain the maximum number of error events, so that it is merged with the all-zero codeword as much as possible.) Finally, the average IRWEFs of the PCCC are given by

$$A^{PC}(W, Z) = \sum_{(1 \le w \le K)} W^w A_w^{PC}(Z) \tag{16.56a}$$

and

$$B^{PC}(W, Z) = \sum_{(1 \le w \le K)} \frac{w}{K} W^w A_w^{PC}(Z) = \sum_{(1 \le w \le K)} W^w B_w^{PC}(Z), \tag{16.56b}$$

and we can compute the average WEFs using (16.26). (In the calculation of the approximate IRWEFs from (16.56), the sums should be taken only over those w for which the approximations of (16.50), (16.53), and (16.55) are reasonably accurate, that is, w's of relatively low weight.) The extension of this analysis to PCCCs with different constituent codes is straightforward.

EXAMPLE 16.7 Computing Approximate WEF's for a PCCC

Consider a $(3K, K-2)$ PCCC in which both constituent codes are generated by the $(2, 1, 2)$ systematic feedback encoder of Example 16.6, but now for a block length of $K \gg \nu = 2$. For input weights $w = 2$ and 3, $h_{max} = 1$ and (16.55) simplifies to

$$A_w^{PC}(Z) \approx w! \cdot K^{(2-w)} \left[A_w^{(1)}(Z)\right]^2 \tag{16.57a}$$

and

$$B_w^{PC}(Z) \approx w \cdot w! \cdot K^{(1-w)} \left[A_w^{(1)}(Z)\right]^2, \tag{16.57b}$$

and for input weights $w = 4$ and 5, $h_{max} = 2$, and (16.55) is given by

$$A_w^{PC}(Z) \approx \frac{w!}{4} \cdot K^{(4-w)} \left[A_w^{(2)}(Z)\right]^2 \tag{16.58a}$$

and

$$B_w^{PC}(Z) \approx w \cdot \frac{w!}{4} \cdot K^{(3-w)} \left[A_w^{(2)}(Z)\right]^2. \tag{16.58b}$$

(Note that in Example 16.6 when $K = 9$, $h_{max} = 2$ also for $w = 6$ and 7, but this is not the case for large K.) Including only these terms in the approximate average IRWEFs for the PCCC gives

$$\begin{aligned} A^{PC}(W, Z) &\approx \sum_{(2 \le w \le 5)} W^w A_w^{PC}(Z) \\ &= 2W^2 \left[A_2^{(1)}(Z)\right]^2 + \frac{6}{K} W^3 \left[A_3^{(1)}(Z)\right]^2 + 6W^4 \left[A_4^{(2)}(Z)\right]^2 + \\ &\quad \frac{30}{K} W^5 \left[A_5^{(2)}(Z)\right]^2 + \cdots \end{aligned} \tag{16.59a}$$

and

$$\begin{aligned} B^{PC}(W, Z) &\approx \sum_{(2 \le w \le 5)} W^w B_w^{PC}(Z) \\ &= \frac{4}{K} W^2 \left[A_2^{(1)}(Z)\right]^2 + \frac{18}{K^2} W^3 \left[A_3^{(1)}(Z)\right]^2 + \frac{24}{K} W^4 \left[A_4^{(2)}(Z)\right]^2 + \\ &\quad \frac{150}{K^2} W^5 \left[A_5^{(2)}(Z)\right]^2 + \cdots . \end{aligned} \tag{16.59b}$$

The single- and double-error event enumerators $A_w^{(1)}(Z)$ and $A_w^{(2)}(Z)$ needed to evaluate (16.59) include all single- and double-error events up to length K; however, the terms of most interest, that is, the lowest-weight terms, can be obtained from

Example 16.6, which counts all error events up to length $K = 9$. Thus, from (16.42) and (16.44), we obtain

$$\begin{aligned} A_2^{(1)}(Z) &\approx Z^4 + Z^6 \quad & A_3^{(1)}(Z) &\approx Z^2 + 3Z^4 + 5Z^6 \\ A_4^{(2)}(Z) &\approx Z^8 & A_5^{(2)}(Z) &\approx 2Z^6 + 2Z^8, \end{aligned} \tag{16.60}$$

and (16.59) becomes

$$\begin{aligned} A^{PC}(W, Z) &\approx 2W^2\left[Z^4 + Z^6\right]^2 + \frac{6}{K}W^3\left[Z^2 + 3Z^4 + 5Z^6\right]^2 + \\ &\quad 6W^4\left[Z^8\right]^2 + \frac{30}{K}W^5\left[2Z^6 + 2Z^8\right]^2 + \cdots \\ &= W^2\left(2Z^8 + 4Z^{10} + 2Z^{12}\right) + W^4\left(6Z^{16}\right) + \\ &\quad W^3\frac{1}{K}(6Z^4 + 36Z^6 + 114Z^8 + 180Z^{10} + 150Z^{12}) + \\ &\quad W^5\frac{1}{K}\left(120Z^{12} + 240Z^{14} + 120Z^{16}\right) + \cdots \end{aligned} \tag{16.61a}$$

and

$$\begin{aligned} B^{PC}(W, Z) &\approx W^2\frac{1}{K}\left(4Z^8 + 8Z^{10} + 4Z^{12}\right) + W^4\frac{1}{K}\left(24Z^{16}\right) + \\ &\quad W^3\frac{1}{K^2}\left(18Z^4 + 108Z^6 + 342Z^8 + 540Z^{10} + 450Z^{12}\right) + \\ &\quad W^5\frac{1}{K^2}\left(600Z^{12} + 1200Z^{14} + 600Z^{16}\right) + \cdots . \end{aligned} \tag{16.61b}$$

Note that the approximate expressions of (16.61) contain no terms of order $(1/K)^3$ or higher. (These terms clearly have little influence on the IRWEFs for large values of K.) Finally, we obtain the approximate average WEFs from (16.26) as follows:

$$A^{PC}(X) \approx \frac{6}{K}X^7 + \frac{36}{K}X^9 + 2X^{10} + \frac{114}{K}X^{11} + 4X^{12} + \frac{180}{K}X^{13} + 2X^{14} + \cdots \tag{16.61c}$$

and

$$B^{PC}(X) \approx \frac{18}{K^2}X^7 + \frac{108}{K^2}X^9 + \frac{4}{K}X^{10} + \frac{342}{K^2}X^{11} + \frac{8}{K}X^{12} + \frac{540}{K^2}X^{13} + \frac{4}{K}X^{14} + \cdots . \tag{16.61d}$$

The following remarks apply to Example 16.7:

- The minimum free distance of the $(3K, K - 2)$ PCCC still appears to be 7, but the average codeword multiplicity is only $A_7 = 6/K$. For large K, this means that the probability that a particular interleaver will result in a weight-7 codeword is very small; that is, the free distance in almost all cases will be greater than 7.

- Approximations (16.61c) and (16.61d) do not indicate the presence of any codewords of weight 8, because we kept only the most significant term in the double summation of (16.54). If the term leading to weight-8 codewords had not been deleted from (16.54), however, it would have order $(1/K)^2$ in $A^{PC}(X)$ and order $(1/K)^3$ in $B^{PC}(X)$; that is, it would have a negligible effect on the WEFs for large K.

- The average multiplicity of weight-9 codewords is $A_9 = 36/K$; that is, for large K the probability of a weight-9 codeword is very small.

- The average multiplicity of weight-10 codewords is $A_{10} = 2$; that is, the interleaver does not purge the code of weight-10 codewords, and in almost all cases the free distance of the PCCC will be 10. We see that the weight-10 codewords are generated by weight-2 input sequences, thus illustrating the importance of weight-2 input sequences when K is large that was noted earlier.

- The (average of) two weight-10 codewords are produced by the $K - 2$ input sequences $\mathbb{m}(D) = D^l(1 + D^3)$, $0 \le l \le K - 3$, where we note that $1 + D^3$ is the shortest weight-2 input sequence that terminates the encoder. This input sequence generates the weight-4 parity sequence $\mathbb{v}(D) = 1 + D + D^2 + D^3$. A pseudorandom interleaver will typically leave two of these $K - 2$ input sequences unchanged (except for time shifts), and in these cases the second encoder will output the same weight-4 parity sequence, resulting in codewords of weight 10. (To see this, assume that the first 1 in one of the foregoing weight-2 input sequences is permuted to an arbitrary position by the interleaver. There are then two positions out of the remaining $K - 1$ positions, that the second 1 can appear and still give the same input pattern, namely, either three positions before or three positions after the first 1. Thus each of the $K - 2$ "bad" input sequences has probability $2/(K - 1)$ of retaining its bad property after interleaving, which leads to the average multiplicity of $A_{10} = 2$ for weight-10 codewords.)

- For each constituent terminated convolutional code by itself, the low-weight codeword multiplicities increase linearly with K (because of time invariance), but for the PCCC the multiplicities of the low-weight codewords are smaller by factors of $(1/K)^p$, $p \ge 1$. Note that the largest factor, that is, $(1/K)^1$, is associated with terms resulting from input sequences of weight 2.

- From (16.61d), we see that the bit multiplicities of the low-weight codewords are $B_7 = 18/K^2$, $B_9 = 108/K^2$, $B_{10} = 4/K$, and so on. Because the bit multiplicities of the low-weight codewords in the PCCC decrease with K, we say that the PCCC possesses *interleaver gain*.

EXAMPLE 16.7 (Continued)

We continue the example by computing the approximate low-weight codeword and bit multiplicities, using (16.61c) and (16.61d), for three specific cases: $K = 100, 1000$, and 10000. The results are presented in Table 16.5, where the entries for the bit multiplicities B_d clearly portray the interleaver gain effect. For each weight d, the

TABLE 16.5: Approximate codeword and bit multiplicities for the $(3K, K-2)$ PCCC of Example 16.7.

K	100		1000		10000	
d	A_d	B_d	A_d	B_d	A_d	B_d
7	0.06	0.0018	0.006	0.000018	0.0006	0.00000018
8	0.00	0.0000	0.000	0.000000	0.0000	0.00000000
9	0.36	0.0108	0.036	0.000108	0.0036	0.00000108
10	2.00	0.0400	2.000	0.004000	2.0000	0.00040000
11	5.40	0.1620	0.540	0.001620	0.0540	0.00001620
12	4.00	0.0800	4.000	0.008000	4.0000	0.00080000
13	1.80	0.0540	0.180	0.000540	0.0180	0.00000540
14	2.00	0.0400	2.000	0.004000	2.0000	0.00040000
15	1.50	0.0450	0.150	0.000450	0.0150	0.00000450
16	0.00	0.0000	0.000	0.000000	0.0000	0.00000000
17	1.20	0.0600	0.120	0.000600	0.0120	0.00000600
18	0.00	0.0000	0.000	0.000000	0.0000	0.00000000
19	2.40	0.1200	0.240	0.001200	0.0240	0.00001200

value of B_d decreases by one or two orders of magnitude for each order of magnitude increase in K. Note in particular that the interleaver gain for codeword weights 7 and 9 is larger by roughly a factor of $(1/K)$ than for codeword weight 10. This is because weight-10 codewords are generated by weight-2 input sequences, whereas weight-7 and weight-9 codewords are generated by weight-3 input sequences. This is another illustration of the importance of weight-2 input sequences when the block length K becomes very large.

In Problems 16.10 and 16.11 we repeat Examples 16.6 and 16.7 for the systematic feedback convolutional encoder whose generator polynomials are inverted, that is, the encoder given by

$$\mathbf{G}_{fb}(D) = \left[\begin{array}{cc} 1 & (1 + D + D^2)/(1 + D^2) \end{array}\right]. \tag{16.62}$$

The $(2, 1, 2)$ convolutional code generated by this encoder also has $d_{free} = 5$; however, in this case, the minimum-weight codeword is produced by an input sequence of weight-2, that is, $\mathbf{u}(D) = 1 + D^2$. Thus, when this code is converted to a $(3K, K-2)$ PCCC, its minimum free distance for both small and large values of K will be 8. (The parity weight corresponding to the weight-2 input sequence is 3, which is repeated twice in the PCCC.) In other words, when the minimum-weight codeword is produced by an input sequence of weight-2, increasing the block length K results in interleaver gain, which reduces the bit multiplicities, but no improvement in free distance is achieved. This contrasts with the code in Examples 16.6 and 16.7, where the free distance improved from 7 to 10 by increasing the block length K.

From the results of Examples 16.6 and 16.7 and Problems 16.10 and 16.11 we conclude that to maximize the free distance of a PCCC, a constituent code should

be chosen whose minimum-weight codeword is produced by an input sequence of weight greater than 2. This is usually the case when the feedback polynomial is chosen to be primitive, as in Examples 16.6 and 16.7. This is because a primitive polynomial has maximum cycle length; that is, it maximizes the length of the weight-2 input sequence needed to terminate the encoder. Depending on the numerator polynomial, this normally results in higher parity weight than when one or more shorter, higher-weight input sequences terminate the encoder. Because the free distance determines the asymptotic, that is, large-SNR behavior of a code, primitive feedback polynomials are good choices to optimize the large-SNR performance of PCCCs. One should not conclude, however, that the choice of primitive feedback polynomials also optimizes the small-SNR performance, where the low-weight codeword multiplicities and the dynamics of iterative decoding are more important than the free distance. In fact, experience with different code selections suggests that, in general, primitive feedback polynomials optimize large-SNR behavior, but nonprimitive polynomials with shorter cycle lengths are better for small SNRs. A more extensive analysis of constituent code design for PCCCs is included in Section 16.4

The preceding discussion, along with a close examination of the power of K, namely, $2h_{max} - w - 1$, in (16.55b) illustrate that the input weights that produce low-weight codewords in the constituent code must be at least 2 to achieve interleaver gain in PCCCs. Interleaver gain results only when the power of K in (16.55b) is less than or equal to -1, so that the bit multiplicities decrease with increasing K. Because h_{max} is always 1 for the lowest-weight input sequences, this means that the lowest input weight w that produces low-weight codewords must be at least 2. When feedback encoders, that is, recursive convolutional codes, are used as constituent codes, almost all weight-1 input sequences produce high-weight parity sequences, and thus high-weight codewords. (The only exception is when the nonzero input does not appear until the end of the information block, but the number of these sequences is small and does not grow linearly with K. Also, when the encoder is terminated, then at least one more 1 must enter the encoder, and hence the total input weight is still at least 2 in this case.) Thus, for recursive constituent codes, all low-weight codewords are produced by input sequences of at least weight 2, and interleaver gain is achieved.

This is not the case when short block codes are used as constituent codes. When a short block code is repeated to achieve a large block length and interleaver size, there exist low-weight codewords produced by a single 1, thus negating the possibility of interleaver gain. If a single long block code is used as a constituent code, the desired property can be achieved by choosing the weight of each row of the generator matrix $\mathbb{G}$ to be large; however, efficient SISO decoding methods for such long codes are difficult to realize.

Interleaver gain is also not possible when short constraint length feedforward convolutional encoders, that is, nonrecursive convolutional codes, are used as constituent codes. This is because single input 1's in short constraint length feedforward encoders produce short, low-weight output sequences, and thus large low-weight codeword multiplicities result from parallel concatenation. This point is illustrated in the next example.

EXAMPLE 16.8 A PCCC with Feedforward Constituent Encoders

Consider the $(3K, K-2)$ turbo code (PCCC) generated by the parallel concatenation of a (2, 1, 2) nonsystematic feedforward encoder, as shown in Figure 16.11. (Recall that for nonsystematic encoders the weight spectrum is represented by the IOWEF rather than the IRWEF.) Assume that the generator matrix of the encoder is given by

$$\mathbb{G}_{ff}^{1}(D) = \begin{bmatrix} 1 + D + D^2 & 1 + D^2 \end{bmatrix}. \tag{16.63}$$

The code produced by this encoder is identical to the code produced by the systematic feedback encoder of Examples 16.6 and 16.7, but the mapping between input sequences and codewords is different. That is, the two codes have identical WEFs but different IOWEFs. To analyze the weight spectrum of this PCCC, we must determine both the conditional IOWEF $A_w^{C_1}(X)$ of the nonsystematic nonrecursive constituent code C_1 and the conditional IRWEF $A_w^{C_2}(Z)$ of the (2, 1, 2) systematic nonrecursive constituent code C_2 whose parity generator is used to reencode the input sequence after interleaving. In our example, this systematic nonrecursive code is generated by the feedforward encoder

$$\mathbb{G}_{ff}^{2}(D) = \begin{bmatrix} 1 & 1 + D^2 \end{bmatrix}. \tag{16.64}$$

Then, the effect of uniform interleaving is represented by

$$A_w^{PC}(X, Z) = \frac{A_w^{C_1}(X) A_w^{C_2}(Z)}{\binom{K}{w}}, \tag{16.65}$$

and the approximate expression for large K is

$$A_w^{PC}(X, Z) \approx \sum_{(1 \le h_1 \le h_{max})} \sum_{(1 \le h_2 \le h_{max})} \frac{c[h_1] c[h_2]}{\binom{K}{w}} A_w^{(h_1)}(X) A_w^{(h_2)}(Z), \tag{16.66}$$

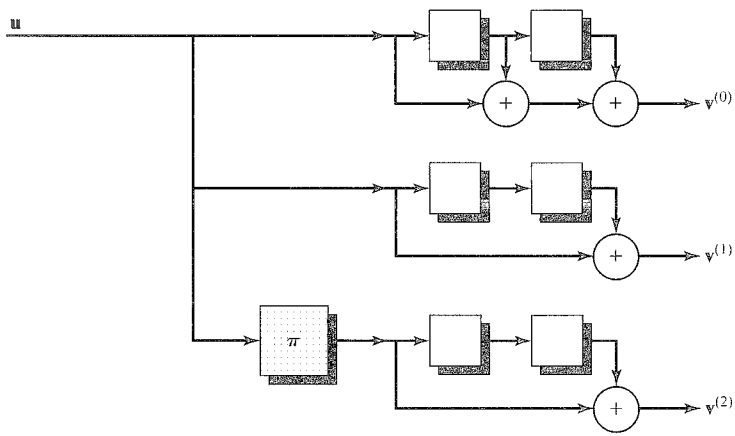


FIGURE 16.11: A PCCC using feedforward constituent encoders.

where $A_w^{(h_1)}(X)$ and $A_w^{(h_2)}(Z)$ are the conditional IO and IR h-error event WEFs of the nonsystematic and systematic constituent codes, C_1 and C_2, respectively. Now, following the approximations of (16.53)–(16.55) results in

$$A_w^{PC}(X, Z) \approx \frac{w!}{h_{1max}! \cdot h_{2max}!} K^{(h_{1max}+h_{2max}-w)} A_w^{(h_{1max})}(X) A_w^{(h_{2max})}(Z) \tag{16.67a}$$

and

$$B_w^{PC}(X, Z) \approx w \cdot \frac{w!}{h_{1max}! \cdot h_{2max}!} K^{(h_{1max}+h_{2max}-w-1)} A_w^{(h_{1max})}(X) A_w^{(h_{2max})}(Z). \tag{16.67b}$$

Because the total codeword weights in the resulting PCCC are sums of powers of X and Z, the average CWEFs are

$$A_w^{PC}(X) = A_w^{PC}(X, Z)\Big|_{Z=X} \tag{16.68a}$$

and

$$B_w^{PC}(X) = B_w^{PC}(X, Z)\Big|_{Z=X}, \tag{16.68b}$$

the average IOWEFs are given by

$$A^{PC}(W, X) = \sum_{(1 \le w \le K)} W^w A_w^{PC}(X) \tag{16.69a}$$

and

$$B^{PC}(W, X) = \sum_{(1 \le w \le K)} \frac{w}{K} W^w A_w^{PC}(X) = \sum_{(1 \le w \le K)} W^w B_w^{PC}(X), \tag{16.69b}$$

and the average WEFs are given by

$$A^{PC}(X) = A^{PC}(W, X)\Big|_{W=1} \tag{16.69c}$$

and

$$B^{PC}(X) = B^{PC}(W, X)\Big|_{W=1}. \tag{16.69d}$$

Beginning with the nonsystematic constituent code generated by (16.63), we see that its IOWEF is given by (see Problem 11.23)

$$A(W, X, L) = WX^5L^3 + W^2L^4(1 + L)X^6 + W^3L^5(1 + L)^2X^7 + \cdots + W^wL^{w+2}(1 + L)^{w-1}X^{w+4} + \cdots. \tag{16.70}$$

Going directly to the large-K case, we note that for any input weight w, $w \ge 1$, $h_{max} = w$, since the weight one single-error event can simply be repeated w times, and therefore

$$A_w^{(w)}(X) = X^{5w}, \quad w \ge 1. \tag{16.71}$$

Now, for the systematic constituent code generated by (16.64), we find that its IRWEF is given by (see Problem 11.24)

$$A(W, Z, L) = WZ^2L^3 + W^2Z^4L^4 + W^2Z^2L^5(1 + WZ^2) + W^3Z^2L^6\left(2Z^2 + WZ^2\right) + \cdots. \tag{16.72}$$

Again, for any input weight w, $w \geq 1$, $h_{max} = w$, and hence,

$$A_w^{(w)}(Z) = Z^{2w}, \quad w \geq 1. \tag{16.73}$$

Now, substituting into (16.67) we obtain

$$A_w^{PC}(X, Z) \approx \frac{w!}{(w!) \cdot (w!)} K^{(w+w-w)} X^{5w} Z^{2w} = \frac{K^w}{w!} X^{5w} Z^{2w} \tag{16.74a}$$

and

$$B_w^{PC}(X, Z) \approx \frac{K^{(w-1)}}{(w-1)!} X^{5w} Z^{2w}. \tag{16.74b}$$

Finally, applying (16.68) and (16.69) gives us the average IOWEFs,

$$A^{PC}(W, X) \approx \sum_{(1 \leq w \leq K)} W^w \frac{K^w}{w!} X^{7w} = KWX^7 + \frac{K^2}{2} W^2 X^{14} + \frac{K^3}{6} W^3 X^{21} + \cdots \tag{16.75a}$$

and

$$B^{PC}(W, X) \approx WX^7 + KW^2 X^{14} + \frac{K^2}{2} W^3 X^{21} + \cdots, \tag{16.75b}$$

and the average WEFs,

$$A^{PC}(X) \approx KX^7 + \frac{K^2}{2} X^{14} + \frac{K^3}{6} X^{21} + \cdots \tag{16.75c}$$

and

$$B^{PC}(X) \approx X^7 + KX^{14} + \frac{K^2}{2} X^{21} + \cdots. \tag{16.75d}$$

We now conclude this section with a few observations regarding Example 16.8.

- The free distance of the $(3K, K - 2)$ PCCC is 7, and the average codeword multiplicity $A_7 \approx K$ grows linearly with K. Also, the average bit multiplicity of the weight-7 codewords is $B_7 = 1$, because the weight-7 codewords are generated by weight-1 input sequences, and the interleaver has no effect (other than time shifts) on weight-1 input sequences. This contrasts with the feedback encoder case, in which the bit multiplicities all decrease with increasing K. Thus interleaver gain does not apply to feedforward encoders.

- The approximate expressions of (16.75) do not indicate the presence of any codewords with weights between 7 and 14, because we kept only the most significant term in the double summation of (16.66). If the terms leading to codeword weights between 7 and 14 are also retained, we find that the corresponding codeword and bit multiplicities do not grow linearly with K, and thus they will have a negligible effect for large K compared with codeword weights 7 and 14.

- From this example it is clear that when feedforward encoders are employed, h_{max} is always equal to w. This is because, in the feedforward case, the weight-1

input sequence produces the minimum length (equal to the constraint length v) terminated output sequence, and hence the maximum number of error events that can be produced by a weight-w input sequence is achieved by simply repeating this shortest error event w times. This is not true for feedback encoders, since in this case the weight-1 input sequence produces an unterminated output sequence.

- As in the case of block constituent codes, long constraint length feedforward encoders could be used as constituent codes, but efficient SISO decoding methods for such long codes are difficult to realize.

In the next section we use the results of the weight spectrum analysis presented in this section, along with standard union bounding techniques, to evaluate the performance of PCCCs.

16.3 PERFORMANCE ANALYSIS OF TURBO CODES

We can now use the union bounding techniques developed in Chapter 12 to estimate the performance of turbo codes. We will consider both the *word-error probability*

$$P_w(E) \leq \sum_{(d_{free} \leq d)} A_d P_d, \tag{16.76}$$

and the *bit-error probability*

$$P_b(E) \leq \sum_{(d_{free} \leq d)} B_d P_d, \tag{16.77}$$

where A_d is the codeword multiplicity, B_d is the bit multiplicity, P_d is the pairwise error probability for a codeword of weight d, and d_{free} is the minimum free distance of the code. In this chapter we concentrate on code performance for a binary-input, unquantized-output AWGN channel. In this case, as shown in Chapter 12, P_d is given by

$$P_d = Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right), \tag{16.78}$$

which can be bounded as

$$P_d \leq f\left(\frac{d_{free}RE_b}{N_0}\right) \cdot (e^{-RE_b/N_0})^d, \tag{16.79}$$

where R is the code rate, E_b/N_0 is the energy per information bit to one-sided noise power spectral density ratio, that is, the SNR, and the function $f(\cdot)$ is defined as

$$f(x) = e^x \cdot Q\left(\sqrt{2x}\right). \tag{16.80}$$

Now, substituting (16.79) into (16.76) and (16.77) gives us the expressions

$$\begin{aligned} P_w(E) &\leq f\left(\frac{d_{free}RE_b}{N_0}\right) \sum_{(d_{free} \leq d)} A_d \left(e^{-RE_b/N_0}\right)^d \\ &= f\left(\frac{d_{free}RE_b}{N_0}\right) \cdot A(X)\Big|_{X=e^{-RE_b/N_0}} \end{aligned} \tag{16.81}$$

and

$$P_b(E) \leq f\left(\frac{d_{free}RE_b}{N_0}\right) \sum_{(d_{free} \leq d)} B_d \left(e^{-RE_b/N_0}\right)^d$$
$$= f\left(\frac{d_{free}RE_b}{N_0}\right) \cdot B(X)\Big|_{X=e^{-RE_b/N_0}} . \tag{16.82}$$

Thus, to bound the performance of a turbo code, all one needs is the free distance d_{free}, the codeword WEF $A(X)$, and the bit WEF $B(X)$. For values of E_b/N_0 above the SNR at which the code rate R equals the channel cutoff rate R_0, that is, for

$$\frac{E_b}{N_0} > \frac{1}{R} \ln\left(2^{1-R} - 1\right)^{-1}, \tag{16.83}$$

the first few terms of each WEF are all that are needed for an accurate estimate of performance. For values of E_b/N_0 below this limit, however, more terms are needed and the bound diverges, a characteristic that is typical of all union bounds.

We now consider several examples illustrating the application of the foregoing bounds to estimate the performance of turbo codes.

EXAMPLE 16.9 Performnace Bounds for Turbo Codes

Consider the rate $R = 1/3$ PCCC generated by the systematic feedback encoder of Examples 16.6 and 16.7. Approximations to the codeword and bit WEFs are given in (16.61c) and (16.61d), and the free distance of this code is $d_{free} = 7$. (Note that for large K, most interleavers will result in a free distance of $d_{free} = 10$, but in evaluating the bound we must use the smallest possible free distance, namely, $d_{free} = 7$.) Substituting these expressions into (16.81) and (16.82) allows us to calculate the bounds on word- and bit-error probability as functions of E_b/N_0. Results are shown in Figure 16.12 for three values of the interleaver size (information block length): $K = 100$, 1000, and 10000. (The program used to plot the bounds of (16.81) and (16.82) was based on the approximations of the WEFs given in (16.61c) and (16.61d) but included more terms for greater accuracy.) $P_w(E)$ is plotted in Figure 16.12(a), and $P_b(E)$ in Figure 16.12(b). For comparison we also show the three corresponding $P_w(E)$ simulation curves for the optimum free distance (3, 1, 4) terminated convolutional code in Figure 16.12(a), and the corresponding $P_b(E)$ simulation curve for the unterminated version of this code in Figure 16.12(b). A 16-state convolutional code is chosen for comparison with the 4-state PCCC because their decoding complexities are roughly equal. Finally, a simulation curve for the PCCC with $K = 10000$ is shown in each figure. The PCCC simulations were generated using a typical pseudorandom interleaver and 18 iterations of decoding.

Several observations regarding Example 16.9 follow:

- The PCCC curves in Figure 16.12(b) clearly indicate the effect of interleaver gain. For each factor-of-10 increase in K, the $P_b(E)$ curves improve by roughly the same factor. (Note, however, that the SNR difference in the $P_b(E)$ curves, that is, the coding gain, achieved by increasing K becomes smaller for large E_b/N_0.)

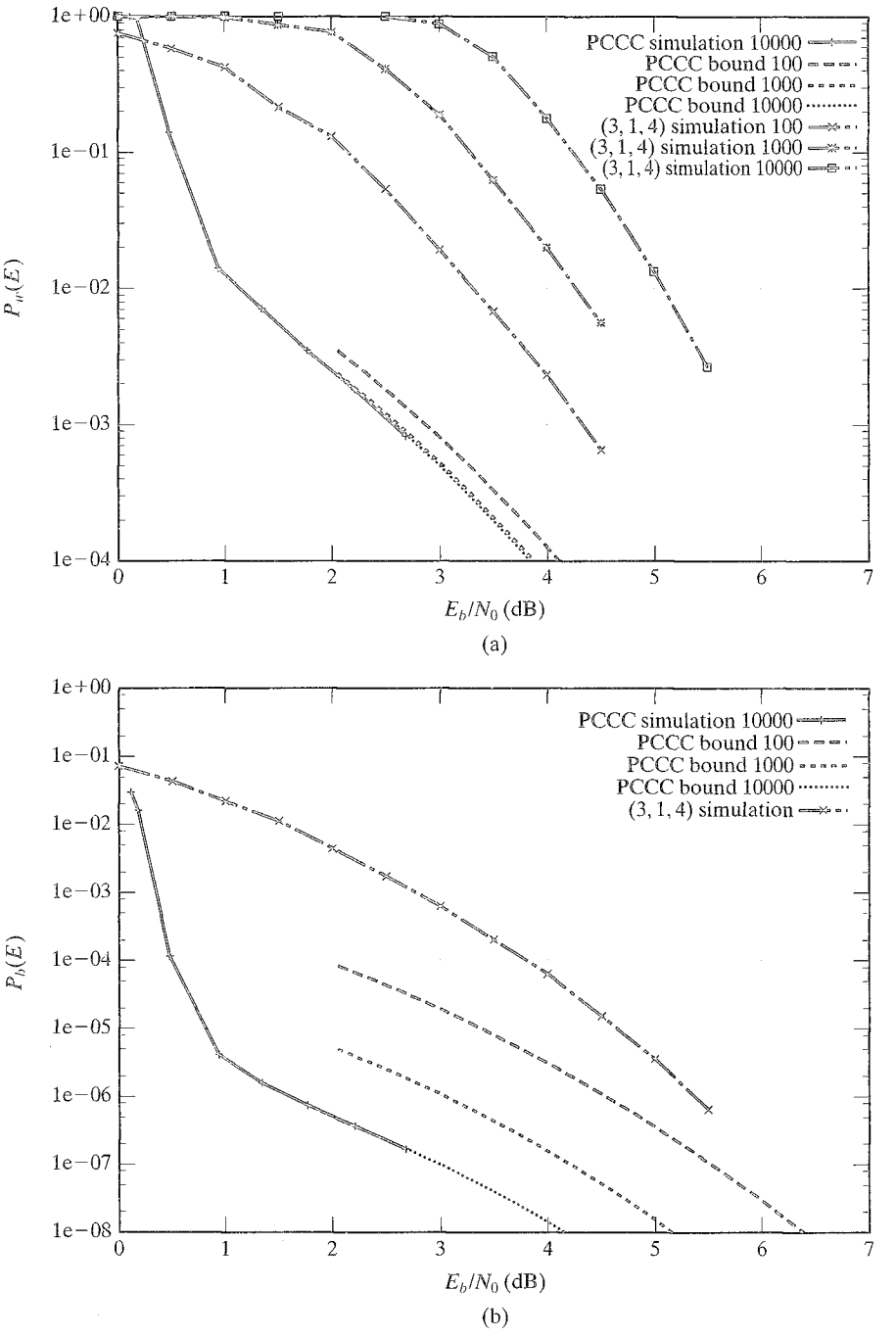


FIGURE 16.12: (a) Word- and (b) bit-error probability curves for a PCCC and a convolutional code with rate $R = 1/3$.

- The $P_w(E)$ curves for the PCCC in Figure 16.12(a) merge for large K because the number of weight-10 codewords remains fixed at two and does not decrease with increasing K. For the convolutional code, on the other hand, $P_w(E)$ becomes worse as K increases. This is to be expected, since with increasing block length the occurrence of at least one error event becomes more likely. This effect is masked by the interleaver for the PCCC, because there are no low-weight codewords whose multiplicities increase linearly with K.

- Compared with the convolutional code, the PCCC offers substantial coding gain, both in terms of $P_w(E)$ and $P_b(E)$. For example, for $P_b(E) = 10^{-5}$, the PCCC with $K = 10000$ achieves a 3.8-dB coding gain compared with the convolutional code, despite the fact that the convolutional code has free distance 12, larger than the (most likely) free distance of 10 for the PCCC.

- The bounds are not extended to SNRs below 2 dB, the cutoff rate limit for rate $R = 1/3$ given by (16.83), since they diverge below this point, as noted previously. (In the case of $K = 10000$, the PCCC simulation is used to extend the bound below 2 dB.) At lower SNRs, the simulation curves show much better performance than predicted by the bounds, but at higher SNRs the bounds predict the performance very accurately. Tighter bounding arguments can be employed to improve the bounds at low SNRs [33], however.

- If the bounds are being used to estimate performance for a particular interleaver and block length for which the low-order terms in the WEFs are known, the factor $f(d_{free}RE_b/N_0)$ should be adjusted to the actual value of d_{free}.

EXAMPLE 16.10 Turbo Code Performance as a Function of Constraint Length

Now, consider the series of rate $R = 1/3$ PCCCs with constraint lengths $\nu = 1, 2, 3$, and 4 whose constituent encoders are shown in Table 16.6. With the exception of code E, each of the generators was chosen to have a primitive feedback polynomial so as to achieve the best possible minimum free distance. The generator for code E has a nonprimitive feedback polynomial. The results of the $P_b(E)$ bound of

TABLE 16.6: Code parameters for Figures 16.13 and 16.14.

Code	Generator matrix $\mathbf{G}(D)$
A	$[1 \quad 1/(1+D)]$
B	$[1 \quad (1+D^2)/(1+D+D^2)]$
C	$[1 \quad (1+D+D^2+D^3)/(1+D+D^3)]$
D	$[1 \quad (1+D+D^2+D^4)/(1+D^3+D^4)]$
E	$[1 \quad (1+D^4)/(1+D+D^2+D^3+D^4)]$

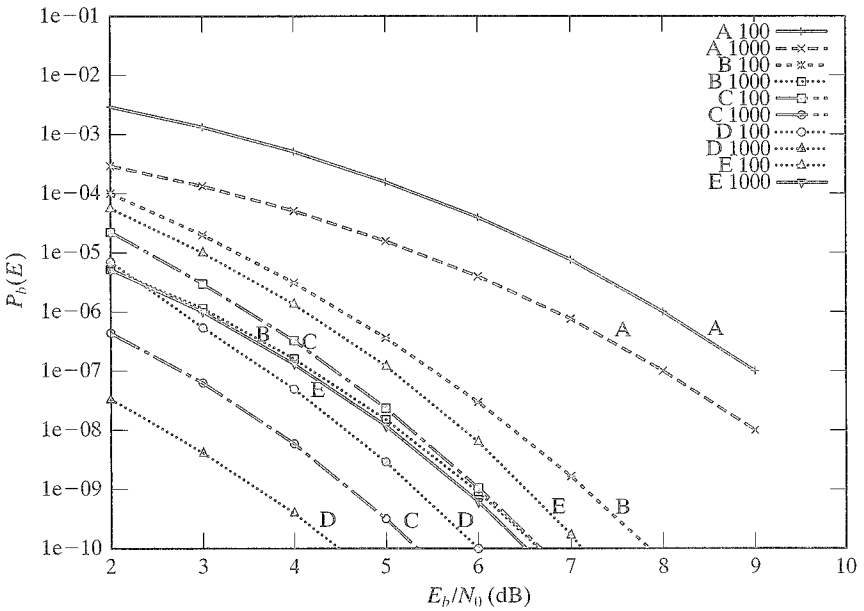


FIGURE 16.13: Bit-error probability bounds for rate $R = 1/3$ PCCCs with $\nu = 1, 2, 3,$ and 4 and $K = 100$ and 1000.

(16.82) for each code are shown in Figure 16.13 for two values of the interleaver size (information block length): $K = 100$ and 1000. In addition, the simulated performance of the two 16-state ($\nu = 4$) codes, codes D and E, is shown in Figure 16.14 for a pseudorandom interleaver of size $K = 2^{10} = 1024$ and 18 iterations of decoding.

The following remarks apply to Example 16.10

- The bounds in Figure 16.13 indicate that substantial coding gains can be achieved at $P_b(E)$'s below 10^{-5} by increasing the constraint length ν of the constituent code. For example, for $P_b(E) = 10^{-8}$ and a block length of $K = 1000$, the 16-state ($\nu = 4$) primitive constituent code (code D) gains 6.5 dB compared with the 2-state ($\nu = 1$) constituent code (code A).

- Selecting a larger constraint length can be used as a substitute for a larger block length if decoding delay is an issue. For example, the 16-state primitive code (code D) with $K = 100$ gains 0.5 dB compared with the 4-state code (code B) with $K = 1000$ for $P_b(E) = 10^{-7}$.

- The bounds in Figure 16.13 indicate that the 16-state primitive constituent code (code D) performs better than the 16-state nonprimitive code (code E), but the simulations in Figure 16.14 show that this is true only at high SNRs. This result illustrates that the bounds accurately predict performance only at SNRs above the cutoff rate limit, 2 dB in this case. (Note that the simulation

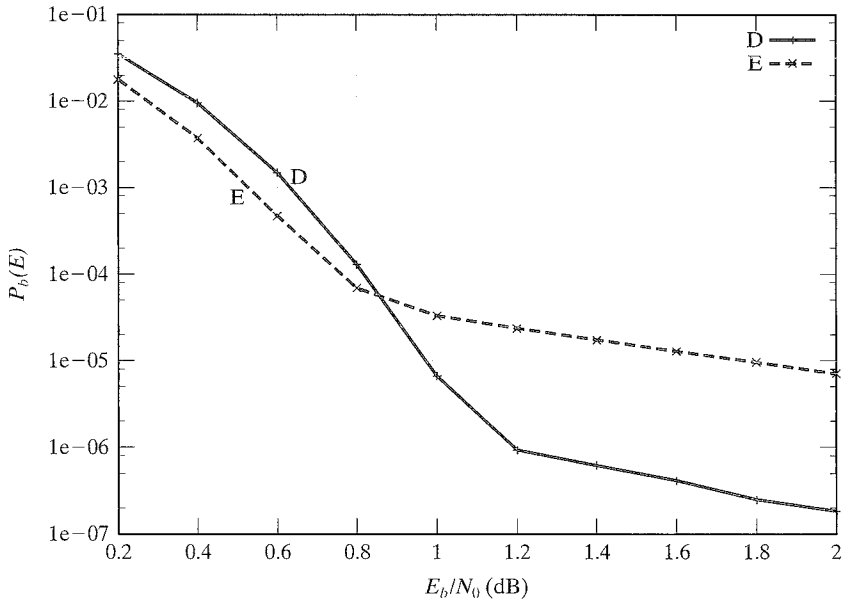


FIGURE 16.14: Bit-error probability simulations for two rate $R = 1/3$ PCCCs with $\nu = 4$ and $K = 1024$.

results in Figure 16.14 show the performance of these two codes at a lower SNR range than the bounds in Figure 16.13).

- The superior performance of the 16-state primitive code (code D) at higher SNRs is due to its larger free distance (see Problem 16.15).

EXAMPLE 16.11 Performance of Recursive and Nonrecursive PCCCs

Finally, consider the two rate $R = 1/3$ PCCCs generated by the nonsystematic feedforward encoder of Example 16.8 and by the systematic feedback encoder of Examples 16.6 and 16.7. As we have noted previously, the constituent encoders by themselves generate identical rate $R = 1/2$ codes. The approximate bit IOWEF of the nonrecursive PCCC is given in (16.75b). For large K, the free distance of this code is $d_{free} = 7$. An approximation to the bit WEF is given in (16.75d). In Figure 16.15 we show the $P_b(E)$ bound obtained by substituting this expression into (16.82), along with the corresponding bound for the recursive PCCC from Examples 16.6 and 16.7, shown previously in Figure 16.12(b). The interleaver size (information block length) is $K = 1000$. Also plotted in the figure are the $P_b(E)$ bounds for the unterminated rate $R = 1/3$ convolutional codes obtained by simply repeating the second output of each $R = 1/2$ code without permuting the input sequence. These latter curves represent the equivalent $R = 1/3$ performance of the two constituent codes (nonrecursive and recursive) without parallel concatenation.

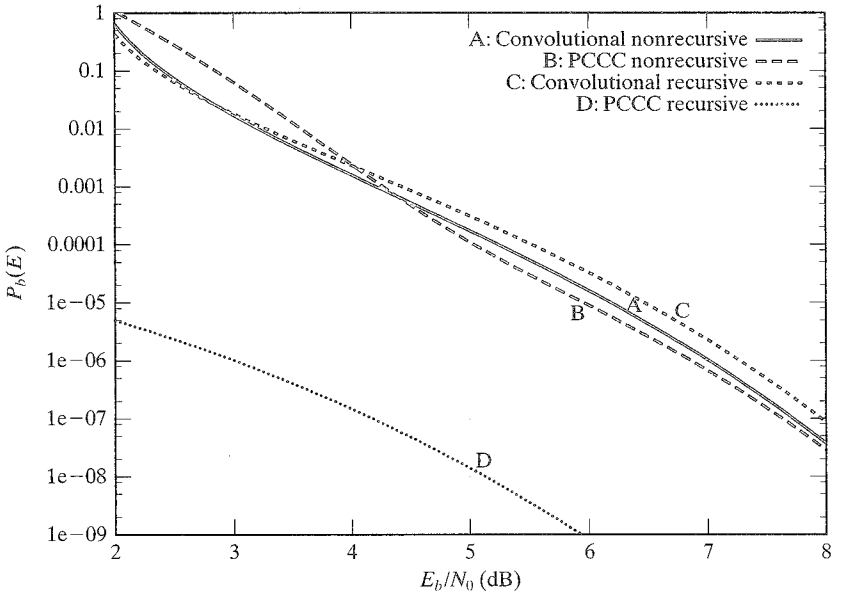


FIGURE 16.15: Bit-error probability bounds for nonrecursive and recursive PCCCs and convolutional codes with rate $R = 1/3$.

We now conclude this section with some comments regarding Example 16.11.

- The $P_b(E)$ curves for the unterminated convolutional codes without parallel concatenation (curves A and C) indicate that the performance of the two codes is almost the same. This is to be expected, since the two encoders generate identical codes, and the only difference in their BERs is a consequence of the different mappings between information sequences and codewords.
- For the nonrecursive codes parallel concatenation (curves A and B) produces little improvement, because there is no interleaver gain for feedforward encoders.
- For the recursive codes, on the other hand, parallel concatenation results in substantial coding gain (curves C and D). For example, for $P_b(E) = 10^{-6}$, the recursive PCCC with $K = 1000$ achieves a 4.3-dB coding gain compared with the rate $R = 1/3$ equivalent code without parallel concatenation.
- The superiority of employing systematic feedback encoders rather than nonsystematic feedforward encoders as constituent codes in a PCCC can clearly be seen by comparing curves B and D. For example, for $P_b(E) = 10^{-6}$, the recursive PCCC gains 3.8 dB compared with the nonrecursive PCCC.

In the next section we continue our analysis of turbo codes by examining the parameters that provide the best constituent code designs.

16.4 DESIGN OF TURBO CODES

For SNRs (values of E_b/N_0) above the cutoff rate limit of (16.83), our discussion of the design of PCCCs is based on the upper bound on bit-error probability derived in the previous section. (For smaller SNRs, considered later in this section, the dynamics of iterative decoding have a larger influence on code design than error probability bounds.) We begin by using (16.25a) and (16.26b) to express the bit WEF $B(X)$ of a systematic code as a sum over input weights w as follows:

$$B(X) = \sum_{(1 \le w \le K)} \frac{w}{K} W^w A_w(Z)\Big|_{W=Z=X}. \tag{16.84}$$

Then, using the looser form of the bound of (16.82) (i.e., without the function f(·), a scaling factor that depends only on d_{free}) and (16.84), we can approximate the bound on the bit-error probability of a PCCC as follows:

$$P_b(E) \approx \sum_{(w_{min} \le w \le K)} \frac{w}{K} W^w A_w(Z)\Big|_{W=Z=e^{-RE_b/N_0}}, \tag{16.85}$$

where w_{min} is the minimum-weight nonzero valid input sequence, that is, the minimum-weight nonzero input sequence that terminates the encoder. Now, using the approximation for large K of (16.55a), we obtain

$$\begin{aligned} P_b(E) &\approx \sum_{(w_{min} \le w \le K)} \frac{w}{K} W^w \frac{w!}{(h_{max}!)^2} K^{(2h_{max}-w)} \left[A_w^{(h_{max})}(Z)\right]^2\Big|_{W=Z=e^{-RE_b/N_0}} \\ &= \sum_{(w_{min} \le w \le K)} w \frac{w!}{(h_{max}!)^2} K^{(2h_{max}-w-1)} W^w \left[A_w^{(h_{max})}(Z)\right]^2\Big|_{W=Z=e^{-RE_b/N_0}}. \end{aligned} \tag{16.86}$$

We now note that for $(n, 1, \nu)$ systematic constituent encoders, the resulting PCCC has rate

$$R = \frac{K-\nu}{(2n-1)K} \approx \frac{1}{2n-1} \quad \text{(for large } K\text{)}. \tag{16.87}$$

We next consider nonrecursive PCCCs with $(n, 1, \nu)$ systematic feedforward constituent encoders. In this case, $w_{min} = 1$, since only zeros are required to terminate an information sequence for a feedforward encoder. Also, as noted in the previous section, $h_{max} = w$ for feedforward encoders. Finally, since for any $(n, 1, \nu)$ systematic feedforward encoder $A_w^{(w)}(Z) = [A_1^{(1)}(Z)]^w$ (see Problem 16.16), we obtain from (16.86) for *nonrecursive PCCCs* with $(n, 1, \nu)$ *systematic feedforward constituent encoders* and large K

$$P_b(E) \approx \sum_{(1 \le w \le K)} \frac{K^{(w-1)}}{(w-1)!} W^w \left[A_1^{(1)}(Z)\right]^{2w}\Big|_{W=Z=e^{-RE_b/N_0}}. \tag{16.88}$$

From (16.88) we see that, even in the best case ($w = 1$), $P_b(E)$ does not decrease with increasing K; that is, no interleaver gain is possible with feedforward encoders.

A similar analysis applies to PCBCs with (hn, hk) block constituent codes and an interleaver size of $K = hk$.

We now consider recursive PCCCs with $(n, 1, \nu)$ systematic feedback constituent encoders. In this case, $w_{min} = 2$, since a weight-1 input sequence cannot terminate the encoder, but there always exists a weight-2 input sequence, of degree no greater than $2^\nu - 1$, that does terminate the encoder (see Problem 16.17). Also, since each single-error event in a multiple-error event requires at least two input 1's to terminate, $h_{max} = \lfloor w/2 \rfloor$ for feedback encoders. We now note that for the term involving the block length K in (16.86)

$$K^{(2h_{max}-w-1)} = \begin{cases} K^{-2} & \text{if } w \text{ is odd} \\ K^{-1} & \text{if } w \text{ is even.} \end{cases} \tag{16.89}$$

Thus, for large K, the terms with odd w are negligible compared with the terms with even w, and in (16.86) we need only consider terms of the type $A_{2w}^{(w)}(Z)$. Further, since for any $(n, 1, \nu)$ systematic feedback encoder $A_{2w}^{(w)}(Z) = \left[A_2^{(1)}(Z)\right]^w$ (see Problem 16.16), we must examine the properties of the term $A_2^{(1)}(Z)$, that is, the parity weight enumerator for single-error events with input weight 2. We begin by considering an example.

EXAMPLE 16.12 Calculating $A_2^{(1)}(Z)$ for a $(2, 1, 2)$ Systematic Feedback Encoder

Consider the state diagram of the $(2, 1, 2)$ systematic feedback encoder of Examples 16.6 and 16.7 shown in Figure 16.10. In this case, it is clear that the shortest single-error event generated by a weight-2 input follows the state sequence $S_0 \to S_1 \to S_3 \to S_2 \to S_0$ and has parity weight 4. All other single-error events with input weight 2 are generated by adding one or more cycles around the loop $S_2 \to S_1 \to S_3 \to S_2$ to the foregoing shortest state sequence. Because each of these cycles adds parity weight 2,

$$A_2^{(1)}(Z) = Z^4 + Z^6 + Z^8 + \cdots = \frac{Z^4}{1 - Z^2}. \tag{16.90}$$

In general, for any $(n, 1, \nu)$ systematic feedback encoder, the input sequence $\mathbf{u} = 1000\cdots$ will produce a cycle in the state diagram with input weight 0 that starts in state $S_1 = (10\cdots0)$, arrives in state $S_{2^{\nu-1}} = (0\cdots01)$ after at most $2^\nu - 2$ steps, and then returns to state S_1 in one step (see Problem 16.18). Let z_c be the parity weight generated by this cycle. Also, it is easy to show that a 1 input from state $S_{2^{\nu-1}} = (0\cdots01)$ terminates the encoder (see Problem 16.19). Further, as in Example 16.12, if each of the $n - 1$ numerator polynomials has the form $1 + \cdots + D^\nu$, that is, they are monic polynomials of degree ν, then the zero-input branch connecting state $S_{2^{\nu-1}} = (0\cdots01)$ to state $S_1 = (10\cdots0)$ has zero parity weight. In this case, if z_t is the total parity weight of the two input weight-1 branches that connect the initial and final states to the zero-input weight cycle, the total parity weight associated with any input weight-2 single-error event is $jz_c + z_t$, where j is the number of zero-input weight cycles included in the error event, and

$$z_{min} = z_c + z_t \tag{16.91}$$

is the minimum parity weight associated with a weight-2 input. (If the preceding condition on the numerator polynomials is not satisfied, the branch connecting state $S_{2^{\nu-1}} = (0\cdots01)$ to state $S_1 = (10\cdots0)$ may have nonzero parity weight. In this case, since this branch is not included in the path that achieves minimum parity weight, its parity weight must be subtracted from the right side of (16.91).) Thus the parity weight enumerator for single-error events with input weight 2 is given by

$$A_2^{(1)}(Z) = Z^{z_{min}} + Z^{(2z_{min}-z_t)} + Z^{(3z_{min}-2z_t)} + \cdots = \frac{Z^{z_{min}}}{1 - Z^{(z_{min}-z_t)}}. \tag{16.92}$$

EXAMPLE 16.12 (Continued)

For the encoder shown in Figure 16.10, $z_c = 2$, $z_t = 2$, and $z_{min} = 4$.

Using (16.86), we can now state the approximate bit-error probability bound for large K for *recursive PCCCs* with $(n, 1, \nu)$ *systematic feedback constituent encoders* as follows:

$$P_b(E) \approx \sum_{(1\le w\le \lfloor K/2\rfloor)} 2w \binom{2w}{w} K^{-1} W^{2w} \left[\frac{Z^{z_{min}}}{1 - Z^{(z_{min}-z_t)}}\right]^{2w}\Bigg|_{W=Z=e^{-RE_b/N_0}}$$

$$= \sum_{(1\le w\le \lfloor K/2\rfloor)} 2w \binom{2w}{w} K^{-1} \frac{\left[Y^{(2+2z_{min})}\right]^w}{\left[1 - Y^{(z_{min}-z_t)}\right]^{2w}}\Bigg|_{Y=e^{-RE_b/N_0}}. \tag{16.93}$$

We see that, unlike (16.88) for feedforward encoders, (16.93) contains the term K^{-1}, indicating the presence of interleaver gain for feedback encoders. Also, the exponential behavior of the most significant term in the bound is determined by z_{min}, the minimum parity weight associated with weight-2 input sequences. Thus, to guarantee good performance for large K, z_{min} should be as large as possible. Finally, since z_{min} depends primarily on z_c, the parity weight in the zero-input weight cycle should be as large as possible. This implies that the cycle length should be as large as possible, which means that a primitive denominator polynomial should be chosen to achieve a large z_{min} (see Problem 16.20).

Because the weight $2 + 2z_{min}$ is the coefficient of RE_b/N_0 in the most significant term of the bound, we define the *effective minimum free distance* of a rate $R = 1/(2n - 1)$ PCCC with an $(n, 1, \nu)$ constituent encoder as

$$d_{eff} = 2 + 2z_{min}. \tag{16.94}$$

EXAMPLE 16.12 (Continued)

For the rate $R = 1/3$ PCCC based on the encoder shown in Figure 16.10, $d_{eff} = 10$.

EXAMPLE 16.13 Calculating d_{eff} for PCCCs

Consider the rate $R = 1/3$ recursive PCCCs generated by the following (2, 1, 3) systematic feedback encoders:

$$\mathbf{G}_p(D) = \begin{bmatrix} 1 & (1 + D + D^2 + D^3)/(1 + D + D^3) \end{bmatrix} \tag{16.95a}$$

and

$$\mathbb{G}_{np}(D) = \begin{bmatrix} 1 & (1 + D + D^3)/(1 + D + D^2 + D^3) \end{bmatrix}, \tag{16.95b}$$

where $\mathbb{G}_p$ has a primitive feedback polynomial, and $\mathbb{G}_{np}$ has a nonprimitive feedback polynomial. The rate $R = 1/2$ convolutional codes generated by these two encoders are identical. Both have free distance $d_{free} = 6$. For the primitive encoder, the shortest terminating weight-2 input sequence is $\mathbb{u}(D) = 1 + D^7$, which generates the parity sequence $\mathbb{v}^{(1)}(D) = 1 + D^2 + D^3 + D^4 + D^6 + D^7$. Thus, $z_{min} = 6$, and $d_{eff} = 14$ for the primitive PCCC. For the nonprimitive encoder, the shortest terminating weight-2 input sequence is $\mathbb{u}(D) = 1 + D^4$, which generates the parity sequence $\mathbb{v}^{(1)}(D) = 1 + D^2 + D^3 + D^4$, and for the nonprimitive PCCC $z_{min} = 4$, and $d_{eff} = 10$.

In Example 16.13, as expected, d_{eff} is larger for the PCCC based on a primitive feedback polynomial. Because the input sequence $\mathbb{u}(D) = 1 + D^2 + D^3 + D^4$ generates the parity sequence $\mathbb{v}^{(1)}(D) = 1 + D^4$ for the primitive encoder, the actual minimum free distance of the corresponding PCCC, assuming the uniform interleaver analysis, is $d_{free} = 8$, smaller than for the nonprimitive encoder; however, since this input sequence has weight 4 and produces a single-error event, the multiplicity associated with the weight-8 codewords decreases with K^2, and thus the probability that a particular interleaver will result in a weight-8 codeword in the corresponding PCCC is very small for large K.

THEOREM 16.1 [12] For an $(n, 1, \nu)$ systematic feedback encoder with generator matrix

$$\mathbb{G}(D) = \begin{bmatrix} 1 & n_1(D)/d(D) & \cdots & n_{n-1}(D)/d(D) \end{bmatrix}, \tag{16.96}$$

such that the denominator polynomial $d(D)$ is a primitive polynomial of degree ν, and the numerator polynomials $n_j(D) \neq d(D)$, $1 \leq j \leq n-1$, the minimum parity weight associated with weight-2 input sequences satisfies

$$z_{min} = s(2^{\nu-1} + 2) + (n - s - 1)\left(2^{\nu-1}\right) \leq (n-1)(2^{\nu-1} + 2), \tag{16.97}$$

where s, $0 \leq s \leq n-1$, is the number of numerator polynomials that are monic and of degree ν.

Proof. We begin by noting that since $d(D)$ is primitive, it is the generator polynomial of a cyclic $(2^\nu - 1, 2^\nu - \nu - 1)$ Hamming code. Also, the minimum-degree *binomial* that has $d(D)$ as a factor is $D^{2^\nu - 1} + 1$, and the quotient polynomial $q(D) = (D^{2^\nu - 1} + 1)/d(D)$ is the generator polynomial of a cyclic $(2^\nu - 1, \nu)$ maximum-length block code.

Next, we consider numerator polynomials $n(D)$ such that $\deg[n(D)] < \nu$. In this case, the weight-2 input sequence $D^{2^\nu - 1} + 1$ generates the parity sequence $n(D)q(D)$, which has a weight of exactly $2^{\nu-1}$, since it is a nonzero codeword in a $(2^\nu - 1, \nu)$ maximum-length code. Also, if $n(D)$ has degree ν but is not monic, it can be expressed as $n(D) = D^r v(D)$, $1 \leq r \leq \nu$, where

$v(D)$ is monic and has degree less than ν. In this case, the parity sequence $n(D)q(D) = D^r v(D)q(D)$ again has weight $2^{\nu-1}$.

Now, consider numerator polynomials $n(D)$ that are monic and of degree ν; that is, $n(D) = 1 + \cdots + D^\nu$. In this case, we can write

$$n(D) = Dn^{(1)}(D) + n^{(2)}(D), \tag{16.98}$$

where $n^{(1)}(D) = D^{\nu-1}$ and $n^{(2)}(D) = n(D) - D^\nu$ both have degree less than ν. Thus, $c^{(1)}(D) \triangleq n^{(1)}(D)q(D)$ and $c^{(2)}(D) \triangleq n^{(2)}(D)q(D)$ are both nonzero codewords in a $(2^\nu - 1, \nu)$ maximum-length code and have weight $2^{\nu-1}$. We note that $c^{(1)}(D)$ has the maximum degree of $2^\nu - 2$ for any code polynomial and $\nu - 1$ leading zeros; that is, $c^{(1)}(D) = D^{\nu-1} + c_\nu^{(1)} D^\nu + \cdots + c_{2^\nu-3}^{(1)} D^{2^\nu-3} + D^{2^\nu-2}$, and $c^{(2)}(D)$ has a nonzero constant term. We can now write the cyclic shift $\mathcal{X}^{(1)}(D)$ of $c^{(1)}(D)$ as

$$\begin{aligned} \mathcal{X}^{(1)}(D) &= 1 + D^\nu + c_\nu^{(1)} D^{\nu+1} + \cdots + c_{2^\nu-3}^{(1)} D^{2^\nu-2} \\ &= Dc^{(1)}(D) + 1 + D^{2^\nu-1}. \end{aligned} \tag{16.99}$$

Because $\mathcal{X}^{(1)}(D)$ and $c^{(2)}(D)$ are both codewords, $\mathcal{X}^{(1)}(D) + c^{(2)}(D)$ is also a codeword, has a zero constant term, and has weight $2^{\nu-1}$. Now, we can write the parity sequence as

$$\begin{aligned} n(D)q(D) &= Dc^{(1)}(D) + c^{(2)}(D) \\ &= \mathcal{X}^{(1)}(D) + c^{(2)}(D) + 1 + D^{2^\nu-1}. \end{aligned} \tag{16.100}$$

Because $\mathcal{X}^{(1)}(D) + c^{(2)}(D)$ has weight $2^{\nu-1}$, a zero constant term, and degree of at most $2^\nu - 2$, the parity sequence $n(D)q(D)$ has a weight of exactly $2^{\nu-1} + 2$. The theorem now follows by defining s as the number of numerator polynomials that are monic and of degree ν. Q.E.D.

In Problem 16.21 it is shown that the upper bound in (16.97) also holds for nonprimitive feedback polynomials. Thus, since for primitive feedback polynomials Theorem 16.1 guarantees that the upper bound can be achieved by properly selecting the numerator polynomials, we can achieve the best performance for large K, that is, the largest effective free distance d_{eff}, by choosing constituent codes with primitive feedback polynomials and monic numerator polynomials of degree ν. Selecting constituent codes that result in large values of d_{eff} guarantees good high-SNR performance. For lower SNRs, however, codeword multiplicities, additional terms in the weight spectrum, and the dynamics of iterative decoding must be considered.

In searching for PCCCs with systematic feedback constituent encoders that offer good performance over the entire SNR range, it is important to maximize z_{min}, and thus the effective free distance d_{eff} of the PCCC, and to minimize the multiplicities of the minimum-weight codewords produced by each of the lowest-weight input sequences, that is, input weights $w = 2, 3, 4, \cdots$. We denote the minimum weight of codewords produced by weight-w input sequences in an encoder by d_w, and the number of such codewords by A_{w,d_w}, the coefficient

TABLE 16.7: Systematic feedback convolutional encoders with optimum-weight spectrum.

(a) LOW RATE ENCODERS

k/n	ν	$\mathbb{g}^{(3)}$	$\mathbb{g}^{(2)}$	$\mathbb{g}^{(1)}$	$\mathbb{g}^{(0)}$	d_2, A_{2,d_2}	d_3, A_{3,d_3}	d_4, A_{4,d_4}	d_5, A_{5,d_5}
1/4	1	2	3	1	3	6,1	∞	∞	∞
1/4	2	6	7	5	7	10,1	10,1	14,2	16,1
1/4	3	11	17	15	13	20,1	12,1	14,1	18,2
1/4	4	27	37	35	23	32,1	16,1	14,1	18,3
1/4	5	71	45	51	67	56,1	23,1	20,3	15,1
1/3	1	—	2	3	3	5,1	∞	∞	∞
1/3	2	—	5	7	7	8,1	8,1	10,1	12,1
1/3	3	—	17	15	13	14,1	10,2	10,1	14,6
1/3	4	—	33	37	23	22,1	12,1	10,1	12,1
1/3	5	—	45	51	67	38,1	17,2	16,6	11,1
1/3	6	—	101	131	163	70,1	23,1	14,1	11,1
1/2	1	—	—	2	3	3,1	∞	∞	∞
1/2	2	—	—	5	7	6,1	5,1	6,1	7,1
1/2	3	—	—	17	13	8,1	7,3	6,1	9,9
1/2	4	—	—	37	23	12,1	8,3	6,1	10,14
1/2	5	—	—	45	67	20,1	10,2	8,1	8,2
1/2	6	—	—	101	147	36,1	13,1	8,2	7,1
1/2	6*	—	—	115	147	36,1	13,3	10,2	9,2

(b) HIGH RATE ENCODERS

k/n	ν	$\mathbb{h}^{(3)}$	$\mathbb{h}^{(2)}$	$\mathbb{h}^{(1)}$	$\mathbb{h}^{(0)}$	d_2, A_{2,d_2}	d_3, A_{3,d_3}	d_4, A_{4,d_4}	d_5, A_{5,d_5}
2/3	1	—	2	2	3	2,1	∞	6,4	∞
2/3	2	—	6	5	7	4,3	3,1	4,1	5,1
2/3	3	—	15	17	13	5,1	4,1	5,1	6,6
2/3	4	—	27	35	23	8,2	5,3	6,9	7,29
2/3	5	—	73	51	75	12,2	6,3	6,3	6,5
2/3	6	—	135	133	147	20,2	8,4	6,3	6,2
3/4	1	2	2	2	3	2,3	∞	5,6	∞
3/4	2	6	2	7	5	3,3	3,3	4,7	5,15
3/4	3	17	13	15	11	4,1	4,3	4,1	5,5
3/4	4	33	31	27	25	6,2	4,1	5,8	5,4
3/4	5	63	51	71	45	9,2	5,2	5,5	5,2

Adapted from [34].

of the lowest-order term in the conditional IOWEF $A_w(X)$. (Note that in this case $d_{eff} = 2d_2 - 2$.) Table 16.7 lists optimum-weight spectrum (n, k, ν) systematic feedback convolutional encoders for rates $R = 1/4, 1/3, 1/2, 2/3$, and $3/4$. The optimization was performed by first maximizing d_w and then minimizing A_{w,d_w}, successively, for increasing values of w. For each encoder, we list the rate $R = k/n$, the constraint length ν, the (right-justified) octal generator sequences $\mathbb{g}^{(0)}$, $\mathbb{g}^{(1)}$, $\mathbb{g}^{(2)}$,

and $\mathbf{g}^{(3)}$ in Table 16.7(a) for rates $R = 1/4$, $1/3$, and $1/2$ and the (right-justified) octal parity-check sequences $\mathbf{h}^{(0)}$, $\mathbf{h}^{(1)}$, $\mathbf{h}^{(2)}$, and $\mathbf{h}^{(3)}$ in Table 16.7(b) for rates $R = 2/3$ and $3/4$, and the pair (d_w, A_{w,d_w}) for $w = 2$, 3, 4, and 5. In the table, $\mathbf{g}^{(0)}$ represents the denominator (feedback) polynomial of a systematic feedback encoder generator matrix $\mathbf{G}(D)$, and $\mathbf{g}^{(1)}$, $\mathbf{g}^{(2)}$, and $\mathbf{g}^{(3)}$ represent the numerator polynomials. For example, for the rate $R = 1/3$, 8-state (3, 1, 3) encoder, $\mathbf{g}^{(0)} = 13$, $\mathbf{g}^{(1)} = 15$, $\mathbf{g}^{(2)} = 17$, and we have

$$\begin{aligned}\mathbf{G}(D) &= \left[\begin{array}{cc} 1 & \mathbf{g}^{(1)}(D)/\mathbf{g}^{(0)}(D) \end{array}\right. \quad \left.\begin{array}{c} \mathbf{g}^{(2)}(D)/\mathbf{g}^{(0)}(D) \end{array}\right] \\ &= \left[\begin{array}{cc} 1 & (1+D^2+D^3)/(1+D+D^3) \end{array}\right. \quad \left.\begin{array}{c} (1+D+D^2+D^3)/(1+D+D^3) \end{array}\right].\end{aligned} \tag{16.101}$$

Similarly, in Table 16.7(b), $\mathbf{h}^{(0)}$ represents the denominator (feedback) polynomial of a systematic feedback encoder parity-check matrix $\mathbf{H}(D)$ and $\mathbf{h}^{(1)}$, $\mathbf{h}^{(2)}$, and $\mathbf{h}^{(3)}$ represent the numerator polynomials. For example, for the rate $R = 2/3$, 8-state (3, 2, 3) encoder, $\mathbf{h}^{(0)} = 13$, $\mathbf{h}^{(1)} = 17$, $\mathbf{h}^{(2)} = 15$ and we have

$$\begin{aligned}\mathbf{H}(D) &= [\mathbf{h}^{(2)}(D)/\mathbf{h}^{(0)}(D) \quad \mathbf{h}^{(1)}(D)/\mathbf{h}^{(0)}(D) \quad 1] \\ &= [(D^3+D^2+1)/(D^3+D+1) \quad (D^3+D^2+D+1)/(D^3+D+1) \quad 1].\end{aligned} \tag{16.102}$$

The bold entries in the table denote the minimum free distance d_{free} of the code, the symbol ∞ indicates that there are no codewords corresponding to that input weight (only single-error events are considered), and the $*$ entry indicates the existence of an encoder with the same d_2 and a larger d_{free}.

The following remarks pertain to these optimum-weight spectrum encoders.

- Many of the encoders in Table 16.7 do not give optimum d_{free} codes, as can be seen by comparison with the code lists in Chapter 12; however, the encoders in Table 16.7 are optimum in the sense of minimizing the number of low-weight codewords corresponding to low-weight input sequences.
- The encoders in Table 16.7 can be used to produce a variety of PCCC code rates, as seen in Table 16.8. Other code rates can be achieved by puncturing,

TABLE 16.8: PCCC code rates for different constituent encoder rates.

Constituent encoder rate	PCCC code rate
1/4	1/7
1/3	1/5
1/2	1/3
2/3	2/4
3/4	3/5

by using constituent encoders of different rates, or by employing multiple interleavers (see Figure 16.2).

- It is important to note that in examining the weight spectrum of PCCCs, the values of interest are given by the lowest-order coefficients in the conditional IOWEFs $A_w(X)$ for small values of w. This contrasts with the case for normal convolutional codes and block codes, in which the low-order coefficients in the standard WEF $A(X)$ are the values of interest. This is because the interleaver in PCCCs has a different effect on input sequences of different weights, thus accentuating the importance of the codewords produced by low-weight input sequences.

We now summarize the major points to remember in designing good PCCCs for high SNRs:

- The bit multiplicities in a PCCC include a factor of K^{1-w}, where K is the interleaver size, w is the weight of an input sequence that produces a terminated codeword, and $w \geq w_{min}$, the minimum-weight input sequence that produces a terminated codeword.

- All feedback encoders have $w_{min} = 2$, resulting in an interleaver gain factor of $1/K$ for the associated PCCCs. In contrast, $w_{min} = 1$ for feedforward encoders and for repeated block codes, so there is no interleaver gain in these cases, although some spectral thinning does occur.

- For large values of K and high SNRs, the performance of a PCCC is optimized by maximizing its effective free distance $d_{eff} = 2 + 2z_{min}$, where z_{min} is the minimum parity weight corresponding to a weight-2 input sequence in the constituent encoder.

- For $(n, 1, \nu)$ systematic feedback convolutional encoders, it is possible to achieve $z_{min} = (n-1)(2^{\nu-1}+2)$ by choosing a primitive denominator polynomial and monic numerator polynomials of degree ν.

- For SNRs closer to the cutoff rate limit, good performance is achieved by optimizing the low-input weight terms in the IOWEF of the constituent encoder.

- These PCCC design rules are based on an average (uniform interleaver) analysis, and performance can vary in a particular case depending on the choice of an interleaver. Thus, it is important to support a code design with appropriate computer simulations.

In addition, although pseudorandom interleavers such as the ones discussed earlier in this chapter give very good performance on average, it is possible to design the interleaver to improve the value of d_{eff} and thus the high-SNR performance of the code. A codeword of weight d_{eff} is produced when a weight-2 input sequence that generates the weight-z_{min} parity sequence in the first constituent code is interleaved in such a way that the interleaved input sequence maintains the same spacing between 1's, thus also generating the weight-z_{min} parity sequence in the

second constituent code. If we refer to a weight-2 input sequence that generates the weight-z_{min} parity sequence in the first constituent code as a "bad" input sequence, interleaver designs that "break up" all bad input sequences, that is, they change the spacing between 1's, result in a larger value of d_{eff}; however, putting too much structure in the interleaver can result in low-weight codewords owing to higher-weight input sequences; that is, the ability of a pseudorandom interleaver to reduce the influence of higher-weight input sequences is mitigated when structure is added. Interleaver designs that attempt to improve d_{eff} by breaking up the bad input sequences while also maintaining most of the characteristics of pseudorandom interleavers are referred to as *semirandom* (or *S-random*) *interleavers* [35]. For example, in an S-random interleaver, integers i in the range $1 \leq i \leq K$ are chosen randomly (without replacement) to denote the interleaved positions of successive bits in the input sequence. If a selected integer is within a distance $\pm S$ of any of the S previous selections, it is rejected and another integer is selected. Choosing S equal to the cycle length of the denominator polynomial guarantees that all bad weight-2 input sequences are broken up; however, the algorithm is not guaranteed to successfully assign all K integers, so that the final permutation may have to be altered further to satisfy the spreading condition.

The preceding rules for designing good PCCCs implicitly assume the use of (optimum) maximum likelihood decoding. Iterative decoding, in contrast, is suboptimum, and different code design rules apply in this case. At moderate-to-high SNRs, iterative decoding is essentially maximum likelihood, and the preceding design rules will give the best codes. At low SNRs, below the channel cutoff rate, however, the dynamics of iterative decoding are more important in determining the performance of PCCCs than the weight spectrum of the code.

To explain the dynamics of iterative decoding, it is helpful to introduce the *extrinsic information transfer chart*, or EXIT chart [36], based on the techniques of density evolution [37, 38] and threshold analysis [39]. These techniques relate a parameter of the input to a constituent decoder, either the *signal-to-noise ratio* of the a priori L-value $L_a(u_l)$ or the *mutual information* between an information bit u_l and its a priori L-value $L_a(u_l)$, to a parameter of the decoder output, the signal-to-noise ratio of the a posteriori extrinsic L-value $L_e(u_l)$ or the mutual information between u_l and its a posteriori extrinsic L-value $L_e(u_l)$, respectively. Following the mutual-information approach, we first model the a priori L-value inputs to a constituent decoder as independent Gaussian random variables with variance σ_a^2 and mean $\mu_a = \pm\sigma_a^2/2$, where the sign of μ_a depends on the transmitted value of u_l. This *Gaussian approximation* is based on two facts: (1) the input channel L-values to a constituent decoder (the terms $L_c r_l^{(0)}$ in (16.17)) are independent Gaussian random variables with variance $2L_c$ and mean $\pm L_c$ (see Problem 16.23), and (2) extensive simulations of the a posteriori extrinsic L-values $L_e(u_l)$ for a constituent decoder with very large block lengths support this assumption [40]. Thus, we can express the mutual information $I_a[u_l; L_a(u_l)]$ between an information bit u_l and its a priori L-value $L_a(u_l)$ as (see [41])

$$I_a[u_l; L_a(u_l)] = \frac{1}{2} \sum_{u_l=-1,+1} \int_{-\infty}^{+\infty} P_{L_a}(\xi|u_l) \log_2 \frac{2P_{L_a}(\xi|u_l)}{P_{L_a}(\xi|u_l=-1) + P_{L_a}(\xi|u_l=+1)}\, d\xi, \tag{16.103a}$$

where the conditional probability density function $P_{L_a}(\xi|u_l)$ of $L_a(u_l)$ is given by

$$P_{L_a}(\xi|u_l) = \frac{1}{\sqrt{2\pi\sigma_a^2}} e^{-(\xi - \frac{\sigma_a^2}{2}u_l)^2/2\sigma_a^2}. \tag{16.103b}$$

Next, histograms of the conditional pdf's $P_{L_e}(\xi|u_l = -1)$ and $P_{L_e}(\xi|u_l = +1)$ associated with the a posteriori extrinsic L-values $L_e(u_l)$ are determined by simulating the BCJR algorithm with independent Gaussian-distributed a priori L-value inputs for a particular constituent code and a large block length K. We then determine the mutual information $I_e[u_l; L_e(u_l)]$ between the information bit u_l and its a posteriori extrinsic L-value $L_e(u_l)$ using (16.103a).

We can now generate a decoder input–output transfer curve by running a series of simulations of the BCJR algorithm for one constituent code and different values of $I_a[u_l; L_a(u_l)]$ (completely determined by σ_a^2), for a fixed channel signal-to-noise ratio E_b/N_0 and plotting the resulting values of $I_e[u_l; L_e(u_l)]$. Note that each point on the curve is obtained by simulating the BCJR algorithm just once for a constituent code and does not require iterative decoding. A set of transfer curves is obtained by repeating this procedure for different values of the channel SNR, E_b/N_0. An example is shown in Figure 16.16 for the original turbo code of Figure 16.1(b), with alternate parity bits punctured to produce a code rate of $R = 1/2$, and various values of the channel SNR. Note that a value of $I_a = 0$ corresponds to the first iteration of decoding when all a priori L-values are equal to 0 and that more positive values of I_a correspond to more reliable a priori estimates. A key observation is that if a transfer curve cuts below the straight line representing $I_a = I_e$, this corresponds

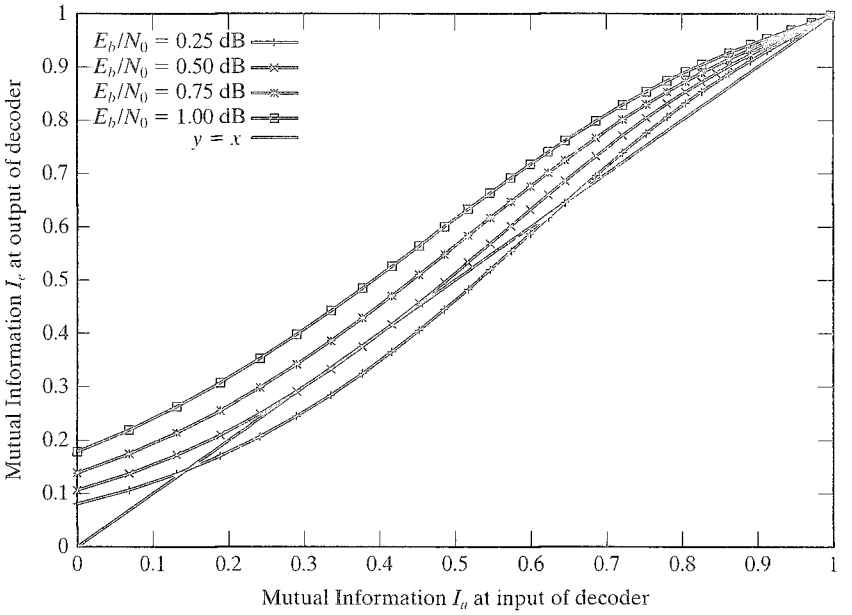


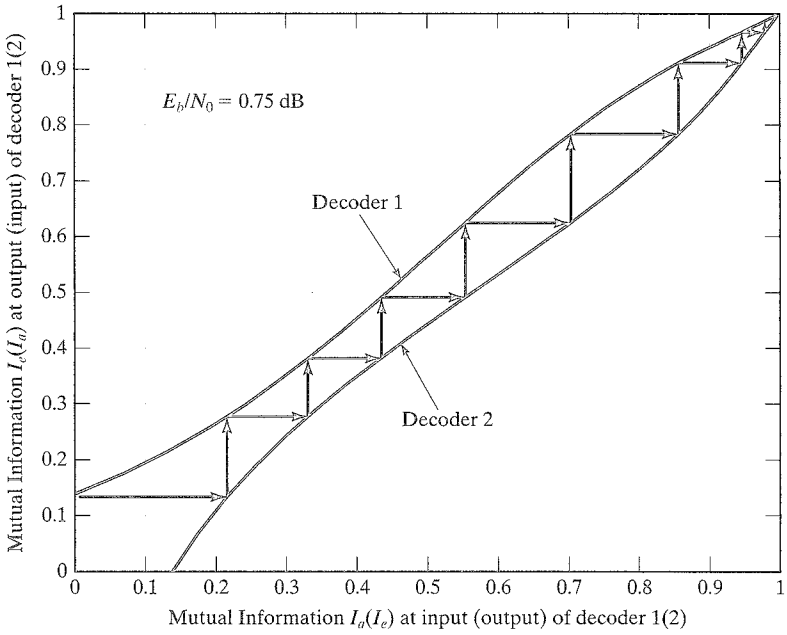
FIGURE 16.16: Decoder input–output transfer curves.

to a situation in which an iteration results in an I_e at the decoder output below the I_a at the decoder input; that is, the extrinsic estimates are getting *less* reliable. In this case iterative decoding fails to converge to a reliable solution, resulting in a high probability of decoding errors. In contrast, if the transfer curve stays above the $I_a = I_e$ line, the extrinsic estimates are getting *more* reliable, and iterative decoding converges to a reliable solution. We now define the *SNR threshold* of iterative decoding for a particular constituent code as the smallest value of the channel SNR, E_b/N_0, for which the transfer curve stays above the $I_a = I_e$ line. For example, in Figure 16.16 we see that the iterative decoding threshold for the original rate $R = 1/2$ turbo code is approximately 0.5 dB. This threshold corresponds to the value of the channel SNR, E_b/N_0, at which the bit-error probability curve of a PCCC begins its characteristic sharp drop, referred to as the *waterfall region* of the BER curve (see Figure 16.3).

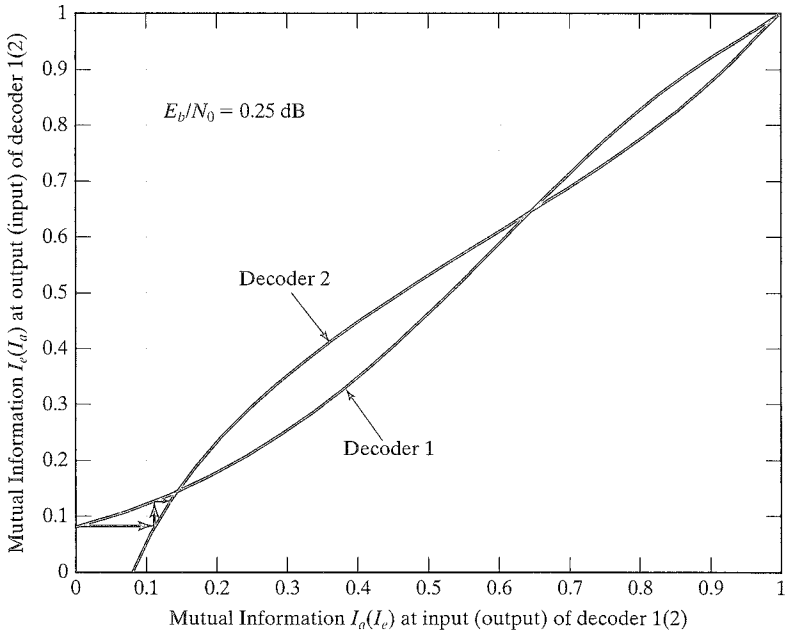
An EXIT chart for a particular channel SNR E_b/N_0 can be formed by plotting a transfer curve and its mirror image (about the line $I_a = I_e$) on the same set of axes, as shown in Figure 16.17(a) for $E_b/N_0 = 0.75$ dB. The two curves can be interpreted as representing the input–output transfer characteristics of the two constituent decoders in an iterative turbo decoder. The chart can then be used to trace the trajectory of iterative decoding as follows. For a given E_b/N_0, assume initially that $I_a = 0$, corresponding to the first iteration of decoder 1, and determine the resulting I_e (vertically) using the transfer curve for decoder 1. Because the a posteriori extrinsic L-value of decoder 1 becomes the a priori L-value of decoder 2, the value of I_e from decoder 1 becomes I_a for the first iteration of decoder 2, and the resulting I_e for decoder 2 is determined (horizontally) using the transfer curve for decoder 2. This procedure is repeated and traces the "staircase" trajectory of iterative decoding, as shown in Figure 16.17(a). If a *tunnel* exists between the two transfer curves, as shown in Figure 16.17(a) for $E_b/N_0 = 0.75$ dB, iterative decoding converges as I_e approaches a value of 1. As the channel SNR is lowered, the two transfer curves come closer together, and the smallest SNR for which the two curves do not touch, that is, for which a tunnel exists, is the SNR threshold. If no tunnel exists, that is, the two transfer curves cross each other, as shown in Figure 16.17(b) for $E_b/N_0 = 0.25$ dB, a decoder *fixed point* exists, I_e becomes stuck at a value less than 1, and iterative decoding does not converge.

The EXIT chart is particularly useful for visualizing the dynamics of iterative decoding. For example, we see from Figures 16.16 and 16.17 that as the channel SNR is lowered the tunnel narrows, and more iterations are needed before the mutual information becomes large enough for the decoder to converge to a reliable solution. We also note that the EXIT chart technique can be used to determine the threshold of a turbo code with asymmetric constituent codes simply by plotting the transfer curves of the two (different) constituent codes, which will no longer be the mirror image of each other.

The foregoing discussion indicates that determining the transfer curves and EXIT charts of various constituent codes is a useful tool in designing PCCCs with good low-SNR performance, that is, good performance in the waterfall region of the BER curve. The design rules to follow in a particular case, that is, the high-SNR uniform interleaver analysis or the low-SNR threshold analysis, depend on the application. In communication systems designed to achieve very low bit



(a)



(b)

FIGURE 16.17: Extrinsic information transfer (EXIT) charts for (a) $E_b/N_0 = 0.75$ dB and (b) $E_b/N_0 = 0.25$ dB.

error probabilities at moderate SNRs, the uniform interleaver analysis is preferred, whereas in systems designed to achieve moderate BERs at very low SNRs close to capacity, the threshold analysis is best.

The usefulness of the SNR threshold analysis discussed here depends on two factors: the accuracies of the Gaussian approximation for the a priori L-values, and the simulated estimates of the conditional pdf's for L_e. Both these factors depend on the length of the input block K used in the simulations. Experience with threshold analysis has shown that, in general, the constituent codes with the lowest SNR thresholds have denominator polynomials with short cycle lengths, in contrast with the long cycle lengths needed to achieve large values of d_{eff}. (See, for example, the list of SNR thresholds given in [39] for 8- and 16-state constituent codes.) Asymmetric code designs for PCCCs attempt to exploit these dual requirements by choosing different constituent codes, one with a short cycle length and the other with a long cycle length [30].

16.5 ITERATIVE DECODING OF TURBO CODES

A brief summary of turbo decoding was given in Section 16.1. As noted there, the best performance is achieved when the individual SISO decoders make use of the MAP, or APP, decoding algorithm. The details of this algorithm were presented earlier, both as an optimum bit-by-bit decoding algorithm for convolutional codes in Chapter 12 and as an optimum method for trellis-based decoding of block codes in Chapter 14. Thus, in this section, we emphasize the use of this algorithm as an iterative decoder of PCCs. Other SISO decoders that are simpler to implement, such as the SOVA or Max-log-MAP algorithms, also can be used to iteratively decode PCCs, but their performance is not as good.

The basic idea behind iterative decoding is to divide the decoding of a PCC, for which optimum decoding would be prohibitively complex, into a suboptimum iterative process involving only the relatively simple APP decoders for the constituent codes. After each iteration, the extrinsic L-values, representing reliability information about the bits to be decoded, are passed from one decoder to the other, ensuring that very little information is lost relative to optimal decoding. The basic structure of an iterative turbo decoder was shown in Figure 16.6. In the first iteration of decoder 1, the inputs are the channel L-values $L_c r_l^{(0)}$ and $L_c r_l^{(1)}$ for each received symbol (information and parity) of constituent code 1 and the a priori L-values $L_a(u_l)$ of each information bit, which all equal 0 for equally likely inputs. In the first iteration of decoder 2, the inputs are the channel L-values $L_c r_l^{(0)}$ and $L_c r_l^{(2)}$ for each received symbol (information and parity) of constituent code 2 and the extrinsic a posteriori L-values $L_e^{(1)}(u_l)$ of each information bit received from the output of decoder 1. These extrinsic L-values, along with the L-values of the received information symbols, must be interleaved according to the same pattern used at the encoder before entering decoder 2. In all subsequent iterations of decoder 1, the a priori L-values $L_a^{(1)}(u_l)$ are replaced by the extrinsic a posteriori L-values $L_e^{(2)}(u_l)$ received, after proper deinterleaving, from the output of decoder 2, and all subsequent iterations of decoder 2 are identical to the first iteration, except that the a priori inputs will be different. Finally, after a sufficient number of iterations, the decoded output can be derived from the a posteriori L-values of the information

bits $L^{(2)}(u_l)$, properly deinterleaved, at the output of decoder 2. Positive L-values are decoded as $+1$ and negative L-values as -1.

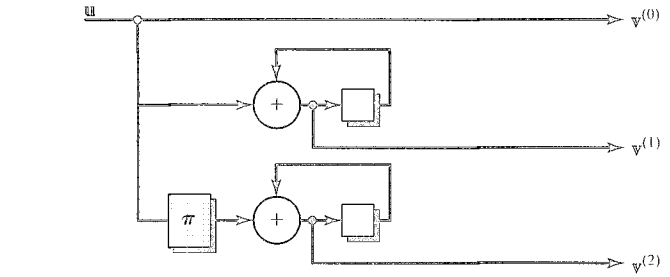
We now present a simple example, using the log-MAP algorithm first presented in Chapter 12, to illustrate the principle of iterative decoding.

EXAMPLE 16.14 Iterative Decoding Using the log-MAP Algorithm

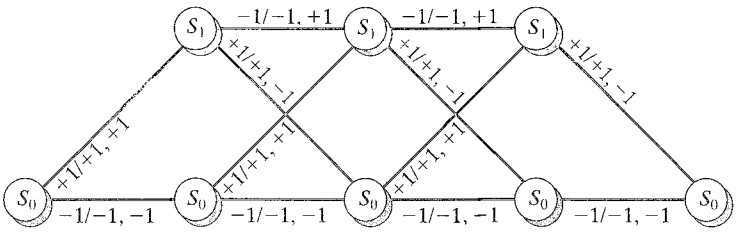
Consider the PCCC formed by using the 2-state $(2, 1, 1)$ SRCC with generator matrix

$$\mathbb{G}(D) = \begin{bmatrix} 1 & 1/(1+D) \end{bmatrix} \tag{16.104}$$

as the constituent code. A block diagram of the encoder is shown in Figure 16.18(a). Also consider an input sequence of length $K = 4$, including one termination bit, along with a 2×2 block (row–column) interleaver, resulting in a $(12, 3)$ PCCC with overall rate $R = 1/4$. (Recall that the termination bit is not an information bit.) The length $K = 4$ decoding trellis for the constituent code is shown in Figure 16.18(b), where the branches are labeled using the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$. The input block is given by the vector $\mathbb{u} = [u_0, u_1, u_2, u_3]$, the interleaved input block is $\mathbb{u}' = [u'_0, u'_1, u'_2, u'_3] = [u_0, u_2, u_1, u_3]$, the parity vector for the first constituent code is given by $\mathbb{p}^{(1)} = \left[p_0^{(1)}, p_1^{(1)}, p_2^{(1)}, p_3^{(1)}\right]$, and the parity vector for the second constituent code is $\mathbb{p}^{(2)} = \left[p_0^{(2)}, p_1^{(2)}, p_2^{(2)}, p_3^{(2)}\right]$. We can represent the 12 transmitted bits in a rectangular array, as shown in Figure 16.19(a), where the input vector $\mathbb{u}$ determines the parity vector $\mathbb{p}^{(1)}$ in the first two rows, and the interleaved input vector $\mathbb{u}'$ determines the parity vector $\mathbb{p}^{(2)}$



(a)



(b)

FIGURE 16.18: (a) A 2-state turbo encoder and (b) the decoding trellis for the (2, 1, 1) constituent code with $K = 4$.

u_0	u_1	$p_0^{(1)}$	$p_1^{(1)}$
u_2	u_3	$p_2^{(1)}$	$p_3^{(1)}$
$p_0^{(2)}$	$p_2^{(2)}$		
$p_1^{(2)}$	$p_3^{(2)}$		

(12, 3) PCCC

(a)

−1	+1	−1	+1
−1	+1	+1	−1
−1	+1		
−1	−1		

Coded values

(b)

+0.8	+1.0	+0.1	−0.5
−1.8	+1.6	+1.1	−1.6
−1.2	+0.2		
+1.2	−1.1		

Received L-values

(c)

−0.32	−0.38
+0.77	+0.47

Extrinsic L-values after first row decoding

(d)

−0.88	−0.69
+0.23	−0.04

Extrinsic L-values after first column decoding

(e)

−0.40	−0.07
−0.80	+2.03

Soft-output L-values after the first row and column decoding

(f)

−0.01	−0.01
+0.43	+0.77

Extrinsic L-values after second row decoding

(g)

−0.98	−0.81
+0.07	−0.21

Extrinsic L-values after second column decoding

(h)

−0.19	+0.18
−1.30	+2.16

Soft-output L-values after the second row and column decoding

(i)

−0.9	−0.9
+1.4	−0.3

Approximate extrinsic L-values after first row decoding

(j)

−0.8	−0.8
+1.1	+0.1

Approximate extrinsic L-values after first column decoding

(k)

−0.9	−0.7
+0.7	+1.4

Approximate soft-output L-values after the first row and column decoding

(l)

−0.1	−0.1
+0.6	+0.5

Approximate extrinsic L-values after second row decoding

(m)

−0.9	−0.7
+0.3	−0.5

Approximate extrinsic L-values after second column decoding

(n)

−0.2	+0.2
−0.9	+1.6

Approximate soft-output L-values after the second row and column decoding

(o)

FIGURE 16.19: Iterative decoding example for a (12, 3) PCCC.

in the first two columns. For purposes of illustration, we assu... ...e the values shown in Figure 16.19(b). We also assume a channel S... ...NR of (−6.02 dB), so that the received channel L-values cor... ...ling t vector $\mathbb{r} = \left[r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, r_2^{(0)} r_2^{(1)} r_2^{(2)}, r_3^{(0)} r_3^{(1)} r_3^{(2)}\right]$ are ...ven b

$$L_c r_l^{(j)} = 4\left(\frac{E_s}{N_0}\right) r_l^{(j)} = r_l^{(j)}, \quad l = 0, 1, 2, 3, \quad j = \ldots \text{ 2.}$$

Again for purposes of illustration, a set of particular r... ...d char given in Figure 16.19(c).

In the first iteration of decoder 1 (row decodin... ...log- is applied to the trellis of the 2-state (2, 1, 1) code s... ...in Fi compute the a posteriori L-values $L^{(1)}(u_l)$ for each ... ...e four the corresponding extrinsic a posteriori L-values $L_e^{(1)}$ ... ...to pa (the column decoder).[2] Similarly, in the first iteration ... ...coder algorithm uses the extrinsic a posteriori L-values $L_e^{(1)}(u_l)$ ... ...ived f the a priori L-values, $L_a^{(2)}(u_l)$ to compute the a posteri... ...values of the four input bits and the corresponding extrinsic a... ...eriori to pass back to decoder 1. Further decoding proceeds it... ...vely i

Before returning to Example 16.14, we develop ...eral e a posteriori L-values $L(u_l)$ and the extrinsic a poste... ...L-va (2, 1, 1) constituent code in Figure 16.18(b) using th... ...og-M simplify notation, we denote the transmitted vector a... ...= (v parity $v_l = (u_l, p_l)$, $l = 0, 1, 2, 3$, u_l is an input bit, and p_l i... ...= (r_{ul} received vector is denoted as $\mathbb{r} = (\mathbb{r}_0, \mathbb{r}_1, \mathbb{r}_2, \mathbb{r}_3)$, where ...ed inp r_{ul} is the received symbol corresponding to the transm... ...rity bi the received symbol corresponding to the transmitted posteriori L-value is given by (see (12.106) and (12.111

$$\begin{aligned} L(u_l) &= \ln \frac{P(u_l = +1 \mid \mathbb{r})}{P(u_l = -1 \mid \mathbb{r})} \\ &= \ln \frac{\sum_{(s',s)\in\Sigma_l^+} p(s', s, \mathbb{r}}{\sum_{(s',s)\in\Sigma_l^-} p(s', s, \mathbb{r}} \end{aligned}$$

... s rep where s' represents a state at time l (denoted by $s' \in \sigma_l$ ... state $l + 1$ (denoted by $s \in \sigma_{l+1}$), and the sums are over al... ...in th $u_l = +1$ or -1, respectively. (Recall that every bran... ...of the state at time l to a state at time $l + 1$ is included in one can write the joint probabilities $p(s', s, \mathbb{r})$ in (16.106) as

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

...uted fo ...e deco a prior

[2] For iterative decoding, extrinsic a posteriori L-values are comp... bits as well as information bits. (Note that for feedback encoders, th... of the terminating bits.) Before the first iteration of decoding, the ... bits are assumed to be zero.

$\gamma_l^*(s', s)$, and $\beta_{l+1}^*(s)$ are the familiar log-domain α's, γ's, and β's
orithm (see (12.129)). (Recall that the three terms in (16.107), $\alpha_l^*(s'$
$\beta_{l+1}^*(s)$ represent the influence of the received sequence before tim
d after time l on the estimate of the input bit at time l, respectively.)
ntinuous-output AWGN channel with an SNR of E_s/N_0, followin
128c), and (12.128e), we can write the MAP decoding equations as

$$\text{netric: } \gamma_l^*(s', s) = \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}\mathbb{r}_l \cdot \mathbb{v}_l, \quad l = 0, 1, 2, 3, \tag{16.108a}$$

$$\text{metric: } \alpha_{l+1}^*(s) = \max_{s' \in \sigma_l}^*[\gamma_l^*(s', s) + \alpha_l^*(s')], \quad l = 0, 1, 2, 3, \tag{16.108b}$$

$$\text{rd metric: } \beta_l^*(s') = \max_{s \in \sigma_{l+1}}^*[\gamma_l^*(s', s) + \beta_{l+1}^*(s)], \tag{16.108c}$$

function is defined in (12.127), and the initial conditions are
$= 0$, and $\alpha_0^*(S_1) = \beta_4^*(S_1) = -\infty$. (Note that for iterative decoding,
16.107a)) is used to compute the branch metrics corresponding to
as well as information bits.) Further simplifying the branch metric,

$$\begin{aligned} ', s) &= \frac{u_l L_a(u_l)}{2} + \frac{L_c}{2}(u_l r_{ul} + p_l r_{pl}) \\ &= \frac{u_l}{2}[L_a(u_l) + L_c r_{ul}] + \frac{p_l}{2} L_c r_{pl}, \quad l = 0, 1, 2, 3. \end{aligned} \tag{16.109}$$

serve from Figure 16.18(b) that to determine the a posteriori
here is only one term in each of the sums in (16.106), because at
ly one +1 transition and one −1 transition in the trellis diagram.
ss the a posteriori L-value of u_0 as

$$\begin{aligned} &= S_0, s = S_1, \mathbb{r}) - \ln\ p(s' = S_0, s = S_0, \mathbb{r}) \\ &+ \gamma_0^*(s' = S_0, s = S_1) + \beta_1^*(S_1)] - \\ &+ \gamma_0^*(s' = S_0, s = S_0) + \beta_1^*(S_0)] \\ &u_0) + L_c r_{u0}] + \tfrac{1}{2} L_c r_{p0} + \beta_1^*(S_1)\Big\} - \\ &{}_0) + L_c r_{u0}] - \tfrac{1}{2} L_c r_{p0} + \beta_1^*(S_0)\Big\} \\ &0) + L_c r_{u0}]\Big\} - \Big\{-\tfrac{1}{2}[L_a(u_0) + L_c r_{u0}]\Big\} + \\ &+ \beta_1^*(S_1) + \tfrac{1}{2} L_c r_{p0} - \beta_1^*(S_0)\Big\} = L_c r_{u0} + L_a(u_0) + L_e(u_0), \end{aligned} \tag{16.110a}$$

where

$$L_e(u_0) \equiv L_c r_{p0} + \beta_1^*(S_1) - \beta_1^*(S_0) \tag{16.110b}$$

represents the extrinsic a posteriori (output) L-value of u_0. The final form of (16.110a) illustrates clearly the three components of the a posteriori L-value of u_0 computed at the output of a log-MAP decoder:

- $L_c r_{u0}$: the received channel L-value corresponding to bit u_0, which was part of the decoder input.
- $L_a(u_0)$: the a priori L-value of u_0, which was also part of the decoder input. Except for the first iteration of decoder 1, this term equals the extrinsic a posteriori L-value of u_0 received from the output of the other decoder. (For the first iteration of decoder 1, $L_a(u_0) = 0$ for equally likely input bits.)
- $L_e(u_0)$: the extrinsic part of the a posteriori L-value of u_0, which does not depend on $L_c r_{u0}$ or $L_a(u_0)$. This term is then sent to the other decoder as its a priori input.

We now proceed in a similar manner to compute the a posteriori L-value of bit u_1. We see from Figure 16.18(b) that in this case there are two terms in each of the sums in (16.106), because at this time there are two $+1$ and two -1 transitions in the trellis diagram. Thus,

$$\begin{aligned}
L(u_1) &= \ln\left[p\left(s' = S_0, s = S_1, \mathbb{r}\right) + p\left(s' = S_1, s = S_0, \mathbb{r}\right)\right] - \\
&\quad \ln\left[p\left(s' = S_0, s = S_0, \mathbb{r}\right) + p\left(s' = S_1, s = S_1, \mathbb{r}\right)\right] \\
&= \max^*\Big\{[\alpha_1^*(S_0) + \gamma_1^*(s' = S_0, s = S_1) + \beta_2^*(S_1)], \\
&\quad [\alpha_1^*(S_1) + \gamma_1^*(s' = S_1, s = S_0) + \beta_2^*(S_0)]\Big\} \\
&\quad - \max^*\Big\{[\alpha_1^*(S_0) + \gamma_1^*(s' = S_0, s = S_0) + \beta_2^*(S_0)], \\
&\quad [\alpha_1^*(S_1) + \gamma_1^*(s' = S_1, s = S_1) + \beta_2^*(S_1)]\Big\} \\
&= \max^*\Big\{(+\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}] + \tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)), \\
&\quad (+\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}] - \tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0))\Big\} \\
&\quad - \max^*\Big\{(-\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}] - \tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)), \\
&\quad (-\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}] + \tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1))\Big\} \\
&= \Big\{+\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}]\Big\} - \Big\{-\tfrac{1}{2}[L_a(u_1) + L_c r_{u1}]\Big\} \\
&\quad + \max^*\Big\{[+\tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)], [-\tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)]\Big\} \\
&\quad - \max^*\Big\{[-\tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)], [+\tfrac{1}{2} L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)]\Big\} \\
&= L_c r_{u1} + L_a(u_1) + L_e(u_1),
\end{aligned} \tag{16.111a}$$

where

$$L_e(u_1) = \max^*\left\{[+\tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)], [-\tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)]\right\} - \max^*\left\{[-\tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)], [+\tfrac{1}{2}L_c r_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)]\right\} \tag{16.111b}$$

and we have made use of the identity (see Problem 16.24) $\max^*(w + x, w + y) \equiv w + \max^*(x, y)$. Continuing, we can use the same procedure to compute the a posteriori L-values of bits u_2 and u_3 as

$$L(u_2) = L_c r_{u2} + L_a(u_2) + L_e(u_2), \tag{16.112a}$$

where

$$L_e(u_2) = \max^*\left\{[+\tfrac{1}{2}L_c r_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_1)], [-\tfrac{1}{2}L_c r_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_0)]\right\} - \max^*\left\{[-\tfrac{1}{2}L_c r_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_0)], [+\tfrac{1}{2}L_c r_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_1)]\right\}, \tag{16.112b}$$

and

$$L(u_3) = L_c r_{u3} + L_a(u_3) + L_e(u_3), \tag{16.113a}$$

where

$$L(u_3) = [-\tfrac{1}{2}L_c r_{p3} + \alpha_3^*(S_1) + \beta_4^*(S_0)] - [-\tfrac{1}{2}L_c r_{p3} + \alpha_3^*(S_0) + \beta_4^*(S_0)] = \alpha_3^*(S_1) - \alpha_3^*(S_0). \tag{16.113b}$$

(Note that in this example, the value of the received parity symbol r_{p3} does not affect $L(u_3)$, since, for the trellis of Figure 16.18(b), $p_3 = 0$ for both branches at time $l = 3$, and thus r_{p3} conveys no information useful for decoding.)

We now need expressions for the terms $\alpha_1^*(S_0)$, $\alpha_1^*(S_1)$, $\alpha_2^*(S_0)$, $\alpha_2^*(S_1)$, $\alpha_3^*(S_0)$, $\alpha_3^*(S_1)$, $\beta_1^*(S_0)$, $\beta_1^*(S_1)$, $\beta_2^*(S_0)$, $\beta_2^*(S_1)$, $\beta_3^*(S_0)$, and $\beta_3^*(S_1)$ that are used to calculate the extrinsic a posteriori L-values $L_e(u_l)$, $l = 0, 1, 2, 3$. Using (16.108) and (16.109) and Figure 16.18(b) and adopting the shorthand notation $L_{ul} \equiv L_c r_{ul} + L_a(u_l)$ and $L_{pl} \equiv L_c r_{pl}$, $l = 0, 1, 2, 3$, for *intrinsic* information bit L-values and parity bit L-values, respectively, we obtain (see Problem 16.25)

$$\alpha_1^*(S_0) = -\tfrac{1}{2}(L_{u0} + L_{p0}) \tag{16.114a}$$

$$\alpha_1^*(S_1) = -\tfrac{1}{2}(L_{u0} + L_{p0}) \tag{16.114b}$$

$$\alpha_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(L_{u1} + L_{p1}) + \alpha_1^*(S_0)], [+\tfrac{1}{2}(L_{u1} - L_{p1}) + \alpha_1^*(S_1)]\right\} \tag{16.114c}$$

$$\alpha_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(L_{u1} + L_{p1}) + \alpha_1^*(S_0)], [-\tfrac{1}{2}(L_{u1} - L_{p1}) + \alpha_1^*(S_1)]\right\} \tag{16.114d}$$

$$\alpha_3^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(L_{u2} + L_{p2}) + \alpha_2^*(S_0)], [+\tfrac{1}{2}(L_{u2} - L_{p2}) + \alpha_2^*(S_1)]\right\} \tag{16.114e}$$

$$\alpha_3^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(L_{u2} + L_{p2}) + \alpha_2^*(S_0)], [-\tfrac{1}{2}(L_{u2} - L_{p2}) + \alpha_2^*(S_1)]\right\} \tag{16.114f}$$

$$\beta_3^*(S_0) = -\tfrac{1}{2}\left(L_{u3} + L_{p3}\right) \tag{16.114g}$$

$$\beta_3^*(S_1) = +\tfrac{1}{2}\left(L_{u3} - L_{p3}\right) \tag{16.114h}$$

$$\beta_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(L_{u2} + L_{p2}) + \beta_3^*(S_0)], [+\tfrac{1}{2}(L_{u2} + L_{p2}) + \beta_3^*(S_1)]\right\} \tag{16.114i}$$

$$\beta_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(L_{u2} - L_{p2}) + \beta_3^*(S_0)], [-\tfrac{1}{2}(L_{u2} - L_{p2}) + \beta_3^*(S_1)]\right\} \tag{16.114j}$$

$$\beta_1^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(L_{u1} + L_{p1}) + \beta_2^*(S_0)], [+\tfrac{1}{2}(L_{u1} + L_{p1}) + \beta_2^*(S_1)]\right\} \tag{16.114k}$$

$$\beta_1^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(L_{u1} - L_{p1}) + \beta_2^*(S_0)], [-\tfrac{1}{2}(L_{u1} - L_{p1}) + \beta_2^*(S_1)]\right\}. \tag{16.114l}$$

We note here that the a priori L-value of a parity bit $L_a(p_l) = 0$ for all l, since for a linear code with equally likely information bits, the parity bits are also equally likely. Also, unlike the information bits, the L-values of the parity bits are not updated by the iterative decoding algorithm, and the parity bit L-values remain constant throughout decoding. Thus, we can write $L_{pl} = L_c r_{pl} + L_a(p_l) = L_c r_{pl}$; that is, the shorthand notation for the intrinsic L-values of information bits L_{ul} and parity bits L_{pl} is consistent. Finally, we can write the extrinsic a posteriori L-values in terms of L_{u2} and L_{p2} as

$$L_e(u_0) = L_{p0} + \beta_1^*(S_1) - \beta_1^*(S_0), \tag{16.115a}$$

$$L_e(u_1) = \max^*\left\{[+\tfrac{1}{2}L_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_1)], [-\tfrac{1}{2}L_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_0)]\right\} - \max^*\left\{[-\tfrac{1}{2}L_{p1} + \alpha_1^*(S_0) + \beta_2^*(S_0)], [+\tfrac{1}{2}L_{p1} + \alpha_1^*(S_1) + \beta_2^*(S_1)]\right\} \tag{16.115b}$$

$$L_e(u_2) = \max^*\left\{[+\tfrac{1}{2}L_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_1)], [-\tfrac{1}{2}L_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_0)]\right\} - \max^*\left\{[-\tfrac{1}{2}L_{p2} + \alpha_2^*(S_0) + \beta_3^*(S_0)], [+\tfrac{1}{2}L_{p2} + \alpha_2^*(S_1) + \beta_3^*(S_1)]\right\} \tag{16.115c}$$

and

$$L_e(u_3) = \alpha_3^*(S_1) - \alpha_3^*(S_0). \tag{16.115d}$$

We note that in the expressions for the extrinsic L-values $L_e(u_l)$, the intrinsic L-value L_{ul} does not appear in any of the terms; that is, the extrinsic L-value of bit u_l does not depend directly on either the received or *a priori* L-values of u_l.

EXAMPLE 16.14 (Continued)

We can now evaluate the extrinsic a posteriori L-values using (16.114) and (16.115) and the received channel L-values in Figure 16.19(c). Beginning with the first iteration of decoder 1 (row decoding) for the bits u_0, u_1, u_2, and u_3, and recalling that the initial a priori L-values in the expressions for L_{u0}, L_{u1}, L_{u2}, and L_{u3} are equal to 0, we obtain from (16.114)

$$\alpha_1^*(S_0) = -\tfrac{1}{2}(0.8 + 0.1) = -0.45 \tag{16.116a}$$

$$\alpha_1^*(S_1) = +\tfrac{1}{2}(0.8 + 0.1) = +0.45 \tag{16.116b}$$

$$\alpha_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(1.0-0.5)-0.45],[+\tfrac{1}{2}(1.0+0.5)+0.45]\right\}$$
$$= \max^*\{-0.70,+1.20\} = 1.34 \tag{16.116c}$$
$$\alpha_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(1.0-0.5)-0.45],[-\tfrac{1}{2}(1.0+0.5)+0.45]\right\}$$
$$= \max^*\{-0.20,-0.30\} = 0.44 \tag{16.116d}$$
$$\alpha_3^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(-1.8+1.1)+1.34],[+\tfrac{1}{2}(-1.8-1.1)+0.44]\right\}$$
$$= \max^*\{1.69,-1.01\} = 1.76 \tag{16.116e}$$
$$\alpha_3^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(-1.8+1.1)+1.34],[-\tfrac{1}{2}(-1.8-1.1)+0.44]\right\}$$
$$= \max^*\{0.99,1.89\} = 2.23 \tag{16.116f}$$
$$\beta_3^*(S_0) = -\tfrac{1}{2}(1.6-1.6) = 0 \tag{16.116g}$$
$$\beta_3^*(S_1) = +\tfrac{1}{2}(1.6+1.6) = 1.60 \tag{16.116h}$$
$$\beta_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(-1.8+1.1)+0],[+\tfrac{1}{2}(-1.8+1.1)+1.60]\right\}$$
$$= \max^*\{0.35,1.25\} = 1.59 \tag{16.116i}$$
$$\beta_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(-1.8-1.1)+0],[-\tfrac{1}{2}(-1.8-1.1)+1.60]\right\}$$
$$= \max^*\{-1.45,3.05\} = 3.06 \tag{16.116j}$$
$$\beta_1^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(1.0-0.5)+1.59],[+\tfrac{1}{2}(1.0-0.5)+3.06]\right\}$$
$$= \max^*\{1.34,3.31\} = 3.44 \tag{16.116k}$$
$$\beta_1^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(1.0+0.5)+1.59],[-\tfrac{1}{2}(1.0+0.5)+3.06]\right\}$$
$$= \max^*\{2.34,2.31\} = 3.02. \tag{16.116l}$$

Then, from (16.115) we obtain

$$L_e^{(1)}(u_0) = 0.1 + 3.02 - 3.44 = -0.32, \tag{16.117a}$$
$$L_e^{(1)}(u_1) = \max^*\left\{[-0.25-0.45+3.06],[0.25+0.45+1.59]\right\}$$
$$-\max^*\left\{[0.25-0.45+1.59],[-0.25+0.45+3.06]\right\}$$
$$= \max^*\{2.36,2.29\} - \max^*\{1.39,3.26\} = 3.02 - 3.40 = -0.38, \tag{16.117b}$$

and, using similar calculations, we have $L_e^{(1)}(u_2) = +0.77$ and $L_e^{(1)}(u_3) = +0.47$. These extrinsic values after the first iteration of row decoding are listed in Figure 16.19(d). Now, using these extrinsic a posteriori L-values as the a priori inputs for decoder 2 (column decoding), so that $L_{ul} = L_c r_{ul} + L_e^{(1)}(u_l)$, we obtain from (16.114)

$$\alpha_1^*(S_0) = -\tfrac{1}{2}(0.8-0.32-1.2) = 0.36 \tag{16.118a}$$

$$\alpha_1^*(S_1) = +\tfrac{1}{2}(0.8 - 0.32 - 1.2) = -0.36 \tag{16.118b}$$

$$\alpha_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(-1.8 + 0.77 + 1.2) + 0.36], [+\tfrac{1}{2}(-1.8 + 0.77 - 1.2) - 0.36]\right\}$$
$$= \max^*\{0.275, -1.475\} = 0.44 \tag{16.118c}$$

$$\alpha_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(-1.8 + 0.77 + 1.2) + 0.36], [-\tfrac{1}{2}(-1.8 + 0.77 - 1.2) - 0.36]\right\}$$
$$= \max^*\{0.445, 0.755\} = 1.31 \tag{16.118d}$$

$$\alpha_3^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(1.0 - 0.38 + 0.2) + 0.44], [+\tfrac{1}{2}(1.0 - 0.38 - 0.2) + 1.31]\right\}$$
$$= \max^*\{0.03, 1.52\} = 1.72 \tag{16.118e}$$

$$\alpha_3^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(1.0 - 0.38 + 0.2) + 0.44], [-\tfrac{1}{2}(1.0 - 0.38 - 0.2) + 1.31]\right\}$$
$$= \max^*\{0.85, 1.1\} = 1.68 \tag{16.118f}$$

$$\beta_3^*(S_0) = -\tfrac{1}{2}(1.6 + 0.47 - 1.1) = -0.485 \tag{16.118g}$$

$$\beta_3^*(S_1) = +\tfrac{1}{2}(1.6 + 0.47 + 1.1) = 1.585 \tag{16.118h}$$

$$\beta_2^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(1.0 - 0.38 + 0.2) - 0.485], [+\tfrac{1}{2}(1.0 - 0.38 + 0.2) + 1.585]\right\}$$
$$= \max^*\{-0.895, 1.995\} = 2.05 \tag{16.118i}$$

$$\beta_2^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(1.0 - 0.38 - 0.2) - 0.485], [-\tfrac{1}{2}(1.0 - 0.38 - 0.2) + 1.585]\right\}$$
$$= \max^*\{-0.275, 1.375\} = 1.55 \tag{16.118j}$$

$$\beta_1^*(S_0) = \max^*\left\{[-\tfrac{1}{2}(-1.8 + 0.77 + 1.2) + 2.05], [+\tfrac{1}{2}(-1.8 + 0.77 + 1.2) + 1.55]\right\}$$
$$= \max^*\{1.965, 1.635\} = 2.51 \tag{16.118k}$$

$$\beta_1^*(S_1) = \max^*\left\{[+\tfrac{1}{2}(-1.8 + 0.77 - 1.2) + 2.05], [-\tfrac{1}{2}(-1.8 + 0.77 - 1.2) + 1.55]\right\}$$
$$= \max^*\{0.935, 2.665\} = 2.83, \tag{16.118l}$$

where we note that the roles of u_1 and u_2 in (16.118) are reversed compared with (16.116), because, after interleaving, $v_1' = u_2$ and $u_2' = v_1$. We now use (16.115) to obtain, for the bits u_0, u_2, u_1, and u_3,

$$L_e^{(2)}(u_0) = -1.2 + 2.83 - 2.51 = -0.88 \tag{16.119a}$$

$$L_e^{(2)}(u_2) = \max^*\left\{[0.6 + 0.36 + 1.55], [-0.6 - 0.36 + 2.05]\right\}$$
$$- \max^*\left\{[-0.6 + 0.36 + 2.05], [0.6 - 0.36 + 1.55]\right\}$$
$$= \max^*\{2.51, 1.09\} - \max^*\{1.81, 1.79\} = 2.72 - 2.49 = +0.23, \tag{16.119b}$$

and, using similar calculations, we have $L_e^{(2)}(u_1) = -0.69$ and $L_e^{(2)}(u_3) = -0.04$. (Again note that in using (16.115) to evaluate the expressions for $L_e^{(2)}(u_l)$, the roles of u_1 and u_2 are reversed.) These extrinsic values after the first iteration of column

decoding are listed in Figure 16.19(e). Finally, the a posteriori L-values of the four information bits after the first complete iteration of decoding are given by

$$L^{(2)}(u_0) = L_c r_{u0} + L_a^{(2)}(u_0) + L_e^{(2)}(u_0) = 0.8 - 0.32 - 0.88 = -0.40 \quad (16.120a)$$

$$L^{(2)}(u_2) = L_c r_{u2} + L_a^{(2)}(u_2) + L_e^{(2)}(u_2) = -1.8 + 0.77 + 0.23 = -0.80 \quad (16.120b)$$

$$L^{(2)}(u_1) = L_c r_{u1} + L_a^{(2)}(u_1) + L_e^{(2)}(u_1) = 1.0 - 0.38 - 0.69 = -0.07 \quad (16.120c)$$

$$L^{(2)}(u_3) = L_c r_{u3} + L_a^{(2)}(u_3) + L_e^{(2)}(u_3) = 1.6 + 0.47 - 0.04 = 2.03 \quad (16.120d)$$

and shown in Figure 16.19(f), where the a priori L-values for decoder 2, $L_a^{(2)}(u_l)$, are equal to the extrinsic a posteriori L-values, $L_e^{(1)}(u_l)$, $l = 0, 1, 2, 3$, after the first iteration of decoder 1. Thus, if a final decoding decision was made after just one iteration, the three information bits would be decoded as

$$\hat{u}_0 = -1, \quad \hat{u}_2 = -1, \quad \hat{u}_1 = -1, \quad (16.121)$$

and bit u_1 would be incorrectly decoded in this case.

A second iteration of decoding would use the extrinsic a posteriori L-values $L_e^{(2)}(u_l)$ from the first iteration of decoder 2 as a priori L-values $L_a^{(1)}(u_l)$, $l = 0, 1, 2, 3$, for decoder 1, thus producing a new set of extrinsic a posteriori L-values $L_e^{(1)}(u_l)$ at the output of decoder 1. These would then be used as a priori inputs to decoder 2, and so on. In Figure 16.19(g), (h), and (i), we give the L-values $L_e^{(1)}(u_l)$, $L_e^{(2)}(u_l)$, and $L^{(2)}(u_l)$, respectively, $l = 0, 1, 2, 3$, for the second iteration of decoding (see Problem 16.26). Note that in this case, if a final decoding decision was made after two iterations of decoding, all information bits would be correctly decoded.

Note that for iterative decoding, although u_3 is not an information bit and does not need to be decoded, its extrinsic L-values must be updated throughout decoding. Also, for this example, all input sequences that terminate encoder 1, when interleaved, also terminate encoder 2. Thus, the operation of the log-MAP algorithm is exactly the same for both decoders. In general, however, the interleaved input sequence will not terminate encoder 2. In this case, the log-MAP algorithm must specify initial values for the β^*'s at each state; that is, $\beta_K^*(S_i)$, $i = 0, 1, \cdots, 2^\nu - 1$, must be specified. These can be set either to all zeros or to the final values of the α^*'s at time K.

The most difficult calculations in the preceding example are the evaluations of the exponential terms needed in the computation of the max* operation. These calculations can be simplified by using the Max-log-MAP algorithm discussed previously in Chapter 12. We now use this simplified algorithm to revisit the above example.

EXAMPLE 16.15 Iterative Decoding Using the Max-log-MAP Algorithm

When the approximation $\max^*(x, y) \approx \max(x, y)$ is applied to the forward and backward recursions in (16.114) and the extrinsic a posteriori L-value calculations

in (16.115), we obtain for the first iteration of decoder 1

$$\alpha_2^*(S_0) \approx \max\{-0.70, 1.20\} = 1.20 \tag{16.122a}$$

$$\alpha_2^*(S_1) \approx \max\{-0.20, -0.30\} = -0.20 \tag{16.122b}$$

$$\begin{aligned}\alpha_3^*(S_0) &\approx \max\left\{[-\tfrac{1}{2}(-1.8+1.1)+1.20], [+\tfrac{1}{2}(-1.8-1.1)-0.20]\right\} \\ &= \max\{1.55, -1.65\} = 1.55\end{aligned} \tag{16.122c}$$

$$\begin{aligned}\alpha_3^*(S_1) &\approx \max\left\{[+\tfrac{1}{2}(-1.8+1.1)+1.20], [-\tfrac{1}{2}(-1.8-1.1)-0.20]\right\} \\ &= \max\{0.85, 1.25\} = 1.25\end{aligned} \tag{16.122d}$$

$$\beta_2^*(S_0) \approx \max\{0.35, 1.25\} = 1.25 \tag{16.122e}$$

$$\beta_2^*(S_1) \approx \max\{-1.45, 3.05\} = 3.05 \tag{16.122f}$$

$$\begin{aligned}\beta_1^*(S_0) &\approx \max\left\{[-\tfrac{1}{2}(1.0-0.5)+1.25], [+\tfrac{1}{2}(1.0-0.5)+3.05]\right\} \\ &= \max\{1.00, 3.30\} = 3.30\end{aligned} \tag{16.122g}$$

$$\begin{aligned}\beta_1^*(S_1) &\approx \max\left\{[+\tfrac{1}{2}(1.0+0.5)+1.25], [-\tfrac{1}{2}(1.0+0.5)+3.05]\right\} \\ &= \max\{2.00, 2.30\} = 2.30,\end{aligned} \tag{16.122h}$$

$$L_e^{(1)}(u_0) \approx 0.1 + 2.30 - 3.30 = -0.90 \tag{16.123a}$$

$$\begin{aligned}L_e^{(1)}(u_1) &\approx \max\left\{[-0.25-0.45+3.05], [0.25+0.45+1.25]\right\} \\ &\quad - \max\left\{[0.25-0.45+1.25], [-0.25+0.45+3.05]\right\} \\ &= \max\{2.35, 1.95\} - \max\{1.05, 3.25\} = 2.35 - 3.25 = -0.90,\end{aligned} \tag{16.123b}$$

and, using similar calculations, we have $L_e^{(1)}(u_2) \approx +1.4$ and $L_e^{(1)}(u_3) \approx -0.3$. Using these approximate extrinsic a posteriori L-values as a priori L-values for decoder 2, and recalling that the roles of u_1 and u_2 are reversed for decoder 2, we obtain

$$\alpha_1^*(S_0) = -\tfrac{1}{2}(0.8 - 0.9 - 1.2) = 0.65 \tag{16.124a}$$

$$\alpha_1^*(S_1) = +\tfrac{1}{2}(0.8 - 0.9 - 1.2) = -0.65 \tag{16.124b}$$

$$\begin{aligned}\alpha_2^*(S_0) &\approx \max\left\{[-\tfrac{1}{2}(-1.8+1.4+1.2)+0.65], [+\tfrac{1}{2}(-1.8+1.4-1.2)-0.65]\right\} \\ &= \max\{0.25, -1.45\} = 0.25\end{aligned} \tag{16.124c}$$

$$\begin{aligned}\alpha_2^*(S_1) &\approx \max\left\{[+\tfrac{1}{2}(-1.8+1.4+1.2)+0.65], [-\tfrac{1}{2}(-1.8+1.4-1.2)-0.65]\right\} \\ &= \max\{1.05, 0.15\} = 1.05\end{aligned} \tag{16.124d}$$

$$\begin{aligned}\alpha_3^*(S_0) &\approx \max\left\{[-\tfrac{1}{2}(1.0-0.9+0.2)+0.25], [+\tfrac{1}{2}(1.0-0.9-0.2)+1.05]\right\} \\ &= \max\{0.10, 1.00\} = 1.00\end{aligned} \tag{16.124e}$$

$$\alpha_3^*(S_1) \approx \max\left\{[+\tfrac{1}{2}(1.0 - 0.9 + 0.2) + 0.25], [-\tfrac{1}{2}(1.0 - 0.9 - 0.2) + 1.05]\right\}$$
$$= \max\{0.40, 1.10\} = 1.10 \tag{16.124f}$$
$$\beta_3^*(S_0) = -\tfrac{1}{2}(1.6 - 0.3 - 1.1) = -0.10 \tag{16.124g}$$
$$\beta_3^*(S_1) = +\tfrac{1}{2}(1.6 - 0.3 + 1.1) = 1.20 \tag{16.124h}$$
$$\beta_2^*(S_0) \approx \max\left\{[-\tfrac{1}{2}(1.0 - 0.9 + 0.2) - 0.10], [+\tfrac{1}{2}(1.0 - 0.9 + 0.2) + 1.20]\right\}$$
$$= \max\{-0.25, 1.35\} = 1.35 \tag{16.124i}$$
$$\beta_2^*(S_1) \approx \max\left\{[+\tfrac{1}{2}(1.0 - 0.9 - 0.2) - 0.10], [-\tfrac{1}{2}(1.0 - 0.9 - 0.2) + 1.20]\right\}$$
$$= \max\{-0.15, 1.25\} = 1.25 \tag{16.124j}$$
$$\beta_1^*(S_0) \approx \max\left\{[-\tfrac{1}{2}(-1.8 + 1.4 + 1.2) + 1.35], [+\tfrac{1}{2}(-1.8 + 1.4 + 1.2) + 1.25]\right\}$$
$$= \max\{0.95, 1.65\} = 1.65 \tag{16.124k}$$
$$\beta_1^*(S_1) \approx \max\left\{[+\tfrac{1}{2}(-1.8 + 1.4 - 1.2) + 1.35], [-\tfrac{1}{2}(-1.8 + 1.4 - 1.2) + 1.25]\right\}$$
$$= \max\{0.55, 2.05\} = 2.05, \tag{16.124l}$$
$$L_e^{(2)}(u_0) \approx -1.2 + 2.05 - 1.65 = -0.80 \tag{16.125a}$$
$$L_e^{(2)}(u_2) \approx \max\left\{[0.6 + 0.65 + 1.25], [-0.6 - 0.65 + 1.35]\right\}$$
$$- \max\left\{[-0.6 + 0.65 + 1.35], [0.6 - 0.65 + 1.25]\right\}$$
$$= \max\{2.5, 0.1\} - \max\{1.4, 1.2\} = 2.5 - 1.4 = 1.10, \tag{16.125b}$$

and, using similar calculations, we have $L_e^{(2)}(u_1) \approx -0.8$ and $L_e^{(2)}(u_3) \approx +0.1$. These approximate extrinsic a posteriori L-values for row and column decoding are listed in Figure 16.19(j) and 16.19(k), respectively. Finally, we calculate the approximate a posteriori L-value of information bit u_0 after the first complete iteration of decoding as

$$L^{(2)}(u_0) = L_c r_{u0} + L_a^{(2)}(u_0) + L_e(u_0) \approx 0.8 - 0.9 - 0.8 = -0.9, \tag{16.126}$$

and we similarly obtain the remaining approximate a posteriori L-values as $L^{(2)}(u_2) \approx +0.7$, $L^{(2)}(u_1) \approx -0.7$, and $L^{(2)}(u_3) \approx +1.4$. These approximate a posteriori L-values are shown in Figure 16.19(l), and we see that in this case the Max-log-MAP algorithm would incorrectly decode both bits u_1 and u_2 if decoding was stopped after one iteration. In Figure 16.19(m), (n), and (o), we give the approximate L-values $L_e^{(1)}(u_l)$, $L_e^{(2)}(u_l)$, and $L^{(2)}(u_l)$, respectively, $l = 0, 1, 2, 3$, for a second iteration of Max-log-MAP decoding (see Problem 16.26). We see from Figure 16.19(o) that if a final decoding decision was made after two iterations of decoding, the Max-log-MAP algorithm would decode all bits correctly, the same as the log-MAP algorithm.

The major lesson of Examples 16.14 and 16.15 can be stated as follows:

Fundamental Principle of Turbo Decoding [16]
Iteratively estimate information symbols using the MAP algorithm, whose inputs include successively updated a priori L-values obtained from all statistically independent information available after the previous iteration of decoding.

We now summarize our discussion of iterative decoding using the log-MAP and Max-log-MAP algorithms:

- The extrinsic a posteriori L-values are no longer strictly independent of the other terms after the first iteration of decoding, which causes the performance improvement from successive iterations to diminish over time.

- The concept of iterative decoding is similar to negative feedback in control theory, in the sense that the extrinsic information from the output that is fed back to the input has the effect of amplifying the SNR at the input, leading to a stable system output.

- Decoding speed can be improved by a factor of 2 by allowing the two decoders to work in parallel. In this case, the a priori L-values for the first iteration of decoder 2 will be the same as for decoder 1 (normally equal to 0), and the extrinsic a posteriori L-values will then be exchanged at the same time prior to each succeeding iteration.

- After a sufficient number of iterations, the final decoding decision can be taken from the a posteriori L-values of either decoder, or from the average of these values, without noticeably affecting performance.

- As noted earlier, the L-values of the parity bits remain constant throughout decoding. In *serially concatenated* iterative decoding systems, however, parity bits from the outer decoder enter the inner decoder, and thus the L-values of these parity bits must be updated during the iterations [25].

- The foregoing approach to iterative decoding is ineffective for nonsystematic constituent codes, since channel L-values for the information bits are not available as inputs to decoder 2; however, the iterative decoder of Figure 16.6 can be modified to decode PCCCs with nonsystematic constituent codes [42].

- A row–column interleaver was used in Examples 16.14 and 16.15 for illustrative purposes only. As noted previously, better performance is normally achieved with pseudorandom interleavers, particularly for large block lengths, and the iterative decoding procedure remains the same.

- Further iterations of decoding in these examples would most likely improve the a posteriori L-values of the input bits; that is, $L^{(2)}(u_0)$ and $L^{(2)}(u_2)$ would become more negative, and $L^{(2)}(u_1)$ and $L^{(2)}(u_3)$ would become more positive (see Problem 16.27). It is possible, however, particularly on very noisy channels, for the decoder to converge to the correct decision and then diverge again, or even to "oscillate" between correct and incorrect decisions.

- Iterations can be stopped after some fixed number, typically in the range 10–20 for most turbo codes, or stopping rules based on reliability statistics can be used to halt decoding.
- The Max-log-MAP algorithm is simpler to implement than the log-MAP algorithm; however, it typically suffers a performance degradation of about 0.5 dB.
- It can be shown that the Max-log-MAP algorithm is equivalent to the SOVA algorithm presented in Chapter 12 [43].

The effect of the number of decoding iterations on BER performance is illustrated in Figure 16.20 for the rate $R = 1/2$ PCCC with information block length $K = 2^{16} = 65536$ obtained by using a pseudorandom interleaver and puncturing alternate parity bits from the encoder of Figure 16.1(b). The performance improvement with increasing iterations, as well as the diminishing effect as the number of iterations gets large, is clearly evident from the figure. This is the original code proposed in the paper by Berrou, Glavieux, and Thitimajshima [2] that launched the field of turbo coding. After 18 iterations, a BER of 10^{-5} is achieved at an SNR of $E_b/N_0 = 0.7$ dB, which is only 0.5 dB away from the Shannon limit for channels with binary inputs.

We conclude this chapter with a brief discussion of stopping rules for iterative decoding, that is, methods of detecting when decoding has converged. One such method is based on the *cross-entropy* (CE) of the APP distributions at the outputs of

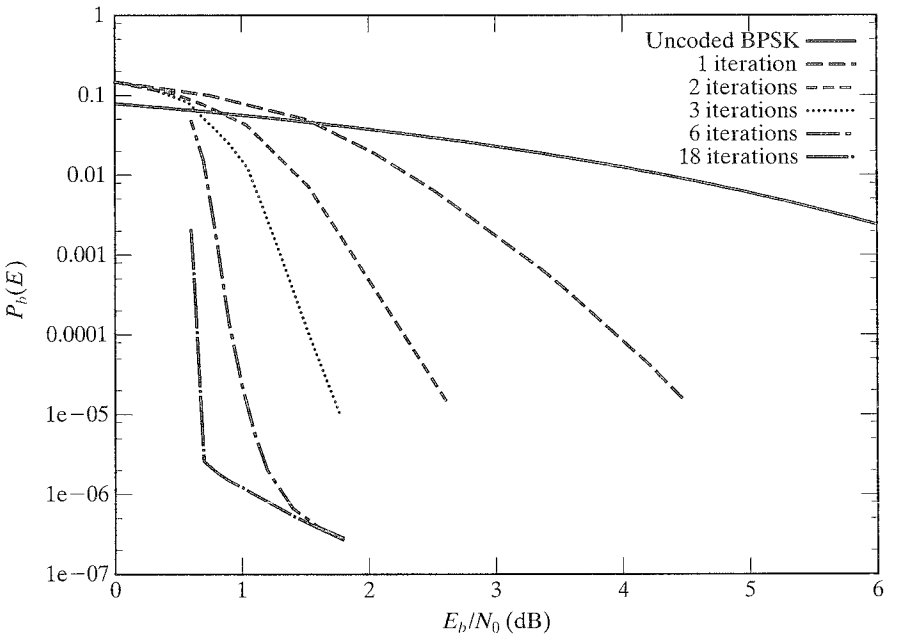


FIGURE 16.20: The effect of iterations on decoding performance.

the two decoders. The cross-entropy $D(P \,||\, Q)$ of two joint probability distributions $P(\mathbb{u})$ and $Q(\mathbb{u})$, assuming statistical independence of the bits in the vector $\mathbb{u} = [u_0, u_1, \cdots, u_l, \cdots, u_{K-1}]$, is defined as (see [41])

$$D(P \,||\, Q) = E_P\left\{\log \frac{P(\mathbb{u})}{Q(\mathbb{u})}\right\} = \sum_{l=0}^{K-1} E_P\left\{\log \frac{P(u_l)}{Q(u_l)}\right\}, \tag{16.127}$$

where $E_P\{\cdot\}$ denotes expectation with respect to the probability distribution $P(u_l)$. $D(P \,||\, Q)$ is a measure of the closeness of two distributions, and

$$D(P \,||\, Q) = 0 \text{ iff } P(u_l) = Q(u_l),\ u_l = \pm 1,\ l = 0, 1, \cdots, K-1. \tag{16.128}$$

Thus, when two distributions are nearly identical, the cross-entropy $D(P \,||\, Q)$ is close to 0, and the value of $D(P \,||\, Q)$ can be used as an indication of when iterative decoding has converged and the decoding iterations can be stopped.

The CE stopping rule is based on the difference between the a posteriori L-values after successive iterations at the outputs of the two decoders. For example, let

$$L_{(i)}^{(1)}(u_l) = L_c r_{ul} + L_{a(i)}^{(1)}(u_l) + L_{e(i)}^{(1)}(u_l) \tag{16.129a}$$

represent the a posteriori L-value at the output of decoder 1 after iteration i, and let

$$L_{(i)}^{(2)}(u_l) = L_c r_{ul} + L_{a(i)}^{(2)}(u_l) + L_{e(i)}^{(2)}(u_l) \tag{16.129b}$$

represent the a posteriori L-value at the output of decoder 2 after iteration i. Now, using the facts that $L_{a(i)}^{(1)}(u_l) = L_{e(i-1)}^{(2)}(u_l)$ and $L_{a(i)}^{(2)}(u_l) = L_{e(i)}^{(1)}(u_l)$, that is, the a priori L-value at the input of one decoder is the extrinsic a posteriori L-value at the output of the other decoder, and letting $Q(u_l)$ and $P(u_l)$ represent the a posteriori probability distributions at the outputs of decoders 1 and 2, respectively, we can write

$$L_{(i)}^{(Q)}(u_l) = L_c r_{ul} + L_{e(i-1)}^{(P)}(u_l) + L_{e(i)}^{(Q)}(u_l) \tag{16.130a}$$

and

$$L_{(i)}^{(P)}(u_l) = L_c r_{ul} + L_{e(i)}^{(Q)}(u_l) + L_{e(i)}^{(P)}(u_l). \tag{16.130b}$$

We can now write the difference in the two soft outputs as

$$L_{(i)}^{(P)}(u_l) - L_{(i)}^{(Q)}(u_l) = L_{e(i)}^{(P)}(u_l) - L_{e(i-1)}^{(P)}(u_l) \triangleq \Delta L_{e(i)}^{(P)}(u_l); \tag{16.131}$$

that is, $\Delta L_{e(i)}^{(P)}(u_l)$ represents the difference in the extrinsic a posteriori L-values of decoder 2 in two successive iterations.

We now compute the CE of the a posteriori probability distributions $P(u_l)$ and $Q(u_l)$ as follows:

$$\begin{aligned} E_P\left\{\log \frac{P(u_l)}{Q(u_l)}\right\} &= P(u_l = +1) \log \frac{P(u_l = +1)}{Q(u_l = +1)} + P(u_l = -1) \log \frac{P(u_l = -1)}{Q(u_l = -1)} \\ &= \frac{e^{L_{(i)}^{(P)}(u_l)}}{1 + e^{L_{(i)}^{(P)}(u_l)}} \log \frac{e^{L_{(i)}^{(P)}(u_l)}}{1 + e^{L_{(i)}^{(P)}(u_l)}} \cdot \frac{1 + e^{L_{(i)}^{(Q)}(u_l)}}{e^{L_{(i)}^{(Q)}(u_l)}} + \\ &\quad \frac{e^{-L_{(i)}^{(P)}(u_l)}}{1 + e^{-L_{(i)}^{(P)}(u_l)}} \log \frac{e^{-L_{(i)}^{(P)}(u_l)}}{1 + e^{-L_{(i)}^{(P)}(u_l)}} \cdot \frac{1 + e^{-L_{(i)}^{(Q)}(u_l)}}{e^{-L_{(i)}^{(Q)}(u_l)}}, \end{aligned} \tag{16.132}$$

where we have used expressions for the a posteriori distributions $P(u_l = \pm 1)$ and $Q(u_l = \pm 1)$ analogous to those given in (12.123). It can be shown (see Problem 16.30) that (16.132) simplifies to

$$E_P\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} = -\frac{\Delta L_{e(i)}^{(P)}(u_l)}{1+e^{L_{(i)}^{(P)}(u_l)}} + \log\frac{1+e^{-L_{(i)}^{(Q)}(u_l)}}{1+e^{-L_{(i)}^{(P)}(u_l)}}. \tag{16.133}$$

This expression can be simplified further under the assumption that the hard decisions based on the signs of the a posteriori L-values of both decoders do not change once decoding has converged; that is, the hard decisions after iteration i, $\hat{u}_l^{(i)}$, satisfy

$$\hat{u}_l^{(i)} = \operatorname{sgn}\left[L_{(i)}^{(P)}(u_l)\right] = \operatorname{sgn}\left[L_{(i)}^{(Q)}(u_l)\right]. \tag{16.134}$$

Using (16.134) and noting that

$$\left|L_{(i)}^{(P)}(u_l)\right| = \operatorname{sgn}\left[L_{(i)}^{(P)}(u_l)\right] L_{(i)}^{(P)}(u_l) = \hat{u}_l^{(i)} L_{(i)}^{(P)}(u_l) \tag{16.135a}$$

and

$$\left|L_{(i)}^{(Q)}(u_l)\right| = \operatorname{sgn}\left[L_{(i)}^{(Q)}(u_l)\right] L_{(i)}^{(Q)}(u_l) = \hat{u}_l^{(i)} L_{(i)}^{(Q)}(u_l), \tag{16.135b}$$

we can show (see Problem 16.30) that (16.133) simplifies further to

$$E_P\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} \approx \frac{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)}{1+e^{\left|L_{(i)}^{(P)}(u_l)\right|}} + \log\frac{1+e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}}{1+e^{-\left|L_{(i)}^{(P)}(u_l)\right|}}. \tag{16.136}$$

We now use the facts that once decoding has converged, the magnitudes of the a posteriori L-values are large; that is,

$$\left|L_{(i)}^{(P)}(u_l)\right| \gg 0 \quad \text{and} \quad \left|L_{(i)}^{(Q)}(u_l)\right| \gg 0, \tag{16.137}$$

and that when x is large, e^{-x} is small, and

$$1 + e^{-x} \approx 1 \quad \text{and} \quad \log\left(1+e^{-x}\right) \approx e^{-x}. \tag{16.138}$$

Applying these approximations to (16.136), we can show that (see Problem 16.30)

$$E_P\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} \approx e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left[1 - e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)}\left(1+\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right)\right]. \tag{16.139}$$

Noting that the magnitude of $\Delta L_{e(i)}^{(P)}(u_l)$ will be smaller than 1 when decoding converges, we can approximate the term $e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)}$ using the first two terms of its series expansion as follows:

$$e^{-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)} \approx 1 - \hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l), \tag{16.140}$$

which leads to the simplified expression

$$E_P\left\{\log\frac{P(u_l)}{Q(u_l)}\right\} \approx e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left[1-\left(1-\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right)\left(1+\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right)\right]$$

$$= e^{-\left|L_{(i)}^{(Q)}(u_l)\right|}\left[\hat{u}_l^{(i)}\Delta L_{e(i)}^{(P)}(u_l)\right]^2 = \frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}}. \tag{16.141}$$

Now, using (16.127) and (16.141) we can write the CE of the probability distributions $P(\mathbb{u})$ and $Q(\mathbb{u})$ at iteration i as

$$D_{(i)}(P \parallel Q) \triangleq E_P\left\{\log\frac{P(\mathbb{u})}{Q(\mathbb{u})}\right\} \approx \sum_{l=0}^{K-1}\frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}}, \tag{16.142}$$

where we note that the statistical independence assumption does not hold exactly as the iterations proceed.

We next define

$$T(i) \triangleq \sum_{l=0}^{K-1}\frac{\left|\Delta L_{e(i)}^{(P)}(u_l)\right|^2}{e^{\left|L_{(i)}^{(Q)}(u_l)\right|}} \tag{16.143}$$

as the approximate value of the CE at iteration i given by (16.142). $T(i)$ can be computed after each iteration. (Note that in computing $T(1)$ we need to find $\Delta L_{e(1)}^{(P)}(u_l) = L_{e(1)}^{(P)}(u_l) - L_{e(0)}^{(P)}(u_l)$, where $L_{e(0)}^{(P)}(u_l)$ is taken to be the initial a priori L-value of the information bit u_l; that is, $L_{e(0)}^{(P)}(u_l) = 0$, $l = 0, 1, \cdots, K-1$, for equally likely inputs.) Experience with computer simulations has shown that once convergence is achieved, $T(i)$ drops by a factor of 10^{-2} to 10^{-4} compared with its initial value, and thus it is reasonable to use

$$T(i) < 10^{-3}T(1) \tag{16.144}$$

as a stopping rule for iterative decoding.

The observation that led to the assumption of (16.134) can also be used to define a simpler stopping rule based only on the hard-decision outputs of each decoder. For example, if (16.134) is satisfied for some number of consecutive iterations (five is a reasonable choice), for all $l = 0, 1, \cdots, K-1$, we can declare that convergence has occurred and stop decoding. Although not quite as effective in detecting convergence as the CE rule, this hard-decision-aided (HDA) rule is extremely simple to implement. The CE stopping rule was discussed in [15], and two HDA stopping rules were introduced in [44].

Another approach to stopping the iterations in turbo decoding is to concatenate a high-rate outer cyclic code with an inner turbo code, as illustrated in Figure 16.21. After each iteration, the hard-decision output of the turbo decoder is used to check the syndrome of the cyclic code. If no errors are detected, decoding is assumed correct and the iterations are stopped. It is important to choose an outer code with a low undetected error probability, so that iterative decoding is not stopped

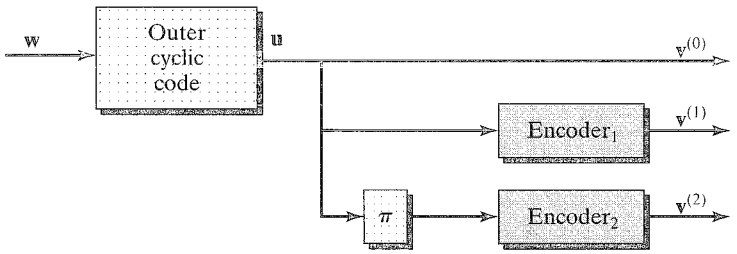


FIGURE 16.21: A concatenation of an outer cyclic code with an inner turbo code.

prematurely. For this reason it is usually advisable not to check the syndrome of the outer code during the first few iterations, when the probability of undetected error may be larger than the probability that the turbo decoder is error free. This method of stopping the iterations is particularly effective for large block lengths, since in this case the rate of the outer code can be made very high, thus resulting in a negligible overall rate loss.

For large block lengths, the foregoing idea can be extended to include outer codes, such as BCH codes, that can correct a small number of errors and still maintain a low undetected error probability. In this case, the iterations are stopped once the number of hard-decision errors at the output of the turbo decoder is within the error-correcting capability of the outer code. This method also provides a low word-error probability for the complete system; that is, the probability that the entire information block contains one or more decoding errors can be made very small. The idea of combining a turbo code with a high-rate outer BCH code was introduced in [45] and further analyzed in [46].

PROBLEMS

16.1 Prove that the general rate $R = 1/3$ turbo encoder shown in Figure 16.1(a), where encoders 1 and 2 are linear convolutional encoders (not necessary identical) separated by an arbitrary interleaver, is a linear system.

16.2 For the length $K = 16$ quadratic interleaver of (16.7), determine all pairs of indices that are interchanged by the permutation.

16.3 Consider a PCBC with two different constituent codes: the $(7, 4, 3)$ Hamming code and the $(8, 4, 4)$ extended Hamming code. Find the CWEFs, IRWEFs, and WEFs of this code assuming a uniform interleaver.

16.4 Find the IRWEFs and WEFs for Example 16.5.

16.5 Repeat Example 16.5 for the case $h = 4$. What is the minimum distance of the $(40, 16)$ PCBC if a 4×4 row–column (block) interleaver is used?

16.6 Consider a PCBC with the $(24, 12, 8)$ extended Golay code in systematic form as the constituent code.

a. Find the CWEFs $A_w(Z)$, $w = 1, 2, \cdots, 12$, of this code by generating the codewords of the $(23, 12, 7)$ Golay code in systematic form and then adding an overall parity check.

Assuming a uniform interleaver,

b. find the CWEFs $A_w^{PC}(Z)$, $w = 1, 2, \cdots, 12$;

c. find the IRWEFs $A^{PC}(W, Z)$ and $B^{PC}(W, Z)$; and

d. find the WEFs $A^{PC}(X)$ and $B^{PC}(X)$.

e. Now, consider a PCBC with the 12-repeated $(24, 12, 8)$ extended Golay code in systematic form as the constituent code. Assume that the information bits are arranged in a square array and that a row–column interleaver is used; that is, encoder 1 encodes across rows of the array, and encoder 2 encodes down columns of the array. Find the parameters (n, k, d) of the PCBC.

16.7 Prove (16.37).

16.8 Find the codeword IRWEF and WEF for the PCCC in Example 16.6.

16.9 Find the bit IRWEFs and WEFs for the PCCC in Example 16.6.

16.10 Repeat Example 16.6 for the encoder with the reversed generators given in (16.62).

16.11 Repeat Example 16.7 for the encoder with the reversed generators given in (16.62).

16.12 Find the multiplicity of weight-8 codewords in Example 16.8.

16.13 Repeat Example 16.8 using the feedforward encoder

$$\mathbb{G}_{ff}(D) = \begin{bmatrix} 1 & 1 + D + D^2 \end{bmatrix}$$

for the second constituent code.

16.14 Consider a rate $R = 1/4$ multiple turbo code (PCCC) with constituent encoder

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

separated by two random interleavers (see Figure 16.2). Assuming a uniform interleaver and large block size K,

a. find the approximate CWEFs $A_w^{PC}(Z)$ and $B_w^{PC}(Z)$ for $w = 2, 3, 4, 5$;

b. find the approximate IRWEFs $A^{PC}(W, Z)$ and $B^{PC}(W, Z)$;

c. find the approximate WEFs $A^{PC}(X)$ and $B^{PC}(X)$; and

d. sketch the union bounds on $P_w(E)$ and $P_b(E)$ for $K = 1000$ and $K = 10000$, assuming a binary-input, unquantized-output AWGN channel.

16.15 Find the minimum-weight codewords corresponding to input weights 2 and 3 for the PCCCs whose generators are given in Table 16.6. In each case determine the free distance d_{free} assuming large K.

16.16 Show that for any $(n, 1, \nu)$ systematic feedforward encoder $A_w^{(w)}(Z) = \left[A_1^{(1)}(Z)\right]^w$ and that for any $(n, 1, \nu)$ systematic feedback encoder $A_{2w}^{(w)}(Z) = \left[A_2^{(1)}(Z)\right]^w$.

16.17 Show that a weight-1 input sequence cannot terminate an $(n, 1, \nu)$ systematic feedback encoder, but there always exists a weight-2 input sequence, of degree no greater than $2^\nu - 1$, that does terminate the encoder.

16.18 For an $(n, 1, \nu)$ systematic feedback encoder, show that the input sequence $\mathbf{u} = (1000\cdots)$ produces a cycle with input weight zero starting in state $S_1 = (10\cdots0)$, arriving in state $S_{2^\nu-1} = (0\cdots01)$ after at most $2^\nu - 2$ steps, and returning to state S_1 in one step.

16.19 For an $(n, 1, \nu)$ systematic feedback encoder, show that a 1 input from state $S_{2^\nu-1} = (0\cdots01)$ terminates the encoder.

16.20 Compute z_{min} and d_{eff} for the turbo codes with primitive and nonprimitive 16-state constituent codes D and E of Table 16.6.

16.21 Show that the bound of (16.97) is also valid for nonprimitive denominator polynomials.

16.22 Use the S-random interleaver algorithm to construct a length $K = 32$ permutation that breaks up all weight-2 input sequences with a spacing of $S = 4$ or less between 1's.

16.23 Prove that the Gaussian random variable $L_c r_l^{(0)}$ in (16.17) has variance $2L_c$ and mean $\pm L_c$.

16.24 Prove that for any real numbers w, x, and y, $\max^*(w+x, w+y) = w+\max^*(x, y)$.

16.25 Verify the α^* and β^* expressions in (16.114).

16.26 Verify all entries in Figure 16.19 that were not computed in the text.

16.27 Complete two more iterations of decoding in Examples 16.14 and 16.15. Is there any change in the decoded output?

16.28 Calculate the cross-entropy at the end of each complete iteration in Examples 16.14 and 16.15 and Problem 16.27.

16.29 [15] Consider the (8, 4, 3) PCBC formed by using $h = 2$ codewords from the (3, 2, 2) systematic single parity check (SPC) code, that is, a (6, 4, 2) 2-repeated SPC code, as the constituent code, along with a 2×2 block (row–column) interleaver of overall size $K = 4$. The information block is given by the vector $\mathbf{u} = [u_{11}, u_{12}, u_{21}, u_{22}]$, where u_{ij} represents the jth information bit in the ith row of the interleaver, $i, j = 1, 2$; the ith parity bit in the row code is given by $p_i^{(1)}, i = 1, 2$; and the jth parity bit in the column code is given by $p_j^{(2)}, j = 1, 2$. The arrangement is shown in Figure P-16.29(a). Assume the particular bit values given in Figure P-16.29(b) and the set of received channel L-values given in Figure P-16.29(c).

a. Use the trellis shown in Figure P-16.29(d) and the log-MAP algorithm to compute the extrinsic L-values for the first iteration of row and column decoding, and the soft-output L-values after the first complete iteration for each of the $K = 4$ information bits.

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

16.30 Starting from (16.132), derive the cross-entropy expressions given in (16.133), (16.136), and (16.139).

u_{11}	u_{12}	$p_1^{(1)}$
u_{21}	u_{22}	$p_2^{(1)}$
$p_1^{(2)}$	$p_2^{(2)}$	

(8, 4, 3) PCBC

(a)

−1	−1	−1
−1	+1	+1
−1	+1	

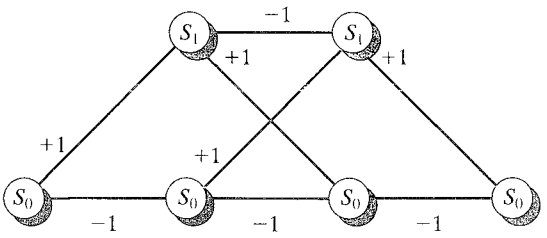
Coded values

(b)

−0.5	−1.5	−1.0
−4.0	−1.0	+1.5
−2.0	+2.5	

Received L-values

(c)



Decoding trellis

(d)

FIGURE P-16.29

BIBLIOGRAPHY

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, 27: 379–423, July 1948.

2. C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," *Proc. IEEE Intl. Conf. Commun. (ICC 93)*, pp. 1064–70, Geneva, Switzerland, May 1993.

3. C. Berrou and A. Glavieux, "Near Optimum Error Correcting and Decoding: Turbo-Codes," *IEEE Trans. Commun.*, COM-44: 1261–71, October 1996.

4. C. Berrou and A. Glavieux, "Reflections on the Prize Paper: Near Optimum Error Correcting Coding and Decoding: Turbo-Codes," *IEEE Inform. Theory Soc. Newsletter*, 48(2): 24–31, June 1998.

5. G. Battail, M. Decouvelaere, and P. Godlewski, "Replication Decoding," *IEEE Trans. Inform. Theory*, IT-25: 332–45, May 1979.

6. G. Battail, "Building Long Codes by Combination of Simple Ones, Thanks to Weighted-Output Decoding," *Proc. URSI Intl. Symp. Signals, Systems, and Electronics (ISSSE 1989)*, pp. 634–37, Erlangen, Germany, September 1989.

7. J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *Proc. IEEE Global Commun. Conf. (GLOBECOM 1989)*, pp. 1680–86, Dallas, Tex., November 1989.

8. J. H. Lodge, P. Hoeher, and J. Hagenauer, "The Decoding of Multidimensional Codes Using Separable MAP filters," *Proc. 16th Biennial Symp. Commun.*, pp. 343–46, Kingston, Ontario, May 1992.

9. J. D. Andersen, "The Turbo Coding Scheme," presented at *IEEE Intl. Symp. Inform. Theory (ISIT 1994)*, Trondheim, Norway, June 1994.

10. P. Robertson, "Illuminating the Structure of Code and Decoder of Parallel Concatenated Recursive Systematic (Turbo) Codes," *Proc. IEEE Global Commun. Conf. (GLOBECOM 1994)*, pp. 1298–1303, San Francisco, Calif., November 1994.

11. S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding," *IEEE Trans. Inform. Theory*, IT-42: 409–28, March 1996.

12. S. Benedetto and G. Montorsi, "Design of Parallel Concatenated Convolutional Codes," *IEEE Trans. Commun.*, COM-44: 591–600, May 1996.

13. L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A Distance Spectrum Interpretation of Turbo Codes," *IEEE Trans. Inform. Theory*, IT-42: 1698–1709, November 1996.

14. G. Battail, "A Conceptual Framework for Understanding Turbo Codes," *IEEE J. Select. Areas Commun.*, SAC-16: 245–54, February 1998.

15. J. Hagenauer, E. Offer, and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, IT-42: 429–45, March 1996.

16. J. Hagenauer, "The Turbo Principle: Tutorial Introduction and State of the Art," *Proc. 1st Intl. Symp. Turbo Codes*, pp. 1–11, Brest, France, September 1997.

17. R. J. McEliece, D. J. C. MacKay, and J.-F. Cheng, "Turbo Decoding As an Instance of Pearl's Belief Propagation Algorithm," *IEEE J. Select. Areas Commun.*, SAC-16: 140–52, February 1998.

18. A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE J. Select. Areas Commun.*, SAC-16: 260–64, February 1998.

19. D. Divsalar and F. Pollara, "Multiple Turbo Codes for Deep-Space Communications," TDA Progress Report 42–121, Jet Propulsion Laboratory, Pasadena, Calif., May 1995.

20. D. Divsalar and F. Pollara, "On the Design of Turbo Codes," TDA Progress Report 42-123, Jet Propulsion Laboratory, Pasadena, Calif., November 1995.

21. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A Soft-Input Soft-Output Maximum a Posteriori (MAP) Module to Decode Parallel and Serial Concatenated Codes," TDA Progress Report 42-127, Jet Propulsion Laboratory, Pasadena, Calif., November 1996.

22. S. Dolinar, D. Divsalar, and F. Pollara, "Code Performance As a Function of Block Size," TMO Progress Report 42-133, Jet Propulsion Laboratory, Pasadena, Calif., May 1998.

23. D. Divsalar and F. Pollara, "Serial and Hybrid Concatenated Codes with Applications," *Proc. 1st Intl. Symp. Turbo Codes*, pp. 80–87, Brest, France, September 1997.

24. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Analysis, Design and Iterative Decoding of Double Serially Concatenated Codes with Interleavers," *IEEE J. Select. Areas Commun.*, SAC-42: 231–44, February 1998.

25. S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design and Iterative Decoding," *IEEE Trans. Inform. Theory*, IT-44: 909–26, May 1998.

26. C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic, Boston, Mass., 1999.

27. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann, San Mateo, Calif., 1988.

28. B. Vucetic and J. Yuan, *Turbo Codes*. Kluwer Academic, Boston, Mass., 2000.

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

30. P. C. Massey and D. J. Costello Jr., "New Developments in Asymmetric Turbo Codes," *Proc. 2nd Intl. Symp. Turbo Codes*, pp. 93–100, Brest, France, September 2000.

31. P. C. Massey and D. J. Costello, Jr., "Turbo Codes with Recursive Nonsystematic Quick-Look-In Constituent Codes," *Proc. IEEE Intl. Symp. Inform. Theory (ISIT 2001)*, p. 141, Washington, DC, June 2001.

32. O. Y. Takeshita and D. J. Costello, Jr., "New Deterministic Interleaver Designs for Turbo Codes," *IEEE Trans. Inform. Theory*, IT-46: 1988–2006, September 2000.

33. I. Sason and S. Shamai (Shitz), "Improved Upper Bounds on the ML Decoding Error Probability of Parallel and Serial Concatenated Turbo Codes via Their Ensemble Distance Spectrum," *IEEE Trans. Inform. Theory*, IT-46: 24–47, January 2000.

34. S. Benedetto, R. Garello, and G. Montorsi, "A Search for Good Convolutional Codes to Be Used in the Construction of Turbo Codes," *IEEE Trans. Commun.*, COM-46: 1101–05, September 1998.

35. S. Dolinar and D. Divsalar, "Weight Distribution for Turbo Codes Using Random and Nonrandom Permutations," TDA Progress Report 42-122, Jet Propulsion Laboratory, Pasadena, Calif., August 1995.

36. S. ten Brink, "Convergence of Iterative Decoding," *IEE Electron. Lett.*, 35: 806–8, May 1999.

37. T. Richardson, "The Geometry of Turbo-Decoding Dynamics," *IEEE Trans. Inform. Theory*, IT-46: 9–23, January 2000.

38. D. Divsalar, S. Dolinar, and F. Pollara, "Iterative Turbo Decoder Analysis Based on Density Evolution," *IEEE J. Select. Areas Commun.*, SAC-19: 891–907, May 2001.

39. H. El Gamal and A. R. Hammons, Jr., "Analyzing the Turbo Decoder Using the Gaussian Approximation," *IEEE Trans. Inform. Theory*, IT-47: 671–86, February 2001.

40. N. Wiberg, H. A. Loeliger, and R. Koetter, "Codes and Iterative Decoding on General Graphs," *Eur. Trans. Telecommun.*, 513–26, September 1995.

41. T. Cover and J. Thomas, *Elements of Information Theory*, Wiley Interscience, New York, 1991.

42. O. M. Collins, O. Y. Takeshita, and D. J. Costello, Jr., "Iterative Decoding of Non-Systematic Turbo Codes," in *Proc. IEEE Intl. Symp. Inform. Theory (ISIT 2000)*, p. 172, Sorrento, Italy, June 2000.

43. M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the Equivalence between SOVA and Max-Log-MAP Decoding," *IEEE Commun. Lett.*, 2: 137–39, May 1998.

44. R. Y. Shao, S. Lin, and M. P. Fossorier, "Two Simple Stopping Criteria for Turbo Decoding," *IEEE Trans. Commun.*, COM-47: 1117–20, August 1999.

45. J. D. Andersen, "Turbo Codes Extended with Outer BCH Code," in *IEE Electron. Lett.*, 32: 2059–60, October 1996.

46. O. Y. Takeshita, O. M. Collins, P. C. Massey, and D. J. Costello, Jr. , "On the Frame Error Rate of Concatenated Turbo-Codes," *IEEE Trans. Commun.*, COM-49: 602–8, April 2001.