Chapter 8
Error Control Coding

Channel coding and interleaving techniques have long been used for combating noise, interference, jamming, fading, and other channel impairments. The basic idea of channel coding is to introduce controlled redundancy into the transmitted signals that is subsequently exploited at the receiver for error correction. Channel coding can also be used for error detection in schemes that use automatic repeat request (ARQ) strategies. ARQ strategies must have a feedback channel to relay the retransmission requests from the receiver back to the transmitter when errors are detected. ARQ schemes require buffering at the transmitter and/or receiver and, therefore, are suitable for data applications but are not suitable for delay sensitive applications such as voice or real-time video. Hybrid ARQ schemes use both error correction and error detection; the code is used to correct the most likely error patterns and to detect the more infrequently occurring error patterns. Upon detection of errors, a retransmission is requested.

There are many different types of error correcting codes, but historically they have been classified into block codes and convolutional codes. Both block codes and convolutional codes find potential applications in wireless systems. To generate a codeword of an (n,k) block code, a block of k data bits is appended by $n-k$ redundant parity bits that are algebraically related to the k data bits, thereby producing a codeword consisting of n code bits. The ratio $R_c = k/n$ is called the code rate, where $0 < R_c \leq 1$. Convolutional codes, on the other hand, are generated by the discrete-time convolution of the input data sequence with the impulse response of the encoder. The memory of the encoder is measured by the duration of the impulse response. While block encoder operates on k-bit blocks of data bits, a convolutional encoder accepts a continuous sequence of input data bits.

In the early application of coding to digital communications, the modulator and coder were treated as separate entities. Hence, a block code or a convolutional code was used to obtain a coding gain at the cost of bandwidth expansion or data rate. Although this approach may be feasible for power limited channels where bandwidth resources are plentiful, it is undesirable and sometimes not even possible for bandwidth limited applications such as cellular radio. If no sacrifices of data rate or bandwidth can be made, then schemes that separate the operations

G.L. Stüber, *Principles of Mobile Communication*, DOI 10.1007/978-1-4614-0364-7_8,

of coding and modulation require a very powerful code just to break even with an uncoded system. In 1974, Massey [173] suggested that the performance of a coded digital communication system could be improved by treating coding and modulation as a single entity. Ungerboeck later developed the basic principles of trellis-coded modulation (TCM) [258] and identified classes of trellis codes that provide substantial coding gains on bandwidth limited additive white Gaussian noise (AWGN) channels.

TCM schemes combine the operations of coding and modulation and can be viewed as a generalization of convolutional codes. While convolutional codes attempt to maximize the minimum Hamming distance between allowed code symbol sequences, trellis-codes attempt to maximize the Euclidean distance between allowed code symbol sequences. By jointly designing the encoder and modulator Ungerboeck showed that, for an AWGN channel, coding gains of 3–6 dB could be obtained relative to an uncoded system using trellis codes with 4–128 encoder states, without sacrificing bandwidth or data rate. This property makes TCM very attractive for wireless applications where high spectral efficiency is needed due to limited bandwidth resources, and good power efficiency is needed to extend battery life in portable devices. TCM experienced an almost immediate and widespread application into high-speed power-efficient and bandwidth-efficient digital modems. In 1984, a variant of the Ungerboeck eight-state 2D trellis code was adopted by CCITT for both 14.4 kb/s leased-line modems and the 9.6 kb/s switched-network modems [36]. In 1985, a TCM-based modem operating at 19.2 kb/s was introduced by Codex [259].

Ungerboeck's work [258] captured the attention of the coding community and laid the foundation for intensified research. Calderbank and Mazo introduced an analytic description of trellis codes [43]. They showed how to realize the two operations (coding and mapping) in Ungerboeck's codes using a single-step procedure. Calderbank and Sloane [44] and Wei [278] proposed multidimensional trellis codes. Spaces with larger dimensionality are attractive, because the signals are spaced at larger Euclidean distances [36]. Calderbank and Sloan [44] and Forney [105] made the observation that the signal constellation should be regarded as a finite set of points taken from an infinite lattice, and the partitioning of the constellation into subsets corresponds to the partitioning of the lattice into a sublattice and its cosets. They then developed a new class of codes, called coset codes, based on this principle.

Many studies have examined the performance of TCM on interleaved flat fading channels [45, 76, 77, 85]. Divsalar and Simon [77, 78] constructed trellis codes that are effective for interleaved flat Ricean and Rayleigh fading channels. Interleaving randomizes the channel with respect to the transmitted symbol sequence and has the effect of reducing the channel memory. Consequently, interleaving improves the performance of codes that have been designed for memoryless channels. Moreover, trellis codes that are designed for flat fading channels exhibit time diversity when combined with interleaving of sufficient depth. It was reported in [45] that interleaving with reasonably long interleaving depths is almost as good as ideal infinite interleaving. The design of trellis codes for interleaved flat fading channels is

not guided by the minimum Euclidean distance used for AWGN channels, but rather by the minimum product squared Euclidean distance (MPSD) and the minimum built-in time diversity (MTD) between any two allowed code symbol sequences. Wei [279] introduced an additional design parameter called the minimum decoding depth and proposed a set of efficient codes for interleaved flat Rayleigh fading channels.

Many studies have also considered the performance of trellis codes on inter-symbol interference (ISI) channels [81, 93, 251, 286]. The coded performance on *static* ISI channels may be significantly degraded compared to that on ISI-free channels. Receivers for TCM on static ISI channels typically use a linear forward equalizer followed by a soft decision Viterbi decoder. For channels with severe ISI, a more appropriate approach is to use a decision feedback equalizer (DFE) in front of the TCM decoder to avoid the problems of noise enhancement. However, the feedback section of the DFE requires that decisions be available with zero delay. Since the zero-delay decisions are unreliable, the performance improvement using the DFE is marginal [50]. It is possible that the performance can be improved if equalization and decoding is performed in a joint manner using maximum likelihood sequence estimation (MLSE) or some other form of sequence estimator. However, the complexity of an MLSE receiver grows exponentially with the number of encoder states and the length of the channel vector.

In 1993, Berrou et al. introduced *parallel* concatenated convolutional codes (PCCCs), called turbo coding [35]. When used in conjunction with an iterative decoding scheme, PCCCs achieve near Shannon capacity limit performance on both the AWGN channel and the interleaved flat fading channel. Simulations of a rate-1/2 turbo code in [35] showed a bit error probability of 10^{-5} at an $E_b/N_o = 0.5$ dB, which is only 0.5 dB from the Shannon capacity limit! Although, the performance of turbo codes is remarkable at low E_b/N_o, their performance at high E_b/N_o is unimpressive. There is a perceivable change in the slope of the bit error rate (BER) curves, which has been loosely termed an "error floor." In 1997, Benedetto et al. showed that iterative decoding of serially concatenated interleaved convolutional codes (SCCCs) can provide large coding gains without the problem of an error floor [32]. In general, SCCCs outperform PCCCs at high E_b/N_o, whereas the opposite is true for low E_b/N_o.

The remainder of this chapter is organized as follows. Section 8.1 gives an introduction to block codes and space-time block codes. Section 8.2 introduces convolutional codes, and decoding algorithms for convolutional codes, including the Viterbi algorithm and BCJR algorithm. Section 8.3 introduces TCM. This is followed the performance analysis of convolutional and trellis codes on AWGN channels in Sect. 8.4. Section 8.5 considers block and convolutional interleavers that are useful for coding on fading channels. This is followed by a consideration of the design and performance analysis of trellis codes on interleaved flat fading channels in Sect. 8.6. The performance of space-time codes and the decoding of space-time codes is considered in Sect. 8.7. Finally, Sect. 8.8 provides a treatment of parallel and serial turbo codes.

8.1 Block Codes

8.1.1 Binary Block Codes

A binary block encoder accepts a length-k input vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$, $a_i \in \{0,1\}$, and generates a length-n codeword $\mathbf{c} = (c_1, c_2, \ldots, c_n)$, $c_i \in \{0,1\}$, through the linear mapping $\mathbf{c} = \mathbf{aG}$, where $\mathbf{G} = [g_{ij}]_{k\times n}$ is a $k \times n$ generator matrix. The matrix $\mathbf{G}$ has full row rank k, and the code $\mathscr{C}$ is generated by taking all linear combinations of the rows of the matrix $\mathbf{G}$, where field operations are performed using modulo-2 arithmetic. The code rate is $R_c = k/n$ and there are 2^k codewords. The task of designing block codes reduces to one of finding generator matrices that produce codes that are both powerful and easy to decode.

For any block code with generator matrix $\mathbf{G}$, there exists an $(n-k) \times n$ parity check matrix $\mathbf{H} = [h_{ij}]_{(n-k)\times n}$ such that $\mathbf{GH}^{\mathrm{T}} = \mathbf{0}_{k\times(n-k)}$. The matrix $\mathbf{H}$ has full row rank $n-k$ and is orthogonal to all codewords, that is, $\mathbf{cH}^{\mathrm{T}} = \mathbf{0}_{n-k}$. The matrix $\mathbf{H}$ is the generator matrix of a dual code $\mathscr{C}^{\mathrm{T}}$, consisting of 2^{n-k} codewords. The parity check matrix of $\mathscr{C}^{\mathrm{T}}$ is the matrix $\mathbf{G}$.

A systematic block code is one having a generator matrix of the form

$$\mathbf{G} = [\mathbf{I}_{k\times k} | \mathbf{P}], \tag{8.1}$$

where $\mathbf{P}$ is a $k \times (n-k)$ matrix. For a systematic block code, the first k coordinates of each codeword are equal to the k-bit input vector $\mathbf{a}$, while the last $n-k$ coordinates are the parity check bits. Using elementary row operations, the generator matrix of any linear block code can be put into systematic form. A systematic block code has the parity check matrix

$$\mathbf{H} = [-\mathbf{P}^{\mathrm{T}} | \mathbf{I}_{(n-k)\times(n-k)}]. \tag{8.2}$$

The parity check matrix in (8.2) is a general form that applies to both binary and nonbinary systematic block codes. For binary codes, the negative sign in front of the $\mathbf{P}^{\mathrm{T}}$ matrix is not necessary. For a binary systematic block code, $\mathbf{GH}^{\mathrm{T}} = \mathbf{I}_{k\times k}\mathbf{P} \oplus \mathbf{PI}_{(n-k)\times(n-k)} = \mathbf{P} \oplus \mathbf{P} = \mathbf{0}_{k\times(n-k)}$, where $\oplus$ indicates modulo-2 addition.

Example 8.1:

The $(n-k) \times n$ parity check matrix of an (n,k) Hamming code is constructed by listing as columns all nonzero binary $(n-k)$-tuples. There are $n = 2^{n-k} - 1$ such nonzero $(n-k)$-tuples. For example, a systematic (7,4) Hamming code has the parity check matrix

$$\mathbf{H} = \left[\begin{array}{cccc:ccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right], \tag{8.3}$$

where the columns of $\mathbf{H}$ consists of all nonzero binary three-tuples. Note that the last three columns form a 3×3 identity matrix since the code is in systematic form, while the first four columns that constitute the $\mathbf{P}^{\mathrm{T}}$ matrix may be placed in any random order with no effect on the performance of the code. Such codes are said to be equivalent.

The generator matrix of this particular (7,4) systematic Hamming code is

$$\mathbf{G} = \left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \tag{8.4}$$

and it can be verified that $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}_{k\times(n-k)}$. The 16 codewords of the $(7,4)$ Hamming code are generated by taking all linear combinations of the rows of $\mathbf{G}$ using modulo-2 arithmetic.

8.1.1.1 Minimum Distance

Let $d(\mathbf{c}_1, \mathbf{c}_2)$ denote the Hamming distance between the codewords $\mathbf{c}_1$ and $\mathbf{c}_2$, equal to the number of coordinates in which they differ. For linear block codes $d(\mathbf{c}_1, \mathbf{c}_2) = w(\mathbf{c}_1 \oplus \mathbf{c}_2)$, where $w(\mathbf{c}_1 \oplus \mathbf{c}_2)$ is the weight of $\mathbf{c}_1 \oplus \mathbf{c}_2$, equal to the number of nonzero coordinates of $\mathbf{c}_1 \oplus \mathbf{c}_2$. The free Hamming distance, d_{free}, of a linear block code is the minimum number of coordinates in which any two codewords differ. For a linear code, the sum of any two codewords $\mathbf{c}_1 \oplus \mathbf{c}_2$ is another codeword $\mathbf{c}$, and the all zeroes vector is a codeword. Hence, the free Hamming distance is

$$d_{\text{free}} = \min_{\mathbf{c}_1, \mathbf{c}_2} d(\mathbf{c}_1, \mathbf{c}_2) \tag{8.5}$$

$$= \min_{\mathbf{c} \neq \mathbf{0}} d(\mathbf{c}, \mathbf{0}) \tag{8.6}$$

$$= \min_{\mathbf{c} \neq \mathbf{0}} w(\mathbf{c}). \tag{8.7}$$

Therefore, d_{free} is equal to the weight of the minimum weight nonzero codeword.

To derive an upper bound on d_{free}, recall that the generator matrix of any linear block code can be put into systematic form $\mathbf{G} = [\mathbf{I}_{k\times k}|\mathbf{P}]$, where $\mathbf{P}$ is a $k \times (n-k)$ matrix. It is certainly the case that the number of nonzero elements in any row of $\mathbf{P}$ cannot exceed $n-k$. Hence, the number of nonzero elements in any row of $\mathbf{G}$

cannot exceed $n-k+1$. Since the rows of the generator matrix $\mathbf{G}$ are themselves valid codewords, it must be true that

$$d_{\text{free}} \leq n-k+1 \tag{8.8}$$

a result known as the Singleton bound. A code that has $d_{\text{free}} = n-k+1$ is called a maximum distance separable (MDS) code. Intuitively, this means that all codewords are as far apart from each other as possible in Hamming distance, and such codes should perform well.

An example of a simple block code that meets the Singleton bound is the binary repetition code

$$\begin{aligned} 0 &\longrightarrow \mathbf{c}_0 = (0,0,\ldots,0)_n, \\ 1 &\longrightarrow \mathbf{c}_1 = (1,1,\ldots,1)_n. \end{aligned}$$

In this case, $d_{\text{free}} = d(\mathbf{c}_0, \mathbf{c}_1) = n-k+1$. The repetition code happens to be the only binary MDS code, and no other binary MDS codes exist. The well-known Reed–Solomon codes are good examples of nonbinary MDS codes.

8.1.1.2 Syndromes

Suppose that the codeword $\mathbf{c}$ is transmitted and the vector $\mathbf{y} = \mathbf{c} \oplus \mathbf{e}$ is received, where $\mathbf{e}$ is defined as the error vector. The syndrome of the received vector $\mathbf{y}$ is defined as the length $n-k$ vector

$$\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{T}}. \tag{8.9}$$

If $\mathbf{s} = \mathbf{0}$, then $\mathbf{y}$ is a codeword; conversely if $\mathbf{s} \neq \mathbf{0}$, then an error must have occurred. Note that if $\mathbf{y}$ is a codeword, then $\mathbf{s} = \mathbf{0}$. Hence, $\mathbf{s} = \mathbf{0}$ does not mean that no errors have occurred. They are just undetectable. Since the sum of any two codewords is another codeword for a linear code, it follows that the number of undetectable error vectors is equal to $2^k - 1$, the number of nonzero codewords. The syndrome only depends upon the error vector because

$$\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{T}} = \mathbf{c}\mathbf{H}^{\mathrm{T}} \oplus \mathbf{e}\mathbf{H}^{T} = \mathbf{0} \oplus \mathbf{e}\mathbf{H}^{\mathrm{T}} = \mathbf{e}\mathbf{H}^{\mathrm{T}}. \tag{8.10}$$

In general, $\mathbf{s} = \mathbf{e}\mathbf{H}^{\mathrm{T}}$ is a system of $n-k$ equations in n variables. Hence, for any given syndrome $\mathbf{s}$, there are 2^k possible solutions having the same error vector $\mathbf{e}$. However, the most likely error pattern $\mathbf{e}$ is the one that has minimum Hamming weight.

8.1.1.3 Error Detection

A linear block code can detect all error patterns of $d_{\text{free}} - 1$ or fewer errors. If $\mathbf{e} \neq \mathbf{0}$ is a codeword, then no errors are detected. There are $2^k - 1$ undetectable error patterns, but there are $2^n - 1$ possible nonzero error patterns. Hence, the number of detectable error patterns is

$$2^n - 1 - (2^k - 1) = 2^n - 2^k.$$

Usually, $2^k - 1$ is a small fraction of $2^n - 2^k$. For the (7,4) Hamming code considered in Example 8.1, there are $2^4 - 1 = 15$ undetectable error patterns and $2^7 - 2^4 = 112$ detectable error patterns.

8.1.1.4 Weight Distribution

Consider a block code $\mathscr{C}$ and let A_i be the number of codewords of weight i. The set $\{A_0, A_1, \ldots, A_n\}$ is called the weight distribution of $\mathscr{C}$. The weight distribution can be expressed as a weight enumerator polynomial

$$A(z) = A_0 z^0 + A_1 z^1 + \ldots + A_n z^n. \tag{8.11}$$

For the (7,4) Hamming code in Example 8.1,

$$A_0 = 1, A_2 = 0, A_3 = 7, A_4 = 7, A_5 = 0, A_6 = 0, A_7 = 1.$$

Hence,

$$A(z) = 1 + 7z^3 + 7z^4 + z^7.$$

The weight enumerator polynomial can be used to evaluate the exact performance of a code. Unfortunately, weight enumerator polynomials are generally difficult to find and are known only for a few classes of codes such as the Hamming codes.

8.1.1.5 Probability of Undetected Error

The probability of undetected error is

$$\begin{aligned} \mathrm{P_e}[u] &= \mathrm{P}[\mathbf{e} \text{ is a nonzero codeword}] \\ &= \sum_{i=1}^{n} A_i \mathrm{P}[w(\mathbf{e}) = i]. \end{aligned} \tag{8.12}$$

The error probability $\mathrm{P}[w(\mathbf{e}) = i]$ depends on the coding channel, defined as that portion of the communication system that is seen by the coding system. The simplest coding channel is the binary symmetric channel (BSC), where

$$\mathrm{P}[y_i \neq c_i] = p = 1 - \mathrm{P}[y_i = c_i]. \tag{8.13}$$

For a BSC, $\mathrm{P}[w(\mathbf{e}) = i] = p^i(1-p)^{n-i}$ and, hence,

$$\mathrm{P_e}[u] = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i}. \tag{8.14}$$

The (7,4) Hamming code in Example 8.1 has an undetected error probability of

$$\mathrm{P_e}[u] = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7. \tag{8.15}$$

For a raw channel error rate of $p = 10^{-2}$, we have $\mathrm{P_e}[u] = 7 \times 10^{-6}$. Hence, the undetected error rate can be very small even for a fairly simple block code.

8.1.1.6 Error Correction

A linear block code can correct all error patterns of t or fewer errors, where

$$t \leq \lfloor \frac{d_{\text{free}} - 1}{2} \rfloor \tag{8.16}$$

and $\lfloor x \rfloor$ is the largest integer contained in x. A code is usually capable of correcting many error patterns of $t+1$ or more errors. In fact, up to 2^{n-k} error patterns may be corrected, which is equal to the number of syndromes.

For a BSC, the probability of codeword error is

$$\begin{aligned} \mathrm{P}[e] &\leq 1 - \mathrm{P}[t \text{ or fewer errors}] \\ &= 1 - \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}. \end{aligned} \tag{8.17}$$

8.1.1.7 Standard Array Decoding

One conceptually simple method for decoding linear block codes is standard array decoding. The standard array of an (n,k) linear block code is constructed as follows:

1. Write out all 2^k codewords in a row starting with $\mathbf{c}_0 = \mathbf{0}$.
2. From the remaining $2^n - 2^k$ n-tuples, select an error vector $\mathbf{e}_2$ of weight 1 and place it under $\mathbf{c}_0$. Under each codeword put $\mathbf{c}_i \oplus \mathbf{e}_2, i = 1, \ldots, 2^k - 1$.
3. Select a minimum weight error vector $\mathbf{e}_3$ from the remaining unused n-tuples and place it under $\mathbf{c}_0 = \mathbf{0}$. Under each codeword put $\mathbf{c}_i \oplus \mathbf{e}_3, i = 1, \ldots, 2^k - 1$.
4. Repeat Step 3 until all n-tuples have been used.

Note that every n-tuple appears once and only once in the standard array.

Example 8.2:
Consider the $(4,2)$ code with generator matrix

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

The standard array is

$$\begin{bmatrix} \mathbf{e}_1\ 0000\ 1100\ 0101\ 1001 \\ \mathbf{e}_2\ 0001\ 1101\ 0100\ 1000 \\ \mathbf{e}_3\ 0010\ 1110\ 0111\ 1011 \\ \mathbf{e}_4\ 0011\ 1111\ 0110\ 1010 \end{bmatrix}.$$

The standard array consists of 2^{n-k} disjoint rows of 2^k elements. These rows are called cosets and the ith row has the elements

$$F_i = \{\mathbf{e}_i, \mathbf{e}_i \oplus \mathbf{c}_1, \ldots, \mathbf{e}_i \oplus \mathbf{c}_{2^k-1}\}.$$

The first element, $\mathbf{e}_i$, is called the coset leader. The standard array also consists of 2^k disjoint columns. The jth column has the elements

$$D_j = \{\mathbf{c}_j, \mathbf{c}_j \oplus \mathbf{e}_2, \ldots, \mathbf{c}_j \oplus \mathbf{e}_{2^{n-k}}\}.$$

To correct errors, the following procedure is used. When $\mathbf{y}$ is received, find $\mathbf{y}$ in the standard array. If $\mathbf{y}$ is in row i and column j, then the coset leader from row i, $\mathbf{e}_i$, is the most likely error pattern to have occurred and $\mathbf{y}$ is decoded into $\mathbf{y} \oplus \mathbf{e}_i = \mathbf{c}_j$. A code is capable of correcting all error patterns that are coset leaders. If the error pattern is not a coset leader, then erroneous decoding will result. Obviously, standard array decoding is only useful for simple codes, since 2^n vectors must be stored in memory. A somewhat simpler decoding strategy is syndrome decoding.

8.1.1.8 Syndrome Decoding

Syndrome decoding relies on the fact that all 2^k n-tuples in the same coset of the standard array have the same syndrome. This is because the syndrome only depends on the coset leader as shown in (8.10). To perform syndrome decoding:

1. Compute the syndrome $\mathbf{s} = \mathbf{y}\mathbf{H}^{\mathrm{T}}$
2. Locate the coset leader $\mathbf{e}_\ell$ where $\mathbf{e}_\ell \mathbf{H}^{\mathrm{T}} = \mathbf{s}$
3. Decode $\mathbf{y}$ into $\mathbf{y} \oplus \mathbf{e}_\ell = \hat{\mathbf{c}}$

This technique can be used for any linear block code. The calculation in Step 2 can be done using a simple look-up table. However, for large $n-k$ it becomes impractical because 2^{n-k} syndromes and 2^{n-k} error patterns must be stored.

8.1.2 Space-Time Block Codes

Space-time block coding is a technique for coding across multiple antennas. In Sect. 6.10, we discussed a simple transmit diversity scheme by Alamouti [11], which uses two transmit antennas in a $2 \times L_{\mathrm{r}}$ arrangement, where L_{r} is the number of receiver antennas. The Alamouti scheme achieves a diversity order of $2L_{\mathrm{r}}$ and has a very simple maximum likelihood decoding algorithm. It can be thought of as a very simple 2×2 space-time block code having the code matrix

$$S = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(1)} \end{bmatrix}, \tag{8.18}$$

where the rows represent time slots and the columns represent the transmissions from the antennas over time. For the Alamouti code, two symbols are transmitted from two antennas over two time slots. The basic idea of the Alamouti code was later generalized by Tarokh, Jafarkhani, and Calderbank to an arbitrary number, L_{t}, of transmit antennas by applying orthogonal designs [249]. While a variety of space-time block codes exist, here we concentrate on orthogonal space-time block codes. Orthogonal space-time block codes achieve a diversity order $L_{\mathrm{t}}L_{\mathrm{r}}$, while allowing for maximum likelihood based on computationally simple linear processing [250].

Let L_{t} be the number of transmit antennas and p represent the number of time slots that are used to transmit one space-time codeword. This gives rise to a $p \times L_{\mathrm{t}}$ space-time code matrix. Each group of k information symbols, chosen from an $M = 2^m$-ary alphabet, is encoded according to the space-time code matrix to generate a $p \times L_{\mathrm{t}}$ space-time codeword. Since the codewords are transmitted simultaneously from L_{t} transmit antennas in p time slots, the space-time code rate is equal to $R = k/p$. Similar to the Alamouti code, the entries of the $p \times L_{\mathrm{t}}$ code matrix are chosen to be a combination of the block of k modulating symbols $\{\tilde{\mathbf{s}}_{(1)}, \ldots, \tilde{\mathbf{s}}_{(\mathrm{k})}\}$ and their complex conjugates $\{\tilde{\mathbf{s}}^*_{(1)}, \ldots, \tilde{\mathbf{s}}^*_{(\mathrm{k})}\}$. For orthogonal space-time codes, the $p \times L_{\mathrm{t}}$ code matrix S satisfies the following orthogonal property [249]:

$$S^{\mathrm{H}} S = I_{L_{\mathrm{t}} \times L_{\mathrm{t}}} \sum_{i=1}^{k} |\tilde{\mathbf{s}}_{(\mathrm{i})}|^2. \tag{8.19}$$

Depending on the type of signal constellation from which the symbols $\tilde{\mathbf{s}}_{(\mathrm{i})}$ are drawn, either real or complex valued orthogonal space-time block codes can be constructed.

8.1.2.1 Real Orthogonal Space-Time Block Codes

A real orthogonal design is a $p \times L_t$ orthogonal matrix containing real-valued elements satisfying the orthogonal property in (8.19). First consider the case where S is a $L_t \times L_t$ square matrix. It is known that real orthogonal designs having square code matrices only exist for $L_t = 2, 4$ or 8 transmit antennas. Examples of real orthogonal space-time block codes are the 2×2 design

$$S_2 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} \end{bmatrix}, \tag{8.20}$$

the 4×4 design

$$S_4 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} \end{bmatrix}, \tag{8.21}$$

and the 8×8 design

$$S_8 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(5)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(7)} \\ -\tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} \\ -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(5)} \\ -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} \end{bmatrix}. \tag{8.22}$$

The reader can verify that the columns of these matrices are mutually orthogonal, that is, their dot product is zero. Hence, the orthogonal property in (8.19) is satisfied. Also, the rate achieved with these code matrices is $R = 1$. For example, with the 8×8 code matrix, $k = 8$ symbols are transmitted in $p = 8$ time slots so that $R = k/p = 1$. Finally, these codes all achieve full transmit diversity of order L_t.

The above orthogonal space-time code designs are based on $L_t \times L_t$ square matrices. Tarokh et al. [249] have developed generalized real orthogonal designs for any number of transmit antennas that achieve rate $R = 1$. They have shown that the minimum number of transmission periods p to achieve full rate is given by [249]

$$p = \min \left\{ 2^{4c+d} \right\}, \tag{8.23}$$

where the minimization is taken over the set

$$\left\{ c, d \mid 0 \leq c, 0 \leq d < 4 \text{ and } 8c + 2^d \geq L_t \right\}. \tag{8.24}$$

Based on (8.23) and (8.24), real orthogonal designs for $L_t = 3, 5, 6$ and 7 transmit antennas can be constructed that have full transmit diversity of order L_t and rate $R = 1$ as follows:

$$S_3 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} \\ -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} \end{bmatrix}, \tag{8.25}$$

$$S_5 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(5)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(6)} \\ -\tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(7)} \\ -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(8)} \\ -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(1)} \\ -\tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(4)} \end{bmatrix}. \tag{8.26}$$

$$S_6 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(5)} & \tilde{\mathbf{s}}_{(6)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(5)} \\ -\tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} \\ -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} \\ -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} \\ -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} \end{bmatrix} \tag{8.27}$$

and

$$S_7 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(5)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(7)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} \\ -\tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(5)} \\ -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} \\ -\tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(7)} & -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(8)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(6)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} \\ -\tilde{\mathbf{s}}_{(8)} & -\tilde{\mathbf{s}}_{(7)} & \tilde{\mathbf{s}}_{(6)} & \tilde{\mathbf{s}}_{(5)} & -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} \end{bmatrix}. \tag{8.28}$$

Taking S_7 as an example, we see that $k = 8$ symbols are transmitted from $L_t = 7$ antennas in $p = 8$ time slots. Hence, S_7 has rate $R = k/p = 1$. Since the columns are mutually orthogonal the code achieves full transmit diversity of order $L_t = 7$.

8.1.2.2 Complex Orthogonal Space-Time Block Codes

Complex orthogonal space-time codes have $p \times L_t$ code matrices containing the elements $\{\tilde{\mathbf{s}}_{(1)}, \ldots, \tilde{\mathbf{s}}_{(k)}\}$ and their complex conjugates $\{\tilde{\mathbf{s}}^*_{(1)}, \ldots, \tilde{\mathbf{s}}^*_{(k)}\}$, and they satisfy the orthogonal property in (8.19). Such codes provide full transmit diversity of order L_t and have rate $R = k/p$. The Alamouti space-time code matrix in (8.18) is one such 2×2 scheme that achieves a transmit diversity of order 2 with a full code rate $R = 1$. The Alamouti scheme is unique in which it is the only complex orthogonal space-time block code having full transmit diversity and rate $R = 1$. Tarkoh et al. [249] have constructed complex orthogonal space-time block codes with rate $R = 1/2$ for $L_t = 3$ and 4 transmit antennas as follows:

$$C_3 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} \\ -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} \\ \tilde{\mathbf{s}}^*_{(1)} & \tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(3)} \\ -\tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(1)} & -\tilde{\mathbf{s}}^*_{(4)} \\ -\tilde{\mathbf{s}}^*_{(3)} & \tilde{\mathbf{s}}^*_{(4)} & \tilde{\mathbf{s}}^*_{(1)} \\ -\tilde{\mathbf{s}}^*_{(4)} & -\tilde{\mathbf{s}}^*_{(3)} & \tilde{\mathbf{s}}^*_{(2)} \end{bmatrix}, \tag{8.29}$$

and

$$C_4 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(3)} \\ -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(4)} & \tilde{\mathbf{s}}_{(1)} & -\tilde{\mathbf{s}}_{(2)} \\ -\tilde{\mathbf{s}}_{(4)} & -\tilde{\mathbf{s}}_{(3)} & \tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} \\ \tilde{\mathbf{s}}^*_{(1)} & \tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(3)} & \tilde{\mathbf{s}}^*_{(4)} \\ -\tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(1)} & -\tilde{\mathbf{s}}^*_{(4)} & \tilde{\mathbf{s}}^*_{(3)} \\ -\tilde{\mathbf{s}}^*_{(3)} & \tilde{\mathbf{s}}^*_{(4)} & \tilde{\mathbf{s}}^*_{(1)} & -\tilde{\mathbf{s}}^*_{(2)} \\ -\tilde{\mathbf{s}}^*_{(4)} & -\tilde{\mathbf{s}}^*_{(3)} & \tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(1)} \end{bmatrix}. \tag{8.30}$$

The reader can verify that the columns of these code matrices are all mutually orthogonal. Thus, these schemes achieve full transmit diversity of order L_t, but lose half of the theoretical bandwidth efficiency since they are only $R = 1/2$ codes.

By allowing linear processing at the transmitter, it is possible to construct higher rate complex space-time block codes. Tarokh et al. [249] have identified the following complex orthogonal designs: the $L_t = 3$ rate $R = 3/4$ code

$$H_3 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} \\ -\tilde{\mathbf{s}}^*_{(2)} & \tilde{\mathbf{s}}^*_{(1)} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} \\ \frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & \frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & \frac{(-\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)}+\tilde{\mathbf{s}}_{(2)}-\tilde{\mathbf{s}}^*_{(2)})}{2} \\ \frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & -\frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & \frac{(\tilde{\mathbf{s}}_{(2)}-\tilde{\mathbf{s}}^*_{(2)}+\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)})}{2} \end{bmatrix} \tag{8.31}$$

and the $L_t = 4$ rate $R = 3/4$ code

$$H_4 = \begin{bmatrix} \tilde{\mathbf{s}}_{(1)} & \tilde{\mathbf{s}}_{(2)} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} \\ -\tilde{\mathbf{s}}_{(2)} & \tilde{\mathbf{s}}_{(1)} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} & -\frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} \\ \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} & \frac{\tilde{\mathbf{s}}_{(3)}}{\sqrt{2}} & \frac{(-\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)}+\tilde{\mathbf{s}}_{(2)}-\tilde{\mathbf{s}}^*_{(2)})}{2} & \frac{(-\tilde{\mathbf{s}}_{(2)}+\tilde{\mathbf{s}}^*_{(2)}+\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)})}{2} \\ \frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & -\frac{\tilde{\mathbf{s}}^*_{(3)}}{\sqrt{2}} & \frac{(\tilde{\mathbf{s}}_{(2)}+\tilde{\mathbf{s}}^*_{(2)}+\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)})}{2} & \frac{(-\tilde{\mathbf{s}}_{(1)}-\tilde{\mathbf{s}}^*_{(1)}-\tilde{\mathbf{s}}_{(2)}+\tilde{\mathbf{s}}^*_{(2)})}{2} \end{bmatrix}. \tag{8.32}$$

8.1.2.3 Decoding Orthogonal Space-Time Block Codes

The Alamouti space-time block code was shown to have an efficient and simple maximum likelihood decoding method in Sect. 6.10. We now discuss the decoding of orthogonal $p \times L_t$ space-time block codes. Following the notation in Sect. 6.10, the received signal vector at antenna j and time t is given by

$$\tilde{\mathbf{r}}_{(t),j} = \sum_{i=1}^{L_t} g_{i,j}\tilde{\mathbf{s}}_{(t),i} + \tilde{\mathbf{n}}_{(t),j}, \tag{8.33}$$

where $g_{i,j}$ is the complex channel gain from transmit antenna i to receiver antenna j, $\tilde{\mathbf{s}}_{(t),i}$ is the symbol vector transmitted from antenna i in time slot t, and the $\tilde{\mathbf{n}}_{(t),j}$ are independent zero-mean complex Gaussian random vectors with variance N_o in each dimension of the signal space. Assuming perfect channel state information, the maximum likelihood receiver computes the decision metric

$$\mu(C) = \sum_{t=1}^{p}\sum_{j=1}^{L_r}\left\|\tilde{\mathbf{r}}_{(t),j} - \sum_{i=1}^{L_t} g_{i,j}\tilde{\mathbf{s}}_{(t),i}\right\|^2 \tag{8.34}$$

over all codewords $C = [\tilde{\mathbf{s}}_{(t),i}]_{p\times L_t}$ and chooses the codeword with the minimum metric.

First consider the real orthogonal space-time block codes with square matrices S_2, S_4 and S_8. Note that the rows of code matrices S_2, S_4, and S_8 are all permutations of the first row, possibly with different signs. Let $\varepsilon_1, \ldots, \varepsilon_{L_t}$ denote the permutations corresponding to these rows, and let $\delta_t^{(i)}$ denote the sign of the entry in row t and

column i of the code matrix. Then $\varepsilon_t^{(i)} = q$ means that $\tilde{\mathbf{s}}_{(t),i}$ is up to a sign change equal to the element in row t and column q of the code matrix. Since the columns of the space-time orthogonal code matrices S_2, S_4, and S_8 are mutually orthogonal, minimizing the metric in (8.34) is equivalent to minimizing [249]

$$\mu(C) = \sum_{i=1}^{L_\mathrm{t}} P_i, \tag{8.35}$$

where

$$P_i = \left(\left\| \left[\sum_{t=1}^{L_\mathrm{t}} \sum_{j=1}^{L_\mathrm{r}} \tilde{\mathbf{r}}_{(t),j} g^*_{\varepsilon_t^{(i)},j} \delta_t^{(i)} \right] - \tilde{\mathbf{s}}_i \right\|^2 + \left(-1 + \sum_{i,j} |g_{i,j}|^2 \right) \|\tilde{\mathbf{s}}_i\|^2 \right). \tag{8.36}$$

Note that $P_i, i = 1, \ldots, L_\mathrm{t}$ only depends on the choice of code symbol $\tilde{\mathbf{s}}_i$, the set of received vectors $\{\tilde{\mathbf{r}}_{(t),j}, j = 1, \ldots, L_\mathrm{r}\}$, the channel fading coefficients $\{g_{i,j}\}$, and the structure of the code matrix. It follows that minimizing the sum in (8.35) is equivalent to minimizing P_i in (8.36) for each $i, 1 \leq i \leq L_\mathrm{t}$. This separable property results in a very simple decoding strategy that provides spatial diversity of order $L_\mathrm{t} L_\mathrm{r}$. The maximum likelihood receiver simply forms the L_t decision variables

$$R_i = \sum_{t=1}^{L_\mathrm{t}} \sum_{j=1}^{L_\mathrm{r}} \tilde{\mathbf{r}}_{(t),j} g^*_{\varepsilon_t^{(i)},j} \delta_t^{(i)}, \; i = 1, \ldots, L_\mathrm{t} \tag{8.37}$$

and decides in favor of symbol $\hat{\mathbf{s}}_i$ if

$$\hat{\mathbf{s}}_i = \arg\min_{\mathbf{s}} \|R_i - \mathbf{s}\|^2 + \left(-1 + \sum_{i,j} |g_{i,j}|^2 \right) \|\tilde{\mathbf{s}}\|^2, \; i = 1, \ldots, L_\mathrm{t}. \tag{8.38}$$

Similar low complexity decoding strategies for the other space-time block codes C_3, C_4, H_3 and H_4 in (8.29), (8.30), (8.31) and (8.32), respectively, are not presented here but are available in [250].

8.2 Convolutional Codes

8.2.1 Encoder Description

The encoder for a rate-$1/n$ binary convolutional code can be viewed as a finite-state machine (FSM) that consists of a ν-stage binary shift register with connections to n modulo-2 adders, and a multiplexer that converts the adder outputs to serial codewords. The constraint length of a convolutional code is defined as the number of shifts through the FSM over which a single input data bit can affect the encoder

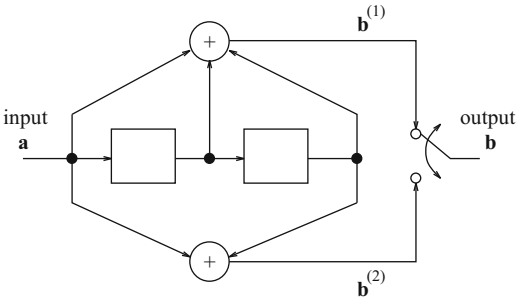


Fig. 8.1 Binary convolutional encoder; $R_c = 1/2$, $K = 3$

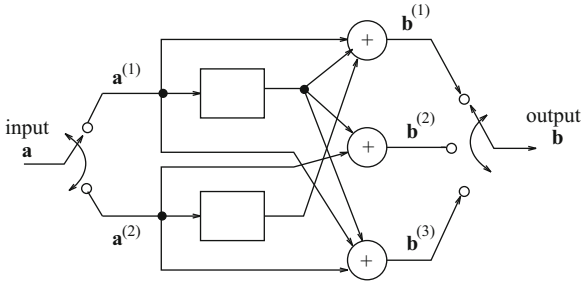


Fig. 8.2 Binary convolutional encoder; $R_c = 2/3$, $K = 2$

output. For an encoder having a single ν-stage shift register, the constraint length is equal to $K = \nu + 1$. A very simple rate-1/2, constraint length $K = 3$, binary convolutional encoder is shown in Fig. 8.1.

The above concept can be generalized to rate-k/n binary convolutional code using k shift registers, n modulo-2 adders, along with input and output multiplexers. For a rate-k/n code, the k-bit information vector $\mathbf{a}_\ell = (a_\ell^{(1)}, \ldots, a_\ell^{(k)})$ is input to the encoder at epoch ℓ to generate the n-bit code vector $\mathbf{b}_\ell = (b_\ell^{(1)}, \ldots, b_\ell^{(n)})$. If K_i denotes the constraint length of the ith shift register, then the overall constraint length is defined as $K = \max_i K_i$. Figure 8.2 shows a simple rate-2/3, constraint length-2 convolutional encoder.

A convolutional encoder can be described by the set of impulse responses, $\{\mathbf{g}_i^{(j)}\}$, where $\mathbf{g}_i^{(j)}$ is the jth output sequence $\mathbf{b}^{(j)}$ that results from the ith input sequence $\mathbf{a}^{(i)} = (1,0,0,0,\ldots)$. The impulse responses can have a duration of at most K and have the form $\mathbf{g}_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \ldots, g_{i,K-1}^{(j)})$. Sometimes the $\{\mathbf{g}_i^{(j)}\}$ are called generator sequences. For the encoder in Fig. 8.1

$$\mathbf{g}^{(1)} = (1,\ 1,\ 1) \qquad \mathbf{g}^{(2)} = (1,\ 0,\ 1) \tag{8.39}$$

and for the encoder in Fig. 8.2

$$\begin{aligned} &\mathbf{g}_1^{(1)} = (1,\ 1),\ \ \mathbf{g}_1^{(2)} = (0,\ 1),\ \ \mathbf{g}_1^{(3)} = (1,\ 1), \\ &\mathbf{g}_2^{(1)} = (0,\ 1),\ \ \mathbf{g}_2^{(2)} = (1,\ 0),\ \ \mathbf{g}_2^{(3)} = (1,\ 0). \end{aligned} \tag{8.40}$$

It follows that the jth output $\mathbf{b}_i^{(j)}$ corresponding to the ith input sequence $\mathbf{a}^{(i)}$ is the discrete convolution $\mathbf{b}_i^{(j)} = \mathbf{a}^{(i)} \circledast \mathbf{g}_i^{(j)}$, where $\circledast$ denotes modulo-2 convolution. The time-domain convolutions can be conveniently replaced by polynomial multiplications in a D-transform domain according to

$$\mathbf{b}_i^{(j)}(D) = \mathbf{a}^{(i)}(D)\mathbf{g}_i^{(j)}(D), \tag{8.41}$$

where

$$\mathbf{a}^{(i)}(D) = \sum_{k=0}^{\infty} a_{i,k} D^k \tag{8.42}$$

is the ith input data polynomial,

$$\mathbf{b}_i^{(j)}(D) = \sum_{k=0}^{\infty} b_{i,k}^{(j)} D^k \tag{8.43}$$

is the jth output polynomial corresponding to the ith input, and

$$\mathbf{g}_i^{(j)}(D) = \sum_{k=0}^{K-1} g_{i,k}^{(j)} D^k \tag{8.44}$$

is the associated generator polynomial. It follows that the jth output sequence is

$$\mathbf{b}^{(j)}(D) = \sum_{i=1}^{k} \mathbf{b}_i^{(j)}(D) = \sum_{i=1}^{k} \mathbf{a}^{(i)}(D)\mathbf{g}_i^{(j)}(D). \tag{8.45}$$

The above expression leads to the matrix form

$$\left(\mathbf{b}^{(i)}(D), \ldots, \mathbf{b}^{(n)}(D)\right) = \left(\mathbf{a}^{(1)}(D), \ldots, \mathbf{a}^{(k)}(D)\right) \mathbf{G}(D), \tag{8.46}$$

where

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_1^{(1)}(D), \ldots, \mathbf{g}_1^{(n)}(D) \\ \vdots \qquad\qquad \vdots \\ \mathbf{g}_k^{(1)}(D), \ldots, \mathbf{g}_k^{(n)}(D) \end{bmatrix} \tag{8.47}$$

is the generator matrix of the code. For the encoder in Fig. 8.1

$$\mathbf{G}(D) = \left[1 + D + D^2 \;\; 1 + D^2\right], \tag{8.48}$$

while for the encoder in Fig. 8.2

$$\mathbf{G}(D) = \begin{bmatrix} 1+D & D & 1+D \\ D & 1 & 1 \end{bmatrix}. \tag{8.49}$$

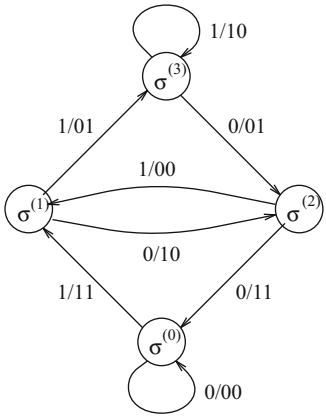


Fig. 8.3 State diagram for the binary convolutional encoder in Fig. 8.1

After multiplexing the outputs, the final codeword has the polynomial representation

$$\mathbf{b}(D) = \sum_{j=1}^{n} D^{j-1}\mathbf{b}^{(j)}(D^n). \tag{8.50}$$

Systematic convolutional codes are those where first k of the n encoder output sequences, $\mathbf{b}^{(1)},\ldots,\mathbf{b}^{(k)}$ are equal to the k encoder input sequences $\mathbf{a}^{(1)},\ldots,\mathbf{a}^{(k)}$.

8.2.2 State and Trellis Diagrams, and Weight Distribution

Since the convolutional encoder is an FSM, its operation can be described by a state diagram and trellis diagram in a manner very similar to the state- and trellis-diagram descriptions of the discrete-time white noise channel model in Chap. 7. The state of the encoder is defined by the shift register contents. For a rate-k/n code, the ith shift register contains ν_i previous information bits. The state of the encoder at epoch ℓ is defined as

$$\sigma_\ell = \left(a_{\ell-1}^{(1)},\ldots,a_{\ell-\nu_1}^{(1)};\ldots;a_{\ell-1}^{(k)},\ldots,a_{\ell-\nu_m}^{(k)}\right). \tag{8.51}$$

There are a total of $N_S = 2^{\nu_T}$ encoder states, where $\nu_T \stackrel{\triangle}{=} \sum_{i=1}^{k}\nu_i$ is defined as the total encoder memory. For a rate-$1/n$ code, the encoder state at epoch ℓ is simply $\sigma_\ell = (a_{\ell-1},\ldots,a_{\ell-\nu})$.

Figures 8.3 and 8.4 show the state diagrams for codes in Figs. 8.1 and 8.2, respectively. The states are labeled using the convention $\sigma^{(i)}, i = 0,\ldots,\nu_T - 1$, where $\sigma^{(i)}$ represents the encoder state $(c_0,\ldots,c_{\nu_T-1})$ corresponding to the integer $i = \sum_{j=0}^{\nu_T-1} c_j 2^j$. In general, for a rate-k/n code there are 2^k branches entering and leaving each state. The branches in the state diagram are labeled with the convention $\mathbf{a}/\mathbf{b} = (a^{(1)},a^{(2)},\ldots,a^{(k)})/(b^{(1)},b^{(2)},\ldots,b^{(n)})$. For example, the state

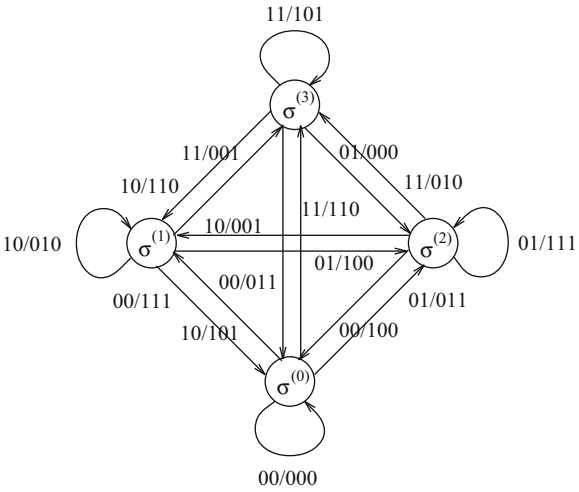


Fig. 8.4 State diagram for the binary convolutional encoder in Fig. 8.2

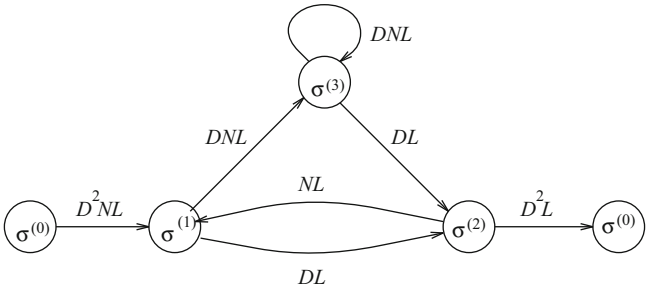


Fig. 8.5 Modified state diagram for the binary convolutional encoder in Fig. 8.1

transition $\sigma^{(1)} \rightarrow \sigma^{(3)}$ in Fig. 8.3 has the label 1/01. This means if the encoder in Fig. 8.1 is in state $\sigma^{(1)} = (01)$ and the input bit is a 1, the encoder will output the code bits 01 and transition to state $\sigma^{(3)} = (11)$.

Convolutional codes are linear codes, meaning that the sum of any two codewords is another codeword and the all-zeroes sequence is a codeword. It follows that the weight distribution and other distance properties of a convolutional code can be obtained from the state diagram. Consider, for example, the encoder in Fig. 8.1 along with its state diagram in Fig. 8.3. Since the self-loop at the zero state $\sigma^{(0)}$ corresponds to the all-zeroes codeword, we can split the zero state $\sigma^{(0)}$ into two nodes, representing the input and output of the state diagram. This leads to the modified state diagram shown in Fig. 8.5. The branches in the modified state diagram have labels of the form $D^i N^j L$, where i is the number of 1's in the encoder output sequence corresponding to a particular state transition, and j is the number of input 1's into the encoder for that transition. Every branch is labeled with the letter L, and the exponent of L is unity because each branch has length one. Each possible path through the modified state diagram corresponds to a non-all-zeroes codeword.

The weight distribution of a convolutional code can be obtained by computing the transfer function $T(D,N,L)$ of the modified state diagram. Any appropriate technique can be used to obtain the transfer function, including flow-graph reduction techniques and Mason's formula [172]. For the example shown in Fig. 8.5, the transfer function is

$$\begin{aligned} T(D,N,L) &= \frac{D^5NL^3}{1-DNL(L+1)} \\ &= D^5L^3N + D^6N^2L^4(L+1) + D^7N^3L^5(L+1)^2 \\ &\quad + \ldots + D^{k+5}N^{k+1}L^{k+3}(L+1)^k + \ldots \end{aligned} \tag{8.52}$$

where the second line was obtained using polynomial division. The terms in the second line of (8.52) enumerate the weight distribution and distance properties of the code. For example, consider the term $D^{k+5}N^{k+1}L^{k+3}(L+1)^k$ appearing in the transfer function. Using the binomial expansion, this term can be rewritten as

$$D^{k+5}N^{k+1}\sum_{n=0}^{k}\binom{k}{n}L^{n+k+3}.$$

Hence, there are 2^k paths through the modified state diagram that are at Hamming distance $k+5$ from the all-zeroes path, that is, have $k+5$ output 1's, that are caused by $k+1$ input 1's. Of these 2^k paths, $\binom{k}{n}$ have length $k+n+3$ branches.

Sometimes the transfer function can be simplified if we are only interested in extracting certain distance properties of the convolutional code. For example, the output weight distribution of the code can be obtained by setting $N=1$ and $L=1$ in the transfer function. For the particular transfer function in (8.52), this leads to

$$\begin{aligned} T(D) &= \frac{D^5}{1-2D} \\ &= D^5 + 2D^6 + 4D^7 + \ldots + 2^kD^{5+k} + \ldots, \end{aligned} \tag{8.53}$$

meaning that there are 2^k codewords at Hamming distance $5+k$ from the all-zeroes codeword. Notice that no nonzero codeword exists with a Hamming distance less than 5 from the all-zeroes codeword. This means that the free Hamming distance of the code is $d_{\text{free}} = 5$. The free Hamming distance for this simple example can also be seen by inspecting the trellis diagram in Fig. 8.6, where the branches in the trellis diagram are labeled with the encoder output bits that correspond to the various state transitions.

Convolutional codes are designed to have the largest possible d_{free} for a given code rate and total encoder memory. Tabulation of convolutional codes that are optimal in this sense can be found in many references, for example, Proakis [217], Lin and Costello [159], and Clark and Cain [59].

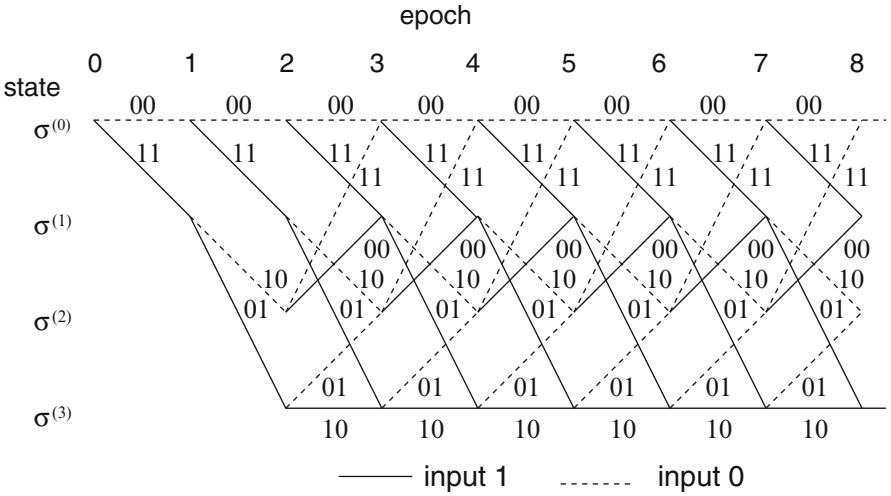


Fig. 8.6 Trellis diagram for the binary convolutional encoder in Fig. 8.1

8.2.3 Recursive Systematic Convolutional Codes

Forney [102] and Costello [67] showed that it is possible to construct a recursive systematic convolutional (RSC) encoder from every rate $R_c = 1/n$ feedforward nonsystematic convolutional encoder, such that the output weight distributions of the codes are identical. Consider a rate-$1/n$ code with generator polynomials $\mathbf{g}_1(D), \ldots, \mathbf{g}_n(D)$. The output sequences are described by the polynomials

$$\mathbf{b}^{(j)}(D) = \mathbf{a}(D)\mathbf{g}^{(j)}(D), \qquad j = 1, \ldots, n. \tag{8.54}$$

To obtain a systematic code, we need to have $\mathbf{b}^{(1)}(D) = \mathbf{a}(D)$. To obtain this, suppose that both sides of (8.54) are divided by $\mathbf{g}^{(1)}(D)$, so that

$$\tilde{\mathbf{b}}^{(1)}(D) = \frac{\mathbf{b}^{(1)}(D)}{\mathbf{g}^{(1)}(D)} = \mathbf{a}(D), \tag{8.55}$$

$$\tilde{\mathbf{b}}^{(j)}(D) = \frac{\mathbf{b}^{(j)}(D)}{\mathbf{g}^{(1)}(D)} = \mathbf{a}(D)\frac{\mathbf{g}^{(j)}(D)}{\mathbf{g}^{(1)}(D)}, \qquad j = 2, \ldots, n. \tag{8.56}$$

Sometimes the $\mathbf{g}^{(j)}(D)$ are called the feedforward polynomials, while $\mathbf{g}^{(1)}(D)$ is called the feedback polynomial. Define a new input sequence $\tilde{\mathbf{a}}(D)$ as

$$\tilde{\mathbf{a}}(D) \triangleq \frac{\mathbf{a}(D)}{\mathbf{g}^{(1)}(D)} \tag{8.57}$$

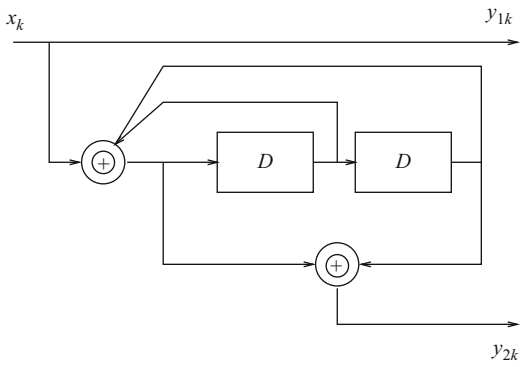


Fig. 8.7 Recursive systematic convolutional (RSC) encoder derived from the feedforward nonsystematic encoder in Fig. 8.1

so that

$$\tilde{\mathbf{b}}^{(1)}(D) = \tilde{\mathbf{a}}(D)\mathbf{g}^{(1)}(D), \tag{8.58}$$

$$\tilde{\mathbf{b}}^{(j)}(D) = \tilde{\mathbf{a}}(D)\mathbf{g}^{(j)}(D)\ , \qquad j = 2,\ldots,n. \tag{8.59}$$

Observe that the transformation between $\mathbf{a}(D)$ and $\tilde{\mathbf{a}}(D)$ in (8.57) is that of a recursive digital filter with modulo-2 operations. This transformation simply reorders the input sequence $\mathbf{a}(D)$. Since the input sequences consist of all possible binary sequences, the filtered sequences $\tilde{\mathbf{a}}(D)$ also consist of all possible binary sequences. Hence, the set of coded sequences $\tilde{\mathbf{b}}(D)$ is the same as the set of coded sequences $\mathbf{b}(D)$, and thus the nonsystematic and systematic codes have the same output weight distribution functions. However, the input weight distributions for the two codes are completely different as we will see.

Example 8.3

Consider, for example, the rate-1/2 encoder in Fig. 8.1 with generators

$$\mathbf{g}^{(1)}(D) = 1 + D + D^2, \tag{8.60}$$

$$\mathbf{g}^{(2)}(D) = 1 + D^2. \tag{8.61}$$

By following the above-described procedure, an RSC code is obtained with generators

$$\hat{\mathbf{g}}^{(1)}(D) = 1,$$

$$\hat{\mathbf{g}}^{(2)}(D) = \frac{\mathbf{g}^{(2)}(D)}{\mathbf{g}^{(1)}(D)} = \frac{1 + D^2}{1 + D + D^2}.$$

The RSC is shown in Fig. 8.7

Similar to their feedforward counterparts, the weight distribution and other distance properties of RSC codes can be obtained by constructing their corresponding modified state diagram and computing the transfer function $T(D,N,L)$. The RSC encoder in Fig. 8.7 has transfer function

$$T(D,N,L) = \frac{D^5N^3L^3 - D^6N^4L^4 + D^6N^2L^4}{1 - DNL - DNL^2 - D^2L^3 + D^2N^2L^3} \tag{8.62}$$

$$= D^5N^3L^3 + D^6N^2L^4 + D^6N^4L^5 + \ldots \tag{8.63}$$

By setting $N = 1$ and $L = 1$, we obtain the output weight distribution of the code, $T(D)$, which is identical to the output weight distribution of the corresponding feedforward nonsystematic encoder in (8.53). However, by comparing the first few terms in their respective transfer functions in (8.52) and (8.63), it can be observed that the input weight distributions are completely different. In particular, codewords can be generated by weight-1 input sequences for the feedforward nonsystematic encoder, while the RSC requires input sequences having at least weight-2 to generate codewords. In fact, any finite weight codeword for the RSC code in Fig 8.7 is generated by an input polynomial $\mathbf{a}(D)$ that is divisible by $1 + D + D^2$. We will see later that these properties are crucial for turbo codes.

Finally, both the feedforward nonsystematic and RSC codes are time invariant. This means that if the input sequence $a(D)$ produces codeword $b(D)$, then the input sequence $D^i a(D)$ produces the codeword $D^i b(D)$. Note that the codewords $b(D)$ and $D^i b(D)$ have the same weight.

8.2.4 Viterbi Algorithm

The Viterbi algorithm was devised by Viterbi for maximum likelihood decoding of convolutional codes [266, 267]. We first note that the convolutional encoder outputs n bits per branch that are subsequently mapped onto a $M = 2^k$-point signal constellation. Here, we assume that there are $\ell = n/k$ modulated symbols per branch where ℓ is an integer. For example, a rate-1/2 code having two code bits per branch may have the two code bits mapped onto either two binary symbols or just one quaternary symbol. To derive the Viterbi algorithm, suppose that a sequence of modulated symbols $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_n\}_{n=1}^k$, $\tilde{\mathbf{s}}_n = (\tilde{\mathbf{s}}_{n,1}, \ldots, \tilde{\mathbf{s}}_{n,\ell})$, corresponding to k branches in the code trellis are transmitted over an interleaved flat fading channel with AWGN.[1] After receiving the sequence $\tilde{\mathbf{r}} = \{\tilde{\mathbf{r}}_n\}_{n=1}^k$, $\tilde{\mathbf{r}}_n = (\tilde{\mathbf{r}}_{n,1}, \ldots, \tilde{\mathbf{r}}_{n,\ell})$, the maximum likelihood receiver uses knowledge of the sequence of complex channel

[1] Here, we are using the complex low-pass vector notation.

gains $\mathbf{g} = \{\mathbf{g}_n\}_{n=1}^{k}$, $\mathbf{g}_n = (g_{n,1}, \ldots, g_{n,\ell})$ (obtained from a separate channel estimator) to decide in favor of the sequence $\tilde{\mathbf{s}}$ that maximizes the likelihood function

$$p(\tilde{\mathbf{r}}_k, \ \ldots, \ \tilde{\mathbf{r}}_1 |\ g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell}, \ \ldots, \ g_{1,1}\tilde{\mathbf{s}}_{1,1}) \tag{8.64}$$

or, equivalently, the log-likelihood function

$$\log\{p(\tilde{\mathbf{r}}_k, \ \ldots, \ \tilde{\mathbf{r}}_1 |\ g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell}, \ \ldots, \ g_{1,1}\tilde{\mathbf{s}}_{1,1})\}. \tag{8.65}$$

The log-likelihood function in (8.65) can be rewritten as

$$\begin{aligned}&\log\{p(\tilde{\mathbf{r}}_k, \ldots, \tilde{\mathbf{r}}_1 | g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell}, \ldots, g_{1,1}\tilde{\mathbf{s}}_{1,1})\} \\ &\quad = \log\{p(\tilde{\mathbf{r}}_k | g_{k,1}\tilde{\mathbf{s}}_{k,1}, \ldots, g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell})\} + \log\{p(\tilde{\mathbf{r}}_{k-1}, \ldots, \tilde{\mathbf{r}}_1 | g_{k-1,\ell}\tilde{\mathbf{s}}_{k-1,\ell}, \ldots, g_{1,1}\tilde{\mathbf{s}}_{1,1})\}.\end{aligned} \tag{8.66}$$

If the second term on the right-hand side of (8.66) has been calculated previously at epoch $k-1$ and stored in memory, then only the first term, called the branch metric, has to be computed for the incoming signal vector $\mathbf{r}_k$ at epoch k. From our treatment in Chap. 5

$$p(\tilde{\mathbf{r}}_k | g_{k,1}\tilde{\mathbf{s}}_{k,1}, \ldots, g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell}) = \frac{1}{(2\pi N_\mathrm{o})^N} \exp\left\{ -\frac{1}{2N_\mathrm{o}} \sum_{m=1}^{\ell} \|\tilde{\mathbf{r}}_{k,m} - g_{k,m}\tilde{\mathbf{s}}_{k,m}\|^2 \right\} \tag{8.67}$$

so that $\log\{p(\tilde{\mathbf{r}}_k | g_{k,1}\tilde{\mathbf{s}}_{k,1}, \ldots, g_{k,\ell}\tilde{\mathbf{s}}_{k,\ell})\}$ yields the Euclidean branch metric

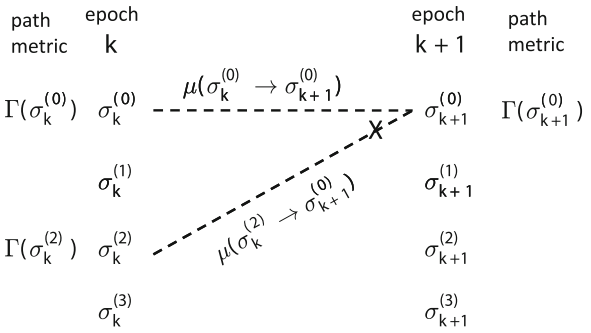
$$\mu_k = -\sum_{m=1}^{\ell} \|\tilde{\mathbf{r}}_{k,m} - g_{k,m}\tilde{\mathbf{s}}_{k,m}\|^2. \tag{8.68}$$

Based on the recursion in (8.66) and the branch metric in (8.68), the Viterbi algorithm searches through the N_S-state code trellis for the most likely transmitted sequence $\tilde{\mathbf{s}}$ given the sequence of received vectors $\tilde{\mathbf{r}}$ and knowledge of the sequence of complex channel gains $\mathbf{g}$. At epoch k, the Viterbi algorithm stores N_S survivors $\check{\mathbf{s}}(\sigma_k^{(i)})$ along with their associated path metrics $\Gamma(\sigma_k^{(i)})$ that terminate at state $\sigma_k^{(i)}$, $i = 0, \ldots, N_\mathrm{S} - 1$. The path metric is defined as

$$\Gamma(\sigma_k^{(i)}) = \sum_{n=1}^{k} \mu_n^{(i)}, \quad i = 0, \ldots, N_\mathrm{S} - 1, \tag{8.69}$$

where $\{\mu_n^{(i)}\}$ is the sequence of branch metrics along the surviving path $\check{\mathbf{s}}(\sigma_k^{(i)})$.

The Viterbi algorithm is initialized at time index $k = 0$, by setting all path metrics to zero, that is, $\Gamma(\sigma_0^{(i)}) = 0$, $i = 1, \ldots, N_\mathrm{S} - 1$.



$$\begin{aligned}\Gamma(\sigma_{k+1}^{(0)}) &= \max[\Gamma(\sigma_k^{(0)}) + \mu(\sigma_k^{(0)} \to \sigma_{k+1}^{(0)}), \Gamma(\sigma_k^{(2)}) + \mu(\sigma_k^{(2)} \to \sigma_{k+1}^{(2)})] \\ &= \Gamma(\sigma_k^{(0)}) + \mu(\sigma_k^{(0)} \to \sigma_{k+1}^{(0)})\end{aligned}$$

Fig. 8.8 Path metric update for the code trellis in Fig. 8.6

1. After the vector $\tilde{\mathbf{r}}_{k+1}$ has been received, compute the set of path metrics $\Gamma(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}) = \Gamma(\sigma_k^{(i)}) + \mu(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)})$ for all possible paths through the trellis that terminate in each state $\sigma_{k+1}^{(j)}, j = 0, \ldots, N_S - 1$, where $\mu(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)})$ is the branch metric defined below. For a modulation alphabet of size M, there will be M such paths that terminate in each state $\sigma_{k+1}^{(j)}$.
2. Find $\Gamma(\sigma_{k+1}^{(j)}) = \max_i \Gamma(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}), j = 0, \ldots, N_S - 1$ where the maximization is over all M possible paths through the trellis that terminate in state $\sigma_{k+1}^{(j)}$.
3. Store $\Gamma(\sigma_{k+1}^{(j)})$ and its associated surviving sequence $\check{\mathbf{s}}(\sigma_{k+1}^{(j)}), j = 0, \ldots, N_S - 1$. Drop all other paths.
4. Increment the time index k, goto Step 1 and repeat the entire algorithm.

In Step 1 above, $\mu(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)})$ is the branch metric associated with the state transition $\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}$ and is computed according to the following variation of (8.68)

$$\mu\left(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}\right) = -\sum_{m=1}^{\ell} \left\| \tilde{\mathbf{r}}_{k,m} - g_{k,m}\tilde{\mathbf{s}}_{k,m}\left(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}\right) \right\|^2, \tag{8.70}$$

where $\tilde{\mathbf{s}}_{k,m}(\sigma_k^{(i)} \to \sigma_{k+1}^{(j)})$ is a symbol that is uniquely determined by the state transition $\sigma_k^{(i)} \to \sigma_{k+1}^{(j)}$ and the symbol mapping being used.

The calculation of the path metric $\Gamma(\sigma_{k+1}^{(0)})$ for state $\sigma^{(0)}$ at epoch $k+1$ is illustrated in Fig. 8.8 for the code trellis shown in Fig. 8.6. In this case, there are two paths merging into state $\sigma^{(0)}$ at epoch $k+1$. The Viterbi algorithm determines the path metric $\Gamma(\sigma_{k+1}^{(0)})$ in this particular example as

$$\Gamma\left(\sigma_0^{(j)}\right) = \max\left\{\Gamma(\sigma_k^{(0)}) + \mu(\sigma_k^{(0)} \to \sigma_{k+1}^{(0)}), \Gamma(\sigma_k^{(2)}) + \mu(\sigma_k^{(2)} \to \sigma_{k+1}^{(0)})\right\}$$
$$= \Gamma\left(\sigma_k^{(0)}\right) + \mu(\sigma_k^{(0)} \to \sigma_{k+1}^{(0)}).$$

Hence, the path that includes the state transition $\sigma_k^{(0)} \to \sigma_{k+1}^{(0)}$ is the survivor and the path that includes the state transition $\sigma_k^{(2)} \to \sigma_{k+1}^{(0)}$ is "dropped" as indicated by the "X" on the path.

8.2.5 BCJR Algorithm

The Viterbi algorithm uses MLSE to find the most likely input sequence. The BCJR algorithm, named after Bahl, Cocke, Jelinek, and Raviv [25], is a symbol-by-symbol maximum a posteriori probability (MAP) algorithm for decoding convolutional codes. Here we describe the BCJR algorithm for rate-$1/n$ convolutional codes with binary modulation.

Having observed the received sequence $\tilde{\mathbf{r}}$, the decoder calculates the a posteriori probabilities (APPs) $\mathrm{P}[a_k = +1|\tilde{\mathbf{r}}]$ and $\mathrm{P}[a_k = -1|\tilde{\mathbf{r}}]$, and decides $a_k = +1$ if $\mathrm{P}[a_k = +1|\tilde{\mathbf{r}}] > \mathrm{P}[a_k = -1|\tilde{\mathbf{r}}]$ and $a_k = -1$ otherwise. Alternatively, the decoder can calculate the a posteriori log-likelihood ratio (LLR)

$$L(a_k|\tilde{\mathbf{r}}) \stackrel{\triangle}{=} \log\left(\frac{\mathrm{P}[a_k = +1|\tilde{\mathbf{r}}]}{\mathrm{P}[a_k = -1|\tilde{\mathbf{r}}]}\right), \tag{8.71}$$

and make the decision $\hat{a}_k = \mathrm{sign}(L(a_k|\tilde{\mathbf{r}}))$. We first note that code bit a_k is output for the state transition $\sigma_k \to \sigma_{k+1}$, and there may be several such state transitions that will output the same code bit. It follows that the APP is

$$\begin{aligned}\mathrm{P}[a_k = u|\tilde{\mathbf{r}}] &= \frac{p(\tilde{\mathbf{r}}, a_k = u)}{p(\tilde{\mathbf{r}})} \\ &\propto p(\tilde{\mathbf{r}}, a_k = u) \\ &= \sum_{\sigma_k \to \sigma_{k+1}: a_k = u} p(\tilde{\mathbf{r}}, \sigma_k, \sigma_{k+1}).\end{aligned} \tag{8.72}$$

Hence, the a posteriori LLR can be written as

$$L(a_k|\tilde{\mathbf{r}}) = \log\left\{\frac{\sum_{\sigma_k \to \sigma_{k+1}: a_k = +1} p(\tilde{\mathbf{r}}, \sigma_k, \sigma_{k+1})}{\sum_{\sigma_k \to \sigma_{k+1}: a_k = -1} p(\tilde{\mathbf{r}}, \sigma_k, \sigma_{k+1})}\right\}. \tag{8.73}$$

One elegant form of the APPs can be obtained by first defining the sequences

$$\tilde{\mathbf{r}}_{j<k} = \{\tilde{\mathbf{r}}_{j,1}, \ldots, \tilde{\mathbf{r}}_{j,n}\}_{j=1}^{k-1}, \tag{8.74}$$

$$\tilde{\mathbf{r}}_{j>k} = \{\tilde{\mathbf{r}}_{j,1}, \ldots, \tilde{\mathbf{r}}_{j,n}\}_{j=k+1}^{N}, \tag{8.75}$$

$$\tilde{\mathbf{r}}_k = \{\tilde{\mathbf{r}}_{k,1}, \ldots, \tilde{\mathbf{r}}_{k,n}\}, \tag{8.76}$$

such that $\tilde{\mathbf{r}} = (\tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k, \tilde{\mathbf{r}}_{j>k})$, where N is the block length (equal to the number of input bits). Then the joint probability $p(\tilde{\mathbf{r}}, \sigma_k, \sigma_{k+1})$ can be split as follows [124]:

$$\begin{aligned} p(\tilde{\mathbf{r}}, \sigma_k, \sigma_{k+1}) &= p(\sigma_k, \sigma_{k+1}, \tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k, \tilde{\mathbf{r}}_{j>k}) \\ &= p(\sigma_k, \sigma_{k+1}, \tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k) p(\tilde{\mathbf{r}}_{j>k} | \sigma_k, \sigma_{k+1}, \tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k) \\ &= p(\sigma_k, \tilde{\mathbf{r}}_{j<k}) p(\tilde{\mathbf{r}}_k, \sigma_{k+1} | \sigma_k, \tilde{\mathbf{r}}_{j<k}) p(\tilde{\mathbf{r}}_{j>k} | \sigma_k, \sigma_{k+1}, \tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k) \\ &= p(\sigma_k, \tilde{\mathbf{r}}_{j<k}) \cdot p(\tilde{\mathbf{r}}_k, \sigma_{k+1} | \sigma_k) \cdot p(\tilde{\mathbf{r}}_{j>k} | \sigma_{k+1}) \\ &= \alpha_k(\sigma_k) \cdot \gamma_k(\sigma_k \to \sigma_{k+1}) \cdot \beta_{k+1}(\sigma_{k+1}), \end{aligned} \tag{8.77}$$

leading to three terms; the branch metric $\gamma_k(\sigma_k \to \sigma_{k+1})$, and the terms $\alpha_k(\sigma_k)$ and $\beta_{k+1}(\sigma_{k+1})$. The second last equality in (8.77) comes from the properties of the code trellis, which causes $\tilde{\mathbf{r}}_{j>k}$ to depend on $\sigma_k, \sigma_{k+1}, \tilde{\mathbf{r}}_{j<k}, \tilde{\mathbf{r}}_k$ only through the state σ_{k+1}, and the pair $\tilde{\mathbf{r}}_k, \sigma_{k+1}$ depends on $\sigma_k, \tilde{\mathbf{r}}_{j<k}$ only through the state σ_k.

The terms $\alpha_k(\sigma_k)$ and $\beta_{k+1}(\sigma_{k+1})$ can be calculated according to a forward recursion and a backward recursion, respectively. The forward recursion is given by

$$\begin{aligned} \alpha_{k+1}(\sigma_{k+1}) &= p(\sigma_{k+1}, \tilde{\mathbf{r}}_{j<k+1}) \\ &= p(\sigma_{k+1}, \tilde{\mathbf{r}}_k, \tilde{\mathbf{r}}_{j<k}) \\ &= \sum_{\sigma_k} p(\sigma_{k+1}, \sigma_k, \tilde{\mathbf{r}}_k, \tilde{\mathbf{r}}_{j<k}) \\ &= \sum_{\sigma_k} p(\tilde{\mathbf{r}}_{j<k}, \sigma_k) p(\tilde{\mathbf{r}}_k, \sigma_{k+1} | \sigma_k) \\ &= \sum_{\sigma_k} \alpha_k(\sigma_k) \cdot \gamma_k(\sigma_k \to \sigma_{k+1}) \end{aligned} \tag{8.78}$$

with the initial condition $\alpha_0(\sigma_0) = 1$, that is, the all-zeroes state σ_0 is the initial state in the code trellis.

Similarly, the backward recursion is given by

$$\begin{aligned} \beta_k(\sigma_k) &= p(\tilde{\mathbf{r}}_{j>k-1} | \sigma_k) \\ &= \sum_{\sigma_{k+1}} p(\sigma_{k+1}, \tilde{\mathbf{r}}_k, \tilde{\mathbf{r}}_{j>k} | \sigma_k) \\ &= \sum_{\sigma_{k+1}} p(\tilde{\mathbf{r}}_k, \sigma_{k+1} | \sigma_k) p(\tilde{\mathbf{r}}_{j>k} | \sigma_{k+1}) \\ &= \sum_{\sigma_{k+1}} \gamma_k(\sigma_k \to \sigma_{k+1}) \cdot \beta_{k+1}(\sigma_{k+1}) \end{aligned} \tag{8.79}$$

with the initial condition $\beta_N(\sigma_0) = 1$, that is, the all-zeroes state σ_0 is the ending state of the code trellis. Note that tail bits are required to terminate the trellis in state σ_0.

The branch metric $\gamma_k(\sigma_k \rightarrow \sigma_{k+1})$ has the form

$$\begin{aligned}\gamma_k(\sigma_k \rightarrow \sigma_{k+1}) &= p(\sigma_k, \tilde{\mathbf{r}}_i, \sigma_{k+1}) - p(\sigma_{k+1}|\sigma_k)p(\tilde{\mathbf{r}}_i|\sigma_{k+1}, \sigma_k) \\ &= \mathrm{P}[a_k = u]p(\tilde{\mathbf{r}}_k|a_k = u) \\ &= \mathrm{P}[a_k = u]p(\tilde{\mathbf{r}}_k|g_{k,1}\tilde{\mathbf{s}}_{k,1}, \ldots, g_{k,n}\tilde{\mathbf{s}}_{k,n}), \end{aligned} \tag{8.80}$$

where $\tilde{\mathbf{s}}_k = \{\tilde{\mathbf{s}}_{k,1}, \ldots, \tilde{\mathbf{s}}_{k,n}\}$ is the sequence of binary modulated symbols transmitted at epoch k, and $\mathbf{g}_k = \{g_{k,1}, \ldots, g_{k,n}\}$ is the corresponding sequence of complex channel gains at epoch k. The third line in (8.80) used the fact that there is a one-to-one correspondence between the state transition $\sigma_k \rightarrow \sigma_{k+1}$ in the code trellis and the input bit $a_k = u$, and where we have assumed that the state transition $\sigma_k \rightarrow \sigma_{k+1}$ is possible in the code trellis. Note that the branch metric depends on the prior probability $\mathrm{P}[a_k = u]$ of the information bit at epoch k, and on the conditional probability $p(\tilde{\mathbf{r}}_k|g_{k,1}\tilde{\mathbf{s}}_{k,1}, \ldots, g_{k,n}\tilde{\mathbf{s}}_{k,n})$ which is given by (8.67).

Finally, using (8.77) and (8.73) along with (8.78), (8.79), and (8.80), we obtain the a posteriori LLR

$$L(a_k|\tilde{\mathbf{r}}) = \log\left\{\frac{\sum_{\sigma_k \rightarrow \sigma_{k+1}: a_k = +1} \alpha_k(\sigma_k) \cdot \gamma_k(\sigma_k \rightarrow \sigma_{k+1}) \cdot \beta_{k+1}(\sigma_{k+1})}{\sum_{\sigma_k \rightarrow \sigma_{k+1}: a_k = -1} \alpha_k(\sigma_k) \cdot \gamma_k(\sigma_k \rightarrow \sigma_{k+1}) \cdot \beta_{k+1}(\sigma_{k+1})}\right\}, \tag{8.81}$$

and make decisions according to $\hat{a}_k = \mathrm{sign}(L(a_k|\tilde{\mathbf{r}}))$. Note that the $L(a_k|\tilde{\mathbf{r}})$ provide a level of certainty of the decoder about the value of a_k and are called soft outputs. These soft outputs are essential for the decoding of turbo codes that will be discussed later in this chapter.

8.2.5.1 Log-MAP Algorithm

The BCJR algorithm as described above exhibits numerical instability in the form of underflows and overflows. An alternative to this algorithm is its log-domain version known as the log-APP or log-MAP algorithm. Instead of using $\alpha_k(\sigma_k)$, $\beta_{k+1}(\sigma_{k+1})$ and $\gamma_k(\sigma_k \rightarrow \sigma_{k+1})$, we define their logarithms as follows:

$$\tilde{\alpha}_k(\sigma_k) = \log\{\alpha_k(\sigma_k)\}, \tag{8.82}$$

$$\tilde{\beta}_{k+1}(\sigma_{k+1}) = \log\{\beta_{k+1}(\sigma_{k+1})\}, \tag{8.83}$$

$$\tilde{\gamma}_k(\sigma_k \rightarrow \sigma_{k+1}) = \log\{\gamma_k(\sigma_k \rightarrow \sigma_{k+1})\}. \tag{8.84}$$

By examining (8.78) and (8.79), it can be seen that

$$\tilde{\alpha}_{k+1}(\sigma_{k+1}) = \log\left\{\sum_{\sigma_k} \mathrm{e}^{\tilde{\alpha}_k(\sigma_k)+\tilde{\gamma}_k(\sigma_k\to\sigma_{k+1})}\right\}, \tag{8.85}$$

$$\tilde{\beta}_k(\sigma_k) = \log\left\{\sum_{\sigma_{k+1}} \mathrm{e}^{\tilde{\gamma}_k(\sigma_k\to\sigma_{k+1})+\tilde{\beta}_{k+1}(\sigma_{k+1})}\right\} \tag{8.86}$$

with the initial conditions $\tilde{\alpha}_0(\sigma_0) = 0$ and $\tilde{\beta}_N(\sigma_0) = 0$, assuming that the code trellis begins and ends in state σ_0. The a posteriori LLR values are calculated according to

$$L(a_k|\tilde{\mathbf{r}}) = \log\left\{\frac{\sum_{\sigma_k\to\sigma_{k+1}:a_k=+1} \mathrm{e}^{\tilde{\alpha}_k(\sigma_k)+\tilde{\gamma}_k(\sigma_k\to\sigma_{k+1})+\tilde{\beta}_{k+1}(\sigma_{k+1})}}{\sum_{\sigma_k\to\sigma_{k+1}:a_k=-1} \mathrm{e}^{\tilde{\alpha}_k(\sigma_k)+\tilde{\gamma}_k(\sigma_k\to\sigma_{k+1})+\tilde{\beta}_{k+1}(\sigma_{k+1})}}\right\}. \tag{8.87}$$

To proceed further, we define the jacobian logarithm

$$\max^*\{x,y\} \stackrel{\triangle}{=} \log\{\mathrm{e}^x + \mathrm{e}^y\}, \tag{8.88}$$

$$\max^*\{x,y,z\} \stackrel{\triangle}{=} \log\{\mathrm{e}^x + \mathrm{e}^y + \mathrm{e}^z\}. \tag{8.89}$$

Note that

$$\max^*\{x,y\} = \max\{x,y\} + \log\{1+\mathrm{e}^{-|x-y|}\}. \tag{8.90}$$

The second term $\log\{1+\mathrm{e}^{-|x-y|}\}$ is small when x and y are not close, and its maximum value is equal to $\log\{2\}$ when $x = y$. Hence, when x and y are not close we can use the approximation

$$\max^*\{x,y\} \approx \max\{x,y\}. \tag{8.91}$$

When the above approximation is used in place of the jacobian logarithm, we obtain a suboptimal (but simpler) implementation the log-MAP algorithm called the max-log-MAP algorithm.

Using the jacobian logarithm, we can write

$$\tilde{\alpha}_{k+1}(\sigma_{k+1}) = \max_{\sigma_k}{}^*\left(\tilde{\alpha}_k(\sigma_k) + \tilde{\gamma}_k(\sigma_k\to\sigma_{k+1})\right), \tag{8.92}$$

$$\tilde{\beta}_k(\sigma_k) = \max_{\sigma_{k+1}}{}^*\left(\tilde{\gamma}_k(\sigma_k\to\sigma_{k+1}) + \tilde{\beta}_{k+1}(\sigma_{k+1})\right), \tag{8.93}$$

and the a posteriori LLR values are

$$L(a_k|\tilde{\mathbf{r}}) = \max_{\substack{\sigma_k \to \sigma_{k+1} \\ a_k=+1}}^{*} \left(\tilde{\alpha}_k(\sigma_k) + \tilde{\gamma}_k(\sigma_k \to \sigma_{k+1}) + \tilde{\beta}_{k+1}(\sigma_{k+1}) \right) \tag{8.94}$$

$$- \max_{\substack{\sigma_k \to \sigma_{k+1} \\ a_k=-1}}^{*} \left(\tilde{\alpha}_k(\sigma_k) + \tilde{\gamma}_k(\sigma_k \to \sigma_{k+1}) + \tilde{\beta}_{k+1}(\sigma_{k+1}) \right).$$

Example 8.4:

Consider the case of a rate-1/2 RSC code, for example, the encoder shown in Fig. 8.7. In this case, the transmitted sequence of code symbols is $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_n\}_{n=1}^{k}$, where $\tilde{\mathbf{s}}_n = (\tilde{s}_{n,s}, \tilde{s}_{n,p})$ and the terms with subscripts s and p correspond to the systematic (information bit) and parity check bit, respectively. For binary modulation, $\tilde{s}_{n,s}, \tilde{s}_{n,p} \in \{-\sqrt{2E_\mathrm{h}}, \sqrt{2E_\mathrm{h}}\}$, where $E_\mathrm{h} = E_\mathrm{c}$ is the energy per code bit. Likewise, the sequence of received vectors is $\tilde{\mathbf{r}} = \{\tilde{\mathbf{r}}_n\}_{n=1}^{k}$, where $\tilde{\mathbf{r}}_n = (\tilde{r}_{n,s}, \tilde{r}_{n,p})$.

Returning to the branch metric in (8.80), we have

$$\begin{aligned}
\gamma_k(\sigma_k \to \sigma_{k+1}) &= \mathrm{P}[a_k = u] p(\tilde{r}_{k,s}, \tilde{r}_{k,p} | g_{k,s}\tilde{s}_{k,s}, g_{k,p}\tilde{s}_{k,p}) \\
&= \frac{\mathrm{P}[a_k = u]}{(2\pi N_\mathrm{o})^N} \exp\left\{ -\frac{1}{2N_\mathrm{o}} \left(|\tilde{r}_{k,s} - g_{k,s}\tilde{s}_{k,s}|^2 + |\tilde{r}_{k,p} - g_{k,p}\tilde{s}_{k,p}|^2 \right) \right\} \\
&= \frac{1}{(2\pi N_\mathrm{o})^N} \exp\left\{ -\frac{1}{2N_\mathrm{o}} \left(|\tilde{r}_{k,s}|^2 + |\tilde{r}_{k,p}|^2 + 4E_\mathrm{c} \right) \right\} \\
&\quad \times \mathrm{P}[a_k = u] \exp\left\{ \frac{1}{N_\mathrm{o}} \left(\mathrm{Re}\{ g_{k,s}^{*} \tilde{r}_{k,s} \tilde{s}_{k,s} \} + \mathrm{Re}\{ g_{k,p}^{*} \tilde{r}_{k,p} \tilde{s}_{k,p} \} \right) \right\}.
\end{aligned} \tag{8.95}$$

Note that the term

$$\frac{1}{(2\pi N_\mathrm{o})^N} \exp\left\{ -\frac{1}{2N_\mathrm{o}} \left(|\tilde{r}_{k,s}|^2 + |\tilde{r}_{k,p}|^2 + 4E_\mathrm{c} \right) \right\}$$

is independent of a_k and will cancel in the numerator and denominator of the a posteriori LLR in (8.87) and, therefore, can be ignored. Also, the numerator in (8.87) has $\tilde{s}_{k,s} = \sqrt{2E_\mathrm{c}}$ corresponding to $a_k = +1$, while the denominator has $\tilde{s}_{k,s} = -\sqrt{2E_\mathrm{c}}$ corresponding to $a_k = -1$. It follows that the a posteriori LLR in (8.81) becomes

$$L(a_k|\tilde{\mathbf{r}}) = \frac{2\sqrt{2E_c}g^*_{k,s}\tilde{r}_{k,s}}{N_o} + \log\left\{\frac{\mathrm{P}[a_k=+1]}{\mathrm{P}[a_k=-1]}\right\} + \log\left\{\frac{\sum_{\sigma_k\to\sigma_{k+1}:a_k=+1}\alpha_k(\sigma_k)\cdot\exp\left\{\frac{\mathrm{Re}\{g^*_{k,p}\tilde{r}_{k,p}\tilde{s}_{k,p}\}}{N_o}\right\}\cdot\beta_{k+1}(\sigma_{k+1})}{\sum_{\sigma_k\to\sigma_{k+1}:a_k=-1}\alpha_k(\sigma_k)\cdot\exp\left\{\frac{\mathrm{Re}\{g^*_{k,p}\tilde{r}_{k,p}\tilde{s}_{k,p}\}}{N_o}\right\}\cdot\beta_{k+1}(\sigma_{k+1})}\right\}. \tag{8.96}$$

Note that the symbols $\tilde{s}_{k,p}$ are not conjugated in the above expression since they are real-valued in this example. If the Log-MAP algorithm is used, then the a posteriori LLR becomes

$$L(a_k|\tilde{\mathbf{r}}) = L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_s) + L_a(a_k) + L_e(a_k|\tilde{\mathbf{r}}_p), \tag{8.97}$$

where $L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_s)$, $L_a(a_k)$ and $L_e(a_k|\tilde{\mathbf{r}}_p)$ are, respectively, defined as

$$L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_s) = \frac{2\sqrt{2E_c}g^*_{k,s}\tilde{r}_{k,s}}{N_o}, \tag{8.98}$$

$$L_a(a_k) = \log\left\{\frac{\mathrm{P}[a_k=+1]}{\mathrm{P}[a_k=-1]}\right\}, \tag{8.99}$$

$$L_e(a_k|\tilde{\mathbf{r}}_p) = \max^*_{\substack{\sigma_k\to\sigma_{k+1}\\ a_k=+1}}\left(\tilde{\alpha}_k(\sigma_k) + \frac{\mathrm{Re}\{g^*_{k,p}\tilde{r}_{k,p}\tilde{s}_{k,p}\}}{N_o} + \tilde{\beta}_{k+1}(\sigma_{k+1})\right) - \max^*_{\substack{\sigma_k\to\sigma_{k+1}\\ a_k=-1}}\left(\tilde{\alpha}_k(\sigma_k) + \frac{\mathrm{Re}\{g^*_{k,p}\tilde{r}_{k,p}\tilde{s}_{k,p}\}}{N_o} + \tilde{\beta}_{k+1}(\sigma_{k+1})\right). \tag{8.100}$$

The a posteriori LLR consists of three terms; the first depends on the channel output due to the systematic component, the second depends on the a priori probabilities of the information bits, and the third depends on the channel outputs due to the parity bits. Usually $\mathrm{P}[a_k=+1] = \mathrm{P}[a_k=-1] = 1/2$ for convolutional decoders, so that the a priori term $L_a(a_k)$ is zero. However, for the iterative decoders that are used with the turbo codes discussed later in this chapter, the decoder will receive *extrinsic* of *soft* information for each a_k which serves as a priori information. Once the a posteriori LLR has been calculated, the extrinsic information can be calculated as

$$L_e(a_k|\tilde{\mathbf{r}}_p) = L(a_k|\tilde{\mathbf{r}}) - L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_s) - L_a(a_k). \tag{8.101}$$

8.3 Trellis Coded Modulation

8.3.1 Encoder Description

Conventional convolutional codes realize a coding gain at the expense of the coded modulation efficiency or bits/s/Hz. Although convolutional codes may be attractive for power-limited applications, they are less suitable for bandwidth-limited applications. One approach for overcoming the loss of coded modulation efficiency is to map the encoder output bits onto a higher-order signal constellation, such as M-PSK or M-QAM. However, the symbol mapping that is used in this case is very critical. Using a straight forward mapping, such as Gray mapping or natural mapping, will give disappointing results because the decreased Euclidean distance between the signals points in the higher-order constellation will tend to offset the benefits gained from using the convolutional code. Ungerboeck showed that a coding gain can be achieved without sacrificing data rate or bandwidth using a rate-$m/(m+r)$ convolutional encoder, and mapping the coded bits onto signal points $\{x_k\}$ through a technique called mapping by set partitioning [258]. This combination of coding and modulation, called TCM, has three basic features:

1. An expanded signal constellation is used that is larger than the one necessary for uncoded modulation at the same data rate. The additional signal points allow redundancy to be inserted without sacrificing data rate or bandwidth.
2. The expanded signal constellation is partitioned such that the intra-subset minimum squared Euclidean distance is maximized at each step in the partition chain.
3. Convolutional encoding and signal mapping is used so that only certain sequences of signal points are allowed.

Figure 8.9 shows the basic encoder structure for Ungerboeck's trellis codes. The n-bit information vector $\mathbf{a}_k = (a_k^{(1)}, \ldots, a_k^{(n)})$ is transmitted at epoch k. At each epoch k, $m \leq n$ data bits are encoded into $m+r$ code bits using a rate-$m/(m+r)$

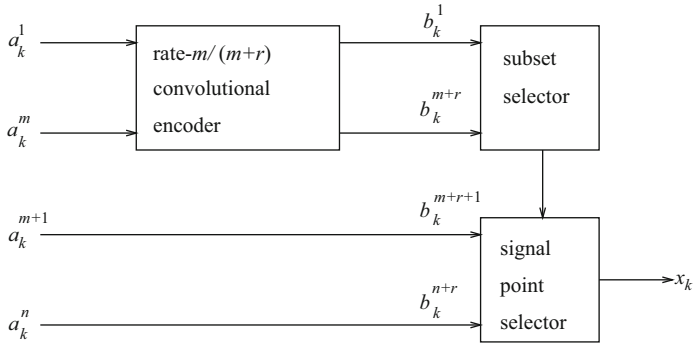


Fig. 8.9 Ungerboeck trellis encoder

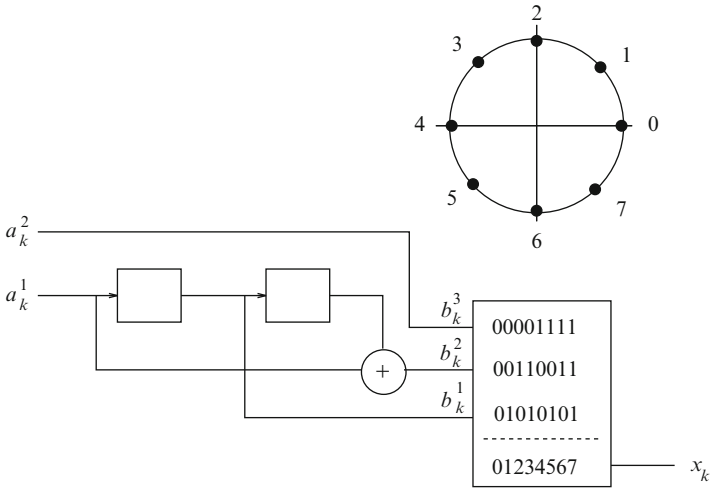


Fig. 8.10 Encoder and signal mapping for the 4-state 8-PSK Ungerboeck trellis code

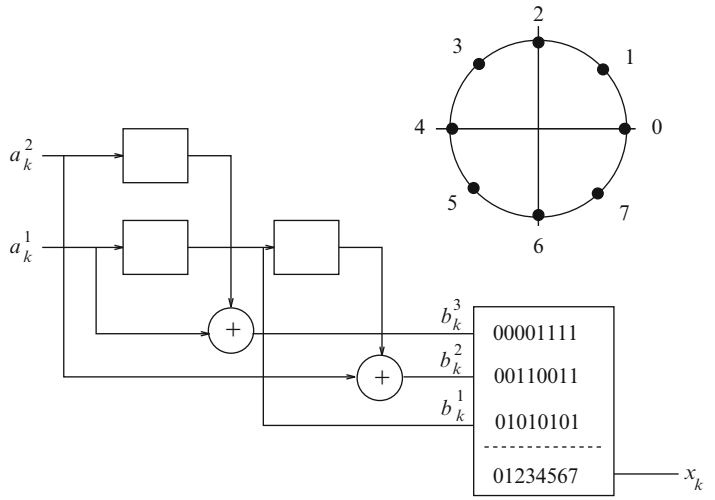


Fig. 8.11 Encoder and signal mapping for the eight-state 8-PSK Ungerboeck trellis code

linear convolutional encoder. The $m + r$ code bits select one of 2^{m+r} subsets of a 2^{n+r}-point signal constellation. The remaining $n - m$ data bits are used to select one of the 2^{n-m} signal points within the selected subset. This principle is best explained by example, and Fig. 8.10 shows a 4-state 8-PSK Ungerboeck trellis code. The equivalent uncoded system is 4-PSK which has a rate of 2 bits/symbol. The 4-state 8-PSK code uses a rate-1/2 convolutional code along with one uncoded bit to select signal points in an expanded 8-PSK signal constellation. Note that the overall rate is still 2 bits/symbol. Figure 8.11 shows another example of an eight-state

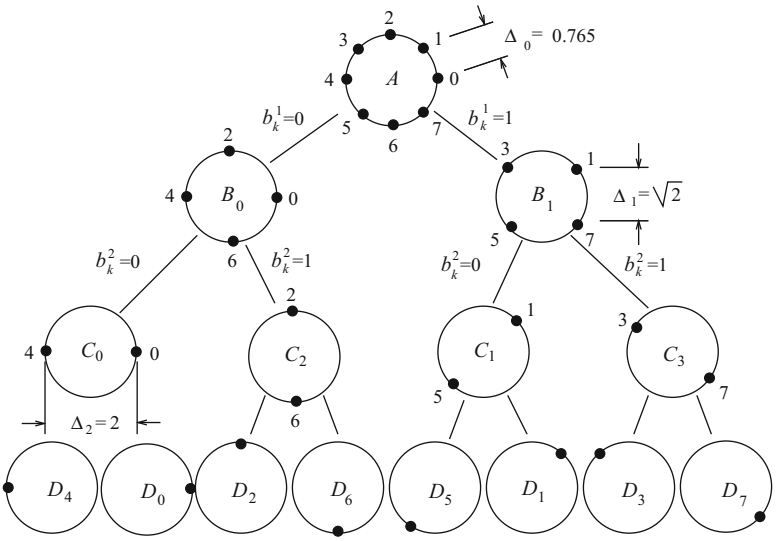


Fig. 8.12 Set partitioning for an 8-PSK signal constellation

8-PSK Ungerboeck trellis code. The equivalent uncoded system is again 8-PSK with 2 bits/symbol. The eight-state 8-PSK code uses a rate-2/3 convolutional code to select one of the points in an expanded 8-PSK signal constellation so that the overall rate is again 2 bits/symbol.

8.3.2 Mapping by Set Partitioning

The critical step in the design of Ungerboeck's codes is the method of mapping by set partitioning. Figure 8.12 shows how the 8-PSK signal constellation is partitioned into subsets such that the intra-subset minimum squared Euclidean distance is maximized for each step in the partition chain. Here we assume a normalized 8-PSK signal constellation having eight signal points uniformly spaced around a circle of unit radius. Notice that the minimum Euclidean distance between signal points in the normalized 8-PSK signal constellation is $\Delta_0 = 0.765$, while the minimum Euclidean distances between signal points in the first and second level partitions are $\Delta_1 = \sqrt{2}$ and $\Delta_2 = 2$, respectively. Notice that the minimum Euclidean distance increases at each level of partitioning.

The advantages of using TCM can most easily be seen by considering the trellis diagram. For both the 4-state and eight-state 8-PSK trellis codes, the equivalent uncoded system is 4-PSK. The trellis diagram for uncoded 4-PSK is shown in Fig. 8.13. The trellis only has one state and there are four parallel transitions between the states. The subsets D_0, D_2, D_4, and D_6 are used as the signal points. The label

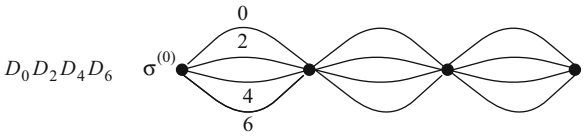


Fig. 8.13 Trellis diagram for uncoded 4-PSK

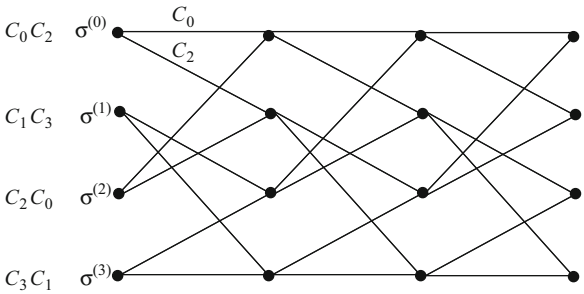


Fig. 8.14 Trellis diagram for 4-state 8-PSK Ungerboeck trellis code

D_0, D_2, D_4, D_6 means that the branches in the trellis diagram are labeled from top to bottom with signal points taken from the sets D_0, D_2, D_4, D_6. The minimum Euclidean distance between any two paths through the trellis is $d_{\min} = \sqrt{2}$.

The trellis diagram for the 4-state 8-PSK code is shown in Fig. 8.14. Each branch in the 4-state trellis is labeled with one of the four subsets C_0, C_1, C_2, and C_3. Again, the label C_iC_j associated with a state means that the branches in the trellis diagram originating from that state are labeled from top to bottom with the subsets C_i and C_j. As shown in Fig. 8.12, each subset C_i contains two signal points. Thus, each branch in the trellis diagram actually contains two parallel transitions. For example, branches with the label C_0 have two parallel transitions that are labeled with the signal points 0 and 4. For the 4-state 8-PSK code, it is possible that two coded sequences could differ by just a single parallel transition with a Euclidean distance of $d = 2$. Also, any two signal paths that diverge from a state and merge with the same state after more that one transition have a minimum Euclidean distance of $d = \sqrt{\Delta_1^2 + \Delta_0^2 + \Delta_1^2} = 2.141$. For example, the closest nonparallel code sequence to the all-zeroes sequence $\mathbf{x} = (0,0,0)$ is the sequence $\mathbf{x} = (2,1,2)$ at Euclidean distance $d = 2.141$. Hence, the minimum Euclidean distance of the code over all parallel and nonparallel pairs of sequences for the 4-state 8-PSK code is $d_{\min} = 2$.

The concept of mapping by set partitioning was developed by Ungerboeck as a method for maximizing the minimum Euclidean distance of a code and consequently to optimize its performance on an AWGN channel. Ungerboeck's construction of the optimum 4-state 8-PSK code was based on the following heuristic rules [259]:

1. Parallel transitions (when they occur) are assigned signal points having the maximum Euclidean distance between them.
2. The transition starting or ending in any state is assigned the subsets (C_0, C_2) or (C_1, C_3) which have a maximum distance between them.
3. All signal points are used in the trellis diagram with equal frequency.

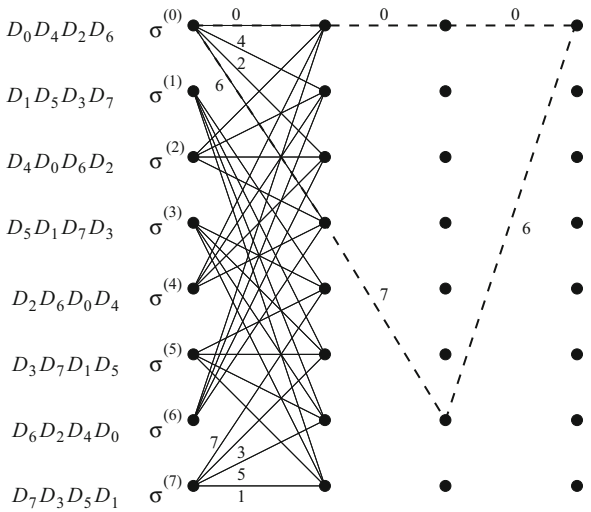


Fig. 8.15 Trellis diagram for 8-state 8-PSK Ungerboeck trellis code. The dashed lines show two minimum distance paths

It is clear that the performance of the 4-state 8-PSK code is limited by the parallel transitions. Larger asymptotic coding gains can be obtained by introducing more code states so that the parallel transitions are eliminated. For example, the above design rules can be applied to the 8-state 8-PSK code to obtain the code trellis shown in Fig. 8.15. In this case, the minimum Euclidean distance is $d_{\min} = \sqrt{\Delta_1^2 + \Delta_0^2 + \Delta_1^2} = 2.141$.

8.4 Code Performance on AWGN Channels

Viterbi originally exploited the trellis structure of convolutional codes and developed the Viterbi algorithm for ML decoding of convolutional codes [266, 267]. Forney recognized the analogy between an ISI channel and a convolutional encoder, and applied the Viterbi algorithm for the detection of digital signals corrupted by ISI and AWGN as discussed in Sect. 7.4.1 [104]. Given the similarity between the trellis structures of ISI channels, convolutional codes, and trellis codes (e.g., compare Figs. 7.11, 8.6, and 8.14), it is not surprising that the union bounding techniques that we used to evaluate the error probability of digital signaling on ISI channels with an MLSE receiver in Sect. 7.5 can be applied, with appropriate modification, to evaluate the error probability of convolutional and trellis codes with an MLSE receiver.

To develop the union bound on decoded bit error probability, let $\mathbf{a} = \{\mathbf{a}_k\}$ denote the transmitted information sequence. For any other sequence $\hat{\mathbf{a}} \neq \mathbf{a}$, define the corresponding error sequence as $\mathbf{e} = \{e_k\} = \mathbf{a} \oplus \hat{\mathbf{a}}$, where $\oplus$ denotes modulo-2 addition. Since the bit error probability at epoch j_1 is of interest, $e_{j_1} \neq 0$ for all error

sequences. An error event occurs between k_1 and k_2 of length $k_2 - k_1$, if $\sigma_{k_1} = \hat{\sigma}_{k_1}$ and $\sigma_{k_2} = \hat{\sigma}_{k_2}$, but $\sigma_j \neq \hat{\sigma}_j$ for $k_1 < j < k_2$, where $k_1 \leq j_1 < k_2$, and $\sigma = \{\sigma_k\}$ and $\hat{\sigma} = \{\hat{\sigma}_k\}$ are the system state sequences associated with $\mathbf{a}$ and $\hat{\mathbf{a}}$, respectively. Let E be the set of error sequences that have the first nonzero element starting at time j_1. Then, the average bit error probability is bounded by

$$P_{\mathrm{b}} \leq \frac{1}{n} \sum_{\mathbf{e} \in E} w_{\mathrm{b}}(\mathbf{e}) \sum_{\mathbf{a}} \mathrm{P}[\mathbf{a}] \; \mathrm{P}\left[\Gamma(\mathbf{a} \oplus \mathbf{e}) \geq \Gamma(\mathbf{a}) \middle| \mathbf{a} \right], \tag{8.102}$$

where $\Gamma(\mathbf{a})$ is the path metric of $\mathbf{a}$, and $w_{\mathrm{b}}(\mathbf{e})$ is the number of bit errors associated with $\mathbf{e}$. The factor $1/n$ appears in front of the first summation, because n information bits are transmitted per epoch (or per branch in the trellis diagram). The second summation is over all possible information sequences, because each sequence $\mathbf{a}$ can have $\mathbf{e}$ as the error sequence. This is necessary for TCM because the signal mapping and, hence, the codes are nonlinear.

Another way of writing the bound on the bit error probability in (8.102) is

$$P_{\mathrm{b}} \leq \sum_{\tilde{\mathbf{s}} \in C} \sum_{\hat{\mathbf{s}} \in C} w_{\mathrm{b}}(\tilde{\mathbf{s}}, \hat{\mathbf{s}}) \mathrm{P}[\tilde{\mathbf{s}}] \mathrm{P}[\tilde{\mathbf{s}} \rightarrow \hat{\mathbf{s}}], \tag{8.103}$$

where C is the set of all coded symbol sequences, $w_{\mathrm{b}}(\tilde{\mathbf{s}}, \hat{\mathbf{s}})$ is the number of bit errors that occur when the complex symbol sequence $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_i\}$ is transmitted and the complex symbol sequence $\hat{\mathbf{s}} \neq \tilde{\mathbf{s}}$ is chosen by the decoder, $\mathrm{P}[\tilde{\mathbf{s}}]$ is the a priori probability of transmitting $\tilde{\mathbf{s}}$, and $\mathrm{P}[\tilde{\mathbf{s}} \rightarrow \hat{\mathbf{s}}]$ is the pairwise error probability.

At high signal-to-noise ratio (SNR), the BER performance on an AWGN channel is dominated by the minimum Euclidean distance error events. The pairwise error probability between two coded symbol sequences $\tilde{\mathbf{s}}$ and $\hat{\mathbf{s}}$ separated by Euclidean distance $d_{\min}$ is

$$\mathrm{P}[\tilde{\mathbf{s}} \rightarrow \hat{\mathbf{s}}] = Q\left(\sqrt{\frac{d_{\min}^2}{4N_{\mathrm{o}}}} \right). \tag{8.104}$$

The asymptotic coding gain (at high SNR) is defined by [36]

$$G_{\mathrm{a}} \; = \; 10 \log_{10} \frac{(d_{\min,\mathrm{coded}}^2 / E_{\mathrm{av,coded}})}{(d_{\min,\mathrm{uncoded}}^2 / E_{\mathrm{av,uncoded}})} \;\; \mathrm{dB} \tag{8.105}$$

where E_{av} is the average energy per symbol in the signal constellation. For the 4-state 8-PSK code shown in Fig. 8.10, the asymptotic coding gain is $G_{\mathrm{a}} = 3$ dB over uncoded 4-PSK. Likewise, the 8-state 8-PSK code in Fig. 8.11 has an asymptotic coding gain of 3.6 dB over uncoded 4-PSK.

8.4.1 Union Bound for Convolutional Codes

For convolutional codes, the upper bound in (8.102) simplifies because the codes are linear, meaning that the sum of any two codewords is another codeword and that all-zeroes sequence is a codeword [159]. Because of this property, we can assume that the all-zeroes sequence was transmitted, that is, $\mathbf{a} = \mathbf{0}$, so that the union bound becomes

$$P_{\mathrm{b}} \leq \frac{1}{k} \sum_{\mathbf{e} \in E} w_{\mathrm{b}}(\mathbf{e})\, \mathrm{P}\left[\, \Gamma(\mathbf{e}\,) \geq \Gamma(\mathbf{0}) \right]. \tag{8.106}$$

Note that we divide by k rather than n in front of the summation, because a convolutional code transmits k bits per epoch, whereas a trellis code transmits n bits per epoch.

For convolutional codes, the set E in (8.106) consists of all sequences that begin and end at the zero-state in the state diagram. The enumeration of these sequences (or codewords) along with their associated Hamming distances, information weights, and lengths was obtained earlier by computing the transfer function, $T(D,N,L)$, of the augmented state diagram. When a particular incorrect path through the trellis is selected over the all-zeroes path at a given node in the trellis, the corresponding number of bits errors, $w_{\mathrm{b}}(\mathbf{e})$, is given by the exponent of N in the transfer function. Multiplying $w_{\mathrm{b}}(\mathbf{e})$ by the pairwise error probability $\mathrm{P}\left[\, \Gamma(\mathbf{e}\,) \geq \Gamma(\mathbf{0}) \right]$ for that path and dividing by the number of input bits per branch, k, give the bit error probability associated with that path. Summing over the set of all possible incorrect sequences E yields a union bound on the bit error probability.

The pairwise error probability in (8.106) depends on the type of modulation, detection, and decoding that is used. The code bits are mapped onto symbols taken from a signal constellation and transmitted over the channel. Assuming an AWGN channel and a coherent receiver, the received vector (see Sect. 5.1) at epoch n is

$$\tilde{\mathbf{r}}_n = \tilde{\mathbf{s}}_n + \tilde{\mathbf{n}}_n, \tag{8.107}$$

where $\tilde{\mathbf{s}}_n$ is the transmitted symbol vector and $\tilde{\mathbf{n}}_n$ is the Gaussian noise vector at epoch n. For convolutional codes, two types of decoding are commonly used: hard decision decoding and soft decision decoding. Soft decision decoders use the sequence of received signal vectors $\mathbf{r} = \{\mathbf{r}_n\}$ to make sequence decisions. For an AWGN channel, the MLSE receiver searches for the sequence of symbol vectors $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_n\}$ that is closest in Euclidean distance to the received sequence of signal vectors $\mathbf{r}$. To do so, the MLSE receiver chooses the sequence $\hat{\mathbf{s}}$ that minimizes the metric

$$\mu(\hat{\mathbf{s}}) = \|\mathbf{r} - \hat{\mathbf{s}}\|^2. \tag{8.108}$$

The decided sequence $\hat{\mathbf{s}}$ maps one-to-one onto the data bit sequence $\hat{\mathbf{a}}$ that is the final estimate of the transmitted information sequence.

In general, the pairwise error probability for an AWGN channel that is associated with an error event of length ℓ beginning at epoch k_1 is

$$P_2(\ell) = Q\left(\sqrt{\frac{\Delta^2}{4N_0}}\right), \tag{8.109}$$

where

$$\Delta^2 = \sum_{k=k_1}^{k_1+\ell+1} \delta_k^2, \tag{8.110}$$

$$\delta_k^2 = \|\tilde{\mathbf{s}}_k - \hat{\mathbf{s}}_k\|^2 \tag{8.111}$$

and $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_k\}$ and $\hat{\mathbf{s}} = \{\hat{\mathbf{s}}_k\}$ are the symbol sequences corresponding to the data bit sequences $\tilde{\mathbf{a}}$ and $\hat{\mathbf{a}}$, respectively. The parameter δ_k^2 is the squared branch Euclidean distance associated with branch k, and Δ^2 is the squared path Euclidean distance associated with the error event. Clearly, the pairwise error probability depends on the particular mapping between the encoder output bits and the points in the signal constellation. Suppose for example that code bits are mapped onto a BPSK signal constellation. Then the pairwise error probability between the two codewords $\tilde{\mathbf{b}}$ and $\hat{\mathbf{b}}$ that differ in d positions is

$$P_2(d) = Q(\sqrt{2R_c d\gamma_b}), \tag{8.112}$$

where γ_b is the received bit energy-to-noise ratio.[2] Note that we have explicitly shown the pairwise error probability to be a function of the Hamming distance between the codewords in (8.112). However, it is important to realize that this property does not always apply. For example, if the outputs of the rate-2/3 convolutional encoder in Fig. 8.2 are mapped onto symbols from an 8-PSK signal constellation, then the pairwise error probability depends not only on the Hamming distance between codewords, but also upon the particular mapping between the 8-PSK symbols and the encoder outputs.

In general, the transfer function $T(D,N)$ for a convolutional code has the form

$$T(D,N) = \sum_{d=d_{\text{free}}}^{\infty} a_d D^d N^{f(d)}, \tag{8.113}$$

where $f(d)$ is the exponent of N as a function of d. For the example in (8.52), $a_d = 2^{d-5}$ and $f(d) = d-4$. Differentiating $T(D,N)$ with respect to N and setting $N = 1$ gives

$$\frac{\mathrm{d}T(D,N)}{\mathrm{d}N}\bigg|_{N=1} = \sum_{d=d_{\text{free}}}^{\infty} a_d f(d) D^d. \tag{8.114}$$

[2]The received symbol energy-to-noise ratio is $\gamma_s = R_c\gamma_b$

Once again, for the example in (8.52) this leads to

$$\frac{\mathrm{d}T(D,N)}{\mathrm{d}N}\Big|_{N=1} = \sum_{d=d_{\text{free}}}^{\infty} 2^{d-5}(d-4)D^d. \tag{8.115}$$

Using this notation, the union bound on bit error probability for convolutional coding with BPSK modulation is

$$P_{\text{b}} \leq \frac{1}{k} \sum_{d=d_{\text{free}}}^{\infty} a_d f(d) P_2(d), \tag{8.116}$$

where $P_2(d)$ is given by (8.112).

In contrast to soft decision decoders, hard decision decoders first make symbol-by-symbol decisions on the sequence of received vectors $\mathbf{r} = \{\mathbf{r}_k\}$ to yield the sequence of symbol decisions $\tilde{\mathbf{s}} = \{\tilde{\mathbf{s}}_k\}$. The decoder then operates on the sequence $\tilde{\mathbf{s}}$ to estimate the most likely transmitted data sequence. A minimum distance decoder is one that decides in favor of the symbol sequence $\hat{\mathbf{s}}$ that is closest in Hamming distance to the received symbol sequence $\tilde{\mathbf{s}}$. Again, the pairwise error probability depends on the particular mapping between the encoder outputs and the points in the signal constellation. If BPSK signaling is used, for example, then the pairwise error probability between two codewords $\tilde{\mathbf{b}}$ and $\hat{\mathbf{b}}$ at Hamming distance d is

$$P_2(d) = \begin{cases} \sum_{k=(d+1)/2}^{d} \binom{d}{k} p^k (1-p)^{d-k}, & d \text{ odd} \\ \sum_{k=d/2+1}^{d} \binom{d}{k} p^k (1-p)^{d-k} + \frac{1}{2}\binom{d}{d/2} p^{d/2}(1-p)^{d/2}, & d \text{ even} \end{cases}, \tag{8.117}$$

where

$$p = Q(\sqrt{2R_{\text{c}}\gamma_{\text{b}}}) \tag{8.118}$$

is the probability of symbol error. Once again, the pairwise error probability for BPSK is a function of the Hamming distance between the codewords.

8.4.1.1 Union-Chernoff Bound for Convolutional Codes

The union bound in (8.116) can be simplified by imposing a Chernoff bound (see Appendix) on the pairwise error probability. First consider the case of soft decision decoding. Suppose that sequence $\tilde{\mathbf{s}}$ is transmitted and $\mathbf{r}$ is the received sequence. Then the pairwise error probability between sequences $\tilde{\mathbf{s}}$ and $\hat{\mathbf{s}}$ with an maximum likelihood receiver can be Chernoff bounded by

$$\begin{aligned} \mathrm{P}[\tilde{\mathbf{s}} \to \hat{\mathbf{s}}] &= \mathrm{P}\left[\|\mathbf{r}-\hat{\mathbf{s}}\|^2 < \|\mathbf{r}-\tilde{\mathbf{s}}\|^2\right] \\ &\leq \mathrm{E}\left[\mathrm{e}^{\lambda\left(\|\mathbf{r}-\tilde{\mathbf{s}}\|^2 - \|\mathbf{r}-\hat{\mathbf{s}}\|^2\right)} \,|\tilde{\mathbf{s}}\right]. \end{aligned} \tag{8.119}$$

Substituting $\mathbf{r} = \tilde{\mathbf{s}} + \mathbf{n}$, taking the expectation over the complex Gaussian random vector $\mathbf{n}$, and simplifying gives

$$\mathrm{P}[\tilde{\mathbf{s}} \to \hat{\mathbf{s}}] \leq \mathrm{e}^{-\lambda \|\tilde{\mathbf{s}}-\hat{\mathbf{s}}\|^2 (1-\lambda 2N_\mathrm{o})}. \tag{8.120}$$

The tightest upper bound is obtained with $\lambda^* = 1/(4N_\mathrm{o})$ yielding

$$\mathrm{P}[\tilde{\mathbf{s}} \to \hat{\mathbf{s}}] \leq \mathrm{e}^{-\|\tilde{\mathbf{s}}-\hat{\mathbf{s}}\|^2/8N_\mathrm{o}}. \tag{8.121}$$

For the case of BPSK signaling on an AWGN channel, the Chernoff bound on the pairwise error probability becomes

$$P_2(d) \leq \mathrm{e}^{-d\gamma_\mathrm{s}} = \mathrm{e}^{-R_\mathrm{c} d \gamma_\mathrm{b}}, \tag{8.122}$$

where d is the number of coordinates in which the sequences $\tilde{\mathbf{s}}$ and $\hat{\mathbf{s}}$ differ, $\gamma_\mathrm{s} = E_\mathrm{h}/N_\mathrm{o}$ is the received symbol energy-to-noise ratio, R_c is the code rate, and γ_b is the received bit energy-to-noise ratio. Likewise, if BPSK signaling is used with hard decision decoding, it can be shown that the pairwise error probability has the Chernoff bound

$$P_2(d) \leq [4p(1-p)]^{d/2}, \tag{8.123}$$

where the code bit error probability p is given by (8.118).

Notice how the Hamming distance d appears in the exponent of the Chernoff bound on pairwise error probability with either hard or soft decision decoding. From (8.114) and (8.116), it is apparent that the decoded bit error probability has the following union-Chernoff bound:

$$P_\mathrm{b} \leq \frac{1}{k} \frac{\mathrm{d}T(D,N)}{\mathrm{d}N} \Big|_{N=1, D=Z}, \tag{8.124}$$

where

$$Z = \begin{cases} \sqrt{4p(1-p)}, & \text{hard decision decoding} \\ \mathrm{e}^{-R_\mathrm{c}\gamma_\mathrm{b}}, & \text{soft decision decoding} \end{cases}. \tag{8.125}$$

At high SNR, the performance is dominated by the error events with minimum Hamming distance. Since the minimum distance error events are not necessarily mutually exclusive, the decoded bit error probability with BPSK at high SNR is approximately

$$P_\mathrm{b} \approx \frac{1}{k} a_{d_\mathrm{free}} f(d_\mathrm{free}) P_2(d_\mathrm{free}) \leq \frac{1}{k} a_{d_\mathrm{free}} f(d_\mathrm{free}) Z^{d_\mathrm{free}}. \tag{8.126}$$

The above procedure for upper bounding the decoded bit error probability is sometimes called the transfer function approach, because it uses the transfer function of the augmented state diagram. However, the transfer function approach has its limitations. First, as the number of encoder states becomes large, it becomes difficult to compute the transfer function $T(D,N)$. Second, for nonbinary signal

constellations, the pairwise error probability will depend on the particular pair of modulated symbol sequences, and not just on the Hamming distance between them. In this case, the branch labeling in the augmented state diagram must be done differently and the Chernoff bound cannot be used. These problems can be overcome using a different approach to compute the upper bound, such as a stack algorithm [240].

8.5 Interleaving

Most error control codes are designed for memoryless channels, where successively transmitted code symbols experience independent channel conditions. Although this memoryless property holds for AWGN channels, it is not usually the case for wireless channels due to the temporal correlation of the channel, as discussed in Chap. 2. An effective method for dealing with such channels is to use interleaving. There are two basic approaches to interleaving, namely symbol interleaving and bit interleaving. A symbol interleaver performs symbol-by-symbol interleaving of the code symbols *after* they have been mapped onto a signal constellation. A bit interleaver, on the other hand, performs bit-by-bit interleaving of the encoder outputs *before* they are mapped onto a signal constellation.

The interleaving/deinterleaving operation serves to reduce the correlation between the fades experienced by successive code bits at the output of the encoder. Since, most error control codes are designed for memoryless channels, this will generally improve the decoded BER performance. Indeed if the interleaver/deinterleaver was eliminated and the channel fades slowly, it is possible that entire codewords could be received with either a low or a high SNR. Under a low instantaneous SNR condition even a very low-rate code will fail, while under a high instantaneous SNR condition even very high-rate codes will succeed. Under such conditions, the error control code is ineffective.

8.5.0.2 Block Interleaving

A block interleaver can be regarded as a buffer with J rows and M columns, where J represents the interleaving depth and M represents the interleaving span. Such an interleaver is called a (J,M) block interleaver, and the length of the interleaver is JM. The block interleaver can be used to perform either symbol interleaving or bit interleaving. In the case of symbol (bit) interleaving, the code symbols (bits) are input to the buffer row-wise from top to bottom and output from the buffer column-wise from left to right. The deinterleaver simply performs the reverse operation. The block interleaver has the following characteristics:

1. Any burst of symbol (bit) errors of length $j \leq J$ results in single symbol (bit) errors at the deinterleaver output that are each separated by at least M symbols (bits).

2. Any burst of symbol (bit) errors of length $j = kJ, k > 1$ results in bursts of no more than $\lceil k \rceil$ symbol (bit) errors that are separated by no less than $M - \lceil k \rceil$ symbols (bits).
3. A periodic sequence of single symbol (bit) errors that are spaced J code symbols (bits) apart results in a single burst of symbol (bit) errors of length M at the deinterleaver output.
4. The end-to-end interleaving delay is equal to $2JM$ code symbols (bits), and the memory requirements for the interleaver and deinterleaver is equal to JM code symbols (bits).

In practise, the interleaver depth J should be chosen so that successive code symbols (bits), which are transmitted J code symbol (bit) durations apart, are independently faded. Chapter 2 showed that the temporal autocorrelation function of a cellular land mobile radio channel observed at either the mobile station or the base station with 2D isotropic scattering around the mobile station is $\phi_{gg}(\tau) = \frac{\Omega_p}{2} J_0(2\pi f_m \tau)$, where $f_m = v/\lambda_c$ with v being the mobile station velocity and λ_c the carrier wavelength. From Fig. 2.5, we can see that the temporal autocorrelation is small provided that $f_m \tau > 0.5$. Hence, for effective interleaving, we should have $J > 0.5\lambda_c/vT$, where T is the code symbol (bit) duration. The choice of the parameter M depends on the type of coding used. For block codes M should be larger than the block length. For convolutional and trellis codes, the interleaver should separate any $L_D + 1$ successive code symbols (bits) as far apart as possible, where L_D is the decoding depth. Hence, we should have $M \geq L_D + 1$. For convolutional and trellis codes, it is well known that the decoding depth $L_D \approx 5K$, where K is the constraint length.

Observe that the required interleaving depth is inversely proportional to the speed of the mobile station and, therefore, slower moving mobile stations require larger interleaving depths. Unfortunately, interleaving introduces delay into the link, since we must fill the $J \times M$ interleaver array before the symbols (bits) can be transmitted, and later fill the $J \times M$ deinterleaver array before the symbols (bits) can be decoded. However, delay critical traffic such as real-time voice will impose a limit on the interleaving delay defined as $t_d = JMT$ and, therefore, the interleaving will be insufficient at low speeds. For example, if $f_c = 900\,\text{MHz}$, $R = 1/T = 24 \times 10^3$, and $v = 30$ km/h, we require $J > 478$. For a constraint length $K = 3$ convolutional code with a decoding depth $L_D = 5K = 15$, the minimum required interleaving delay satisfies

$$t_d = JMT > 0.5\lambda_c(L_D + 1)/v = 319 \text{ ms}.$$

Such a delay is quite large, especially for real-time voice applications, and the problem is exasperated by lower mobile station speeds. One possible solution is to use a code with a smaller decoding depth L_D. The other solution is to use improved interleaving techniques to reduce the interleaving delay. Once such interleaver is the convolutional interleaver discussed below. However, at low speeds the required interleaving delay may be excessive with any type of interleaver. Under such conditions, adaptive closed loop power control techniques are effective for combatting fading.

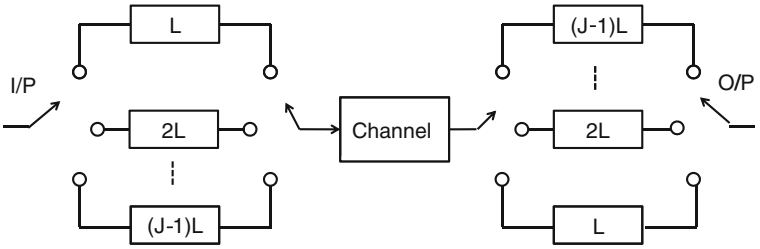


Fig. 8.16 Convolutional interleaver implemented with shift registers

8.5.0.3 Convolutional Interleaving

Convolutional interleavers have been proposed by Ramsey [222] and Forney [103], and here we discuss the structure proposed by Forney. Defining the parameter $M = LJ$, the interleaver is referred to as a (J,M) interleaver and has properties that are similar to a (J,M) block interleaver.

As shown in Fig. 8.16, the code symbols (bits) are input sequentially into the bank of J registers of increasing lengths. With each new code symbol (bit), the commutator switches to the next register, the code symbol (bit) is input to the register and the oldest code symbol (bit) in that register is shifted out to the channel. The input and output commutators of the interleaver operate in a synchronous fashion. The deinterleaver performs the reverse operation, and the deinterleaving commutators must be synchronized with the interleaving commutators. The most important properties of the convolutional interleaver are as follows:

1. The minimum separation at the interleaver output is J symbols (bits) for any two symbols that are separated by less than M symbols (bits) at the interleaver output.
2. Any burst of symbol (bit) errors of length $j \leq J$ results in single symbol (bit) errors at the deinterleaver output that are each separated by at least M symbols (bits).
3. A periodic sequence of single symbol (bit) errors that are spaced $M+1$ code symbols (bits) apart results in a single burst of symbol (bit) errors of length J at the deinterleaver output.
4. The end-to-end interleaving delay is equal to $M(J-1)$ code symbols (bits), and the memory requirements for both the interleaver and deinterleaver are equal to $M(J-1)/2$ code symbols (bits). This is half the delay and memory requirement of a (J,M) block interleaver.

The parameters J and M are chosen in the same manner as for a block interleaver, and the performance is very similar.

8.6 Code Performance on Interleaved Flat Fading Channels

8.6.1 TCM with Symbol Interleaving

Figure 8.17 is a block diagram of a coded communication system operating on a flat fading channel with symbol interleaving. The information sequence $\mathbf{a}$ is encoded and mapped onto a signal set to generate the modulated symbol sequence $\mathbf{s}$ using either a convolutional code or trellis code. The modulated symbol sequence is then time interleaved (or scrambled) and the resulting sequence $\check{\mathbf{s}}$ is transmitted over the channel. If a coherent detection is used, the receiver uses a correlator or matched filter detector as discussed in Sect. 5.2 to generate the sequence of received vectors $\mathbf{r} = \{\mathbf{r}_k\}$. With hard decision decoding, the received vector $\mathbf{r}_k$ at each epoch k is applied to a decision device to yield an estimate of the transmitted symbol sequence $\hat{\mathbf{s}}$. The estimated symbol sequence $\hat{\mathbf{s}}$ is then deinterleaved and input to the decoder. With soft decision decoding, on the other hand, the received vector $\mathbf{r}$ is deinterleaved and applied directly to the decoder.

For analytical purposes, an infinite interleaving depth is often assumed so that the deinterleaved sequence of complex channel gains $\mathbf{g} = \alpha \cdot e^{j\phi}$ constitutes a sequence of independent random variables. In this case the conditional density of $\mathbf{r}$ has the product from

$$p(\mathbf{r}|p(\mathbf{r}|\mathbf{g}\cdot\tilde{\mathbf{s}}) = \prod_k p(\mathbf{r}_k|g_k\tilde{\mathbf{s}}_k), \tag{8.127}$$

where $\mathbf{g}\cdot\tilde{\mathbf{s}}$ is the vector dot product of $\mathbf{g}$ and $\tilde{\mathbf{s}}$. Suppose that sequence $\tilde{\mathbf{s}}$ is transmitted and the sequence $\mathbf{r} = \mathbf{g}\cdot\tilde{\mathbf{s}} + \mathbf{n}$ is received. An ML receiver having perfect knowledge of $\mathbf{g}$ chooses the sequence $\hat{\mathbf{s}}$ that minimizes the squared Euclidean distance metric

$$\mu(\hat{\mathbf{s}}) = \|\mathbf{r} - \mathbf{g}\cdot\hat{\mathbf{s}}\|^2. \tag{8.128}$$

The pairwise error probability between the sequences $\tilde{\mathbf{s}}$ and $\hat{\mathbf{s}}$ has the Chernoff bound (see Appendix)

$$\mathrm{P}[\tilde{\mathbf{s}} \to \hat{\mathbf{s}}] \le \exp\left\{-\frac{\|\alpha\cdot(\tilde{\mathbf{s}} - \hat{\mathbf{s}})\|^2}{8N_\mathrm{o}}\right\}. \tag{8.129}$$

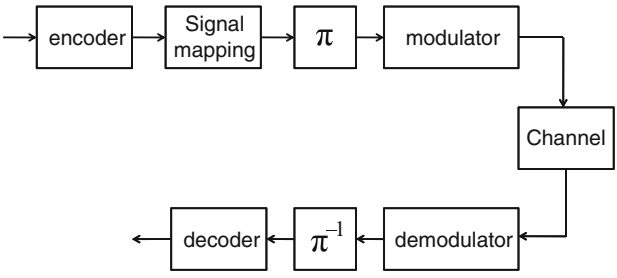


Fig. 8.17 TCM with symbol interleaving on a flat fading channel

If we assume the normalization $\mathrm{E}[|x_k|^2] = 1$ so that $E_\mathrm{s} = \frac{1}{2}\mathrm{E}[\|\mathbf{s}_k\|^2] = E_\mathrm{s}\mathrm{E}[|x_k|^2]$, that is, $\mathbf{s}_k = \sqrt{2E_\mathrm{s}}x_k$, then the Chernoff bound becomes

$$\mathrm{P}[\tilde{\mathbf{s}} \to \hat{\mathbf{s}}] \leq \exp\left\{-\frac{E_\mathrm{s}}{4N_\mathrm{o}}\|\alpha \cdot (\tilde{\mathbf{x}} - \hat{\mathbf{x}})\|^2\right\}, \tag{8.130}$$

where E_s is the symbol energy.

Here we assume that the channel is characterized by flat Ricean fading, so that the pdf of α_k is

$$p_{\breve{\alpha}}(x) = \frac{2x(1+K)}{\Omega_\mathrm{p}}\exp\left\{-K - \frac{(K+1)x^2}{\Omega_\mathrm{p}}\right\} I_0\left(2x\sqrt{\frac{K(K+1)}{\Omega_\mathrm{p}}}\right), \tag{8.131}$$

where $\Omega_\mathrm{p} = \mathrm{E}[\breve{\alpha}_k^2]$ is the average envelope power. Averaging (8.130) over the probability density function in (8.131) gives [76]

$$\mathrm{P}[\mathbf{s} \to \hat{\mathbf{s}}] \leq \prod_{i \in A} \frac{1+K}{1+K+\frac{\bar{\gamma}_\mathrm{s}}{4}|x_i - \hat{x}_i|^2}\exp\left\{-\frac{K\frac{\bar{\gamma}_\mathrm{s}}{4}|x_i - \hat{x}_i|^2}{1+K+\frac{\bar{\gamma}_\mathrm{s}}{4}|x_i - \hat{x}_i|^2}\right\}, \tag{8.132}$$

where $\bar{\gamma}_\mathrm{s} = \mathrm{E}[\alpha^2]E_\mathrm{s}/N_\mathrm{o}$ is the average received symbol energy-to-noise ratio, and $A = \{i | x_i \neq \hat{x}_i\}$. At sufficiently high $\bar{\gamma}_\mathrm{s}$, (8.132) simplifies to

$$\mathrm{P}[\mathbf{s} \to \hat{\mathbf{s}}] \ \leq \ \prod_{i \in A} \frac{4(1+K)}{\bar{\gamma}_\mathrm{s}|x_i - \hat{x}_i|^2}\mathrm{e}^{-K}. \tag{8.133}$$

It follows that the bound in (8.102) will be dominated by the error event path having the smallest number of elements in set A. Divsalar and Simon [76, 77] called this path the shortest error event path and defined its length as L_min. Based on previous arguments, the bit error probability can be approximated as

$$P_\mathrm{b} \ \simeq \ C\left(\frac{(1+K)\mathrm{e}^{-K}}{\bar{\gamma}_\mathrm{s}}\right)^{L_\mathrm{min}}, \ \ \bar{\gamma}_\mathrm{s} \gg K, \tag{8.134}$$

where C is a constant that depends on the distance structure of the code. Observe that P_b varies inversely with $(\bar{\gamma}_\mathrm{s})^{L_\mathrm{min}}$, yielding a diversity effect of order L_min. Wei [279] called L_min the MTD. The MTD dominates the performance of TCM on an interleaved flat fading channel, and the maximization of the MTD is the major design criterion for TCM on interleaved flat fading channels.

The pairwise error probability in (8.132) can be written in the form

$$\mathrm{P}[\mathbf{s} \to \hat{\mathbf{s}}] \ , \ \leq \ \mathrm{e}^{-\frac{\bar{\gamma}_\mathrm{s}}{4}d^2}, \tag{8.135}$$

where

$$d^2 = \sum_{i\in A} \frac{|x_i - \hat{x}_i|^2 K}{1 + K + \frac{\bar{\gamma}_s}{4}|x_i - \hat{x}_i|^2} + \left(\frac{\bar{\gamma}_s}{4}\right)^{-1} \ln\left(\frac{1 + K + \frac{\bar{\gamma}_s}{4}|x_i - \hat{x}_i|^2}{1 + K}\right)$$
$$= \sum_{i\in A} d_{1i}^2 + d_{2i}^2. \tag{8.136}$$

Two special cases are associated with (8.136), $K = \infty$ and $K = 0$. For $K = \infty$ (no fading),

$$d_{1i}^2 = |x_i - \hat{x}_i|^2, \qquad d_{2i}^2 = 0 \tag{8.137}$$

and, therefore, d^2 becomes the sum of the squared Euclidean distances over the error event path. Maximizing d^2 under this condition is the TCM design criterion for AWGN channels.

For $K = 0$ (Rayleigh fading),

$$d_{1i}^2 = 0, \qquad d_{2i}^2 = \left(\frac{\bar{\gamma}_s}{4}\right)^{-1} \ln\left(1 + \frac{\bar{\gamma}_s}{4}|x_i - \hat{x}_i|^2\right). \tag{8.138}$$

For reasonably large SNR, d^2 is the sum of the logarithms of the squared Euclidean distances, each weighted by $\bar{\gamma}_s$. In this case, the pairwise error probability is given by

$$\mathrm{P}[\mathbf{s} \to \hat{\mathbf{s}}] \le \left(\prod_{i\in A} \frac{\bar{\gamma}_s}{4}|x_i - \hat{x}_i|^2\right)^{-1}, \tag{8.139}$$

which is inversely proportional to the product of the squared Euclidean distances along the error event path. The MPSD between any two valid sequences,

$$\min_{\mathbf{x},\hat{\mathbf{x}}} \prod_{i\in A} |x_i - \hat{x}_i|^2 \tag{8.140}$$

is another design parameter for Rayleigh fading channels. For values of K between 0 and ∞, the equivalent squared Euclidean distance in (8.136) becomes a mixture of the two limiting cases given above.

If interleaving is not used, then the assumption that the fading is independent from symbol to symbol is no longer valid. If the fading is slow enough to be considered constant over the duration of the minimum distance error event path, then for coherent detection with the metric in (8.128), the bit error probability at high SNR is approximately,

$$P_b \simeq C_1 \mathrm{E}\left[\mathrm{e}^{-\gamma_s d_{\min}^2/4}\right], \tag{8.141}$$

where C_1 is a constant, $\gamma_s = \alpha^2 E_s/N_o$ is the received symbol energy-to-noise ratio, $d^2_{\min}$ is the minimum Euclidean distance of the code, and the expectation is with respect to the density in (8.131). Taking this average gives

$$P_b \simeq C_1 \frac{1+K}{1+K+d^2_{\min}\frac{\bar{\gamma}_s}{4}|x_i-\hat{x}_i|^2} \exp\left\{-\frac{Kd^2_{\min}\frac{\bar{\gamma}_s}{4}|x_i-\hat{x}_i|^2}{1+K+d^2_{\min}\frac{\bar{\gamma}_s}{4}|x_i-\hat{x}_i|^2}\right\}, \tag{8.142}$$

which can be approximated at large $\bar{\gamma}_s$ by

$$P_b \simeq 4C_1 \frac{1+K}{d^2_{\min}\bar{\gamma}_s} e^{-K}. \tag{8.143}$$

Observe that without interleaving, P_b is asymptotically inverse linear with $\bar{\gamma}_s$, independent of the trellis code. If follows that interleaving is required to achieve diversity with TCM on a flat fading channel.

8.6.1.1 Design Rules for Symbol Interleaved TCM on Flat Fading Channels

According to the previous section, when symbol interleaved TCM is used on Ricean fading channel, the design of the code for optimum performance is guided by the MTD of the code. For Rayleigh fading channels, the design of the code is also guided by the MPSD of the code. The minimum Euclidean distance, which is the principal design criterion for TCM AWGN channels, plays a less significant role on Ricean fading channels as the K factor decreases, and no role for Rayleigh fading channels ($K = 0$). A third design criterion is to minimize the decoding depth of the code.

The design of trellis codes for interleaved flat fading channels can be based on Ungerboeck's principle of mapping by set partitioning, but now the partitioning is done to maximize the MTD and MPSD of the code. This can be accomplished by maximizing the intra-subset MTD and MPSD, but it should be pointed out that large MTD and MPSD can be sometimes achieved even if the partitioning is done to maximize the minimum Euclidean distance.

In general, the following guidelines are followed when designing trellis codes for symbol interleaved flat fading channels:

1. All signals occur with equal frequency and with regularity and symmetry.
2. Transitions originating from the even and odd numbered states are assigned signals from the first and second subsets, respectively, of the first partitioning level.
3. Whenever possible, the transitions joining in the same state receive signals from either the first or second subset of the first partitioning level.
4. Parallel transitions receive signals from the same subset of the finest partitioning level.

5. The state transitions originating from each current state and going to even-numbered next states are assigned signals from subsets whose inter-subset MTD and MPSD are maximized. The same applies for the transition originating from each current state and going to odd-numbered next states.

The first four rules are similar to those suggested by Ungerboeck [258], but now the subsets used may be different. The fifth rule is used to reduce the decoding depth of the code. Wei [279] developed several codes based on minimizing the decoding depth of a code. He defined two minimum decoding depths (MDD1, MDD2) to characterize a code. MDD1+1 is defined as the length (in symbols) of the longest valid sequence of signal points, say $\tilde{\mathbf{x}} = \{\tilde{x}_k\}$, which originates from the same state as another valid sequence $\mathbf{x} = \{x_k\}$ and merges into the same last state as $\mathbf{x}$ and whose Hamming distance from $\mathbf{x}$ is the same as the MTD of the code. Note that the performance of a code is mainly governed by the pairs of sequences which determine the MTD of the code. Each such pair of sequences differ in at most MDD1+1 successive symbols. The farther these symbols are separated, the better the performance of the code. Hence, to benefit from the MTD of the code, the symbol interleaver should separate the output symbols corresponding to each sequence of MDD1+1 consecutive input symbols as far apart as possible.

MDD2 is defined as the length of the longest unmerged valid sequence of signal points, say $\check{\mathbf{x}}$, which originates from the same state as another valid sequence, say $\mathbf{x}$, and whose Hamming distance from $\mathbf{x}$ is not greater than the MTD of the code. In case the Hamming distance between the two sequences is equal to the MTD of the code, the squared product distance between the two sequences must be less than the MPSD of the code. Since MDD2 is greater than MDD1, the decoding depth should be at least equal to MDD2 to realize the MTD and MPSD of a code. It suffices if the decoding depth is few symbols longer than MDD2. Finally, to benefit from both the MTD and MPSD of a code, the symbol interleaver should separate the output symbols corresponding to each sequence of MDD2+1 consecutive input symbols as far apart as possible.

8.6.2 Bit Interleaved Coded Modulation

Bit interleaved coded modulation (BICM) interleaves the code bits at the output of a convolutional encoder before symbol mapping, along with an appropriately defined soft-decision metric as an input to the Viterbi decoder. The fundamental idea of BICM is to separate the encoder and modulator to make the code diversity equal to the number of distinct *bits* rather than the number of distinct modulated *symbols* along any error event. Such an approach cannot achieve optimum Euclidean distance. However, it will yield better performance on fading channels than TCM with symbol interleaving due to the increased code diversity. The idea is to transform the channel that is generated by a multilevel constellation of size $M = 2^k$ into a parallel collection of k channels that each carry one binary symbol from the signal

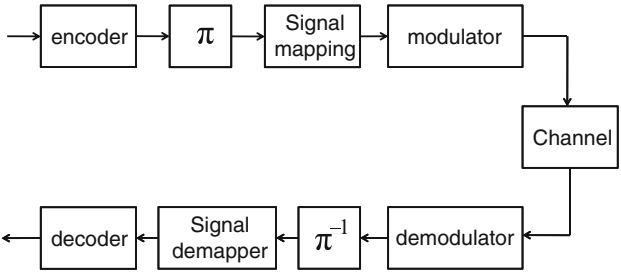


Fig. 8.18 Bit interleaved coded modulation (BICM) on a flat fading channel

mapping. To make the k channels independent, they are interleaved before the signal mapping. The decoder in a BICM system must reflect the fact that we are interleaving the bits before signal mapping. The basic structure of a BICM system is shown in Fig. 8.18.

Suppose that we transmit the length-n codeword

$$\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_n) \tag{8.144}$$

and we received the sequence

$$\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n) \tag{8.145}$$

at the output of the bank of receiver matched filters or correlators. With TCM on a memoryless channel, we perform decoding using the log-likelihood metric

$$\mu(\hat{\mathbf{s}}) = \log\{p(\mathbf{r}|\alpha \cdot \hat{\mathbf{s}})\} = \log\left\{\prod_{k=1}^{n} p(\mathbf{r}_k|\alpha_k\hat{\mathbf{s}}_k)\right\} = \sum_{k=1}^{n} \log\{p(\mathbf{r}_k|\alpha_k\hat{\mathbf{s}}_k)\}. \tag{8.146}$$

When BICM is used, on the other hand, instead of the *symbol* metric $p(\mathbf{r}_k|\alpha_k\hat{\mathbf{s}}_k)$, we must use the *bit* metric

$$\mu(b) = \log\left\{\sum_{x_i \in \mathscr{X}(b,j)} p(\mathbf{r}_i|\alpha_i\mathbf{s}_i)\right\}, \quad b = 0,1, \quad j = 1,2,\ldots,k = \log_2 M, \tag{8.147}$$

where $\mathscr{X}(b,j)$ denotes the subset of the size $M = 2^k$ point signal constellation $\mathscr{X}$ that has bit b in position j of its signal mapping label. The computation of this metric may be complicated due to calculation of the logarithm. A suboptimum metric can be defined based on

$$\log\left\{\sum_j z_j\right\} \approx \max_j \log\{z_j\}, \tag{8.148}$$

which yields

$$\mu(b) = \max_{x_i \in \mathcal{X}(b,j)} \log\{p(\mathbf{r}_i | \alpha_i \mathbf{s}_i)\}. \tag{8.149}$$

Except for the maximization operation, this metric is the same as the TCM symbol metric. Finally, the performance of BICM strongly depends on the signal mapping that is used. Gray mapping is known to perform better than mapping by set partitioning.

8.7 Performance of Space-Time Codes

The orthogonal space-time block codes described in Sect. 8.1.2 are just one possible construction for the space-time code matrix. We now discuss some useful criteria for designing more general $p \times L_\mathrm{t}$ space-time codes for quasi-static fading channels, where the channel remains static over p time slots; the rank and determinant criteria. These criteria are derived under the condition of independent fades, where the complex channel gains $g_{i,j}$ are all independent.

To derive the rank and determinant criteria for space-time codes, we first need to derive the pairwise error probability between any two space-time codewords. Suppose that a space time codeword $C = [\tilde{\mathbf{s}}_{(\mathrm{t}),i}]_{p \times L_\mathrm{t}}$ is transmitted over an $L_\mathrm{t} \times L_\mathrm{r}$ MIMO channel in p time slots, where the modulated symbol vectors are chosen from the symbol alphabet of a linear modulation scheme, such as QAM and PSK. By observing the noisy received signal vectors in (8.33), the receiver decides which space-time codeword was transmitted using the maximum likelihood metric in (8.34). Suppose that the receiver erroneously decides in favor of another space-time codeword $\hat{C} = [\hat{\mathbf{s}}_{(\mathrm{t}),i}]_{p \times L_\mathrm{t}}$. This is the pairwise error probability $\mathrm{P}[C \to \hat{C}]$. Assuming ideal channel state information, the pairwise error probability can be written as

$$\begin{aligned}
\mathrm{P}[C \to \hat{C}] &= \mathrm{P}\left[\sum_{j=1}^{L_\mathrm{r}} \sum_{t=1}^{p} \left\| \tilde{\mathbf{r}}_{(\mathrm{t}),j} - \sum_{i=1}^{L_\mathrm{t}} g_{i,j} \tilde{\mathbf{s}}_{(\mathrm{t}),i} \right\|^2 > \sum_{j=1}^{L_\mathrm{r}} \sum_{t=1}^{p} \left\| \tilde{\mathbf{r}}_{(\mathrm{t}),j} - \sum_{i=1}^{L_\mathrm{t}} g_{i,j} \hat{\mathbf{s}}_{(\mathrm{t}),i} \right\|^2 \right] \\
&= \mathrm{P}\left[\sum_{j=1}^{L_\mathrm{r}} \sum_{t=1}^{p} 2\mathrm{Re}\left\{ \tilde{\mathbf{n}}^H_{(\mathrm{t}),j} \sum_{i=1}^{L_\mathrm{t}} g_{i,j} (\hat{\mathbf{s}}_{(\mathrm{t}),i} - \tilde{\mathbf{s}}_{(\mathrm{t}),i}) \right\} \right. \\
&\qquad \left. > \sum_{j=1}^{L_\mathrm{r}} \sum_{t=1}^{p} \left\| \sum_{i=1}^{L_\mathrm{t}} g_{i,j} (\hat{\mathbf{s}}_{(\mathrm{t}),i} - \tilde{\mathbf{s}}_{(\mathrm{t}),i}) \right\|^2 \right] \\
&= Q\left(\sqrt{\frac{\Delta^2 E_\mathrm{s}}{4N_\mathrm{o}}} \right),
\end{aligned} \tag{8.150}$$

where

$$\Delta^2 = \frac{1}{E_\mathrm{s}} \sum_{j=1}^{L_\mathrm{r}} \sum_{t=1}^{p} \left\| \sum_{i=1}^{L_\mathrm{t}} g_{i,j} (\tilde{\mathbf{s}}_{(\mathrm{t}),i} - \hat{\mathbf{s}}_{(\mathrm{t}),i}) \right\|^2 . \tag{8.151}$$

Applying a Chernoff bound on the Gaussian Q function (see Appendix) gives

$$\mathrm{P}[C \to \hat{C}] \leq \frac{1}{2} \exp\left\{ -\frac{\Delta^2 E_\mathrm{s}}{4N_\mathrm{o}} \right\}. \tag{8.152}$$

We can rewrite (8.151) as

$$\Delta^2 = \frac{1}{E_\mathrm{s}} \sum_{j=1}^{L_\mathrm{r}} \sum_{i=1}^{L_\mathrm{t}} \sum_{i'=1}^{L_\mathrm{t}} g_{i,j} g^*_{i',j} \sum_{t=1}^{p} (\tilde{\mathbf{s}}_{(\mathrm{t}),i} - \hat{\mathbf{s}}_{(\mathrm{t}),i})(\tilde{\mathbf{s}}_{(\mathrm{t}),i'} - \hat{\mathbf{s}}_{(\mathrm{t}),i'})^\mathrm{H}. \tag{8.153}$$

Now let $\mathbf{g}_j = (g_{1,j} \ldots, g_{L_\mathrm{t},j})$. After some manipulations, we can rewrite (8.153) as

$$\Delta^2 = \sum_{j=1}^{L_\mathrm{r}} \mathbf{g}_j \mathbf{A}(C, \hat{C}) \mathbf{g}_j^\mathrm{H}, \tag{8.154}$$

where $\mathbf{A}(C, \hat{C}) = [A_{p,q}]_{L_\mathrm{t} \times L_\mathrm{t}} = \frac{1}{E_\mathrm{s}} \mathbf{s}_p \mathbf{s}_q^\mathrm{T}$, and where

$$\mathbf{s}_p = (\tilde{\mathbf{s}}_{(\mathrm{p}),1} - \hat{\mathbf{s}}_{(\mathrm{p}),1}, \ldots, \tilde{\mathbf{s}}_{(\mathrm{p}),t} - \hat{\mathbf{s}}_{(\mathrm{p}),t}), \tag{8.155}$$

$$\mathbf{s}_q = (\tilde{\mathbf{s}}_{(\mathrm{q}),1} - \hat{\mathbf{s}}_{(\mathrm{q}),1}, \ldots, \tilde{\mathbf{s}}_{(\mathrm{q}),t} - \hat{\mathbf{s}}_{(\mathrm{q}),t}). \tag{8.156}$$

Hence, the Chernoff bound in (8.152) becomes

$$\mathrm{P}[C \to \hat{C}] \leq \frac{1}{2} \prod_{j=1}^{L_\mathrm{r}} \exp\left\{ -\frac{\mathbf{g}_j \mathbf{A}(C, \hat{C}) \mathbf{g}_j^\mathrm{H} E_\mathrm{s}}{4N_\mathrm{o}} \right\}. \tag{8.157}$$

Since $\mathbf{A}(C, \hat{C}) = \mathbf{A}^\mathrm{H}(C, \hat{C})$, there exists a unitary matrix $\mathbf{V}$, such that $\mathbf{V}\mathbf{V}^\mathrm{H} = \mathbf{I}_{L_\mathrm{t} \times L_\mathrm{t}}$, and a real diagonal matrix $\mathbf{D}$ such that $\mathbf{V}\mathbf{A}(C, \hat{C})\mathbf{V}^\mathrm{H} = \mathbf{D}$. The rows $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{L_\mathrm{t}}$ of $\mathbf{V}$ are a complete complex orthonormal basis for an L_t-dimensional complex vector space. Also, the diagonal elements of $\mathbf{D}$ are the eigenvalues, $\lambda_i, i = 1, \ldots, L_\mathrm{t}$ of $\mathbf{A}(C, \hat{C})$ that may include repeated eigenvalues. By construction, the matrix

$$\mathbf{B}(C, \hat{C}) = \frac{1}{\sqrt{E_\mathrm{s}}} \begin{bmatrix} \tilde{\mathbf{s}}_{(1),1} - \hat{\mathbf{s}}_{(1),1} & \tilde{\mathbf{s}}_{(2),1} - \hat{\mathbf{s}}_{(2),1} & \ldots & \tilde{\mathbf{s}}_{(\mathrm{p}),1} - \hat{\mathbf{s}}_{(\mathrm{p}),1} \\ \tilde{\mathbf{s}}_{(1),2} - \hat{\mathbf{s}}_{(1),2} & \tilde{\mathbf{s}}_{(2),2} - \hat{\mathbf{s}}_{(2),2} & \ldots & \tilde{\mathbf{s}}_{(\mathrm{p}),2} - \hat{\mathbf{s}}_{(\mathrm{p}),2} \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{s}}_{(1),L_\mathrm{t}} - \hat{\mathbf{s}}_{(1),L_\mathrm{t}} & \tilde{\mathbf{s}}_{(2),L_\mathrm{t}} - \hat{\mathbf{s}}_{(2),L_\mathrm{t}} & \ldots & \tilde{\mathbf{s}}_{(\mathrm{p}),L_\mathrm{t}} - \hat{\mathbf{s}}_{(\mathrm{p}),L_\mathrm{t}} \end{bmatrix} \tag{8.158}$$

is the matrix square root of $\mathbf{A}(C,\hat{C})$ that is, $\mathbf{A}(C,\hat{C}) = \mathbf{B}(C,\hat{C})\mathbf{B}(C,\hat{C})^{\mathrm{H}}$. Therefore, the eigenvalues of $\mathbf{A}(C,\hat{C})$ are all nonnegative real numbers. Next, we express Δ^2 as a function of the eigenvalues of the matrix $\mathbf{A}(C,\hat{C})$. Let $(\beta_{1,j},\beta_{2,j},\ldots,\beta_{L_t,j}) = \mathbf{g}_j\mathbf{V}^{\mathrm{H}}$. Then

$$\mathbf{g}_j\mathbf{A}(C,\hat{C})\mathbf{g}_j^{\mathrm{H}} = \mathbf{g}_j\mathbf{V}^{\mathrm{H}}\mathbf{D}\mathbf{V}\mathbf{g}_j^{\mathrm{H}} = \sum_{i=1}^{L_t}\lambda_i|\beta_{i,j}|^2. \tag{8.159}$$

Next recall that $g_{i,j}$ are all complex Gaussian random variables. Since $\mathbf{V}$ is unitary, the $\beta_{i,j}$ are independent complex Gaussian random variables with a normalized variance $1/2$ per dimension. Assuming that the $g_{i,j}$ have zero mean, the $|\beta_{i,j}|$ are independent Rayleigh random variables with the density function

$$p_{\beta_{i,j}}(x) = 2x\mathrm{e}^{x^2}\ ,\ \ x \geq 0. \tag{8.160}$$

Thus to obtain Chernoff bound on the pairwise error probability, we average

$$\prod_{j=1}^{L_r}\exp\left\{-\frac{E_s}{4N_o}\sum_{i=1}^{L_t}\lambda_i|\beta_{i,j}|^2\right\} \tag{8.161}$$

over the distribution of the $|\beta i,j|$ in (8.160). This leads to the bound on the pairwise error probability

$$\mathrm{P}[C\rightarrow\hat{C}] \leq \frac{1}{2}\left(\frac{1}{\prod_{i=1}^{L_t}(1+\lambda_i E_s/4N_o)}\right)^{L_r}. \tag{8.162}$$

Let r denote the rank of the matrix $\mathbf{A}(C,\hat{C})$, where $1 \leq r \leq L_t$. Then for sufficiently high E_s/N_o such that $1+\lambda_i E_s/4N_o \approx \lambda_i E_s/4N_o$, it follows that the pairwise error probability has the upper bound

$$\mathrm{P}[C\rightarrow\hat{C}] \leq \frac{1}{2}\left[\left(\prod_{i=1}^{r}\lambda_i\right)^{1/r}\right]^{rL_r}\left(\frac{E_s}{4N_o}\right)^{rL_r}. \tag{8.163}$$

Therefore, this space-time codeword pair provides diversity of order rL_r and a coding gain of $(\lambda_1\cdot\lambda_2\ldots\lambda_r)^{1/r}$ which is equal to the geometric mean of the nonzero eigenvalues. The coding gain is measured with respect to an uncoded system with maximal ratio combining that is operating with the same diversity order. Finally, we note that the ranks of $\mathbf{A}(C,\hat{C})$ and $\mathbf{B}(C,\hat{C})$ are equal.

The above development leads to the following two simple design criteria for space-time codes on quasi-static Rayleigh fading channels:

Rank Criterion: To achieve the maximum diversity order L_tL_r, the matrix $\mathbf{B}(C,\hat{C})$ must have full rank L_t for all pairs of distinct codewords. If the minimum rank of $\mathbf{B}(C,\hat{C})$ is equal to r, then a diversity order of rL_r is achieved.

Determinant Criterion: If a diversity order of $L_t L_r$ is achieved, then the coding gain is optimized by maximizing the minimum determinant of $\mathbf{A}(C,\hat{C})$ over all pairs of distinct codewords. If a diversity order of rL_r is achieved, then the minimum value of $(\lambda_1 \cdot \lambda_2 \ldots \lambda_r)^{1/r}$ should be maximized over all pairs of distinct codewords.

8.7.1 Space-Time Trellis Codes

8.7.1.1 Encoder Description

A space-time trellis code (STTC) maps the information bit stream into L_t streams of modulated symbols that are transmitted simultaneously from L_t transmit antennas. The total transmission power is divided equally among the L_t transmit antennas. STTCs can be designed according to the rank and determinant criteria as discussed in Sect. 8.7, although other criteria exist as well.

SSTCs are an extension of TCM to multi-antenna systems. The encoder for a STTC is similar to that for a trellis code, except that the encoder must begin and end each frame in the all-zeroes state. The STTC encoder can be described in terms of its generator sequences or generator polynomials. For a system with L_t transmit antennas, the ith input sequence $\mathbf{a}^{(i)},\ i = 1,\ldots,L_t$ has the polynomial representation

$$\mathbf{a}^{(i)}(D) = \sum_{k=0}^{\infty} a_{i,k} D^k \ \ , i = 1,\ldots,L_t, \tag{8.164}$$

where the $a_{i,k} \in \{0,1\}$ are binary input data bits. The generator polynomial corresponding to the ith input sequence, where $i = 1,\ldots,L_t$, and the jth transmit antenna, where $j = 1,\ldots,L_t$, can be written as

$$\mathbf{g}_i^{(j)}(D) = \sum_{k=0}^{K-1} g_{i,k}^{(j)} D^k, \tag{8.165}$$

where the $g_{i,k}^{(j)}$ are nonbinary coefficients chosen from the set $\{0,1,\ldots,M-1\}$, where M is the size of the signal constellation, and K is the overall constraint length of the encoder (similar to convolutional codes). Very often the generator sequences for STTCs are tabulated in the following format:

$$\begin{aligned}\mathbf{g}_i = [&(g_{i,0}^{(1)}, g_{i,0}^{(2)}, \ldots, g_{i,0}^{(L_t)}), (g_{i,1}^{(1)}, g_{i,1}^{(2)}, \ldots, g_{i,1}^{(L_t)}), \ldots \\ &\ldots (g_{i,K-1}^{(1)}, g_{i,K-1}^{(2)}, \ldots, g_{i,K-1}^{(L_t)})],\ i = 1,\ldots,L_t.\end{aligned} \tag{8.166}$$

A simple 2-transmit antenna, 16-state, STTC encoder is shown in Fig. 8.19. For the constraint length $K = 3$ encoder shown in Fig. 8.19, we might have

$$\mathbf{g}_1 = [(0,1),(1,2),(2,0)], \qquad \mathbf{g}_2 = [(0,2),(2,0),(0,2)], \tag{8.167}$$

which just happens to satisfy the rank and determinant criteria in Sect. 8.7.

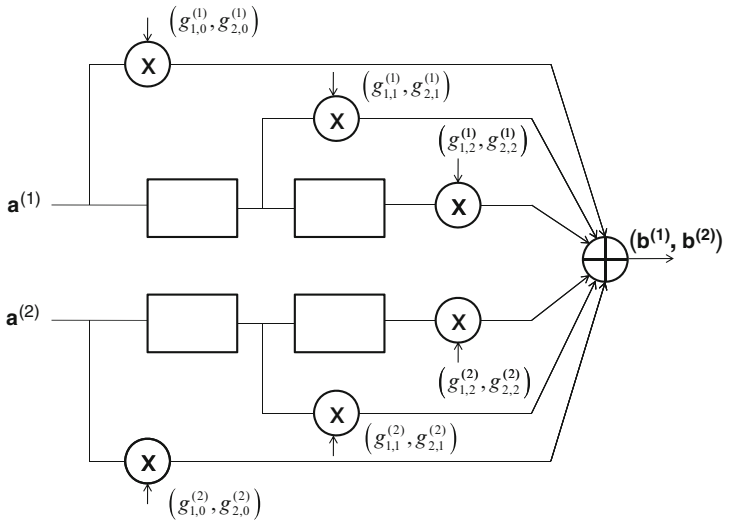


Fig. 8.19 Encoder for a 2-transmit antenna, 16-state, STTC

The multiplier outputs, as illustrated in Fig. 8.19, are summed with modulo-M addition. Hence, the encoded sequence that is transmitted from antenna j is given by

$$\mathbf{b}^{(j)}(D) = \sum_{i=1}^{L_t} \mathbf{a}^{(i)}(D)\mathbf{g}_i^{(j)}(D) \ \text{ modulo } M \ , \ \ i = 1, \ldots, L_t. \tag{8.168}$$

The above expression can be expressed in the matrix form

$$\left[\mathbf{b}^{(i)}(D), \ldots, \mathbf{b}^{(L_t)}(D)\right] = \left[\mathbf{a}^{(1)}(D), \ldots, \mathbf{a}^{(L_t)}(D)\right] \begin{bmatrix} \mathbf{g}_1^{(1)}(D), \ \ldots, \ \mathbf{g}_1^{(L_t)}(D) \\ \vdots \qquad\qquad \vdots \\ \mathbf{g}_{L_t}^{(1)}(D), \ \ldots, \ \mathbf{g}_{L_t}^{(L_t)}(D) \end{bmatrix}, \tag{8.169}$$

where

$$\mathbf{G}(D) = \begin{bmatrix} \mathbf{g}_1^{(1)}(D), \ \ldots, \ \mathbf{g}_1^{(L_t)}(D) \\ \vdots \qquad\qquad \vdots \\ \mathbf{g}_{L_t}^{(1)}(D), \ \ldots, \ \mathbf{g}_{L_t}^{(L_t)}(D) \end{bmatrix} \tag{8.170}$$

is the generator matrix of the STTC. The elements of the code symbol sequences $\mathbf{b}_i = \{b_{i,k}\}, i = 1, \ldots, L_t$ at the output of the encoder are mapped onto symbols chosen from an M-ary signal constellation, such as M-PSK or M-QAM according to a one-to-one mapping. This yields the sequence of modulated symbols $\mathbf{s}_i = \{s_{i,k}\}, i = 1, \ldots, L_t$ that are transmitted from the L_t transmit antennas.

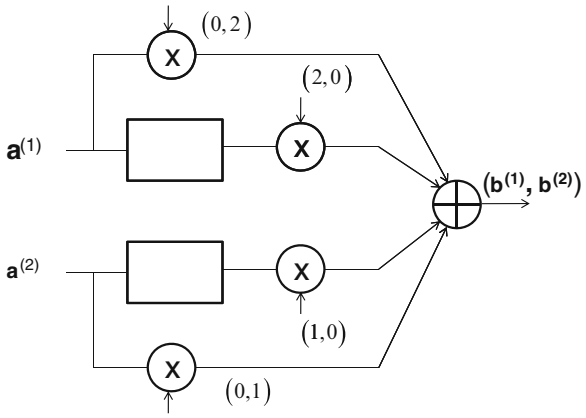


Fig. 8.20 Encoder for a 2-transmit antenna, 4-state, 4-PSK STTC

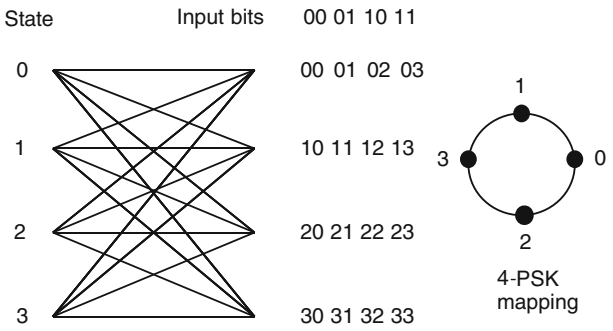


Fig. 8.21 STTC trellis for the 2-transmit antenna, 4-state, 4-PSK STTC in Fig. 8.20

8.7.1.2 STTC Code Trellis and the Viterbi Algorithm

Similar to convolutional and trellis codes, the STTC encoder can be described by a state diagram and trellis diagram. As an example, consider the 2-transmit antenna, 4-state, 4-PSK STTC shown in Fig. 8.20, where the generators are

$$\mathbf{g}_1 = [(0,1),(1,0)], \qquad \mathbf{g}_2 = [(0,2),(2,0)]. \tag{8.171}$$

The corresponding trellis diagram for this code is shown in Fig. 8.21. The branch label $b_1 b_2$ means that symbol b_1 is transmitted from the first antenna, while the symbol b_2 from the second antenna. The symbol pairs in each row label the branch transitions out of a given state, in order, from top to bottom. The symbols $b_1, b_2 \in \{0,1,2,3\}$ are mapped onto a 4-PSK signal constellation using the mapping shown in Fig. 8.21, which can be expressed mathematically as $s_i = \sqrt{2E_h}(j)^{b_i}$. The encoder is required to begin and end each frame in the zero-state. Beginning at State 0, if the two input bits are 11, then the encoder outputs symbol 0 on Antenna 1 and symbol 3 on Antenna 2, and transitions to State 3.

Given their trellis structure, it is apparent that STTCs can be efficiently decoded using the Viterbi algorithm with an appropriately defined branch metric. For quasi-static flat fading channels, the branch metric is

$$\mu\left(\rho_k^{(i)} \to \rho_{k+1}^{(j)}\right) = -\sum_{d=1}^{L_\mathrm{r}} \left\| \tilde{\mathbf{r}}_{(\mathrm{t}),j} - \sum_{i=1}^{L_\mathrm{t}} g_{i,j} \hat{\mathbf{s}}_{(\mathrm{t}),i}\left(\rho_k^{(i)} \to \rho_{k+1}^{(j)}\right) \right\|^2, \quad (8.172)$$

where $\hat{\mathbf{s}}_{(\mathrm{t}),i}(\rho_k^{(i)} \to \rho_{k+1}^{(j)})$ is a symbol that is uniquely determined by the state transition $\rho_k^{(i)} \to \rho_{k+1}^{(j)}$.

8.8 Turbo Codes

The principle of turbo coding or concatenated coding is to construct long random-like codes that have a structure that permits practical decoding [26]. Turbo codes are interleaved concatenated codes that are constructed from simple component codes and pseudo-random interleavers. The interleaver makes the code appear random. Since the component codes are easy to decode, the overall code can be decoded by iteratively decoding the component codes. There are two basic types of turbo codes depending on the type of concatenation, namely parallel concatenated codes and serial concatenated codes. The component codes can be either convolutional codes or block codes that are realized in systematic form. Here we just consider convolutional component codes. PCCCs use RSC component codes. SCCCs use a recursive or nonrecursive convolutional outer code along with a recursive convolutional inner code.

8.8.1 PCCC Encoder

Figure 8.22 shows a PCCC encoder structure which is a parallel concatenation of two RSC component codes.[3] The component codes must be recursive for reasons that we will see later. Notice that both the systematic and parity bits of the first encoder are used, while only the parity bits of the second encoder are used. If the component codes have rates $R_\mathrm{c}^{(1)} = k/n_1$ and $R_\mathrm{c}^{(2)} = k/n_2$, then the PCCC has code rate

$$R_\mathrm{T} = \frac{R_\mathrm{c}^{(1)} R_\mathrm{c}^{(2)}}{R_\mathrm{c}^{(1)} + R_\mathrm{c}^{(2)} - R_\mathrm{c}^{(1)} R_\mathrm{c}^{(2)}} = \frac{k}{n_1 + n_2 - k}. \quad (8.173)$$

[3]The parallel concatenation of more than two component codes is possible, but we will consider only two component codes for simplicity.

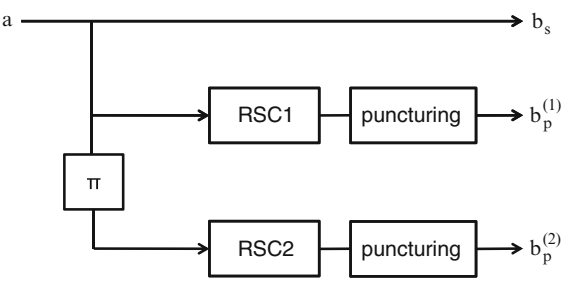


Fig. 8.22 PCCC encoder

The input data sequence **a** is first encoded by RSC1. Suppose, for example, that RSC1 and RSC2 happen to be identical rate-1/2 codes. Then feedforward and feedback generator polynomials of RSC1 and RSC2 are $\mathbf{g}^{(2)}(D) = g_0^{(2)} + g_1^{(2)}D + \ldots + g_\nu^{(2)}D^\nu$ and $\mathbf{g}^{(1)}(D) = g_0^{(1)} + g_1^{(1)}D + \ldots + g_\nu^{(1)}D^\nu$, respectively, where ν is the encoder memory. The outputs of RSC1 are the systematic component $\mathbf{b}_\mathrm{s} = \{b_{s_k}\}$ and the parity component $\mathbf{b}_\mathrm{p}^{(1)} = \{b_{p_k}^{(1)}\}$ defined by

$$b_{s_k} = a_k,$$

$$b_{p_k}^{(1)} = \sum_{i=0}^{\nu} g_i^{(1)} d_{k-i},$$

where

$$d_k = a_k \oplus \sum_{i=1}^{\nu} g_i^{(2)} d_{k-i}. \tag{8.174}$$

The data sequence **a** is interleaved by a turbo interleaver π of size $N = kN'$ into the sequence $\tilde{\mathbf{a}}$ and encoded using RSC2 to produce the parity sequence $\mathbf{b}_\mathrm{p}^{(2)}$. The interleaving operation can be defined by a mapping $i \to \pi(i)$ of the input bit position i to output bit position $\pi(i)$. For example, the interleaver might perform the mapping

$$\{0,1,2,3,\ldots,N-1\}_N \to \{23,12,6,7,\ldots,1\}_N.$$

For turbo codes the choice of interleaver is crucial. Interleavers that have a structure, such as block interleavers, are not suitable. Usually, random interleavers are used, where the interleaving mapping is randomly generated. In other cases, an S-random interleaver is used, where interleaver inputs that are separated by less than S positions, $|i-j| < S$, are interleaved into interleaver outputs that are separated by at least S positions, $|\pi(i) - \pi(j)| \geq S$.

A PCCC code word $\mathbf{b} = (\mathbf{b}_\mathrm{s}, \mathbf{b}_\mathrm{p}^{(1)}, \mathbf{b}_\mathrm{p}^{(2)})$ is formed by the parallel concatenation (or multiplexing) of the systematic component and the parity sequences. If higher code rates are desired, then the parity outputs of the RSC component encoders can be punctured according to a puncturing pattern. For example, if RSC1 and RSC2

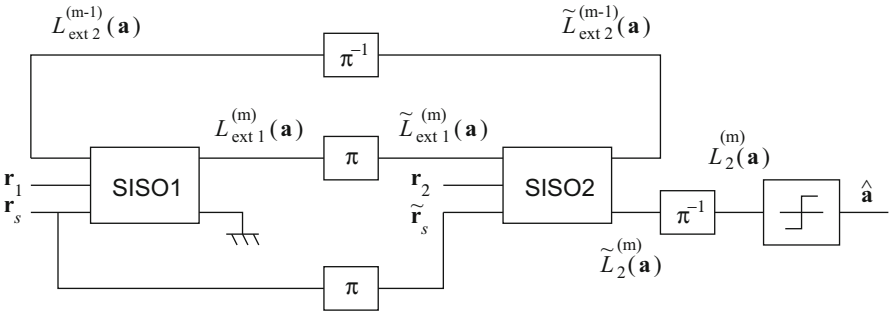


Fig. 8.23 PCCC decoder

in Fig. 8.22 are rate-1/2 codes and no puncturing is used, then the code rate will be $R_{\text{T}} = 1/3$. However, suppose that puncturing is used with the pattern

$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{8.175}$$

where, a "1" in the puncturing pattern means that the code bit is transmitted, while a "0" means that the code bit is not transmitted. This particular puncturing pattern means that the parity bits from RSC1 and RSC2 are transmitted in an alternate fashion, and overall code rate is increased to $R_{\text{T}} = 1/2$. Finally, tail bits are typically added to the data sequence to terminate RSC1 in the all-zeroes state while the trellis of RSC2 is left "open."

8.8.2 PCCC Decoder

The turbo decoder is an iterative structure consisting of several identical stages, each consisting of two soft-input soft-output (SISO) decoding units for the case of two constituent codes. Suppose that the codeword $\mathbf{b} = (\mathbf{b}_{\text{s}}, \mathbf{b}_{\text{p}}^{(1)}, \mathbf{b}_{\text{p}}^{(2)})$ is transmitted and the received vector is $\tilde{\mathbf{r}} = (\tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)})$. The decoder structure for PCCCs is shown in Fig. 8.23. The SISO modules generate APPs

$$\text{P}(a_k | \tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)}) \tag{8.176}$$

or, for binary codes, aposteriori LLRs

$$L(a_k | \tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)}) = \log \left\{ \frac{\text{P}(a_k = +1 | \tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)})}{\text{P}(a_k = -1 | \tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)})} \right\} \tag{8.177}$$

of each information bit a_k based on the received vector $\tilde{\mathbf{r}} = (\tilde{\mathbf{r}}_{\text{s}}, \tilde{\mathbf{r}}_{\text{p}}^{(1)}, \tilde{\mathbf{r}}_{\text{p}}^{(2)})$ and the *extrinsic* information passed between the two SISO modules.

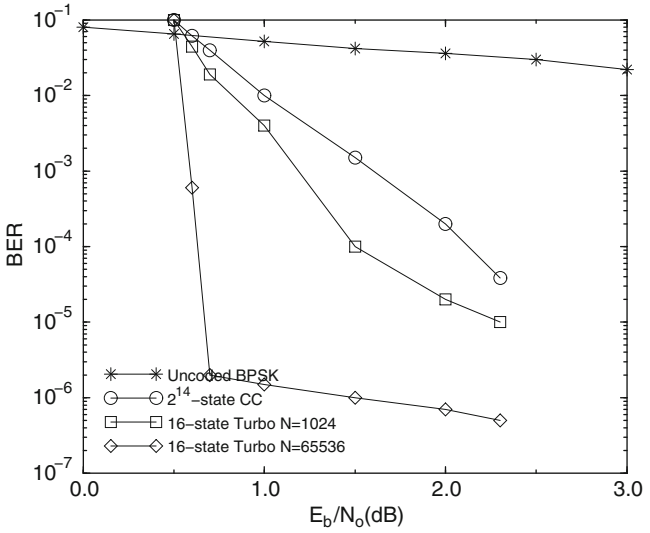


Fig. 8.24 Typical PCCC performance on an AWGN channel

The iterative decoding operation of parallel turbo codes can be explained as follows, using LLRs as an example. At the mth iteration, $m \geq 1$, the LLRs generated by the SISO decoders for data bit a_k are

$$L_1^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)}) = L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}}) + L_{\mathrm{ext2}}^{(m-1)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)}) + L_{\mathrm{ext1}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)}), \tag{8.178}$$

$$L_2^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)}) = L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}}) + L_{\mathrm{ext1}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)}) + L_{\mathrm{ext2}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)}), \tag{8.179}$$

where $L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}})$ is the aposteriori LLR due to the systematic component, and $L_{\mathrm{ext1}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)})$ and $L_{\mathrm{ext2}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)})$ are the extrinsic information for each bit generated at the mth decoding stage by SISO1 and SISO2, respectively, and can be expressed as

$$L_{\mathrm{ext1}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)}) = f(L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}}), L_{\mathrm{ext2}}^{(m-1)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)})), \tag{8.180}$$

$$L_{\mathrm{ext2}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)}) = f(L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}}), L_{\mathrm{ext1}}^{(m)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(1)})), \tag{8.181}$$

where $f(\,\cdot\,)$ denotes the SISO decoding unit. The extrinsic information can be calculated using the BCJR algorithm as discussed in Sect. 8.2.5. For binary turbo codes, $L_{\mathrm{sys}}(a_k|\tilde{\mathbf{r}}_{\mathrm{s}})$ is given by (8.98). Note that the extrinsic information that is generated by one SISO decoding unit serves at the a priori information for the other SISO decoding unit. The iterative procedure is started with initial condition $L_{\mathrm{ext2}}^{(0)}(a_k|\tilde{\mathbf{r}}_{\mathrm{p}}^{(2)}) = 0$, since the data symbols are assumed random and equally likely. The final bit decision for a_k is determined as $\hat{a}_k = \mathrm{sign}(L_2^{(m)}(a_k))$.

As mentioned previously, turbo codes can provide near Shannon limit performance. Figure 8.24 shows the typical performance of a rate-1/2, 16-state, PCCC on

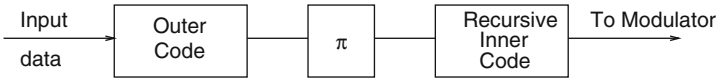


Fig. 8.25 SCCC encoder

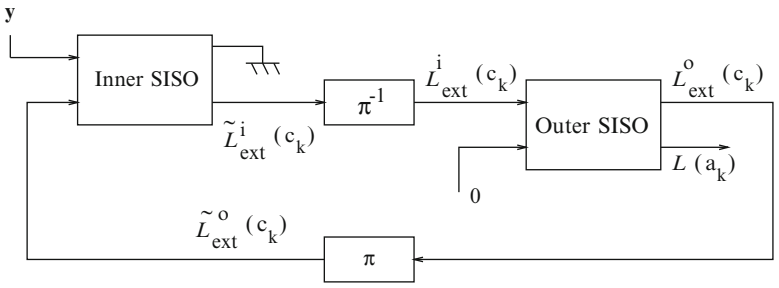


Fig. 8.26 SCCC decoder

an AWGN channel for different random interleaver sizes. The capacity limit for a rate-1/2 binary turbo code is -0.817 dB. Also included is a 2^{16}-state convolutional code for comparison. Observe that a simple 16-state PCCC can easily outperform a very complex 2^{16}-state convolutional code, at low E_b/N_o. At high E_b/N_o, the BER slope of PCCCs is shallow, loosely termed an *error floor*. The error floor is not actually an error floor, but rather a change in the slope of the error rate curve due to the relatively small free Hamming distance of turbo codes as we will see.

8.8.3 SCCC Encoder and Decoder

Figure 8.25 shows a SCCC encoder which is a serial concatenation of two component codes separated by an interleaver. In a SCCC scheme, the input data sequence of length N' is first encoded by an outer convolutional code C_o with rate $R^o = k/p$. The output of C_o is interleaved using a pseudo-random interleaver of length $N = N'/R^o$, and then encoded using an inner convolutional code C_i with rate $R^i = p/n$. The SCCC has code rate

$$R_T = R_c^{(1)} R_c^{(2)} = (k/p)(p/n) = k/n. \tag{8.182}$$

The codewords of the outer and inner codes are referred to as outer and inner codewords, respectively. Consequently, the inner codewords are also the codewords of the SCCC. With SCCCs, the inner encoder must be recursive for reasons to be seen later. The outer code does not have to be recursive.

The structure of the SCCC decoder is shown in Fig. 8.26. It operates in an iterative fashion similar to the PCCC decoder. However, the SISO modules now produce APPs or LLRs for the information bits, a_k, and the code bits c_k from the outer coder.

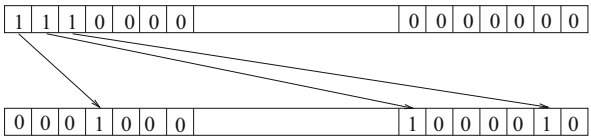


Fig. 8.27 Random turbo interleaver

8.8.4 Weight Distribution

It is sometimes useful to view PCCCs and SCCCs as equivalent block codes with input sequences of length $N' = N/k$ and $N' = Nk/p$, respectively, where N is the interleaver size. Like block codes, turbo codes can be described by a distance spectrum (d, A_d), where A_d is the number of codewords of weight Hamming weight d. The conditional weight enumerating function (CWEF) of a block code is defined as [30]

$$A_w(z) \stackrel{\triangle}{=} \sum_d A_{w,d} z^d, \tag{8.183}$$

where $A_{w,d}$ is the number of weight-d codewords having information-weight w. Note that $A_d = \sum_w A_{w,d}$. The smallest nonzero value of d is the free Hamming distance of the code, denoted by d_{free}. The union bound on the probability of bit error is

$$P_{\text{b}}(e) \leq \frac{1}{N'} \sum_w \sum_{d=d_{\text{free}}} w A_{w,d} P_2(d), \tag{8.184}$$

where $P_2(d)$ is the pairwise error probability between two coded sequences separated by Hamming distance d.

To obtain a low $P_{\text{b}}(e)$, there are generally two approaches; we can either decrease $A_{w,d}$ or increase d_{free}. With convolutional codes, A_d increases rapidly with d and, as a result, convolutional codes are said to have a *dense distance spectrum*.[4] Also, $A_d \propto N'$ with convolutional codes, due to their time invariant property. Hence, for convolutional codes a decrease in $P_{\text{b}}(e)$ is obtained by increasing d_{free}, which is ultimately obtained by increasing the total encoder memory. Turbo codes take other approach by drastically decreasing A_d. This property is called *spectral thinning*.

The spectral thinning property of turbo codes can be explained intuitively as follows. Considering PCCCs, the total weight of a PCCC codeword is equal to the weight of the systematic and parity components

$$w(\mathbf{b}) = w(\mathbf{b}_{\text{s}}) + w(\mathbf{b}_{\text{p}}^{(1)}) + w(\mathbf{b}_{\text{p}}^{(2)}). \tag{8.185}$$

Consider, for example, an RSC with generator matrix $\left[1, \frac{1+D^2}{1+D+D^2}\right]$ and the random interleaver shown in Fig. 8.27. Certain input sequences $\mathbf{a}$ will lead to low output

[4] It is important to realize that A_d is not equal to a_d (in our earlier discussion of convolutional codes), since the turbo codewords can consist of multiple error events.

weights $w(\mathbf{b}_{\mathrm{p}}^{(1)})$ from the first encoder RSC1. For example, the input sequence $\mathbf{a}(D) = 1 + D^3$ produces the output $\mathbf{b}_{\mathrm{p}}^{(1)}(D) = 1 + D + D^2 + D^3$ from the first encoder RSC1. However, the interleaved sequence $\tilde{\mathbf{a}}(D)$ will usually lead to a high output weight $w(\mathbf{b}_{\mathrm{p}}^{(2)})$ from the second encoder RSC2. Consequently, most codewords have large weight. However, some input sequences that produce low weight codewords in one encoder, after interleaving will also produce low weight codewords in the other encoder. Therefore, there are a few codewords with small weight. For most random interleavers, this event occurs with high probability [80]. At high $E_{\mathrm{b}}/N_{\mathrm{o}}$, the error events corresponding to these low-weight codewords dominate the BER performance with the result that the BER curves of PCCCs flatten at high $E_{\mathrm{b}}/N_{\mathrm{o}}$. This has been loosely termed as an *error floor* [30, 68].

In the sequel, convolutional codes, PCCC and SCCC are discussed simultaneously and, to avoid confusion, the quantities associated with them are distinguished by the superscripts c, T, and S, respectively.

For convolutional codes, every nonzero codeword corresponds to an error event or a concatenation of error events. The weight of a codeword equals the sum of the weights of the error events. Let $A^c_{w,d,i}$ denote the number of weight d codewords having information weight w and formed by the concatenation of i error events. Then, the number of weight d codewords with information weight w is $A^c_{w,d} = \sum_{i=1}^{n_{\max}} A^c_{w,d,i}$, where $n_{\max}$ is the maximum number of possible error events for a length-N' input sequence.

The distance spectrum of turbo codes is difficult to determine for a particular turbo interleaver. Fortunately, Benedetto and Montorsi [30] solved this problem by introducing a hypothetical interleaver called a *uniform interleaver* that permutes a given weight-w sequence onto any of the $\binom{N}{w}$ possible interleaved sequences with equal probability. The distance spectrum of a turbo code with a uniform interleaver can be obtained by averaging the distance spectrum over all possible interleaver mappings. At least half the random interleavers are guaranteed to yield a weight distribution that is as good as the average weight distribution. Furthermore, most of the randomly generated interleavers have a weight distribution that is close to the average weight distribution. Hence, the typical performance of a turbo code with a randomly chosen interleaver can be obtained from the average weight distribution with a uniform interleaver.

8.8.4.1 Weight Distribution of PCCCs

With a uniform interleaver the number of weight-d turbo codewords with weight-w input sequences is, for large N, [29]

$$A^T_{w,d} \approx \sum_{l=0}^{d} \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{\binom{N}{n_1}\binom{N}{n_2}}{\binom{N}{w}} A^c_{w,l,n_1} A^c_{w,d-l,n_2}. \tag{8.186}$$

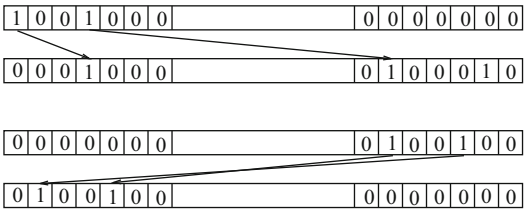


Fig. 8.28 Bad random interleaver mappings

Using the approximation $\binom{N}{n} \approx \frac{N^n}{n!}$ gives

$$A^T_{w,d} \approx \sum_{l=0}^{d} \sum_{n_1=1}^{n_{\max}} \sum_{n_2=1}^{n_{\max}} \frac{w!}{n_1! \cdot n_2!} N^{n_1+n_2-w} A^c_{w,l,n_1} A^c_{w,d-l,n_2}. \tag{8.187}$$

Observe that the multiplicity, $A^T_{w,d}$, of the PCCC codewords is inversely proportional to the interleaver length N. Consequently, increasing N results in very small multiplicity, a phenomenon called spectral thinning, and is the reason for the remarkable performance of turbo codes. In contrast, we note that the time-invariant property of convolutional codes implies that $A^c_d \propto N$.

The uniform interleaver is hypothetical and impractical. For reasonably large interleaver sizes N, random interleavers perform very well [80]. To see why, consider a rate-1/3, 8-state, PCCC code where the RSC component encoders have generator matrices $\left[1, \frac{1+D^2}{1+D+D^2}\right]$. Since the component codes are recursive, all weight-1 input sequences produce infinite-weight output sequences. The minimum distance error event at the output of *each* RSC encoder corresponds to an input error sequence of the form $D^i(1+D+D^2)$. However, the random interleaver permutes such sequences very effectively so that the output of the other encoder has high weight [80]. Weight-2 input error sequences to RSC1 of the form $D^i(1+D^3)$ will produce a finite-weight output sequence having the form $D^i(1+D+D^2+D^3)$. However, the random interleaver permutes these sequences into sequences which are *not* of the form $D^j(1+D^3)$ with high probability [80]. However, an occasional *bad mapping* occurs, where input sequences of the form $D^i(1+D^3)$ are permuted into input sequences of the form $D^j(1+D^3)$ for some i, j. This is illustrated in Fig. 8.28. Such input sequences produce low-weight outputs from both encoders and define the minimum Hamming distance of the PCCC code. The probability that an input sequence $D^i\mathbf{a}$ of weight-w is interleaved into a sequence $\tilde{\mathbf{a}}$ of the form $D^j\mathbf{a}$ for at least one pair i, j is proportional to N^{w-2} [80]. Hence, bad mappings are very likely to occur for weight-2 input sequences and very unlikely to occur for weight $w > 2$ input sequences. So the minimum distance error event corresponds to a weight-2 input sequence with very high probability. If the smallest weight RSC output corresponding to all weight-2 input sequences is d_{ceff}, then the free Hamming distance of the PCCC code is $d^T_{\text{free}} = 2 + 2d_{\text{ceff}}$. For our example PCCC code, the free distance is $d_{\text{free}} = 2 + 4 + 4 = 10$, which is rather small. This small free Hamming distance is typical of PCCCs precisely the reason for the so-called BER and frame

error rate (FER) floor of PCCCs. Finally, we note that other types of interleavers, such as the S-random interleaver, are generally very difficult to analyze, but most of the above arguments are valid.

PCCCs inherently provide unequal error protection, because the bad interleaver mappings define certain bit positions are affected by the dominant error events. Such bad mappings affect only a very few bit positions, but they nevertheless result in a BER floor. In contrast, for convolutional codes all bit positions in the input sequence are affected by the same error events. Consequently, all bit positions are equally likely to be in error. So PCCCs are inherently unequal error-protecting codes.

It is instructive to understand how the expected number of bad mappings changes with the interleaver size, N. The total number of possible interleaver mappings for a block of N bits is $N!$ The number of *bad mappings*, where a sequence of the form $D^i(1+D^3)$ is mapped into a sequence of the form $D^j(1+D^3)$, is approximately $N \times 2 \times (N-2)!$ The approximation is due to the fact that edge effects have been ignored which is a valid assumption for large N. Therefore, the probability that a sequence of the form $D^i(1+D^3)$ is mapped onto a sequence of the form $D^j(1+D^3)$ is

$$\mathrm{P}[D^i(1+D^3) \rightarrow D^j(1+D^3)] = \frac{2N(N-2)!}{N!} = \frac{2}{N-1}. \tag{8.188}$$

Assuming that the mappings for the different bit positions are independent and ignoring the edge effects, the distribution of the total number of such *bad* mappings k, in a block of length N, can be approximated by a binomial distribution for small k, that is,[5]

$$\mathrm{P}[\text{total number of bad mappings} = k] = \binom{N}{k} \left(\frac{2}{N-1}\right)^k \left(1 - \frac{2}{N-1}\right)^{N-k}.$$

The mean number of *bad* mappings is $N\frac{2}{N-1}$, which converges to 2 for large N. Therefore, the mean number of data bits affected by *bad* mappings converges to 4 for large N, since the *bad* mappings correspond to weight-2 input error sequences.

8.8.4.2 Weight Distribution of SCCCs

Consider the serial concatenation system in Fig. 8.25. Let the input block length is N' bits. The length of the outer codeword and, therefore, the interleaver size and length of the input to the inner encoder is $N = N'/R^{\mathrm{o}} = N'p/k$ bits. Under the assumption of a uniform interleaver, the number of weight h code words that are generated by weight w input sequences is [31]

[5]The case of large k is not of interest because the probability of many bad mappings is extremely small and, therefore, does not contribute significantly to the mean of the distribution.

$$A^S_{w,d} = \sum_{l=d^{\mathrm{o}}_{\mathrm{f}}}^{N} \sum_{n^{\mathrm{o}}=1}^{n^{\mathrm{o}}_{\mathrm{M}}} \sum_{n^{\mathrm{i}}=1}^{n^{\mathrm{i}}_{\mathrm{M}}} \frac{\binom{N'/p}{n^{\mathrm{o}}}\binom{N/p}{n^{\mathrm{i}}}}{\binom{N}{l}} A^{C_{\mathrm{o}}}_{w,l,n^{\mathrm{o}}} A^{C_{\mathrm{i}}}_{l,d,n^{\mathrm{i}}}, \tag{8.189}$$

where $d^{\mathrm{o}}_{\mathrm{f}}$ is the minimum free distance of the outer code, and $n^{\mathrm{o}}_{\mathrm{M}}$ and $n^{\mathrm{i}}_{\mathrm{M}}$ refer to the maximum number of error events possible for the outer and inner codes, respectively. Using the approximation $\binom{N}{n} \approx \frac{N^n}{n!}$ [31]

$$A^S_{w,d} \approx \sum_{l=d^{\mathrm{o}}_{\mathrm{f}}}^{N} \sum_{n^{\mathrm{o}}=1}^{n^{\mathrm{o}}_{\mathrm{M}}} \sum_{n^{\mathrm{i}}=1}^{n^{\mathrm{i}}_{\mathrm{M}}} N^{n^{\mathrm{o}}+n^{\mathrm{i}}-l-1} \frac{l!}{p^{n^{\mathrm{o}}+n^{\mathrm{i}}} n^{\mathrm{o}}! n^{\mathrm{i}}!} \frac{1}{n} A^{o}_{w,l,n^{\mathrm{o}}} A^{i}_{l,h,n^{\mathrm{i}}}, \tag{8.190}$$

where $w^{\mathrm{o}}_{\mathrm{m}}$ is the minimum-weight of all input sequences that will produce an error event for the outer code.

Observe from (8.190) that the contribution of each codeword to the BER is multiplied by the term $N^{n^{\mathrm{o}}+n^{\mathrm{i}}-l-1}$. Therefore, when $n^{\mathrm{o}}+n^{\mathrm{i}}-l-1<0$, increasing N decreases the BER exponentially. This effect is called the interleaver gain. Consider a weight-l outer codeword which is a result of n^{o} error events of the outer code. If the inner encoder is nonrecursive, then a weight-l outer codeword can result in a maximum of l error events (each "1" in the outer codeword can cause an error event). Therefore, n^{i} can be equal to l. In this case, the exponent of N will be $n^{\mathrm{o}}-1$, and, when $n^{\mathrm{o}}>1$, the exponent of N will be positive. Consequently, increasing N increases the contribution of such codewords to the final BER [31]. When $n^{\mathrm{o}}=1$, the exponent of N will be zero, implying that the interleaver does not impact the multiplicity of such codewords or, equivalently, no interleaving gain is possible.

When the inner encoder is recursive, only input sequences having weight-2 or greater can cause error events. Therefore, a weight-l outer codeword can cause at most $\lfloor l/2 \rfloor$ error events for the inner code. Consequently, the exponent of N is $n^{\mathrm{o}} - \lceil l/2 \rceil - 1$. If all outer codewords corresponding to one error event of the outer code ($n^{\mathrm{o}}=1$) have weight $l>2$ or, equivalently, the free distance of the outer code is greater than 2, the exponent of N is always negative. This implies that increasing N will always decrease the BER.

Problems

8.1. Consider a simple repetition code that generates codewords by simply repeating each information symbol L times.

(a) What is the rate of the code?
(b) How many codewords are there in the code?
(c) What is the minimum distance of the code?
(d) How many channel errors would have to occur to confuse one codeword with another?

8.2. The generator matrix for a (6,3) linear binary block code is

$$\mathbf{G} = \begin{bmatrix} 0\,1\,1\,1\,0\,0 \\ 1\,0\,1\,0\,1\,0 \\ 1\,1\,0\,0\,0\,1 \end{bmatrix}$$

(a) What is the parity check matrix for this code?
(b) Generate the standard array for this code.
(c) Calculate the syndrome vector for all of the correctable error patterns.
(d) Decode the received sequence $\mathbf{y} = 101101$.

8.3. The parity check matrix $\mathbf{H}$ for a linear block code is given as follows:

$$\mathbf{H} = \begin{bmatrix} 1\,0\,0\,1\,1 \\ 0\,1\,0\,1\,1 \\ 0\,0\,1\,0\,1 \end{bmatrix}$$

(a) Construct a standard array decoding table for this code.
(b) How many error patterns can this code correct?
(c) If this code is used for error detection on a BSC with crossover probability p, what is the probability of undetected error?

8.4. Consider a systematic (15,11) Hamming code.

(a) Construct a parity check matrix for a systematic (15,11) Hamming code.
(b) Construct a syndrome table for the code defined by the parity check matrix.
(c) If the (15,11) Hamming code is used for error detection on a BSC with crossover probability p, what is the probability of undetected error?

8.5. Determine the decision variables for the separate maximum likelihood decoding of the symbols in the following rate-3/4 space-time block code

$$C = \begin{bmatrix} \mathbf{s}_{(1)} & \mathbf{s}_{(2)} & \mathbf{s}_{(3)} \\ -\mathbf{s}^*_{(2)} & \mathbf{s}^*_{(1)} & 0 \\ \mathbf{s}^*_{(3)} & 0 & -\mathbf{s}^*_{(1)} \\ 0 & \mathbf{s}^*_{(3)} & -\mathbf{s}^*_{(2)} \end{bmatrix}.$$

8.6. Determine the decision variables for the separate maximum likelihood decoding of the symbols in the following rate-1/2 orthogonal space-time block code in (8.30).

8.7. The generator matrix for a rate-1 space-time block code is given as

$$C = \begin{bmatrix} \mathbf{s}_{(1)} & \mathbf{s}_{(2)} & \mathbf{s}_{(3)} & \mathbf{s}_{(4)} \\ -\mathbf{s}^*_{(2)} & \mathbf{s}^*_{(1)} & -\mathbf{s}^*_{(4)} & \mathbf{s}^*_{(3)} \\ -\mathbf{s}^*_{(3)} & -\mathbf{s}^*_{(4)} & \mathbf{s}^*_{(1)} & \mathbf{s}^*_{(2)} \\ \mathbf{s}_{(4)} & -\mathbf{s}_{(3)} & -\mathbf{s}_{(2)} & \mathbf{s}_{(1)} \end{bmatrix}.$$

(a) Determine the matrix $C^{\mathrm{H}}C$, and thus show that the code is not orthogonal.
(b) Show that the maximum likelihood decoder can perform pairwise maximum likelihood detection
(c) What is the diversity order achieved by the code?

8.8. Consider a rate-1/3 convolutional code with generators $\mathbf{g}^{(1)} = (111)$, $\mathbf{g}^{(2)} = (111)$, and $\mathbf{g}^{(3)} = (101)$.

(a) Draw a block diagram of the encoder structure.
(b) Draw the state diagram and trellis diagram.
(c) Determine the output sequence corresponding to the input sequence 1110101.

8.9. The output of a rate-1/3 convolutional encoder with constraint length 3 to the input $\mathbf{a} = (1,1,0,\ldots)$ is $\mathbf{b} = (111,110,010,\ldots)$

(a) Determine the transfer function $T(D,N,L)$.
(b) Determine the number of paths through the state diagram or trellis that diverge from the all-zeroes state and merge with the all-zeroes state 7 branches later.
(c) Determine the number of paths of Hamming distance 20 from the all zeroes sequence.

8.10. Consider the rate-1/3 code in Problem 8.8.

(a) Determine the transfer function $T(D,N,L)$ of the code. What is the free Hamming distance d_{free}?
(b) Assuming the use of BPSK signaling and an AWGN channel, derive a union-Chernoff bound on the decoded bit error probability with (1) hard decision decoding and (2) soft decision decoding.
(c) Repeat part (b) assuming an interleaved flat Rayleigh fading channel, where the receiver has perfect knowledge of the channel.

8.11. Consider the 8-PAM and 32-CROSS signal constellations in Fig. 8.29.

(a) Construct the partition chain as in Fig. 8.12 and compute the minimum Euclidean distance between signal points at each step in the partition chain.
(b) What is the average symbol energy for each of the signal constellations.

8.12. Consider the 2-state, rate-1/2, trellis encoder shown in Fig. 8.30. Using this encoder with a 4-PAM and 8-PAM signal constellation, we can construct TCM systems having bandwidth efficiencies of 1 bit/s/Hz and 2 bits/s/Hz, respectively.

(a) Determine the appropriate partitions for the signal constellation for the 2-state, 4-PAM and 8-PAM trellis codes.
(b) Construct and label the trellis diagrams for the 2-state 4-PAM and 8-PAM trellis codes.
(c) Determine the minimum Euclidean distance for each trellis code, and the asymptotic coding gain on an AWGN channel relative to the equivalent uncoded systems.

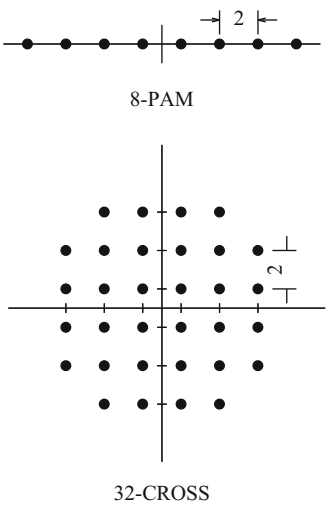


Fig. 8.29 Signal constellations for Prob. 8.11. (**a**) 8-PAM (**b**) 32-CROSS

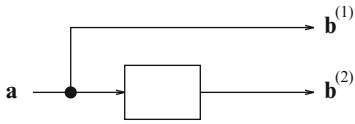


Fig. 8.30 Trellis encoder for Prob. 8.12

8.13. To simplify the calculation of performance bounds, a Chernoff bound is often imposed on the pairwise error probability.

(a) Derive the Chernoff bound on the pairwise error probability for an AWGN channel with soft decision decoding, given by (8.122).
(b) Derive the Chernoff bound on the pairwise error probability for an AWGN channel with hard decision decoding, given by (8.123).
(c) Derive the Chernoff bound on the pairwise error probability for an interleaved flat fading channel with soft decision decoding, given by (8.132).