CHAPTER
5

Information Theory

5.1 Introduction

As mentioned in Chapter 1 and reiterated along the way, the purpose of a communication system is to facilitate the transmission of signals generated by a source of information over a communication channel. But, in basic terms, what do we mean by the term information? To address this important issue, we need to understand the fundamentals of information theory.[1]

The rationale for studying the fundamentals of information theory at this early stage in the book is threefold:

1. Information theory makes extensive use of probability theory, which we studied in Chapter 3; it is, therefore, a logical follow-up to that chapter.
2. It adds meaning to the term "information" used in previous chapters of the book.
3. Most importantly, information theory paves the way for many important concepts and topics discussed in subsequent chapters.

In the context of communications, information theory deals with mathematical modeling and analysis of a communication system rather than with physical sources and physical channels. In particular, it provides answers to two fundamental questions (among others):

1. What is the irreducible complexity, below which a signal cannot be compressed?
2. What is the ultimate transmission rate for reliable communication over a noisy channel?

The answers to these two questions lie in the entropy of a source and the capacity of a channel, respectively:

1. *Entropy* is defined in terms of the probabilistic behavior of a source of information; it is so named in deference to the parallel use of this concept in thermodynamics.
2. *Capacity* is defined as the intrinsic ability of a channel to convey information; it is naturally related to the noise characteristics of the channel.

A remarkable result that emerges from information theory is that if the entropy of the source is less than the capacity of the channel, then, ideally, error-free communication over the channel can be achieved. It is, therefore, fitting that we begin our study of information theory by discussing the relationships among uncertainty, information, and entropy.

5.2 Entropy

Suppose that a *probabilistic experiment* involves observation of the output emitted by a discrete source during every signaling interval. The source output is modeled as a

stochastic process, a sample of which is denoted by the discrete random variable S. This random variable takes on symbols from the fixed finite *alphabet*

$$\mathcal{S} = \{s_0, s_1, \ldots, s_{K-1}\} \tag{5.1}$$

with probabilities

$$\mathbb{P}(S=s_k) = p_k, \qquad k = 0, 1, \ldots, K-1 \tag{5.2}$$

Of course, this set of probabilities must satisfy the normalization property

$$\sum_{k=0}^{K-1} p_k = 1, \qquad p_k \geq 0 \tag{5.3}$$

We assume that the symbols emitted by the source during successive signaling intervals are statistically independent. Given such a scenario, can we find a *measure* of how much information is produced by such a source? To answer this question, we recognize that the idea of information is closely related to that of uncertainty or surprise, as described next.

Consider the event $S = s_k$, describing the emission of symbol s_k by the source with probability p_k, as defined in (5.2). Clearly, if the probability $p_k = 1$ and $p_i = 0$ for all $i \neq k$, then there is no "surprise" and, therefore, no "information" when symbol s_k is emitted, because we know what the message from the source must be. If, on the other hand, the source symbols occur with different probabilities and the probability p_k is low, then there is more surprise and, therefore, information when symbol s_k is emitted by the source than when another symbol s_i, $i \neq k$, with higher probability is emitted. Thus, the words *uncertainty*, *surprise*, and *information* are all related. Before the event $S = s_k$ occurs, there is an amount of uncertainty. When the event $S = s_k$ occurs, there is an amount of surprise. After the occurrence of the event $S = s_k$, there is gain in the amount of information, the essence of which may be viewed as the *resolution of uncertainty*. Most importantly, the amount of information is related to the inverse of the probability of occurrence of the event $S = s_k$.

We define the *amount of information* gained after observing the event $S = s_k$, which occurs with probability p_k, as the logarithmic function[2]

$$I(s_k) = \log\left(\frac{1}{p_k}\right) \tag{5.4}$$

which is often termed "self-information" of the event $S = s_k$. This definition exhibits the following important properties that are intuitively satisfying:

PROPERTY 1

$$I(s_k) = 0 \qquad \text{for } p_k = 1 \tag{5.5}$$

Obviously, if we are absolutely *certain* of the outcome of an event, even before it occurs, there is *no* information gained.

PROPERTY 2

$$I(s_k) \geq 0 \qquad \text{for } 0 \leq p_k \leq 1 \tag{5.6}$$

That is to say, the occurrence of an event $S = s_k$ either provides some or no information, but never brings about a *loss* of information.

PROPERTY 3

$$I(s_k) > I(s_i) \qquad \text{for } p_k < p_i \tag{5.7}$$

That is, the less probable an event is, the more information we gain when it occurs.

PROPERTY 4

$$I(s_k, s_l) = I(s_k) + I(s_l) \text{ if } s_k \text{ and } s_l \text{ are statistically independent}$$

This additive property follows from the logarithmic definition described in (5.4).

The base of the logarithm in (5.4) specifies the units of information measure. Nevertheless, it is standard practice in information theory to use a logarithm to base 2 with binary signaling in mind. The resulting unit of information is called the *bit*, which is a contraction of the words binary digit. We thus write

$$\begin{aligned} I(s_k) &= \log_2\left(\frac{1}{p_k}\right) \\ &= -\log_2 p_k \qquad \text{for } k = 0, 1, \ldots, K-1 \end{aligned} \tag{5.8}$$

When $p_k = 1/2$, we have $I(s_k) = 1$ bit. We may, therefore, state:

> One bit is the amount of information that we gain when one of two possible and equally likely (i.e., equiprobable) events occurs.

Note that the information $I(s_k)$ is positive, because the logarithm of a number less than one, such as a probability, is negative. Note also that if p_k is zero, then the self-information I_{s_k} assumes an unbounded value.

The amount of information $I(s_k)$ produced by the source during an arbitrary signaling interval depends on the symbol s_k emitted by the source at the time. The self-information $I(s_k)$ is a discrete random variable that takes on the values $I(s_0)$, $I(s_1)$, …, $I(s_{K-1})$ with probabilities p_0, p_1, …, p_{K-1} respectively. The *expectation* of $I(s_k)$ over all the probable values taken by the random variable S is given by

$$\begin{aligned} H(S) &= \mathbb{E}[I(s_k)] \\ &= \sum_{k=0}^{K-1} p_k I(s_k) \\ &= \sum_{k=0}^{K-1} p_k \log_2\left(\frac{1}{p_k}\right) \end{aligned} \tag{5.9}$$

The quantity $H(S)$ is called the *entropy*,[3] formally defined as follows:

> The entropy of a discrete random variable, representing the output of a source of information, is a measure of the average information content per source symbol.

Note that the entropy $H(S)$ is independent of the alphabet $\mathcal{S}$; it depends only on the probabilities of the symbols in the alphabet $\mathcal{S}$ of the source.

Properties of Entropy

Building on the definition of entropy given in (5.9), we find that entropy of the discrete random variable S is bounded as follows:

$$0 \le H(S) \le \log_2 K \tag{5.10}$$

where K is the number of symbols in the alphabet $\mathcal{S}$.

Elaborating on the two bounds on entropy in (5.10), we now make two statements:

1. $H(S) = 0$, if, and only if, the probability $p_k = 1$ for some k, and the remaining probabilities in the set are all zero; this lower bound on entropy corresponds to *no uncertainty.*
2. $H(S) = \log K$, if, and only if, $p_k = 1/K$ for all k (i.e., all the symbols in the source alphabet $\mathcal{S}$ are equiprobable); this upper bound on entropy corresponds to maximum uncertainty.

To prove these properties of $H(S)$, we proceed as follows. First, since each probability p_k is less than or equal to unity, it follows that each term $p_k \log_2(1/p_k)$ in (5.9) is always nonnegative, so $H(S) \geq 0$. Next, we note that the product term $p_k \log_2(1/p_k)$ is zero if, and only if, $p_k = 0$ or 1. We therefore deduce that $H(S) = 0$ if, and only if, $p_k = 0$ or 1 for some k and all the rest are zero. This completes the proofs of the lower bound in (5.10) and statement 1.

To prove the upper bound in (5.10) and statement 2, we make use of a property of the natural logarithm:

$$\log_e x \leq x - 1, \qquad x \geq 0 \tag{5.11}$$

where $\log_e$ is another way of describing the *natural logarithm*, commonly denoted by ln; both notations are used interchangeably. This inequality can be readily verified by plotting the functions $\ln x$ and $(x - 1)$ versus x, as shown in Figure 5.1. Here we see that the line $y = x - 1$ always lies above the curve $y = \log_e x$. The equality holds only at the point $x = 1$, where the line is tangential to the curve.

To proceed with the proof, consider first any two different probability distributions denoted by $p_0, p_1, \ldots, p_{K-1}$ and $q_0, q_1, \ldots, q_{K-1}$ on the alphabet $\mathcal{S} = \{s_0, s_1, \ldots, s_{K-1}\}$ of a discrete source. We may then define the *relative entropy* of these two distributions:

$$D(p||q) = \sum_{k=0}^{K-1} p_k \log_2\left(\frac{p_k}{q_k}\right) \tag{5.12}$$

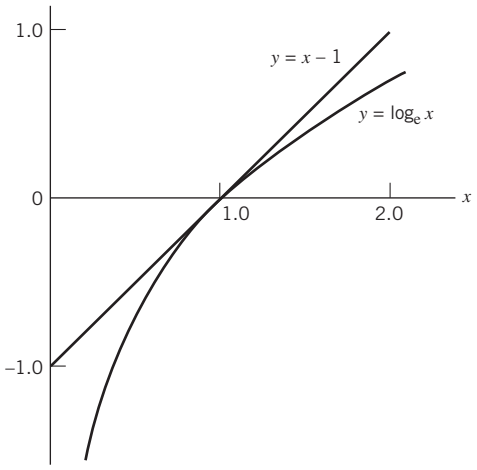


Figure 5.1 Graphs of the functions $x - 1$ and $\log x$ versus x.

Hence, changing to the natural logarithm and using the inequality of (5.11), we may express the summation on the right-hand side of (5.12) as follows:

$$\begin{aligned}\sum_{k=0}^{K-1} p_k \log_2\left(\frac{p_k}{q_k}\right) &= -\sum_{k=0}^{K-1} p_k \log_2\left(\frac{q_k}{p_k}\right)\\ &\geq \frac{1}{\ln 2}\sum_{k=0}^{K-1} p_k\left(\frac{q_k}{p_k} - 1\right)\\ &= \frac{1}{\log_e 2}\sum_{k=0}^{K-1}(q_k - p_k)\\ &= 0\end{aligned}$$

where, in the third line of the equation, it is noted that the sums over p_k and q_k are both equal to unity in accordance with (5.3). We thus have the *fundamental property* of probability theory:

$$D(p||q) \geq 0 \tag{5.13}$$

In words, (5.13) states:

> The relative entropy of a pair of different discrete distributions is always nonnegative; it is zero only when the two distributions are identical.

Suppose we next put

$$q_k = \frac{1}{K}, \qquad k = 0, 1, \ldots, K-1$$

which corresponds to a source alphabet $\mathcal{S}$ with *equiprobable* symbols. Using this distribution in (5.12) yields

$$\begin{aligned}D(p||q) &= \sum_{k=0}^{K-1} p_k \log_2 p_k + \log_2 K \sum_{k=0}^{K-1} p_k\\ &= -H(S) + \log_2 K\end{aligned}$$

where we have made use of (5.3) and (5.9). Hence, invoking the fundamental inequality of (5.13), we may finally write

$$H(S) \leq \log_2 K \tag{5.14}$$

Thus, $H(S)$ is always less than or equal to $\log_2 K$. The equality holds if, and only if, the symbols in the alphabet $\mathcal{S}$ are equiprobable. This completes the proof of (5.10) and with it the accompanying statements 1 and 2.

EXAMPLE 1

Entropy of Bernoulli Random Variable

To illustrate the properties of $H(S)$ summed up in (5.10), consider the Bernoulli random variable for which symbol 0 occurs with probability p_0 and symbol 1 with probability $p_1 = 1 - p_0$.

The entropy of this random variable is

$$\begin{aligned} H(S) &= -p_0 \log_2 p_0 - p_1 \log_2 p_1 \\ &= -p_0 \log_2 p_0 - (1 - p_0) \log_2 (1 - p_0) \quad \text{bits} \end{aligned} \tag{5.15}$$

from which we observe the following:

1. When $p_0 = 0$, the entropy $H(S) = 0$; this follows from the fact that $x \log_e x \to 0$ as $x \to 0$.
2. When $p_0 = 1$, the entropy $H(S) = 0$.
3. The entropy $H(S)$ attains its maximum value $H_{max} = 1$ bit when $p_1 = p_0 = 1/2$; that is, when symbols 1 and 0 are equally probable.

In other words, $H(S)$ is symmetric about $p_0 = 1/2$.

The function of p_0 given on the right-hand side of (5.15) is frequently encountered in information-theoretic problems. It is customary, therefore, to assign a special symbol to this function. Specifically, we define

$$H(p_0) = -p_0 \log_2 p_0 - (1 - p_0) \log_2 (1 - p_0) \tag{5.16}$$

We refer to $H(p_0)$ as the *entropy function*. The distinction between (5.15) and (5.16) should be carefully noted. The $H(S)$ of (5.15) gives the entropy of the Bernoulli random variable S. The $H(p_0)$ of (5.16), on the other hand, is a function of the prior probability p_0 defined on the interval [0, 1]. Accordingly, we may plot the entropy function $H(p_0)$ versus p_0, defined on the interval [0, 1], as shown in Figure 5.2. The curve in Figure 5.2 highlights the observations made under points 1, 2, and 3.

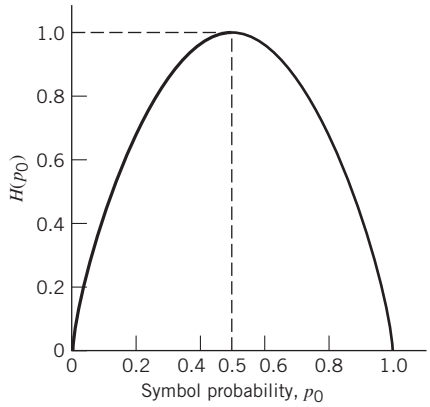


Figure 5.2
Entropy function $H(p_0)$.

Extension of a Discrete Memoryless Source

To add specificity to the discrete source of symbols that has been the focus of attention up until now, we now assume it to be *memoryless* in the sense that the symbol emitted by the source at any time is independent of previous and future emissions.

In this context, we often find it useful to consider *blocks* rather than individual symbols, with each block consisting of n successive source symbols. We may view each such block

as being produced by an *extended source* with a source alphabet described by the Cartesian product of a set S^n that has K^n distinct blocks, where K is the number of distinct symbols in the source alphabet S of the original source. With the source symbols being statistically independent, it follows that the probability of a source symbol in S^n is equal to the product of the probabilities of the n source symbols in S that constitute a particular source symbol of S^n. We may thus intuitively expect that $H(S^n)$, the entropy of the extended source, is equal to n times $H(S)$, the entropy of the original source. That is, we may write

$$H(S^{(n)}) = nH(S) \tag{5.17}$$

We illustrate the validity of this relationship by way of an example.

EXAMPLE 2 **Entropy of Extended Source**

Consider a discrete memoryless source with source alphabet $\mathcal{S} = \{s_0, s_1, s_2\}$, whose three distinct symbols have the following probabilities:

$$p_0 = \frac{1}{4}$$

$$p_1 = \frac{1}{4}$$

$$p_2 = \frac{1}{2}$$

Hence, the use of (5.9) yields the entropy of the discrete random variable S representing the source as

$$\begin{aligned} H(S) &= p_0 \log_2\left(\frac{1}{p_0}\right) + p_1 \log_2\left(\frac{1}{p_1}\right) + p_2 \log_2\left(\frac{1}{p_2}\right) \\ &= \frac{1}{4}\log_2(4) + \frac{1}{4}\log_2(4) + \frac{1}{2}\log_2(2) \\ &= \frac{3}{2} \text{ bits} \end{aligned}$$

Consider next the second-order extension of the source. With the source alphabet $\mathcal{S}$ consisting of three symbols, it follows that the source alphabet of the extended source $S^{(2)}$ has nine symbols. The first row of Table 5.1 presents the nine symbols of $S^{(2)}$, denoted by $\sigma_0, \sigma_1, \ldots, \sigma_8$. The second row of the table presents the composition of these nine symbols in terms of the corresponding sequences of source symbols s_0, s_1, and s_2, taken two at a

Table 5.1 **Alphabets of second-order extension of a discrete memoryless source**

Symbols of $S^{(2)}$	σ_0	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
Corresponding sequences of symbols of S	s_0s_0	s_0s_1	s_0s_2	s_1s_0	s_1s_1	s_1s_2	s_2s_0	s_2s_1	s_2s_2
Probability $\mathbb{P}(\sigma_i)$, $i = 0, 1, \ldots, 8$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{4}$

time. The probabilities of the nine source symbols of the extended source are presented in the last row of the table. Accordingly, the use of (5.9) yields the entropy of the extended source as

$$
\begin{aligned}
H(S^{(2)}) &= \sum_{i=0}^{8} p(\sigma_i) \log_2\left(\frac{1}{p(\sigma_i)}\right) \\
&= \frac{1}{16}\log_2(16) + \frac{1}{16}\log_2(16) + \frac{1}{8}\log_2(8) + \frac{1}{16}\log_2(16) \\
&+ \frac{1}{16}\log_2(16) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{4}\log_2(4) \\
&= 3 \text{ bits}
\end{aligned}
$$

We thus see that $H(S^{(2)}) = 2H(S)$ in accordance with (5.17).

5.3 Source-coding Theorem

Now that we understand the meaning of entropy of a random variable, we are equipped to address an important issue in communication theory: the representation of data generated by a discrete source of information.

The process by which this representation is accomplished is called *source encoding*. The device that performs the representation is called a *source encoder*. For reasons to be described, it may be desirable to know the statistics of the source. In particular, if some source symbols are known to be more probable than others, then we may exploit this feature in the generation of a *source code* by assigning *short* codewords to *frequent* source symbols, and *long* codewords to *rare* source symbols. We refer to such a source code as a *variable-length code*. The *Morse code*, used in telegraphy in the past, is an example of a variable-length code. Our primary interest is in the formulation of a source encoder that satisfies two requirements:

1. The codewords produced by the encoder are in *binary* form.
2. The source code is *uniquely decodable*, so that the original source sequence can be reconstructed perfectly from the encoded binary sequence.

The second requirement is particularly important: it constitutes the basis for a *perfect source code*.

Consider then the scheme shown in Figure 5.3 that depicts a discrete memoryless source whose output s_k is converted by the source encoder into a sequence of 0s and 1s, denoted by b_k. We assume that the source has an alphabet with K different symbols and that the kth symbol s_k occurs with probability p_k, $k = 0, 1, \ldots, K-1$. Let the binary

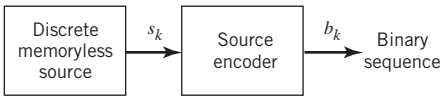


Figure 5.3 Source encoding.

codeword assigned to symbol s_k by the encoder have length l_k, measured in bits. We define the *average codeword length* $\bar{L}$ of the source encoder as

$$\bar{L} = \sum_{k=0}^{K-1} p_k l_k \tag{5.18}$$

In physical terms, the parameter $\bar{L}$ represents the *average number of bits per source symbol* used in the source encoding process. Let $L_{\min}$ denote the *minimum* possible value of L. We then define the *coding efficiency* of the source encoder as

$$\eta = \frac{L_{\min}}{\bar{L}} \tag{5.19}$$

With $\bar{L} \geq L_{\min}$, we clearly have $\eta \leq 1$. The source encoder is said to be *efficient* when η approaches unity.

But how is the minimum value $L_{\min}$ determined? The answer to this fundamental question is embodied in Shannon's first theorem: the *source-coding theorem*,[4] which may be stated as follows:

> Given a discrete memoryless source whose output is denoted by the random variable S, the entropy $H(S)$ imposes the following bound on the average codeword length $\bar{L}$ for any source encoding scheme:
>
> $$\bar{L} \geq H(S) \tag{5.20}$$

According to this theorem, the entropy $H(S)$ represents a *fundamental limit* on the average number of bits per source symbol necessary to represent a discrete memoryless source, in that it can be made as small as but no smaller than the entropy $H(S)$. Thus, setting $L_{\min} = H(S)$, we may rewrite (5.19), defining the efficiency of a source encoder in terms of the entropy $H(S)$ as shown by

$$\eta = \frac{H(S)}{\bar{L}} \tag{5.21}$$

where as before we have $\eta \leq 1$.

5.4 Lossless Data Compression Algorithms

A common characteristic of signals generated by physical sources is that, in their natural form, they contain a significant amount of *redundant* information, the transmission of which is therefore wasteful of primary communication resources. For example, the output of a computer used for business transactions constitutes a redundant sequence in the sense that any two adjacent symbols are typically correlated with each other.

For efficient signal transmission, the redundant information should, therefore, be removed from the signal prior to transmission. This operation, with no loss of information, is ordinarily performed on a signal in digital form, in which case we refer to the operation as *lossless data compression*. The code resulting from such an operation provides a representation of the source output that is not only efficient in terms of the average number of bits per symbol, but also exact in the sense that the original data can be reconstructed with no loss of information. The entropy of the source establishes the fundamental limit on the removal of redundancy from the data. Basically, lossless data compression is achieved

by assigning short descriptions to the most frequent outcomes of the source output and longer descriptions to the less frequent ones.

In this section we discuss some source-coding schemes for lossless data compression. We begin the discussion by describing a type of source code known as a prefix code, which not only is uniquely decodable, but also offers the possibility of realizing an average codeword length that can be made arbitrarily close to the source entropy.

Prefix Coding

Consider a discrete memoryless source of alphabet $\{s_0, s_1, \ldots, s_{K-1}\}$ and respective probabilities $\{p_0, p_1, \ldots, p_{K-1}\}$. For a source code representing the output of this source to be of practical use, the code has to be uniquely decodable. This restriction ensures that, for each finite sequence of symbols emitted by the source, the corresponding sequence of codewords is different from the sequence of codewords corresponding to any other source sequence. We are specifically interested in a special class of codes satisfying a restriction known as the *prefix condition*. To define the prefix condition, let the codeword assigned to source symbol s_k be denoted by $(m_{k_1}, m_{k_2}, \ldots, m_{k_n})$, where the individual elements $m_{k_1}, \ldots, m_{k_n}$ are 0s and 1s and n is the codeword length. The initial part of the codeword is represented by the elements $m_{k_1}, \ldots, m_{k_i}$ for some $i \leq n$. Any sequence made up of the initial part of the codeword is called a *prefix* of the codeword. We thus say:

> A *prefix code* is defined as a code in which no codeword is the prefix of any other codeword.

Prefix codes are distinguished from other uniquely decodable codes by the fact that the end of a codeword is always recognizable. Hence, the decoding of a prefix can be accomplished as soon as the binary sequence representing a source symbol is fully received. For this reason, prefix codes are also referred to as *instantaneous codes*.

EXAMPLE 3 **Illustrative Example of Prefix Coding**

To illustrate the meaning of a prefix code, consider the three source codes described in Table 5.2. Code I is not a prefix code because the bit 0, the codeword for s_0, is a prefix of 00, the codeword for s_2. Likewise, the bit 1, the codeword for s_1, is a prefix of 11, the codeword for s_3. Similarly, we may show that code III is not a prefix code but code II is.

Table 5.2 Illustrating the definition of a prefix code

Symbol source	Probability of occurrence	Code I	Code II	Code III
s_0	0.5	0	0	0
s_1	0.25	1	10	01
s_2	0.125	00	110	011
s_3	0.125	11	111	0111

Decoding of Prefix Code

To decode a sequence of codewords generated from a prefix source code, the *source decoder* simply starts at the beginning of the sequence and decodes one codeword at a time. Specifically, it sets up what is equivalent to a *decision tree*, which is a graphical portrayal of the codewords in the particular source code. For example, Figure 5.4 depicts the decision tree corresponding to code II in Table 5.2. The tree has an *initial state* and four *terminal states* corresponding to source symbols s_0, s_1, s_2, and s_3. The decoder always starts at the initial state. The first received bit moves the decoder to the terminal state s_0 if it is 0 or else to a second decision point if it is 1. In the latter case, the second bit moves the decoder one step further down the tree, either to terminal state s_1 if it is 0 or else to a third decision point if it is 1, and so on. Once each terminal state emits its symbol, the decoder is reset to its initial state. Note also that each bit in the received encoded sequence is examined only once. Consider, for example, the following encoded sequence:

$$\underbrace{10}_{s_1}\ \underbrace{111}_{s_3}\ \underbrace{110}_{s_2}\ \underbrace{0}_{s_0}\ \underbrace{0}_{s_0}\ \ldots$$

This sequence is readily decoded as the source sequence $s_1 s_3 s_2 s_0 s_0 \ldots$. The reader is invited to carry out this decoding.

As mentioned previously, a prefix code has the important property that it is instantaneously decodable. But the converse is not necessarily true. For example, code III in Table 5.2 does not satisfy the prefix condition, yet it is uniquely decodable because the bit 0 indicates the beginning of each codeword in the code.

To probe more deeply into prefix codes, exemplified by that in Table 5.2, we resort to an inequality, which is considered next.

Kraft Inequality

Consider a discrete memoryless source with source alphabet $\{s_0, s_1, \ldots, s_{K-1}\}$ and source probabilities $\{p_0, p_1, \ldots, p_{K-1}\}$, with the codeword of symbol s_k having length l_k, $k = 0, 1,$

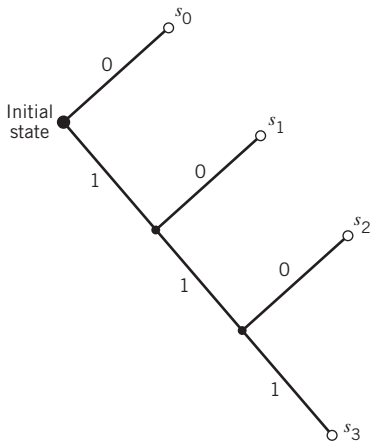


Figure 5.4 Decision tree for code II of Table 5.2.

…, $K-1$. Then, according to the *Kraft inequality*,[5] the codeword lengths always satisfy the following inequality:

$$\sum_{k=0}^{K-1} 2^{-l_k} \leq 1 \tag{5.22}$$

where the factor 2 refers to the number of symbols in the binary alphabet. The Kraft inequality is a necessary but not sufficient condition for a source code to be a prefix code. In other words, the inequality of (5.22) is merely a condition on the codeword lengths of a prefix code and not on the codewords themselves. For example, referring to the three codes listed in Table 5.2, we see:

- Code I violates the Kraft inequality; it cannot, therefore, be a prefix code.
- The Kraft inequality is satisfied by both codes II and III, but only code II is a prefix code.

Given a discrete memoryless source of entropy $H(S)$, a prefix code can be constructed with an average codeword length $\bar{L}$, which is bounded as follows:

$$H(S) \leq \bar{L} < H(S) + 1 \tag{5.23}$$

The left-hand bound of (5.23) is satisfied with equality under the condition that symbol s_k is emitted by the source with probability

$$p_k = 2^{-l_k} \tag{5.24}$$

where l_k is the length of the codeword assigned to source symbol s_k. A distribution governed by (5.24) is said to be a *dyadic distribution*. For this distribution, we naturally have

$$\sum_{k=0}^{K-1} 2^{-l_k} = \sum_{k=0}^{K-1} p_k = 1$$

Under this condition, the Kraft inequality of (5.22) confirms that we can construct a prefix code, such that the length of the codeword assigned to source symbol s_k is $-\log_2 p_k$. For such a code, the average codeword length is

$$\bar{L} = \sum_{k=0}^{K-1} \frac{l_k}{2^{l_k}} \tag{5.25}$$

and the corresponding entropy of the source is

$$\begin{aligned} H(S) &= \sum_{k=0}^{K-1} \left(\frac{1}{2^{l_k}} \right) \log_2 (2^{l_k}) \\ &= \sum_{k=0}^{K-1} \left(\frac{l_k}{2^{l_k}} \right) \end{aligned} \tag{5.26}$$

Hence, in this special (rather meretricious) case, we find from (5.25) and (5.26) that the prefix code is *matched* to the source in that $\bar{L} = H(S)$.

But how do we match the prefix code to an arbitrary discrete memoryless source? The answer to this basic problem lies in the use of an *extended code*. Let $\bar{L}_n$ denote the

average codeword length of the extended prefix code. For a uniquely decodable code, $\bar{L}_n$ is the smallest possible. From (5.23), we find that

$$nH(S) \leq \bar{L}_n < nH(S) + 1 \tag{5.27}$$

or, equivalently,

$$H(S) \leq \frac{\bar{L}_n}{n} < H(S) + \frac{1}{n} \tag{5.28}$$

In the limit, as n approaches infinity, the lower and upper bounds in (5.28) converge as shown by

$$\lim_{n \to \infty} \frac{1}{n} \bar{L}_n = H(S) \tag{5.29}$$

We may, therefore, make the statement:

> By making the order n of an extended prefix source encoder large enough, we can make the code faithfully represent the discrete memoryless source S as closely as desired.

In other words, the average codeword length of an extended prefix code can be made as small as the entropy of the source, provided that the extended code has a high enough order in accordance with the source-coding theorem. However, the price we have to pay for decreasing the average codeword length is increased decoding complexity, which is brought about by the high order of the extended prefix code.

Huffman Coding

We next describe an important class of prefix codes known as Huffman codes. The basic idea behind *Huffman coding*[6] is the construction of a simple algorithm that computes an *optimal* prefix code for a given distribution, optimal in the sense that the code has the *shortest expected length*. The end result is a source code whose average codeword length approaches the fundamental limit set by the entropy of a discrete memoryless source, namely $H(S)$. The essence of the *algorithm* used to synthesize the Huffman code is to replace the prescribed set of source statistics of a discrete memoryless source with a simpler one. This *reduction* process is continued in a step-by-step manner until we are left with a final set of only two source statistics (symbols), for which (0, 1) is an optimal code. Starting from this trivial code, we then work backward and thereby construct the Huffman code for the given source.

To be specific, the Huffman *encoding algorithm* proceeds as follows:

1. The source symbols are listed in order of decreasing probability. The two source symbols of lowest probability are assigned 0 and 1. This part of the step is referred to as the *splitting stage*.
2. These two source symbols are then *combined* into a new source symbol with probability equal to the sum of the two original probabilities. (The list of source symbols, and, therefore, source statistics, is thereby *reduced* in size by one.) The probability of the new symbol is placed in the list in accordance with its value.
3. The procedure is repeated until we are left with a final list of source statistics (symbols) of only two for which the symbols 0 and 1 are assigned.

The code for each (original) source is found by working backward and tracing the sequence of 0s and 1s assigned to that symbol as well as its successors.

EXAMPLE 4

Huffman Tree

To illustrate the construction of a Huffman code, consider the five symbols of the alphabet of a discrete memoryless source and their probabilities, which are shown in the two leftmost columns of Figure 5.5b. Following through the Huffman algorithm, we reach the end of the computation in four steps, resulting in a *Huffman tree* similar to that shown in Figure 5.5; the Huffman tree is not to be confused with the decision tree discussed previously in Figure 5.4. The codewords of the Huffman code for the source are tabulated in Figure 5.5a. The average codeword length is, therefore,

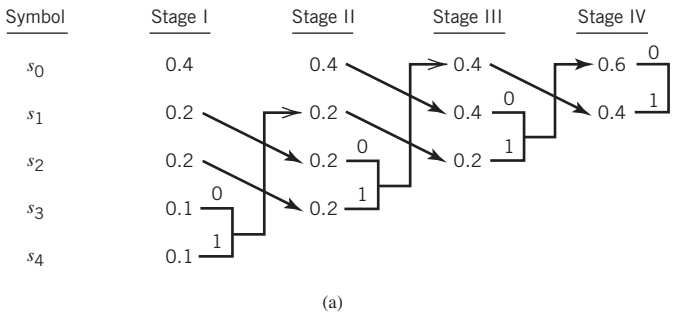
$$\begin{aligned}\bar{L} &= 0.4(2) + 0.2(2) + 0.2(2) + 0.1(3) + 0.1(3) \\ &= 2.2 \text{ binary symbols}\end{aligned}$$

The entropy of the specified discrete memoryless source is calculated as follows (see (5.9)):

$$\begin{aligned}H(S) &= 0.4 \log_2\left(\frac{1}{0.4}\right) + 0.2 \log_2\left(\frac{1}{0.2}\right) + 0.2 \log_2\left(\frac{1}{0.2}\right) + 0.1 \log_2\left(\frac{1}{0.1}\right) + 0.1 \log_2\left(\frac{1}{0.1}\right) \\ &= 0.529 + 0.464 + 0.464 + 0.332 + 0.332 \\ &= 2.121 \text{ bits}\end{aligned}$$

For this example, we may make two observations:

1. The average codeword length $\bar{L}$ exceeds the entropy $H(S)$ by only 3.67%.
2. The average codeword length $\bar{L}$ does indeed satisfy (5.23).



(a)

Symbol	Probability	Code word
s_0	0.4	00
s_1	0.2	10
s_2	0.2	11
s_3	0.1	010
s_4	0.1	011

(b)

Figure 5.5 (a) Example of the Huffman encoding algorithm. (b) Source code.

It is noteworthy that the Huffman encoding process (i.e., the Huffman tree) is not unique. In particular, we may cite two variations in the process that are responsible for the nonuniqueness of the Huffman code. First, at each splitting stage in the construction of a Huffman code, there is arbitrariness in the way the symbols 0 and 1 are assigned to the last two source symbols. Whichever way the assignments are made, however, the resulting differences are trivial. Second, ambiguity arises when the probability of a *combined*

symbol (obtained by adding the last two probabilities pertinent to a particular step) is found to equal another probability in the list. We may proceed by placing the probability of the new symbol as *high* as possible, as in Example 4. Alternatively, we may place it as *low* as possible. (It is presumed that whichever way the placement is made, high or low, it is consistently adhered to throughout the encoding process.) By this time, noticeable differences arise in that the codewords in the resulting source code can have different lengths. Nevertheless, the average codeword length remains the same.

As a measure of the variability in codeword lengths of a source code, we define the *variance* of the average codeword length $\bar{L}$ over the ensemble of source symbols as

$$\sigma^2 = \sum_{k=0}^{K-1} p_k (l_k - \bar{L})^2 \tag{5.30}$$

where $p_0, p_1, \ldots, p_{K-1}$ are the source statistics and l_k is the length of the codeword assigned to source symbol s_k. It is usually found that when a combined symbol is moved as high as possible, the resulting Huffman code has a significantly smaller variance σ^2 than when it is moved as low as possible. On this basis, it is reasonable to choose the former Huffman code over the latter.

Lempel–Ziv Coding

A drawback of the Huffman code is that it requires knowledge of a probabilistic model of the source; unfortunately, in practice, source statistics are not always known a priori. Moreover, in the modeling of text we find that storage requirements prevent the Huffman code from capturing the higher-order relationships between words and phrases because the codebook grows exponentially fast in the size of each super-symbol of letters (i.e., grouping of letters); the efficiency of the code is therefore compromised. To overcome these practical limitations of Huffman codes, we may use the *Lempel–Ziv algorithm*,[7] which is intrinsically *adaptive* and simpler to implement than Huffman coding.

Basically, the idea behind encoding in the Lempel–Ziv algorithm is described as follows:

> The source data stream is parsed into segments that are the shortest subsequences not encountered previously.

To illustrate this simple yet elegant idea, consider the example of the binary sequence

000101110010100101 …

It is assumed that the binary symbols 0 and 1 are already stored in that order in the code book. We thus write

Subsequences stored: 0, 1
Data to be parsed: 000101110010100101 …

The encoding process begins at the left. With symbols 0 and 1 already stored, the *shortest subsequence* of the data stream encountered for the first time and not seen before is 00; so we write

Subsequences stored: 0, 1, 00
Data to be parsed: 0101110010100101 …

The second shortest subsequence not seen before is 01; accordingly, we go on to write

Subsequences stored: 0, 0, 00, 01
Data to be parsed: 01110010100101 …

The next shortest subsequence not encountered previously is 011; hence, we write

Subsequences stored: 0, 1, 00, 01, 011
Data to be parsed: 10010100101 …

We continue in the manner described here until the given data stream has been completely parsed. Thus, for the example at hand, we get the *code book* of binary subsequences shown in the second row of Figure 5.6.[8]

The first row shown in this figure merely indicates the numerical positions of the individual subsequences in the code book. We now recognize that the first subsequence of the data stream, 00, is made up of the concatenation of the *first* code book entry, 0, with itself; it is, therefore, represented by the number 11. The second subsequence of the data stream, 01, consists of the *first* code book entry, 0, concatenated with the *second* code book entry, 1; it is, therefore, represented by the number 12. The remaining subsequences are treated in a similar fashion. The complete set of numerical representations for the various subsequences in the code book is shown in the third row of Figure 5.6. As a further example illustrating the composition of this row, we note that the subsequence 010 consists of the concatenation of the subsequence 01 in position 4 and symbol 0 in position 1; hence, the numerical representation is 41. The last row shown in Figure 5.6 is the binary encoded representation of the different subsequences of the data stream.

The last symbol of each subsequence in the code book (i.e., the second row of Figure 5.6) is an *innovation symbol*, which is so called in recognition of the fact that its appendage to a particular subsequence distinguishes it from all previous subsequences stored in the code book. Correspondingly, the last bit of each uniform block of bits in the binary encoded representation of the data stream (i.e., the fourth row in Figure 5.6) represents the innovation symbol for the particular subsequence under consideration. The remaining bits provide the equivalent binary representation of the "pointer" to the *root subsequence* that matches the one in question, except for the innovation symbol.

The *Lempel–Ziv decoder* is just as simple as the encoder. Specifically, it uses the pointer to identify the root subsequence and then appends the innovation symbol. Consider, for example, the binary encoded block 1101 in position 9. The last bit, 1, is the innovation symbol. The remaining bits, 110, point to the root subsequence 10 in position 6. Hence, the block 1101 is decoded into 101, which is correct.

From the example described here, we note that, in contrast to Huffman coding, the Lempel–Ziv algorithm uses fixed-length codes to represent a variable number of source symbols; this feature makes the Lempel–Ziv code suitable for synchronous transmission.

Numerical positions	1	2	3	4	5	6	7	8	9
Subsequences	0	1	00	01	011	10	010	100	101
Numerical representations			11	12	42	21	41	61	62
Binary encoded blocks			0010	0011	1001	0100	1000	1100	1101

Figure 5.6 Illustrating the encoding process performed by the Lempel–Ziv algorithm on the binary sequence 000101110010100101 …

In practice, fixed blocks of 12 bits long are used, which implies a code book of $2^{12} = 4096$ entries.

For a long time, Huffman coding was unchallenged as the algorithm of choice for lossless data compression; Huffman coding is still optimal, but in practice it is hard to implement. It is on account of practical implementation that the Lempel–Ziv algorithm has taken over almost completely from the Huffman algorithm. The Lempel–Ziv algorithm is now the standard algorithm for file compression.

5.5 Discrete Memoryless Channels

Up to this point in the chapter we have been preoccupied with discrete memoryless sources responsible for *information generation*. We next consider the related issue of *information transmission*. To this end, we start the discussion by considering a discrete memoryless channel, the counterpart of a discrete memoryless source.

A *discrete memoryless channel* is a statistical model with an input X and an output Y that is a *noisy* version of X; both X and Y are random variables. Every unit of time, the channel accepts an input symbol X selected from an alphabet $\mathcal{X}$ and, in response, it emits an output symbol Y from an alphabet $\mathcal{Y}$. The channel is said to be "discrete" when both of the alphabets $\mathcal{X}$ and $\mathcal{Y}$ have *finite* sizes. It is said to be "memoryless" when the current output symbol depends *only* on the current input symbol and *not* any previous or future symbol.

Figure 5.7a shows a view of a discrete memoryless channel. The channel is described in terms of an *input alphabet*

$$\mathcal{X} = \{x_0, x_1, \ldots, x_{J-1}\} \tag{5.31}$$

and an *output alphabet*

$$\mathcal{Y} = \{y_0, y_1, \ldots, y_{K-1}\} \tag{5.32}$$

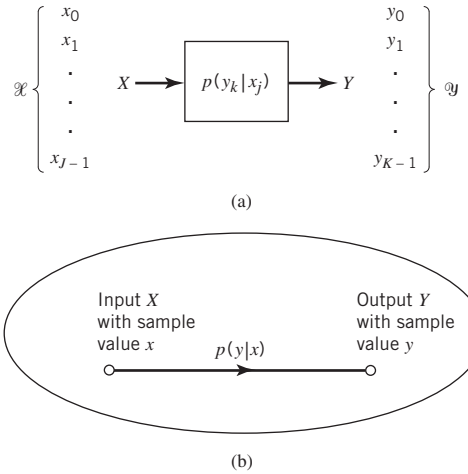


Figure 5.7 (a) Discrete memoryless channel; (b) Simplified graphical representation of the channel.

The *cardinality* of the alphabets $\mathcal{X}$ and $\mathcal{Y}$, or any other alphabet for that matter, is defined as the number of elements in the alphabet. Moreover, the channel is characterized by a set of *transition probabilities*

$$p(y_k|x_j) = \mathbb{P}(Y = y_k|X = x_j) \quad \text{for all } j \text{ and } k \tag{5.33}$$

for which, according to probability theory, we naturally have

$$0 \le p(y_k|x_j) \le 1 \quad \text{for all } j \text{ and } k \tag{5.34}$$

and

$$\sum_k p(y_k|x_j) = 1 \quad \text{for fixed } j \tag{5.35}$$

When the number of input symbols J and the number of output symbols K are not large, we may depict the discrete memoryless channel graphically in another way, as shown in Figure 5.7b. In this latter depiction, each input–output symbol pair (x, y), characterized by the transition probability $p(y|x) > 0$, is joined together by a line labeled with the number $p(y|x)$.

Also, the input alphabet $\mathcal{X}$ and output alphabet $\mathcal{Y}$ need not have the same size; hence the use of J for the size of $\mathcal{X}$ and K for the size of $\mathcal{Y}$. For example, in channel coding, the size K of the output alphabet $\mathcal{Y}$ may be larger than the size J of the input alphabet $\mathcal{X}$; thus, $K \ge J$. On the other hand, we may have a situation in which the channel emits the same symbol when either one of two input symbols is sent, in which case we have $K \le J$.

A convenient way of describing a discrete memoryless channel is to arrange the various transition probabilities of the channel in the form of a *matrix*

$$\mathbf{P} = \begin{bmatrix} p(y_0|x_0) & p(y_1|x_0) & \cdots & p(y_{K-1}|x_0) \\ p(y_0|x_1) & p(y_1|x_1) & \cdots & p(y_{K-1}|x_1) \\ \vdots & \vdots & & \vdots \\ p(y_0|x_{J-1}) & p(y_1|x_{J-1}) & \cdots & p(y_{K-1}|x_{J-1}) \end{bmatrix} \tag{5.36}$$

The J-by-K matrix $\mathbf{P}$ is called the *channel matrix*, or *stochastic matrix*. Note that each *row* of the channel matrix $\mathbf{P}$ corresponds to a *fixed channel input*, whereas each column of the matrix corresponds to a *fixed channel output*. Note also that a fundamental property of the channel matrix $\mathbf{P}$, as defined here, is that the sum of the elements along any row of the stochastic matrix is always equal to one, according to (5.35).

Suppose now that the inputs to a discrete memoryless channel are selected according to the probability distribution $\{p(x_j), j = 0, 1, \ldots, J-1\}$. In other words, the event that the channel input $X = x_j$ occurs with probability

$$p(x_j) = \mathbb{P}(X = x_j) \quad \text{for } j = 0, 1, \ldots, J-1 \tag{5.37}$$

Having specified the random variable X denoting the channel input, we may now specify the second random variable Y denoting the channel output. The *joint probability distribution* of the random variables X and Y is given by

$$\begin{aligned} p(x_j, y_k) &= \mathbb{P}(X = x_j, Y = y_k) \\ &= \mathbb{P}(Y = y_k|X = x_j)\mathbb{P}(X = x_j) \\ &= p(y_k|x_j)p(x_j) \end{aligned} \tag{5.38}$$

The *marginal probability distribution* of the output random variable Y is obtained by averaging out the dependence of $p(x_j, y_k)$ on x_j, obtaining

$$\begin{aligned} p(y_k) &= \mathbb{P}(Y = y_k) \\ &= \sum_{j=0}^{J-1} \mathbb{P}(Y = y_k | X = x_j)\mathbb{P}(X = x_j) \\ &= \sum_{j=0}^{J-1} p(y_k | x_j)p(x_j) \qquad \text{for } k = 0, 1, \ldots, K-1 \end{aligned} \tag{5.39}$$

The probabilities $p(x_j)$ for $j = 0, 1, \ldots, J-1$, are known as the *prior probabilities* of the various input symbols. Equation (5.39) states:

> If we are given the input prior probabilities $p(x_j)$ and the stochastic matrix (i.e., the matrix of transition probabilities $p(y_k|x_j)$), then we may calculate the probabilities of the various output symbols, the $p(y_k)$.

EXAMPLE 5 **Binary Symmetric Channel**

The *binary symmetric channel* is of theoretical interest and practical importance. It is a special case of the discrete memoryless channel with $J = K = 2$. The channel has two input symbols ($x_0 = 0$, $x_1 = 1$) and two output symbols ($y_0 = 0$, $y_1 = 1$). The channel is symmetric because the probability of receiving 1 if 0 is sent is the same as the probability of receiving 0 if 1 is sent. This conditional probability of error is denoted by p (i.e., the probability of a bit flipping). The *transition probability diagram* of a binary symmetric channel is as shown in Figure 5.8. Correspondingly, we may express the stochastic matrix as

$$\mathbf{P} = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}$$

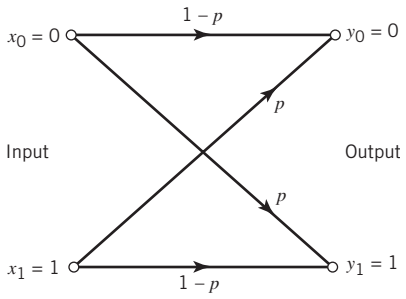


Figure 5.8 Transition probability diagram of binary symmetric channel.

5.6 Mutual Information

Given that we think of the channel output Y (selected from alphabet $\mathcal{Y}$) as a noisy version of the channel input X (selected from alphabet $\mathcal{X}$) and that the entropy $H(X)$ is a measure of the prior uncertainty about X, how can we measure the uncertainty about X after observing Y? To answer this basic question, we extend the ideas developed in Section 5.2 by defining the *conditional entropy* of X selected from alphabet $\mathcal{X}$, given $Y = y_k$. Specifically, we write

$$H(X|Y = y_k) = \sum_{j=0}^{J-1} p(x_j|y_k)\log_2\left(\frac{1}{p(x_j|y_k)}\right) \tag{5.40}$$

This quantity is itself a random variable that takes on the values $H(X|Y = y_0)$, …, $H(X|Y = y_{K-1})$ with probabilities $p(y_0)$, …, $p(y_{K-1})$, respectively. The expectation of entropy $H(X|Y = y_k)$ over the output alphabet $\mathcal{Y}$ is therefore given by

$$\begin{aligned} H(X|Y) &= \sum_{k=0}^{K-1} H(X|Y = y_k)p(y_k) \\ &= \sum_{k=0}^{K-1}\sum_{j=0}^{J-1} p(x_j|y_k)p(y_k)\,\log_2\left(\frac{1}{p(x_j|y_k)}\right) \\ &= \sum_{k=0}^{K-1}\sum_{j=0}^{J-1} p(x_j, y_k)\,\log_2\left(\frac{1}{p(x_j|y_k)}\right) \end{aligned} \tag{5.41}$$

where, in the last line, we used the definition of the probability of the joint event ($X = x_j$, $Y = y_k$) as shown by

$$p(x_j, y_k) = p(x_j|y_k)p(y_k) \tag{5.42}$$

The quantity $H(X|Y)$ in (5.41) is called the *conditional entropy,* formally defined as follows:

> The conditional entropy, $H(X|Y)$, is the average amount of uncertainty remaining about the channel input after the channel output has been observed.

The conditional entropy $H(X|Y)$ relates the channel output Y to the channel input X. The entropy $H(X)$ defines the entropy of the channel input X by itself. Given these two entropies, we now introduce the definition

$$I(X;Y) = H(X) - H(X|Y) \tag{5.43}$$

which is called the *mutual information* of the channel. To add meaning to this new concept, we recognize that the entropy $H(X)$ accounts for the uncertainty about the channel input *before* observing the channel output and the conditional entropy $H(X|Y)$ accounts for the uncertainty about the channel input *after* observing the channel output. We may, therefore, go on to make the statement:

> The mutual information $I(X;Y)$ is a measure of the uncertainty about the channel input, which is resolved by observing the channel output.

Equation (5.43) is not the only way of defining the mutual information of a channel. Rather, we may define it in another way, as shown by

$$I(Y;X) = H(Y) - H(Y|X) \tag{5.44}$$

on the basis of which we may make the next statement:

> The mutual information $I(Y;X)$ is a measure of the uncertainty about the channel output that is resolved by *sending* the channel input.

On first sight, the two definitions of (5.43) and (5.44) look different. In reality, however, they embody equivalent statements on the mutual information of the channel that are worded differently. More specifically, they could be used interchangeably, as demonstrated next.

Properties of Mutual Information

PROPERTY 1 Symmetry

The mutual information of a channel is symmetric in the sense that

$$I(X;Y) = I(Y;X) \tag{5.45}$$

To prove this property, we first use the formula for entropy and then use (5.35) and (5.38), in that order, obtaining

$$\begin{aligned} H(X) &= \sum_{j=0}^{J-1} p(x_j) \log_2\left(\frac{1}{p(x_j)}\right) \\ &= \sum_{j=0}^{J-1} p(x_j) \log_2\left(\frac{1}{p(x_j)}\right) \sum_{k=0}^{K-1} p(y_k|x_j) \\ &= \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(y_k|x_j)p(x_j) \log_2\left(\frac{1}{p(x_j)}\right) \\ &= \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(x_j, y_k) \log_2\left(\frac{1}{p(x_j)}\right) \end{aligned} \tag{5.46}$$

where, in going from the third to the final line, we made use of the definition of a joint probability. Hence, substituting (5.41) and (5.46) into (5.43) and then combining terms, we obtain

$$I(X;Y) = \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(x_j, y_k) \log_2\left(\frac{p(x_j|y_k)}{p(x_j)}\right) \tag{5.47}$$

Note that the double summation on the right-hand side of (5.47) is *invariant* with respect to swapping the x and y. In other words, the symmetry of the mutual information $I(X;Y)$ is already evident from (5.47).

To further confirm this property, we may use *Bayes' rule* for conditional probabilities, previously discussed in Chapter 3, to write

$$\frac{p(x_j|y_k)}{p(x_j)} = \frac{p(y_k|x_j)}{p(y_k)} \tag{5.48}$$

Hence, substituting (5.48) into (5.47) and interchanging the order of summation, we get

$$\begin{aligned} I(X;Y) &= \sum_{k=0}^{K-1}\sum_{j=0}^{J-1} p(x_j, y_k)\log_2\left(\frac{p(y_k|x_j)}{p(y_k)}\right) \\ &= I(Y;X) \end{aligned} \tag{5.49}$$

which proves Property 1.

PROPERTY 2 **Nonnegativity**

The mutual information is always nonnegative; that is;

$$I(X;Y) \geq 0 \tag{5.50}$$

To prove this property, we first note from (5.42) that

$$p(x_j|y_k) = \frac{p(x_j, y_k)}{p(y_k)} \tag{5.51}$$

Hence, substituting (5.51) into (5.47), we may express the mutual information of the channel as

$$I(X;Y) = \sum_{j=0}^{J-1}\sum_{k=0}^{K-1} p(x_j, y_k)\log_2\left(\frac{p(x_j, y_k)}{p(x_j)p(y_k)}\right) \tag{5.52}$$

Next, a direct application of the fundamental inequality of (5.12) on relative entropy confirms (5.50), with equality if, and only if,

$$p(x_j, y_k) = p(x_j)p(y_k) \qquad \text{for all } j \text{ and } k \tag{5.53}$$

In words, Property 2 states the following:

> We cannot lose information, on the average, by observing the output of a channel.

Moreover, the mutual information is zero if, and only if, the input and output symbols of the channel are statistically independent; that is, when (5.53) is satisfied.

PROPERTY 3 **Expansion of the Mutual Information**

The mutual information of a channel is related to the joint entropy of the channel input and channel output by

$$I(X;Y) = H(X) + H(Y) - H(X, Y) \tag{5.54}$$

where the joint entropy H(X, Y) is defined by

$$H(X, Y) = \sum_{j=0}^{J-1}\sum_{k=0}^{K-1} p(x_j, y_k)\log_2\left(\frac{1}{p(x_j, y_k)}\right) \tag{5.55}$$

To prove (5.54), we first rewrite the joint entropy in the equivalent form

$$H(X, Y) = \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(x_j, y_k) \log_2\left(\frac{p(x_j)p(y_k)}{p(x_j, y_k)}\right) + \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(x_j, y_k) \log_2\left(\frac{1}{p(x_j)p(y_k)}\right) \tag{5.56}$$

The first double summation term on the right-hand side of (5.56) is recognized as the negative of the mutual information of the channel, $I(X;Y)$, previously given in (5.52). As for the second summation term, we manipulate it as follows:

$$\begin{aligned}
\sum_{j=0}^{J-1} \sum_{k=0}^{K-1} p(x_j, y_k) \log_2\left(\frac{1}{p(x_j)p(y_k)}\right) &= \sum_{j=0}^{J-1} \log_2\left(\frac{1}{p(x_j)}\right) \sum_{k=0}^{K-1} p(x_j, y_k) \\
&\quad + \sum_{k=0}^{K-1} \log_2\left(\frac{1}{p(y_k)}\right) \sum_{j=0}^{J-1} p(x_j, y_k) \\
&= \sum_{j=0}^{J-1} p(x_j) \log_2\left(\frac{1}{p(x_j)}\right) + \sum_{k=0}^{K-1} p(y_k) \log_2\left(\frac{1}{p(y_k)}\right) \\
&= H(X) + H(Y)
\end{aligned} \tag{5.57}$$

where, in the first line, we made use of the following relationship from probability theory:

$$\sum_{k=0}^{K-1} p(x_j, y_k) = p(y_k)$$

and a similar relationship holds for the second line of the equation.

Accordingly, using (5.52) and (5.57) in (5.56), we get the result

$$H(X, Y) = -I(X;Y) + H(X) + H(Y) \tag{5.58}$$

which, on rearrangement, proves Property 3.

We conclude our discussion of the mutual information of a channel by providing a diagramatic interpretation in Figure 5.9 of (5.43), (5.44), and (5.54).

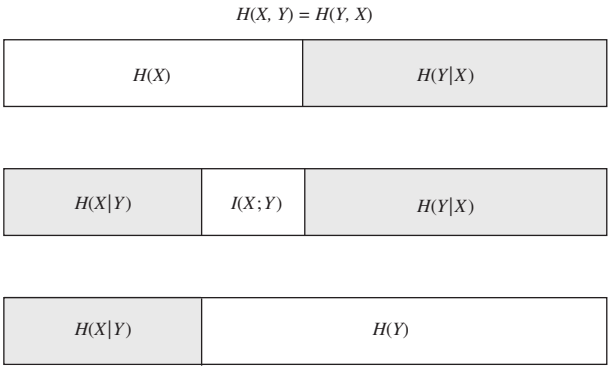


Figure 5.9 Illustrating the relations among various channel entropies.

5.7 Channel Capacity

The concept of entropy introduced in Section 5.2 prepared us for formulating Shannon's first theorem: the source-coding theorem. To set the stage for formulating Shannon's second theorem, namely the channel-coding theorem, this section introduces the concept of *capacity*, which, as mentioned previously, defines the intrinsic ability of a communication channel to convey information.

To proceed, consider a discrete memoryless channel with input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$, and transition probabilities $p(y_k|x_j)$, where $j = 0, 1, \ldots, J-1$ and $k = 0, 1, \ldots, K-1$. The mutual information of the channel is defined by the first line of (5.49), which is reproduced here for convenience:

$$I(X;Y) = \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} p(x_j, y_k) \log_2\left(\frac{p(y_k|x_j)}{p(y_k)}\right)$$

where, according to (5.38),

$$p(x_j, y_k) = p(y_k|x_j)p(x_j)$$

Also, from (5.39), we have

$$p(y_k) = \sum_{j=0}^{J-1} p(y_k|x_j)p(x_j)$$

Putting these three equations into a single equation, we write

$$I(X;Y) = \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} p(y_k, x_j)p(x_j) \log_2\left(\frac{p(y_k|x_j)}{\sum_{j=0}^{J-1} p(y_k|x_j)p(x_j)}\right)$$

Careful examination of the double summation in this equation reveals two different probabilities, on which the essence of mutual information $I(X;Y)$ depends:

- the probability distribution $\{p(x_j)\}_{j=0}^{J-1}$ that characterizes the channel input and
- the transition probability distribution $\{p(y_k|x_j)\}_{j=0,\,k=0}^{j=J-1,\,K-1}$ that characterizes the channel itself.

These two probability distributions are obviously independent of each other. Thus, given a channel characterized by the transition probability distribution $\{p(y_k|x_j\}$, we may now introduce the *channel capacity*, which is formally defined in terms of the mutual information between the channel input and output as follows:

$$C = \max_{\{p(x_j)\}} I(X;Y) \qquad \text{bits per channel use} \tag{5.59}$$

The maximization in (5.59) is performed, subject to two input probabilistic constraints:

$$p(x_j) \geq 0 \qquad \text{for all } j$$

and

$$\sum_{j=0}^{J-1} p(x_j) = 1$$

Accordingly, we make the following statement:

> The channel capacity of a discrete memoryless channel, commonly denoted by C, is defined as the maximum mutual information $I(X;Y)$ in any single use of the channel (i.e., signaling interval), where the maximization is over all possible input probability distributions $\{p(x_j)\}$ on X.

The channel capacity is clearly an intrinsic property of the channel.

EXAMPLE 6 **Binary Symmetric Channel (Revisited)**

Consider again the *binary symmetric channel*, which is described by the *transition probability diagram* of Figure 5.8. This diagram is uniquely defined by the conditional probability of error p.

From Example 1 we recall that the entropy $H(X)$ is maximized when the channel input probability $p(x_0) = p(x_1) = 1/2$, where x_0 and x_1 are each 0 or 1. Hence, invoking the defining equation (5.59), we find that the mutual information $I(X;Y)$ is similarly maximized and thus write

$$C = I(X;Y)\big|_{p(x_0) = p(x_1) = 1/2}$$

From Figure 5.8 we have

$$p(y_0|x_1) = p(y_1|x_0) = p$$

and

$$p(y_0|x_0) = p(y_1|x_1) = 1 - p$$

Therefore, substituting these channel transition probabilities into (5.49) with $J = K = 2$ and then setting the input probability $p(x_0) = p(x_1) = 1/2$ in (5.59), we find that the capacity of the binary symmetric channel is

$$C = 1 + p \log_2 p + (1 - p) \log_2 (1 - p) \tag{5.60}$$

Moreover, using the definition of the entropy function introduced in (5.16), we may reduce (5.60) to

$$C = 1 - H(p)$$

The channel capacity C varies with the probability of error (i.e., transition probability) p in a convex manner as shown in Figure 5.10, which is symmetric about $p = 1/2$. Comparing the curve in this figure with that in Figure 5.2, we make two observations:

1. When the channel is *noise free*, permitting us to set $p = 0$, the channel capacity C attains its maximum value of one bit per channel use, which is exactly the information in each channel input. At this value of p, the entropy function $H(p)$ attains its minimum value of zero.
2. When the conditional probability of error $p = 1/2$ due to channel noise, the channel capacity C attains its minimum value of zero, whereas the entropy function $H(p)$

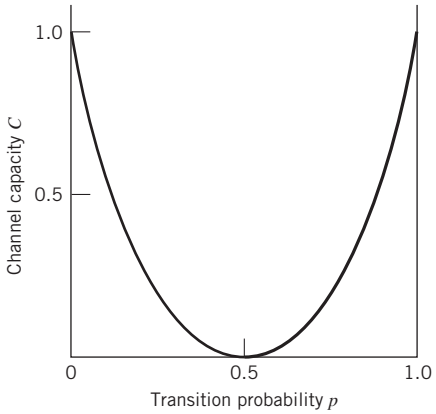


Figure 5.10
Variation of channel capacity of a binary symmetric channel with transition probability p.

attains its maximum value of unity; in such a case, the channel is said to be *useless* in the sense that the channel input and output assume statistically independent structures.

5.8 Channel-coding Theorem

With the entropy of a discrete memoryless source and the corresponding capacity of a discrete memoryless channel at hand, we are now equipped with the concepts needed for formulating Shannon's second theorem: the channel-coding theorem.

To this end, we first recognize that the inevitable presence of *noise* in a channel causes discrepancies (errors) between the output and input data sequences of a digital communication system. For a relatively noisy channel (e.g., wireless communication channel), the probability of error may reach a value as high as 10^{-1}, which means that (on the average) only 9 out of 10 transmitted bits are received correctly. For many applications, this *level of reliability* is utterly unacceptable. Indeed, a probability of error equal to 10^{-6} or even lower is often a necessary practical requirement. To achieve such a high level of performance, we resort to the use of channel coding.

The design goal of channel coding is to increase the resistance of a digital communication system to channel noise. Specifically, *channel coding* consists of *mapping* the incoming data sequence into a channel input sequence and *inverse mapping* the channel output sequence into an output data sequence in such a way that the overall effect of channel noise on the system is minimized. The first mapping operation is performed in the transmitter by a *channel encoder*, whereas the inverse mapping operation is performed in the receiver by a *channel decoder*, as shown in the block diagram of Figure 5.11; to simplify the exposition, we have not included source encoding (before channel encoding) and source decoding (after channel decoding) in this figure.[9]

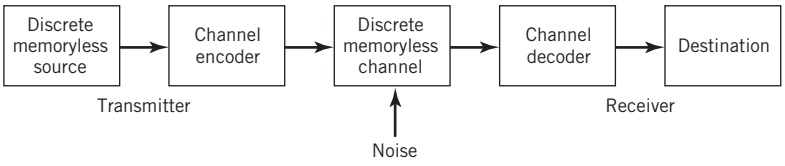


Figure 5.11
Block diagram of digital communication system.

The channel encoder and channel decoder in Figure 5.11 are both under the designer's control and should be designed to optimize the overall reliability of the communication system. The approach taken is to introduce *redundancy* in the channel encoder in a controlled manner, so as to reconstruct the original source sequence as accurately as possible. In a rather loose sense, we may thus view channel coding as the *dual* of source coding, in that the former introduces controlled redundancy to improve reliability whereas the latter reduces redundancy to improve efficiency.

Treatment of the channel-coding techniques is deferred to Chapter 10. For the purpose of our present discussion, it suffices to confine our attention to *block codes*. In this class of codes, the message sequence is subdivided into sequential blocks each k bits long, and each k-bit block is *mapped* into an n-bit block, where $n > k$. The number of redundant bits added by the encoder to each transmitted block is $n - k$ bits. The ratio k/n is called the *code rate*. Using r to denote the code rate, we write

$$r = \frac{k}{n} \tag{5.61}$$

where, of course, r is less than unity. For a prescribed k, the code rate r (and, therefore, the system's coding efficiency) approaches zero as the block length n approaches infinity.

The accurate reconstruction of the original source sequence at the destination requires that the *average probability of symbol error* be arbitrarily low. This raises the following important question:

> Does a channel-coding scheme exist such that the probability that a message bit will be in error is less than any positive number ε (i.e., as small as we want it), and yet the channel-coding scheme is efficient in that the code rate need not be too small?

The answer to this fundamental question is an emphatic "yes." Indeed, the answer to the question is provided by Shannon's second theorem in terms of the channel capacity C, as described in what follows.

Up until this point, *time* has not played an important role in our discussion of channel capacity. Suppose then the discrete memoryless source in Figure 5.11 has the source alphabet $\mathcal{S}$ and entropy $H(S)$ bits per source symbol. We assume that the source emits symbols once every T_s seconds. Hence, the *average information rate* of the source is $H(S)/T_s$ bits per second. The decoder delivers decoded symbols to the destination from the source alphabet S and at the same source rate of one symbol every T_s seconds. The discrete memoryless channel has a channel capacity equal to C bits per use of the channel. We assume that the channel is capable of being used once every T_c seconds. Hence, the *channel capacity per unit time* is C/T_c bits per second, which represents the maximum rate of information transfer over the channel. With this background, we are now ready to state Shannon's second theorem, the *channel-coding theorem,*[10] in two parts as follows:

1. Let a discrete memoryless source with an alphabet $\mathcal{S}$ have entropy $H(S)$ for random variable S and produce symbols once every T_s seconds. Let a discrete memoryless channel have capacity C and be used once every T_c seconds, Then, if

$$\frac{H(S)}{T_s} \leq \frac{C}{T_c} \tag{5.62}$$

there exists a coding scheme for which the source output can be transmitted over the channel and be reconstructed with an arbitrarily small probability of error. The parameter C/T_c is called the *critical rate*; when (5.62) is satisfied with the equality sign, the system is said to be signaling at the critical rate.

2. Conversely, if

$$\frac{H(S)}{T_s} > \frac{C}{T_c}$$

it is not possible to transmit information over the channel and reconstruct it with an arbitrarily small probability of error.

The channel-coding theorem is the single most important result of information theory. The theorem specifies the channel capacity C as a *fundamental limit* on the rate at which the transmission of reliable error-free messages can take place over a discrete memoryless channel. However, it is important to note two limitations of the theorem:

1. The channel-coding theorem does not show us how to construct a good code. Rather, the theorem should be viewed as an *existence proof* in the sense that it tells us that if the condition of (5.62) is satisfied, then good codes do exist. Later, in Chapter 10, we describe good codes for discrete memoryless channels.
2. The theorem does not have a precise result for the probability of symbol error after decoding the channel output. Rather, it tells us that the probability of symbol error tends to zero as the length of the code increases, again provided that the condition of (5.62) is satisfied.

Application of the Channel-coding Theorem to Binary Symmetric Channels

Consider a discrete memoryless source that emits equally likely binary symbols (0s and 1s) once every T_s seconds. With the source entropy equal to one bit per source symbol (see Example 1), the information rate of the source is $(1/T_s)$ bits per second. The source sequence is applied to a channel encoder with code rate r. The channel encoder produces a symbol once every T_c seconds. Hence, the encoded symbol transmission rate is $(1/T_c)$ symbols per second. The channel encoder engages a binary symmetric channel once every T_c seconds. Hence, the channel capacity per unit time is (C/T_c) bits per second, where C is determined by the prescribed channel transition probability p in accordance with (5.60). Accordingly, part (1) of the channel-coding theorem implies that if

$$\frac{1}{T_s} \leq \frac{C}{T_c} \tag{5.63}$$

then the probability of error can be made arbitrarily low by the use of a suitable channel-encoding scheme. But the ratio T_c/T_s equals the code rate of the channel encoder:

$$r = \frac{T_c}{T_s} \tag{5.64}$$

Hence, we may restate the condition of (5.63) simply as

$$r \leq C$$

That is, for $r \leq C$, there exists a code (with code rate less than or equal to channel capacity C) capable of achieving an arbitrarily low probability of error.

EXAMPLE 7 **Repetition Code**

In this example we present a graphical interpretation of the channel-coding theorem. We also bring out a surprising aspect of the theorem by taking a look at a simple coding scheme.

Consider first a binary symmetric channel with transition probability $p = 10^{-2}$. For this value of p, we find from (5.60) that the channel capacity $C = 0.9192$. Hence, from the channel-coding theorem, we may state that, for any $\varepsilon > 0$ and $r \leq 0.9192$, there exists a code of large enough length n, code rate r, and an appropriate decoding algorithm such that, when the coded bit stream is sent over the given channel, the average probability of channel decoding error is less than ε. This result is depicted in Figure 5.12 for the limiting value $\varepsilon = 10^{-8}$.

To put the significance of this result in perspective, consider next a simple coding scheme that involves the use of a *repetition code*, in which each bit of the message is repeated several times. Let each bit (0 or 1) be repeated n times, where $n = 2m + 1$ is an odd integer. For example, for $n = 3$, we transmit 0 and 1 as 000 and 111, respectively.

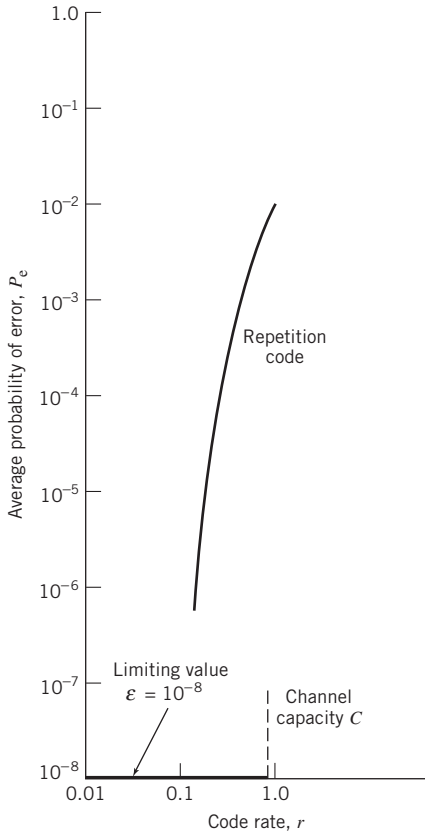


Figure 5.12 Illustrating the significance of the channel-coding theorem.

Intuitively, it would seem logical to use a *majority rule* for decoding, which operates as follows:

> If in a block of *n repeated* bits (representing one bit of the message) the number of 0s exceeds the number of 1s, the decoder decides in favor of a 0; otherwise, it decides in favor of a 1.

Hence, an error occurs when $m + 1$ or more bits out of $n = 2m + 1$ bits are received incorrectly. Because of the assumed symmetric nature of the channel, the *average probability of error,* denoted by P_e, is independent of the *prior* probabilities of 0 and 1. Accordingly, we find that P_e is given by

$$P_e = \sum_{i=m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{5.65}$$

where p is the transition probability of the channel.

Table 5.3 gives the average probability of error P_e for a repetition code that is calculated by using (5.65) for different values of the code rate r. The values given here assume the use of a binary symmetric channel with transition probability $p = 10^{-2}$. The improvement in reliability displayed in Table 5.3 is achieved at the cost of decreasing code rate. The results of this table are also shown plotted as the curve labeled "repetition code" in Figure 5.12. This curve illustrates the *exchange of code rate for message reliability*, which is a characteristic of repetition codes.

This example highlights the unexpected result presented to us by the channel-coding theorem. The result is that it is not necessary to have the code rate r approach zero (as in the case of repetition codes) to achieve more and more reliable operation of the communication link. The theorem merely requires that the code rate be less than the channel capacity C.

Table 5.3 Average probability of error for repetition code

Code rate, $r = 1/n$	Average probability of error, P_e
1	10^{-2}
$\frac{1}{3}$	3×10^{-4}
$\frac{1}{5}$	10^{-6}
$\frac{1}{7}$	4×10^{-7}
$\frac{1}{9}$	10^{-8}
$\frac{1}{11}$	5×10^{-10}

5.9 Differential Entropy and Mutual Information for Continuous Random Ensembles

The sources and channels considered in our discussion of information-theoretic concepts thus far have involved ensembles of random variables that are *discrete* in amplitude. In this section, we extend these concepts to *continuous* random variables. The motivation for doing so is to pave the way for the description of another fundamental limit in information theory, which we take up in Section 5.10.

Consider a continuous random variable X with the *probability density function* $f_X(x)$. By analogy with the entropy of a discrete random variable, we introduce the following definition:

$$h(X) = \int_{-\infty}^{\infty} f_X(x) \log_2\left[\frac{1}{f_X(x)}\right] dx \tag{5.66}$$

We refer to the new term $h(X)$ as the *differential entropy* of X to distinguish it from the ordinary or absolute entropy. We do so in recognition of the fact that, although $h(X)$ is a useful mathematical quantity to know, it is *not* in any sense a measure of the randomness of X. Nevertheless, we justify the use of (5.66) in what follows. We begin by viewing the continuous random variable X as the limiting form of a discrete random variable that assumes the value $x_k = k\Delta x$, where $k = 0, \pm 1, \pm 2, \ldots$, and Δx approaches zero. By definition, the continuous random variable X assumes a value in the interval $[x_k, x_k + \Delta x]$ with probability $f_X(x_k)\Delta x$. Hence, permitting Δx to approach zero, the ordinary entropy of the continuous random variable X takes the limiting form

$$\begin{aligned}
H(X) &= \lim_{\Delta x \to 0} \sum_{k=-\infty}^{\infty} f_X(x_k)\Delta x \log_2\left(\frac{1}{f_X(x_k)\Delta x}\right) \\
&= \lim_{\Delta x \to 0} \left(\sum_{k=-\infty}^{\infty} f_X(x_k) \log_2\left(\frac{1}{f_X(x_k)}\right)\Delta x - \log_2 \Delta x \sum_{k=-\infty}^{\infty} f_X(x_k)\Delta x \right) \\
&= \int_{-\infty}^{\infty} f_X(x) \log_2\left(\frac{1}{f_X(x)}\right) dx - \lim_{\Delta x \to 0}\left(\log_2 \Delta x \int_{-\infty}^{\infty} f_X(x_k)\, dx\right) \\
&= h(X) - \lim_{\Delta x \to 0} \log_2 \Delta x
\end{aligned} \tag{5.67}$$

In the last line of (5.67), use has been made of (5.66) and the fact that the total area under the curve of the probability density function $f_X(x)$ is unity. In the limit as Δx approaches zero, the term $-\log_2 \Delta x$ approaches infinity. This means that the entropy of a continuous random variable is infinitely large. Intuitively, we would expect this to be true because a continuous random variable may assume a value anywhere in the interval $(-\infty, \infty)$; we may, therefore, encounter uncountable infinite numbers of probable outcomes. To avoid the problem associated with the term $\log_2 \Delta x$, we adopt $h(X)$ as a *differential entropy*, with the term $-\log_2 \Delta x$ serving merely as a reference. Moreover, since the information transmitted over a channel is actually the difference between two entropy terms that have a common reference, the information will be the same as the difference between the corresponding

differential entropy terms. We are, therefore, perfectly justified in using the term $h(X)$, defined in (5.66), as the differential entropy of the continuous random variable X.

When we have a continuous random vector $\mathbf{X}$ consisting of n random variables $X_1, X_2, \ldots, X_n$, we define the differential entropy of $\mathbf{X}$ as the *n-fold integral*

$$h(\mathbf{X}) = \int_{-\infty}^{\infty} f_{\mathbf{X}}(\mathbf{x}) \log_2\left[\frac{1}{f_{\mathbf{X}}(\mathbf{x})}\right] d\mathbf{x} \tag{5.68}$$

where $f_{\mathbf{X}}(\mathbf{x})$ is the joint probability density function of $\mathbf{X}$.

EXAMPLE 8

Uniform Distribution

To illustrate the notion of differential entropy, consider a random variable X uniformly distributed over the interval $(0, a)$. The probability density function of X is

$$f_X(x) = \begin{cases} \frac{1}{a}, & 0 < x < a \\ 0, & \text{otherwise} \end{cases}$$

Applying (5.66) to this distribution, we get

$$\begin{aligned} h(X) &= \int_0^a \frac{1}{a} \log(a)\, dx \\ &= \log a \end{aligned} \tag{5.69}$$

Note that $\log a < 0$ for $a < 1$. Thus, this example shows that, unlike a discrete random variable, the differential entropy of a continuous random variable can assume a negative value.

Relative Entropy of Continuous Distributions

In (5.12) we defined the relative entropy of a pair of different discrete distributions. To extend that definition to a pair of continuous distributions, consider the continuous random variables X and Y whose respective probability density functions are denoted by $f_X(x)$ and $f_Y(x)$ for the same sample value (argument) x. The *relative entropy*[11] of the random variables X and Y is defined by

$$D(f_Y \| f_X) = \int_{-\infty}^{\infty} f_Y(x) \log_2\left(\frac{f_Y(x)}{f_X(x)}\right) dx \tag{5.70}$$

where $f_X(x)$ is viewed as the "reference" distribution. In a corresponding way to the fundamental property of (5.13), we have

$$D(f_Y \| f_X) \geq 0 \tag{5.71}$$

Combining (5.70) and (5.71) into a single inequality, we may thus write

$$\int_{-\infty}^{\infty} f_Y(x) \log_2\left(\frac{1}{f_Y(x)}\right) dx \leq \int_{-\infty}^{\infty} f_Y(x) \log_2\left(\frac{1}{f_X(x)}\right) dx$$

The expression on the left-hand side of this inequality is recognized as the differential entropy of the random variable Y, namely $h(Y)$. Accordingly,

$$h(Y) \leq \int_{-\infty}^{\infty} f_Y(x) \log_2\left(\frac{1}{f_Y(x)}\right) \mathrm{d}x \tag{5.72}$$

The next example illustrates an insightful application of (5.72).

EXAMPLE 9 **Gaussian Distribution**

Suppose two random variables, X and Y, are described as follows:

- the random variables X and Y have the common mean μ and variance σ^2;
- the random variable X is *Gaussian distributed* (see Section 3.9) as shown by

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] \tag{5.73}$$

Hence, substituting (5.73) into (5.72) and changing the base of the logarithm from 2 to e = 2.7183, we get

$$h(Y) \leq -\log_2 \mathrm{e} \int_{-\infty}^{\infty} f_Y(x) \left[-\frac{(x-\mu)^2}{2\sigma^2} - \log(\sqrt{2\pi}\sigma)\right] \mathrm{d}x \tag{5.74}$$

where e is the base of the natural algorithm. We now recognize the following characterizations of the random variable Y (given that its mean is μ and its variance is σ^2):

$$\int_{-\infty}^{\infty} f_Y(x)\, \mathrm{d}x = 1$$

$$\int_{-\infty}^{\infty} (x-\mu)^2 f_Y(x)\, \mathrm{d}x = \sigma^2$$

We may, therefore, simplify (5.74) as

$$h(Y) \leq \frac{1}{2}\log_2(2\pi \mathrm{e}\sigma^2) \tag{5.75}$$

The quantity on the right-hand side of (5.75) is, in fact, the differential entropy of the Gaussian random variable X:

$$h(X) = \frac{1}{2}\log_2(2\pi \mathrm{e}\sigma^2) \tag{5.76}$$

Finally, combining (5.75) and (5.76), we may write

$$h(Y) \leq h(X), \qquad \begin{cases} X\text{: Gaussian random variable} \\ Y\text{: nonGaussian random variable} \end{cases} \tag{5.77}$$

where equality holds if, and only if, $Y = X$.

We may now summarize the results of this important example by describing two entropic properties of a random variable:

PROPERTY 1 *For any finite variance, a Gaussian random variable has the largest differential entropy attainable by any other random variable.*

PROPERTY 2 *The entropy of a Gaussian random variable is uniquely determined by its variance (i.e., the entropy is independent of the mean).*

Indeed, it is because of Property 1 that the Gaussian channel model is so widely used as a conservative model in the study of digital communication systems.

Mutual Information

Continuing with the information-theoretic characterization of continuous random variables, we may use analogy with (5.47) to define the *mutual information* between the pair of continuous random variables X and Y as follows:

$$I(X;Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{X,Y}(x,y)\log_2\left[\frac{f_X(x|y)}{f_X(x)}\right]\mathrm{d}x\,\mathrm{d}y \tag{5.78}$$

where $f_{X,Y}(x,y)$ is the joint probability density function of X and Y and $f_X(x|y)$ is the conditional probability density function of X given $Y = y$. Also, by analogy with (5.45), (5.50), (5.43), and (5.44), we find that the mutual information between the pair of Gausian random variables has the following properties:

$$I(X;Y) = I(Y;X) \tag{5.79}$$

$$I(X;Y) \geq 0 \tag{5.80}$$

$$\begin{aligned} I(X;Y) &= h(X) - h(X|Y) \\ &= h(Y) - h(Y|X) \end{aligned} \tag{5.81}$$

The parameter $h(X)$ is the differential entropy of X; likewise for $h(Y)$. The parameter $h(X|Y)$ is the *conditional differential entropy* of X given Y; it is defined by the double integral (see (5.41))

$$h(X|Y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_{X,Y}(x,y)\log_2\left[\frac{1}{f_X(x|y)}\right]\mathrm{d}x\,\mathrm{d}y \tag{5.82}$$

The parameter $h(Y|X)$ is the conditional differential entropy of Y given X; it is defined in a manner similar to $h(X|Y)$.

5.10 Information Capacity Law

In this section we use our knowledge of probability theory to expand Shannon's channel-coding theorem, so as to formulate the information capacity for a *band-limited, power-limited Gaussian channel*, depicted in Figure 5.13. To be specific, consider a zero-mean stationary process $X(t)$ that is band-limited to B hertz. Let X_k, $k = 1, 2, \ldots, K$, denote the continuous random variables obtained by uniform sampling of the process $X(t)$ at a rate of $2B$ samples per second. The rate $2B$ samples per second is the smallest permissible rate for a bandwidth B that would not result in a loss of information in accordance with the sampling theorem; this is discussed in Chapter 6. Suppose that these samples are

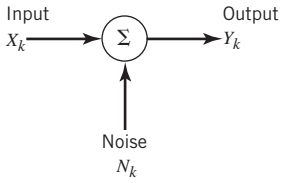


Figure 5.13 Model of discrete-time, memoryless Gaussian channel.

transmitted in T seconds over a noisy channel, also band-limited to B hertz. Hence, the total number of samples K is given by

$$K = 2BT \tag{5.83}$$

We refer to X_k as a sample of the *transmitted signal*. The channel output is perturbed by *additive white Gaussian noise* (AWGN) of zero mean and power spectral density $N_0/2$. The noise is band-limited to B hertz. Let the continuous random variables Y_k, $k = 1, 2, \ldots, K$, denote the corresponding samples of the channel output, as shown by

$$Y_k = X_k + N_k, \quad k = 1, 2, \ldots, K \tag{5.84}$$

The noise sample N_k in (5.84) is Gaussian with zero mean and variance

$$\sigma^2 = N_0 B \tag{5.85}$$

We assume that the samples Y_k, $k = 1, 2, \ldots, K$, are statistically independent.

A channel for which the noise and the received signal are as described in (5.84) and (5.85) is called a *discrete-time*, *memoryless Gaussian channel*, modeled as shown in Figure 5.13. To make meaningful statements about the channel, however, we have to assign a *cost* to each channel input. Typically, the transmitter is *power limited*; therefore, it is reasonable to define the cost as

$$\mathbb{E}[X_k^2] \le P, \quad k = 1, 2, \ldots, K \tag{5.86}$$

where P is the *average transmitted power*. The *power-limited Gaussian channel* described herein is not only of theoretical importance but also of practical importance, in that it models many communication channels, including line-of-sight radio and satellite links.

The *information capacity* of the channel is defined as the maximum of the mutual information between the channel input X_k and the channel output Y_k over all distributions of the input X_k that satisfy the power constraint of (5.86). Let $I(X_k;Y_k)$ denote the mutual information between X_k and Y_k. We may then define the *information capacity* of the channel as

$$C = \max_{f_{X_k}(x)} I(X_k;Y_k), \quad \text{subject to the constraint } \mathbb{E}[X_k^2] = P \quad \text{for all } k \tag{5.87}$$

In words, maximization of the mutual information $I(X_k;Y_k)$ is done with respect to all probability distributions of the channel input X_k, satisfying the power constraint $\mathbb{E}[X_k^2] = P$.

The mutual information $I(X_k;Y_k)$ can be expressed in one of the two equivalent forms shown in (5.81). For the purpose at hand, we use the second line of this equation to write

$$I(X_k;Y_k) = h(Y_k) - h(Y_k|X_k) \tag{5.88}$$

Since X_k and N_k are independent random variables and their sum equals Y_k in accordance with (5.84), we find that the conditional differential entropy of Y_k given X_k is equal to the differential entropy of N_k, as shown by

$$h(Y_k|X_k) = h(N_k) \tag{5.89}$$

Hence, we may rewrite (5.88) as

$$I(X_k;Y_k) = h(Y_k) - h(N_k) \tag{5.90}$$

With $h(N_k)$ being independent of the distribution of X_k, it follows that maximizing $I(X_k;Y_k)$ in accordance with (5.87) requires maximizing the differential entropy $h(Y_k)$. For $h(Y_k)$ to be maximum, Y_k has to be a Gaussian random variable. That is to say, samples of the channel output represent a noiselike process. Next, we observe that since N_k is Gaussian by assumption, the sample X_k of the channel input must be Gaussian too. We may therefore state that the maximization specified in (5.87) is attained by choosing samples of the channel input from a noiselike Gaussian-distributed process of average power P. Correspondingly, we may reformulate (5.87) as

$$C = I(X_k;Y_k)\text{: for Gaussian } X_k \text{ and } \mathbb{E}[X_k^2] = P \qquad \text{for all } k \tag{5.91}$$

where the mutual information $I(X_k;Y_k)$ is defined in accordance with (5.90).

For evaluation of the information capacity C, we now proceed in three stages:

1. The variance of sample Y_k of the channel output equals $P + \sigma^2$, which is a consequence of the fact that the random variables X and N are statistically independent; hence, the use of (5.76) yields the differential entropy

$$h(Y_k) = \frac{1}{2}\log_2[2\pi e(P + \sigma^2)] \tag{5.92}$$

2. The variance of the noisy sample N_k equals σ^2; hence, the use of (5.76) yields the differential entropy

$$h(N_k) = \frac{1}{2}\log_2[2\pi e\sigma^2] \tag{5.93}$$

3. Substituting (5.92) and (5.93) into (5.90), and recognizing the definition of information capacity given in (5.91), we get the formula:

$$C = \frac{1}{2}\log_2\left(1 + \frac{P}{\sigma^2}\right) \text{ bits per channel use} \tag{5.94}$$

With the channel used K times for the transmission of K samples of the process $X(t)$ in T seconds, we find that the information capacity per unit time is (K/T) times the result given in (5.94). The number K equals $2BT$, as in (5.83). Accordingly, we may express the information capacity of the channel in the following equivalent form:

$$C = B\log_2\left(1 + \frac{P}{N_0B}\right) \text{ bits per second} \tag{5.95}$$

where N_0B is the total noise power at the channel output, defined in accordance with (5.85).

Based on the formula of (5.95), we may now make the following statement

> The information capacity of a continuous channel of bandwidth B hertz, perturbed by AWGN of power spectral density $N_0/2$ and limited in bandwidth

to B, is given by the formula

$$C = B \log_2\left(1 + \frac{P}{N_0 B}\right) \text{ bits per second}$$

where P is the average transmitted power.

The *information capacity law*[12] of (5.95) is one of the most remarkable results of Shannon's information theory. In a single formula, it highlights most vividly the interplay among three key system parameters: channel bandwidth, average transmitted power, and power spectral density of channel noise. Note, however, that the dependence of information capacity C on channel bandwidth B is *linear*, whereas its dependence on signal-to-noise ratio $P/(N_0B)$ is *logarithmic*. Accordingly, we may make another insightful statement:

> It is easier to increase the information capacity of a continuous communication channel by expanding its bandwidth than by increasing the transmitted power for a prescribed noise variance.

The information capacity formula implies that, for given average transmitted power P and channel bandwidth B, we can transmit information at the rate of C bits per second, as defined in (5.95), with arbitrarily small probability of error by employing a sufficiently complex encoding system. It is not possible to transmit at a rate higher than C bits per second by any encoding system without a definite probability of error. Hence, the channel capacity law defines the *fundamental limit* on the permissible rate of error-free transmission for a power-limited, band-limited Gaussian channel. To approach this limit, however, the transmitted signal must have statistical properties approximating those of white Gaussian noise.

Sphere Packing

To provide a plausible argument supporting the information capacity law, suppose that we use an encoding scheme that yields K codewords, one for each sample of the transmitted signal. Let n denote the length (i.e., the number of bits) of each codeword. It is presumed that the coding scheme is designed to produce an acceptably low probability of symbol error. Furthermore, the codewords satisfy the power constraint; that is, the average power contained in the transmission of each codeword with n bits is nP, where P is the average power per bit.

Suppose that any codeword in the code is transmitted. The received vector of n bits is Gaussian distributed with a mean equal to the transmitted codeword and a variance equal to $n\sigma^2$, where σ^2 is the noise variance. With a high probability, we may say that the received signal vector at the channel output lies inside a sphere of radius $\sqrt{n\sigma^2}$; that is, centered on the transmitted codeword. This sphere is itself contained in a larger sphere of radius $\sqrt{n(P + \sigma^2)}$, where $n(P + \sigma^2)$ is the average power of the received signal vector.

We may thus visualize the sphere packing[13] as portrayed in Figure 5.14. With everything inside a small sphere of radius $\sqrt{n\sigma^2}$ assigned to the codeword on which it is

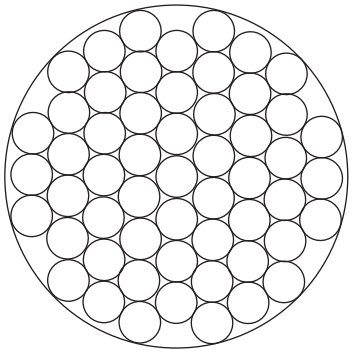


Figure 5.14 The sphere-packing problem.

centered. It is therefore reasonable to say that, when this particular codeword is transmitted, the probability that the received signal vector will lie inside the correct "decoding" sphere is high. The key question is:

> How many decoding spheres can be packed inside the larger sphere of received signal vectors? In other words, how many codewords can we in fact choose?

To answer this question, we want to eliminate the overlap between the decoding spheres as depicted in Figure 5.14. Moreover, expressing the volume of an n-dimensional sphere of radius r as $A_n r^n$, where A_n is a scaling factor, we may go on to make two statements:

1. The volume of the sphere of received signal vectors is $A_n[n(P + \sigma^2)]^{n/2}$.
2. The volume of the decoding sphere is $A_n(n\sigma^2)^{n/2}$.

Accordingly, it follows that the maximum number of *nonintersecting* decoding spheres that can be packed inside the sphere of possible received signal vectors is given by

$$\frac{A_n[n(P+\sigma^2)]^{n/2}}{A_n(n\sigma^2)^{n/2}} = \left(1 + \frac{P}{\sigma^2}\right)^{n/2} = 2^{(n/2)\log_2(1+P/\sigma^2)} \tag{5.96}$$

Taking the logarithm of this result to base 2, we readily see that the maximum number of bits per transmission for a low probability of error is indeed as defined previously in (5.94).

A final comment is in order: (5.94) is an idealized manifestation of Shannon's channel-coding theorem, in that it provides an upper bound on the physically realizable information capacity of a communication channel.

5.11 Implications of the Information Capacity Law

Now that we have a good understanding of the information capacity law, we may go on to discuss its implications in the context of a Gaussian channel that is limited in both power

and bandwidth. For the discussion to be useful, however, we need an ideal framework against which the performance of a practical communication system can be assessed. To this end, we introduce the notion of an *ideal system*, defined as a system that transmits data at a bit rate R_b equal to the information capacity C. We may then express the average transmitted power as

$$P = E_b C \tag{5.97}$$

where E_b is the *transmitted energy per bit*. Accordingly, the ideal system is defined by the equation

$$\frac{C}{B} = \log_2\left(1 + \frac{E_b}{N_0}\frac{C}{B}\right) \tag{5.98}$$

Rearranging this formula, we may define the *signal energy-per-bit to noise power spectral density ratio*, E_b/N_0, in terms of the ratio C/B for the ideal system as follows:

$$\frac{E_b}{N_0} = \frac{2^{C/B} - 1}{C/B} \tag{5.99}$$

A plot of the bandwidth efficiency R_b/B versus E_b/N_0 is called the *bandwidth-efficiency diagram*. A generic form of this diagram is displayed in Figure 5.15, where the curve labeled "capacity boundary" corresponds to the ideal system for which $R_b = C$.

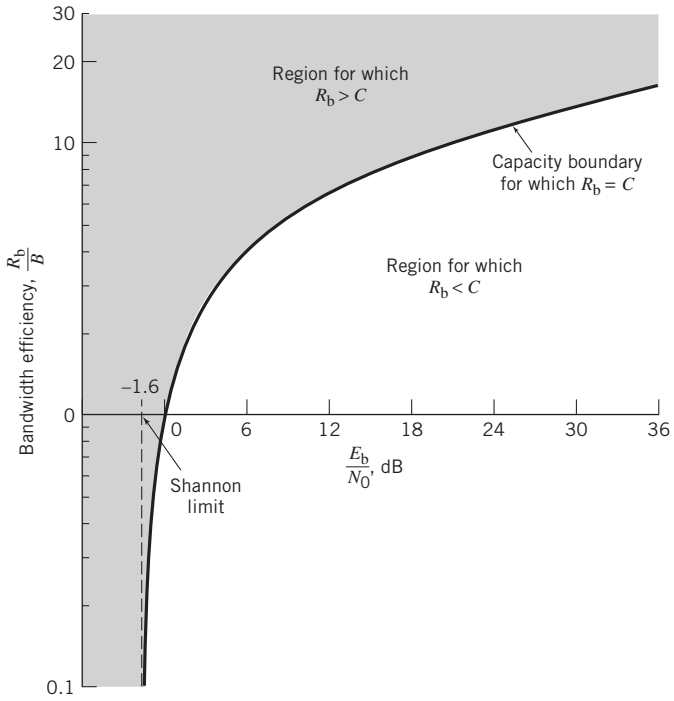


Figure 5.15 Bandwidth-efficiency diagram.

Based on Figure 5.15, we can make three observations:

1. For *infinite channel bandwidth*, the ratio E_b/N_0 approaches the limiting value

$$\left(\frac{E_b}{N_0}\right)_\infty = \lim_{B \to \infty}\left(\frac{E_b}{N_0}\right) = \log_e 2 = 0.693 \tag{5.100}$$

   where $\log_e$ stands for the natural logarithm ln. The value defined in (5.100) is called the *Shannon limit* for an AWGN channel, assuming a code rate of zero. Expressed in decibels, the Shannon limit equals –1.6 dB. The corresponding limiting value of the channel capacity is obtained by letting the channel bandwidth B in (5.95) approach infinity, in which case we obtain

$$C_\infty = \lim_{B \to \infty} C = \left(\frac{P}{N_0}\right)\log_2 e \tag{5.101}$$

2. The *capacity boundary* is defined by the curve for the critical bit rate $R_b = C$. For any point on this boundary, we may flip a fair coin (with probability of 1/2) whether we have error-free transmission or not. As such, the boundary separates combinations of system parameters that have the potential for supporting error-free transmission ($R_b < C$) from those for which error-free transmission is not possible ($R_b > C$). The latter region is shown shaded in Figure 5.15.
3. The diagram highlights potential *trade-offs* among three quantities: the E_b/N_0, the ratio R_b/B, and the probability of symbol error P_e. In particular, we may view movement of the operating point along a horizontal line as trading P_e versus E_b/N_0 for a fixed R_b/B. On the other hand, we may view movement of the operating point along a vertical line as trading P_e versus R_b/B for a fixed E_b/N_0.

EXAMPLE 10 Capacity of Binary-Input AWGN Channel

In this example, we investigate the capacity of an AWGN channel using *encoded* binary antipodal signaling (i.e., levels –1 and +1 for binary symbols 0 and 1, respectively). In particular, we address the issue of determining the minimum achievable bit error rate as a function of E_b/N_0 for varying code rate r. It is assumed that the binary symbols 0 and 1 are equiprobable.

Let the random variables X and Y denote the channel input and channel output respectively; X is a discrete variable, whereas Y is a continuous variable. In light of the second line of (5.81), we may express the mutual information between the channel input and channel output as

$$I(X;Y) = h(Y) - h(Y|X)$$

The second term, $h(Y|X)$, is the conditional differential entropy of the channel output Y, given the channel input X. By virtue of (5.89) and (5.93), this term is just the entropy of a Gaussian distribution. Hence, using σ^2 to denote the variance of the channel noise, we write

$$h(Y|X) = \frac{1}{2}\log_2(2\pi e\sigma^2)$$

Next, the first term, $h(Y)$, is the differential entropy of the channel output Y. With the use of binary antipodal signaling, the probability density function of Y, given $X = x$, is a mixture of two Gaussian distributions with common variance σ^2 and mean values –1 and +1, as shown by

$$f_Y(y_i|x) = \frac{1}{2}\left\{\frac{\exp[-(y_i+1)^2/2\sigma^2]}{\sqrt{2\pi}\sigma} + \frac{\exp[-(y_i-1)^2/2\sigma^2]}{\sqrt{2\pi}\sigma}\right\} \tag{5.102}$$

Hence, we may determine the differential entropy of Y using the formula

$$h(Y) = -\int_{-\infty}^{\infty} f_Y(y_i|x)\log_2[f_Y(y_i|x)]\,dy_i$$

where $f_Y(y_i \mid x)$ is defined by (5.102). From the formulas of $h(Y|X)$ and $h(Y)$, it is clear that the mutual information is solely a function of the noise variance σ^2. Using $M(\sigma^2)$ to denote this functional dependence, we may thus write

$$I(X;Y) = M(\sigma^2)$$

Unfortunately, there is no closed formula that we can derive for $M(\sigma^2)$ because of the difficulty of determining $h(Y)$. Nevertheless, the differential entropy $h(Y)$ can be well approximated using *Monte Carlo integration*; see Appendix E for details.

Because symbols 0 and 1 are equiprobable, it follows that the channel capacity C is equal to the mutual information between X and Y. Hence, for error-free data transmission over the AWGN channel, the code rate r must satisfy the condition

$$r < M(\sigma^2) \tag{5.103}$$

A robust measure of the ratio E_b/N_0, is

$$\frac{E_b}{N_0} = \frac{P}{N_0 r} = \frac{P}{2\sigma^2 r}$$

where P is the average transmitted power and $N_0/2$ is the two-sided power spectral density of the channel noise. Without loss of generality, we may set $P = 1$. We may then express the noise variance as

$$\sigma^2 = \frac{N_0}{2E_b r} \tag{5.104}$$

Substituting Equation (5.104) into (5.103) and rearranging terms, we get the desired relation:

$$\frac{E_b}{N_0} = \frac{1}{2rM^{-1}(r)} \tag{5.105}$$

where $M^{-1}(r)$ is the *inverse* of the mutual information between the channel input and putput, expressed as a function of the code rate r.

Using the Monte Carlo method to estimate the differential entropy $h(Y)$ and therefore $M^{-1}(r)$, the plots of Figure 5.16 are computed.[14] Figure 5.16a plots the minimum E_b/N_0 versus the code rate r for error-free transmission. Figure 5.16b plots the minimum

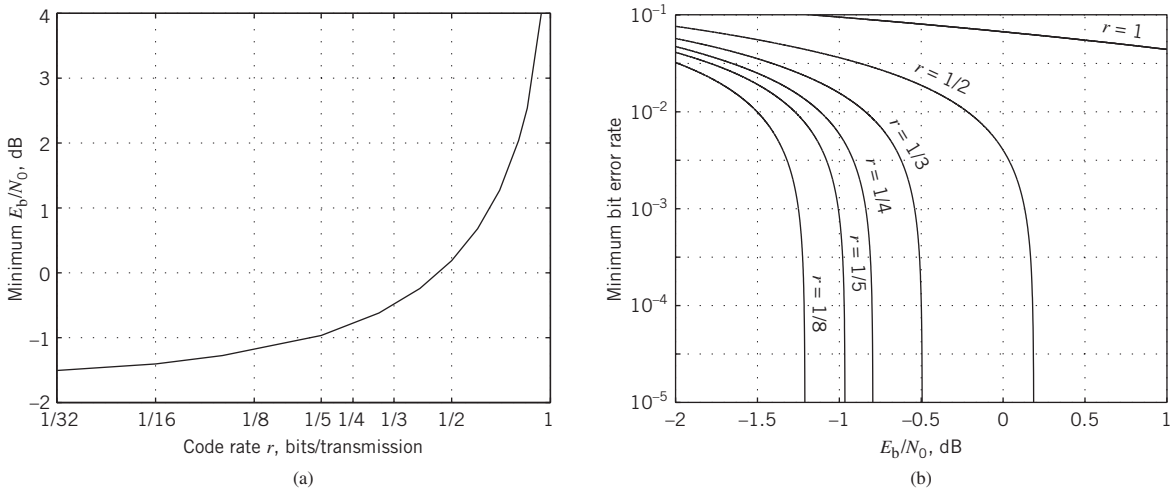


Figure 5.16 Binary antipodal signaling over an AWGN channel. (a) Minimum E_b/N_0 versus the code rate r. (b) Minimum bit error rate versus E_b/N_0 for varying code rate r.

achievable bit error rate versus E_b/N_0 with the code rate r as a running parameter. From Figure 5.16 we may draw the following conclusions:

- For uncoded binary signaling (i.e., $r = 1$), an infinite E_b/N_0 is required for error-free communication, which agrees with what we know about uncoded data transmission over an AWGN channel.
- The minimum E_b/N_0, decreases with decreasing code rate r, which is intuitively satisfying. For example, for $r = 1/2$, the minimum value of E_b/N_0 is slightly less than 0.2 dB.
- As r approaches zero, the minimum E_b/N_0 approaches the limiting value of –1.6 dB, which agrees with the Shannon limit derived earlier; see (5.100).

5.12 Information Capacity of Colored Noisy Channel

The information capacity theorem as formulated in (5.95) applies to a band-limited white noise channel. In this section we extend Shannon's information capacity law to the more general case of a *nonwhite*, or *colored*, *noisy channel*.[15] To be specific, consider the channel model shown in Figure 5.17a where the transfer function of the channel is denoted by $H(f)$. The channel noise $n(t)$, which appears additively at the channel output, is modeled as the sample function of a stationary Gaussian process of zero mean and power spectral density $S_N(f)$. The requirement is twofold:

1. Find the input ensemble, described by the power spectral density $S_{xx}(f)$, that maximizes the mutual information between the channel output $y(t)$ and the channel input $x(t)$, subject to the constraint that the average power of $x(t)$ is fixed at a constant value P.
2. Hence, determine the optimum information capacity of the channel.

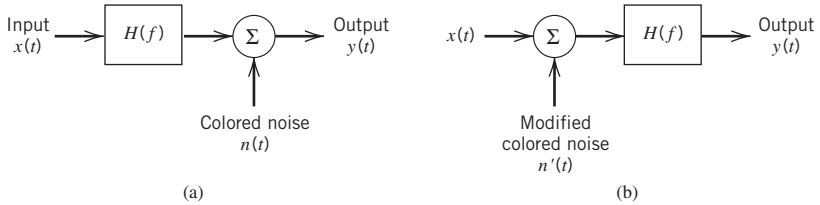


Figure 5.17 (a) Model of band-limited, power-limited noisy channel. (b) Equivalent model of the channel.

This problem is a constrained optimization problem. To solve it, we proceed as follows:

- Because the channel is linear, we may replace the model of Figure 5.17a with the equivalent model shown in Figure 5.17b. From the viewpoint of the spectral characteristics of the signal plus noise measured at the channel output, the two models of Figure 5.17 are equivalent, provided that the power spectral density of the noise $n'(t)$ in Figure 5.17b is defined in terms of the power spectral density of the noise $n(t)$ in Figure 5.17a as

$$S_{N'N'}(f) = \frac{S_{NN}(f)}{|H(f)|^2} \tag{5.106}$$

  where $|H(f)|$ is the magnitude response of the channel.
- To simplify the analysis, we use the "principle of divide and conquer" to approximate the continuous $|H(f)|$ described as a function of frequency f in the form of a staircase, as illustrated in Figure 5.18. Specifically, the channel is divided into a large number of adjoining frequency slots. The smaller we make the incremental frequency interval Δf of each subchannel, the better this approximation is.

The net result of these two points is that the original model of Figure 5.17a is replaced by the parallel combination of a finite number of subchannels, N, each of which is corrupted essentially by "band-limited white Gaussian noise."

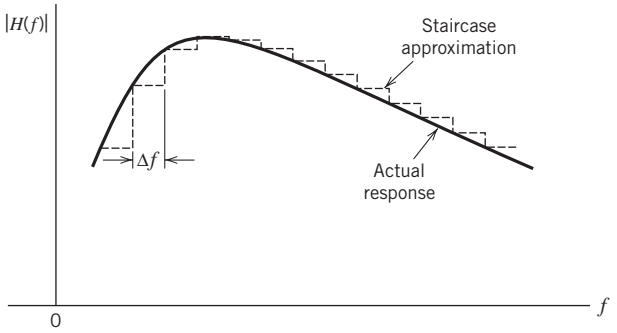


Figure 5.18 Staricase approximation of an arbitrary magnitude response $|H(f)|$; only the positive frequency portion of the response is shown.

The kth subchannel in the approximation to the model of Figure 5.17b is described by

$$y_k(t) = x_k(t) + n_k(t), \qquad k = 1, 2, \ldots, N \tag{5.107}$$

The average power of the signal component $x_k(t)$ is

$$P_k = S_{XX}(f_k)\Delta f, \qquad k = 1, 2, \ldots, N \tag{5.108}$$

where $S_X(f_k)$ is the power spectral density of the input signal evaluated at the frequency $f = f_k$. The variance of the noise component $n_k(t)$ is

$$\sigma_k^2 = \frac{S_{NN}(f_k)}{\left|H(f_k)\right|^2}\Delta f, \qquad k = 1, 2, \ldots, N \tag{5.109}$$

where $S_N(f_k)$ and $|H(f_k)|$ are the noise spectral density and the channel's magnitude response evaluated at the frequency f_k, respectively. The information capacity of the kth subchannel is

$$C_k = \frac{1}{2}\Delta f \log_2\left(1 + \frac{P_k}{\sigma_k^2}\right), \qquad k = 1, 2, \ldots, N \tag{5.110}$$

where the factor 1/2 accounts for the fact that Δf applies to both positive and negative frequencies. All the N subchannels are independent of one another. Hence, the total capacity of the overall channel is approximately given by the summation

$$\begin{aligned} C &\approx \sum_{k=1}^{N} C_k \\ &= \frac{1}{2}\sum_{k=1}^{N} \Delta f \log_2\left(1 + \frac{P_k}{\sigma_k^2}\right) \end{aligned} \tag{5.111}$$

The problem we have to address is to maximize the overall information capacity C subject to the constraint

$$\sum_{k=1}^{N} P_k = P = \text{constant} \tag{5.112}$$

The usual procedure to solve a constrained optimization problem is to use the *method of Lagrange multipliers* (see Appendix D for a discussion of this method). To proceed with this optimization, we first define an objective function that incorporates both the information capacity C and the constraint (i.e., (5.111) and (5.112)), as shown by

$$J(P_k) = \frac{1}{2}\sum_{k=1}^{N} \Delta f \log_2\left(1 + \frac{P_k}{\sigma_k^2}\right) + \lambda\left(P - \sum_{k=1}^{N} P_k\right) \tag{5.113}$$

where λ is the Lagrange multiplier. Next, differentiating the objective function $J(P_k)$ with respect to P_k and setting the result equal to zero, we obtain

$$\frac{\Delta f \log_2 e}{P_k + \sigma_k^2} - \lambda = 0$$

To satisfy this optimizing solution, we impose the following requirement:

$$P_k + \sigma_k^2 = K\Delta f \qquad \text{for } k = 1, 2, \ldots, N \tag{5.114}$$

where K is a constant that is the same for all k. The constant K is chosen to satisfy the average power constraint.

Inserting the defining values of (5.108) and (5.109) in the optimizing condition of (5.114), simplifying, and rearranging terms we get

$$S_{XX}(f_k) = K - \frac{S_{NN}(f_k)}{|H(f_k)|^2}, \qquad k = 1, 2, \ldots, N \tag{5.115}$$

Let $\mathcal{F}_A$ denote the frequency range for which the constant K satisfies the condition

$$K \geq \frac{S_{NN}(f_k)}{|H(f_k)|^2}$$

Then, as the incremental frequency interval Δf is allowed to approach zero and the number of subchannels N goes to infinity, we may use (5.115) to formally state that the power spectral density of the input ensemble that achieves the optimum information capacity is a nonnegative quantity defined by

$$S_{XX}(f) = \begin{cases} K - \dfrac{S_{NN}(f)}{|H(f)|^2}, & f \in \mathcal{F}_A \\ 0, & \text{otherwise} \end{cases} \tag{5.116}$$

Because the average power of a random process is the total area under the curve of the power spectral density of the process, we may express the average power of the channel input $x(t)$ as

$$P = \int_{f \in \mathcal{F}_A} \left(K - \frac{S_{NN}(f)}{|H(f)|^2} \right) \mathrm{d}f \tag{5.117}$$

For a prescribed P and specified $S_N(f)$ and $H(f)$, the constant K is the solution to (5.117).

The only thing that remains for us to do is to find the optimum information capacity. Substituting the optimizing solution of (5.114) into (5.111) and then using the defining values of (5.108) and (5.109), we obtain

$$C \approx \frac{1}{2} \sum_{k=1}^{N} \Delta f \log_2 \left(K \frac{|H(f_k)|^2}{S_{NN}(f_k)} \right)$$

When the incremental frequency interval Δf is allowed to approach zero, this equation takes the limiting form

$$C = \frac{1}{2} \int_{-\infty}^{\infty} \log_2 \left(K \frac{|H(f)|^2}{S_{NN}(f)} \right) \mathrm{d}f \tag{5.118}$$

where the constant K is chosen as the solution to (5.117) for a prescribed input signal power P.

Water-filling Interpretation of the Information Capacity Law

Equations (5.116) and (5.117) suggest the picture portrayed in Figure 5.19. Specifically, we make the following observations:

- The appropriate input power spectral density $S_X(f)$ is described as the bottom regions of the function $S_N(f)/|H(f)|^2$ that lie below the constant level K, which are shown shaded.
- The input power P is defined by the total area of these shaded regions.

The spectral-domain picture portrayed here is called the *water-filling (pouring) interpretation*, in the sense that the process by which the input power is distributed across the function $S_N(f)/|H(f)|^2$ is identical to the way in which water distributes itself in a vessel.

Consider now the idealized case of a band-limited signal in AWGN channel of power spectral density $N(f) = N_0/2$. The transfer function $H(f)$ is that of an ideal band-pass filter defined by

$$H(f) = \begin{cases} 1, & 0 \le f_c - \frac{B}{2} \le |f| \le f_c + \frac{B}{2} \\ 0, & \text{otherwise} \end{cases}$$

where f_c is the midband frequency and B is the channel bandwidth. For this special case, (5.117) and (5.118) reduce respectively to

$$P = 2B\left(K - \frac{N_0}{2}\right)$$

and

$$C = B \log_2\left(\frac{2K}{N_0}\right)$$

Hence, eliminating K between these two equations, we get the standard form of Shannon's capacity theorem, defined by (5.95).

EXAMPLE 11 **Capacity of NEXT-Dominated Channel**

Digital subscriber lines (DSLs) refer to a family of different technologies that operate over a closed transmission loop; they will be discussed in Chapter 8, Section 8.11. For the

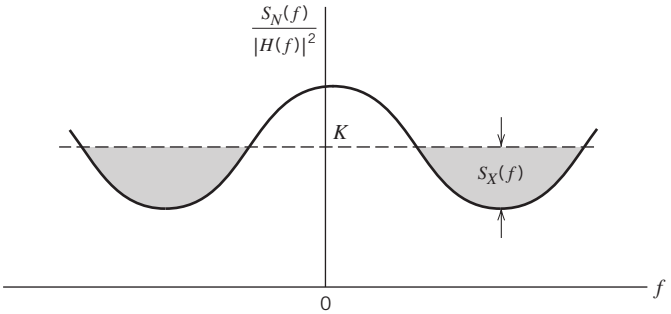


Figure 5.19 Water-filling interpretation of information-capacity theorem for a colored noisy channel.

present, it suffices to say that a DSL is designed to provide for data transmission between a user terminal (e.g., computer) and the central office of a telephone company. A major channel impairment that arises in the deployment of a DSL is the near-end cross-talk (NEXT). The power spectral density of this crosstalk may be taken as

$$S_N(f) = |H_{\text{NEXT}}(f)|^2 S_X(f) \tag{5.119}$$

where $S_X(f)$ is the power spectral density of the transmitted signal and $H_{\text{NEXT}}(f)$ is the transfer function that couples adjacent twisted pairs. The only constraint we have to satisfy in this example is that the power spectral density function $S_X(f)$ be *nonnegative for all f*. Substituting (5.119) into (5.116), we readily find that this condition is satisfied by solving for K as

$$K = \left(1 + \frac{|H_{\text{NEXT}}(f)|^2}{|H(f)|^2}\right) S_X(f)$$

Finally, using this result in (5.118), we find that the capacity of the NEXT-dominated digital subscriber channel is given by

$$C = \frac{1}{2}\int_{\mathcal{F}_A} \log_2\left(1 + \frac{|H(f)|^2}{|H_{\text{NEXT}}(f)|^2}\right) \mathrm{d}f$$

where $\mathcal{F}_A$ is the set of positive and negative frequencies for which $S_X(f) > 0$.

5.13 Rate Distortion Theory

In Section 5.3 we introduced the source-coding theorem for a discrete memoryless source, according to which the average codeword length must be at least as large as the source entropy for perfect coding (i.e., perfect representation of the source). However, in many practical situations there are constraints that force the coding to be imperfect, thereby resulting in unavoidable *distortion*. For example, constraints imposed by a communication channel may place an upper limit on the permissible code rate and, therefore, on average codeword length assigned to the information source. As another example, the information source may have a continuous amplitude as in the case of speech, and the requirement is to quantize the amplitude of each sample generated by the source to permit its representation by a codeword of finite length as in pulse-code modulation to be discussed in Chapter 6. In such cases, the problem is referred to as *source coding with a fidelity criterion*, and the branch of information theory that deals with it is called *rate distortion theory*.[16] Rate distortion theory finds applications in two types of situations:

- Source coding where the permitted coding alphabet cannot exactly represent the information source, in which case we are forced to do lossy *data compression*.
- Information transmission at a rate greater than channel capacity.

Accordingly, rate distortion theory may be viewed as a natural extension of Shannon's coding theorem.

Rate Distortion Function

Consider a discrete memoryless source defined by an M-ary alphabet $\mathcal{X}$: $\{x_i | i = 1, 2, \ldots, M\}$, which consists of a set of statistically independent symbols together with the associated symbol probabilities $\{p_i | i = 1, 2, \ldots, M\}$. Let R be the average code rate in bits per codeword. The representation codewords are taken from another alphabet $\mathcal{Y}$:$\{y_j | j = 1, 2, \ldots, N\}$. The source-coding theorem states that this second alphabet provides a perfect representation of the source provided that $R > H$, where H is the source entropy. But if we are forced to have $R < H$, then there is unavoidable distortion and, therefore, loss of information.

Let $p(x_i, y_j)$ denote the joint probability of occurrence of source symbol x_i and representation symbol y_j. From probability theory, we have

$$p(x_i, y_j) = p(y_j | x_i)p(x_i) \tag{5.120}$$

where $p(y_j|x_i)$ is a transition probability. Let $d(x_i, y_j)$ denote a measure of the cost incurred in representing the source symbol x_i by the symbol y_j; the quantity $d(x_i, y_j)$ is referred to as a *single-letter distortion measure*. The statistical average of $d(x_i, y_j)$ over all possible source symbols and representation symbols is given by

$$\bar{d} = \sum_{i=1}^{M} \sum_{j=1}^{N} p(x_i)p(y_j|x_i)d(x_i|y_j) \tag{5.121}$$

Note that the average distortion $\bar{d}$ is a nonnegative continuous function of the transition probabilities $p(y_j|x_i)$ that are determined by the source encoder–decoder pair.

A conditional probability assignment $p(y_j|x_i)$ is said to be *D-admissible* if, and only if, the average distortion $\bar{d}$ is less than or equal to some acceptable value D. The set of all D-admissible conditional probability assignments is denoted by

$$\mathcal{P}_D = \{p(y_j|x_i): \bar{d} \leq D\} \tag{5.122}$$

For each set of transition probabilities, we have a mutual information

$$I(X;Y) = \sum_{i=1}^{M} \sum_{j=1}^{N} p(x_i)p(y_j|x_i) \log\left(\frac{p(y_j|x_i)}{p(y_j)}\right) \tag{5.123}$$

A *rate distortion function* $R(D)$ is defined as *the smallest coding rate possible for which the average distortion is guaranteed not to exceed D*. Let $\mathcal{P}_D$ denote the set to which the conditional probability $p(y_j|x_i)$ belongs for a prescribed D. Then, for a fixed D we write[17]

$$R(D) = \min_{p(y_j|x_i) \in \mathcal{P}_D} I(X;Y) \tag{5.124}$$

subject to the constraint

$$\sum_{j=1}^{N} p(y_j|x_i) = 1 \qquad \text{for } i = 1, 2, \ldots, M \tag{5.125}$$

The rate distortion function $R(D)$ is measured in units of bits if the base-2 logarithm is used in (5.123). Intuitively, we expect the distortion D to decrease as the rate distortion function $R(D)$ is increased. We may say conversely that tolerating a large distortion D permits the use of a smaller rate for coding and/or transmission of information.

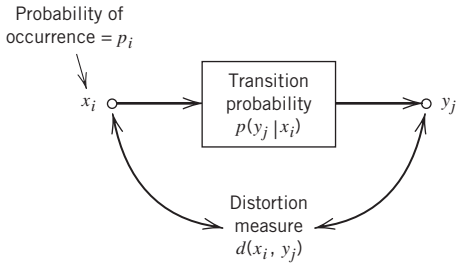


Figure 5.20 Summary of rate distortion theory.

Figure 5.20 summarizes the main parameters of rate distortion theory. In particular, given the source symbols $\{x_i\}$ and their probabilities $\{p_i\}$, and given a definition of the single-letter distortion measure $d(x_i, y_j)$, the calculation of the rate distortion function $R(D)$ involves finding the conditional probability assignment $p(y_j|x_i)$ subject to certain constraints imposed on $p(y_j|x_i)$. This is a variational problem, the solution of which is unfortunately not straightforward in general.

EXAMPLE 12 Gaussian Source

Consider a discrete-time, memoryless Gaussian source with zero mean and variance σ^2. Let x denote the value of a sample generated by such a source. Let y denote a quantized version of x that permits a finite representation of it. The *square-error distortion*

$$d(x, y) = (x - y)^2$$

provides a distortion measure that is widely used for continuous alphabets. The rate distortion function for the Gaussian source with square-error distortion, as described herein, is given by

$$R(D) = \begin{cases} \frac{1}{2}\log\left(\frac{\sigma^2}{D}\right), & 0 \le D \le \sigma^2 \\ 0, & D > \sigma^2 \end{cases} \tag{5.126}$$

In this case, we see that $R(D) \to \infty$ as $D \to 0$, and $R(D) = 0$ for $D = \sigma^2$.

EXAMPLE 13 Set of Parallel Gaussian Sources

Consider next a set of N independent Gaussian random variables $\{X_i\}_{i=1}^{N}$, where X_i has zero mean and variance σ_i^2.Using the distortion measure

$$d = \sum_{i=1}^{N} (x_i - \hat{x}_i)^2, \qquad \hat{x}_i = \text{estimate of } x_i$$

and building on the result of Example 12, we may express the rate distortion function for the set of parallel Gaussian sources described here as

$$R(D) = \sum_{i=1}^{N} \frac{1}{2}\log\left(\frac{\sigma_i^2}{D_i}\right) \tag{5.127}$$

where D_i is itself defined by

$$D_i = \begin{cases} \lambda, & \lambda < \sigma_i^2 \\ \sigma_i^2, & \lambda \geq \sigma_i^2 \end{cases} \tag{5.128}$$

and the constant λ is chosen so as to satisfy the condition

$$\sum_{i=1}^{N} D_i = D \tag{5.129}$$

Compared to Figure 5.19, (5.128) and (5.129) may be interpreted as a kind of "water-filling in reverse," as illustrated in Figure 5.21. First, we choose a constant λ and only the subset of random variables whose variances exceed the constant λ. No bits are used to describe the remaining subset of random variables whose variances are less than the constant λ.

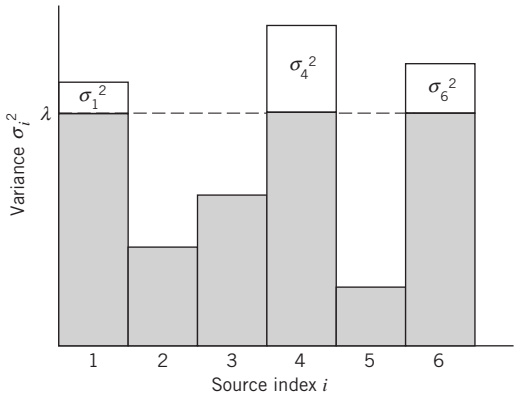


Figure 5.21 Reverse water-filling picture for a set of parallel Gaussian processes.

5.14 Summary and Discussion

In this chapter we established two fundamental limits on different aspects of a communication system, which are embodied in the source-coding theorem and the channel-coding theorem.

The *source-coding theorem*, Shannon's first theorem, provides the mathematical tool for assessing *data compaction;* that is, *lossless compression* of data generated by a discrete memoryless source. The theorem teaches us that we can make the average number of binary code elements (bits) per source symbol as small as, but no smaller than, the entropy of the source measured in bits. The *entropy* of a source is a function of the probabilities of the source symbols that constitute the alphabet of the source. Since

entropy is a measure of uncertainty, the entropy is maximum when the associated probability distribution generates maximum uncertainty.

The *channel-coding theorem*, Shannon's second theorem, is both the most surprising and the single most important result of information theory. For a *binary symmetric channel*, the channel-coding theorem teaches us that, for any *code rate r* less than or equal to the *channel capacity C*, codes do exist such that the average probability of error is as small as we want it. A binary symmetric channel is the simplest form of a discrete memoryless channel. It is symmetric, because the probability of receiving symbol 1 if symbol 0 is sent is the same as the probability of receiving symbol 0 if symbol 1 is sent. This probability, the probability that an error will occur, is termed a *transition probability.* The transition probability p is determined not only by the additive noise at the channel output, but also by the kind of receiver used. The value of p uniquely defines the channel capacity C.

The *information capacity law*, an application of the channel-coding theorem, teaches us that there is an upper limit to the rate at which any communication system can operate reliably (i.e., free of errors) when the system is constrained in power. This maximum rate, called the *information capacity*, is measured in bits per second. When the system operates at a rate greater than the information capacity, it is condemned to a high probability of error, regardless of the choice of signal set used for transmission or the receiver used for processing the channel output.

When the output of a source of information is compressed in a lossless manner, the resulting data stream usually contains redundant bits. These redundant bits can be removed by using a lossless algorithm such as Huffman coding or the Lempel–Ziv algorithm for data compaction. We may thus speak of data compression followed by data compaction as two constituents of the *dissection of source coding*, which is so called because it refers exclusively to the sources of information.

We conclude this chapter on Shannon's information theory by pointing out that, in many practical situations, there are constraints that force source coding to be imperfect, thereby resulting in unavoidable *distortion*. For example, constraints imposed by a communication channel may place an upper limit on the permissible code rate and, therefore, average codeword length assigned to the information source. As another example, the information source may have a continuous amplitude, as in the case of speech, and the requirement is to *quantize* the amplitude of each sample generated by the source to permit its representation by a codeword of finite length, as in pulse-code modulation discussed in Chapter 6. In such cases, the information-theoretic problem is referred to as *source coding with a fidelity criterion*, and the branch of information theory that deals with it is called *rate distortion theory,* which may be viewed as a natural extension of Shannon's coding theorem.

Problems

Entropy

5.1 Let p denote the probability of some event. Plot the amount of information gained by the occurrence of this event for $0 \leq p \leq 1$.

5.2 A source emits one of four possible symbols during each signaling interval. The symbols occur with the probabilities

$$p_0 = 0.4$$
$$p_1 = 0.3$$
$$p_2 = 0.2$$
$$p_3 = 0.1$$

which sum to unity as they should. Find the amount of information gained by observing the source emitting each of these symbols.

5.3 A source emits one of four symbols s_0, s_1, s_2, and s_3 with probabilities 1/3, 1/6, 1/4 and 1/4, respectively. The successive symbols emitted by the source are statistically independent. Calculate the entropy of the source.

5.4 Let X represent the outcome of a single roll of a fair die. What is the entropy of X?

5.5 The sample function of a Gaussian process of zero mean and unit variance is uniformly sampled and then applied to a uniform quantizer having the input–output amplitude characteristic shown in Figure P5.5. Calculate the entropy of the quantizer output.

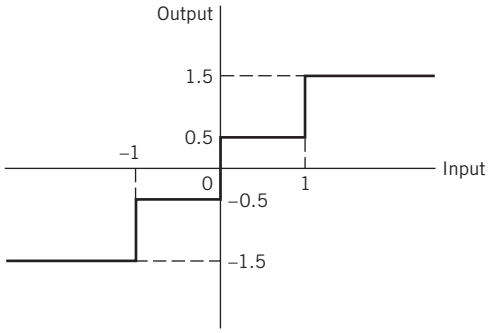


Figure P5.5

5.6 Consider a discrete memoryless source with source alphabet $S = \{s_0, s_1, \ldots, s_{K-1}\}$ and source statistics $\{p_0, p_1, \ldots, p_{K-1}\}$. The nth extension of this source is another discrete memoryless source with source alphabet $S^{(n)} = \{\sigma_0, \sigma_1, \ldots, \sigma_{M-1}\}$, where $M = K^n$. Let $P(\sigma_i)$ denote the probability of σ_i.

a. Show that, as expected,

$$\sum_{i=0}^{M-1} P(\sigma_i) = 1$$

b. Show that

$$\sum_{i=0}^{M-1} P(\sigma_i) \log_2\left(\frac{1}{p_{i_k}}\right) = H(S), \qquad k = 1, 2, \ldots, n$$

where p_{i_k} is the probability of symbol s_{i_k} and $H(S)$ is the entropy of the original source.

c. Hence, show that

$$H(S^{(n)}) = \sum_{i=0}^{M-1} P(\sigma_i) \log_2\left(\frac{1}{P(\sigma_i)}\right)$$
$$= nH(S)$$

5.7 Consider a discrete memoryless source with source alphabet $S = \{s_0, s_1, s_2\}$ and source statistics $\{0.7, 0.15, 0.15\}$.

a. Calculate the entropy of the source.

b. Calculate the entropy of the second-order extension of the source.

5.8 It may come as a surprise, but the number of bits needed to store text is much less than that required to store its spoken equivalent. Can you explain the reason for this statement?

5.9 Let a discrete random variable X assume values in the set $\{x_1, x_2, \ldots, x_n\}$. Show that the entropy of X satisfies the inequality

$$H(X) \leq \log n$$

and with equality if, and only if, the probability $p_i = 1/n$ for all i.

Lossless Data Compression

5.10 Consider a discrete memoryless source whose alphabet consists of K equiprobable symbols.

a. Explain why the use of a fixed-length code for the representation of such a source is about as efficient as any code can be.

b. What conditions have to be satisfied by K and the codeword length for the coding efficiency to be 100%?

5.11 Consider the four codes listed below:

Symbol	Code I	Code II	Code III	Code IV
s_0	0	0	0	00
s_1	10	01	01	01
s_2	110	001	011	10
s_3	1110	0010	110	110
s_4	1111	0011	111	111

a. Two of these four codes are prefix codes. Identify them and construct their individual decision trees.

b. Apply the Kraft inequality to codes I, II, III, and IV. Discuss your results in light of those obtained in part a.

5.12 Consider a sequence of letters of the English alphabet with their probabilities of occurrence

Letter	a	i	l	m	n	o	p	y
Probability	0.1	0.1	0.2	0.1	0.1	0.2	0.1	0.1

Compute two different Huffman codes for this alphabet. In one case, move a combined symbol in the coding procedure as high as possible; in the second case, move it as low as possible. Hence, for each of the two codes, find the average codeword length and the variance of the average codeword length over the ensemble of letters. Comment on your results.

5.13 A discrete memoryless source has an alphabet of seven symbols whose probabilities of occurrence are as described here:

Symbol	s_0	s_1	s_2	s_3	s_4	s_5	s_6
Probability	0.25	0.25	0.125	0.125	0.125	0.0625	0.0625

Compute the Huffman code for this source, moving a "combined" symbol as high as possible. Explain why the computed source code has an efficiency of 100%.

5.14 Consider a discrete memoryless source with alphabet $\{s_0, s_1, s_2\}$ and statistics $\{0.7, 0.15, 0.15\}$ for its output.

a. Apply the Huffman algorithm to this source. Hence, show that the average codeword length of the Huffman code equals 1.3 bits/symbol.

b. Let the source be extended to order two. Apply the Huffman algorithm to the resulting extended source and show that the average codeword length of the new code equals 1.1975 bits/symbol.

c. Extend the order of the extended source to three and reapply the Huffman algorithm; hence, calculate the average codeword length.

d. Compare the average codeword length calculated in parts b and c with the entropy of the original source.

5.15 Figure P5.15 shows a Huffman tree. What is the codeword for each of the symbols A, B, C, D, E, F, and G represented by this Huffman tree? What are their individual codeword lengths?

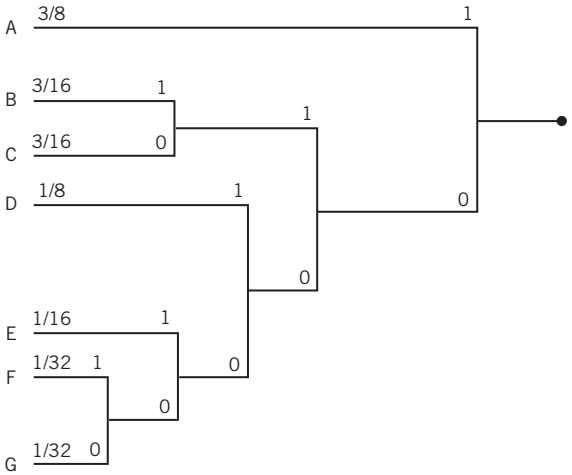


Figure P5.15

5.16 A computer executes four instructions that are designated by the codewords (00, 01, 10, 11). Assuming that the instructions are used independently with probabilities (1/2, 1/8, 1/8, 1/4), calculate the percentage by which the number of bits used for the instructions may be reduced by the use of an optimum source code. Construct a Huffman code to realize the reduction.

5.17 Consider the following binary sequence

$$11101001100010110100 \ldots$$

Use the Lempel–Ziv algorithm to encode this sequence, assuming that the binary symbols 0 and 1 are already in the cookbook.

Binary Symmetric Channel

5.18 Consider the transition probability diagram of a binary symmetric channel shown in Figure 5.8. The input binary symbols 0 and 1 occur with equal probability. Find the probabilities of the binary symbols 0 and 1 appearing at the channel output.

5.19 Repeat the calculation in Problem 5.18, assuming that the input binary symbols 0 and 1 occur with probabilities 1/4 and 3/4, respectively.

Mutual Information and Channel Capacity

5.20 Consider a binary symmetric channel characterized by the transition probability p. Plot the mutual information of the channel as a function of p_1, the a priori probability of symbol 1 at the channel input. Do your calculations for the transition probability $p = 0, 0.1, 0.2, 0.3, 0.5$.

5.21 Revisiting (5.12), express the mutual information $I(X;Y)$ in terms of the relative entropy

$$D(p(x,y)||p(x)p(y))$$

5.22 Figure 5.10 depicts the variation of the channel capacity of a binary symmetric channel with the transition probability p. Use the results of Problem 5.19 to explain this variation.

5.23 Consider the binary symmetric channel described in Figure 5.8. Let p_0 denote the probability of sending binary symbol $x_0 = 0$ and let $p_1 = 1 - p_0$ denote the probability of sending binary symbol $x_1 = 1$. Let p denote the transition probability of the channel.

a. Show that the mutual information between the channel input and channel output is given by

$$I(X;Y) = H(z) - H(p)$$

where the two entropy functions

$$H(z) = z \log_2\left(\frac{1}{z}\right) + (1-z) \log_2\left(\frac{1}{1-z}\right)$$

$$z = p_0 p + (1-p_0)(1-p)$$

and

$$H(p) = p \log_2\left(\frac{1}{p}\right) + (1-p) \log_2\left(\frac{1}{1-p}\right)$$

b. Show that the value of p_0 that maximizes $I(X;Y)$ is equal to 1/2.

c. Hence, show that the channel capacity equals

$$C = 1 - H(p)$$

5.24 Two binary symmetric channels are connected in cascade as shown in Figure P5.24. Find the overall channel capacity of the cascaded connection, assuming that both channels have the same transition probability diagram of Figure 5.8.

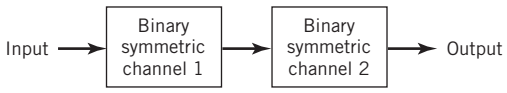


Figure P5.24

5.25 The *binary erasure channel* has two inputs and three outputs as described in Figure P5.25. The inputs are labeled 0 and 1 and the outputs are labeled 0, 1, and e. A fraction α of the incoming bits is erased by the channel. Find the capacity of the channel.

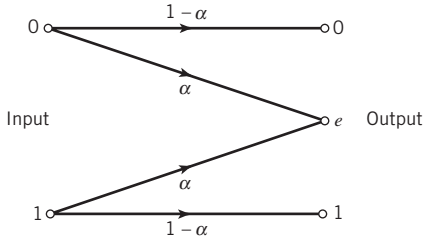


Figure P5.25

5.26 Consider a digital communication system that uses a *repetition code* for the channel encoding/decoding. In particular, each transmission is repeated n times, where $n = 2m + 1$ is an odd integer. The decoder operates as follows. If in a block of n received bits the number of 0s exceeds the number of 1s, then the decoder decides in favor of a 0; otherwise, it decides in favor of a 1. An error occurs when $m + 1$ or more transmissions out of $n = 2m + 1$ are incorrect. Assume a binary symmetric channel.

a. For $n = 3$, show that the average probability of error is given by

$$P_e = 3p^2(1-p) + p^3$$

where p is the transition probability of the channel.

b. For $n = 5$, show that the average probability of error is given by

$$P_e = 10p^3(1-p)^2 + 5p^4(1-p) + p^5$$

c. Hence, for the general case, deduce that the average probability of error is given by

$$P_e = \sum_{i=m+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

5.27 Let X, Y, and Z be three discrete random variables. For each value of the random variable Z, represented by sample z, define

$$A(z) = \sum_x \sum_y p(y)p(z|x,y)$$

Show that the conditional entropy $H(X \mid Y)$ satisfies the inequality

$$H(X|Y) \leq H(z) + \mathbb{E}[\log A]$$

where $\mathbb{E}$ is the expectation operator.

5.28 Consider two correlated discrete random variables X and Y, each of which takes a value in the set $\{x_i\}_{i=1}^n$. Suppose that the value taken by Y is known. The requirement is to guess the value of X. Let P_e denote the probability of error, defined by

$$P_e = \mathbb{P}[X \neq Y]$$

Show that P_e is related to the conditional entropy of X given Y by the inequality

$$H(X|Y) \leq H(P_e) + P_e \log(n-1)$$

This inequality is known as *Fano's inequality. Hint:* Use the result derived in Problem 5.27.

5.29 In this problem we explore the *convexity* of the mutual information $I(X;Y)$, involving the pair of discrete random variables X and Y.

Consider a discrete memoryless channel, for which the transition probability $p(y|x)$ is fixed for all x and y. Let X_1 and X_2 be two input random variables, whose input probability distributions are respectively denoted by $p(x_1)$ and $p(x_2)$. The corresponding probability distribution of X is defined by the convex combination

$$p(x) = a_1 p(x_1) + a_2 p(x_2)$$

where a_1 and a_2 are arbitrary constants. Prove the inequality

$$I(X;Y) \geq a_1 I(X_1;Y_1) + a_2 I(X_2;Y_2)$$

where X_1, X_2, and X are the channel inputs, and Y_1, Y_2, and Y are the corresponding channel outputs.

For the proof, you may use the following form of *Jensen's inequality*:

$$\sum_x \sum_y p_1(x,y) \log\left(\frac{p(y)}{p_1(y)}\right) \leq \log\left[\sum_x \sum_y p_1(x,y)\left(\frac{p(y)}{p_1(y)}\right)\right]$$

Differential Entropy

5.30 The differential entropy of a continuous random variable X is defined by the integral of (5.66). Similarly, the differential entropy of a continuous random vector $\mathbf{X}$ is defined by the integral of (5.68). These two integrals may not exist. Justify this statement.

5.31 Show that the differential entropy of a continuous random variable X is invariant to translation; that is,

$$h(X + c) = h(X)$$

for some constant c.

5.32 Let $X_1, X_2, \ldots, X_n$ denote the elements of a Gaussian vector $\mathbf{X}$. The X_i are independent with mean m_i and variance σ_i^2, $i = 1, 2, \ldots, n$. Show that the differential entropy of the vector $\mathbf{X}$ is given by

$$h(\mathbf{X}) = \frac{n}{2}\log_2[2\pi e(\sigma_1^2\sigma_2^2\ldots\sigma_n^2)^{1/n}]$$

where e is the base of the natural logarithm.What does $h(\mathbf{X})$ reduce to if the variances are all equal?

5.33 A continuous random variable X is constrained to a peak magnitude M; that is,

$$-M < X < M$$

a. Show that the differential entropy of X is maximum when it is uniformly distributed, as shown by

$$f_X(x) = \begin{cases} 1/(2M), & -M < x \leq M \\ 0, & \text{otherwise} \end{cases}$$

b. Determine the maximum differential entropy of X.

5.34 Referring to (5.75), do the following:

a. Verify that the differential entropy of a Gaussian random variable of mean μ and variance σ^2 is given by $1/2\ \log_2(2\pi\ e\sigma^2)$, where e is the base of the natural algorithm.

b. Hence, confirm the inequality of (5.75).

5.35 Demonstrate the properties of symmetry, nonnegativity, and expansion of the mutual information $I(X;Y)$ described in Section 5.6.

5.36 Consider the continuous random variable Y, defined by

$$Y = X + N$$

where the random variables X and N are statistically independent. Show that the conditional differential entropy of Y, given X, equals

$$h(Y \mid X) = h(N)$$

where $h(N)$ is the differential entropy of N.

Information Capacity Law

5.37 A voice-grade channel of the telephone network has a bandwidth of 3.4 kHz.

a. Calculate the information capacity of the telephone channel for a signal-to-noise ratio of 30 dB.

b. Calculate the minimum signal-to-noise ratio required to support information transmission through the telephone channel at the rate of 9600 bits/s.

5.38 Alphanumeric data are entered into a computer from a remote terminal through a voice-grade telephone channel. The channel has a bandwidth of 3.4 kHz and output signal-to-noise ratio of 20 dB. The terminal has a total of 128 symbols. Assume that the symbols are equiprobable and the successive transmissions are statistically independent.

a. Calculate the information capacity of the channel.

b. Calculate the maximum symbol rate for which error-free transmission over the channel is possible.

5.39 A black-and-white television picture may be viewed as consisting of approximately 3×10^5 elements, each of which may occupy one of 10 distinct brightness levels with equal probability. Assume that (1) the rate of transmission is 30 picture frames per second and (2) the signal-to-noise ratio is 30 dB.

Using the information capacity law, calculate the minimum bandwidth required to support the transmission of the resulting video signal.

5.40 In Section 5.10 we made the statement that it is easier to increase the information capacity of a communication channel by expanding its bandwidth B than increasing the transmitted power for a prescribed noise variance N_0B. This statement assumes that the noise spectral density N_0 varies inversely with B. Why is this inverse relationship the case?

5.41 In this problem, we revisit Example 5.10, which deals with coded binary antipodal signaling over an additive white Gaussian noise (AWGN) channel. Starting with (5.105) and the underlying theory, develop a software package for computing the minimum E_b/N_0 required for a given bit error rate, where E_b is the signal energy per bit, and $N_0/2$ is the noise spectral density. Hence, compute the results plotted in parts *a* and *b* of Figure 5.16.

As mentioned in Example 5.10, the computation of the mutual information between the channel input and channel output is well approximated using Monte Carlo integration. To explain how this method works, consider a function $g(y)$ that is difficult to sample randomly, which is indeed the case for the problem at hand. (For this problem, the function $g(y)$ represents the complicated integrand in the formula for the differential entropy of the channel output.) For the computation, proceed as follows:

- Find an area A that includes the region of interest and that is easily sampled.
- Choose N points, uniformly randomly inside the area A.

Then the *Monte Carlo integration theorem* states that the integral of the function $g(y)$ with respect to y is approximately equal to the area A multiplied by the fraction of points that reside below the curve of g, as illustrated in Figure P5.41. The accuracy of the approximation improves with increasing N.

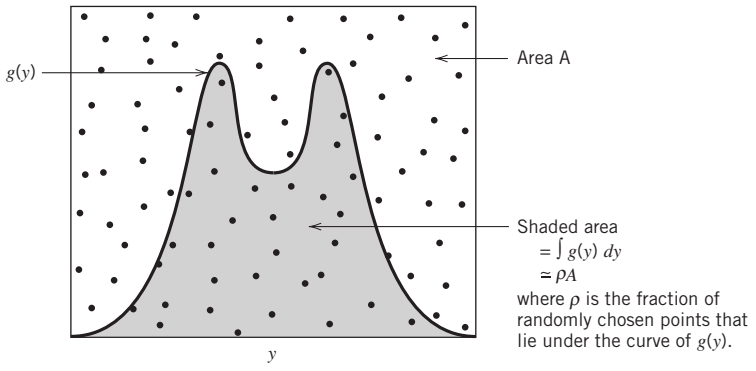


Figure P5.41

Notes

1. According to Lucky (1989), the first mention of the term *information theory* by Shannon occurred in a 1945 memorandum entitled "A mathematical theory of cryptography." It is rather curious that the term was never used in Shannon's (1948) classic paper, which laid down the foundations of information theory. For an introductory treatment of information theory, see Part 1 of the book by McEliece (2004), Chapters 1–6. For an advanced treatment of this subject, viewed in a rather broad context and treated with rigor, and clarity of presentation, see Cover and Thomas (2006).

For a collection of papers on the development of information theory (including the 1948 classic paper by Shannon), see Slepian (1974). For a collection of the original papers published by Shannon, see Sloane and Wyner (1993).

2. The use of a logarithmic measure of information was first suggested by Hartley (1928); however, Hartley used logarithms to base 10.

3. In statistical physics, the entropy of a physical system is defined by (Rief, 1965: 147)

$$L = k_B \ln \Omega$$

where k_B is *Boltzmann's constant*, Ω is the number of states accessible to the system, and ln denotes the natural logarithm. This entropy has the dimensions of energy, because its definition involves the constant k_B. In particular, it provides a *quantitative measure of the degree of randomness of the system*. Comparing the entropy of statistical physics with that of information theory, we see that they have a similar form.

4. For the original proof of the source coding theorem, see Shannon (1948). A general proof of the source coding theorem is also given in Cover and Thomas (2006). The source coding theorem is also referred to in the literature as the *noiseless coding theorem*, noiseless in the sense that it establishes the condition for error-free encoding to be possible.

5. For proof of the Kraft inequality, see Cover and Thomas (2006). The Kraft inequality is also referred to as the Kraft–McMillan inequality in the literature.

6. The Huffman code is named after its inventor D.A. Huffman (1952). For a detailed account of Huffman coding and its use in data compaction, see Cover and Thomas (2006).

7. The original papers on the Lempel–Ziv algorithm are Ziv and Lempel (1977, 1978). For detailed treatment of the algorithm, see Cover and Thomas (2006).

8. It is also of interest to note that once a "parent" subsequence is joined by its two children, that parent subsequence can be replaced in constructing the Lempel–Ziv algorithm. To illustrate this nice feature of the algorithm, suppose we have the following example sequence:

$$01, 010, 011, \ldots$$

where 01 plays the role of a parent and 010 and 011 play the roles of the parent's children. In this example, the algorithm removes the 01, thereby reducing the length of the table through the use of a pointer.

9. In Cover and Thomas (2006), it is proved that the two-stage method, where the source coding and channel coding are considered separately as depicted in Figure 5.11, is as good as any other method of transmitting information across a noisy channel. This result has practical implications, in that the design of a communication system may be approached in two separate parts: source coding followed by channel coding. Specifically, we may proceed as follows:

- Design a source code for the most efficient representation of data generated by a discrete memoryless source of information.
- Separately and independently, design a channel code that is appropriate for a discrete memoryless channel.

The combination of source coding and channel coding designed in this manner will be as efficient as anything that could be designed by considering the two coding problems jointly.

10. To prove the channel-coding theorem, Shannon used several ideas that were new at the time; however, it was some time later when the proof was made rigorous (Cover and Thomas, 2006: 199).

Perhaps the most thoroughly rigorous proof of this basic theorem of information theory is presented in Chapter 7 of the book by Cover and Thomas (2006). Our statement of the theorem, though slightly different from that presented by Cover and Thomas, in essence is the same.

11. In the literature, the relative entropy is also referred to as the *Kullback–Leibler divergence* (KLD).

12. Equation (5.95) is also referred to in the literature as the *Shannon–Hartley law* in recognition of the early work by Hartley on information transmission (Hartley, 1928). In particular, Hartley showed that the amount of information that can be transmitted over a given channel is proportional to the product of the channel bandwidth and the time of operation.

13. A lucid exposition of sphere packing is presented in Cover and Thomas (2006); see also Wozencraft and Jacobs (1965).

14. Parts a and b of Figure 5.16 follow the corresponding parts of Figure 6.2 in the book by Frey (1998).

15. For a rigorous treatment of information capacity of a colored noisy channel, see Gallager (1968). The idea of replacing the channel model of Figure 5.17a with that of Figure 5.17b is discussed in Gitlin, Hayes, and Weinstein (1992)

16. For a complete treatment of rate distortion theory, see the classic book by Berger (1971); this subject is also treated in somewhat less detail in Cover and Thomas (1991), McEliece (1977), and Gallager (1968).

17. For the derivation of (5.124), see Cover and Thomas (2006). An algorithm for computation of the rate distortion function $R(D)$ defined in (5.124) is described in Blahut (1987) and Cover and Thomas (2006).

CHAPTER
10

Error-Control Coding

10.1 Introduction

In the previous three chapters we studied the important issue of data transmission over communication channels under three different channel-impairment scenarios:

- In Chapter 7 the focus of attention was on the kind of channels where AWGN is the main source of channel impairment. An example of this first scenario is a satellite-communication channel.
- In Chapter 8 the focus of attention was intersymbol interference as the main source of channel impairment. An example of this second scenario is the telephone channel.
- Then, in Chapter 9, we focused on multipath as a source of channel impairment. An example for this third scenario is the wireless channel.

Although, indeed, these three scenarios are naturally quite different from each other, they do share a common practical shortcoming: *reliability*. This is where the need for error-control coding, the topic of this chapter, assumes paramount importance.

Given these physical realities, the task facing the designer of a digital communication system is that of providing a cost-effective facility for transmitting information from one end of the system at a rate and level of reliability and quality that are acceptable to a user at the other end.

From a communication theoretic perspective, the key system parameters available for achieving these practical requirements are limited to two:

- transmitted signal power, and
- channel bandwidth.

These two parameters, together with the power spectral density of receiver noise, determine the signal energy per bit-to-noise power spectral density ratio, E_b/N_0. In Chapter 7 we showed that this ratio uniquely determines the BER produced by a particular modulation scheme operating over a Gaussian noise channel. Practical considerations usually place a limit on the value that we can assign to E_b/N_0. To be specific, in practice, we often arrive at a modulation scheme and find that it is not possible to provide acceptable data quality (i.e., low enough error performance). For a fixed E_b/N_0, the only practical option available for changing data quality from problematic to acceptable is to use *error-control coding*, which is the focus of attention in this chapter. In simple terms, by incorporating a fixed number of redundant bits into the structure of a codeword at the transmitter, it is feasible to provide reliable communication over a noisy channel, provided

that Shannon's code theorem, discussed in Chapter 5, is satisfied. In effect, channel bandwidth is traded off for reliable communication.

Another practical motivation for the use of coding is to reduce the required E_b/N_0 for a fixed BER. This reduction in E_b/N_0 may, in turn, be exploited to reduce the required transmitted power or reduce the hardware costs by requiring a smaller antenna size in the case of radio communications.

10.2 Error Control Using Forward Error Correction

Error control for data integrity may be exercised by means of *forward error correction* (FEC).[1] Figure 10.1a shows the model of a digital communication system using such an approach. The discrete source generates information in the form of binary symbols. The *channel encoder* in the transmitter accepts message bits and adds *redundancy* according to a prescribed rule, thereby producing an encoded data stream at a higher bit rate. The *channel decoder* in the receiver exploits the redundancy to decide which message bits in the original data stream, given a noisy version of the encoded data stream, were actually transmitted. The combined goal of the channel encoder and decoder is to minimize the effect of channel noise. That is, the number of errors between the channel encoder input (derived from the source) and the channel decoder output (delivered to the user) is minimized.

For a fixed modulation scheme, the addition of redundancy in the coded messages implies the need for *increased transmission bandwidth*. Moreover, the use of error-control coding adds *complexity* to the system. Thus, the design trade-offs in the use of error-control coding to achieve acceptable error performance include considerations of bandwidth and system complexity.

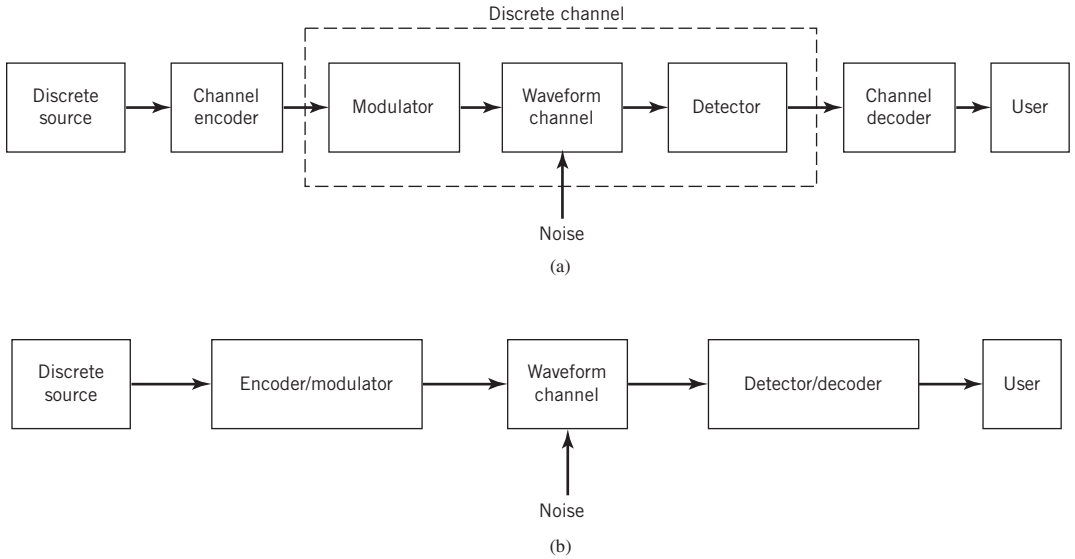


Figure 10.1 Simplified models of a digital communication system. (a) Coding and modulation performed separately. (b) Coding and modulation combined.

There are many different error-correcting codes (with roots in diverse mathematical disciplines) that we can use. Historically, these codes have been classified into *block codes* and *convolutional codes*. The distinguishing feature for this particular classification is the presence or absence of *memory* in the encoders for the two codes.

To generate an (n, k) block code, the channel encoder accepts information in successive k-bit *blocks*; for each block, it adds $n - k$ redundant bits that are algebraically related to the k message bits, thereby producing an overall encoded block of n bits, where $n > k$. The n-bit block is called a *codeword*, and n is called the *block length* of the code. The channel encoder produces bits at the rate $R_0 = (n/k)R_s$, where R_s is the bit rate of the information source. The dimensionless ratio $r = k/n$ is called the *code rate*, where $0 < r < 1$. The bit rate R_0, coming out of the encoder, is called the *channel data rate*. Thus, the code rate is a dimensionless ratio, whereas the data rate produced by the source and the channel data rate produced by the encoder are both measured in bits per second.

In a convolutional code, the encoding operation may be viewed as the *discrete-time convolution* of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a "sliding window" equal in duration to its own memory. This, in turn, means that in a convolutional code, unlike in a block code, the channel encoder accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.

In the model depicted in Figure 10.1a, the operations of channel coding and modulation are performed separately in the transmitter; and likewise for the operations of detection and decoding in the receiver. When, however, bandwidth efficiency is of major concern, the most effective method of implementing forward error-control correction coding is to combine it with modulation as a single function, as shown in Figure 10.1b. In this second approach, coding is redefined as a process of imposing certain patterns on the transmitted signal and the resulting code is called a *trellis code*.

Block codes, convolutional codes, and trellis codes represent the *classical family of codes* that follow traditional approaches rooted in algebraic mathematics in one form or another. In addition to these classical codes, we now have a "new" generation of coding techniques exemplified by *turbo codes* and *low-density parity-check* (*LDPC*) *codes*. These new codes are not only fundamentally different, but they have also already taken over the legacy coding schemes very quickly in many practical systems. Simply put, turbo codes and LDPC codes are structured in such a way that decoding can be split into a number of manageable steps, thereby making it possible to *construct powerful codes in a computationally feasible manner*, which is not attainable with the legacy codes. Turbo codes and LDPC codes are discussed in the latter part of the chapter.

10.3 Discrete Memoryless Channels

Returning to the model of Figure 10.1a, the waveform channel is said to be *memoryless* if in a given interval the detector output depends only on the signal transmitted in that interval and not on any previous transmission. Under this condition, we may model the combination of the modulator, the waveform channel, and the demodulator (detector) as a *discrete memoryless channel*. Such a channel is completely described by the set of *transition probabilities* denoted by $p(j|i)$, where i denotes a modulator input symbol, j

denotes a demodulator output symbol, and $p(j|i)$ is the probability of receiving symbol j given that symbol i was sent. (Discrete memoryless channels were described previously at some length in Chapter 5 on information theory.)

The simplest discrete memoryless channel results from the use of binary input and binary output symbols. When binary coding is used, the modulator has only the binary symbols 0 and 1 as inputs. Likewise, the decoder has only binary inputs if binary quantization of the demodulator output is used; that is, a *hard decision* is made on the demodulator output as to which binary symbol was actually transmitted. In this situation, we have a *binary symmetric channel* with a *transition probability diagram* as shown in Figure 10.2. From Chapter 5, we recall that the binary symmetric channel, assuming a channel noise modeled as AWGN, is completely described by the *transition probability.* Hard-decision decoding takes advantage of the special algebraic structure that is built into the design of channel codes; the decoding is therefore relatively easy to perform.

However, the use of hard decisions prior to decoding causes an irreversible loss of valuable information in the receiver. To reduce this loss, *soft-decision* coding can be used. This is achieved by including a multilevel quantizer at the demodulator output, as illustrated in Figure 10.3a for the case of binary PSK signals. The input–output characteristic of the quantizer is shown in Figure 10.3b. The modulator has only binary symbols 0 and 1 as inputs, but the demodulator output now has an alphabet with Q symbols. Assuming the use of the three-level quantizer described in Figure 10.3b, we have $Q = 8$. Such a channel is called a *binary input*, *Q-ary output discrete memoryless channel*. The corresponding channel transition probability diagram is shown in Figure 10.3c. The form of this distribution, and consequently the decoder performance, depends on the location of the representation levels of the quantizer, which, in turn, depends on the signal level and noise variance. Accordingly, the demodulator must incorporate automatic gain control if an effective multilevel quantizer is to be realized. Moreover, the use of soft decisions complicates the implementation of the decoder. Nevertheless, soft-decision decoding offers significant improvement in performance over hard-decision decoding by taking a probabilistic rather than an algebraic approach. It is for this reason that soft-decision decoders are also referred to as *probabilistic decoders*.

Channel Coding Theorem Revisited

In Chapter 5 on information theory we established the concept of *channel capacity*, which, for a discrete memoryless channel, represents the maximum amount of information that

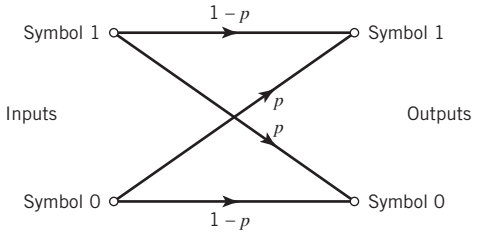


Figure 10.2 Transition probability diagram of binary symmetric channel.

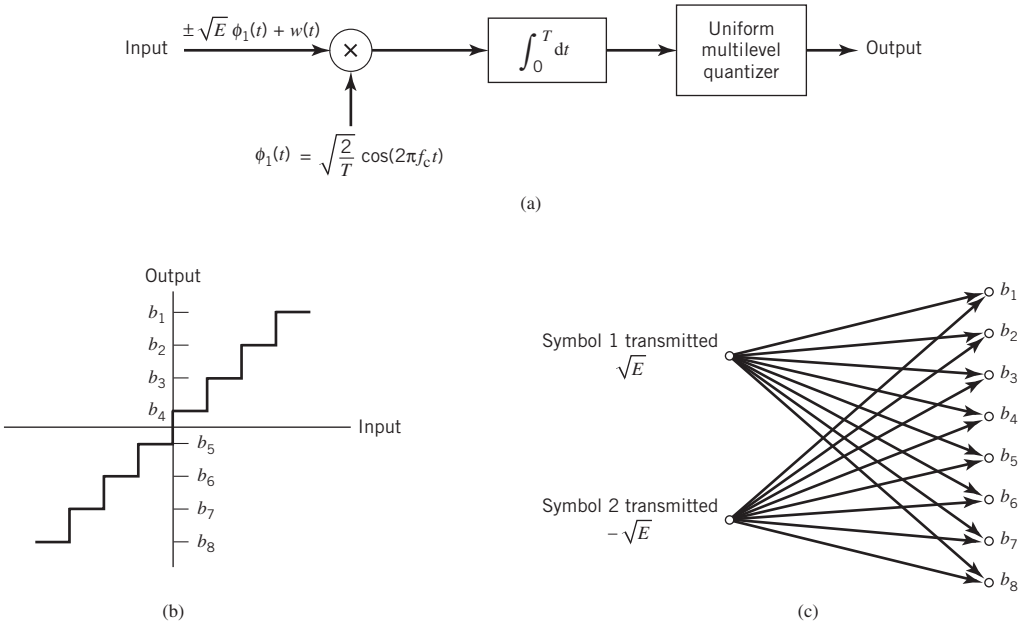


Figure 10.3 Binary input, Q-ary output discrete memoryless channel. (a) Receiver for binary PSK. (b) Transfer characteristic of a multilevel quantizer. (c) Channel transition probability diagram. Parts (b) and (c) are illustrated for eight levels of quantization.

can be transmitted per channel use in a reliable manner. The *channel coding theorem* states:

> If a discrete memoryless channel has capacity C and a source generates information at a rate less than C, then there exists a coding technique such that the output of the source may be transmitted over the channel with an arbitrarily low probability of symbol error.

For the special case of a binary symmetric channel, the theorem teaches us that if the code rate r is less than the channel capacity C, then it is possible to find a code that achieves error-free transmission over the channel. Conversely, it is not possible to find such a code if the code rate r is greater than the channel capacity C. Thus, the channel coding theorem specifies the channel capacity C as a *fundamental limit* on the rate at which the transmission of reliable (error-free) messages can take place over a discrete memoryless channel. The issue that matters here is not the SNR, so long as it is large enough, but how the channel input is encoded.

The most unsatisfactory feature of the channel coding theorem, however, is its nonconstructive nature. The theorem asserts the existence of good codes but does not tell us how to find them. By *good codes* we mean families of channel codes that are capable of providing reliable transmission of information (i.e., at arbitrarily small probability of symbol error) over a noisy channel of interest at bit rates up to a maximum value less than the capacity of that channel. The error-control coding techniques described in this chapter provide different methods of designing good codes.

Notation

Many of the codes described in this chapter are *binary codes*, for which the alphabet consists only of binary symbols 0 and 1. In such a code, the encoding and decoding functions involve the binary arithmetic operations of *modulo-2 addition and multiplication* performed on codewords in the code.

Throughout this chapter, we use the ordinary plus sign (+) to denote modulo-2 addition. The use of this terminology will not lead to confusion because the whole chapter relies on binary arithmetic. In so doing, we avoid use of the special symbol $\oplus$, as we did in previous parts of the book. Thus, according to the notation used in this chapter, the rules for modulo-2 addition are as follows:

$$0 + 0 = 0$$
$$1 + 0 = 1$$
$$0 + 1 = 1$$
$$1 + 1 = 0$$

Because $1 + 1 = 0$, it follows that $1 = -1$. Hence, in binary arithmetic, subtraction is the same as addition. The rules for modulo-2 multiplication are as follows:

$$0 \times 0 = 0$$
$$1 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 1 = 1$$

Division is trivial, in that we have

$$1 \div 1 = 1$$
$$0 \div 1 = 0$$

and division by 0 is not permitted. Modulo-2 addition is the EXCLUSIVE-OR operation in logic and modulo-2 multiplication is the AND operation.

10.4 Linear Block Codes

By definition:

> A code is said to be linear if any two codewords in the code can be added in modulo-2 arithmetic to produce a third codeword in the code.

Consider, then, an (n,k) linear block code, in which k bits of the n code bits are always identical to the message sequence to be transmitted. The $(n - k)$ bits in the remaining portion are computed from the message bits in accordance with a prescribed encoding rule that determines the mathematical structure of the code. Accordingly, these $(n - k)$ bits are referred to as *parity-check bits*. Block codes in which the message bits are transmitted in unaltered form are called *systematic codes*. For applications requiring *both* error detection and error correction, the use of systematic block codes simplifies implementation of the decoder.

Let $m_0, m_1, \ldots, m_{k-1}$ constitute a block of k arbitrary message bits. Thus, we have 2^k distinct message blocks. Let this sequence of message bits be applied to a linear block

encoder, producing an n-bit codeword whose elements are denoted by $c_0, c_1, \ldots, c_{n-1}$. Let $b_0, b_1, \ldots, b_{n-k-1}$ denote the $(n-k)$ *parity-check bits* in the codeword. For the code to possess a systematic structure, a codeword is divided into two parts, one of which is occupied by the message bits and the other by the parity-check bits. Clearly, we have the option of sending the message bits of a codeword before the parity-check bits, or vice versa. The former option is illustrated in Figure 10.4, and its use is assumed in the following.

According to the representation of Figure 10.4, the $(n-k)$ leftmost bits of a codeword are identical to the corresponding parity-check bits and the k rightmost bits of the codeword are identical to the corresponding message bits. We may therefore write

$$c_i = \begin{cases} b_i, & i = 0, 1, \ldots, n-k-1 \\ m_{i+k-n}, & i = n-k, n-k+1, \ldots, n-1 \end{cases} \tag{10.1}$$

The $(n-k)$ parity-check bits are *linear sums* of the k message bits, as shown by the generalized relation

$$b_i = p_{0i}m_0 + p_{1i}m_1 + \ldots + p_{k-1,\,i}m_{k-1} \tag{10.2}$$

where the coefficients are defined as follows:

$$p_{ij} = \begin{cases} 1 & \text{if } b_i \text{ depends on } m_j \\ 0 & \text{otherwise} \end{cases} \tag{10.3}$$

The coefficients p_{ij} are chosen in such a way that the rows of the generator matrix are linearly independent and the parity-check equations are *unique*. The p_{ij} used here should not be confused with the $p(j|i)$ introduced in Section 10.3.

The system of (10.1) and (10.2) defines the mathematical structure of the (n,k) linear block code. This system of equations may be rewritten in a compact form using matrix notation. To proceed with this reformulation, we respectively define the 1-by-k *message vector* $\mathbf{m}$, the 1-by-$(n-k)$ parity-check vector $\mathbf{b}$, and the 1-by-n code vector $\mathbf{c}$ as follows:

$$\mathbf{m} = [m_0, m_1, \ldots, m_{k-1}] \tag{10.4}$$

$$\mathbf{b} = [b_0, b_1, \ldots, b_{n-k-1}] \tag{10.5}$$

$$\mathbf{c} = [c_0, c_1, \ldots, c_{n-1}] \tag{10.6}$$

Note that all three vectors are *row vectors*. The use of row vectors is adopted in this chapter for the sake of being consistent with the notation commonly used in the coding literature. We may thus rewrite the set of simultaneous equations defining the parity check-bits in the compact matrix form

$$\mathbf{b} = \mathbf{mP} \tag{10.7}$$

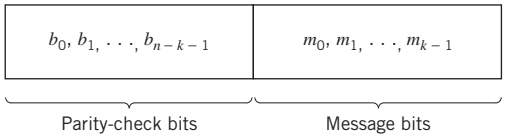


Figure 10.4 Structure of systematic codeword.

The **P** in (10.7) is the k-by-$(n-k)$ *coefficient matrix* defined by

$$\mathbf{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,\,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,\,n-k-1} \\ \vdots & \vdots & & \vdots \\ p_{k-1,\,0} & p_{k-1,\,1} & \cdots & p_{k-1,\,n-k-1} \end{bmatrix} \tag{10.8}$$

where the element p_{ij} is 0 or 1.

From the definitions given in (10.4)–(10.6), we see that **c** may be expressed as a partitioned row vector in terms of the vectors **m** and **b** as follows:

$$\mathbf{c} = \begin{bmatrix} \mathbf{b} & \vdots & \mathbf{m} \end{bmatrix} \tag{10.9}$$

Hence, substituting (10.7) into (10.9) and factoring out the common message vector **m**, we get

$$\mathbf{c} = \mathbf{m}\begin{bmatrix} \mathbf{P} & \vdots & \mathbf{I}_k \end{bmatrix} \tag{10.10}$$

where $\mathbf{I}_k$ is the k-by-k *identity matrix*:

$$\mathbf{I}_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{10.11}$$

Define the k-by-n *generator matrix*

$$\mathbf{G} = \begin{bmatrix} \mathbf{P} & \vdots & \mathbf{I}_k \end{bmatrix} \tag{10.12}$$

The generator matrix **G** of (10.12) is said to be in the *canonical form*, in that its k rows are linearly independent; that is, it is not possible to express any row of the matrix **G** as a linear combination of the remaining rows. Using the definition of the generator matrix **G**, we may simplify (10.10) as

$$\mathbf{c} = \mathbf{mG} \tag{10.13}$$

The full set of codewords, referred to simply as *the code*, is generated in accordance with (10.13) by passing the message vector **m** range through the set of all 2^k binary k-tuples (1-by-k vectors). Moreover, the sum of any two codewords in the code is another codeword. This basic property of linear block codes is called *closure*. To prove its validity, consider a pair of code vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ corresponding to a pair of message vectors $\mathbf{m}_i$ and $\mathbf{m}_j$, respectively. Using (10.13), we may express the sum of $\mathbf{c}_i$ and $\mathbf{c}_j$ as

$$\begin{aligned} \mathbf{c}_i + \mathbf{c}_j &= \mathbf{m}_i\mathbf{G} + \mathbf{m}_j\mathbf{G} \\ &= (\mathbf{m}_i + \mathbf{m}_j)\mathbf{G} \end{aligned}$$

The modulo-2 sum of $\mathbf{m}_i$ and $\mathbf{m}_j$ represents a new message vector. Correspondingly, the modulo-2 sum of $\mathbf{c}_i$ and $\mathbf{c}_j$ represents a new code vector.

There is another way of expressing the relationship between the message bits and parity-check bits of a linear block code. Let **H** denote an $(n-k)$-by-n matrix, defined as

$$\mathbf{H} = \left[\mathbf{I}_{n-k} \;\vdots\; \mathbf{P}^{\mathrm{T}}\right] \tag{10.14}$$

where $\mathbf{P}^{\mathrm{T}}$ is an $(n-k)$-by-k matrix, representing the transpose of the coefficient matrix $\mathbf{P}$, and $\mathbf{I}_{n-k}$ is the $(n-k)$-by-$(n-k)$ identity matrix. Accordingly, we may perform the following multiplication of partitioned matrices:

$$\begin{aligned}\mathbf{H}\mathbf{G}^{\mathrm{T}} &= \left[\mathbf{I}_{n-k} \;\vdots\; \mathbf{P}^{\mathrm{T}}\right]\begin{bmatrix}\mathbf{P}^{\mathrm{T}}\\ \text{----}\\ \mathbf{I}_k\end{bmatrix}\\ &= \mathbf{P}^{\mathrm{T}} + \mathbf{P}^{\mathrm{T}}\end{aligned}$$

where we have used the fact that multiplication of a rectangular matrix by an identity matrix of compatible dimensions leaves the matrix unchanged. In modulo-2 arithmetic, the matrix sum $\mathbf{P}^{\mathrm{T}} + \mathbf{P}^{\mathrm{T}}$ is $\mathbf{0}$. We therefore have

$$\mathbf{H}\mathbf{G}^{\mathrm{T}} = \mathbf{0} \tag{10.15}$$

Equivalently, we have $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, where $\mathbf{0}$ is a new null matrix. Postmultiplying both sides of (10.13) by $\mathbf{H}^{\mathrm{T}}$, the transpose of $\mathbf{H}$, and then using (10.15), we get the inner product

$$\begin{aligned}\mathbf{c}\mathbf{H}^{\mathrm{T}} &= \mathbf{m}\mathbf{G}\mathbf{H}^{\mathrm{T}}\\ &= \mathbf{0}\end{aligned} \tag{10.16}$$

The matrix $\mathbf{H}$ is called the *parity-check matrix* of the code and the equations specified by (10.16) are called *parity-check equations*.

The generator equation (10.13) and the parity-check detector equation (10.16) are basic to the description and operation of a linear block code. These two equations are depicted in the form of block diagrams in Figure 10.5a and b, respectively.

Syndrome: Definition and Properties

The generator matrix $\mathbf{G}$ is used in the encoding operation at the transmitter. On the other hand, the parity-check matrix $\mathbf{H}$ is used in the decoding operation at the receiver. In the context of the latter operation, let $\mathbf{r}$ denote the 1-by-n *received vector* that results from

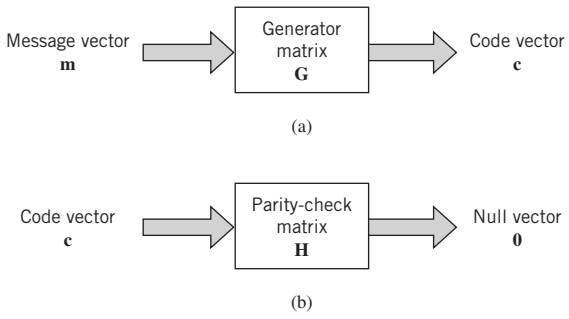


Figure 10.5 Block diagram representations of the generator equation (10.13) and the parity-check equation (10.16).

sending the code vector $\mathbf{c}$ over a noisy binary channel. We express the vector $\mathbf{r}$ as the sum of the original code vector $\mathbf{c}$ and a new vector $\mathbf{e}$, as shown by

$$\mathbf{r} = \mathbf{c} + \mathbf{e} \tag{10.17}$$

The vector $\mathbf{e}$ is called the *error vector* or *error pattern*. The ith element of $\mathbf{e}$ equals 0 if the corresponding element of $\mathbf{r}$ is the same as that of $\mathbf{c}$. On the other hand, the ith element of $\mathbf{e}$ equals 1 if the corresponding element of $\mathbf{r}$ is different from that of $\mathbf{c}$, in which case an *error* is said to have occurred in the ith location. That is, for $i = 1, 2, \ldots, n$, we have

$$e_i = \begin{cases} 1 & \text{if an error has occurred in the } i\text{th location} \\ 0 & \text{otherwise} \end{cases} \tag{10.18}$$

The receiver has the task of decoding the code vector $\mathbf{c}$ from the received vector $\mathbf{r}$. The algorithm commonly used to perform this decoding operation starts with the computation of a 1-by-$(n - k)$ vector called the *error-syndrome vector* or simply the *syndrome*.[2] The importance of the syndrome lies in the fact that it depends only upon the error pattern.

Given a 1-by-n received vector $\mathbf{r}$, the corresponding syndrome is formally defined as

$$\mathbf{s} = \mathbf{r}\mathbf{H}^{\mathrm{T}} \tag{10.19}$$

Accordingly, the syndrome has the following important properties.

PROPERTY 1 *The syndrome depends only on the error pattern and not on the transmitted codeword.*

To prove this property, we first use (10.17) and (10.19), and then (10.16) to write

$$\begin{aligned} \mathbf{s} &= (\mathbf{c} + \mathbf{e})\mathbf{H}^{\mathrm{T}} \\ &= \mathbf{c}\mathbf{H}^{\mathrm{T}} + \mathbf{e}\mathbf{H}^{\mathrm{T}} \\ &= \mathbf{e}\mathbf{H}^{\mathrm{T}} \end{aligned} \tag{10.20}$$

Hence, the parity-check matrix $\mathbf{H}$ of a code permits us to compute the syndrome $\mathbf{s}$, which depends only upon the error pattern $\mathbf{e}$.

To expand on Property 1, suppose that the error pattern $\mathbf{e}$ contains a pair of errors in locations i and j caused by the additive channel noise, as shown by

$$\mathbf{e} = [0 \ldots 01_i 0 \ldots 01_j 0 \ldots 0]$$

Then, substituting this error pattern into (10.20) yields the syndrome

$$\mathbf{s} = \mathbf{h}_i + \mathbf{h}_j$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are respectively the ith and jth rows of the matrix $\mathbf{H}^{\mathrm{T}}$. In words, we may state the following *corollary* to Property 1:

> For a linear block code, the syndrome $\mathbf{s}$ is equal to the sum of those rows of the transposed parity-check matrix $\mathbf{H}^{\mathrm{T}}$ where errors have occurred due to channel noise.

PROPERTY 2 *All error patterns that differ by a codeword have the same syndrome.*

For k message bits, there are 2^k distinct code vectors denoted as $\mathbf{c}_i$, where $i = 0, 1, \ldots, 2^k - 1$. Correspondingly, for any error pattern $\mathbf{e}$ we define the 2^k distinct vectors $\mathbf{e}_i$ as follows

$$\mathbf{e}_i = \mathbf{e} + \mathbf{c}_i \qquad \text{for } i = 0, 1, \ldots, 2^k - 1 \tag{10.21}$$

The set of vectors ($\mathbf{e}_i$, $i = 0, 1, \ldots, 2^k - 1$) defined in (10.21) is called a *coset* of the code. In other words, a coset has exactly 2^k elements that differ at most by a code vector. Thus, an (n,k) linear block code has 2^{n-k} possible cosets. In any event, multiplying both sides of (10.21) by the matrix $\mathbf{H}^T$ and again using (10.16), we get

$$\begin{aligned}\mathbf{e}_i\mathbf{H}^T &= \mathbf{e}\mathbf{H}^T + \mathbf{c}_i\mathbf{H}^T \\ &= \mathbf{e}\mathbf{H}^T\end{aligned} \tag{10.22}$$

which is independent of the index i. Accordingly, we may say:

> Each coset of the code is characterized by a unique syndrome.

We may put Properties 1 and 2 in perspective by expanding (10.20). Specifically, with the matrix $\mathbf{H}$ having the systematic form given in (10.14), where the matrix $\mathbf{P}$ is itself defined by (10.8), we find from (10.20) that the $(n - k)$ elements of the syndrome $\mathbf{s}$ are linear combinations of the n elements of the error pattern $\mathbf{e}$, as shown by

$$\begin{aligned}
s_0 &= e_0 + e_{n-k}p_{00} + e_{n-k+1}p_{10} + \cdots + e_{n-k}p_{k-1,0} \\
s_1 &= e_1 + e_{n-k}p_{01} + e_{n-k+1}p_{11} + \cdots + e_{n-k}p_{k-1,1} \\
&\vdots \\
s_{n-k-1} &= e_{n-k-1} + e_{n-k}p_{0,n-k-1} + \cdots + e_{n-1}p_{(k-1,n-k+1)}
\end{aligned} \tag{10.23}$$

This set of $(n - k)$ linear equations clearly shows that the syndrome contains information about the error pattern and may, therefore, be used for error detection. However, it should be noted that the set of equations (10.23) is *underdetermined*, in that we have more unknowns than equations. Accordingly, there is *no* unique solution for the error pattern. Rather, there are 2^n error patterns that satisfy (10.23) and, therefore, result in the same syndrome, in accordance with Property 2 and (10.22). In particular, with 2^{n-k} possible syndrome vectors, the information contained in the syndrome $\mathbf{s}$ about the error pattern $\mathbf{e}$ is *not* enough for the decoder to compute the exact value of the transmitted code vector. Nevertheless, knowledge of the syndrome $\mathbf{s}$ reduces the search for the true error pattern $\mathbf{e}$ from 2^n to 2^{n-k} possibilities. Given these possibilities, the decoder has the task of making the best selection from the cosets corresponding to $\mathbf{s}$.

Minimum Distance Considerations

Consider a pair of code vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ that have the same number of elements. The *Hamming distance*, denoted by $d(\mathbf{c}_1,\mathbf{c}_2)$, between such a pair of code vectors is defined as the number of locations in which their respective elements differ.

The *Hamming weight* $w(\mathbf{c})$ of a code vector $\mathbf{c}$ is defined as the number of nonzero elements in the code vector. Equivalently, we may state that the Hamming weight of a code vector is the distance between the code vector and the all-zero code vector. In a corresponding way, we may introduce a new parameter called the *minimum distance* $d_{\min}$, for which we make the statement:

> The minimum distance $d_{\min}$ of a linear block code is the smallest Hamming distance between any pair of codewords.

That is, the minimum distance is the same as the smallest Hamming weight of the difference between any pair of code vectors. From the closure property of linear block codes, the sum (or difference) of two code vectors is another code vector. Accordingly, we may also state:

> The minimum distance of a linear block code is the smallest Hamming weight of the nonzero code vectors in the code.

The minimum distance $d_{\min}$ is related to the structure of the parity-check matrix $\mathbf{H}$ of the code in a fundamental way. From (10.16) we know that a linear block code is defined by the set of all code vectors for which $\mathbf{cH}^{\mathrm{T}} = \mathbf{0}$, where $\mathbf{H}^{\mathrm{T}}$ is the transpose of the parity-check matrix $\mathbf{H}$. Let the matrix $\mathbf{H}$ be expressed in terms of its columns as shown by

$$\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n] \tag{10.24}$$

Then, for a code vector $\mathbf{c}$ to satisfy the condition $\mathbf{cH}^{\mathrm{T}} = \mathbf{0}$, the vector $\mathbf{c}$ must have ones in such positions that the corresponding rows of $\mathbf{H}^{\mathrm{T}}$ sum to the zero vector $\mathbf{0}$. However, by definition, the number of ones in a code vector is the Hamming weight of the code vector. Moreover, the smallest Hamming weight of the nonzero code vectors in a linear block code equals the minimum distance of the code. Hence, we have another useful result stated as follows:

> The minimum distance of a linear block code is defined by the minimum number of rows of the matrix $\mathbf{H}^{\mathrm{T}}$ whose sum is equal to the zero vector.

From this discussion, it is apparent that the minimum distance $d_{\min}$ of a linear block code is an important parameter of the code. Specifically, $d_{\min}$ determines the *error-correcting capability of the code*. Suppose an (n,k) linear block code is required to detect and correct all error patterns over a binary symmetric channel, and whose Hamming weight is less than or equal to t. That is, if a code vector $\mathbf{c}_i$ in the code is transmitted and the received vector is $\mathbf{r} = \mathbf{c}_i + \mathbf{e}$, we require that the decoder output $\hat{\mathbf{c}} = \mathbf{c}_i$ whenever the error pattern $\mathbf{e}$ has a Hamming weight

$$w(\mathbf{e}) \leq t$$

We assume that the 2^k code vectors in the code are transmitted with equal probability. The best strategy for the decoder then is to pick the code vector closest to the received vector $\mathbf{r}$; that is, the one for which the Hamming distance $d(\mathbf{c}_i, \mathbf{r})$ is the smallest. With such a strategy, the decoder will be able to detect and correct all error patterns of Hamming weight $w(\mathbf{e})$, provided that the minimum distance of the code is equal to or greater than $2t + 1$. We may demonstrate the validity of this requirement by adopting a geometric interpretation of the problem. In particular, the transmitted 1-by-n code vector and the 1-by-n received vector are represented as points in an n-dimensional space. Suppose that we construct two spheres, each of radius t, around the points that represent code vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ under two different conditions:

1. Let these two spheres be disjoint, as depicted in Figure 10.6a. For this condition to be satisfied, we require that $d(\mathbf{c}_i, \mathbf{c}_j) \geq 2t + 1$. If, then, the code vector $\mathbf{c}_i$ is transmitted and the Hamming distance $d(\mathbf{c}_i, \mathbf{r}) \leq t$, it is clear that the decoder will pick $\mathbf{c}_i$, as it is the code vector closest to the received vector $\mathbf{r}$.
2. If, on the other hand, the Hamming distance $d(\mathbf{c}_i, \mathbf{c}_j) \leq 2t$, the two spheres around $\mathbf{c}_i$ and $\mathbf{c}_j$ intersect, as depicted in Figure 10.6b. In this second situation, we see that if $\mathbf{c}_i$

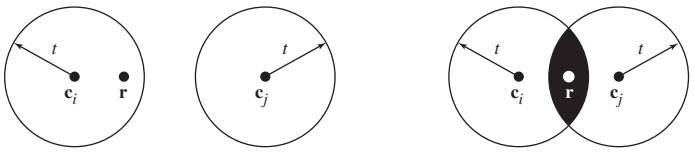


Figure 10.6 (a) Hamming distance $d(\mathbf{c}_i,\mathbf{c}_j) \geq 2t + 1$. (b) Hamming distance $d(\mathbf{c}_i,\mathbf{c}_j) < 2t$. The received vector is denoted by $\mathbf{r}$.

is transmitted, there exists a received vector $\mathbf{r}$ such that the Hamming distance $d(\mathbf{c}_i,\mathbf{r}) \leq t$, yet $\mathbf{r}$ is as close to $\mathbf{c}_j$ as it is to $\mathbf{c}_i$. Clearly, there is now the possibility of the decoder picking the vector $\mathbf{c}_j$, which is wrong.

We thus conclude the ideas presented thus far by saying:

> An (n,k) linear block code has the power to correct all error patterns of weight t or less if, and only if, $d(\mathbf{c}_i,\mathbf{c}_j) \geq 2t + 1$, *for all* $\mathbf{c}_i$ *and* $\mathbf{c}_j$.

By definition, however, the smallest distance between any pair of code vectors in a code is the minimum distance $d_{\min}$ of the code. We may, therefore, go on to state:

> An (n,k) linear block code of minimum distance $d_{\min}$ can correct up to t *errors if, and only if,*
>
> $$t \leq \left\lfloor \frac{1}{2}(d_{\min} - 1) \right\rfloor \tag{10.25}$$
>
> where $\lfloor\ \rfloor$ *denotes the largest integer less than or equal to the enclosed quantity.*

The condition described in (10.25) is important because it gives the error-correcting capability of a linear block code a quantitative meaning.

Syndrome Decoding

We are now ready to describe a syndrome-based decoding scheme for linear block codes. Let $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2^k}$ denote the 2^k code vectors of an (n, k) linear block code. Let $\mathbf{r}$ denote the received vector, which may have one of 2^n possible values. The receiver has the task of partitioning the 2^n possible received vectors into 2^k disjoint subsets $D_1, D_2, \ldots, D_{2^k}$ in such a way that the ith subset D_i corresponds to code vector $\mathbf{c}_i$ for $1 \leq i \leq 2^k$. The received vector $\mathbf{r}$ is decoded into $\mathbf{c}_i$ if it is in the ith subset. For the decoding to be correct, $\mathbf{r}$ must be in the subset that belongs to the code vector $\mathbf{c}_i$ that was actually sent.

The 2^k subsets described herein constitute a *standard array* of the linear block code. To construct it, we exploit the linear structure of the code by proceeding as follows:

1. The 2^k code vectors are placed in a row with the all-zero code vector $\mathbf{c}_1$ as the leftmost element.
2. An error pattern $\mathbf{e}_2$ is picked and placed under $\mathbf{c}_1$, and a second row is formed by adding $\mathbf{e}_2$ to each of the remaining code vectors in the first row; it is important that

$$
\begin{array}{ccccccc}
\mathbf{c}_1 = \mathbf{0} & \mathbf{c}_2 & \mathbf{c}_3 & \cdots & \mathbf{c}_i & \cdots & \mathbf{c}_{2^k} \\
\mathbf{e}_2 & \mathbf{c}_2 + \mathbf{e}_2 & \mathbf{c}_3 + \mathbf{e}_2 & \cdots & \mathbf{c}_i + \mathbf{e}_2 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_2 \\
\mathbf{e}_3 & \mathbf{c}_2 + \mathbf{e}_3 & \mathbf{c}_3 + \mathbf{e}_3 & \cdots & \mathbf{c}_i + \mathbf{e}_3 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_3 \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
\mathbf{e}_j & \mathbf{c}_2 + \mathbf{e}_j & \mathbf{c}_3 + \mathbf{e}_j & \cdots & \mathbf{c}_i + \mathbf{e}_j & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_j \\
\vdots & \vdots & \vdots & & \vdots & & \vdots \\
\mathbf{e}_{2^{n-k}} & \mathbf{c}_2 + \mathbf{e}_{2^{n-k}} & \mathbf{c}_3 + \mathbf{e}_{2^{n-k}} & & \mathbf{c}_i + \mathbf{e}_{2^{n-k}} & & \mathbf{c}_{2^k} + \mathbf{e}_{2^{n-k}}
\end{array}
$$

Figure 10.7 Standard array for an (n,k) block code.

the error pattern chosen as the first element in a row has not previously appeared in the standard array.

3. Step 2 is repeated until all the possible error patterns have been accounted for.

Figure 10.7 illustrates the structure of the standard array so constructed. The 2^k columns of this array represent the disjoint subsets $D_1, D_2, \ldots, D_{2^k}$. The 2^{n-k} rows of the array represent the cosets of the code, and their first elements $\mathbf{e}_2, \ldots, \mathbf{e}_{2^{n-k}}$ are called *coset leaders*.

For a given channel, the probability of decoding error is minimized when the most likely error patterns (i.e., those with the largest probability of occurrence) are chosen as the coset leaders. In the case of a binary symmetric channel, the smaller we make the Hamming weight of an error pattern, the more likely it is for an error to occur. Accordingly, the standard array should be constructed with each coset leader having the minimum Hamming weight in its coset.

We are now ready to describe a decoding procedure for linear block codes:

1. For the received vector $\mathbf{r}$, compute the syndrome $\mathbf{s} = \mathbf{r}\mathbf{H}^T$.
2. Within the coset characterized by the syndrome $\mathbf{s}$, identify the coset leader (i.e., the error pattern with the largest probability of occurrence); call it $\mathbf{e}_0$.
3. Compute the code vector

$$\mathbf{c} = \mathbf{r} + \mathbf{e}_0 \tag{10.26}$$

as the decoded version of the received vector $\mathbf{r}$.

This procedure is called *syndrome decoding*.

EXAMPLE 1

Hamming Codes

For any positive integer $m \geq 3$, there exists a linear block code with the following parameters:

code length	$n = 2^m - 1$
number of message bits	$k = 2^m - m - 1$
number of parity-check bits	$n - k = m$

Such a linear block code for which the error-correcting capability $t = 1$ is called a Hamming code.[3] To be specific, consider the example of $m = 3$, yielding the $(7, 4)$ Hamming code with $n = 7$ and $k = 4$. The generator of this code is defined by

$$\mathbf{G} = \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{I}_k}$$

which conforms to the systematic structure of (10.12).

The corresponding parity-check matrix is given by

$$\mathbf{H} = \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{I}_{n-k}} \underbrace{\begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}}_{\mathbf{P}^{\mathrm{T}}}$$

The operative property embodied in this equation is that the columns of the parity-check matrix $\mathbf{P}$ consist of all the nonzero m-tuples, where $m = 3$.

With $k = 4$, there are $2^k = 16$ distinct message words, which are listed in Table 10.1. For a given message word, the corresponding codeword is obtained by using (10.13). Thus, the application of this equation results in the 16 codewords listed in Table 10.1.

In Table 10.1, we have also listed the Hamming weights of the individual codewords in the (7,4) Hamming code. Since the smallest of the Hamming weights for the nonzero codewords is 3, it follows that the minimum distance of the code is 3, which is what it should be by definition. Indeed, all Hamming codes have the property that the minimum distance $d_{\min} = 3$, independent of the value assigned to the number of parity bits m.

To illustrate the relation between the minimum distance $d_{\min}$ and the structure of the parity-check matrix $\mathbf{H}$, consider the codeword 0110100. In matrix multiplication, defined

Table 10.1 Codewords of a (7,4) Hamming code

Message word	Codeword	Weight of codeword	Message word	Codeword	Weight of codeword
0000	0000000	0	1000	1101000	3
0001	1010001	3	1001	0111001	4
0010	1110010	4	1010	0011010	3
0011	0100011	3	1011	1001011	3
0100	0110100	3	1100	1011100	4
0101	1100101	4	1101	0001101	3
0110	1000110	3	1110	0101110	4
0111	0010111	4	1111	1111111	7

by (10.16), the nonzero elements of this codeword "sift" out the second, third, and fifth columns of the matrix **H**, yielding

$$\begin{bmatrix}0\\1\\0\end{bmatrix}+\begin{bmatrix}0\\0\\1\end{bmatrix}+\begin{bmatrix}0\\1\\1\end{bmatrix}=\begin{bmatrix}0\\0\\0\end{bmatrix}$$

We may perform similar calculations for the remaining 14 nonzero codewords. We thus find that the smallest number of columns in **H** that sums to zero is 3, reconfirming the defining condition $d_{\min} = 3$.

An important property of binary Hamming codes is that they satisfy the condition of (10.25) with the equality sign, assuming that $t = 1$. Thus, assuming single-error patterns, we may formulate the error patterns listed in the right-hand column of Table 10.2. The corresponding eight syndromes, listed in the left-hand column, are calculated in accordance with (10.20). The zero syndrome signifies no transmission errors.

Suppose, for example, the code vector [1110010] is sent and the received vector is [1100010] with an error in the third bit. Using (10.19), the syndrome is calculated to be

$$\mathbf{s} = [1100010]\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\\1&1&0\\0&1&1\\1&1&1\\1&0&1\end{bmatrix}$$

$$= \begin{bmatrix}0&0&1\end{bmatrix}$$

From Table 10.2 the corresponding coset leader (i.e., error pattern with the highest probability of occurrence) is found to be [0010000], indicating correctly that the third bit of the received vector is erroneous. Thus, adding this error pattern to the received vector, in accordance with (10.26), yields the correct code vector actually sent.

Table 10.2 Decoding table for the (7,4) Hamming code defined in Table 10.1

Syndrome	Error pattern
000	0000000
100	1000000
010	0100000
001	0010000
110	0001000
011	0000100
111	0000010
101	0000001

10.5 Cyclic Codes

Cyclic codes form a subclass of linear block codes. Indeed, many of the important linear block codes discovered to date are either cyclic codes or closely related to cyclic codes. An advantage of cyclic codes over most other types of codes is that they are easy to encode. Furthermore, cyclic codes possess a well-defined mathematical structure, which has led to the development of very efficient decoding schemes for them.

A binary code is said to be a *cyclic code* if it exhibits two fundamental properties:

PROPERTY 1 Linearity Property

The sum of any two codewords in the code is also a codeword.

PROPERTY 2 Cyclic Property

Any cyclic shift of a codeword in the code is also a codeword.

Property 1 restates the fact that a cyclic code is a linear block code (i.e., it can be described as a parity-check code). To restate Property 2 in mathematical terms, let the n-tuple $c_0, c_1, \ldots, c_{n-1}$ denote a codeword of an (n, k) linear block code. The code is a cyclic code if the n-tuples

$$\begin{aligned}&(c_{n-1}, c_0, \ldots, c_{n-2})\\ &(c_{n-2}, c_{n-1}, \ldots, c_{n-3})\\ &\vdots\\ &(c_1, c_2, \ldots, c_{n-1}, c_0)\end{aligned}$$

are all codewords in the code.

To develop the algebraic properties of cyclic codes, we use the elements $c_0, c_1, \ldots, c_{n-1}$ of a codeword to define the code polynomial

$$\mathbf{c}(X) = c_0 + c_1 X + c_2 X^2 + \cdots + c_{n-1} X^{n-1} \tag{10.27}$$

where X is an indeterminate. Naturally, for binary codes, the coefficients are 1s and 0s. Each power of X in the polynomial $\mathbf{c}(X)$ represents a one-bit *shift* in time. Hence, multiplication of the polynomial $\mathbf{c}(X)$ by X may be viewed as a shift to the right. The key question is: How do we make such a shift *cyclic*? The answer to this question is addressed next.

Let the code polynomial $\mathbf{c}(X)$ in (10.27) be multiplied by X^i, yielding

$$X^i \mathbf{c}(X) = c_0 X^i + c_1 X^{i+1} + \cdots + c_{n-i-1} X^{n-1} + \cdots + c_{n-1} X^{n+i-1}$$

Recognizing, for example, that $c_{n-i} + c_{n-i} = 0$ in modulo-2 addition, we may manipulate the preceding equation into the following compact form:

$$X^i \mathbf{c}(X) = \mathbf{q}(X)(X^n + 1) + \mathbf{c}^{(i)}(X) \tag{10.28}$$

where the polynomial $\mathbf{q}(X)$ is defined by

$$\mathbf{q}(X) = c_{n-i} + c_{n-i+1} X + \cdots + c_{n-1} X^{i-1} \tag{10.29}$$

As for the polynomial $\mathbf{c}^{(i)}X$ in (10.28), it is recognized as the code polynomial of the codeword $(c_{n-i}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-i-1})$ obtained by applying i cyclic shifts to the codeword $c_0, c_1, \ldots, c_{n-i-1}, c_{n-i}, \ldots, c_{n-1}$. Moreover, from (10.28) we readily see that $\mathbf{c}^{(i)}(X)$ is the remainder that results from dividing $X^i\mathbf{c}(X)$ by $(X^n + 1)$. We may thus formally state the cyclic property in polynomial notation as follows:

> If $\mathbf{c}(X)$ is a code polynomial, then the polynomial
>
> $$\mathbf{c}^{(i)}(X) = X^i\mathbf{c}(X) \mod (X^n + 1) \tag{10.30}$$
>
> is also a code polynomial for any cyclic shift i; the term mod is the abbreviation for modulo.

The special form of polynomial multiplication described in (10.30) is referred to as *multiplication modulo* $X^n + 1$. In effect, the multiplication is subject to the constraint $X^n = 1$, the application of which restores the polynomial $X^i\mathbf{c}(X)$ to order $n - 1$ for all $i < n$. Note that, in modulo-2 arithmetic, $X^n + 1$ has the same value as $X^n - 1$.

Generator Polynomial

The polynomial $X^n + 1$ and its factors play a major role in the generation of cyclic codes. Let $\mathbf{g}(X)$ be a polynomial of degree $n - k$ that is a factor of $X^n + 1$; as such, $\mathbf{g}(X)$ is the polynomial of least degree in the code. In general, $\mathbf{g}(X)$ may be expanded as follows:

$$\mathbf{g}(X) = 1 + \sum_{i=1}^{n-k-1} g_i X^i + X^{n-k} \tag{10.31}$$

where the coefficient g_i is equal to 0 or 1 for $i = 1, \ldots, n-k-1$. According to this expansion, the polynomial $\mathbf{g}(X)$ has two terms with coefficient 1 separated by $n - k - 1$ terms. The polynomial $\mathbf{g}(X)$ is called the *generator polynomial* of a cyclic code. A cyclic code is uniquely determined by the generator polynomial $\mathbf{g}(X)$ in that each code polynomial in the code can be expressed in the form of a polynomial product as follows:

$$\mathbf{c}(X) = \mathbf{a}(X)\mathbf{g}(X) \tag{10.32}$$

where $\mathbf{a}(X)$ is a polynomial in X with degree $k - 1$. The $\mathbf{c}(X)$ so formed satisfies the condition of (10.30) since $\mathbf{g}(X)$ is a factor of $X^n + 1$.

Suppose we are given the generator polynomial $\mathbf{g}(X)$ and the requirement is to encode the message sequence $(m_0, m_1, \ldots, m_{k-1})$ into an (n, k) *systematic* cyclic code. That is, the message bits are transmitted in unaltered form, as shown by the following structure for a codeword (see Figure 10.4):

$$(\underbrace{b_0, b_1, \ldots, b_{n-k-1}}_{n-k \text{ parity-check bits}}, \underbrace{m_0, m_1, \ldots, m_{k-1}}_{k \text{ message bits}})$$

Let the *message polynomial* be defined by

$$\mathbf{m}(X) = m_0 + m_1 X + \cdots + m_{k-1}X^{k-1} \tag{10.33}$$

and let

$$\mathbf{b}(X) = b_0 + b_1 X + \cdots + b_{n-k-1}X^{n-k-1} \tag{10.34}$$

Then, according to (10.1), we want the code polynomial to be in the form

$$\mathbf{c}(X) = \mathbf{b}(X) + X^{n-k}\mathbf{m}(X) \tag{10.35}$$

To this end, the use of (10.32) and (10.35) yields

$$\mathbf{a}(X)\mathbf{g}(X) = \mathbf{b}(X) + X^{n-k}\mathbf{m}(X)$$

Equivalently, invoking modulo-2 addition, we may also write

$$\frac{X^{n-k}\mathbf{m}(X)}{\mathbf{g}(X)} = \mathbf{a}(X) + \frac{\mathbf{b}(X)}{\mathbf{g}(x)} \tag{10.36}$$

Equation (10.36) states that the polynomial $\mathbf{b}(X)$ is the remainder left over after dividing $X^{n-k}\mathbf{m}(X)$ by $\mathbf{g}(X)$.

We may now summarize the steps involved in the encoding procedure for an (n,k) cyclic code, assured of a systematic structure. Specifically, we proceed as follows:

Step 1: Premultiply the message polynomial $\mathbf{m}(X)$ by X^{n-k}.

Step 2: Divide $X^{n-k}\mathbf{m}(X)$ by the generator polynomial $\mathbf{g}(X)$, obtaining the remainder $\mathbf{b}(X)$.

Step 3: Add $\mathbf{b}(X)$ to $X^{n-k}\mathbf{m}(X)$, obtaining the code polynomial $\mathbf{c}(X)$.

Parity-Check Polynomial

An (n,k) cyclic code is uniquely specified by its generator polynomial $\mathbf{g}(X)$ of order $(n-k)$. Such a code is also uniquely specified by another polynomial of degree k, which is called the *parity-check polynomial*, defined by

$$\mathbf{h}(X) = 1 + \sum_{i=1}^{k-1} h_i X^i + X^k \tag{10.37}$$

where the coefficients h_i are 0 or 1. The parity-check polynomial $\mathbf{h}(X)$ has a form similar to the generator polynomial, in that there are two terms with coefficient 1, but separated by $k-1$ terms.

The generator polynomial $\mathbf{g}(X)$ is equivalent to the generator matrix $\mathbf{G}$ as a description of the code. Correspondingly, the parity-check polynomial $\mathbf{h}(X)$ is an equivalent representation of the parity-check matrix $\mathbf{H}$. We thus find that the matrix relation $\mathbf{H}\mathbf{G}^{\mathrm{T}} = \mathbf{0}$ presented in (10.15) for linear block codes corresponds to the relationship

$$\mathbf{g}(X)\mathbf{h}(X) \mod(X^n + 1) = \mathbf{0} \tag{10.38}$$

Accordingly, we may make the statement:

> The generator polynomial $\mathbf{g}(X)$ and the parity-check polynomial $\mathbf{h}(X)$ are factors of the polynomial $X^n + 1$, as shown by
>
> $$\mathbf{g}(X)\mathbf{h}(X) = X^n + 1 \tag{10.39}$$

This statement provides the basis for selecting the generator or parity-check polynomial of a cyclic code. In particular, if $\mathbf{g}(X)$ is a polynomial of degree $(n-k)$ and it is also a factor of $X^n + 1$, then $\mathbf{g}(X)$ is the generator polynomial of an (n,k) cyclic code. Equivalently, if

$\mathbf{h}(X)$ is a polynomial of degree k and it is also a factor of $X^n + 1$, then $\mathbf{h}(X)$ is the parity-check polynomial of an (n,k) cyclic code.

A final comment is in order. Any factor of $X^n + 1$ with degree $(n - k)$ can be used as a generator polynomial. The fact of the matter is that, for large values of n, the polynomial $X^n + 1$ may have many factors of degree $n - k$. Some of these polynomial factors generate good cyclic codes, whereas some of them generate bad cyclic codes. The issue of how to select generator polynomials that produce good cyclic codes is very difficult to resolve. Indeed, coding theorists have expended much effort in the search for good cyclic codes.

Generator and Parity-Check Matrices

Given the generator polynomial $\mathbf{g}(X)$ of an (n,k) cyclic code, we may construct the generator matrix $\mathbf{G}$ of the code by noting that the k polynomials $\mathbf{g}(X)$, $X\mathbf{g}(X)$, …, $X^{k-1}\mathbf{g}(X)$ span the code. Hence, the n-tuples corresponding to these polynomials may be used as rows of the k-by-n generator matrix $\mathbf{G}$.

However, the construction of the parity-check matrix $\mathbf{H}$ of the cyclic code from the parity-check polynomial $\mathbf{h}(X)$ requires special attention, as described here. Multiplying (10.39) by $\mathbf{a}$(x) and then using (10.32), we obtain

$$\mathbf{c}(X)\mathbf{h}(X) = \mathbf{a}(X) + X^n\mathbf{a}(X) \tag{10.40}$$

The polynomials $\mathbf{c}(X)$ and $\mathbf{h}(X)$ are themselves defined by (10.27) and (10.37) respectively, which means that their product on the left-hand side of (10.40) contains terms with powers extending up to $n + k - 1$. On the other hand, the polynomial $\mathbf{a}(X)$ has degree $k - 1$ or less, the implication of which is that the powers of $X^k, X^{k+1}, \ldots, X^{n-1}$ do *not* appear in the polynomial on the right-hand side of (10.40). Thus, setting the coefficients of $X^k, X^{k-1}, \ldots, X^{n-1}$ in the expansion of the product polynomial $\mathbf{c}(X)\mathbf{h}(X)$ equal to zero, we obtain the following set of $n - k$ equations:

$$\sum_{i=j}^{j+k} c_i h_{k+j-i} = 0 \qquad \text{for } 0 \le j \le n-k-1 \tag{10.41}$$

Comparing (10.41) with the corresponding relation (10.16), we may make the following important observation:

> The coefficients of the parity-check polynomial $\mathbf{h}(X)$ involved in the polynomial multiplication described in (10.41) are arranged in reversed order with respect to the coefficients of the parity-check matrix $\mathbf{H}$ involved in forming the inner product of vectors described in (10.16).

This observation suggests that we define the *reciprocal of the parity-check polynomial* as follows:

$$\begin{aligned} X^k\mathbf{h}(X^{-1}) &= X^k\left(1 + \sum_{i=1}^{k-1} h_i X^{-i} + X^{-k}\right) \\ &= 1 + \sum_{i=1}^{k-1} h_{k-1} X^i + X^k \end{aligned} \tag{10.42}$$

which is also a factor of $X^n + 1$. The n-tuples pertaining to the $(n-k)$ polynomials $X^k h(X^{-1}), X^{k+1} h(X^{-1}), \ldots, X^{n-1} h(X^{-1})$ may now be used in rows of the $(n-k)$-by-n parity-check matrix $\mathbf{H}$.

In general, the generator matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$ constructed in the manner described here are not in their systematic forms. They can be put into their systematic forms by performing simple operations on their respective rows, as illustrated in Example 1.

Encoding of Cyclic Codes

Earlier we showed that the encoding procedure for an (n,k) cyclic code in systematic form involves three steps:

- multiplication of the message polynomial $\mathbf{m}(X)$ by X^{n-k},
- division of $X^{n-k}\mathbf{m}(X)$ by the generator polynomial $\mathbf{g}(X)$ to obtain the remainder $\mathbf{b}(X)$, and
- addition of $\mathbf{b}(X)$ to $X^{n-k}\mathbf{m}(X)$ to form the desired code polynomial.

These three steps can be implemented by means of the encoder shown in Figure 10.8, consisting of a *linear feedback shift register* with $(n-k)$ stages.

The boxes in Figure 10.8 represent *flip-flops*, or *unit-delay elements*. The flip-flop is a device that resides in one of two possible states denoted by 0 and 1. An *external clock* (not shown in Figure 10.8) controls the operation of all the flip-flops. Every time the clock ticks, the contents of the flip-flops (initially set to the state 0) are shifted out in the direction of the arrows. In addition to the flip-flops, the encoder of Figure 10.8 includes a second set of logic elements, namely *adders*, which compute the modulo-2 sums of their respective inputs. Finally, the *multipliers* multiply their respective inputs by the associated coefficients. In particular, if the coefficient $g_i = 1$, the multiplier is just a direct "connection." If, on the other hand, the coefficient $g_i = 0$, the multiplier is "no connection."

The operation of the encoder shown in Figure 10.8 proceeds as follows:

1. The gate is switched on. Hence, the k message bits are shifted into the channel. As soon as the k message bits have entered the shift register, the resulting $(n-k)$ bits in the register form the parity-check bits. (Recall that the parity-check bits are the same as the coefficients of the remainder $\mathbf{b}(X)$.)

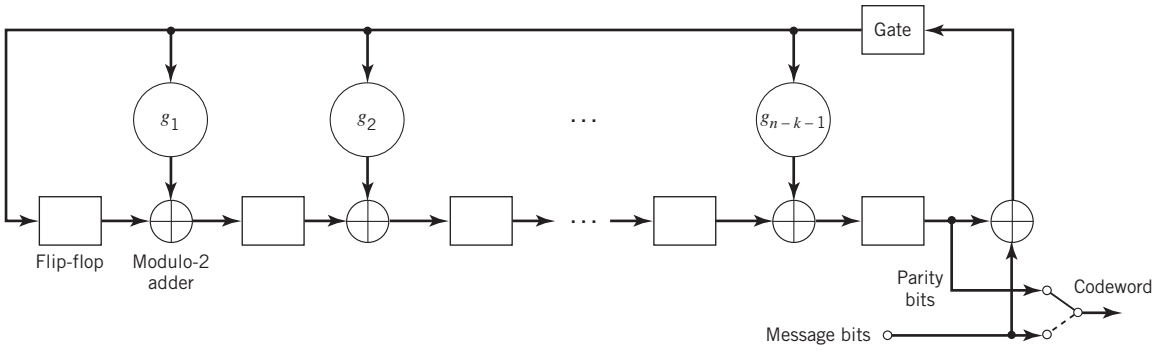


Figure 10.8 Encoder for an (n,k) cyclic code.

2. The gate is switched off, thereby breaking the feedback connections.
3. The contents of the shift register are read out into the channel.

Calculation of the Syndrome

Suppose the codeword $(c_0, c_1, \ldots, c_{n-1})$ is transmitted over a noisy channel, resulting in the received word $r_0, r_1, \ldots, r_{n-1}$. From Section 10.3, we recall that the first step in the decoding of a linear block code is to calculate the syndrome for the received word. If the syndrome is zero, there are no transmission errors in the received word. If, on the other hand, the syndrome is nonzero, the received word contains transmission errors that require correction.

In the case of a cyclic code in systematic form, the syndrome can be calculated easily. Let the received vector be represented by a polynomial of degree $n-1$ or less, as shown by

$$\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{n-1} X^{n-1}$$

Let $\mathbf{q}(X)$ denote the quotient and $\mathbf{s}(X)$ denote the remainder, which are the results of dividing $\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$. We may therefore express $\mathbf{r}(X)$ as follows:

$$\mathbf{r}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{s}(X) \tag{10.43}$$

The remainder $\mathbf{s}(X)$ is a polynomial of degree $n-k-1$ or less, which is the result of interest. It is called the *syndrome polynomial* because its coefficients make up the $(n-k)$-by-1 syndrome $\mathbf{s}$.

Figure 10.9 shows a *syndrome calculator* that is identical to the encoder of Figure 10.8 except for the fact that the received bits are fed into the $n-k$ stages of the feedback shift register from the left. As soon as all the received bits have been shifted into the shift register, its contents define the syndrome $\mathbf{s}$.

The syndrome polynomial $\mathbf{s}(X)$ has the following useful properties that follow from the definition given in (10.43).

PROPERTY 1 *The syndrome of a received word polynomial is also the syndrome of the corresponding error polynomial.*

Given that a cyclic code with polynomial $\mathbf{c}(X)$ is sent over a noisy channel, the received word polynomial is defined by

$$\mathbf{r}(X) = \mathbf{c}(X) + \mathbf{e}(X)$$

where $\mathbf{e}(X)$ is the error polynomial. Equivalently, we may write

$$\mathbf{e}(X) = \mathbf{r}(X) + \mathbf{c}(X)$$

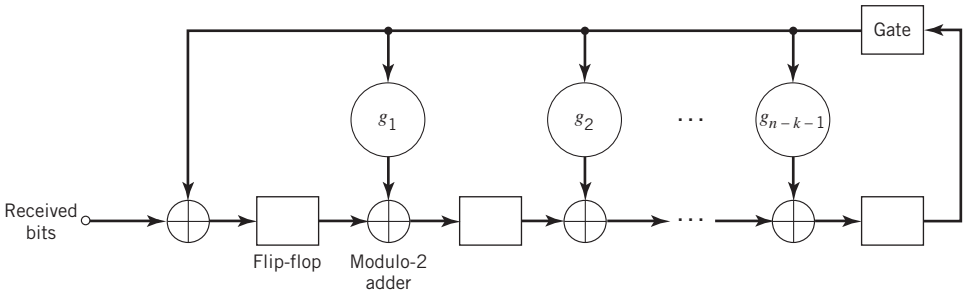


Figure 10.9 Syndrome computer for (n, k) cyclic code.

Hence, substituting (10.32) and (10.43) into the preceding equation, we get

$$\mathbf{e}(X) = \mathbf{u}(X)\mathbf{g}(X) + \mathbf{s}(X) \tag{10.44}$$

where the quotient is $\mathbf{u}(X) = \mathbf{a}(X) + \mathbf{q}(X)$. Equation (10.44) shows that $\mathbf{s}(X)$ is also the syndrome of the error polynomial $\mathbf{e}(X)$. The implication of this property is that when the syndrome polynomial $\mathbf{s}(X)$ is nonzero, the presence of transmission errors in the received vector is detected.

PROPERTY 2 *Let $\mathbf{s}(X)$ be the syndrome of a received word polynomial $\mathbf{r}(X)$. Then, the syndrome of $X\mathbf{r}(X)$, representing a cyclic shift of $\mathbf{r}(X)$, is $X\mathbf{s}(X)$.*

Applying a cyclic shift to both sides of (10.43), we get

$$X\mathbf{r}(X) = X\mathbf{q}(X)\mathbf{g}(X) + X\mathbf{s}(X) \tag{10.45}$$

from which we readily see that $X\mathbf{s}(X)$ is the remainder of the division of $X\mathbf{r}(X)$ by $\mathbf{g}(X)$. Hence, the syndrome of $X\mathbf{r}(X)$ is $X\mathbf{s}(X)$ as stated. We may generalize this result by stating that if $\mathbf{s}(X)$ is the syndrome of $\mathbf{r}(X)$, then $X^i\mathbf{s}(X)$ is the syndrome of $X^i\mathbf{r}(X)$.

PROPERTY 3 *The syndrome polynomial $\mathbf{s}(X)$ is identical to the error polynomial $\mathbf{e}(X)$, assuming that the errors are confined to the $(n-k)$ parity-check bits of the received word polynomial $\mathbf{r}(X)$.*

The assumption made here is another way of saying that the degree of the error polynomial $\mathbf{e}(X)$ is less than or equal to $(n-k-1)$. Since the generator polynomial $\mathbf{g}(X)$ is of degree $(n-k)$, by definition, it follows that (10.44) can only be satisfied if the quotient $\mathbf{u}(X)$ is zero. In other words, the error polynomial $\mathbf{e}(X)$ and the syndrome polynomial $\mathbf{s}(X)$ are one and the same. The implication of Property 3 is that, under the aforementioned conditions, error correction can be accomplished simply by adding the syndrome polynomial $\mathbf{s}(X)$ to the received vector $\mathbf{r}(X)$.

EXAMPLE 2 **Hamming Codes Revisited**

To illustrate the issues relating to the polynomial representation of cyclic codes, we consider the generation of a (7,4) cyclic code. With the block length $n = 7$, we start by factorizing $X^7 + 1$ into three *irreducible polynomials*:

$$X^7 + 1 = (1 + X)(1 + X^2 + X^3)(1 + X + X^3)$$

By an "irreducible polynomial" we mean a polynomial that cannot be factored using only polynomials with coefficients from the binary field. An irreducible polynomial of degree m is said to be *primitive* if the smallest positive integer n for which the polynomial divides $X^n + 1$ is $n = 2^m - 1$. For the example at hand, the two polynomials $(1 + X^2 + X^3)$ and $(1 + X + X^3)$ are primitive. Let us take

$$\mathbf{g}(X) = 1 + X + X^3$$

as the generator polynomial, whose degree equals the number of parity-check bits. This means that the parity-check polynomial is given by

$$\begin{aligned}\mathbf{h}(X) &= (1 + X)(1 + X^2 + X^3)\\ &= 1 + X + X^2 + X^4\end{aligned}$$

whose degree equals the number of message bits $k = 4$.

Next, we illustrate the procedure for the construction of a codeword by using this generator polynomial to encode the message sequence 1001. The corresponding message vector is given by

$$\mathbf{m}(X) = 1 + X^3$$

Hence, multiplying $\mathbf{m}(X)$ by $X^{n-k} = X^3$, we get

$$X^{n-k}\mathbf{m}(X) = X^3 + X^6$$

The second step is to divide $X^{n-k}\mathbf{m}(X)$ by $\mathbf{g}(X)$, the details of which (for the example at hand) are given below:

$$\begin{array}{r|l} X^3 + X + 1 & X^3 + X \\ \hline X^6 + X^3 & \\ X^6 + X^4 + X^3 & \\ \hline X^4 & \\ X^4 + X^2 + X & \\ \hline X^2 + X & \end{array}$$

Note that in this long division we have treated subtraction the same as addition since we are operating in modulo-2 arithmetic. We may thus write

$$\frac{X^3 + X^6}{1 + X + X^3} = X + X^3 + \frac{X + X^2}{1 + X + X^3}$$

That is, the quotient $\mathbf{a}(X)$ and remainder $\mathbf{b}(X)$ are as follows, respectively:

$$\mathbf{a}(X) = X + X^3$$

$$\mathbf{b}(X) = X + X^2$$

Hence, from (10.35) we find that the desired code vector is

$$\begin{aligned} \mathbf{c}(X) &= \mathbf{b}(X) + X^{n-k}\mathbf{m}(X) \\ &= X + X^2 + X^3 + X^6 \end{aligned}$$

The codeword is therefore 0111001. The four rightmost bits, 1001, are the specified message bits. The three leftmost bits, 011, are the parity-check bits. The codeword thus generated is exactly the same as the corresponding one shown in Table 10.1 for a (7,4) Hamming code.

We may generalize this result by stating that:

> Any cyclic code generated by a primitive polynomial is a Hamming code of minimum distance 3.

We next show that the generator polynomial $\mathbf{g}(X)$ and the parity-check polynomial $\mathbf{h}(X)$ uniquely specify the generator matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$, respectively.

To construct the 4-by-7 generator matrix $\mathbf{G}$, we start with four vectors represented by $\mathbf{g}(X)$ and three cyclic-shifted versions of it, as shown by

$$\begin{aligned}\mathbf{g}(X) &= 1 + X + X^3\\ X\mathbf{g}X &= X + X^2 + X^4\\ X^2\mathbf{g}(X) &= X^2 + X^3 + X^5\\ X^3\mathbf{g}(X) &= X^3 + X^4 + X^6\end{aligned}$$

The vectors $\mathbf{g}(X)$, $X\mathbf{g}(X)$, $X^2\mathbf{g}(X)$, and $X^3\mathbf{g}(X)$ represent code polynomials in the (7,4) Hamming code. If the coefficients of these polynomials are used as the elements of the rows of a 4-by-7 matrix, we get the following generator matrix:

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

Clearly, the generator matrix $\mathbf{G}'$ so constructed is not in systematic form. We can put it into a systematic form by adding the first row to the third row, and adding the sum of the first two rows to the fourth row. These manipulations result in the desired generator matrix:

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

which is exactly the same as that in Example 1.

We next show how to construct the 3-by-7 parity-check matrix $\mathbf{H}$ from the parity-check polynomial $\mathbf{h}(X)$. To do this, we first take the reciprocal of $\mathbf{h}(X)$, namely $X^4\mathbf{h}(X^{-1})$. For the problem at hand, we form three vectors represented by $X^4\mathbf{h}(X^{-1})$ and two shifted versions of it, as shown by

$$\begin{aligned}X^4\mathbf{h}(X^{-1}) &= 1 + X^2 + X^3 + X^4\\ X^5\mathbf{h}(X^{-1}) &= X + X^3 + X^4 + X^5\\ X^6\mathbf{h}(X^{-1}) &= X^2 + X^4 + X^5 + X^6\end{aligned}$$

Using the coefficients of these three vectors as the elements of the rows of the 3-by-7 parity-check matrix, we get

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

Here again we see that the matrix $\mathbf{H}'$ is not in systematic form. To put it into a systematic form, we add the third row to the first row to obtain

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

which is exactly the same as that of Example 1.

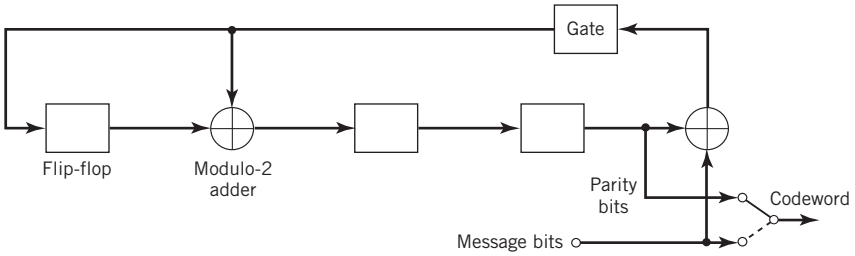


Figure 10.10 Encoder for the (7,4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

Figure 10.10 shows the encoder for the (7,4) cyclic Hamming code generated by the polynomial $\mathbf{g}(X) = 1 + X + X^3$. To illustrate the operation of this encoder, consider the message sequence (1001). The contents of the shift register are modified by the incoming message bits as in Table 10.3. After four shifts, the contents of the shift register, and therefore the parity-check bits, are (011). Accordingly, appending these parity-check bits to the message bits (1001), we get the codeword (0111001); this result is exactly the same as that determined earlier in Example 1.

Table 10.3 Contents of the shift register in the encoder of Figure 10.10 for message sequence (1001)

Shift	Input bit	Contents of shift register
		000 (initial state)
1	1	110
2	0	011
3	0	111
4	1	011

Figure 10.11 shows the corresponding syndrome calculator for the (7,4) Hamming code. Let the transmitted codeword be (0111001) and the received word be (0110001); that is, the middle bit is in error. As the received bits are fed into the shift register, initially set to zero, its contents are modified as in Table 10.4. At the end of the seventh shift, the syndrome is identified from the contents of the shift register as 110. Since the syndrome is nonzero, the received word is in error. Moreover, from Table 10.2, we see that the error pattern corresponding to this syndrome is 0001000. This indicates that the error is in the middle bit of the received words, which is indeed the case.

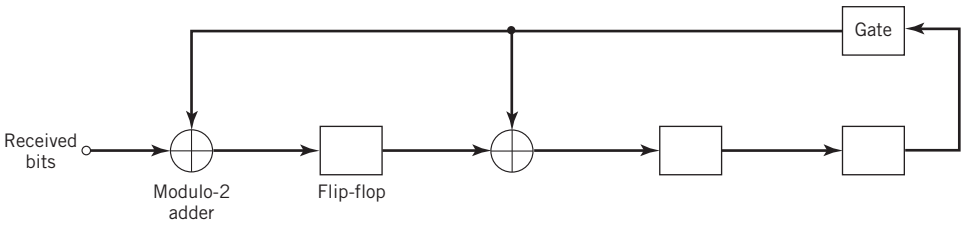


Figure 10.11 Syndrome calculator for the (7,4) cyclic code generated by the polynomial $\mathbf{g}(X) = 1 + X + X^3$.

Table 10.4 Contents of the syndrome calculator in Figure 10.11 for the received word (0110001)

Shift	Input bit	Contents of shift register
		000 (initial state)
1	1	100
2	0	010
3	0	001
4	0	110
5	1	111
6	1	001
7	0	110

EXAMPLE 3 **Maximal-Length Codes**

For any positive integer $m \geq 3$, there exists a *maximal-length code*[4] with the following parameters:

block length: $n = 2^m - 1$
number of message bits: $k = m$
minimum distance: $d_{\min} = 2^{m-1}$

Maximal-length codes are generated by vectors of the form

$$\mathbf{g}(X) = \frac{1 + X^n}{\mathbf{h}(X)} \tag{10.46}$$

where $\mathbf{h}(X)$ is any primitive polynomial of degree m. Earlier we stated that any cyclic code generated by a primitive polynomial is a Hamming code of minimum distance 3 (see Example 2). It follows, therefore, that maximal-length codes are the *dual* of Hamming codes.

The polynomial $\mathbf{h}(X)$ defines the feedback connections of the encoder. The generator polynomial $\mathbf{g}(X)$ defines one period of the maximal-length code, assuming that the encoder is in the initial state 00 … 01. To illustrate this, consider the example of a (7,3) maximal-length code, which is the dual of the (7,4) Hamming code described in Example 2. Thus, choosing

$$\mathbf{h}(X) = 1 + X + X^3$$

we find that the generator polynomial of the (7,3) maximal-length code is

$$\mathbf{g}(X) = 1 + X + X^2 + X^4$$

Figure 10.12 shows the encoder for the (7,3) maximal-length code. The period of the code is $n = 7$. Thus, assuming that the encoder is in the initial state 001, as indicated in Figure 10.12, we find the output sequence is described by

$$\underbrace{1\ 0\ 0}_{\text{initial state}} \qquad \underbrace{1\ 1\ 1\ 0\ 1\ 0\ 0}_{\mathbf{g}(X) = 1 + X + X^2 + X^4}$$

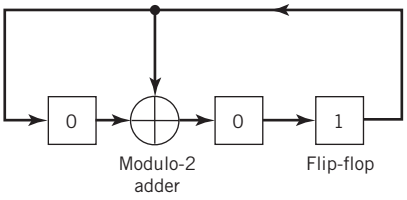


Figure 10.12 Encoder for the (7,3) maximal-length code; the initial state of the encoder is shown in the figure.

This result is readily validated by cycling through the encoder of Figure 10.12.

Note that if we were to choose the other primitive polynomial

$$\mathbf{h}(X) = 1 + X^2 + X^3$$

for the (7,3) maximal-length code, we would simply get the "image" of the code described above, and the output sequence would be "reversed" in time.

Reed–Solomon Codes

A study of cyclic codes for error control would be incomplete without a discussion of *Reed–Solomon codes,*[5] albeit briefly.

Unlike the cyclic codes considered in this section, Reed–Solomon codes are *nonbinary codes*. A cyclic code is said to be nonbinary in that given the code vector

$$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$$

the coefficients $\{c_i\}_{i=0}^{n-1}$ are not binary 0 or 1. Rather, the c_i are themselves made up of sequences of 0s and 1s, with each sequence being of length k. A Reed–Solomon code is therefore said to be a q-ary code, which means that the size of the alphabet used in construction of the code is $q = 2^k$. To be specific, a Reed–Solomon (n,k) code is used to encode m-bit symbols into blocks consisting of $n = 2^m - 1$ symbols; that is, $m(2^m - 1)$ bits, where $m \geq 1$. Thus, the encoding algorithm expands a block of k symbols to n symbols by adding $n - k$ redundant symbols. When m is an integer power of 2, the m-bit symbols are called *bytes*. A popular value of m is 8; indeed, 8-bit Reed–Solomon codes are extremely powerful.

A t-error-correcting Reed–Solomon code has the following parameters:

block length	$n = 2^m - 1$ symbols
message size	k symbols
parity-check size	$n - k = 2t$ symbols
minimum distance	$d_{\min} = 2t + 1$ symbols

The block length of the Reed–Solomon code is one less than the size of a code symbol, and the minimum distance is one greater than the number of parity-check symbols. Reed–Solomon codes make highly efficient use of redundancy; block lengths and symbol sizes

can be adjusted readily to accommodate a wide range of message sizes. Moreover, Reed–Solomon codes provide a wide range of code rates that can be chosen to optimize performance, and efficient techniques are available for their use in certain practical applications. In particular, a distinctive feature of Reed–Solomon codes is their ability to correct *bursts of errors*, hence their application in wireless communications to combat the fading phenomenon.

10.6 Convolutional Codes

In block coding, the encoder accepts a k-bit message block and generates an n-bit codeword, which contains $n - k$ parity-check bits. Thus, codewords are produced on a block-by-block basis. Clearly, provision must be made in the encoder to buffer an entire message block before generating the associated codeword. There are applications, however, where the message bits come in *serially* rather than in large blocks, in which case the use of a buffer may be undesirable. In such situations, the use of *convolutional coding* may be the preferred method. A convolutional coder generates redundant bits by using *modulo-2 convolutions*; hence the name *convolutional codes*[6].

The encoder of a binary convolutional code with rate $1/n$, measured in bits per symbol, may be viewed as a *finite-state machine* that consists of an M-stage shift register with prescribed connections to n modulo-2 adders and a multiplexer that serializes the outputs of the adders. A sequence of message bits produces a coded output sequence of length $n(L + M)$ bits, where L is the length of the message sequence. The *code rate* is therefore given by

$$\begin{aligned} r &= \frac{L}{n(L+M)} \\ &= \frac{1}{n(1+M/L)} \quad \text{bits/symbol} \end{aligned} \tag{10.47}$$

Typically, we have $L \gg M$, in which case the code rate is approximately defined by

$$r \approx \frac{1}{n} \quad \text{bits/symbol} \tag{10.48}$$

An important characteristic of a convolutional code is its constraint length, which we define as follows:

> The constraint length of a convolutional code, expressed in terms of message bits, is the number of shifts over which a single incoming message bit can influence the encoder output.

In an encoder with an M-stage shift register, the *memory* of the encoder equals M message bits. Correspondingly, the constraint length, denoted by ν, equals $M + 1$ shifts that are required for a message bit to enter the shift register and finally come out.

Figure 10.13 shows a convolutional encoder with the number of message bits $n = 2$ and constraint length $\nu = 3$. In this example, the code rate of the encoder is 1/2. The encoder operates on the incoming message sequence, one bit at a time, through a convolution process; it is therefore said to be a *nonsystematic* code.

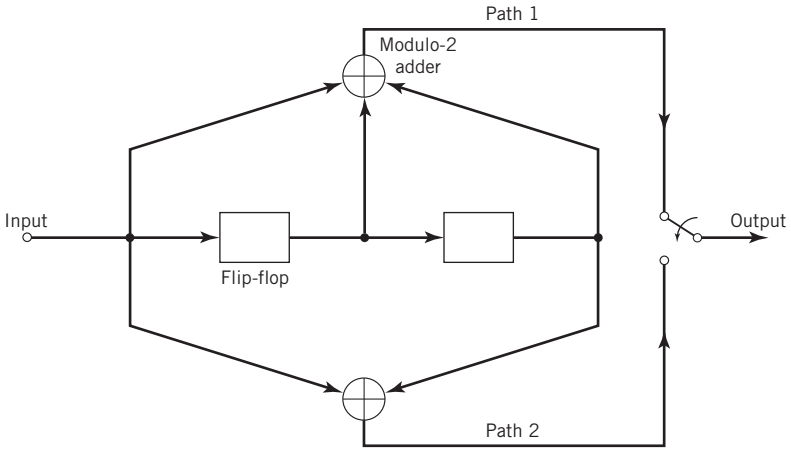


Figure 10.13
Constraint length-3, rate -1/2 convolutional encoder.

Each path connecting the output to the input of a convolutional encoder may be characterized in terms of its *impulse response*, defined as follows:

> The impulse response of a particular path in the convolutional encoder is the response of that path in the encoder to symbol 1 applied to its input, with each flip-flop in the encoder set initially to the zero state.

Equivalently, we may characterize each path in terms of a generator polynomial, defined as the *unit-delay transform* of the impulse response. To be specific, let the *generator sequence* $\left(g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)}\right)$ denote the impulse response of the ith path, where the coefficients $g_0^{(i)}, g_1^{(i)}, g_2^{(i)}, \ldots, g_M^{(i)}$ equal symbol 0 or 1. Correspondingly, the *generator polynomial* of the ith path is defined by

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)}D + g_2^{(i)}D^2 + \cdots + g_M^{(i)}D^M \tag{10.49}$$

where D denotes the *unit-delay variable*. The complete convolutional encoder is described by the set of generator polynomials $\{g^{(i)}(D)\}_{i=1}^{M}$.

EXAMPLE 4

Convolutional Encoder

Consider again the convolutional encoder of Figure 10.13, which has two paths numbered 1 and 2 for convenience of reference. The impulse response of path 1 (i.e., upper path) is (1, 1, 1). Hence, the generator polynomial of this path is

$$g^{(1)}(D) = 1 + D + D^2$$

The impulse response of path 2 (i.e., lower path) is (1, 0, 1). The generator polynomial of this second path is

$$g^{(2)}(D) = 1 + D^2$$

For an incoming message sequence given by (10011), for example, we have the polynomial representation

$$m(D) = 1 + D^3 + D^4$$

As with Fourier transformation, convolution in the time domain is transformed into multiplication in the D-domain. Hence, the output polynomial of path 1 is given by

$$\begin{aligned} c^{(1)}(D) &= g^{(1)}(D)m(D) \\ &= (1 + D + D^2)(1 + D^3 + D^4) \\ &= 1 + D + D^2 + D^3 + D^6 \end{aligned}$$

where it is noted that the sums $D^4 + D^4$ and $D^5 + D^5$ are both zero in accordance with the rules of binary arithmetic. We therefore immediately deduce that the output sequence of path 1 is (1111001). Similarly, the output polynomial of path 2 is given by

$$\begin{aligned} c^{(2)}(D) &= g^{(2)}(D)m(D) \\ &= (1 + D^2)(1 + D^3 + D^4) \\ &= 1 + D^2 + D^3 + D^4 + D^5 + D^6 \end{aligned}$$

The output sequence of path 2 is therefore (1011111). Finally, *multiplexing* the two output sequences of paths 1 and 2, we get the encoded sequence

$$\mathbf{c} = (11, 10, 11, 11, 01, 01, 11)$$

Note that the message sequence of length $L = 5$ bits produces an encoded sequence of length $n(L + \nu - 1) = 14$ bits. Note also that for the shift register to be restored to its initial all-zero state, a terminating sequence of $\nu - 1 = 2$ zeros is appended to the last input bit of the message sequence. The terminating sequence of $\nu - 1$ zeros is called the *tail of the message*.

Code Tree, Trellis Graph, and State Graph

Traditionally, the structural properties of a convolutional encoder are portrayed in graphical form by using any one of three equivalent graphs: code tree, trellis graph, and state graph.

Although, indeed, these three graphical representations of a convolutional encoder look different, their compositions follow the same underlying rule:

> A code branch produced by input bit 0 is drawn as a solid line, whereas a code branch produced by input bit 1 is a dashed line.

Hereafter, we refer to this convention as the *graphical rule* of a convolutional encoder.

We will use the convolutional encoder of Figure 10.13 as a running example to illustrate the insights that each one of these three diagrams provides.

Code Tree

We begin the graphical representation of a convolutional encoder with the *code tree* of Figure 10.14. Each branch of the tree represents an input bit, with the corresponding pair of output bits indicated on the branch. The convention used to distinguish the input bits 0 and 1 follows the graphical rule described above. Thus, a specific *path* in the tree is traced from left to right in accordance with the message sequence. The corresponding coded bits

on the branches of that path constitute the message sequence (10011) applied to the input of the encoder of Figure 10.13. Following the procedure just described, we find that the corresponding encoded sequence is (11, 10, 11, 11, 01), which agrees with the first five pairs of bits in the encoded sequence $\{c_i\}$ that was derived in Example 4.

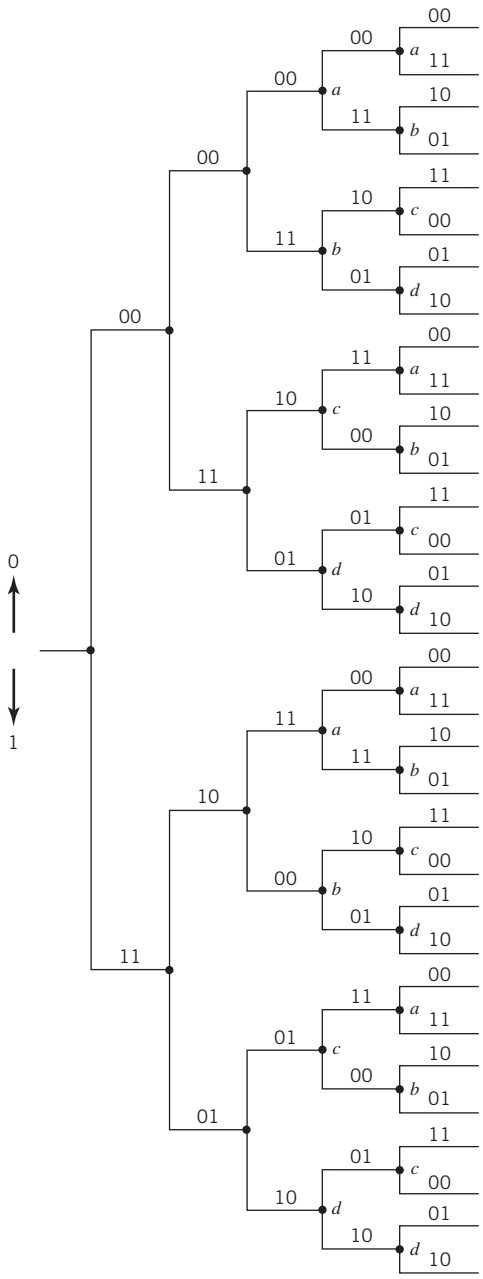


Figure 10.14 Code tree for the convolutional encoder of Figure 10.13.

Trellis Graph

From Figure 10.14, we observe that the tree becomes *repetitive* after the first three branches. Indeed, beyond the third branch, the two nodes labeled *a* are identical and so are all the other node pairs that are identically labeled. We may establish this repetitive property of the tree by examining the associated encoder of Figure 10.13. The encoder has memory $M = \nu - 1 = 2$ message bits. We therefore find that, when the third message bit enters the encoder, the first message bit is shifted out of the register. Consequently, after the third branch, the message sequences (100 m_3m_4…) and (000 m_3m_4…) generate the same code symbols, and the pair of nodes labeled *a* may be joined together. The same reasoning applies to the other nodes in the code tree. Accordingly, we may collapse the code tree of Figure 10.14 into the new form shown in Figure 10.15, which is called a *trellis*. It is so called since a trellis is a treelike structure with re-emerging branches. The convention used in Figure 10.15 to distinguish between input symbols 0 and 1 is as follows:

> A code branch in a trellis produced by input binary symbol 0 is drawn as a solid line, whereas a code branch produced by an input 1 is drawn as a dashed line.

As before, each message sequence corresponds to a specific path through the trellis. For example, we readily see from Figure 10.15 that the message sequence (10011) produces the encoded output sequence (11, 10, 11, 11, 01), which agrees with our previous result.

The Notion of State

In conceptual terms, a trellis is more instructive than a tree. We say so because it brings out explicitly the fact that the associated convolutional encoder is in actual fact a *finite-state machine*. Basically, such a machine consists of a tapped shift register and, therefore, has a finite state; hence the name of the machine. Thus, we may conveniently say the following:

> The state of a rate $1/n$ convolutional encoder is determined by the smallest number of message bits stored in memory (i.e., the shift register).

For example, the convolutional encoder of Figure 10.13 has a shift register made up of two memory cells. With the message bit stored in each memory cell being 0 or 1, it follows that this encoder can assume any one of $2^2 = 4$ possible states, as described in Table 10.5.

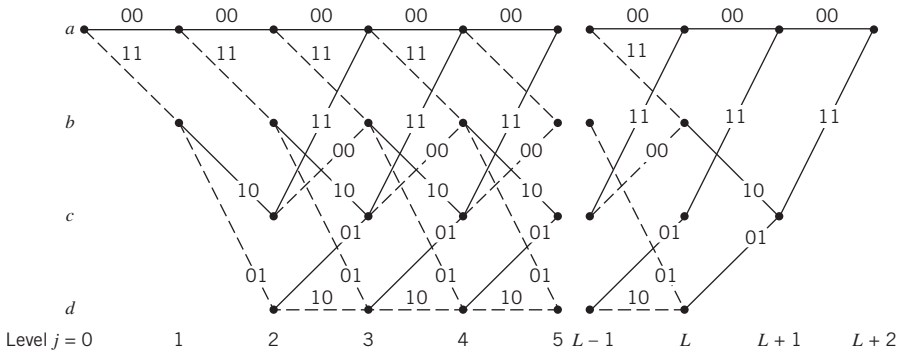


Figure 10.15 Trellis for the convolutional encoder of Figure 10.13.

Table 10.5 State table for the convolutional encoder of Figure 10.13

State	Binary description
a	00
b	10
c	01
d	11

In describing a convolutional encoder, the notion of state is important in the following sense:

> Given the current message bit and the state of the encoder, the codeword produced at the output of the encoder is completely determined.

To illustrate this statement, consider the general case of a rate $1/n$ convolutional encoder of constraint length ν. Let the state of the encoder at time-unit j be denoted by

$$S = (m_{j-1}, m_{j-2}, \ldots, m_{j-\nu+1})$$

The jth codeword c_j is completely determined by the state S together with the current message bit m_j.

Now that we understand the notion of state, the trellis graph of the simple convolutional encoder of Figure 10.13 for $\nu = 3$ is presented in Figure 10.15. From this latter figure, we now clearly see a unique characteristic of the trellis diagram:

> The trellis depicts the evolution of the convolutional encoder's state across time.

To be more specific, the first $\nu - 1 = 2$ time-steps correspond to the encoder's departure from the initial zero state and the last $\nu - 1 = 2$ time-steps correspond to the encoder's return to the initial zero state. Naturally, not all the states of the encoder can be reached in these two particular portions of the trellis. However, in the central portion of the trellis, for which time-unit j lies in the range $\nu - 1 \leq j \leq L$, where L is the length of the incoming message sequence, we do see that all the four possible states of the encoder are reachable. Note also that the central portion of the trellis exhibits a *fixed periodic structure*, as illustrated in Figure 10.16a.

State Graph

The periodic structure characterizing the trellis leads us next to the state diagram of a convolutional encoder. To be specific, consider a central portion of the trellis corresponding to times j and $j + 1$. We assume that for $j \geq 2$ in the example of Figure 10.13, it is possible for the current state of the encoder to be a, b, c, or d. For convenience of presentation, we have reproduced this portion of the trellis in Figure 10.16a. The left nodes represent the four possible current states of the encoder, whereas the right nodes represent the next states. Clearly, we may coalesce the left and right nodes. By so doing, we obtain the *state graph* of the encoder, shown in Figure 10.16b. The nodes of the figure represent the four possible states of the encoder a, b, c, and d, with each node having two incoming branches and two outgoing branches, following the graphical rule described previously.

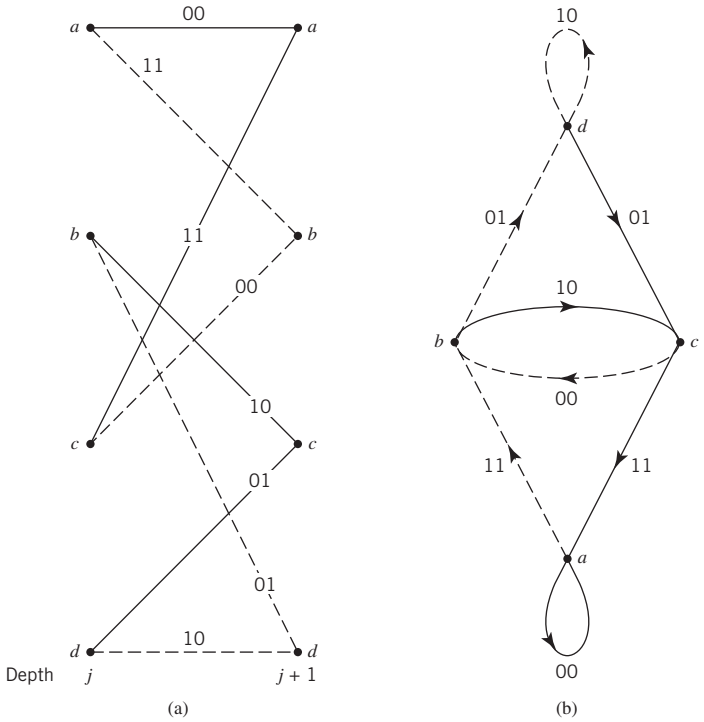


Figure 10.16 (a) A portion of the central part of the trellis for the encoder of Figure 10.13. (b) State graph of the convolutional encoder of Figure 10.13.

The binary label on each branch represents the encoder's output as it moves from one state to another. Suppose, for example, the current state of the encoder is (01), which is represented by node *c*. The application of input symbol 1 to the encoder of Figure 10.13 results in the state (10) and the encoded output (00). Accordingly, with the help of this state diagram, we may readily determine the output of the encoder of Figure 10.13 for any incoming message sequence. We simply start at state *a*, the *initial all-zero state*, and walk through the state graph in accordance with the message sequence. We follow a solid branch if the input is bit 0 and a dashed branch if it is bit 1. As each branch is traversed, we output the corresponding binary label on the branch. Consider, for example, the message sequence (10011). For this input, we follow the path *abcabd*, and therefore output the sequence (11, 10, 11, 11, 01), which agrees exactly with our previous result. Thus, the input–output relation of a convolutional encoder is also completely described by its state graph.

Recursive Systematic Convolutional Codes

The convolutional codes described thus far in this section have been feedforward structures of the nonsystematic variety. There is another type of linear convolutional codes that are the exact opposite, being recursive as well as systematic; they are called *recursive systematic convolutional (RSC) codes*.

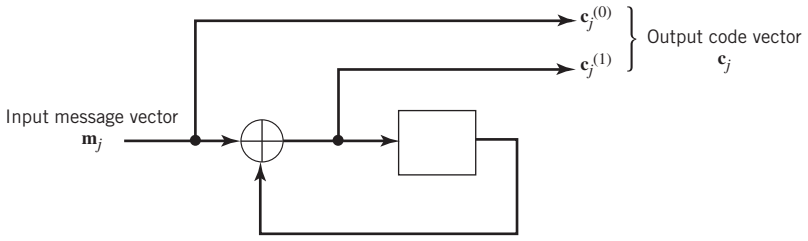


Figure 10.17 Example of a recursive systematic convolutional (RSC) encoder.

Figure 10.17 illustrates a simple example of an RSC code, two distinguishing features of which stand out in the figure:

1. The code is *systematic*, in that the incoming message vector $\mathbf{m}_j$ at time-unit j defines the systematic part of the code vector $\mathbf{c}_j$ at the output of the encoder.
2. The code is *recursive* by virtue of the fact that the other constituent of the code vector, namely the parity-check vector $\mathbf{b}_j$, is related to the message vector $\mathbf{m}_j$ by the *modulo-2 recursive equation*

$$\mathbf{m}_j + \mathbf{b}_{j-1} = \mathbf{b}_j \tag{10.50}$$

where $\mathbf{b}_{j-1}$ is the past value of $\mathbf{b}_j$ stored in the memory of the encoder.

From an analytic point of view, in studying RSC codes, it is more convenient to work in the transform D-domain than the time domain. By definition, we have

$$\mathbf{b}_{j-1} = D[\mathbf{b}_j] \tag{10.51}$$

and therefore rewrite (10.50) in the equivalent form:

$$\mathbf{b}_j = \frac{1}{1+D}[\mathbf{m}_j] \tag{10.52}$$

where the transfer function $1/(1 + D)$ operates on $\mathbf{m}_j$ to produce $\mathbf{b}_j$. With the code vector $\mathbf{c}_j$ consisting of the message vector $\mathbf{m}_j$ followed by the parity-check vector $\mathbf{b}_j$, we may express the code vector $\mathbf{c}_j$ produced in response to the message vector $\mathbf{m}_j$ as follows:

$$\begin{aligned}\mathbf{c}_j &= (\mathbf{m}_j, \mathbf{b}_j) \\ &= \left(1, \frac{1}{1+D}\right)\mathbf{m}_j\end{aligned} \tag{10.53}$$

It follows, therefore, that the *code generator* for the RSC code of Figure 10.17 is given by the matrix

$$\mathbf{G}(D) = \left(1, \frac{1}{1+D}\right) \tag{10.54}$$

Generalizing, we may now make the statement:

> For recursive systematic convolutional codes, the transform-domain matrix $\mathbf{G}(D)$ is easier to use as the code generator than the corresponding time-domain matrix $\mathbf{G}$ whose entries contain sequences of infinite length.

The same statement applies equally well to the parity-check generator $\mathbf{H}(D)$ compared with its time-domain counterpart $\mathbf{H}$.

The rationale behind making convolutional codes recursive is to feed one or more of the tap-outputs in the shift register back to the encoder input, which, in turn, makes the internal state of the shift register depend on past outputs. This modification, compared with a feedforward convolutional code, affects the behavior of error patterns in a profound way, which is emphasized in the following statement:

> A single error in the systematic bits of an RSC code produces an infinite number of parity-check errors due to the use of feedback in the encoder.

This property of recursive convolutional codes turns out to be one of the key factors behind the outstanding performance achieved by the class of turbo codes, to be discussed in Section 10.12. Therein, we shall see that feedback plays a key role not only in the encoder of turbo codes but also the decoder. For reasons that will become apparent later, further work on turbo codes will be deferred to Section 10.12.

10.7 Optimum Decoding of Convolutional Codes

In the meantime, we resume the discussion on convolutional codes whose encoders are of the feedforward variety, aimed at the development of two different decoding algorithms, each of which is optimum according to a criterion of its own.

The first algorithm is the *maximum likelihood (ML) decoding algorithm*; the decoder is itself referred to as the *maximum likelihood decoder* (maximum likelihood estimation was discussed in Chapter 3). A distinctive feature of this decoder is that it produces a codeword as output, the conditional probability of which is always maximized on the assumption that each codeword in the code is equiprobable. From Chapter 3 on probability theory, we recall that the conditional probability density function of a random variable X given a quantity θ can be rethought as the likelihood function of θ with that function being dependent on X, given a parameter θ. We may therefore make the statement:

> In the maximum likelihood decoding of a convolutional code, the metric to be maximized is the likelihood function of a codeword, expressed as a function of the noisy channel output.

The second algorithm is the *maximum a posteriori (MAP) probability decoding algorithm*; the decoder is correspondingly referred to as a *MAP decoder*. In light of this second algorithm's name, we may make the statement:

> In MAP decoding of a convolutional code, the metric to be maximized is the posterior of a codeword, expressed as the product of the likelihood function of a given bit and the a priori probability of that bit.

These two decoding algorithms, *optimal* in accordance with their own respective criteria, are distinguished from each other as follows:

> The ML decoding algorithm produces the most likely codeword as its output. On the other hand, the MAP decoding algorithm operates on the received sequence on a bit-by-bit basis to produce the most likely symbol as output.

Stated in another way, we may say:

> The ML decoder minimizes the probability of selecting the wrong codeword, whereas the MAP decoder minimizes the decoded BER.

Typically, the ML decoder is simpler to implement; hence its popular use in practice. However, the MAP decoding algorithm is preferred over the ML decoding algorithm in the following two situations:

1. The information bits are not equally likely.
2. Iterative decoding is used in the receiver, in which case the a priori probabilities of the message bits change from one iteration to the next; such a situation arises in turbo decoding, which is discussed in Section 10.12.

Applications of the Two Decoding Algorithms

The ML decoding algorithm is applied to convolutional codes in Section 10.8; in so doing, we are, in effect, opting for a simple approach to decode convolutional codes. This simple approach is also applicable to another class of codes, called trellis-coded modulation, which is discussed in Section 10.15.

Then, in Section 10.9 we move on to study the MAP decoding algorithm; the length of that section and the illustrative example in Section 10.10 are testimony to the complexity of this second approach to decoding convolutional codes. Equipped with the MAP algorithm and its modified forms, Section 10.12 and 10.13 discuss their application to turbo codes. It is in the material covered in those two sections that we find the practical benefits of feedback in decoding turbo codes.

10.8 Maximum Likelihood Decoding of Convolutional Codes

We begin the discussion of decoding convolutional codes by first describing the underlying theory of maximum likelihood decoding. The description is best understood by focusing on a trellis that represents each time step in the decoding process with a separate state graph.

Let $\mathbf{m}$ denote a *message vector* and $\mathbf{c}$ denote the corresponding *code vector* applied by the encoder to the input of a discrete memoryless channel. Let $\mathbf{r}$ denote the *received vector*, which, in practice, will invariably differ from the transmitted code vector $\mathbf{c}$ due to additive channel noise. Given the received vector $\mathbf{r}$, the decoder is required to make an *estimate* $\hat{\mathbf{m}}$ of the message vector $\mathbf{m}$. Since there is a one-to-one correspondence between the message vector $\mathbf{m}$ and the code vector $\mathbf{c}$, the decoder may equivalently produce an estimate $\hat{\mathbf{c}}$ of the code vector. We may then put

$$\hat{\mathbf{m}} = \mathbf{m} \text{ if and only if } \hat{\mathbf{c}} = \mathbf{c}$$

Otherwise, a *decoding error* is committed in the receiver. The *decoding rule* for choosing the estimate $\hat{\mathbf{c}}$, given the received vector $\mathbf{r}$, is said to be optimum when the *probability of decoding error* is minimized. In light of the material presented on signaling over AWGN channel in Chapter 7, we may state:

For equiprobable messages, the probability of decoding error is minimized if the estimate $\hat{\mathbf{c}}$ is chosen to maximize the log-likelihood function.

Let $\mathbb{P}(\mathbf{r}|\mathbf{c})$ denote the conditional probability of receiving $\mathbf{r}$, given that $\mathbf{c}$ was sent. The log-likelihood function equals $\ln\mathbb{P}(\mathbf{r}|\mathbf{c})$, where ln denotes the natural logarithm. The *maximum likelihood decoder* for decision making is described as follows:

Choose the estimate $\hat{\mathbf{c}}$ for which the log-likelihood function $\ln\mathbb{P}(\mathbf{r}|\mathbf{c})$ is maximum.

Consider next the special case of a binary symmetric channel. In this case, both the transmitted code vector $\mathbf{c}$ and the received vector $\mathbf{r}$ represent binary sequences of some length N. Naturally, these two sequences may differ from each other in some locations because of errors due to channel noise. Let c_i and r_i denote the ith elements of $\mathbf{c}$ and $\mathbf{r}$, respectively. We then have

$$\mathbb{P}(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^{N} p(r_i|c_i)$$

Correspondingly, the log-likelihood function is

$$\ln\mathbb{P}(\mathbf{r}|\mathbf{c}) = \sum_{i=1}^{N} \ln p(r_i|c_i) \tag{10.55}$$

The term $p(r_i|c_i)$ in (10.55) denotes a *transition probability*, which is defined by

$$p(r_i|c_i) = \begin{cases} p, & \text{if } r_i \neq c_i \\ 1-p, & \text{if } r_i = c_i \end{cases} \tag{10.56}$$

Suppose also that the received vector $\mathbf{r}$ differs from the transmitted code vector $\mathbf{c}$ in exactly d places in the codeword, By definition, the number d is the *Hamming distance* between the vectors $\mathbf{r}$ and $\mathbf{c}$. Hence, we may rewrite the log-likelihood function in (10.55) as follows:

$$\begin{aligned} \ln p(\mathbf{r}|\mathbf{c}) &= d\ln p + (N-d)\ln(1-p) \\ &= d\ln\left(\frac{p}{1-p}\right) + N\ln(1-p) \end{aligned} \tag{10.57}$$

In general, the probability of an error occurring is low enough for us to assume $p < 1/2$. We also recognize that $N\ln(1-p)$ is a constant for all $\mathbf{c}$. Accordingly, we may restate the maximum-likelihood decoding rule for the binary symmetric channel as follows:

Choose the estimate $\hat{\mathbf{c}}$ that minimizes the Hamming distance between the received vector $\mathbf{r}$ and the transmitted vector $\mathbf{c}$.

That is, for the binary symmetric channel, the maximum-likelihood decoder for a convolutional code reduces to a *minimum distance decoder*. In such a decoder, the received vector $\mathbf{r}$ is compared with each possible transmitted code vector $\mathbf{c}$, and the particular one closest to $\mathbf{r}$ is chosen as the correct transmitted code vector. The term

"closest" is used in the sense of minimum number of differing binary symbols (i.e., Hamming distance) between the code and received vectors under investigation.

The Viterbi Algorithm

The equivalence between maximum likelihood decoding and minimum distance decoding for the binary symmetric channel implies that we may decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places. Since a code tree is equivalent to a trellis, we may equally limit our choice to the possible paths in the trellis representation of the code. The reason for preferring the trellis over the tree is that the number of nodes at each time instant does not continue to grow as the number of incoming message bits increases; rather, it remains constant at $2^{\nu-1}$, where ν is the constraint length of the code.

Consider, for example, the trellis diagram of Figure 10.15 for a convolutional code with rate $r = 1/2$ and constraint length $\nu = 3$. We observe that, at time-unit $j = 3$, there are two paths entering any of the four nodes in the trellis. Moreover, these two paths will be identical onward from that point. Clearly, a minimum distance decoder may make a decision at that point as to which of those two paths to retain, without any loss of performance. A similar decision may be made at time-unit $j = 4$, and so on. This sequence of decisions is exactly what the *Viterbi algorithm*[7] does as it walks through the trellis. The algorithm operates by computing a *metric* (i.e., discrepancy) for every possible path in the trellis; hence the following statement:

> The metric for a particular path is defined as the Hamming distance between the coded sequence represented by that path and the received sequence.

Thus, for each node (state) in the trellis of Figure 10.15 the algorithm compares the two paths entering the node. The path with the lower metric is retained and the other path is discarded. This computation is repeated for every time-unit j of the trellis in the range $M \leq j \leq L$, where $M = \nu - 1$ is the encoder's memory and L is the length of the incoming message sequence. The paths that are retained by the algorithm are called *survivor* or *active paths*. For a convolutional code of constraint length $\nu = 3$, for example, no more than $2^{\nu-1} = 4$ survivors and their metrics will ever be stored. The list of $2^{\nu-1}$ paths computed in the manner just described is always guaranteed to contain the maximum-likelihood choice.

A difficulty that may arise in the application of the Viterbi algorithm is the possibility that when the paths entering a state are compared, their metrics are found to be identical. In such a situation, we simply make the choice by flipping a fair coin (i.e., simply make a random guess).

To sum up:

> The Viterbi algorithm is a maximum-likelihood decoder, which is optimum for an AWGN channel as well as a binary symmetric channel.

The algorithm proceeds in a step-by-step fashion, as summarized in Table 10.6.

Table 10.6 Summary of the Viterbi algorithm

The Viterbi algorithm is a maximum likelihood decoder, which is optimal for any discrete memoryless channel. It proceeds in three basic steps. In computational terms, the so-called *add–compare–select (ACS) operation* in Step 2 is at the heart of the Viterbi algorithm.

Initialization

Set the all-zero state of the trellis to zero.

Computation Step 1: time-unit j

Start the computation at some time-unit j and determine the metric for the path that enters each state of the trellis. Hence, identify the survivor and store the metric for each one of the states.

Computation Step 2: time-unit $j + 1$

For the next time-unit $j + 1$, determine the metrics for all $2^{\nu - 1}$ paths that enter a state where ν is the constraint length of the convolutional encoder; hence do the following:

a. *Add* the metrics entering the state to the metric of the survivor at the preceding time-unit j;

b. *Compare* the metrics of all 2^{ν} paths entering the state;

c. *Select* the survivor with the largest metric, store it along with its metric, and discard all other paths in the trellis.

Computation Step 3: continuation of the search to convergence

Repeat Step 2 for time-unit $j < L + L'$, where L is the length of the message sequence and L' is the length of the termination sequence.

Stop the computation once the time-unit $j = L + L'$ is reached.

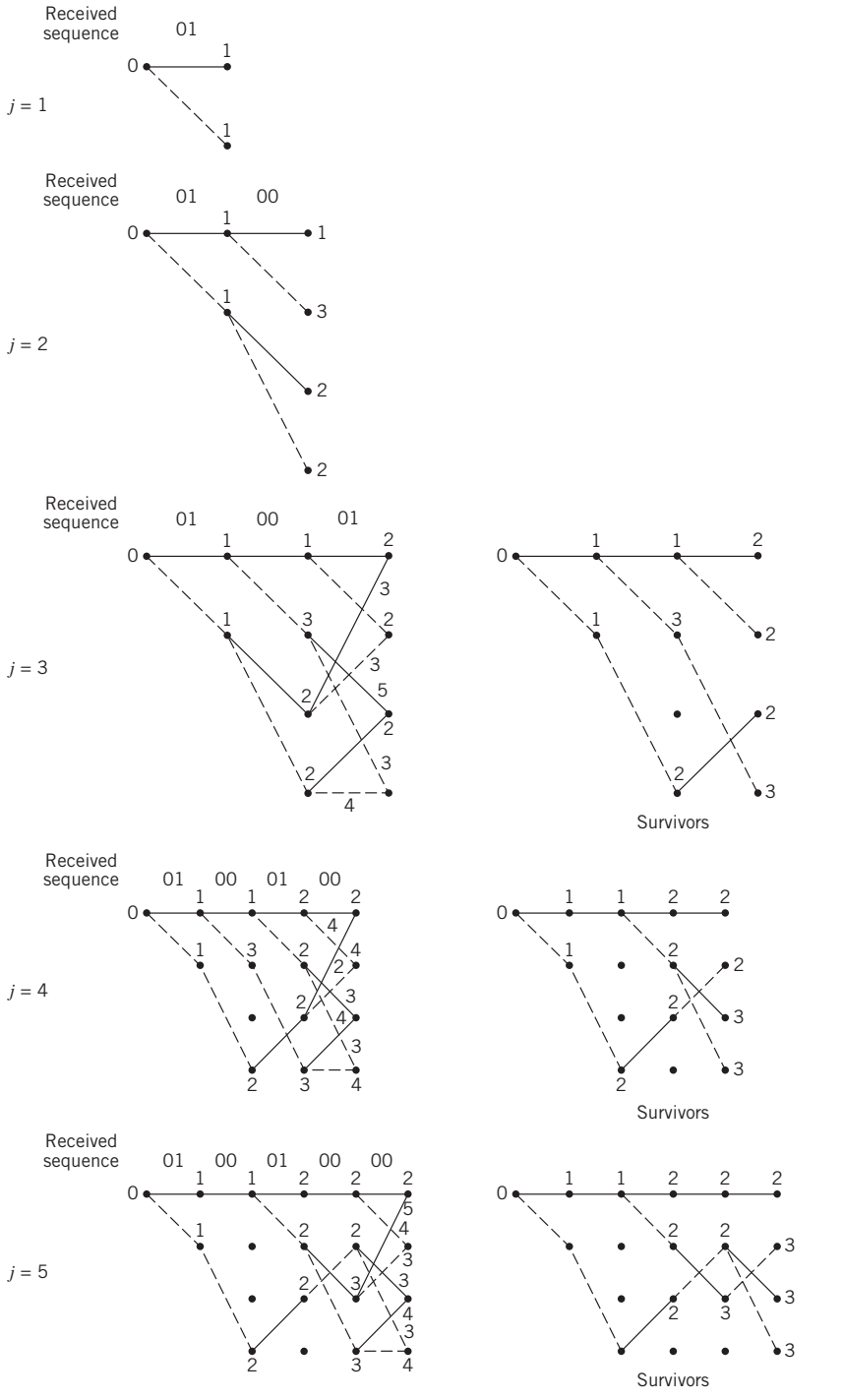
EXAMPLE 5 **Correct Decoding of Received All-Zero Sequence**

Suppose that the encoder of Figure 10.13 generates an all-zero sequence that is sent over a binary symmetric channel and that the received sequence is (0100010000 …). There are two errors in the received sequence due to noise in the channel: one in the second bit and the other in the sixth bit. We wish to show that this double-error pattern is correctable through the application of the Viterbi decoding algorithm.

In Figure 10.18 we show the results of applying the algorithm for time-unit $j = 1, 2, 3, 4, 5$. We see that for $j = 2$ there are (for the first time) four paths, one for each of the four states of the encoder. The figure also includes the metric of each path for each level in the computation.

In the left side of Figure 10.18, for time-unit $j = 3$ we show the paths entering each of the states, together with their individual metrics. In the right side of the figure we show the four survivors that result from application of the algorithm for time-unit $j = 3, 4, 5$. Examining the four survivors in the figure for $j = 5$, we see that the all-zero path has the smallest metric and will remain the path of smallest metric from this point forward. This clearly shows that the all-zero sequence is indeed the maximum likelihood choice of the Viterbi decoding algorithm, which agrees exactly with the transmitted sequence.

Figure 10.18
Illustrating steps in the Viterbi algorithm for Example 5.



EXAMPLE 6

Incorrect Decoding of Received All-Zero Sequence

Suppose next that the received sequence is (1100010000 …), which contains three errors compared with the transmitted all-zero sequence; two of the errors are adjacent to each other and the third is some distance away.

In Figure 10.19, we show the results of applying the Viterbi decoding algorithm for levels $j = 1, 2, 3, 4$. We see that in this second example on Viterbi decoding the correct path has been eliminated by time-unit $j = 3$. Clearly, a triple-error pattern is uncorrectable by the Viterbi algorithm when applied to a convolutional code of rate 1/2 and constraint length $\nu = 3$. The exception to this algorithm is a triple-error pattern spread over a time span longer than one constraint length, in which case it is likely to be correctable.

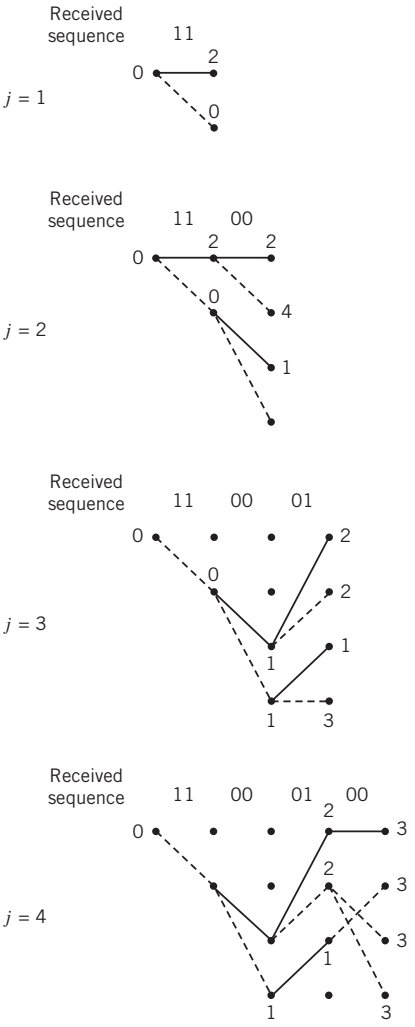


Figure 10.19
Illustrating breakdown of the Viterbi algorithm in Example 6.

What Have We Learned from Examples 5 and 6?

In Example 5 there were two errors in the received sequence, whereas in Example 6 there were three errors, two of which were in adjacent symbols and the third one was some distance away. In both examples the encoder used to generate the transmitted sequence was the same. The difference between the two examples was attributed to the fact that the number of errors in Example 6 was beyond the error-correcting capability of the maximum likelihood decoding algorithm, which is the next topic for discussion.

Free Distance of a Convolutional Code

The performance of a convolutional code depends not only on the decoding algorithm used but also on the distance properties of the code. In this context, the most important single measure of a convolutional code's ability to combat errors due to channel noise is the *free distance* of the code, denoted by d_{free}; it is defined as follows:

> The free distance of a convolutional code is given by the minimum Hamming distance between any two codewords in the code.

A convolutional code with free distance d_{free} can, therefore, correct t errors if, and only if, d_{free} is greater than $2t$.

The free distance can be obtained quite simply from the state graph of the convolutional encoder. Consider, for example, Figure 10.16b, which shows the state graph of the encoder of Figure 10.13. Any nonzero code sequence corresponds to a complete path beginning and ending at the 00 state (i.e., node a). We thus find it useful to split this node in the manner shown in the modified state graph of Figure 10.20, which may be viewed as a *signal-flow graph* with a single input and single output.

A signal-flow graph consists of *nodes* and directed *branches*; it operates by the following set of rules:

1. A branch multiplies the signal at its input node by the *transmittance* characterizing that branch.
2. A node with incoming branches *sums* the signals produced by all of those branches.

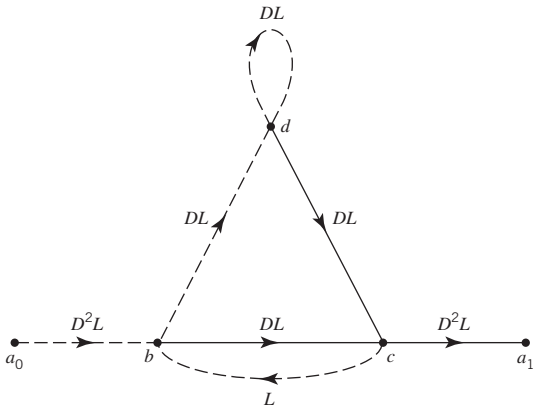


Figure 10.20 Modified state graph of convolutional encoder.

3. The signal at a node is applied equally to all the branches outgoing from that node.
4. The *transfer function* of the graph is the ratio of the output signal to the input signal.

Returning to the signal-flow graph of Figure 10.20, the exponent of D on a branch in this graph describes the Hamming weight of the encoder output corresponding to that branch; the symbol D used here should not be confused with the unit-delay variable in Section 10.6 and the symbol L used herein should not be confused with the length of the message sequence. The exponent of L is always equal to one, since the length of each branch is one. Let $T(D,L)$ denote the *transfer function of the signal-flow graph*, with D and L playing the role of dummy variables. For the example of Figure 10.20, we may readily use rules 1, 2, and 3 to obtain the following input-output relations:

$$\left.\begin{aligned} b &= D^2La_0 + Lc \\ c &= DLb + DLd \\ d &= DLb + DLd \\ a_1 &= D^2Lc \end{aligned}\right\} \tag{10.58}$$

where a_0, b, c, d, and a_1 denote the node signals of the graph. Solving the system of four equations in (10.58) for the ratio a_1/a_0, we obtain the transfer function

$$T(D, L) = \frac{D^5L^3}{1 - DL(1 + L)} \tag{10.59}$$

Using the binomial expansion, we may equivalently express $T(D,L)$ as follows:

$$\begin{aligned} T(D, L) &= D^5L^3(1 - DL(1 + L))^{-1} \\ &= D^5L^3\sum_{i=0}^{\infty}(DL(1 + L))^i \end{aligned}$$

Setting $L = 1$ in this formula, we thus get the *distance transfer function* expressed in the form of a power series as follows:

$$T(D, 1) = D^5 + 2D^6 + 4D^7 + \cdots \tag{10.60}$$

Since the free distance is the minimum Hamming distance between any two codewords in the code and the distance transfer function $T(D,1)$ enumerates the number of codewords that are a given distance apart, it follows that the exponent of the first term in the expansion of $T(D,1)$ in (10.60) defines the free distance. Thus, on the basis of this equation, the convolutional code of Figure 10.13 has the free distance $d_{\text{free}} = 5$.

This result indicates that up to two errors in the received sequence are correctable, as two or fewer transmission errors will cause the received sequence to be at most at a Hamming distance of 2 from the transmitted sequence but at least at a Hamming distance of 3 from any other code sequence in the code. In other words, in spite of the presence of any pair of transmission errors, the received sequence remains closer to the transmitted sequence than any other possible code sequence. However, this statement is no longer true if there are three or more *closely spaced* transmission errors in the received sequence. The observations made here reconfirm the results reported earlier in Examples 5 and 6.

Asymptotic Coding Gain

The transfer function of the encoder's state graph, modified in a manner similar to that illustrated in Figure 10.20, may be used to evaluate a *bound on the BER* for a given decoding scheme; details of this evaluation are, however, beyond the scope of our present discussion.[8] Here, we simply summarize the results for two special channels, namely the binary symmetric channel and the binary-input AWGN channel, assuming the use of binary PSK with coherent detection.

1. *Binary symmetric channel.*

   The binary symmetric channel may be modeled as an AWGN channel with binary PSK as the modulation in the transmitter followed by hard-decision demodulation in the receiver. The transition probability p of the binary symmetric channel is then equal to the BER for the uncoded binary PSK system. From Chapter 7 we recall that for large values of E_b/N_0, denoting the ratio of signal energy per bit-to-noise power spectral density, the BER for binary PSK without coding is dominated by the exponential factor $\exp(-E_b/N_0)$. On the other hand, the BER for the same modulation scheme with convolutional coding is dominated by the exponential factor $\exp(-d_{\text{free}} r E_b/2N_0)$, where r is the code rate and d_{free} is the free distance of the convolutional code. Therefore, as a *figure of merit* for measuring the improvement in error performance made by the use of coding with hard-decision decoding, we may set aside the E_b/N_0 to use the remaining exponent to define the *asymptotic coding gain* (in decibels) as follows:

$$G_a = 10\log_{10}\left(\frac{d_{\text{free}} r}{2}\right) \text{ dB} \tag{10.61}$$

2. *Binary-input AWGN channel.*

   Consider next the case of a memoryless binary-input AWGN channel with no output quantization (i.e., the output amplitude lies in the interval $(-\infty, \infty)$). For this channel, theory shows that for large values of E_b/N_0 the BER for binary PSK with convolutional coding is dominated by the exponential factor $\exp(-d_{\text{free}} r E_b/N_0)$, where the parameters are as previously defined. Accordingly, in this second case, we find that the asymptotic coding gain is defined by

$$G_a = 10\log_{10}(d_{\text{free}} r) \text{ dB} \tag{10.62}$$

Comparing (10.61) and (10.62) for cases 1 and 2, respectively, we see that the asymptotic coding gain for the binary-input AWGN channel is greater than that for the binary symmetric channel by 3 dB. In other words, for large E_b/N_0, the transmitter for a binary symmetric channel must generate an additional 3 dB of signal energy (or power) over that for a binary-input AWGN channel if we are to achieve the same error performance. Clearly, there is an advantage to be gained by using an unquantized demodulator output in place of making hard decisions. This improvement in performance, however, is attained at the cost of increased decoder complexity due to the requirement for accepting analog inputs.

It turns out that the asymptotic coding gain for a binary-input AWGN channel is approximated to within about 0.25 dB by a binary input Q-ary output discrete memoryless channel with the number of representation levels $Q = 8$. This means that, for practical

purposes, we may avoid the need for an analog decoder by using a *soft-decision decoder* that performs finite output quantization (typically, $Q = 8$), and yet realize a performance close to the optimum.

Practical Limitations of the Viterbi Algorithm

When the received sequence is very long, the storage requirement of the Viterbi algorithm becomes too high, in which case some compromises must be made. The approach usually taken in practice is to "truncate" the path memory of the decoder as follows:

> A decoding window of length l is specified and the algorithm operates on a corresponding frame of the received sequence, always stopping after l steps. A decision is then made on the "best" path and the symbol associated with the first branch on that path is released to the user. The symbol associated with the last branch of the path is dropped. Next, the decoding window is moved forward one time interval. A decision on the next code frame is made, and the process is repeated.

Naturally, decoding decisions made in the way just described are no longer truly maximum likelihood, but they can be made almost as good provided that the decoding window is chosen long enough. Experience and analysis have shown that satisfactory results are obtained if the decoding window length l is on the order of five times the constraint length ν of the convolutional code or more.

10.9 Maximum a Posteriori Probability Decoding of Convolutional Codes

Summarizing the discussion on convolutional decoding presented in Section 10.8, we may say that, given a received vector $\mathbf{r}$ that is the noisy version of a convolutionally encoded vector $\mathbf{c}$, the Viterbi algorithm computes the code vector $\hat{\mathbf{c}}$ for which the log-likelihood function is maximum; for a binary symmetric channel, the code vector $\hat{\mathbf{c}}$ minimizes the Hamming distance between the received vector $\mathbf{r}$ and the transmitted vector $\mathbf{c}$. For the more general case of an AWGN channel, this result is equivalent to finding the vector $\hat{\mathbf{c}}$ that is the closest to the received vector $\mathbf{r}$ in Euclidean distance. Simply put then: given the vector $\mathbf{r}$, the Viterbi algorithm finds the most likely vector $\hat{\mathbf{c}}$ that minimizes the conditional probability $\mathbb{P}(\hat{\mathbf{c}} \neq \mathbf{c}|\mathbf{r})$, which is the *sequence error* or the *word error rate*.

In practice, however, we are often interested in the BER, defined as the conditional probability $\mathbb{P}(\hat{m}_i \neq m_i|\mathbf{r})$, where m_i is an estimate of the ith bit of message vector $\hat{\mathbf{m}}$. Recognizing the fact that the BER can indeed assume a value different from the sequence error, we need a probabilistic decoding algorithm that minimizes the BER.

Bahl, Cocke, Jelinek, and Raviv (1974) are credited for deriving an algorithm that maximizes the a posteriori probabilities of the states in the decoding model as well as the transition probability from one state to another. In the course of time, this decoding algorithm has become known as the *BCJR algorithm* in honor of its four inventors. The BCJR algorithm is applicable to any linear code, be it of a block or convolutional kind. However, as we may well expect, computational complexity of the BCJR algorithm is greater than that of the Viterbi algorithm. But, when the message bits in the received

vector **r** are equally likely, the Viterbi algorithm is preferred over the BCJR algorithm. When, however, the message bits are not equally likely, then the BCJR algorithm provides a better decoding performance than the Viterbi algorithm. Moreover, in iterative decoding exemplified by turbo decoding (to be discussed in Section 10.12), the a priori probabilities of the message bits may change from one iteration to the next; in such a scenario, the BCJR algorithm provides the best performance.

Henceforth, the two terminologies, BCJR algorithm and maximum a posteriori probability (MAP) decoding algorithm, are used interchangeably.

The MAP Decoding Algorithm

The function of the MAP decoder is to compute the values of *log-a-posteriori ratios*, on the basis of which estimates of the original message bits are computed in the receiver. In what follows, we derive the *MAP decoding algorithm* for the case of rate = $1/n$ convolutional codes applied to a binary input–continuous output AWGN channel.[9]

Henceforth, in this section, we use the mapping of bits 0 and 1 as follows:

$$\begin{aligned}\text{bit } 0 &\rightarrow \text{level } -1\\ \text{bit } 1 &\rightarrow \text{level } +1\end{aligned}$$

Thus, given a message sequence of block length L, we express the message vector **m** as follows:

$$\mathbf{m} = (m_0, m_1, \ldots, m_{L-1})$$

where

$$m_j = \pm 1 \qquad \text{for } j = 0, 1, \ldots, L-1$$

The individual elements in the message vector **m** are referred to as *message bits*. In any event, the vector **m** is encoded into the codeword **c**, which, in turn, produces the noisy received signal vector **r** at the channel output. Note, however, the elements of the vector **r** can assume positive as well as negative values, which, in theory, can be infinitely large due to the analog nature of the additive channel noise.

Before proceeding further, there are two natural logarithmic concepts, namely log-likelihood ratios, that will occupy our attention in deriving the MAP decoding algorithm:

1. *A priori L-values*, denoted by $L_a(m_j)$, which define the natural logarithmic ratio of a priori probabilities of message bits, $m_j = -1$ and $m_j = +1$, generated by a source at the encoder input in the transmitter.
2. *A posteriori L-values*, denoted by $L_p(m_j)$, which define the log-likelihood ratio of the conditional a posteriori probabilities of the message bits $m_j = -1$ and $m_j = +1$, given the channel output at the decoder input in the receiver.

In what follows, we will focus on $L_p(m_j)$ first, deferring the discussion of $L_a(m_j)$ until later in this section.

With the message $m_j = \pm 1$, there are two conditional probabilities to be considered: $\mathbb{P}(m_j = +1|\mathbf{r})$ and $\mathbb{P}(m_j = -1|\mathbf{r})$. These two probabilities are called the *a posteriori probabilities (APPs)*. In terms of these two APPs, the *log-a-posteriori L-value* is defined by

$$L_p(m_j) = \ln\left(\frac{\mathbb{P}(m_j = +1|\mathbf{r})}{\mathbb{P}(m_j = -1|\mathbf{r})}\right) \tag{10.63}$$

Hereafter, for the sake of brevity, we refer to the $L_p(m_j)$ simply as the *a posteriori L-value* of message bit m_j at time-unit j. Having computed a set of L_p-values, the decoder makes a *hard decision* by applying the two-part formula:

$$\hat{m}_j = \begin{cases} +1 \text{ if } L_p(m_j) > 0, \\ -1 \text{ if } L_p(m_j) < 0, \end{cases} \qquad j = 0, 1, \ldots, L-1 \tag{10.64}$$

where L is the length of the message sequence; L must not be confused with the two L-values, $L_a(m_j)$ and $L_p(m_j)$.

Given the received vector $\mathbf{r}$, the conditional probability $\mathbb{P}(m_j = +1|\mathbf{r})$ is expressed in terms of the joint probability density function $f(m_j = +1, \mathbf{r})$ as follows:

$$\mathbb{P}(m_j = +1|\mathbf{r}) = \frac{f(m_j = +1|\mathbf{r})}{f(\mathbf{r})}$$

where $f(\mathbf{r})$ is the probability density function of the received vector $\mathbf{r}$; this formula follows from the definition of joint probability.

Similarly, we may express the second conditional probability $\mathbb{P}(m_j = -1|\mathbf{r})$ as follows:

$$\mathbb{P}(m_j = -1|\mathbf{r}) = \frac{f(m_j = -1|\mathbf{r})}{f(\mathbf{r})}$$

Accordingly, using these two conditional properties and canceling the common term $f(\mathbf{r})$, we may reformulate the a posteriori L-values of (10.63) in the equivalent form

$$L_p(m_j) = \ln\left(\frac{f(m_j = +1|\mathbf{r})}{f(m_j = -1|\mathbf{r})}\right) \tag{10.65}$$

which sets the stage for deriving the MAP decoding algorithm.

Lattice-based Framework for the Derivation

With computational complexity being at a premium, we propose to exploit the lattice structure of the convolutional code as the basis for deriving the MAP decoding algorithm. To this end, let Σ_j^+ denote the set of all *state-pairs* for which the states $s_j = s'$ and $s_{j+1} = s$ correspond to message bit $m_j = +1$. We may then express the conditional probability density function $f(m_j = +1|\mathbf{r})$ in the expanded form:

$$f(m_j = +1|\mathbf{r}) \propto \sum_{(s', s) \in \Sigma_j^+} f(s_j = s', s_{j+1} = s, \mathbf{r}) \tag{10.66}$$

where the symbol $\propto$ stands for proportionality. In a similar way, we may reformulate the other conditional probability density function as follows:

$$f(m_j = -1|\mathbf{r}) \propto \sum_{(s', s) \in \Sigma_j^-} f(s_j = +1, s_j = -1, \mathbf{r}) \tag{10.67}$$

where Σ_j^- is the set of all state-pairs for which the state-pair $s_j = s'$ and $s_j + 1 = s$ corresponds to the message bit $m_j = -1$. Hence, substituting (10.66) and (10.67) into (10.65) and recognizing that the proportionality factor is common to both (10.66) and

(10.67), thereby canceling out, the a posteriori L_p-value of message bit m_j at time-unit j takes the following equivalent form:

$$L_p(m_j) = \ln\left(\frac{\sum_{(s',s)\in\Sigma_j^+} f(s_j = s', s_{j+1} = s, \mathbf{r})}{\sum_{(s',s)\in\Sigma_j^-} f(s_j = s', s_{j+1} = s, \mathbf{r})}\right) \tag{10.68}$$

Equation (10.68) provides the mathematical basis for forward–backward computation of the MAP decoding algorithm. In this context, it is important to note the following point in (10.68):

> Every branch in the trellis, connecting a state at time-unit j to a state at the next time-unit $j + 1$, is always in one of the two summation terms in (10.68).

Forward–Backward Recursions: Background Terminology and Assumptions

Our next task is to show how the pair of joint probability density functions in (10.68) can be computed *recursively*, using forward and backward recursions.

With this important point in mind, we introduce some new and relevant terminology. First, we express the received vector $\mathbf{r}$ as the *triplet*

$$\mathbf{r} = (\mathbf{r}_{t>j}, \mathbf{r}_j, \mathbf{r}_{t<j})$$

where the two new terms $\mathbf{r}_{t<j}$ and $\mathbf{r}_{t>j}$ denote those portions of the received vector $\mathbf{r}$ that appear *before* and *after* time-unit j, respectively. Moreover, we simplify the notation by using s' and s in place of $s_j = s'$ and $s_{j+1} = s$, respectively, recognizing that the time-unit j is implicitly contained in the $L_p(m_j)$.

In particular, the joint probability density function common to the numerator and denominator in (10.68) is now rewritten as

$$f(s_j = s', s_{j+1} = s, \mathbf{r}) = f(s', s, \mathbf{r}_{t>j}, \mathbf{r}_j, \mathbf{r}_{t<j}) \tag{10.69}$$

Moreover, before proceeding further, we find it instructive to introduce two assumptions that are basic to derivation of the MAP decoding algorithm:

1. *Markovian Assumption*

   In a convolutional code represented by a trellis, the present state of the encoder depends only on two entities: the immediate past state and the input message bit.

   Under this assumption, convolutional encoding of the message vector performed in the transmitter is said to be a *Markov chain*.

2. *Memoryless Assumption*

   The channel connecting the receiver to the transmitter is memoryless.

   In other words, the channel has no knowledge of the past.

Resuming the discussion on the log a posteriori L-value, $L_p(m_j)$ in (10.68), we use the definition of joint probability density function to express the right-hand side of (10.69) as follows:

$$f(s', s, \mathbf{r}_{t>j}, \mathbf{r}_j, \mathbf{r}_{t<j}) = f(\mathbf{r}_{t>j} | s', s, \mathbf{r}_j, \mathbf{r}_{t<j}) f(s', s, \mathbf{r}_j, \mathbf{r}_{t<j})$$

Focusing on the conditional probability density function on the right-hand side of this equality, we invoke the Markovian assumption to recognize that the vector $\mathbf{r}_{t>j}$ representing the received vector $\mathbf{r}$ after time-unit j subsumes knowledge of the following three entities:

- the state $s' = s_j$,
- the vector $\mathbf{r}_j$ at time-unit j, and
- the vector $\mathbf{r}_{j<t}$ received before time-unit j.

Accordingly, we may simplify matters by writing

$$f(\mathbf{r}_{t>j} | s', s, \mathbf{r}_j, \mathbf{r}_{t<j}) = f(\mathbf{r}_{t>j} | s) \tag{10.70}$$

where s denotes the state s_{j+1}.

Next, we again use the definition of joint probability density function to write

$$f(s', s, \mathbf{r}_j, \mathbf{r}_{t<j}) = f(s, \mathbf{r}_j | s', \mathbf{r}_{t<j}) \, f(s', \mathbf{r}_{t<j})$$

Focusing on the second conditional probability density function $f(s, \mathbf{r}_j | s', \mathbf{r}_{t<j})$ and invoking the Markovian assumption one more time, we recognize that the received vector $\mathbf{r}_j$ at time-unit j subsumes knowledge of the past vector $\mathbf{r}_{t<j}$. Hence, we may further simplify matters by writing

$$f(s, \mathbf{r}_j | s', \mathbf{r}_{t<j}) = f(s, \mathbf{r}_j | s') \tag{10.71}$$

where the states $s = s_{j+1}$ and $s' = s_j$.

Collecting the results obtained in (10.70) and (10.71), we are finally ready to express the probability density function common to the numerator and denominator of (10.68) as follows:

$$f(s', s, \mathbf{r}) = f(\mathbf{r}_{t>j} | s) f(s, \mathbf{r}_j | s') f(s', \mathbf{r}_{t<j}) \tag{10.72}$$

which provides the mathematical basis for recursive implementation of the MAP decoding algorithm.

Three New Algorithmic Metrics

To simplify the computational steps involved in deriving the algorithm, we now introduce the following three *algorithmic metrics:*

$$\alpha_j(s') = f(s', \mathbf{r}_{t<j}) \tag{10.73}$$

$$\gamma_j(s', s) = f(s, \mathbf{r}_j | s') \tag{10.74}$$

$$\beta_{j+1}(s) = f(\mathbf{r}_{t>j} | s) \tag{10.75}$$

Using these three metrics, we may finally express the probability density function common to the numerator and denominator of (10.68) in the simplified form:

$$f(s', s, \mathbf{r}) = \beta_{j+1}(s) \gamma_j(s', s) \alpha_j(s') \tag{10.76}$$

in light of which, hereafter, the three metrics are referred to as follows:

forward metric	$\alpha_j(s')$
branch metric	$\gamma_j(s', s)$
backward metric	$\beta_{j+1}(s)$

As the names would imply, the forward and backward metrics play key roles in the forward and backward recursions of the MAP decoding algorithm, respectively. As for the branch metric, its role is to couple these two recursions to work together in a harmonious manner.

Forward Recursion

Updating the forward metric has the effect of moving from state s' at time-unit j to state s at time-unit $j + 1$; hence, we write

$$\begin{aligned}\alpha_{j+1}(s) &= f(s, \mathbf{r}_{t<j+1}) \\ &= \sum_{s' \in \sigma_j} f(s', s, \mathbf{r}_{t<j+1})\end{aligned}$$

where σ_j is the set of all the states at time-unit j. Using the definition of a joint probability density function, we write

$$\begin{aligned}\alpha_{j+1}(s) &= \sum_{s' \in \sigma_j} f(s, \mathbf{r}_j | s', \mathbf{r}_{t<j}) f(s', \mathbf{r}_{t<j}) \\ &= \sum_{s' \in \sigma_j} f(s, \mathbf{r}_j | s') f(s', \mathbf{r}_{t<j})\end{aligned}$$

where, in the second line, we used the Markovian assumption for $\mathbf{r}_j$ subsuming $\mathbf{r}_{t<j}$. Hence, using the defining equations for the branch and forward metrics in (10.74) and (10.73), respectively, we simplify matters by writing

$$\alpha_{j+1}(s) = \sum_{s' \in \sigma_j} \gamma_j(s', s)\alpha_j(s') \tag{10.77}$$

For obvious reasons, (10.77) is called the *forward recursion*; this recursion is illustrated graphically in Figure 10.21a.

Backward Recursion

To formulate the recursion for the backward metric, we move from state s at time-unit $j + 1$ back to state s' at time-unit j. Adapting the use of (10.75) to the scenario just described, we write

$$\beta_j(s') = f(\mathbf{r}_{t>j-1} | s')$$

The portion of received vector denoted by $\mathbf{r}_{t>j-1}$ may be equivalently expressed as follows:

$$\begin{aligned}\mathbf{r}_{t>j-1} &= \mathbf{r}_{t+1>j} \\ &= (\mathbf{r}_j, \mathbf{r}_{t>j})\end{aligned}$$

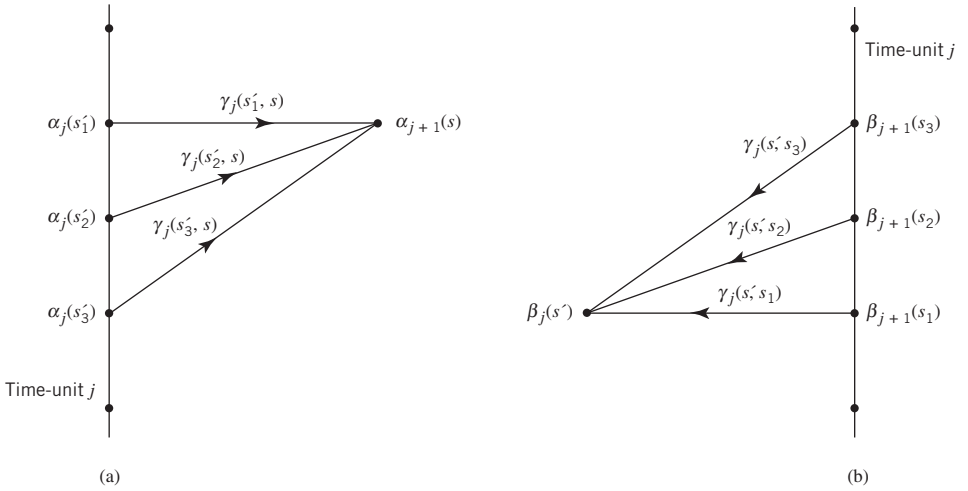


Figure 10.21 Illustrating the computation of forward-metric and backward-metric recursions.

Correspondingly, the backward metric $\beta_j(s')$ is reformulated as shown by

$$\begin{aligned}\beta_j(s') &= f(\mathbf{r}_j, \mathbf{r}_{t>j}|s') \\ &= \sum_{s \in \sigma_{j+1}} f(s, \mathbf{r}_j, \mathbf{r}_{t>j}|s')\end{aligned}$$

where σ_{j+1} is the set of all states at time-unit j. Here again, using the definition of joint probability density function, we write

$$\begin{aligned}\beta_j(s') &= \sum_{s \in \sigma_{j+1}} f(s, \mathbf{r}_j, \mathbf{r}_{t>j}|s') \\ &= \sum_{s \in \sigma_{j+1}} \frac{1}{\mathbb{P}(s')} f(s', s, \mathbf{r}_j, \mathbf{r}_{t>j}) \\ &= \sum_{s \in \sigma_{j+1}} \frac{1}{\mathbb{P}(s')} f(\mathbf{r}_{t>j}|s', s, \mathbf{r}_j) f(s', s, \mathbf{r}_j)\end{aligned}$$

To simplify matters, we note the following two points:

1. Under the memoryless assumption, the received vector $\mathbf{r}_{t>j}$ at the channel output depends only on the state in which the encoder was residing at $j-1$, namely s. We may, therefore, write

$$\begin{aligned}f(\mathbf{r}_{t>j}|s', s, \mathbf{r}_j) &= f(\mathbf{r}_{t>j}|s) \\ &= \beta_{j+1}(s)\end{aligned}$$

2. Invoking the definition of joint probability density function one more time, we have

$$\begin{aligned}f(s', s, \mathbf{r}_j) &= f(s, \mathbf{r}_j|s')\mathbb{P}(s') \\ &= \gamma_j(s', s)\mathbb{P}(s')\end{aligned}$$

Accordingly, substituting the two results under points 1 and 2 into the formula for $\beta_j(s')$ and canceling the common term $\mathbb{P}(s')$ we get

$$\beta_j(s') = \sum_{s \in \sigma_{j+1}} \gamma_j(s', s)\beta_{j+1}(s) \tag{10.78}$$

For obvious reasons, (10.78) is called the *backward recursion*; this second recursion is illustrated graphically in Figure 10.21b.

Initial Conditions for Forward and Backward Recursions

Typically, the encoder starts in the all-zero state, denoted by $s_0 = \mathbf{0}$. Correspondingly, the forward recursion of (10.77) begins operating at time-unit $j = 0$ under the following initial condition:

$$\alpha_0(s) = \begin{cases} 1, & s = \mathbf{0} \\ 0, & s \neq \mathbf{0} \end{cases} \tag{10.79}$$

which follows from the fact that the convolutional encoder starts in the all-zero state. Thus, $\alpha_{j+1}(s)$ is recursively computed forward in time at $j = 0, 1, \ldots, K-1$, where the overall length of the input data stream is

$$K = L + L'$$

in which L and L' denote the lengths of the message and termination sequences.

Similarly, the backward recursion of (10.78) begins at time-unit $j = K$ under the following initial condition:

$$\beta_K(s) = \begin{cases} 1, & s = \mathbf{0} \\ 0, & s \neq \mathbf{0} \end{cases} \tag{10.80}$$

Since the encoder ends in the all-zero state, we recursively compute $\beta_j(s')$ backward in time at $j = K-1, K-2, \ldots, 0$.

Branch Metric Evaluation for the AWGN Channel

Thus far, we have accounted for all the issues important to the MAP decoder except for the discrete-input, continuous-output AWGN channel, which naturally comes into play in evaluating the branch metric: a necessary requirement. This issue was discussed in Example 10 in Chapter 5. For this evaluation, we first rewrite the defining equation (10.74) as follows:

$$\begin{aligned} \gamma_j(s', s) &= f(s, \mathbf{r}_j | s') \\ &= \frac{1}{\mathbb{P}(s')} f(s, s', \mathbf{r}_j) \\ &= \left(\frac{\mathbb{P}(s', s)}{\mathbb{P}(s')}\right) \cdot \left(\frac{f(s', s, \mathbf{r}_j)}{\mathbb{P}(s', s)}\right) \\ &= \mathbb{P}(s|s') f(\mathbf{r}_j | s', s) \end{aligned}$$

which may be transformed into a more desirable form that involves the message bit m_j and the corresponding code vector $\mathbf{c}_j$, as shown by

$$\gamma_j(s', s) = \mathbb{P}(m_j) f(\mathbf{r}_j | \mathbf{c}_j) \tag{10.81}$$

Justification for this transformation may be explained as follows:

1. The transition from the state $s' = s_j$ to the new state $s = s_{j+1}$ is attributed to the message bit inputing the convolutional encoder at time-unit j; hence, we may substitute the probability $\mathbb{P}(m_j)$ for the conditional probability $\mathbb{P}(s|s')$.
2. The state transition (s, s') may be viewed as another way of referring to the code vector $\mathbf{c}_j$; hence, we may substitute the conditional probability $\mathbb{P}(\mathbf{r}_j|\mathbf{c}_j)$ for $f(\mathbf{r}_j|s, s')$.

In (10.81), m_j is the message bit at the encoder's input and $\mathbf{c}_j$ is the code vector defining the encoded bits pertaining to the state transition $s' \rightarrow s$ at time-unit j. When this state transition is a valid one, the conditional probability density function $f(\mathbf{r}_j | \mathbf{c}_j)$, defining the input–output statistical behavior of the channel, assumes the following form:

$$f(\mathbf{r}_j|\mathbf{c}_j) = \left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n \exp\left(-\frac{E_s}{N_0}\|\mathbf{r}_j - \mathbf{c}_j\|^2\right) \tag{10.82}$$

where E_s is transmitted energy per symbol, n is the number of bits in each codeword, $N_0/2$ is the power spectral density of the additive white Gaussian channel noise, and $\|\mathbf{r}_j - \mathbf{c}_j\|^2$ is the squared Euclidean distance between the transmitted vector $\mathbf{c}_j$ at the channel input and the received vector $\mathbf{r}_j$ at the channel output at time-unit j. Thus, substituting (10.82) into (10.81) yields

$$\gamma_j(s', s) = \mathbb{P}(m_j)\left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n \exp\left(-\frac{E_s}{N_0}\|\mathbf{r}_j - \mathbf{c}_j\|^2\right) \tag{10.83}$$

This equation holds if, and only if, the state transition $s' \rightarrow s$ at time-unit j is a valid one; otherwise, the state-transition probability $p(s', s)$ is zero, in which case the branch metric $\gamma_j(s', s)$ is also zero.

A priori L-value, $L_a(m_j)$

At this point in the discussion, we are ready to revisit the a priori L-value $L_a(m_j)$, introduced previously on page 624. Specifically, with the message bit m_j taking the value +1 or –1, we may follow the format of (10.63) to define the *a priori L-value* of m_j as follows:

$$\begin{aligned} L_a(m_j) &= \ln\frac{\mathbb{P}(m_j = +1)}{\mathbb{P}(m_j = -1)} \\ &= \ln\left(\frac{\mathbb{P}(m_j = +1)}{1 - \mathbb{P}(m_j = +1)}\right) \end{aligned} \tag{10.84}$$

where, in the second line, we used the following axiom from probability theory:

$$\mathbb{P}(m_j = +1) + \mathbb{P}(m_j = -1) = 1$$

or, equivalently,

$$\mathbb{P}(m_j = -1) = 1 - \mathbb{P}(m_j = +1)$$

Solving the second line of (10.84) for $\mathbb{P}(m_j = +1)$ in terms of the *a priori L-value* $L_a(m_j)$, we get

$$\mathbb{P}(m_j = +1) = \frac{1}{1 + \exp(-L_a(m_j))}$$

Correspondingly,

$$\mathbb{P}(m_j = -1) = \frac{\exp(-L_a(m_j))}{1 + \exp(-L_a(m_j))}$$

This latter pair of equations for the two probabilities of $m_j = -1$ and $m_j = +1$ may be combined into a single equation, as shown by

$$\mathbb{P}(m_j) = \left(\frac{\exp(-L_a(m_j)/2)}{1 + \exp(-L_a(m_j))}\right)\exp\left(\frac{1}{2}m_j L_a(m_j)\right) \tag{10.85}$$

where $m_j = \pm 1$. The important point to note in (10.85) is that the first term on the right-hand side of the equation turns out to be independent of $m_j = \pm 1$; hence, this term may be treated as a constant.

Turning next to the exponential term in (10.83), we may express the exponent of the second term as follows:

$$\begin{aligned} -\frac{E_s}{N_0}\|\mathbf{r}_j - \mathbf{c}_j\|^2 &= -\frac{E_s}{N_0}\left[\sum_{l=1}^{n}(r_{jl} - c_{jl})^2\right] \\ &= -\frac{E_s}{N_0}\left[\sum_{l=1}^{n}(r_{jl}^2 - 2r_{jl}c_{jl} + c_{jl}^2)\right] \\ &= -\frac{E_s}{N_0}(\|\mathbf{r}_j\|^2 - 2\mathbf{r}_j^{\mathrm{T}}\mathbf{c}_j + \|\mathbf{c}_j\|^2) \end{aligned} \tag{10.86}$$

where E_s is the transmitted symbol energy, and the terms inside the parentheses are

$$\|\mathbf{r}_j\|^2 = \sum_{l=1}^{n}(r_{jl})^2 \tag{10.87}$$

$$\mathbf{r}_j^T\mathbf{c}_j = \sum_{l=1}^{n} r_{jl}c_{jl} \tag{10.88}$$

$$\|\mathbf{c}_j\|^2 = \sum_{l=1}^{n}(c_{jl})^2 = n \tag{10.89}$$

The terms r_{jl} and c_{jl} denote the individual bits in the received vector $\mathbf{r}_j$ and code vector $\mathbf{c}_j$ at time-unit j, and n denotes the number of bits in each of $\mathbf{r}_j$ and $\mathbf{c}_j$. Note also that in (10.88) the term $\mathbf{r}_j^{\mathrm{T}}\mathbf{c}_j$ denotes the inner product of the vectors $\mathbf{r}_j$ and $\mathbf{c}_j$.

In light of (10.87) to (10.89), we make three observations:

1. The term $(E_s/N_0)\|\mathbf{r}_j\|^2$ depends only on the channel SNR and the squared magnitude of the received vector $\mathbf{r}_j$.
2. The third product term $(E_s/N_0)\|\mathbf{c}_j\|^2$ depends only on the channel SNR and the squared magnitude of the transmitted code vector $\mathbf{c}_j$.
3. The remaining product term $2(E_s/N_0)\mathbf{r}_j^T\mathbf{c}_j$ is the only one that contains useful information for detection in the receiver by virtue of the inner product $\mathbf{r}_j^T\mathbf{c}_j$ that *correlates* the received vector $\mathbf{r}$ with the transmitted code vector $\mathbf{c}$, as shown in (10.88).

In light of these observations and the observation made previously that the bracketed fractional term in (10.85) does not depend on whether the symbol m_j is +1 or –1, we may simplify the formula for the transition metric $\gamma_j(s', s)$ in (10.83) as follows:

$$\gamma_j(s', s) = A_j B_j \exp\left(\frac{1}{2}m_j L_a(m_j)\right)\exp\left(\frac{1}{2}L_c(\mathbf{r}_j^T\mathbf{c}_j)\right), \qquad j = 0, 1, \ldots, L-1 \tag{10.90}$$

where L_c denotes the *channel reliability factor*, defined by

$$L_c = \frac{4E_s}{N_0} \tag{10.91}$$

As for the two multiplying factors A_j and B_j, they are respectively defined by

$$A_j = \frac{\frac{1}{2}\exp(-L_a(m_j))}{1 + \exp(-L_a(m_j))}, \qquad j = 0, 1, \ldots, L-1 \tag{10.92}$$

and

$$B_j = \left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n \exp\left[-\frac{E_s}{N_0}(\|\mathbf{r}_j\|^2 + n)\right], \qquad j = 0, 1, \ldots, L-1 \tag{10.93}$$

where, as before, n is the number of bits in each transmitted codeword.

Equations (10.90), (10.92), and (10.93) apply to the message bits of length L. However, for the *termination bits* we have

$$\mathbb{P}(m_j) = 1 \quad \text{and} \quad L_a(m_j) = \pm\infty, \qquad j = L, L+1, \ldots, K-1 \tag{10.94}$$

for each valid state transition; the K in (10.94) denotes the combined length of the message and termination bits. Accordingly, (10.90) for the termination bits simplifies to

$$\gamma_j(s', s) = B_j \exp\left(\frac{1}{2}L_c(\mathbf{r}_j^T\mathbf{c}_j)\right), \qquad j = L, L+1, \ldots, K-1 \tag{10.95}$$

Examining (10.92), we find that the factor A_j is independent of the algebraic sign of message bit m_j; it is therefore a constant. Moreover, from (10.76) and the follow-up formulas of (10.77) and (10.78) for updating recursive computations of the forward and backward metrics, we find that the joint probability density function $f(s', s, \mathbf{r})$ contains the factors

$$\prod_{j=0}^{L-1} A_j \quad \text{and} \quad \prod_{j=0}^{K-1} B_j$$

With these factors being common to every term in the numerator and denominator of (10.68), they both cancel out and may, therefore, be ignored. Thus, we may simplify (10.90) and (10.95) into the following two-part formula:

$$\gamma_j(s', s) = \begin{cases} \exp\left(\frac{1}{2}m_j L_a(m_j)\right)\exp\left(\frac{1}{2}L_c(\mathbf{r}_j^T\mathbf{c}_j)\right), & j = 0, 1, \ldots, L-1 \text{ for message bits} \\ \exp\left(\frac{1}{2}L_c(\mathbf{r}_j^T\mathbf{c}_j)\right), & j = L, L+1, \ldots, K-1 \text{ for termination bits} \end{cases} \tag{10.96}$$

One last comment is in order. When the original message bits are *equally likely*, we have

$$\mathbb{P}(m_j) = \frac{1}{2} \text{ and } L_a(m_j) = 0 \quad \text{for all } j \tag{10.97}$$

Under these two conditions, we have a simple expression for the transition metric for the entire stream of bits, as shown by

$$\gamma_j(s', s) = \exp\left(\frac{1}{2}L_c(\mathbf{r}_j^T\mathbf{c}_j)\right), \quad j = 0, 1, \ldots, K-1 \tag{10.98}$$

The a Posteriori *L*-Value Finalized

With the forward and backward recursions as well as the branch metric that ties them together all now at hand, we are equipped to finalize the formula for computing the a posteriori L-value $L_p(m_j)$ defined way back in (10.68). Specifically, using (10.69) and (10.76), we may now write

$$\begin{aligned} L_p(m_j) &= \ln\left(\frac{\sum_{(s',s)\in\Sigma_j^+} f(s_j{=}s', s_{j+1}{=}s, \mathbf{r})}{\sum_{(s',s)\in\Sigma_j^-} f(s_j{=}s', s_{j+1}{=}s, \mathbf{r})}\right) \\ &= \ln\left(\frac{\sum_{(s',s)\in\Sigma_j^+} f(s', s, \mathbf{r})}{\sum_{(s',s)\in\Sigma_j^-} f(s', s, \mathbf{r})}\right) \\ &= \ln\left(\frac{\sum_{(s',s)\in\Sigma_j^+} \beta_{j+1}(s)\gamma_j(s', s)\alpha_j(s')}{\sum_{(s',s)\in\Sigma_j^-} \beta_{j+1}(s)\gamma_j(s', s)\alpha_j(s')}\right) \end{aligned} \tag{10.99}$$

It is the a posterior L-value $L_p(m_j)$ defined in the last line of (10.99), which is delivered by the MAP decoding algorithm given the received vector $\mathbf{r}$.

Summary of the MAP Algorithm

Starting with given AWGN channel values, namely E_s/N_0 and received vector $\mathbf{r}_j$ at time-unit j, the *computational flow diagram* of Figure 10.22 provides a visual summary of the key recursions involved in using the MAP decoding algorithm. Specifically, the functional blocks pertaining to the forward metric $\alpha_{j+1}(s)$, the backward metric $\beta_j(s')$, the branch metric $\gamma_j(s', s)$, and the a posteriori L-value $L_{\mathrm{p}}(m_j)$ are all identified together with their respective equation numbers.

Modifications of the MAP Decoding Algorithm

The MAP algorithm, credited to Bahl *et al.* (1974), is roughly three times as computationally complex as the Viterbi algorithm. It was on account of this high computational complexity that the MAP algorithm was largely ignored in the literature for almost two decades. However, its pioneering application in turbo codes by Berrou *et al.* (1993) re-ignited interest in the MAP algorithm, which, in turn, led to the formulation of procedures for significant reductions in computational complexity.

Specifically, we may mention the following two modifications of the MAP algorithm, the first one being exact and the second one being approximate:

1. *Log-MAP Algorithm*

   Examination of the forward and backward metrics of the MAP algorithm for continuous-output AWGN channels reveals that they are sums of exponential terms, one for each valid state transition in the trellis. This finding, in turn, leads to the idea of simplifying the MAP computations by making use of the following identity (Robertson *et al.*, 1995):

$$\ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|}) \tag{10.100}$$

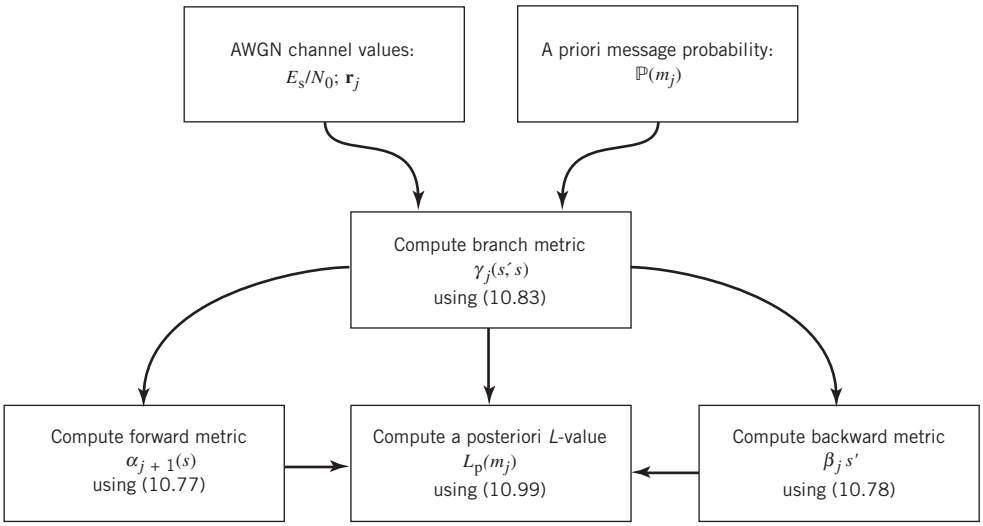


Figure 10.22 Computational flow diagram displaying the key recursions in the MAP algorithm.

where the computationally difficult operation $\ln(e^x + e^y)$ is replaced by the sum of two simpler computations:

a. the *max function,* $\max(x,y)$, equals x or y, depending on which is larger;
b. the *correction term,* $\ln(1 + e^{-|x-y|})$, may be evaluated using a look-up table.

The resulting algorithm, called the *log-MAP algorithm*, is considerably simpler in implementation and provides greater numerical stability than the original MAP algorithm. We say so because its formulation is based on two relatively simple entities: a max function and a look-up table. Note, however, that in developing the log-MAP algorithm, no approximations whatsoever are made.[10]

2. *Max-log-MAP Algorithm*

We may simplify the computational complexity of the MAP decoding algorithm even further by ignoring the correction term $\ln(1 + e^{-|x-y|})$ altogether. In effect, we simply use the approximation

$$\ln(e^x + e^y) \approx \max(x, y) \tag{10.101}$$

The correction term, ignored in this approximate formula, is bounded by

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

The approximate formula of (10.101) yields reasonably good results whenever the condition

$$|\max(x, y)| \geq 7$$

holds. The decoding algorithm that uses the max function $\max(x,y)$ in place of $\ln(e^x, e^y)$ is called the *max-log-MAP algorithm*. In this simplified algorithm, the max function plays a role similar to the ACS described previously in the Viterbi algorithm; we therefore find that the forward recursion in the max-log-MAP algorithm is equivalent to a forward Viterbi algorithm, and the backward recursion in the max-log-MAP algorithm is equivalent to a Viterbi algorithm performed in the backward direction. In other words, computational complexity of the max-log-MAP algorithm is roughly twice that of the Viterbi algorithm, thereby providing a significant improvement in computational terms over the original MAP decoding algorithm. However, unlike the log-MAP algorithm, this improvement is attained at the expense of some degradation in decoding performance.

Details of the Max-Log-MAP Algorithm

To develop a detailed mathematical description of the max-log-MAP algorithm, we have to come up with simplified computations of the forward metric $\alpha_{j+1}(s)$ and backward metric $\beta_j(s')$, both of which play critical roles in computing the log-a-posteriori L-value $L(m_j)$ in (10.99). To this end, we introduce three new definitions in the log-domain:

$$\alpha_j^*(s') = \ln\alpha_j(s'), \text{ equivalently, } \alpha_j(s') = \exp(\alpha_j^*(s')) \tag{10.102}$$

$$\beta_{j+1}^*(s) = \ln\beta_{j+1}(s), \text{ equivalently, } \beta_{j+1}(s) = \exp(\beta_{j+1}^*(s)) \tag{10.103}$$

$$\gamma_j^*(s', s) = \ln\gamma_j(s', s), \text{ equivalently, } \gamma_j(s', s) = \exp(\gamma_j^*(s', s)) \tag{10.104}$$

where the asterisk for all three metrics is intended to signify the use of natural logarithm and must, therefore, not be confused with complex conjugation.

The motivation for these new definitions is to exploit the physical presence of exponentials in the forward and backward metrics so as to facilitate applying the approximate formula of (10.101). Thus, substituting the recursion of (10.104) into (10.77), we get

$$\begin{aligned}\alpha^*_{j+1}(s) &= \ln\left(\sum_{s' \in \sigma_j} \gamma_j(s', s)\alpha_j(s')\right) \\ &= \ln\left(\sum_{s' \in \sigma_j} \exp(\gamma^*_j(s', s) + \alpha^*_j(s'))\right)\end{aligned} \tag{10.105}$$

where σ_j is a subset of Σ_j. Hence, application of the approximate formula of (10.101) yields

$$\alpha^*_{j+1}(s) \approx \max_{s' \in \sigma_j} (\gamma^*_j(s', s) + \alpha^*_j(s')), \qquad j = 0, 1, \ldots, K-1 \tag{10.106}$$

Equation (10.106) indicates that, for each path in the trellis from the old state s' at time-unit j to the updated state s at time-unit $j+1$, the max-log MAP algorithm adds the branch metric $\gamma^*_j(s', s)$ to the old value $\alpha^*_j(s')$ to produce the updated value $\alpha^*_{j+1}(s)$; this update is the "maximum" of all the α^* values of the previous paths terminating on the state $s_{j+1} = s$, that is, $j = 1, 0, \ldots, K-1$. The process just described may be thought of as that of selecting the one particular path viewed as the "survivor" with all the other paths in the trellis reaching the state s being discarded. We may, therefore, view (10.106) as a mathematical basis for describing the forward recursion in the max-log-MAP algorithm in exactly the same way as the forward recursion in the Viterbi algorithm.

Proceeding in a manner similar to that for the forward recursion, we may write

$$\begin{aligned}\beta^*_j(s') &= \ln\left(\sum_{s \in \sigma_{j+1}} \gamma_j(s', s)\beta_{j+1}(s)\right) \\ &= \ln\left(\sum_{s \in \sigma_{j+1}} \exp(\gamma^*_j(s', s) + \beta^*_{j+1}(s))\right)\end{aligned} \tag{10.107}$$

whose approximate form is given by

$$\beta^*_j(s') \approx \max_{s \in \sigma_{j+1}} (\gamma^*_j(s', s) + \beta^*_{j+1}(s)), \qquad j = K-1, \ldots, 1, 0 \tag{10.108}$$

Next, proceeding onto the branch metric, we may similarly write the two-part formula

$$\gamma^*_j(s', s) = \begin{cases} \frac{1}{2}m_j L_a(m_j) + \frac{1}{2}L_c \mathbf{r}_j^T \mathbf{c}_j, & j = 0, 1, \ldots, L-1 \text{ for message bits} \\ \frac{1}{2}L_c \mathbf{r}_j^T \mathbf{c}_j, & j = L, L+1, \ldots, K-1 \text{ for termination bits} \end{cases} \tag{10.109}$$

where, for the message bits in the first line of the equation, the additive term $\frac{1}{2}m_j L(m_j)$ accounts for a priori information.

At long last, using (10.105), (10.107), and (10.109) in (10.99), we may finally express the a posteriori L-value for the log-MAP algorithm as follows:

$$L_p(m_j) = \ln\left(\frac{\sum_{(s',s)\in\Sigma_j^+}\beta_{j+1}(s)\gamma_j(s',s)\alpha_j(s')}{\sum_{(s',s)\in\Sigma_j^-}\beta_{j+1}(s)\gamma_j(s',s)\alpha_j(s')}\right) \tag{10.110}$$

$$= \ln\sum_{(s',s)\in\Sigma_j^+}\exp(\beta_{j+1}^*(s)+\gamma_j^*(s',s)+\alpha_j^*(s')) - \ln\sum_{(s',s)\in\Sigma_j^-}\exp(\beta_{j+1}^*(s)+\gamma_j^*(s',s)+\alpha_j^*(s'))$$

A couple of reminders:

- Σ_j^+ is the set of all state pairs $s_j = s'$ and $s_{j+1} = s$ that correspond to the original message bit $m_j = +1$ at time-unit j.
- Σ_j^- is the set of all other state pairs $s_j = s'$ and $s_{j+1} = s$ that correspond to the original message bit $m_j = -1$ at time-unit j.

Correspondingly, the approximate form of the $L_p(m_j)$ in the max-log-MAP algorithm is defined by

$$L_p(m_j) \approx \max_{(s',s)\in\Sigma_j^+}(\beta_{j+1}^*(s)+\gamma_j(s',s)+\alpha_j^*(s')) - \max_{(s',s)\in\Sigma_j^-}(\beta_{j+1}^*(s)+\gamma_j^*(s',s)+\alpha_j^*(s')) \tag{10.111}$$

10.10 Illustrative Procedure for Map Decoding in the Log-Domain

In the preceding section we described three different algorithms for decoding a convolutional code, as summarized here:

1. *The BCJR algorithm*, which distinguishes itself from the Viterbi algorithm in that it performs MAP decoding on a bit-by-bit basis. However, a shortcoming of this algorithm is its computational complexity, which, as mentioned previously, is roughly three times that of the Viterbi algorithm for the same convolutional code.
2. *The log-MAP-algorithm*, which simplifies the BCJR algorithm by replacing the computationally difficult logarithmic operation, namely $\ln(e^x + e^y)$, with the so-called *max function* plus a *look-up table* for evaluating $\ln(1 + e^{-|x-y|})$ in accordance with (10.100). The attractive feature of this second algorithm is twofold:
   - transformation of the BCJR algorithm into the log-MAP algorithm is exact;
   - its computational complexity is twice that of the Viterbi algorithm, thereby providing a significant reduction in complexity compared to the BCJR algorithm.

3. *The max-log-MAP algorithm*, which simplifies computational complexity even further by doing away with the look-up table; this simplification may result in some degradation in decoding performance depending on the application of interest.

In this section, we illustrate how the simpler of the latter two algorithms, namely the max-log-MAP algorithm, is used to decode an RSC code by way of an example.

EXAMPLE 7

Max-Log-MAP Decoding of Rate 3/8 Recursive Systematic Convolutional Code over AWGN Channel

In this example, we revisit the simple RSC code discussed previously at the tail end of Section 10.6 on convolutional codes.

For convenience of presentation, the two-state RSC encoder of Figure 10.17 is reproduced in Figure 10.23a. The message vector applied to the encoder is denoted by

$$\mathbf{m} = \{m_j\}_{j=0}^{3}$$

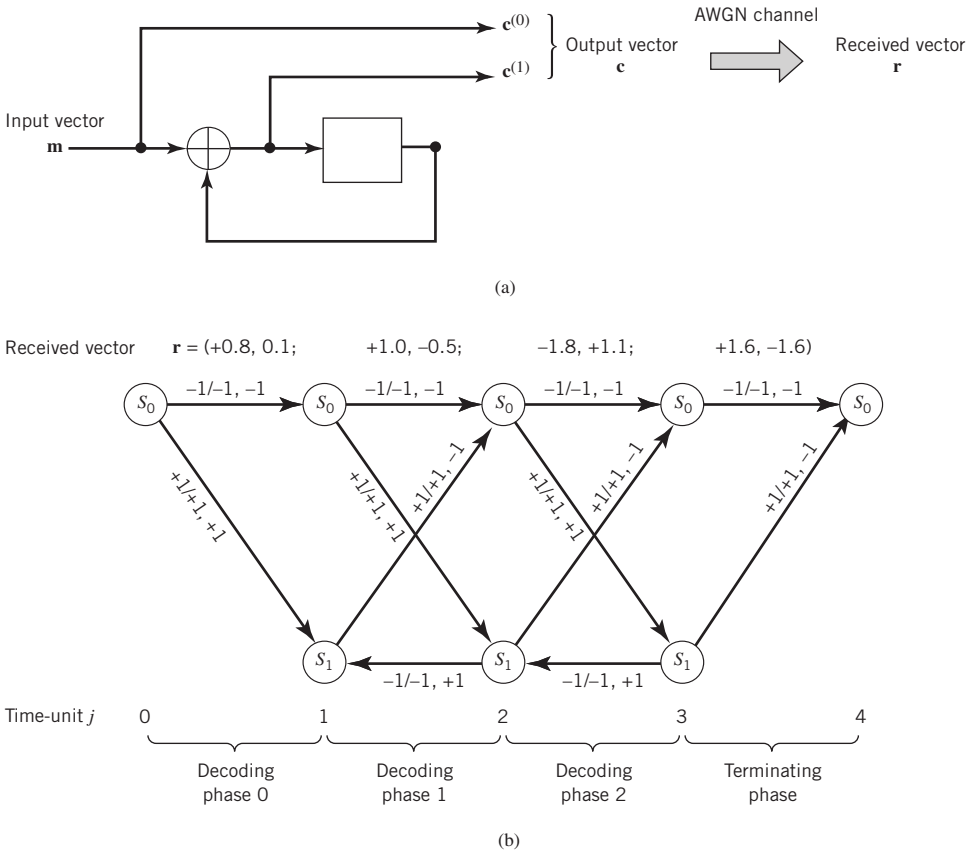


Figure 10.23 (a) Block diagram of rate-3/8, two-state recursive systematic convolutional (RSC) encoder. (b) Trellis graph of the encoder.

which produces the encoded output vector

$$\mathbf{c} = \left\{ c_j^{(0)}, c_j^{(1)} \right\}_{j=0}^{3}$$

Correspondingly, the received vector at the channel output is denoted by

$$\mathbf{r} = \left\{ r_j^{(0)}, r_j^{(1)} \right\}_{j=0}^{3}$$

The first three elements of the message vector $\mathbf{m}$, namely m_0, m_1, and m_2, are *message bits*. The last element, m_3, is a *termination bit*. With the encoded output vector, $\mathbf{c}$, consisting of eight bits, it follows that the code rate $r = 3/8$.

Figure 10.23b shows the trellis diagram of the RSC encoder. The underlying points covering the ways in which the branches of the trellis diagram have been labeled should be carefully noted:

1. The encoder is initialized to the *all-zero state* and, on termination of the encoding process, it returns to the all-zero state.
2. The encoder has a single memory unit; hence, there are only two states denoted by: S_0 represented by bit 0 and S_1 represented by bit 1.
3. Figure 10.24 illustrates the four different ways in which the state transitions take place:

$$\begin{aligned} S_0 &\text{ to } S_0{:}0/00 \\ S_0 &\text{ to } S_1{:}1/11 \\ S_1 &\text{ to } S_1{:}0/01 \\ S_1 &\text{ to } S_0{:}1/10 \end{aligned}$$

   where, in each case, the first bit on the right-hand side is an input bit and the following two bits (shown separately) are encoded bits. Since the encoder is systematic, it follows that the encoder input bit and the first encoded bit are exactly the same. The remaining second encoded bit is determined by the *modulo-2 recursion*:

$$m_j + b_{j-1} = b_j, \qquad j = 0, 1, 2, 3 \tag{10.112}$$

   where the initializing bit b_{-1} is 0. The two-bit code is defined by

$$\begin{aligned} \mathbf{c}_j &= (c_j^{(0)}, c_j^{(1)}) \\ &= (m_j, b_j), \qquad j = 0, 1, 2, 3 \end{aligned}$$

   We may thus use the notation $m_j/c_j^{(0)}, c_j^{(1)}$ to denote the branch labels. Hence, following this notation and the state transitions described in Figure 10.23b, we may identify the desired branch labels for the trellis diagram in terms of bits 0 and 1, respectively. More specifically, using the mapping rule: levels –1 and +1 for bits 0 and 1, respectively, we get the branch labels actually described in Figure 10.23b.

4. One last point is in order: owing to the use of feedback in the encoder, the lower branch leaving each state does not necessarily correspond to a bit 1 (level +1) and the upper branch does not necessarily correspond to a bit 0 (level –1).

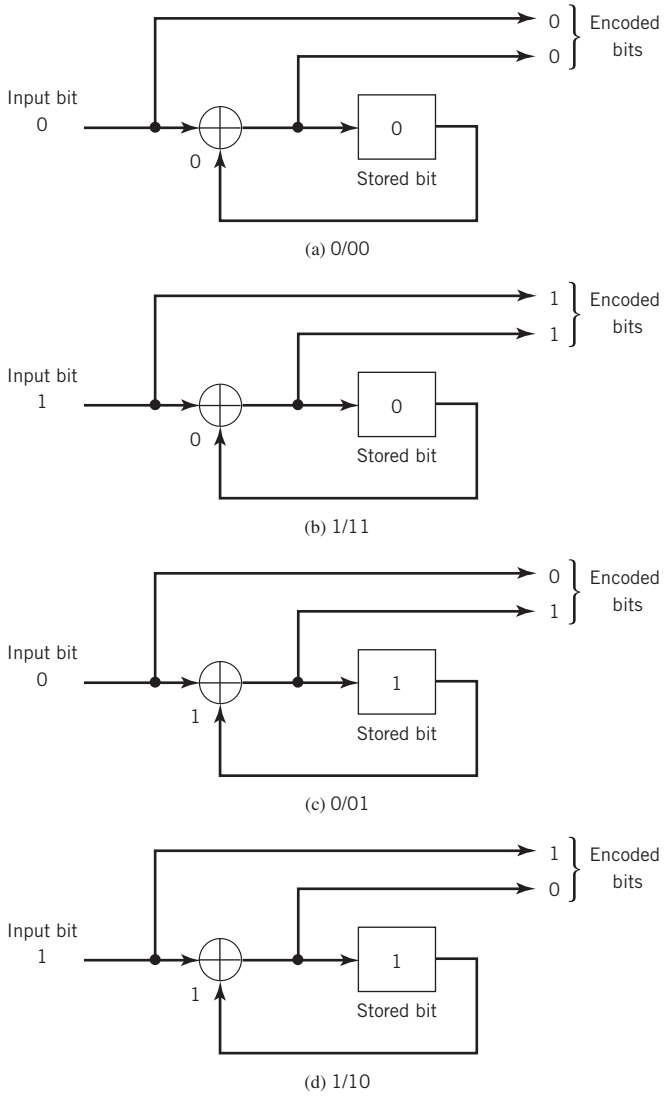


Figure 10.24 Illustration of the operations involved in the four possible state transitions.

To continue the background material for the example, we need to bring in a *mapper* that transforms the encoded signal into a form suitable for transmission over the AWGN channel. To this end, consider the simple example of binary PSK as the mapper. We may then express the SNR at the channel output (i.e., receiver input) as follows (see Problem 10.35):

$$\begin{aligned}(\text{SNR})_{\text{channel output}} &= \frac{E_s}{N_0} \\ &= r\left(\frac{E_b}{N_0}\right)\end{aligned} \tag{10.113}$$

where E_b is the signal energy per message bit applied to the encoder input, and r is the code rate of the convolutional encoder. Thus for the SNR = 1/2, that is –3.01 dB and $r = 3/8$, the required E_b/N_0 is 4/3.

In transmitting the coded vector $\mathbf{c}$ over the AWGN environment, it is assumed that the received signal vector, normalized with respect to $\sqrt{E_s}$, is given by

$$\mathbf{r} = (\underbrace{+0.8, 0.1}_{\mathbf{r}_0}; \underbrace{+1.0, -0.5}_{\mathbf{r}_1}; \underbrace{-1.8, 1.1}_{\mathbf{r}_2}; \underbrace{+1.6, -1.6}_{\mathbf{r}_3})$$

The received vector $\mathbf{r}$ is included at the top of the trellis diagram in Figure 10.23b.

We are now fully prepared to proceed with decoding the received vector $\mathbf{r}$ using the max-log-MAP algorithm described next, assuming the message bits are equally likely.

Computation of the Decoded Message Vector

To prepare the stage for this computation, we find it convenient to reproduce the following equations, starting with the formula for the log-domain transition metrics:

$$\gamma_j^*(s', s) = \frac{1}{2} L_c(\mathbf{r}_j^T \mathbf{c}_j), \qquad j = 0, 1, \ldots, K-1 \tag{10.114}$$

Then for the log-domain forward metrics:

$$\alpha_{j+1}^*(s) \approx \max_{s' \in \sigma_j^+} (\gamma_j^*(s', s) + \alpha_j^*(s')), \qquad j = 0, 1, \ldots, K-1 \tag{10.115}$$

Next, for the log-domain backward metrics:

$$\beta_j^*(s') = \max_{s \in \sigma_{j+1}} (\gamma_j^*(s', s) + \beta_{j+1}^*(s)) \tag{10.116}$$

And finally for computation of the a posteriori L-values:

$$L_p(m_j) = \max_{(s, s') \in \Sigma_j^+} \beta_{j+1}^*(s) + \gamma_j^*(s', s) + \alpha_j^*(s') - \max_{(s, s') \in \Sigma_j^-} \beta_{j+1}^*(s) + \gamma_j^*(s', s) + \alpha_j^*(s') \tag{10.117}$$

A Matlab code has been used to perform the computation, starting with the initial conditions for the forward and backward metrics, $\alpha_0(s)$ and $\beta_K(s')$, defined in (10.79) and (10.80), respectively. The results of the computation are summarized as follows:

1. *Log-domain transition metrics*

$$\text{Gamma 0}: \begin{cases} \gamma_0^*(S_0, S_0) = -0.9 \\ \gamma_0^*(S_0, S_1) = 0.9 \end{cases}$$

$$\text{Gamma 1}: \begin{cases} \gamma_1^*(S_0, S_1) = -0.5 \\ \gamma_1^*(S_1, S_0) = 1.5 \\ \gamma_1^*(S_0, S_1) = 0.5 \\ \gamma_1^*(S_1, S_1) = -1.5 \end{cases}$$

$$\text{Gamma 2}: \begin{cases} \gamma_2^*(S_0, S_0) = 0.7 \\ \gamma_2^*(S_1, S_0) = -2.9 \\ \gamma_2^*(S_0, S_1) = -0.7 \\ \gamma_2^*(S_1, S_1) = 2.9 \end{cases}$$

$$\text{Gamma 3}: \begin{cases} \gamma_3^*(S_0, S_0) = 0 \\ \gamma_3^*(S_1, S_0) = 3.2 \end{cases}$$

2. *Log-domain forward metrics*

$$\text{Alpha 0}: \begin{cases} \alpha_0^*(S_0) = 0 \\ \alpha_0^*(S_1) = 0 \end{cases}$$

$$\text{Alpha 1}: \begin{cases} \alpha_1^*(S_0) = -0.9 \\ \alpha_1^*(S_1) = 0.9 \end{cases}$$

$$\text{Alpha 2}: \begin{cases} \alpha_2^*(S_0) = 2.4 \\ \alpha_2^*(S_1) = -0.4 \end{cases}$$

3. *Log-domain backward metrics*

$$\beta_K: \begin{cases} \beta_K^*(S_0) = 0 \\ \beta_K^*(S_1) = 0 \end{cases}$$

$$\text{Beta 3}: \begin{cases} \beta_3^*(S_0) = 0 \\ \beta_3^*(S_1) = 3.2 \end{cases}$$

$$\text{Beta 2}: \begin{cases} \beta_2^*(S_0) = 2.5 \\ \beta_2^*(S_1) = 6.1 \end{cases}$$

$$\text{Beta 1}: \begin{cases} \beta_1^*(S_0) = 6.6 \\ \beta_1^*(S_1) = 4.6 \end{cases}$$

4. A posteriori L-values

$$\left.\begin{aligned} L_p(m_0) &= -0.2 \\ L_p(m_1) &= 0.2 \\ L_p(m_2) &= -0.8 \end{aligned}\right\} \tag{10.118}$$

5. *Final decision*

   Decoded version of the original message vector

$$\hat{\mathbf{m}} = [-1, 1, -1] \tag{10.119}$$

   In binary form, we may equivalently write

$$\hat{\mathbf{m}} = [0, 1, 0]$$

Two Final Remarks on Example 7

1. In arriving at the decoded output of (10.119) we made use of the termination bit, m_3. Although m_3 is *not* a message bit, the same procedure was used to calculate its a posteriori L-value. Lin and Costello (2004) showed that this kind of calculation is a necessary requirement in the iterative decoding of turbo codes. Specifically, with the turbo decoder consisting of two stages, "soft-output" a posteriori L-values are passed as a priori inputs to a second decoder.
2. In Example 7, we focused attention on the application of the max-log-MAP algorithm to decode the rate-3/8 RSC code produced by the two-state encoder of Figure 10.23a. The procedure described herein, embodying six steps, applies equally well to the log-MAP algorithm with no approximations. In Problem 10.34 at the end of the chapter, the objective is to show that the corresponding decoded output is (+1, +1, –1), which is different from that of Example 7. Naturally, in arriving at this new result, the calculations are somewhat more demanding but more accurate in the final decision-making.

10.11 New Generation of Probabilistic Compound Codes

Traditionally, the design of good codes has been tackled by constructing codes with a great deal of algebraic structure, for which there are feasible decoding schemes. Such an approach is exemplified by the linear block codes, cyclic codes, and convolutional codes discussed in preceding sections of this chapter. The difficulty with these traditional codes is that, in an effort to approach the theoretical limit for Shannon's channel capacity, we need to increase the codeword length of a linear block code or the constraint length of a convolutional code, which, in turn, causes the computational complexity of a maximum likelihood or maximum a posteriori decoder to increase exponentially. Ultimately, we reach a point where complexity of the decoder is so high that it becomes physically impractical.

Ironically enough, in his 1948 paper, Shannon showed that the "average" performance of a randomly chosen ensemble of codes results in an exponentially decreasing decoding

error with increasing block length. Unfortunately, as it was with his coding theorem, Shannon did not provide guidance on how to construct randomly chosen codes.

The Turbo Revolution Followed by LDPC Rediscovery

Interest in the use of randomly chosen codes was essentially dormant for a long time until the new idea of *turbo coding* was described by Berrou *et al*. (1993); that idea was based on two design initiatives:

1. The design of a good code, the construction of which is characterized by random-like properties.
2. The iterative design of a decoder that makes use of soft-output values by exploiting the maximum a posteriori decoding algorithm due to Bahl *et al*. (1974).

By exploiting these two ideas, it was experimentally demonstrated that turbo coding can approach the Shannon limit at a computational cost that would have been infeasible with traditional algebraic codes. Therefore, it can be said that the invention of turbo coding deserves to be ranked among the major technical achievements in the design of communication systems in the 20th century.

What is also remarkable is the fact that the discovery of turbo coding and iterative decoding flamed theoretical as well as practical interest in some prior work by Gallager (1962, 1963) on *LDPC codes*. These codes also possess the information-processing power to approach the Shannon limit in their own individual ways. The important point to note here is the fact that both turbo codes and LDPC codes are capable of approaching the Shannon limit at a similar level of computational complexity, provided that they both have a sufficiently long codeword. Specifically, turbo codes require a long turbo interleaver, whereas LDPC codes require a longer codeword at a given code rate (Hanzo, 2012).

We thus have two basic classes of *probabilistic compound coding techniques*: turbo codes and LDPC codes, which complement each other in the following sense:

> Turbo encoders are simple to design but the decoding algorithm can be demanding. In contrast, LDPC encoders are relatively complex but they are simple to decode.

With these introductory remarks, the stage is set for the study of turbo codes in Section 10.12, followed by LDPC codes in Section 10.14.

10.12 Turbo Codes

Turbo Encoder

As mentioned in the preceding section, the use of a good code with random-like properties is basic to turbo coding. In the first successful implementation of turbo codes[11], Berrou *et al*. achieved this design objective by using *concatenated codes*. The original idea of concatenated codes was conceived by Forney (1966). To be more specific, concatenated codes can be of two types: *parallel* or *serial*. The type of concatenated codes used by Berrou *et al*. was of the parallel type, which is discussed in this section. Discussion of the serial type of concatenated codes will be taken up in Section 10.16.

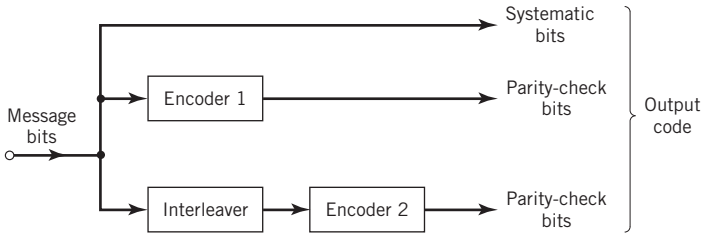


Figure 10.25 Block diagram of turbo encoder of the parallel type.

Figure 10.25 depicts the most basic form of a turbo code generator that consists of two constituent systematic encoders, which are concatenated by means of an interleaver.

The *interleaver* is an input–output mapping device that permutes the ordering of a sequence of symbols from a fixed alphabet in a completely deterministic manner; that is, it takes the symbols at the input and produces identical symbols at the output but in a different temporal order. Turbo codes use a *pseudo-random interleaver*, which operates only on the systematic (i.e., message) bits. (Interleavers are discussed in Appendix F.) The size of the interleaver used in turbo codes is typically very large, on the order of several thousand bits.

There are two reasons for the use of an interleaver in a turbo code:

1. The interleaver ties together errors that are easily made in one half of the turbo code to errors that are exceptionally unlikely to occur in the other half; this is indeed one reason why the turbo code performs better than a traditional code.
2. The interleaver provides robust performance with respect to mismatched decoding, a problem that arises when the channel statistics are not known or have been incorrectly specified.

Ordinarily, but not necessarily, the same code is used for both constituent encoders in Figure 10.25. The constituent codes recommended for turbo codes are *short constraint-length RSC codes*. The reason for making the convolutional codes recursive (i.e., feeding one or more of the tap outputs in the shift register back to the input) is to make the internal state of the shift register depend on past outputs. This affects the behavior of the error patterns, with the result that a better performance of the overall coding strategy is attained.

EXAMPLE 8 **Two-State Turbo Encoder**

Figure 10.26 shows the block diagram of a specific turbo encoder using an identical pair of two-state RSC constituent encoders. The generator matrix of each constituent encoder is given by

$$\mathbf{G}(D) = \left(1, \frac{1}{1+D}\right)$$

The input sequence of bits has length $K = 4$, made up of three message bits and one termination bit. (This RSC encoder was discussed previously in Section 10.9.) The input vector is given by

$$\mathbf{m} = (m_0, m_1, m_2, m_3)$$

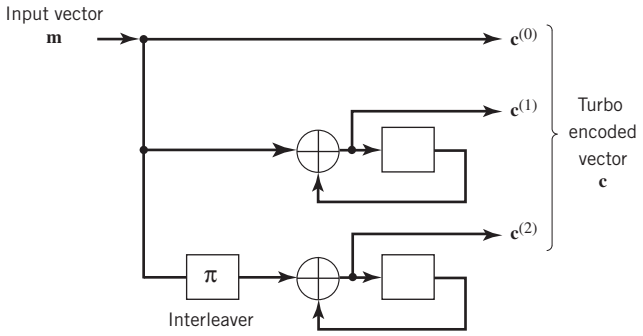


Figure 10.26 Two-state turbo encoder for Example 8.

The parity-check vector produced by the first constituent encoder is given by

$$\mathbf{b}^{(1)} = (b_0^{(1)}, b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

Similarly, the parity-check vector produced by the second constituent encoder is given by

$$\mathbf{b}^{(2)} = (b_0^{(2)}, b_1^{(2)}, b_2^{(2)}, b_3^{(2)})$$

The transmitted code vector is therefore defined by

$$\mathbf{c} = (\mathbf{c}^{(0)}, \mathbf{c}^{(1)}, \mathbf{c}^{(2)})$$

With the convolutional code being systematic, we thus have

$$\mathbf{c}^{(0)} = \mathbf{m}$$

As for the remaining two sub-vectors constituting the code vector $\mathbf{c}$, they are defined by

$$\mathbf{c}^{(1)} = \mathbf{b}^{(1)}$$

and

$$\mathbf{c}^{(2)} = \mathbf{b}^{(2)}$$

The transmitted code vector $\mathbf{c}$ is therefore made up of 12 bits. However, recalling that the termination bit m_3 is not a message bit, it follows that the code rate of the turbo code described in Figure 10.26 is

$$r = \frac{3}{12} = \frac{1}{4}$$

One last point is in order: with each RSC encoder having two states, the interleaver has a two-by-two (row–column) structure. Note also that the interleaver in Figure 10.26 is denoted by the symbol π, which is a common usage; this practice is adopted throughout the book.

In Figure 10.25, the input data stream is applied directly to encoder 1 and the pseudo-randomly reordered version of the same data stream is applied to encoder 2. The systematic bits (i.e., original message bits) and the two sets of parity-check bits generated

by the two encoders constitute the output of the turbo encoder. Although the constituent codes are convolutional, in reality, turbo codes are block codes with the block size being determined by the periodic size of the interleaver. Moreover, both RSC encoders in Figure 10.25 are linear. We may therefore describe turbo codes generally as *linear block codes*.

The block nature of the turbo code raises a practical issue:

How do we know the beginning and the end of a codeword?

The common practice is to initialize the encoder to the *all-zero state* and then encode the data. After encoding a certain number of data bits, a number of tail bits are added so as to make the encoder return to the all-zero state at the end of each block; thereafter, the cycle is repeated. The *termination* approaches of turbo codes include the following:

- A simple approach is to terminate the first RSC code in the encoder and leave the second one undetermined. A drawback of this approach is that the bits at the end of the block due to the second RSC code are more vulnerable to noise than the other bits. Experimental work has shown that turbo codes exhibit a leveling off in performance as the SNR increases. This behavior is not like an error floor; rather, it has the appearance of an error floor compared with the steep drop in error performance at low SNR. The *error floor* is affected by a number of factors, the dominant one of which is the choice of interleaver.
- A more refined approach is to terminate both constituent codes in the encoder in a symmetric manner. Through the combined use of a good interleaver and dual termination, the error floor can be reduced by an order of magnitude compared to the simple termination approach.

In the original version of the turbo encoder described in Berrou *et al.* (1993), the parity-check bits generated by the two encoders in Figure 10.25 were punctured prior to data transmission over the channel to maintain the rate at 1/2. A *punctured code* is constructed by deleting certain parity-check bits, thereby increasing the data rate; the message bits in the puncturing process are of course unaffected. Basically, puncturing is the inverse of extending a code. It should, however, be emphasized that the use of a puncture map is not a necessary requirement for the generation of turbo codes.

As mentioned previously, the encoding scheme of Figure 10.25 is of the *parallel concatenation* type, the novelty of which is twofold:

- the use of *RSC codes* and
- the insertion of a pseudo-random interleaver between the two encoders.

The net result of parallel concatenation is a turbo code that appears essentially *random* to the channel by virtue of the pseudo-random interleaver, yet it possesses sufficient structure for the decoding to be physically realizable. Coding theory asserts that a code chosen at random is capable of approaching Shannon's channel capacity, provided that the block size is sufficiently large. This is indeed the reason behind the impressive performance of turbo codes, as discussed next.

Performance of Turbo Codes

Figure 10.27 shows the error performance of a 1/2-rate turbo code with a large block size for binary data transmission over an AWGN channel.[12] The code uses an interleaver of size 65,536 bits and a MAP decoder.

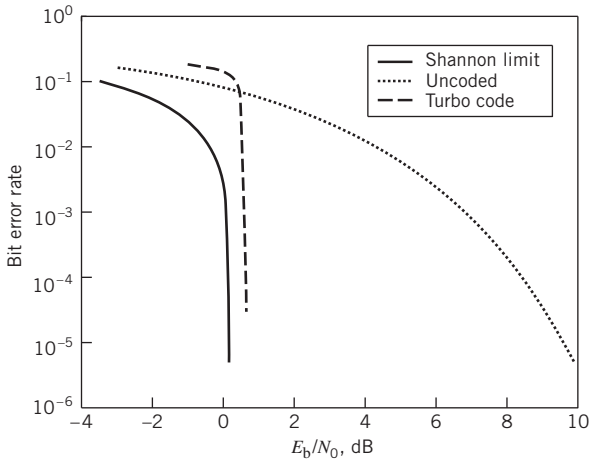


Figure 10.27 Noise performance of 1/2 rate, turbo code and uncoded transmission for AWGN channel; the figure also includes Shannon's theoretical limit on channel capacity for code rate $r = 1/2$.

For the purpose of comparison, Figure 10.27 also includes two other curves for the same AWGN channel:

- uncoded transmission (i.e., code rate $r = 1$);
- Shannon's theoretical limit for code rate 1/2, which follows from Figure 5.18b.

From Figure 10.27, we may draw two important conclusions:

1. Although the BER for the turbo-coded transmission is significantly higher than that for uncoded transmission at low E_b/N_0, the BER for the turbo-coded transmission drops very rapidly once a critical value of E_b/N_0 has been reached.
2. At a BER of 10^{-5}, the turbo code is less than 0.5 dB from Shannon's theoretical limit.

Note, however, attaining this highly impressive performance requires that the size of the interleaver or, equivalently, the block length of the turbo code be large. Also, the large number of iterations needed to improve performance increases the decoder latency. This drawback is due to the fact that the digital processing of information does not lend itself readily to the application of feedback, which is a distinctive feature of the turbo decoder.

Extrinsic Information

Before proceeding to describe the operation of the turbo decoder, we find it desirable to introduce the notion of extrinsic information. The most convenient representation for this new concept is in terms of the log-likelihood ratio, in which case extrinsic information is computed as the difference between two a posteriori L-values as depicted in Figure 10.28. Formally, *extrinsic information*, generated by a decoding stage for a set of systematic (message) bits, is defined as follows:

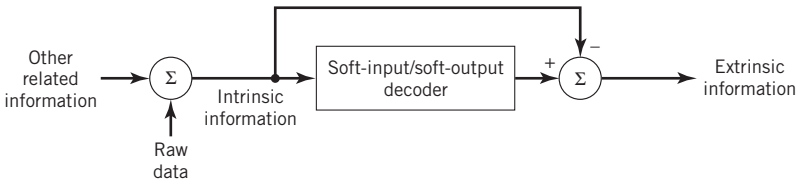


Figure 10.28 Block diagram for illustrating the concept of extrinsic information.

> Extrinsic information is the difference between the log-likelihood ratio computed at the output of a decoding stage and the intrinsic information represented by the log-likelihood ratio applied to the input of that decoding stage.

In effect, extrinsic information is the *incremental information* gained by exploiting the dependencies that exist between a message bit of interest and incoming raw data bits processed by the decoder. Extrinsic information plays a key role in the iterative decoding process, as discussed next.

Turbo Decoder

Figure 10.29a shows the block diagram of the two-stage *turbo decoder*. Using a MAP decoding algorithm discussed in Section 10.9, the decoder operates on noisy versions of the systematic bits and the two sets of parity-check bits in two decoding stages to produce an estimate of the original message bits.

A distinctive feature of the turbo decoder that is immediately apparent from the block diagram of Figure 10.29a is the use of *feedback*, manifesting itself in producing extrinsic information from one decoder to the next in an iterative manner. In a way, this decoding process is analogous to the feedback of exhaust gases experienced in a turbo-charged engine; indeed, turbo codes derive their name from this analogy. In other words, the term "turbo" in turbo codes has more to do with the decoding rather than the encoding process.

In operational terms, the turbo encoder in Figure 10.29a operates on *noisy* versions of the following inputs, obtained by demultiplexing the channel output, $\mathbf{r}_j$,

- systematic (i.e., message) bits, denoted by $\mathbf{r}_j^{(0)}$;
- parity-check bits corresponding to encoder 1 in Figure 10.25, denoted by $\mathbf{r}_j^{(1)}$;
- parity-check bits corresponding to encoder 2 in Figure 10.25, denoted by $\mathbf{r}_j^{(2)}$.

The net result of the decoding algorithm, given the received vector $\mathbf{r}_j$, is an estimate of the original message vector, namely $\hat{\mathbf{m}}$, which is delivered at the decoder output to the user.

Another important point to note in the turbo decoder of Figure 10.29a is the way in which the interleaver and de-interleaver are positioned inside the feedback loop. Bearing in mind the fact that the definition of extrinsic information requires the use of intrinsic information, we see that decoder 1 operates on three inputs:

- the noisy systematic (i.e., original message) bits,
- the noisy parity-check bits due to encoder 1, and
- de-interleaved extrinsic information computed by decoder 2.

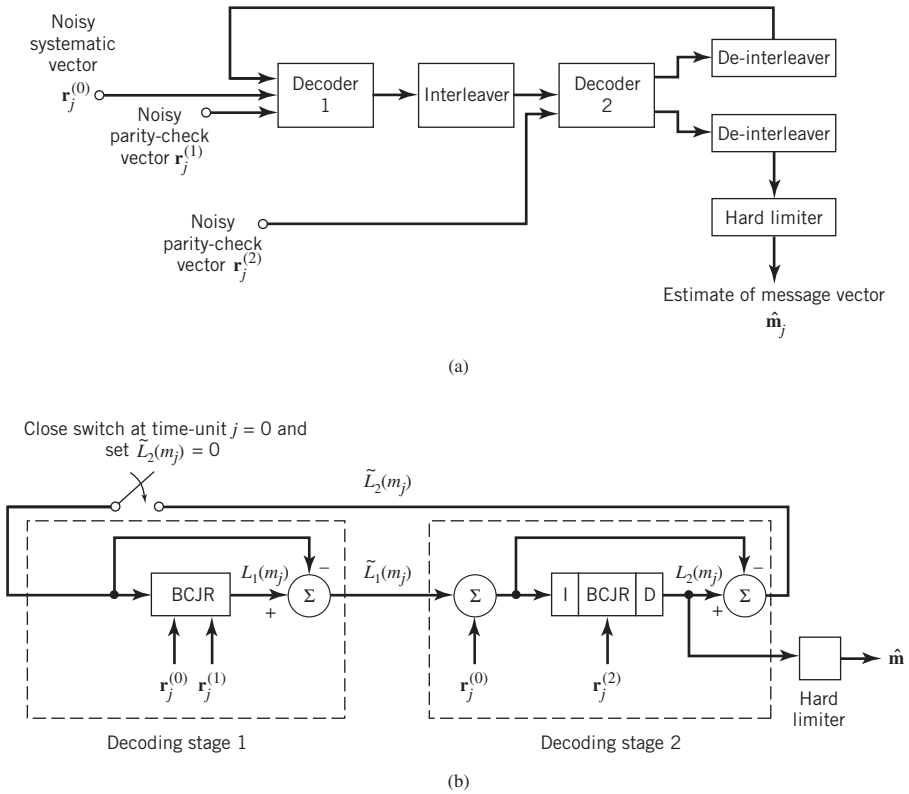


Figure 10.29 (a) Block diagram of turbo decoder. (b) Extrinsic form of turbo decoder, where I stands for interleaver, D for deinterleaver, and BCJR for BCJR for BCJR algorithm for log-MAP decoding.

In a complementary manner, decoder 2 operates on two inputs of its own:

- the noisy parity-check bits due to encoder 2 and
- the interleaved version of the extrinsic information computed by decoder 1.

For this iterative exchange of information between the two decoders inside the feedback loop to *continuously reinforce each other*, the de-interleaver and interleaver would have to separate the two decoders in the manner depicted in Figure 10.29a. Moreover, the structure of the decoder in the receiver is configured to be consistent with the structure of the encoder in the transmitter.

Mathematical Feedback Analysis

To put the two-state turbo decoding process just described on a mathematical basis, we structure the flow of information around the feedback loop as depicted in Figure 10.29a. For the sake of simplicity without loss of generality, we assume the use of a code rate $r = 1/3$ parallel concatenated convolutional code without puncturing. At time-unit j, let

$\mathbf{r}_j^{(0)}$ denote the noisy vector of systematic bits,

$\mathbf{r}_j^{(1)}$ denote the noisy vector of parity-check bits produced by encoder 1, and
$\mathbf{r}_j^{(2)}$ denote the noisy vector of parity-check bits produced by encoder 2.

The notations described herein are consistent with those adopted in the encoder of Figure 10.25. Moreover, it is assumed that all three vectors, $\mathbf{r}_j^{(0)}$, $\mathbf{r}_j^{(1)}$, $\mathbf{r}_j^{(2)}$, are of dimensionality K.

Proceeding with the analysis, decoder 1 in Figure 10.29b uses the BCJR decoding algorithm to produce a "soft estimate" of symmetric bit m_j by computing the a posteriori L-values for decoder 1, namely

$$L_1(m_j) = \ln\left(\frac{\mathbb{P}(m_j = +1|\mathbf{r}_j^{(0)}, \mathbf{r}_j^{(1)}, \tilde{L}_2(\mathbf{m}))}{\mathbb{P}(m_j = -1|\mathbf{r}_j^{(0)}, \mathbf{r}_j^{(1)}, \tilde{L}_2(\mathbf{m}))}\right), \qquad j = 0, 1, \ldots, K-1 \tag{10.120}$$

where $\tilde{L}_2(\mathbf{m})$ denotes the extrinsic information about the message vector $\mathbf{m}$ that is computed by decoder 2. Note also that in (10.120) we have used the usual mapping: +1 for bit 1 and –1 for bit 0. Assuming that the L message bits are statistically independent, the overall extrinsic information computed by decoder 1 is given by the summation:

$$L_1(\mathbf{m}) = \sum_{j=0}^{K-1} L_1(m_j) \tag{10.121}$$

Accordingly, the extrinsic information about the message vector $\mathbf{m}$ computed by decoder 1 is given by the difference

$$\tilde{L}_1(\mathbf{m}) = L_1(\mathbf{m}) - \tilde{L}_2(\mathbf{m}) \tag{10.122}$$

where $\tilde{L}_2(\mathbf{m})$ is to be defined.

Before proceeding to use (10.122) in the second decoding stage, the extrinsic information $\tilde{L}_1(\mathbf{m})$ is reordered (i.e., de-interleaved) to compensate for the pseudo-random interleaving introduced originally in the turbo encoder in the manner indicated in both Figure 10.29b. In addition to $\tilde{L}_1(\mathbf{m})$, the input applied to decoder 2 also includes the vector of noisy parity-check bits $\mathbf{r}^{(2)}$. Accordingly, by using the BCJR algorithm, decoder 2 produces a more refined soft estimate of the message vector $\mathbf{m}$. Next, as indicated in Figure 10.29b, this refined estimate of the message vector is re-interleaved to compute the a posteriori L-values for decoder 2, namely

$$\tilde{L}_2(\mathbf{m}) = \sum_{j=0}^{K-1} L_2(m_j) \tag{10.123}$$

where

$$\tilde{L}_2(m_j) = \ln\left(\frac{\mathbb{P}(m_j = +1|\mathbf{r}_j^{(2)}, \tilde{\mathbf{L}}_1(\mathbf{m}))}{\mathbb{P}(m_j = -1|\mathbf{r}_j^{(2)}, \tilde{\mathbf{L}}_1(\mathbf{m}))}\right), \qquad j = 0, 1, \ldots, K-1 \tag{10.124}$$

Accordingly, the extrinsic information $\tilde{L}_2(\mathbf{m})$ fed back to the input of decoder 1 is given by

$$\tilde{L}_2(\mathbf{m}) = L_2(\mathbf{m}) - \tilde{L}_1(\mathbf{m}) \tag{10.125}$$

and with it the feedback loop, embodying constituent decoders 1 and 2, is closed.

As indicated in Figure 10.29b, the decoding process is *initiated* by setting the a posteriori extrinsic L-value

$$\tilde{L}_2(m_j) = 0, \qquad \text{for } j = 0 \tag{10.126}$$

The decoding process is *stopped* when it reaches a point at which no further improvement in performance is attainable. At this point, an estimate of the message vector $\mathbf{m}$ is computed by hard-limiting the a priori L-value at the output of decoder 2, yielding

$$\hat{\mathbf{m}} = \text{sgn}(L_2(\mathbf{m})) \tag{10.127}$$

To conclude the discussion on turbo decoding, two other points are noteworthy:

1. Although the noisy vector of systematic bits $\mathbf{r}^{(0)}$ is applied only to decoder 1, its influence on decoder 2 manifests itself *indirectly* through the a posteriori extrinsic L-value, $\tilde{L}_1(\mathbf{m})$, computed by decoder 1.
2. Equations (10.122) and (10.125) assume that the a posteriori extrinsic L-values, $\tilde{L}_1(\mathbf{m})$ and $\tilde{L}_2(\mathbf{m})$, passed between decoders 1 and 2, are statistically independent of the message vector $\mathbf{m}$. In reality, however, this condition applies only to the first iteration of the decoding process. Thereafter, the extrinsic information becomes less helpful in realizing successively more reliable estimates of the message vector $\mathbf{m}$.

EXAMPLE 9

UMTS Codec Using Binary PSK Modulation[13]

In this example, we study the *Universal Mobile Telecommunications Systems* (UMTS) standard's codec. To simplify the study, binary PSK modulation is used for data transmission over an AWGN channel. The basic RSC encoder of the UMTS turbo codes is as follows:

code-rate	$r = 1/3$
constraint length	$\nu = 4$
memory length	$m = 3$

The UMTS Turbo Encoder

Figure 10.30a shows the block diagram of the UMTS turbo encoder, which consists of two identical concatenated RSC encoders, operating in parallel with an interleaver separating them. To be specific:

- Each encoder is made up of a *linear feedback shift register* (LFSR) whose number of flip-flops $m = 3$; in each LFSR, therefore, we have a finite-state machine with
$$2^m = 2^3 = 8 \text{ states}$$
- The encoding process is initialized by setting each LFSR to the all-zero state.
- To activate the encoding process, the two switches in Figure 10.30a are closed, thereby applying the message vector $\mathbf{m}$ to the top RSC encoder and applying the interleaved version of $\mathbf{m}$, namely $\mathbf{n}$, to the bottom RSC encoder. The length of $\mathbf{m}$ is denoted by K.
- Each RSC constituent encoder produces a sequence of parity-check bits, the length of which is $K + m$.
- Once the encoding process is completed, a set of m bits is appended to each block of encoded bits, so as to force each LFSR back to the initial all-zero state.

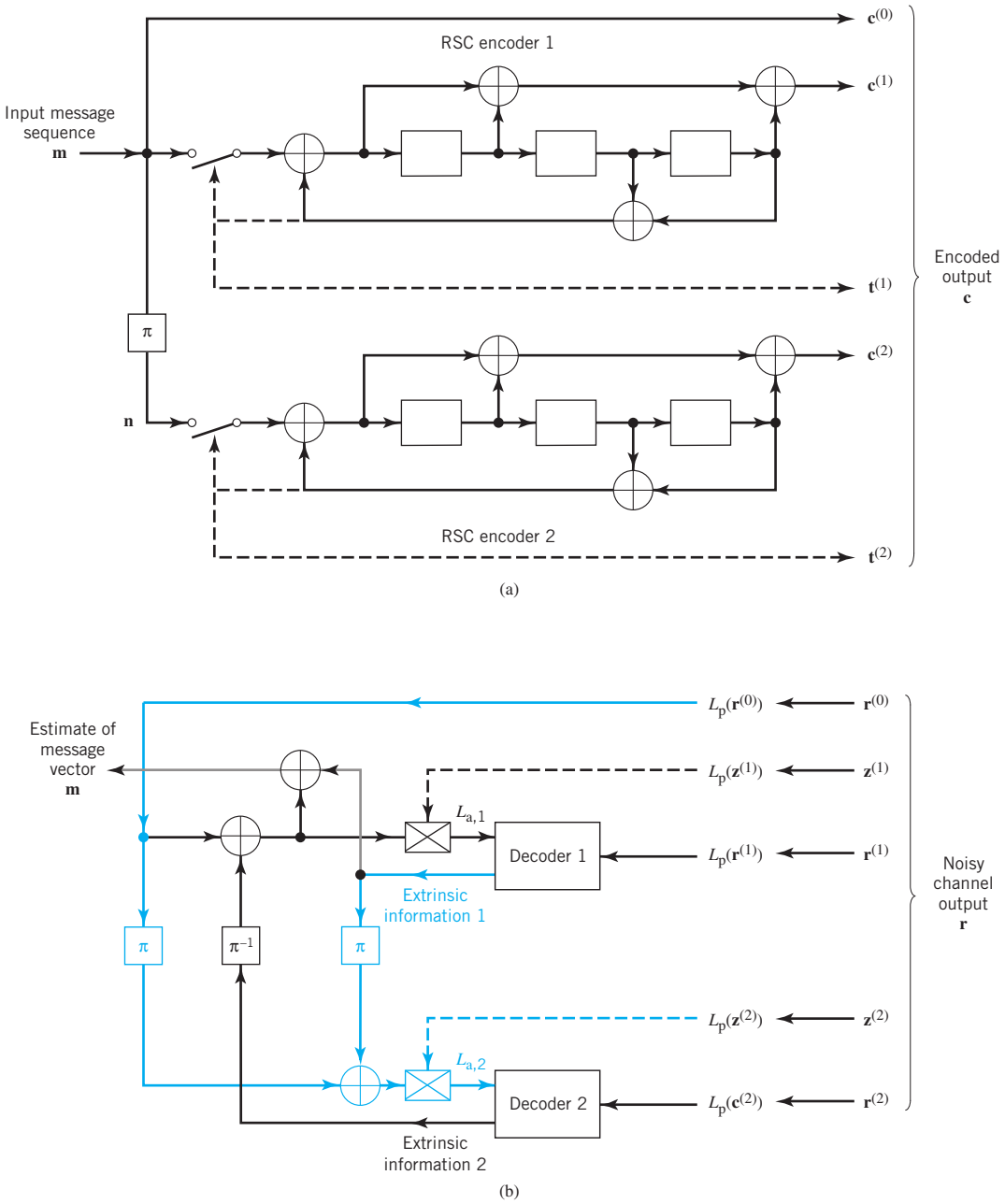


Figure 10.30 Block diagram of UMTS codec. (a) Encoder, (b) Decoder. *Notes:* 1. The received vectors $\{\mathbf{r}^{(0)}, \mathbf{z}^{(1)}, \mathbf{r}^{(1)}, \mathbf{z}^{(2)}, \mathbf{r}^{(2)}\}$ correspond to the transmitted vectors $\{\mathbf{c}^{(0)}, \mathbf{t}^{(0)}, \mathbf{c}^{(1)}, \mathbf{t}^{(2)}, \mathbf{c}^{(2)}\}$. 2. The block labeled π : interleaver. The block labeled π^{-1} : de-interleaver.

From this description, it is apparent that the *overall code-rate* of the turbo code is lower than the UMTS code-rate, namely 1/3, as shown by

$$r_{\text{overall}} = \frac{K}{3K + 4m}$$

Note that if we set the memory length $m = 0$, the code rate r_{overall} is increased again to 1/3.

On the basis of this description, each block of the *multiplexed output* of the turbo encoder is composed as follows:

$\mathbf{c}^{(0)}$	vector of systematic bits (i.e., message bits), followed by
$\mathbf{c}^{(1)}$ and $\mathbf{c}^{(2)}$	pair of vectors, representing the parity-check bits produced by the top and bottom RSC encoders, respectively, then followed by
$\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$	pair of vectors, representing encoder termination-tail bits for forcing the top and bottom RSC constituent encoders back to all-zero state, respectively.

In the UMTS standard, the block length of the turbo code lies in the range [40, 5114].

The UMTS Turbo Decoder

Figure 10.30b shows a block diagram of the UMTS decoder. Specifically, proceeding from top to bottom on the right-hand side of the figure, we have five sequences of a posteriori L-values computed in the receiver, namely $L_p(\mathbf{c}^{(0)})$, $L_p(\mathbf{t}^{(1)})$, $L_p(\mathbf{c}^{(1)})$, $L_p(\mathbf{t}^{(2)})$, and $L_p(\mathbf{c}^{(2)})$; these L-values correspond to the encoded sequences $\mathbf{c}^{(0)}$, $\mathbf{t}^{(1)}$, $\mathbf{c}^{(1)}$, $\mathbf{t}^{(2)}$, and $\mathbf{c}^{(2)}$, respectively.

Considering, first, how decoder 1 operates in the receiver, we find from Figure 10.30b that it receives two input sequences of L-values, the first one of which, namely the a posteriori L-value $L_p(\mathbf{c}^{(1)})$, comes directly from the channel. The other input, the a priori L-value denoted by $L_{a,1}$, is made up of three components:

1. The a posteriori L-value, $L_p(\mathbf{c}^{(0)})$, which accounts for the received systematic bits, $\mathbf{c}^{(0)}$.
2. The reordered version of the extrinsic information produced by decoder 2, resulting from the de-interleaver π^{-1}.
3. The a posteriori L-value, $L_p(\mathbf{t}^{(1)})$ attributed to the systematic vector of termination bits, $\mathbf{t}^{(1)}$, which is appended to the sum of components 1 and 2 to complete $L_{a,1}$.

In a corresponding but slightly different way, decoder 2 receives two input sequences of L-values, the first one of which, namely the a posteriori L-value $L_p(\mathbf{c}^{(2)})$, comes directly from the channel. The other input, a priori L-value $L_{a,2}$, is also made up of three components:

1. The reordered version of a posteriori L-value, $L_p(\mathbf{c}^{(0)})$ is due to the received vector of systematic bits, $\mathbf{c}^{(0)}$, where the reordering is produced by the interleaver to the left of the de-interleaver π^{-1}.
2. The reordered version of extrinsic information is produced by decoder 1, where the reordering is performed by the second interleaver, π, to the right of the de-interleaver π^{-1}.
3. The a posteriori L-value, $L_p(\mathbf{t}^{(2)})$ is attributed to the systematic vector of termination bits, $\mathbf{t}^{(2)}$; this time, however, $L_p(\mathbf{t}^{(2)})$ is removed before it is interleaved and passed to decoder 2.

Simulation Results

In Figure 10.31, we have plotted the BER chart for the iterative decoding process, using the turbo codec of Figure 10.30. The results were obtained for the case of 5000 systematic bits, as follows:

- For a prescribed E_b/N_0 ratio, the bit errors were averaged over 50 Monte Carlo runs.
- Each point in the BER chart was the result of 100 bits per point-count in the decoding process.
- The computations were repeated for different values of E_b/N_0.

The remarkable points to observe from Figure 10.31 are summarized here:

1. In the course of just four iterations, the BER of the UMTS decoder drops to 10^{-14} at an SNR = 3 dB, which, for all practical purposes, is zero.
2. The steepness of the BER plot on iteration 4 is showing signs of the turbo cliff, but is not there yet. Unfortunately, to get there would require a great deal more computation.[14] (The turbo cliff is illustrated in Figure 10.32 in the next section).

Rudimentary Comparison of Viterbi and MAP Algorithms

For a rather rudimentary but plausible approach, to address the issue of computational complexity in a fair-minded way, consider a convolutional code that has $m = 6$ states and, therefore, requires

$$2^6 = 64$$

ACS operations for Viterbi decoding.

To match this computational complexity, using the turbo decoder of Figure 10.29b with 16 ACS operations, we need the following number of decoding iterations:

$$\frac{64}{16} = 4$$

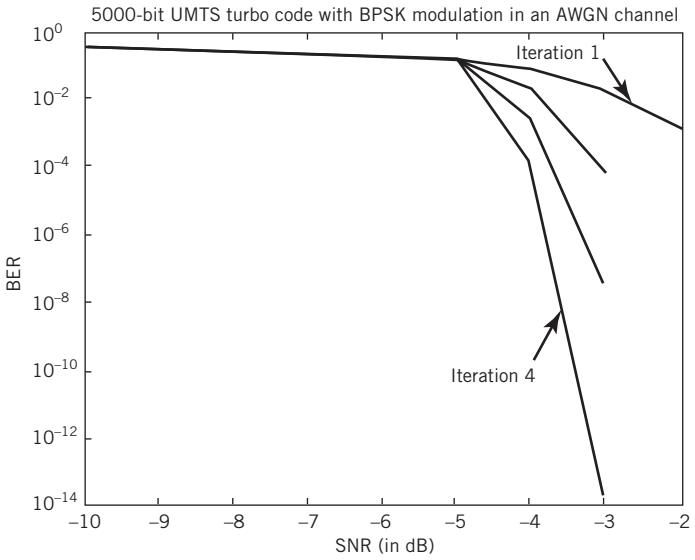


Figure 10.31
The bit error rate (BER) diagram for the UMTS-turbo decoder, using 5000 systematic bits and –3 dB SNR.

Correspondingly, Figure 10.31 plots the BER chart for the turbo decoder for the sequences of decoding iterations: 1, 2, 3, and 4. Of special interest is the BER chart for four iterations, for which we find that the turbo decoder with BER $\approx 10^{-14}$ outperforms the Viterbi decoder significantly for the same computational complexity, namely a total of 64 ACS operations.

10.13 EXIT Charts

In an idealized BER chart exemplified by that in Figure 10.32, we may identify three distinct regions, described as follows:

a. *Low BER region*, for which the E_b/N_0 ratio is correspondingly low.

b. *Waterfall region*, also referred to as the *turbo cliff* in the turbo coding literature, which is characterized by a persistent reduction in BER over the span of a small fraction of dB in SNR.

c. *BER floor region*, where a rather small improvement in decoding performance is achieved for medium to large values of SNRs.

As informative as the BER chart of Figure 10.32 is, from a practical perspective it has a serious drawback. Simply put, the BER chart lacks insight into the underlying dynamics (i.e., convergence behavior) of iterative decoding algorithms, particularly around the turbo-cliff region. Furthermore, since the BER occurs at low BERs, excessive simulation runs are required.

The question is: how do we overcome this serious drawback of the BER chart? The answer lies in using the *extrinsic information chart*, or EXIT chart for short, which was formally introduced by ten Brink (2001).

The EXIT chart is insightful because it provides a graphical procedure for visualizing the underlying dynamics of the turbo decoding process for a prescribed E_b/N_0. Moreover,

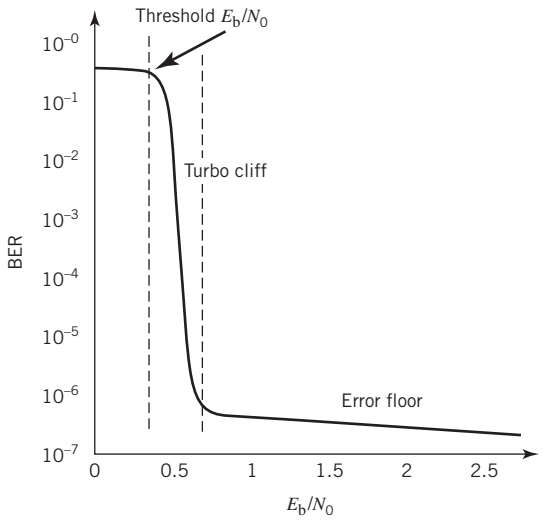


Figure 10.32 Idealized BER chart for turbo decoding.

the procedure provides a tool for the design of turbo codes characterized by good performance in the turbo-cliff region. In any event, development of the EXIT chart exploits the idea of mutual information in Shannon's information theory, which was discussed previously in Chapter 5.

Development of the EXIT Chart

Consider a constituent decoder in the turbo decoder of Figure 10.29b, which is, for convenience of presentation, labeled decoder 1; the other constituent decoder is labeled decoder 2. Let $I_1(m_j; L_a(m_j))$ denote the mutual information between a transmitted message bit m_j and the a priori L-value $L_a(m_j)$ for a prescribed E_b/N_0. Correspondingly, let $L_p(m_j)$ denote the a posteriori L-value of message bit m_j and let $I_2(m_j; L_p(m_j))$ denote the mutual information between m_j and $L_p(m_j)$ for the same E_b/N_0. Then, with $I_2(m_j; L_p(m_j))$ viewed as a function f $I_1(m_j; L_a(m_j))$, we may express the *extrinsic information transfer characteristic* of constituent decoder 1 for some operator $T(\cdot)$ and the prescribed E_b/N_0 as follows:

$$I_2(m_j; L_p(m_j)) = T(I_1(m_j; L_a(m_j)) \tag{10.128}$$

In the continuum, it is shown that both mutual informations, I_1 and I_2, lie within the range [0,1]. Thus, a plot of $I_2(m_j; L_p(m_j))$ versus $I_1(m_j; L_a(m_j))$, depicted in Figure 10.33a, displays graphically the extrinsic information transfer characteristic of the constituent decoder 1.

Since the two constituent decoders are similar and they are connected together sequentially inside a closed feedback loop, it follows that the extrinsic information transfer characteristic of constituent decoder 2 is the *mirror image* of the curve in Figure 10.33a with respect to the straight line $I_1 = I_2$, as shown in Figure 10.33b. With this relationship in mind, we may go on to put the transfer characteristic curves of the two constituent decoders side by side, but keeping the same horizontal and vertical axes of Figure 10.33a. We thus get the composite picture depicted in Figure 10.33c. In effect, this latter figure represents the input–output extrinsic transfer characteristic of the two constituent decoders working together in a turbo-decoding algorithm for the prescribed E_b/N_0.

To elaborate on the practical utility of Figure 10.33a, suppose that the iterative turbo-decoding algorithm begins with $I_1^{(1)} = 0$, representing the initial condition of constituent decoder 1 for the first iteration in the decoding process. Then, in proceeding forward, we keep the following two points in mind:

- First, the a posteriori L-value of constituent decoder 1 becomes the a priori L-value of constituent decoder 2, and similarly when these two decoders are interchanged, as we proceed from one iteration to the next.
- Second, the message bits $m_1, m_2, m_3, \ldots$ occur on consecutive iterations.

Hence, we will experience the following sequence of extrinsic information transfers between the two constituent decoders from one message bit to the next one for some prescribed E_b/N_0:

Initial condition: $I_1^{(1)}(m_1) = 0$.

Iteration 1: message bit, m_1

Decoder 1: $I_1^{(1)}(m_1)$ defines $I_2^{(1)}(m_1)$.

Decoder 2: $I_2^{(1)}(m_1)$ initiates $I_1^{(2)}(m_2)$ for iteration 2.

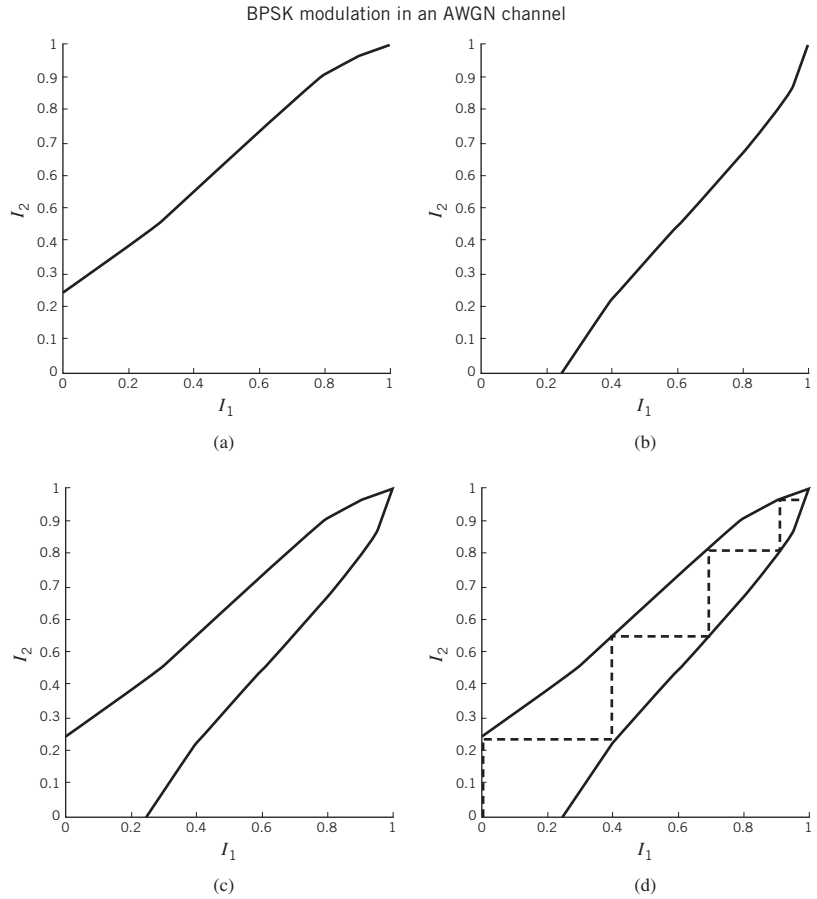


Figure 10.33 (a) Extrinsic information transfer characteristic of decoder 1; (b) Extrinsic information transfer characteristic of decoder 2; (c) Input-output extrinsic transfer characteristic of the two constituent decoders working together; (d) EXIT chart, including the staircase (shown dashed) embracing the extrinsic information transfer characteristics of both constituent decoders.

Iteration 2: message bit, m_2

Decoder 1: $I_1^{(2)}(m_2)$ defines $I_2^{(2)}(m_2)$.

Decoder 2: $I_2^{(2)}(m_2)$ initiates $I_1^{(3)}(m_3)$ for iteration 3.

Iteration 3: message bit, m_3

Decoder 1: $I_1^{(3)}(m_3)$ defines $I_2^{(3)}(m_3)$.

Decoder 2: $I_2^{(3)}(m_3)$ initiates $I_1^{(4)}(m_4)$ for iteration 4.

Iteration 4: message bit, m_4

Decoder 1: $I_1^{(4)}(m_4)$ defines $I_2^{(4)}(m_4)$.

Decoder 2: $I_2^{(4)}(m_4)$ initiates $I_1^{(4)}(m_5)$ for iteration 5.

and so on.

Proceeding in the way just described, we may construct the EXIT chart illustrated in Figure 10.33d, which embodies a trajectory that moves from one constituent decoder to the other in the form of a *staircase*. Specifically, the extrinsic information transfer curve from constituent decoder 1 to constituent decoder 2 proceeds in a horizontal manner and, by the same token, the extrinsic information transfer curve from constituent decoder 2 to constituent decoder 1 proceeds in a vertical manner. Hereafter, construction of the sequence of extrinsic information transfer curves from one constituent decoder to another is called the staircase-shaped extrinsic information transfer trajectory between constituent decoders 1 and 2.

Examination of the EXIT chart depicted in Figure 10.33d prompts us to make the following two observations:

a. Provided that the SNR at the channel output is sufficiently high, then the extrinsic information transfer curve of constituent decoder 1 stays above the straight line $I_1 = I_2$, while the corresponding extrinsic information transfer curve of constituent decoder 2 stays below this line. It follows, therefore, that an open *tunnel* exists between the extrinsic information transfer curves of the two constituent decoders. Under this scenario, the turbo-decoding algorithm *converges to a stable solution for the prescribed* E_b/N_0.

b. The estimates of extrinsic information in the turbo-decoding algorithm continually become *more reliable* from one iteration to the next as the stable solution is approached.

If, however, in contrast to the picture depicted in Figure 10.33d, *no* open tunnel exists between the extrinsic information transfer curves of constituent decoders 1 and 2 when the prescribed E_b/N_0 is relatively low, then the turbo-decoding algorithm fails to converge (i.e., the turbo-decoding algorithm is unstable). This behavior is illustrated in the EXIT chart of Figure 10.34 where the SNR has been reduced compared to that in Figure 10.33.

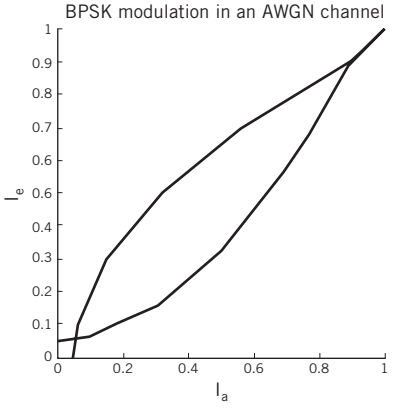


Figure 10.34
EXIT chart demonstrating nonconvergent behavior of the turbo decoder when the E_b/N_0 is reduced compared to that in Figure 10.33d.

The stage is now set for us to introduce the following statement:

> The E_b/N_0 threshold of a turbo-decoding algorithm is that smallest value of $E_b/N_0$for which an open tunnel exists in the EXIT chart.

It is the graphical simplicity of this important statement that makes the EXIT chart such a useful practical tool in the design of iterative decoding algorithms.

Moreover, if it turns out that the EXIT curves of the two constituent decoders do not intersect before the (1, 1)-point of perfect convergence and the staircase-shaped decoding trajectory also succeeds in reaching this critical point, then a vanishingly low BER is expected (Hanzo, 2012).

Approximate Gaussian Model

For an approximate model needed to display the underlying dynamics of iterative decoding algorithms, the first step is to assume that the a priori L-values for message bit m_j, namely $L_a(m_j)$, constitute independent Gaussian random variables. With $m_j = \pm 1$, the $L_a(m_j)$ assumes a variance σ_a^2 and a mean value $(\sigma_a^2/2)\, m_j$. Equivalently, we may express the statistical dependence of L_a on m_j as follows:

$$L_a(m_j) = \left(\frac{\sigma_a^2}{2}\right) m_j + n_a \tag{10.129}$$

where n_a is the sample value of a zero-mean Gaussian random variable with variance σ_a^2.

The rationale for the approximate Gaussian model just described is motivated by the following two points (Lin and Costello, 2003):

a. For an AWGN channel with soft (i.e., unquantized) output, the log-likelihood ratio, L-value, denoted by $L_a(m_j|r_j^{(0)})$ of a transmitted message bit m_j given the receiver signal $r_j^{(0)}$, may be modeled as follows (see Problem 10.36):

$$L_a(m_j|r_j^{(0)}) = L_c r_j^{(0)} + L_a(m_j) \tag{10.130}$$

where $L_c = 4(E_s/N_0)$ is the channel reliability factor defined in (10.91) and $L_a(m_j)$ is the a priori L-value of message bit m_j. The point to note here is that the product terms $L_c r_j^{(0)}$ for varying j are independent Gaussian random variables with variance $2L_c$ and mean $\pm L_c$.

b. Extensive Monte Carlo simulations of the a posteriori extrinsic L-values, $L_e(m_j)$, for a constituent decoder with large block length appear to support the Gaussian-model assumption of (10.129); see Wiberg *et al.* (1999).

Accordingly, using the Gaussian approximation of (10.129), we may express the conditional probability density function of the a priori L-value as follows:

$$f_{L_a}(\xi|m_j) = \frac{1}{\sqrt{2\pi}\,\sigma_a} \exp\left[-\frac{(\xi - m_j\sigma_a^2/2)^2}{2\sigma_a^2}\right] \tag{10.131}$$

where ξ is a dummy variable, representing a sample value of $L_a(m_j)$. Note also that ξ is continuous whereas, of course, m_j is discrete. It follows that in formulating the mutual information between the message bit $m_j = +1$ and a priori L-value $L_a(m_j)$ we have a binary

input AWGN channel to deal with; such a channel was discussed previously in Example 5 of Chapter 5 on information theory. Building on the results of that example, we may express the first desired mutual information, denoted by $I_1(m_j; L_a)$, as follows:

$$I_1(m_j; L_a) = \frac{1}{2} \sum_{m_j = -1, +1} \int_{-\infty}^{\infty} f_{L_a}(\xi | m_j) \log_2 \left(\frac{2 f_{L_a}(\xi | m_j)}{f_{L_a}(\xi | m_j = -1) + f_{L_a}(\xi | m_j = +1)} \right) d\xi \quad (10.132)$$

where the summation accounts for the binary nature of the information bit m_j and the integral accounts for the continuous nature of L_a. Using (10.131) and (10.132) and manipulating the results, we get (ten Brink, 2001):

$$I_1(m_j; L_a) = 1 - \int_{-\infty}^{\infty} \frac{\exp\left[-\dfrac{(\xi - m_j \sigma_a^2/2)^2}{2\sigma_a^2} \right]}{\sqrt{2\pi}\, \sigma_a} \log_2[1 + \exp(-\xi)]\, d\xi \quad (10.133)$$

which, as expected, depends solely on the variance σ_a^2. To emphasize this fact, let the new function

$$\mathcal{J}(\sigma_a) := I_1(m_j; L_a) \quad (10.134)$$

with the following two limiting values:

$$\lim_{\sigma_a \to 0} \mathcal{J}(\sigma_a) = 0$$

and

$$\lim_{\sigma_a \to \infty} \mathcal{J}(\sigma_a) = 1$$

In other words, we have

$$0 \le I_1(m_j; L_a) \le 1 \quad (10.135)$$

Moreover, $\mathcal{J}(\sigma_a)$ increases monotonically with increasing σ_a, which means that if the value of the mutual information $I_1(m_j; L_a)$ is given, then the corresponding value of σ_a is uniquely determined by the inverse formula:

$$\sigma_a = \mathcal{J}^{-1}(I_1) \quad (10.136)$$

and with it, the corresponding Gaussian random variable $L_a(m_j)$ defined in (10.129) is obtained.

Referring back to (10.128), we note that for us to construct the EXIT chart we also need to know the second mutual information between the message bit m_j and the a posteriori extrinsic L-value $L_p(m_j)$. To this end, we may build on the formula of (10.132) to write

$$I_2(m_j; L_p) = \frac{1}{2} \sum_{m_j = -1, +1} \int_{-\infty}^{\infty} f_{L_p}(\xi | m_j) \log_2 \left(\frac{2 f_{L_p}(\xi | m_j)}{f_{L_p}(\xi | m_j = -1) + f_{L_p}(\xi | m_j = +1)} \right) d\xi \quad (10.137)$$

where, in a manner similar to the a priori mutual information $I_1(m_j; L_a(m_j))$, we also have

$$0 \le I_2(m_j; L_p) \le 1 \quad (10.138)$$

Accordingly, with the two mutual informations $I_1(m_j;L_a)$ and $I_2(m_j;L_p)$ at hand, we may go on to compute the EXIT chart for an iterative decoding algorithm by merely focusing on a single constituent decoder in the turbo decoding algorithm.

The next issue to be considered is how to perform this computation, which we now address.

Histogram Method for Computing the EXIT Chart

For turbo codes having long interleavers, the approximate Gaussian model of (10.129) is good enough for practical purposes. Hence, we may use this model to formulate the traditional *histogram method*, described in ten Brink (2001) to compute the EXIT chart. Specifically, (10.137) is used to compute the mutual information $I_2(m_j;L_p)$ for a prescribed E_b/N_0. To this end, *Monte Carlo simulation* (i.e., histogram measurements) is used to compute the required probability density function, $f_{L_p}(\xi|L_p(m_j))$, on which *no* Gaussian assumption can be imposed for obvious reasons. Computaton of this probability density function is central to the EXIT chart, which may proceed in a step-by-step manner for a prescribed E_b/N_0, as follows:

Step 1: Apply the independent Gaussian random variable defined in (10.129) to constituent decoder 1 in the turbo decoder. The corresponding value of the mutual information $I_1(m_j;L_p)$ is obtained by choosing the variance σ_a^2 in accordance with (10.129).

Step 2: Using Monte Carlo simulation, compute the probability density function $f_{L_p}(\xi|L_p)$. Hence, compute the second mutual information $I_2(m_j;L_p)$, and with it a certain point for the extrinsic information transfer curve of constituent decoder 1 is determined.

Step 3: Continue Steps 1 and 2 until we have sufficient points to construct the extrinsic information transfer curve of constituent decoder 1.

Step 4: Construct the extrinsic information transfer curve of constituent decoder 2 as the mirror image of the curve for constituent decoder 1 computed in Step 3, respecting the straight line $I_1 = I_2$.

Step 5: Construct the EXIT chart for the turbo decoder by combining the extrinsic information transfer curves of constituent decoders 1 and 2.

Step 6: Starting with some prescribed initial condition, for example $I_1(m_1) = 0$ for message bit m_1, construct the staircase information transfer trajectory between constituent decoders 1 and 2.

A desirable feature of the histogram method for computing the EXIT chart is the fact that, except for the approximate Gaussian model of (10.129), there are *no* other assumptions needed for the computations involved in Steps 1 through 6.

Averaging Method for Computing EXIT Charts

For another method to compute EXIT charts, we may use the so-called *averaging method*, which represents an alternative approach to the histogram method.

As a reminder, the basic issue in computing an EXIT chart is to measure the mutual information between the information bits, m_j, at the turbo encoder input in the transmitter

and the corresponding L-values produced at the output of the corresponding BCJR decoder in the receiver. Due to the inherently nonlinear input–output characteristic of the BCJR decoder, the underlying distribution of the L-values is not only unknown, but also highly likely to be non-Gaussian as well, thereby complicating the measurement. To get around this difficulty, we may invoke the *ergodic theorem*, which was discussed previously in Chapter 4 on stochastic processes. As explained therein, under certain conditions, it is feasible to replace the operation of ensemble averaging (i.e., expectation) with time averaging. Thus, in proceeding along this ergodic path, we have a new nonlinear transformation, where the *time average* of a large set of L-value samples, available at the output of the BCJR decoder, provides an estimate of the desired mutual information; moreover, it does so without requiring knowledge of the original data (i.e., the m_j). It is for this reason that the second method of computing EXIT charts is called the averaging method.[15]

Just as the use of a single constituent decoder suffices for computing an EXIT chart in the histogram method, the same decoding scenario applies equally well to the averaging method. It is with this point in mind that the underlying scheme for the averaging method is as depicted in Figure 10.35. Most importantly, this scheme is designed in such a way that the following requirements are satisfied:

1. Implementations of channel estimation, carrier receiver, modulation, and demodulation are all perfect.
2. The turbo decoder is perfectly synchronized with the turbo encoder.
3. The BCJR algorithm or exact equivalent (i.e., the log-MAP algorithm) is used to optimize the turbo decoder.

Moreover, the following analytic correspondences between the constituent encoder 1 at the top of Figure 10.35 and the turbo decoder 2 at the bottom of the figure are carefully noted: the code vectors $\mathbf{c}^{(0)}$, $\mathbf{c}^{(1)}$, and termination vector $\mathbf{t}^{(1)}$ in the encoder map onto the a posteriori L-values $L_p(\mathbf{r}^{(0)})$, $L_p(\mathbf{r}^{(1)})$, and $L_p(\mathbf{z}^{(1)})$ in the decoder, respectively.

It can therefore be justifiably argued that in light of these rigorous requirements, the underlying algorithm for the averaging method is well designed and therefore trustworthy in the following sense: in the course of computing the EXIT chart, the algorithm trusts what the computed L-values actually say; that is, they do not under or over represent their confidence in the message bits. This important characteristic of the averaging method is to be contrasted against the histogram method. Indeed, it is for this reason that the histogram method compares the L-values against the true values of the message bits, hence requiring knowledge of them.

In summary, we may say that trustworthy L-values are those L-values that satisfy the *consistency condition*. A simple way of testing this condition is to do the following: use the averaging and histogram methods to compute two respective sets of L-values. If, then, both methods yield the same value for the mutual information, then the consistency condition is satisfied (Maunder, 2012).

Procedure for Measuring the EXIT Chart

Referring back to the scheme of Figure 10.35, the demultiplexer outputs denoted by $\mathbf{r}^{(0)}$, $\mathbf{r}^{(1)}$and $\mathbf{z}^{(1)}$ represent the L-values corresponding to the encoder outputs $\mathbf{c}^{(0)}$, $\mathbf{c}^{(1)}$ and $\mathbf{t}^{(1)}$, respectively. Thus, following the way in which the turbo decoder of Figure 10.33b was described, the internally generated input applied to BCJR decoder 1 assumes exactly the

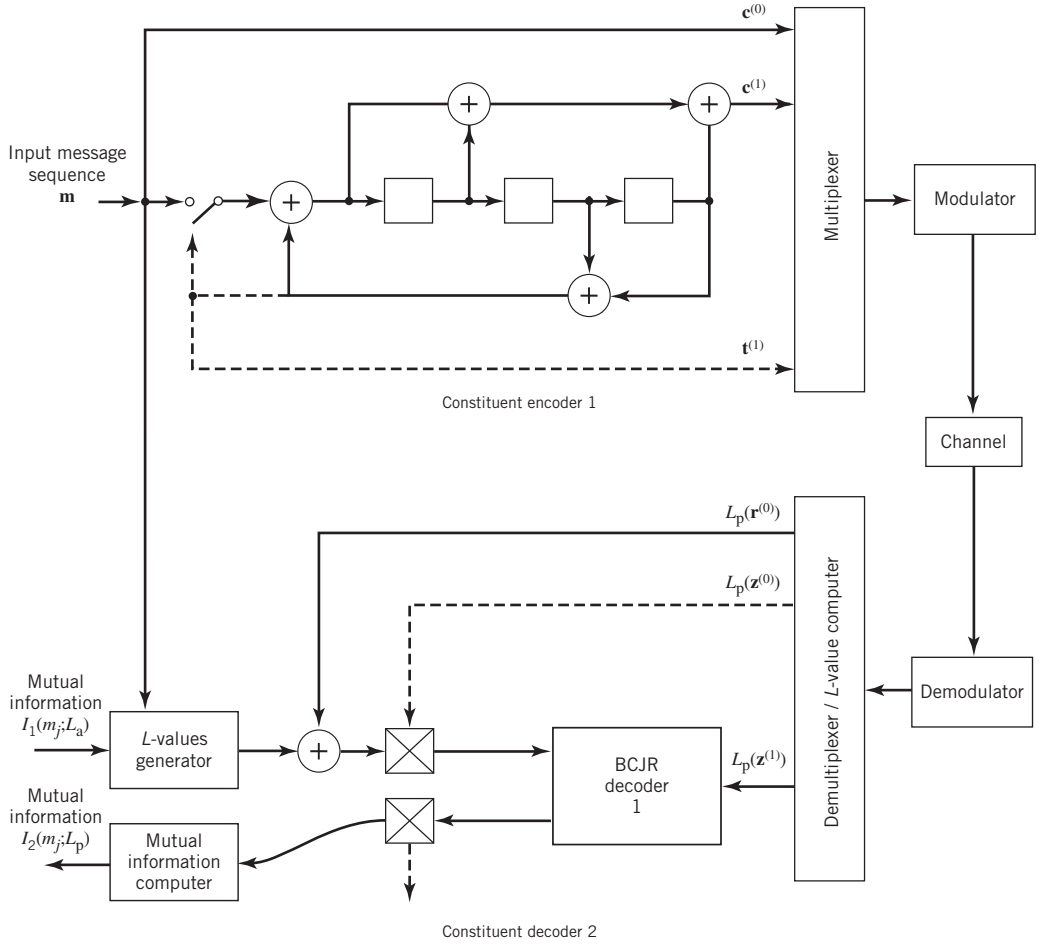


Figure 10.35 Schematic diagram for computing the EXIT chart for the UMTS-turbo code, based on the averaging method.

same value as that produced in computing the BER. With the objective being that of constructing an EXIT chart, we need to provide values for the mutual information that was previously denoted by $I_1(m_j; L_a)$, where m_j is the jth message bit and L_a is the corresponding a priori L-value. As indicated, the mutual information is the externally supplied input applied to the block labeled *L-value generator*. We may therefore assign to $I_1(m_j; L_a)$ any values that we like. However, recognizing that $0 \leq I_1(m_j; L_a) \leq 1$, a sensible choice of values for $I_1(m_j; L_a)$ would be the set {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}. Such a choice provides inputs for eleven different experiments based on the averaging method.

For each one of these inputs, the constituent decoder 1 produces a corresponding value for the a posteriori extrinsic L-value, L_p, which is applied to the block labeled *mutual-information computer* in Figure 10.35. The resulting output of this second computation is the second desired mutual information, namely $I_2(m_j; L_e)$. At this point, a question that begs itself is: how can this computation be performed in the absence of the message m_j?

The answer to this question is that, as pointed out previously, the averaging method is designed to trust what the extrinsic L-values say; hence, the computation of $I_2(m_j; L_e)$ does not require any knowledge of the message bit m_j.

Computer-Oriented Experiments for EXIT Charts

The EXIT charts plotted in Figures 10.33 and 10.34 were computed in Matlab for the UMTS codec using the averaging method discussed in Example 9 and 5000 message bits.

In Figure 10.33, the computations were performed for the SNR: $E_b/N_0 = -4$ dB. In this case, the tunnel is open, indicating that the UMTS decoder is convergent (stable).

In Figure 10.34, the computation was performed for a smaller SNR: $E_b/N_0 = -6$ dB. In this case, the tunnel is closed, indicating that the UMTS decoder is nonconvergent.

These computer experiments confirm the practical importance of EXIT charts when the issue of interest is that of evaluating the dynamic behavior of a turbo decoder.

10.14 Low-Density Parity-Check Codes

Turbo codes discussed in Section 10.12 and LDPC codes[16] to be discussed in this section belong to a broad family of error-control coding techniques, collectively called *compound probabilistic codes*. The two most important advantages of LDPC codes over turbo codes are:

- absence of low-weight codewords and
- iterative decoding of lower complexity.

With regard to the issue of low-weight codewords, we usually find that a small number of codewords in a turbo codeword are undesirably close to the given codeword. Owing to this closeness in weights, once in a while the channel noise causes the transmitted codeword to be mistaken for a nearby code. Indeed, it is this behavior that is responsible for the error floor that was mentioned in Section 10.13. In contrast, LDPC codes can be easily constructed so that they do not have such low-weight codewords and they can, therefore, achieve *vanishingly small* BERs. (The error-floor problem in turbo codes can be alleviated by careful design of the interleaver.)

Turning next to the issue of decoding complexity, we note that the computational complexity of a turbo decoder is dominated by the MAP algorithm, which operates on the trellis for representing the convolutional code used in the encoder. The number of computations in each recursion of the MAP algorithm scales linearly with the number of states in the trellis. Commonly used turbo codes employ trellises with 16 states or more. In contrast, LDPC codes use a simple parity-check trellis that has just two states. Consequently, the decoders for LDPC codes are significantly simpler to design than those for turbo decoders. However, a practical objection to the use of LDPC codes is that, for large block lengths, their encoding complexity is high compared with turbo codes.

It can be argued that LDPC codes and turbo codes complement each other, giving the designer more flexibility in selecting the right code for extraordinary decoding performance.

Construction of LDPC Codes

LDPC codes are specified by a parity-check matrix denoted by $\mathbf{A}$, which is purposely chosen to be *sparse*; that is, the code consists mainly of 0s and a small number of 1s. In particular, we speak of (n, t_c, t_r) LDPC codes, where n denotes the block length, t_c denotes

the weight (i.e., number of 1s) in each column of the matrix **A**, and t_r denotes the weight of each row with $t_r > t_c$. The rate of such an LDPC code is defined by

$$r = 1 - \frac{t_c}{t_r}$$

whose validity may be justified as follows. Let ρ denote the *density* of 1s in the parity-check matrix **A**. Then, following the terminology introduced in Section 10.4, we may set

$$t_c = \rho(n - k)$$

and

$$t_r = \rho n$$

where $(n - k)$ is the number of rows in **A** and n is the number of columns (i.e., the block length). Therefore, dividing t_c by t_r, we get

$$\frac{t_c}{t_r} = 1 - \frac{k}{n} \tag{10.139}$$

By definition, the code rate of a block code is k/n; hence, the result of (10.139) follows. For this result to hold, however, the rows of **A** must be *linearly independent*.

The structure of LDPC codes is well portrayed by bipartite graphs, which were introduced by Tanner (1981) and, therefore, are known as *Tanner graphs*.[17] Figure 10.36 shows such a graph for the example code of $n = 10$, $t_c = 3$, and $t_r = 5$. The left-hand nodes in the graph are *variable (symbol) nodes*, which correspond to elements of the codeword. The right-hand nodes of the graph are *check nodes*, which correspond to the set of parity-check constraints satisfied by codewords in the code. LDPC codes of the type exemplified by the graph of Figure 10.36 are said to be *regular*, in that all the nodes of a similar kind have exactly the same degree. In Figure 10.36, the degree of the variable nodes is $t_c = 3$ and the degree of the check nodes is $t_r = 5$. As the block length n approaches infinity, each check node is connected to a vanishingly small fraction of variable nodes; hence the term "low density."

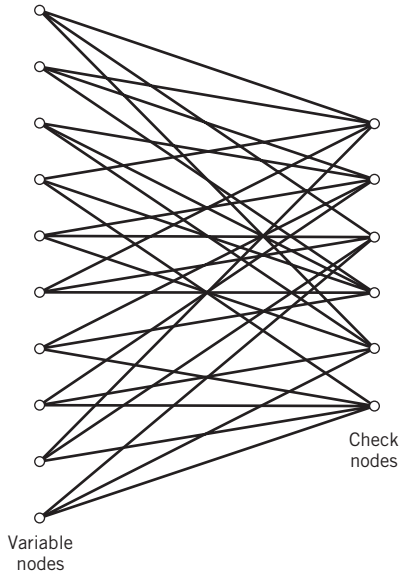


Figure 10.36
Bipartite graph of the (10, 3, 5) LDPC code.

The matrix **A** is constructed by putting 1s in **A** at *random*, subject to the *regularity constraints*:

- each column of matrix **A** contains a small fixed number t_c of 1s;
- each row of the matrix contains a small fixed number t_r of 1s.

In practice, these regularity constraints are often violated slightly in order to avoid having linearly dependent rows in the parity-check matrix **A**.

Unlike the linear block codes discussed in Section 10.4, the parity-check matrix **A** of LDPC codes is *not* systematic (i.e., it does not have the parity-check bits appearing in diagonal form); hence the use of a symbol different from that used in Section 10.4. Nevertheless, for coding purposes, we may derive a generator matrix **G** for LDPC codes by means of Gaussian elimination performed in modulo-2 arithmetic; this procedure is illustrated later in Example 10. Following the terminology introduced in Section 10.4, the 1-by-n code vector **c** is first partitioned as shown by

$$\mathbf{c} = \left[\mathbf{b} \mid \mathbf{m}\right]$$

where **m** is the k-by-1 message vector and **b** is the $(n-k)$-by-1 parity-check vector; see (10.9). Correspondingly, the parity-check matrix **A** is partitioned as

$$\mathbf{A}^{\mathrm{T}} = \begin{bmatrix} \mathbf{A}_1 \\ \hline \mathbf{A}_2 \end{bmatrix} \tag{10.140}$$

where $\mathbf{A}_1$ is a square matrix of dimensions $(n-k) \times (n-k)$ and $\mathbf{A}_2$ is a rectangular matrix of dimensions $k \times (n-k)$; transposition symbolized by the superscript T is used in the partitioning of matrix **A** for convenience of presentation. Imposing a constraint on the LDPC code similar to that of (10.16) we may write

$$\left[\mathbf{b} \mid \mathbf{m}\right] \begin{bmatrix} \mathbf{A}_1 \\ \hline \mathbf{A}_2 \end{bmatrix} = \mathbf{0}$$

or, equivalently,

$$\mathbf{b}\mathbf{A}_1 + \mathbf{m}\mathbf{A}_2 = \mathbf{0} \tag{10.141}$$

Recall from (10.7) that the vectors **m** and **b** are related by

$$\mathbf{b} = \mathbf{m}\mathbf{P}$$

where **P** is the *coefficient matrix*. Hence, substituting this relation into (10.141), we readily find that, after ignoring the common factor **m** for any nonzero message vector, the coefficient matrix of LDPC codes satisfies the condition

$$\mathbf{P}\mathbf{A}_1 + \mathbf{A}_2 = \mathbf{0} \tag{10.142}$$

This equation holds for *all* nonzero message vectors and, in particular, for **m** in the form [0 … 0 1 0 … 0] that will isolate individual rows of the generator matrix.

Solving (10.142) for matrix **P**, we get

$$\mathbf{P} = \mathbf{A}_2\mathbf{A}_1^{-1} \tag{10.143}$$

where $\mathbf{A}_1^{-1}$ is the *inverse* of matrix $\mathbf{A}_1$, which is naturally defined in modulo-2 arithmetic. Finally, building on (10.12), the generator matrix of LDPC codes is defined by

$$\begin{aligned}\mathbf{G} &= \left[\mathbf{P} \;\vdots\; \mathbf{I}_k\right] \\ &= \left[\mathbf{A}_2\mathbf{A}_1^{-1} \;\vdots\; \mathbf{I}_k\right]\end{aligned} \tag{10.144}$$

where $\mathbf{I}_k$ is the k-by-k identity matrix.

It is important to note that if we take the parity-check matrix $\mathbf{A}$ for some arbitrary LDPC code and just pick $(n-k)$ columns of $\mathbf{A}$ at random to form a square matrix $\mathbf{A}_1$, there is *no* guarantee that $\mathbf{A}_1$ will be *nonsingular* (i.e., the inverse $\mathbf{A}_1^{-1}$ will exist), even if the rows of $\mathbf{A}$ are linearly independent. In fact, for a typical LDPC code with large block length n, such a randomly selected $\mathbf{A}_1$ is highly unlikely to be nonsingular because it is very likely that at least one row of $\mathbf{A}_1$ will be all 0s. Of course, when the rows of $\mathbf{A}$ are linearly independent, there will be *some* set of $(n-k)$ columns of $\mathbf{A}$ that will result in a nonsingular $\mathbf{A}_1$, to be illustrated in Example 10. For some construction methods for LDPC codes, the first $(n-k)$ columns of $\mathbf{A}$ may be guaranteed to produce a nonsingular $\mathbf{A}_1$, or at least do so with a high probability, but that is *not* true in general.

EXAMPLE 10 **(10, 3, 5) LDPC Code**

Consider the Tanner graph of Figure 10.34 pertaining to a (10, 3, 5) LDPC code. The parity-check matrix of the code is defined by

$$\mathbf{A} = \underbrace{\left[\begin{array}{cccccc} 1&1&0&1&0&1 \\ 0&1&1&0&1&0 \\ 1&0&0&0&1&1 \\ 0&1&1&1&0&1 \\ 1&0&1&0&1&0 \\ 0&0&0&1&0&0 \end{array}\right.}_{\mathbf{A}_1^{\mathrm{T}}} \underbrace{\left.\begin{array}{cccc} 0&0&1&0 \\ 1&1&0&0 \\ 0&0&1&1 \\ 1&0&0&0 \\ 0&1&0&1 \\ 1&1&1&1 \end{array}\right]}_{\mathbf{A}_2^{\mathrm{T}}}$$

which appears to be random, while maintaining the regularity constraints: $t_c = 3$ and $t_r = 5$. Partitioning the matrix $\mathbf{A}$ in the manner just described, we write

$$\mathbf{A}_1 = \begin{bmatrix} 1&0&1&0&1&0 \\ 1&1&0&1&0&0 \\ 0&1&0&1&1&0 \\ 1&0&0&1&0&1 \\ 0&1&1&0&1&0 \\ 1&0&1&1&0&0 \end{bmatrix}$$

$$\mathbf{A}_2 = \begin{bmatrix} 0&1&0&1&0&1 \\ 0&1&0&0&1&1 \\ 1&0&1&0&0&1 \\ 0&0&1&0&1&1 \end{bmatrix}$$

To derive the inverse of matrix $\mathbf{A}_1$, we first use (10.140) to write

$$\underbrace{[\, b_0, b_1, b_2, b_3, b_4, b_5 \,]}_{\mathbf{b}} \underbrace{\begin{bmatrix} 1&0&1&0&1&0 \\ 1&1&0&1&0&0 \\ 0&1&0&1&1&0 \\ 1&0&0&1&0&1 \\ 0&1&1&0&1&0 \\ 1&0&1&1&0&0 \end{bmatrix}}_{\mathbf{A}_1} = \underbrace{[\, u_0, u_1, u_2, u_3, u_4, u_5 \,]}_{\mathbf{u} = \mathbf{m}\mathbf{A}_2}$$

where we have introduced the vector $\mathbf{u}$ to denote the matrix product $\mathbf{m}\mathbf{A}_2$. By using *Gaussian elimination*, *modulo-2*, the matrix $\mathbf{A}_1$ is transformed into lower diagonal form (i.e., all the elements above the main diagonal are zero), as shown by

$$\mathbf{A}_1 \rightarrow \begin{bmatrix} 1&0&0&0&0&0 \\ 1&1&0&0&0&0 \\ 0&1&1&0&0&0 \\ 1&0&1&1&0&0 \\ 0&1&0&1&1&0 \\ 1&0&0&1&0&1 \end{bmatrix}$$

This transformation is achieved by the following modulo-2 additions performed on the columns of square matrix $\mathbf{A}_1$:

- columns 1 and 2 are added to column 3;
- column 2 is added to column 4;
- columns 1 and 4 are added to column 5;
- columns 1, 2, and 5 are added to column 6.

Correspondingly, the vector $\mathbf{u}$ is transformed as shown by

$$\mathbf{u} \rightarrow [u_0, u_1, u_0 + u_1 + u_2, u_1 + u_3, u_0 + u_3 + u_4, u_0 + u_1 + u_4 + u_5]$$

Accordingly, premultiplying the transformed matrix $\mathbf{A}_1$ by the parity vector $\mathbf{b}$, using successive eliminations in modulo-2 arithmetic working backwards and putting the solutions for the elements of the parity vector $\mathbf{b}$ in terms of the elements of the vector $\mathbf{u}$ in matrix form, we get

$$\underbrace{[\, u_0, u_1, u_2, u_3, u_4, u_5 \,]}_{\mathbf{u}} \underbrace{\begin{bmatrix} 0&0&1&0&1&1 \\ 1&0&1&0&0&1 \\ 1&1&1&0&0&0 \\ 1&1&0&0&1&0 \\ 0&1&0&0&1&1 \\ 1&1&1&1&0&1 \end{bmatrix}}_{\mathbf{A}_1^{-1}} = \underbrace{[\, b_0, b_1, b_2, b_3, b_4, b_5 \,]}_{\mathbf{b}}$$

The inverse of matrix $\mathbf{A}_1$ is therefore

$$\mathbf{A}_1^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Using the given value of $\mathbf{A}_2$ and the value of $\mathbf{A}_1^{-1}$ just found, the matrix product $\mathbf{A}_2\mathbf{A}_1^{-1}$ is given by

$$\mathbf{A}_2\mathbf{A}_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Finally, using (10.144), the generator of the (10, 3, 5) LDPC code is defined by

$$\mathbf{G} = \underbrace{\left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{array}\right.}_{\mathbf{A}_2\mathbf{A}_1^{-1}} \underbrace{\left.\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right]}_{\mathbf{I}_k}$$

It is important to recognize that the LDPC code described in this example is intended only for the purpose of illustrating the procedure involved in the generation of such a code. In practice, the block length n is orders of magnitude larger than that considered in this example. Moreover, in constructing the matrix $\mathbf{A}$, we may constrain all pairs of columns to have a *matrix overlap* (i.e., inner product of any two columns in matrix $\mathbf{A}$) not to exceed one; such a constraint, over and above the regularity constraints, is expected to improve the performance of LDPC codes. Unfortunately, with a small block length as that considered in this example, it is difficult to satisfy this additional requirement.

Minimum Distance of LDPC Codes

In practice, the block length of an LDPC code is large, ranging from 10^3 to 10^6, which means that the number of codewords in a particular code is correspondingly large. Consequently, the algebraic analysis of LDPC codes is rather difficult. As such, it is much more productive to perform a *statistical analysis* on an ensemble of LDPC codes. Such an analysis permits us to make statistical statements about certain properties of member codes in the ensemble. An LDPC code with these properties can be found with high probability by a *random selection from the ensemble*, hence the inherent probabilistic structure of the code.

Among these properties, the minimum distance of the member codes is of particular interest. From Section 10.4, we recall that the minimum distance of a linear block code is the smallest Hamming distance between any pair of code vectors in the code. In contrast, we say:

> Over an ensemble of LDPC codes, the minimum distance of a member code in the ensemble is naturally a random variable.

Elsewhere,[18] it is shown that as the block length n increases for fixed $t_c \geq 3$ and $t_r > t_c$, the probability distribution of the minimum distance can be overbounded by a function that approaches a unit step function at a fixed fraction $\Delta_{t_c t_r}$ of the block length n. Thus, for large n, practically all the LDPC codes in the ensemble have a minimum distance of at least $n\Delta_{t_c t_r}$. Table 10.7 presents the rate and $\Delta_{t_c t_r}$ of LDPC codes for different values of the weight-pair (t_c, t_r). From this table we see that for $t_c = 3$ and $t_r = 6$, the code rate r attains its highest value of 1/2 and the fraction $\Delta_{t_c t_r}$ attains its smallest value; hence the preferred choice of $t_c = 3$ and $t_r = 6$ in constructing the LDPC code.

Probabilistic Decoding of LDPC Codes

At the transmitter, a message vector $\mathbf{m}$ is encoded into a code vector $\mathbf{c} = \mathbf{mG}$, where $\mathbf{G}$ is the generator matrix for a specified weight-pair (t_c, t_r) and, therefore, minimum distance $d_{\min}$. The vector $\mathbf{c}$ is transmitted over a noisy channel to produce the received vector

$$\mathbf{r} = \mathbf{c} + \mathbf{e}$$

where $\mathbf{e}$ is the error vector due to channel noise; see (10.17). By construction, the matrix $\mathbf{A}$ is the parity-check matrix of the LDPC code; that is, $\mathbf{AG}^T = \mathbf{0}$. Given the received vector $\mathbf{r}$, the bit-by-bit decoding problem is to find the most probable vector $\hat{\mathbf{c}}$ that satisfies the condition $\hat{\mathbf{c}}\mathbf{A}^T = \mathbf{0}$ in accordance with the constraint imposed on matrix $\mathbf{A}$ in (10.140).

In what follows, a bit refers to an element of the received vector $\mathbf{r}$ and a check refers to a row of matrix $\mathbf{A}$. Let $\mathcal{J}(i)$ denote the set of bits that participate in check i. Let $\mathcal{J}(j)$ denote the set of checks in which bit j participates. A set of $\mathcal{J}(i)$ that excludes bit j is denoted by $\mathcal{J}(i)\backslash j$. Likewise, a set of $\mathcal{J}(j)$ that excludes check i is denoted by $\mathcal{J}(j)\backslash i$.

Table 10.7 The rate and fractional term of LDPC codes for varying weight-pairs*

t_c	t_r	Rate r	$\Delta_{t_c t_r}$
5	6	0.167	0.255
4	5	0.2	0.210
3	4	0.25	0.122
4	6	0.333	0.129
3	5	0.4	0.044
3	6	0.5	0.023

The decoding algorithm has two alternating steps: a *horizontal step* and a *vertical step*, which run along the rows and columns of matrix $\mathbf{A}$, respectively. In the course of decoding, two probabilistic quantities associated with nonzero elements of matrix $\mathbf{A}$ are alternately updated. One quantity, denoted by P_{ij}^{x}, defines the probability that bit j is symbol x (i.e., symbol 0 or 1), given the information derived via checks performed in the horizontal step except for check i. The second quantity, denoted by Q_{ij}^{x}, defines the probability that check i is satisfied given that bit j is fixed at the value x and the other bits have the probabilities P_{ij} where we have $j' \in \mathcal{J}(i)\backslash j$.

The LDPC decoding algorithm then proceeds as follows.[19]

Initialization

The variables $P_{ij}^{(0)}$ and $P_{ij}^{(1)}$ are set equal to the a priori probabilities $p_j^{(0)}$ and $p_j^{(1)}$ of symbols 0 and 1, respectively, with $p_j^{(0)} + p_j^{(1)} = 1$ for all j.

Horizontal Step

In the horizontal step of the algorithm, we run through the checks i. To this end, define

$$\Delta P_{ij} = P_{ij}^{(0)} - P_{ij}^{(1)}$$

For each weight-pair (i, j), compute

$$\Delta Q_{ij} = \prod_{j' \in \mathcal{J}(i)\backslash j} \Delta P_{ij'}$$

Hence, set

$$Q_{ij}^{(0)} = \frac{1}{2}(1 + \Delta Q_{ij})$$

$$Q_{ij}^{(1)} = \frac{1}{2}(1 - \Delta Q_{ij})$$

Vertical Step

In the vertical step of the algorithm, values of the probabilities $P_{ij}^{(0)}$ and $P_{ij}^{(1)}$ are updated using the quantities computed in the horizontal step. In particular, for each bit j, compute

$$P_{ij}^{(0)} = \alpha_{ij} p_j^{(0)} \prod_{i' \in \mathcal{J}(i)\backslash j} \Delta Q_{i'j}^{(0)}$$

$$P_{ij}^{(1)} = \alpha_{ij} p_j^{(1)} \prod_{i' \in \mathcal{J}(i)\backslash j} \Delta Q_{i'j}^{(1)}$$

where the scaling factor α_{ij} is chosen so as to satisfy the condition

$$P_{ij}^{(0)} + P_{ij}^{(1)} = 1 \qquad \text{for all } ij$$

In the vertical step, we may also update the *pseudo-posterior probabilities*:

$$P_j^{(0)} = \alpha_j p_j^{(0)} \prod_{i \in \mathcal{J}(j)} Q_{ij}^{(0)}$$

$$P_j^{(1)} = \alpha_j p_j^{(1)} \prod_{i \in \mathcal{J}(j)} Q_{ij}^{(1)}$$

where α_j is chosen so as to make

$$P_j^{(0)} + P_j^{(1)} = 1 \qquad \text{for all } j$$

The quantities obtained in the vertical step are used to compute a tentative estimate $\hat{\mathbf{c}}$. If the condition $\hat{\mathbf{c}}\mathbf{A}^{\mathrm{T}} = \mathbf{0}$ is satisfied, the decoding algorithm is terminated. Otherwise, the algorithm goes back to the horizontal step. If after some maximum number of iterations (e.g., 100 or 200) there is no valid decoding, then a decoding failure is declared. The decoding procedure described herein is a special case of the general low-complexity sum–product algorithm.

Simply stated, the *sum–product algorithm*[20] passes probabilistic quantities between the check nodes and variable nodes of the Tanner graph. On account of the fact that each parity-check constraint can be represented by a simple convolutional coder with one bit of memory, we find that LDPC decoders are simpler to implement than turbo decoders, as stated earlier on in the section.

In terms of performance, however, we may say the following in light of experimental results reported in the literature:

> Regular LDPC codes do not appear to come as close to Shannon's limit as do their turbo code counterparts.

Irregular LDPC Codes

Thus far in this section, we have focused attention on regular LDPC codes, which distinguish themselves in the following way: referring to the Tanner (bipartite) graph in Figure 10.36, all variable nodes on the left-hand side of the graph have the same degree and likewise for the check nodes on the right-hand side of the graph.

To go beyond the performance attainable with regular LDPC codes and thereby come increasingly closer to the Shannon limit, we look to irregular LDPC codes, in the context of which we introduce the following definition:

> An LDPC code is said to be irregular if the variable nodes in the code's Tanner graph have multiple degrees, and so do the check nodes in the graph.

To be specific, an irregular LDPC code distinguishes itself from its regular counterpart in that its Tanner graph involves the following two degree distributions:

a. The degree distribution of the variable nodes in the Tanner graph of an irregular LDPC code is described by:

$$\lambda(X) = \sum_{d=1}^{d_N} \lambda_d X^{d-1} \tag{10.145}$$

where X denotes a node variable in the code's Tanner graph, λ_d denotes the fraction of variable nodes with degree d in the graph, and d_N denotes the maximum degree of a variable node in the graph.

b. Correspondingly, the degree distribution of the check nodes in the irregular code's Tanner graph is described by

$$\rho(X) = \sum_{d=1}^{d_c} \rho_d X^{d-1} \tag{10.146}$$

where X denotes the check node in the code's Tanner graph, ρ_d denotes the fraction of check nodes with degree d in the graph, and d_c denotes the maximum degree of a check node in the graph.

The irregular LDPC code embodies the regular LDPC code as a special case. Specifically, (10.145) and (10.146) simplify as follows for the variable and check nodes of a regular LDPC code, respectively:

$$\lambda(X) = X^{\omega_N - 1} \quad \text{for } \lambda_d = 1 \text{ and } d_N = \omega_N \tag{10.147}$$

and

$$\rho(X) = X^{\omega_c - 1} \quad \text{for } \rho_d = 1 \text{ and } d_c = \omega_c \tag{10.148}$$

By exploiting the two degree distributions of (10.145) and (10.146) for the variable and check nodes, respectively, irregular LDPC codes are commonly constructed on the basis of their Tanner graphs. Such an approach is exemplified by the irregular LDPC codes reported in Richardson *et al*. (2001), and Richardson and Urbanke (2001).[21]

10.15 Trellis-Coded Modulation

In the different approaches to channel coding described up to this point in the chapter, the one common feature that describes them all may be summarized as follows:

> Encoding is performed separately from modulation in the transmitter, and likewise for decoding and detection in the receiver.

Moreover, error control is provided by transmitting additional redundant bits in the code, which has the effect of lowering the information bit rate per channel bandwidth. That is, bandwidth efficiency is traded for increased power efficiency.

To attain a more effective utilization of available resources, namely bandwidth and power, coding and modulation would have to be treated as a combined (single) entity. We may deal with this new paradigm by invoking the statement

> Coding is redefined as *the process of imposing certain patterns on the modulated signal in the transmitter.*

Indeed, this definition includes the traditional idea of parity-check coding.

Trellis codes for band-limited channels result from the treatment of modulation and coding as a *combined* entity rather than as two separate operations. The combination itself is referred to as *trellis-coded modulation* (TCM).[22] This form of signaling has three basic requirements:

1. The number of signal points in the constellation used is larger than what is required for the modulation format of interest with the same data rate; the additional signal points allow redundancy for forward error-control coding without sacrificing bandwidth.
2. Convolutional coding is used to introduce a certain dependency between successive signal points, such that only certain *patterns* or *sequences of signal points* are permitted for transmission.
3. *Soft-decision decoding* is performed in the receiver, in which the permissible sequence of signals is modeled as a trellis structure; hence the name *trellis codes*.

Requirement 3 is the result of using an enlarged signal constellation. By increasing the size of the constellation, the probability of symbol error increases for a fixed SNR. Hence, with hard-decision demodulation, we would face a performance loss before we begin. Performing soft-decision decoding on the combined code and modulation trellis ameliorates this problem.

In an AWGN channel, we look to the following approach:

> Maximum likelihood decoding of trellis codes consists of finding that particular path through the trellis with *minimum squared Euclidean distance* to the received sequence.

Thus, in the design of trellis codes, the emphasis is on maximizing the Euclidean distance between code vectors (equivalently codewords) rather than maximizing the Hamming distance of an error-correcting code. The reason for this approach is that, except for conventional coding with binary PSK and QPSK, maximizing the Hamming distance is not the same as maximizing the squared Euclidean distance. Accordingly, in what follows, the Euclidean distance between code vectors is adopted as the distance measure of interest. Moreover, while a more general treatment is possible, the discussion is (by choice) confined to the case of *two-dimensional constellations of signal points*. The implication of such a choice is to restrict the development of trellis codes to multilevel amplitude and/or phase modulation schemes such as M-ary PSK and M-ary QAM.

EXAMPLE 11 Two-level Partitioning of 8-PSK Constellation

The approach used to design this restricted type of trellis codes involves partitioning an M-ary constellation of interest successively into 2, 4, 8, … subsets with size $M/2$, $M/4$, $M/8$, …, and having progressively larger increasing minimum Euclidean distance between their respective signal points. Such a design approach by *set-partitioning* represents the key idea in the construction of efficient coded modulation techniques for band-limited channels.

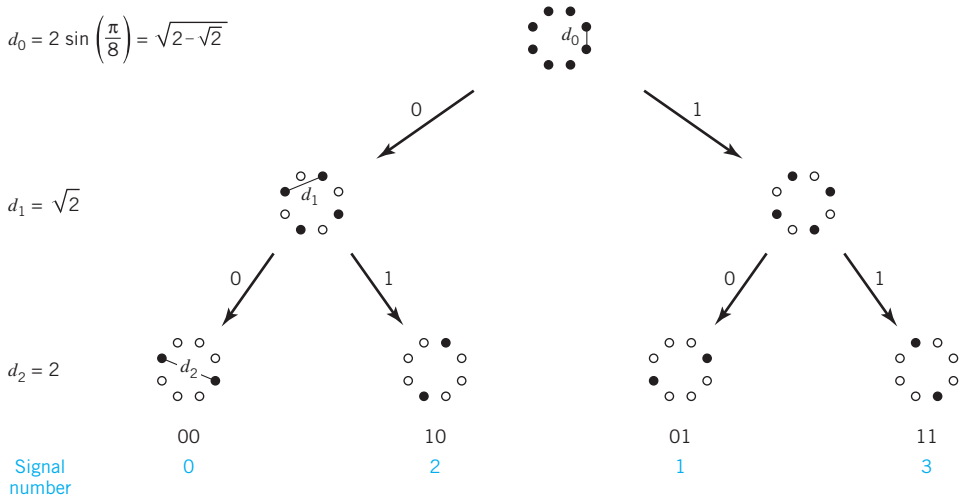


Figure 10.37 Partitioning of 8-PSK constellation, which shows that $d_0 < d_1 < d_2$.

In Figure 10.37, we illustrate the partitioning procedure by considering a circular constellation that corresponds to 8-PSK. The figure depicts the constellation itself and the two and four subsets resulting from two levels of partitioning. These subsets share the common property that the minimum Euclidean distances between their individual points follow an increasing pattern, namely:

$$d_0 < d_1 < d_2$$

EXAMPLE 12 **Three-level Partitioning of QAM Constellation**

For a different two-dimensional example, Figure 10.38 illustrates the partitioning of a rectangular constellation corresponding to 16-QAM. Here again, we see that the subsets have increasing within-subset Euclidean distances, as shown by

$$d_0 < d_1 < d_2 < d_3$$

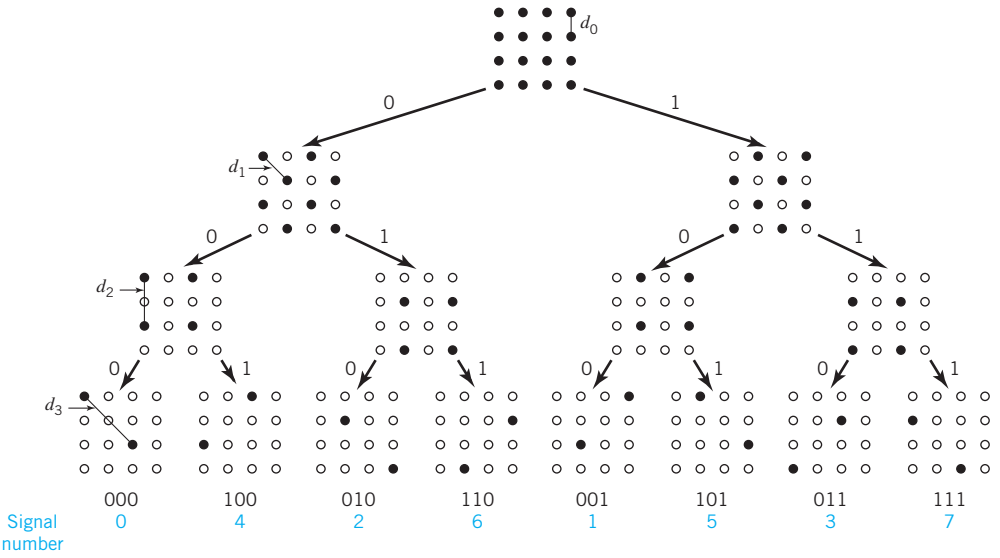


Figure 10.38 Partitioning of 16-QAM constellation, which shows that $d_0 < d_1 < d_2 < d_3$.

Based on the subsets resulting from successive partitioning of a two-dimensional constellation, illustrated in Examples 11 and 12, we may devise relatively simple, yet highly effective coding schemes. Specifically, to send n bits/symbol with *quadrature modulation* (i.e., one that has in-phase and quadrature components), we start with a two-dimensional constellation of 2^{n+1} signal points appropriate for the modulation format of interest; a circular grid is used for M-ary PSK and a rectangular one for M-ary QAM. In any event, the constellation is partitioned into four or eight subsets. One or two incoming message bits per symbol enter a rate-1/2 or rate-2/3 binary convolutional encoder, respectively; the resulting two or three coded bits per symbol determine the selection of a particular subset. The remaining uncoded messege bits determine which particular signal point from the selected

subset is to be signaled over the AWGN channel. This class of trellis codes is known as *Ungerboeck codes* in recognition of their originator.

Since the modulator has memory, we may use the *Viterbi algorithm* (discussed in Section 10.8) to perform maximum likelihood sequence estimation at the receiver. Each branch in the trellis of the Ungerboeck code corresponds to a subset rather than an individual signal point. The first step in the detection is to determine the signal point within each subset that is closest to the received signal point in the Euclidean sense. The signal point so determined and its metric (i.e., the squared Euclidean distance between it and the received point) may be used thereafter for the branch in question, and the Viterbi algorithm may then proceed in the usual manner.

Ungerboeck Codes for 8-PSK

The scheme of Figure 10.39a depicts the simplest Ungerboeck 8-PSK code for the transmission of 2 bits/symbol. The scheme uses a rate-1/2 convolutional encoder; the corresponding trellis of the code is shown in Figure 10.39b, which has four states. Note that the most significant bit of the incoming message sequence is left uncoded. Therefore, each branch of the trellis may correspond to two different output values of the 8-PSK modulator or, equivalently, to one of the four two-point subsets shown in Figure 10.37. The trellis of Figure 10.39b also includes the minimum distance path.

The scheme of Figure 10.40a depicts another Ungerboeck 8-PSK code for transmitting 2 bits/sample; it is next in the level of increased complexity, compared to the scheme of Figure 10.39a. This second scheme uses a rate-2/3 convolutional encoder. Therefore, the corresponding trellis of the code has eight states, as shown in Figure 10.40b. In this latter scheme, both bits of the incoming message sequence are encoded. Hence, each branch of the trellis corresponds to a specific output value of the 8-PSK modulator. The trellis of Figure 10.40b also includes the minimum distance path.

Figures 10.39b and 10.40b also include the pertinent encoder states. In Figure 10.39a, the state of the encoder is defined by the contents of the two-stage shift register. On the other hand, in Figure 10.40a, it is defined by the content of the single-stage (top) shift register followed by that of the two-stage (bottom) shift register.

Asymptotic Coding Gain

Following the discussion in Section 10.8 on maximum likelihood decoding of convolutional codes, we define the asymptotic coding gain of Ungerboeck codes as follows:

$$G_a = 10 \log_{10}\left(\frac{d_{\text{free}}^2}{d_{\text{ref}}^2}\right) \tag{10.149}$$

where d_{free} is the *free Euclidean distance* of the code and d_{ref} is the minimum Euclidean distance of an uncoded modulation scheme operating with the same signal energy per bit. For example, by using the Ungerboeck 8-PSK code of Figure 10.39a, the signal constellation has eight message points and we send two message bits per signal point. Hence, uncoded transmission requires a signal constellation with four message points. We

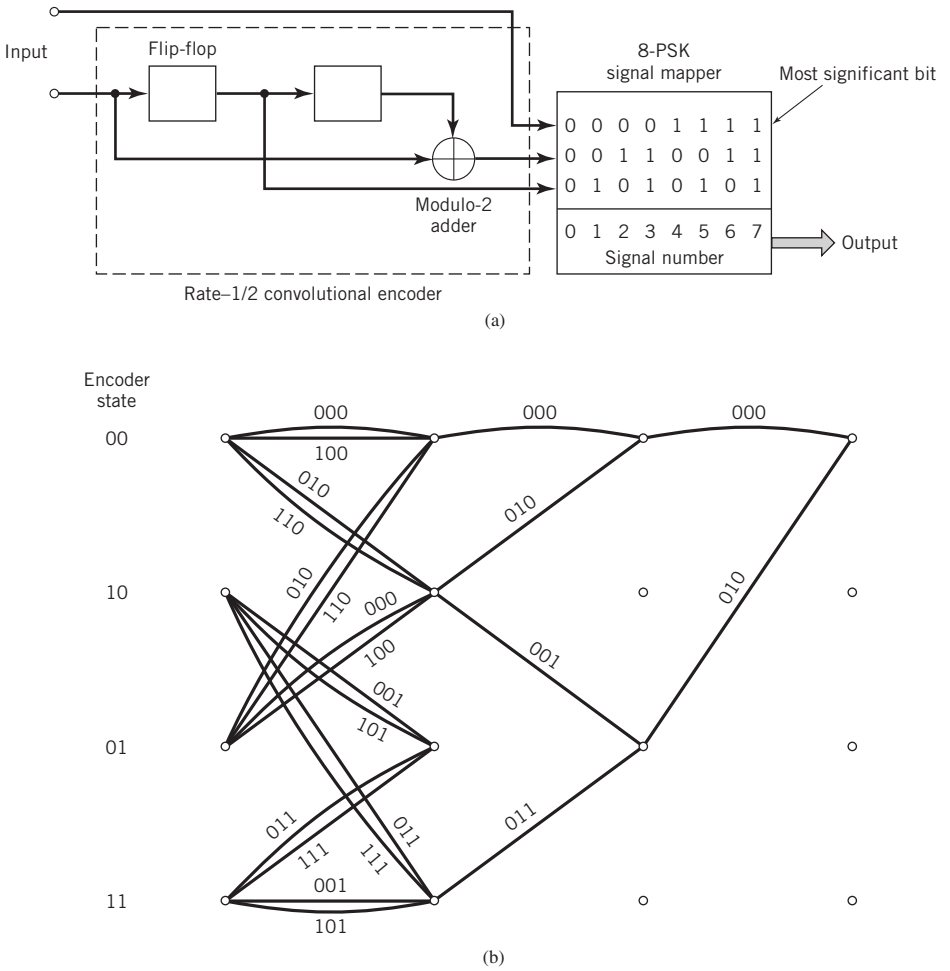


Figure 10.39 (a) Four-state Ungerboeck code for 8-PSK; the mapper follows Figure 10.37. (b) Trellis of the code.

may therefore regard uncoded 4-PSK as the frame of reference for the Ungerboeck 8-PSK code of Figure 10.39a.

The Ungerboeck 8-PSK code of Figure 10.39a achieves an asymptotic coding gain of 3 dB, which is calculated as follows:

1. Each branch of the trellis in Figure 10.39b corresponds to a subset of two antipodal signal points. Hence, the free Euclidean distance d_{free} of the code can be no larger than the Euclidean distance d_2 between the antipodal signal points of such a subset. We may therefore write

$$d_{\text{free}} = d_2 = 2$$

where the distance d_2 is defined in Figure 10.41a.

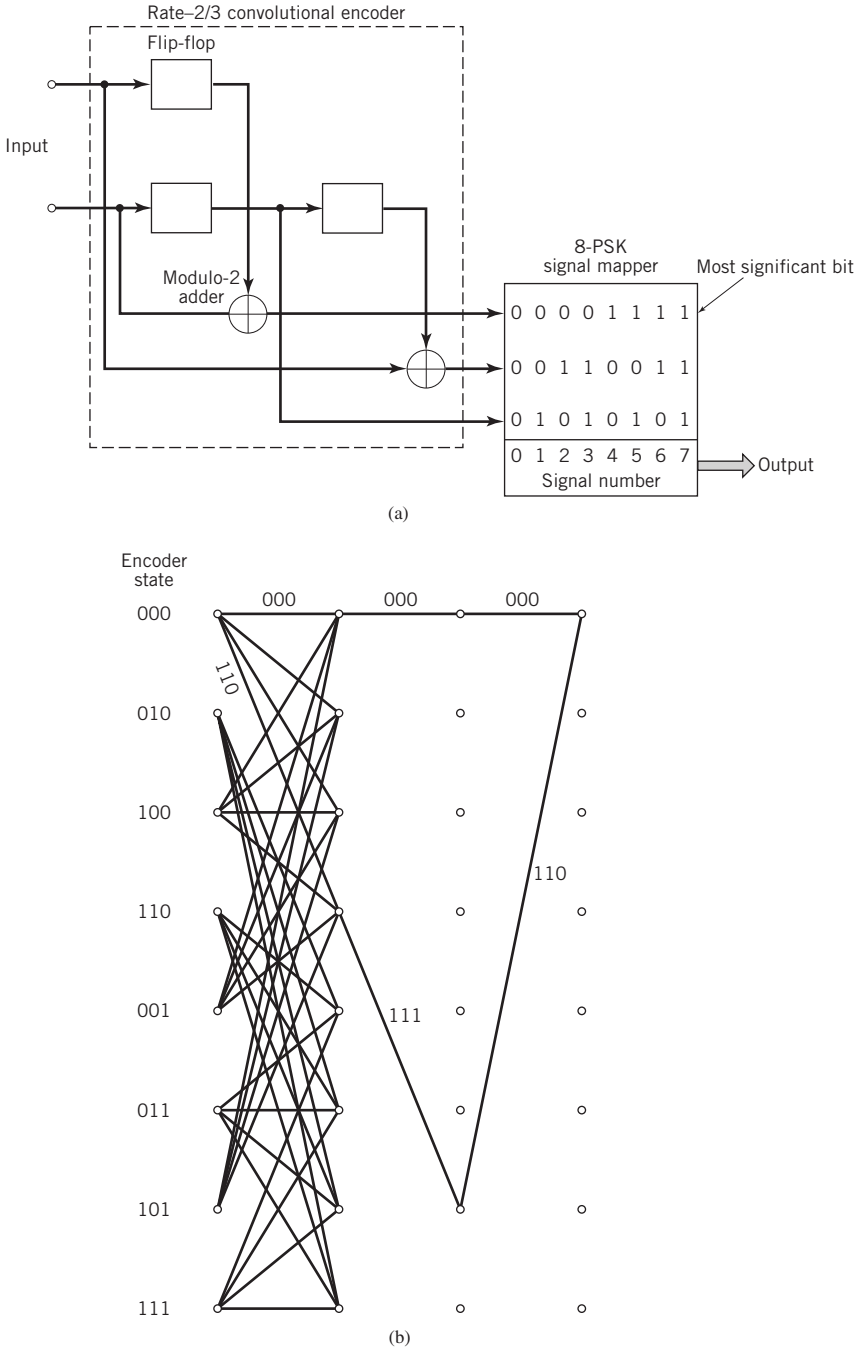


Figure 10.40 (a) Eight-state Ungerboeck code for 8-PSK; the mapper follows Figure 10.37. (b) Trellis of the code with only some of the branches shown.

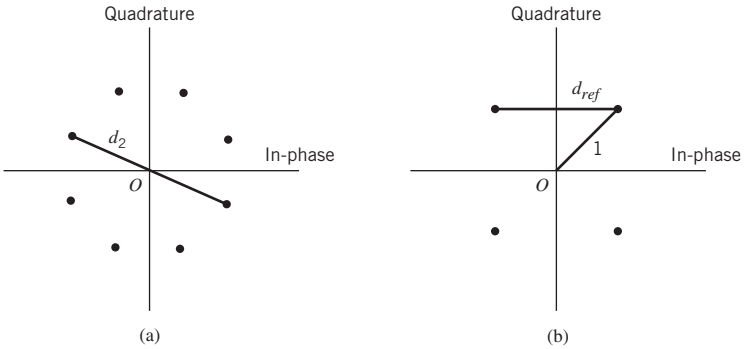


Figure 10.41 Signal-space diagrams for calculation of asymptotic coding gain of Ungerboeck 8-PSK code: (a) definition of distance d_2; (b) definition of reference distance d_{ref}.

2. From Figure 10.41b, we see that the minimum Euclidean distance of an uncoded QPSK, viewed as the frame of reference operating with the same signal energy per bit, assumes the following value:

$$d_{\text{ref}} = \sqrt{2}$$

Hence, as previously stated, the use of (10.149) yields an asymptotic coding gain of

$$10 \log_{10} 2 = 3 \text{ dB}.$$

The asymptotic coding gain achievable with Ungerboeck codes increases with the number of states in the convolutional encoder. Table 10.8 presents the asymptotic coding gain (in dB) for Ungerboeck 8-PSK codes for increasing number of states, expressed with respect to uncoded 4-PSK. Note that improvements on the order of 6 dB require codes with a very large number of states.

Table 10.8 Asymptotic coding gain of Ungerboeck 8-PSK codes, with respect to uncoded 4-PSK

Number of states	4	8	16	32	64	128	256	512
Coding gain (dB)	3	3.6	4.1	4.6	4.8	5	5.4	5.7

10.16 Turbo Decoding of Serial Concatenated Codes

In Section 10.12 we pointed out that there are two types of concatenated codes: parallel and serial. The original turbo coding scheme involved a *parallel concatenated code*, since the two encoders operate in parallel on the same set of message bits. We now turn our attention in this section to a *serial concatenation* scheme as depicted in Figure 10.42, comprised of an "outer" encoder whose output feeds an "inner" encoder. Whereas the serial concatenation idea can be traced to as early as Shannon's seminal work, the

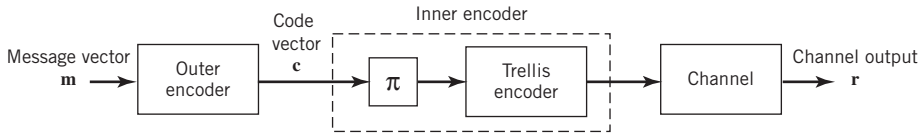


Figure 10.42 Serial concatenated codes; as usual, π denotes an interleaver.

connection with turbo coding occurred only after the parallel concatenated scheme of Berrou *et al.* (see Section 10.12) gained widespread acclaim. The iterative decoding algorithm for the serial concatenated scheme was first analyzed in detail by Benedetto and coworkers (Benedetto and Montorsi, 1996; Benedetto *et al.*, 1998); the algorithm follows a similar logic to the parallel concatenated scheme, in the form of information exchange between the two decoders as in Figure 10.43. This iterative information exchange is observed to significantly improve the overall error-correction abilities of the decoder, just as in the conventional turbo decoder. We shall review the basics of the iterative decoding algorithm in what follows in order to emphasize the common points with the iterative algorithm described in Section 10.12.

The particular interest in the serial concatenated scheme, however, becomes apparent once we recognize that the inner encoder–decoder pair need not be a conventional error-correction code, but in fact may assume more general forms that are often encountered in communication systems. A few examples may be highlighted as follows:

1. The inner encoder may in fact be a TCM stage, as studied in Section 10.15. The iterative decoding algorithm connecting the trellis-coded demodulator with the outer error-correction code leads to *turbo TCM*.[23]
2. The inner encoder may be the communication channel itself, which is of interest when the channel induces ISI. The output symbols of the channel may then be expressed as a convolution between the input symbol sequence and the channel impulse response, and the decoder operation corresponds to channel equalization (Chang and Hancock, 1966). Combining the equalizer with the outer channel decoder gives rise to *turbo equalization*.[24]

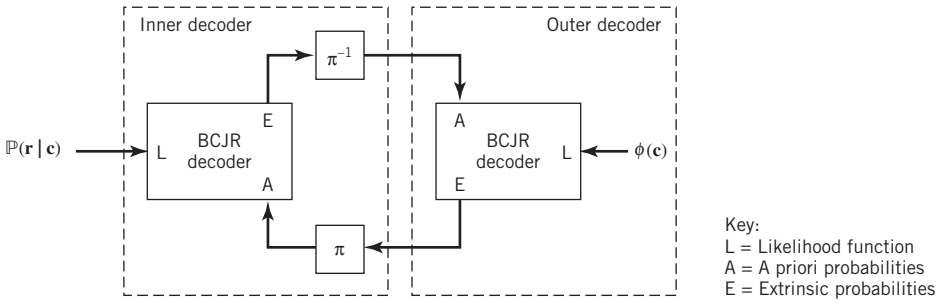


Figure 10.43 Iterative decoder structure.

3. In multi-user communication systems, the inner encoder may represent a single user's access point to a shared channel through, say, direct-sequence CDMA, in which users sharing a channel are distinguished by an assigned repetition code. The inner decoder is a multiuser detector, which aims to separate the multiple users into distinct symbol streams; when combined with the outer decoder using information exchange, a *turbo CDMA* system results.[25]

The above list is by no means exhaustive, but merely represents some of the more commonly studied variants of iterative receiver design. We will focus here on the basic iterative decoding scheme using an error-correction code for the inner encoder–decoder pair, and then briefly illustrate its applications to turbo equalization.

Serial Turbo Codes

Consider first the case in which both encoders in the serial concatenation of Figure 10.42 implement forward error-correction coding. For efficiency reasons, we assume that the outer encoder implements a systematic code, so that the codeword $\mathbf{c}$ it produces appears as follows:

$$\mathbf{c} = [\,\mathbf{b} \,\vdots\, \mathbf{m}\,] \tag{10.150}$$

in which $\mathbf{m}$ contains the k message bits and $\mathbf{b}$ contains the $n - k$ parity-check bits. By choosing a recursive systematic encoder, the corresponding decoding operation can exploit the BCJR algorithm discussed in Section 10.9.

The second, or "inner," encoder is also based on a trellis code (although not necessarily systematic) so that it, too, will admit an efficient decoder using the MAP decoding algorithm. As illustrated in Figure 10.42, the inner encoder also integrates an interleaver, denoted by π, which permutes the order of the bits in the code vector $\mathbf{c}$ prior to the second encoding operation. Without this interleaver, the serial concatenation of two trellis codes would merely give a larger-dimension trellis code having limited error-correction capabilities. The inclusion of the interleaver alters markedly the minimum distance properties of the code, and constitutes an essential ingredient in obtaining a good error-correction code.

Probabilistic Considerations

The output from the inner encoder is sent across the channel, which may be a binary symmetric channel or an AWGN channel, to produce the received vector $\mathbf{r}$. The simplest way to decode the received signal is to cascade the corresponding inner and outer decoders. A refined approach is to allow information exchange between the two decoders, to trigger the turbo effect; this idea is illustrated in Figure 10.43, and the manner of information exchange is developed in what follows.

To begin, the inner decoder aims to obtain the bitwise a posteriori probability ratios

$$\frac{\mathbb{P}(c_i = +1|\mathbf{r})}{\mathbb{P}(c_i = -1|\mathbf{r})}, \qquad i = 1, 2, \ldots, n \tag{10.151}$$

As $\mathbb{P}(c_i|\mathbf{r})$ is a marginal probability calculated from the conditional probability $\mathbb{P}(\mathbf{c}|\mathbf{r})$, the bit-wise a posteriori probability ratio may be developed into the new form

$$\frac{\mathbb{P}(c_i = +1|\mathbf{r})}{\mathbb{P}(c_i = -1|\mathbf{r})} = \frac{\sum_{\mathbf{c}:c_i = +1} \mathbb{P}(\mathbf{c}|\mathbf{r})}{\sum_{\mathbf{c}:c_i = -1} \mathbb{P}(\mathbf{c}|\mathbf{r})}$$

$$= \frac{\sum_{\mathbf{c}:c_i = +1} \mathbb{P}(\mathbf{r}|\mathbf{c})\,\mathbb{P}(\mathbf{c})}{\sum_{\mathbf{c}:c_i = -1} \mathbb{P}(\mathbf{r}|\mathbf{c})\,\mathbb{P}(\mathbf{c})}, \qquad i = 1, 2, \ldots, n \tag{10.152}$$

In the second line of (10.152), we used Bayes' rule

$$\mathbb{P}(\mathbf{c}|\mathbf{r}) = \mathbb{P}(\mathbf{r}|\mathbf{c})\mathbb{P}(\mathbf{c})/\mathbb{P}(\mathbf{r})$$

to expose the likelihood function $\mathbb{P}(\mathbf{r}|\mathbf{c})$ as well as the *a priori* probability function $\mathbb{P}(\mathbf{c})$; the term $\mathbb{P}(\mathbf{r})$ is common to the numerator and denominator, and so cancels in the ratio.

We now make the assumption that the a priori probability $\mathbb{P}(\mathbf{c})$ factors into the product of its marginals; that is,

$$\mathbb{P}(\mathbf{c}) = \mathbb{P}(c_1)\mathbb{P}(c_2)\ldots\mathbb{P}(c_n) \tag{10.153}$$

Strictly speaking, this is incorrect, since $\mathbf{c}$ contains both the message bits $\mathbf{m}$ and parity-check bits $\mathbf{b}$ from the outer encoder, and we know that the message bits determine the parity-check bits once the outer encoder is specified. The reason for invoking this assumption is to facilitate decoding via the BCJR algorithm. In particular, inserting this factorization of the *a priori* probability function into the *a posteriori* probability ratio, we may continue our development as shown by

$$\frac{\mathbb{P}(c_i = +1|\mathbf{r})}{\mathbb{P}(c_i = -1|\mathbf{r})} = \frac{\sum_{\mathbf{c}:c_i = +1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{j=1}^{n} \mathbb{P}(c_j)}{\sum_{\mathbf{c}:c_i = -1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{j=1}^{n} \mathbb{P}(c_j)}$$

$$= \underbrace{\frac{\mathbb{P}(c_i = +1)}{\mathbb{P}(c_i = -1)}}_{\text{prior ratio}} \times \underbrace{\frac{\sum_{\mathbf{c}:c_i = +1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{P}(c_j)}{\sum_{\mathbf{c}:c_i = -1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{P}(c_j)}}_{\text{extrinsic information ratio}}, \qquad i = 1, 2, \ldots, n \tag{10.154}$$

We obtained the second line in (10.154) by noting that each term in the numerator contains the factor $\mathbb{P}(c_i = +1)$ and, similarly, each term in the denominator contains the factor $\mathbb{P}(c_i = -1)$, hence the reason for the prior ratio factoring out of the expression. The remaining term is the extrinsic information ratio for bit c_i from the inner decoder.

To facilitate passing information to the outer decoder, we may interpret each extrinsic information ratio from the inner decoder as the probability ratio of an auxiliary probability mass function $T(\mathbf{c})$ that fulfills two properties:

- The probability mass function $T(\mathbf{c})$ factors into its bitwise marginal functions according to
$$T(\mathbf{c}) = T_1(c_1)T_2(c_2) \ldots T_n(c_n) \tag{10.155}$$
- The bitwise marginal evaluations, each, sum to one, $T_i(+1) + T_i(-1) = 1$, and are chosen such that their ratios match the extrinsic information ratios from the inner decoder:

$$\frac{T_i(c_i = +1)}{T_i(c_i = -1)} = \frac{\displaystyle\sum_{\mathbf{c}:c_i = +1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{P}(c_j)}{\displaystyle\sum_{\mathbf{c}:c_i = -1} \mathbb{P}(\mathbf{r}|\mathbf{c}) \prod_{\substack{j=1 \\ j \neq i}}^{n} \mathbb{P}(c_j)}, \qquad i = 1, 2, \ldots, n \tag{10.156}$$

Now, we note that by natural taking logarithms, the *log extrinsic ratio* becomes

$$\ln[T_i(+1)/T_i(-1)] = L_{\mathrm{p}}(c_i) - L_{\mathrm{a}}(c_i) \tag{10.157}$$

where $L_{\mathrm{p}}(c_i)$ is the *log posterior ratio* and $L_{\mathrm{a}}(c_i)$ is the *log prior ratio*.[26]

Next, we note that the outer decoder does not have the usual channel likelihood evaluations available, but must instead take information from the inner decoder. While many possibilities in this direction may be envisaged, a successful iterative decoding algorithm results by replacing the *a posteriori* probability according to

$$\mathbb{P}(\mathbf{c}|\mathbf{r}) \leftarrow \phi(\mathbf{c})T(\mathbf{c}) \tag{10.158}$$

in which $\phi(\mathbf{c})$ is the indicator function for the outer code, that is

$$\phi(\mathbf{c}) = \begin{cases} 1, & \text{if } \mathbf{c} \text{ is a code vector} \\ 0, & \text{otherwise} \end{cases} \tag{10.159}$$

We may think of the function $\phi(\mathbf{c})$ as replacing the conventional channel likelihood function $\mathbb{P}(\mathbf{r}|\mathbf{c})$, since it vanishes whenever $\mathbf{c}$ is not a code vector, and $T(\mathbf{c}) = T_1(c_1) \ldots T_n(c_n)$ as replacing the *a priori* probability on each bit, since it factors into the product of its marginals. The conventional posterior probability ratio for the outer decoder is thus replaced with

$$\frac{\mathbb{P}(c_i = +1|\mathbf{r})}{\mathbb{P}(c_i = -1|\mathbf{r})} = \frac{\displaystyle\sum_{\mathbf{c}:c_i=+1} \phi(\mathbf{c}) \prod_{j=1}^{n} T_j(c_j)}{\displaystyle\sum_{\mathbf{c}:c_i=-1} \phi(\mathbf{c}) \prod_{j=1}^{n} T_j(c_j)}$$

$$= \underbrace{\frac{T_i(c_i = +1)}{T_i(c_i = -1)}}_{\text{prior ratio}} \underbrace{\frac{\displaystyle\sum_{\mathbf{c}:c_i=+1} \phi(\mathbf{c}) \prod_{\substack{j=1 \\ j\neq i}}^{n} T_j(c_j)}{\displaystyle\sum_{\mathbf{c}:c_i=-1} \phi(\mathbf{c}) \prod_{\substack{j=1 \\ j\neq i}}^{n} T_j(c_j)}}_{\text{extrinsic information ratio}}, \quad i = 1, 2, \ldots, n \tag{10.160}$$

In (10.160) the second line is obtained upon noting that each term in the numerator (denominator) contains a factor $T_i(c_i = +1)$ ($T_i(c_i = -1)$). This separates the "pseudo-prior" ratio from the outer decoder's extrinsic information ratio.

Now, to couple the information from the outer decoder back to the inner decoder, we map the outer decoder's extrinsic information values to a probability mass function $U(\mathbf{c})$ which, akin to $T(\mathbf{c})$ introduced above, fulfills two properties:

1. The probability mass function $U(\mathbf{c})$ factors into the product of its bitwise marginal functions according to
$$U(\mathbf{c}) = U_1(c_1)\, U_2(c_2) \ldots U_n(c_n) \tag{10.161}$$
2. The bitwise marginal evaluations each sum to one, $U_i(+1) + U_i(-1) = 1$, and are chosen such that their ratios match the extrinsic information ratios for the outer decoder:
$$\frac{U_i(c_i = +1)}{U_i(c_i = -1)} = \frac{\displaystyle\sum_{\mathbf{c}:c_i=+1} \phi(\mathbf{c}) \prod_{\substack{j=1 \\ j\neq i}}^{n} T_j(c_j)}{\displaystyle\sum_{\mathbf{c}:c_i=-1} \phi(\mathbf{c}) \prod_{\substack{j=1 \\ j\neq i}}^{n} T_j(c_j)}, \quad i = 1, 2, \ldots, n \tag{10.162}$$

The marginal probability functions $U_i(c_i)$ then replace the *a priori* probability values $\mathbb{P}(c_i)$ in the inner decoder, and the procedure iterates, thus defining the turbo decoder. In this fashion, we say the following:

> The turbo decoder for serially concatenated codes follows the same logic as for parallel concatenated codes, in that the extrinsic information values furnished from one decoder replace the a priori probability values required of the other.

Turbo Equalization

In high-speed communication systems, further signal degradation can come from multipath artifacts in wireless environments, or reflection artifacts due to impedance mismatches in wireline systems. When such degradations have a temporal duration commensurate with the symbol period, the signal impinging on the receiver at a given sample instant is the composite influence of successive transmitted symbols, giving rise to ISI. Its severity worsens as the symbol period diminishes relative to the "delay spread" of the system, meaning that higher rate data systems must contend with ISI as a significant channel distortion mechanism.

In such cases, the received symbol r_i at sample instant i is a weighted combination of a set of successive transmitted symbols, according to

$$r_i = \sum_{k=0}^{L-1} h_k s_{i-k} + \zeta_i \tag{10.163}$$

Here, ζ_i is the additive background noise, $\{s_i\}$ is the transmitted symbol sequence obtained by interleaving the bit sequence $\{c_i\}$ (and possibly followed by symbol mapping if using TCM), and $\{h_0, h_1, ..., h_{L-1}\}$ is the channel impulse response of length L.

If we consider a simple case in which each s_i is antipodal ($s_i = \pm 1$) and $k = 0, 1, 2$, we see that the noise-free channel outputs can be obtained through the trellis graph of Figure 10.44: The transitions are determined by whether the input symbol is $s_i = +1$ or $s_i = -1$, while the noise-free outputs are drawn from a finite set comprised of sums and differences of the channel impulse response coefficients. Thus, a convolutional channel which induces ISI may itself be viewed as a trellis code, and the BCJR algorithm may be applied directly to estimate the a posteriori probabilities of the transmitted symbols, and thus of the codeword bits $\{c_i\}$. The new result is that we have the traditional MAP equalizer.

A turbo equalizer results upon noting that the convolutional channel and its MAP equalizer may be viewed as the inner encoder–decoder pair of a serial cascade scheme albeit one dictated by the communication channel and thus beyond the designer's control. The outer encoder may again be chosen as a recursive systematic trellis code, whose decoder is coupled with the MAP equalizer in precisely the same manner: the extrinsic probabilities from one decoder are used in place of the a priori probabilities of the other, resulting in an iterative decoding and equalization scheme.

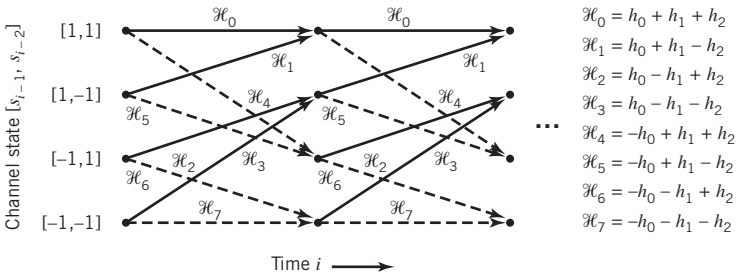


Figure 10.44 Trellis graph for a three-tap channel model, with transition branches listing the noise-free channel outputs. Solid transitions occur when the channel input is $s_i = +1$; dashed transitions occur when $s_i = -1$.

10.17 Summary and Discussion

In this rather long chapter we studied error-control coding techniques that have established themselves as indispensable tools for reliable digital communication over noisy channels. The effect of errors occurring during transmission is reduced by adding redundancy to the data prior to transmission in a controlled manner. The redundancy is used to enable a decoder in the receiver to detect and correct errors.

Regardless of how they are designed, error-control coding schemes rely on Shannon's 1948 landmark paper, particularly the celebrated *coding theorem*, which asserts the following statement:

> Given the proper strategy for encoding the incoming message bits, unavoidable errors produced by a noisy channel can be corrected without having to sacrifice the rate of data transmission.

The coding theorem was discussed in Chapter 5 on information theory. Restating it here, for the last time in the final chapter of the book, is intended to emphasize the importance of the theorem, which will last forever.

In a historical context, error-control coding schemes may be divided into two broadly defined families:

1. *Legacy Codes*

   As the name would imply, the family of legacy codes embodies several kinds of *linear codes* that originated in 1950 and, in the course of three decades or so, broadened its scope in depth as well as breadth. A distinctive feature of legacy codes is that of exploiting *abstract algebraic structures* built into their design in different ways and increasing mathematical abstraction.

   Specifically, legacy codes cover the following four schemes:

   a. *Linear block codes*, the first kind of which were described independently by Golay in 1949 and Hamming in 1950. Hamming codes are simple to construct and just as easy to decode using a look-up table based on the notion of syndrome. It is because of their computational simplicity and the ability to operate at high data rates, that we find that Hamming codes are widely used in digital communications.

   b. *Cyclic codes*, which form an importance subclass of linear block codes. Indeed, many of the block codes used in practice are cyclic codes for two compelling reasons:

      - The use of linear feedback shift registers for encoding and syndrome computation.
      - The inherent algebraic structure used to develop various practical decoding algorithms.

      Examples of cyclic codes include Hamming codes for digital communications, and most importantly, Reed–Solomon codes for combatting both random and burst errors encountered in difficult environments such as deep-space communications and compact discs.

   c. *Convolutional codes*, which distinguish themselves from linear block codes in the use of *memory* in the form of a finite-state shift register for implementing the

encoder. For decoding convolutional codes, the Viterbi algorithm (based on maximum likelihood decoding) is commonly used; this algorithm is designed to minimize the symbol-error rate on a symbol-by-symbol basis.

d. *Trellis coded modulation*, which distinguishes itself from linear convolutional codes in the combined use of encoding and modulation in a single entity. The next result of so doing is the achievement of significant coding gains over conventional uncoded multilevel modulation schemes without having to sacrifice bandwidth efficiency in decoding.

2. *Probabilistic Compound Codes*

This second family of error-control coding schemes is exemplified by turbo codes and LDPC codes, which, as different as they are from each other, share a common property:

Random encoding of a linear block kind.

To be more specific, in their own individual ways, they are both revolutionary:

In practical terms, turbo codes and LDPC codes have made it possible to achieve coding gains on the order of 10 dB, thereby approaching the Shannon limit not attainable by the legacy codes.

Moreover, in some specialized cases, very long rate-1/2 irregular LPDC codes have approached the Shannon limit to within 0.0045 dB for AWGN channels, which is truly remarkable (Chung *et al.*, 2001).

These impressive coding gains have been exploited to dramatically extend the range of digital communication receivers, substantially increase the bit rates of digital communication systems, or significantly decrease the transmitted signal energy per symbol. The benefits have significant implications for the design of wireless communications and deep-space communications, just to mention two important applications of digital communications. Indeed, turbo codes have already been standardized for use on both of these applications.

One last comment is in order: Turbo codes have not only impacted digital communications in the different ways just described, but the turbo decoding paradigm has also impacted applications outside the traditional scope of error-control coding. One such example is that of *turbo equalization*, briefly described in Section 10.16. Indeed, we may justifiably say the following as the last statement of the chapter:

The turbo-decoding paradigm, by virtue of its broadly defined scope of applications, stands out as one of the ground-breaking achievements in modern telecommunications.

Problems

Soft-Decision Coding

10.1 Consider a binary input Q-ary output discrete memoryless channel. The channel is said to be symmetric if the channel transition probability $p(j|i)$ satisfies the condition

$$p(j|0) = p(Q-1-j|1), \qquad j = 0, 1, \ldots, Q-1$$

Suppose that the channel input bits 0 and 1 are equally likely. Show that the channel output symbols are also equally likely; that is,

$$p(j) = \frac{1}{Q}, \qquad j = 0, 1, \ldots, Q-1$$

10.2 Consider the quantized demodulator for binary PSK signals shown in Figure 10.3a. The quantizer is a four-level quantizer, normalized as in Figure P10.2. Evaluate the transition probabilities of the binary input–quarternary output discrete memoryless channel so characterized. Hence, show that it is a symmetric channel. Assume that the transmitted signal energy per bit is E_b and the AWGN has zero mean and power spectral density $N_0/2$.

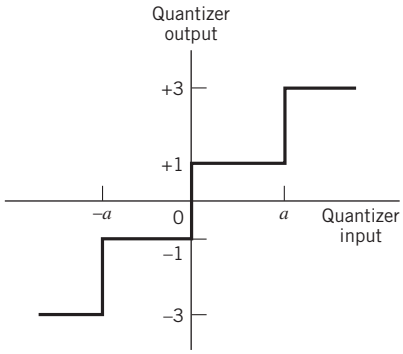


Figure P10.2

10.3 Consider a binary input AWGN channel, in which the bits 1 and 0 are equally likely. The bits are transmitted over the channel by means of phase-shift keying. The code symbol energy is E and the AWGN has zero mean and power spectral density $N_0/2$. Show that the channel transition probability is given by

$$p(y|0) = \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(y + \sqrt{\frac{2E}{N_0}}\right)^2\right], \qquad -\infty < y < \infty$$

Linear Block Codes

10.4 Hamming codes are said to be perfect single-error correcting codes. Justify the fact that Hamming codes are *perfect*.

10.5 Consider the following statement:

> An (n, k) code is often said to be a good code.

Explain the conditions under which this statement is justified.

10.6 In a *repetition code*, a single message bit is encoded into a block of identical bits to produce an (n, 1). Considering the (5, 1) repetition code, evaluate the syndrome for:

a. All five possible single-error patterns.

b. All ten possible double-error patterns.

10.7 In a *single-parity-check code*, a single parity bit is appended to a block of k message bits (m_0, m_1, …, m_{k-1}). The single parity bit b_0 is chosen so that the codeword satisfies the *even parity rule*:

$$m_0 + m_1 + \ldots + m_{k-1} + b_{k-1} = 0, \qquad \text{mod } 2$$

For $K = 3$, set up the 2^k possible codewords in the code defined by this rule.

10.8 Compare the parity-check matrix of the (7,4) Hamming code considered in Example 1 with that of a (4,1) repetition code.

10.9 Consider the (7,4) Hamming code of Example 1. The generator matrix $\mathbf{G}$ and the parity-check matrix $\mathbf{H}$ of the code are described in that example. Show that these two matrices satisfy the condition

$$\mathbf{H}\mathbf{G}^{\mathrm{T}} = \mathbf{0}$$

10.10 a. For the (7,4) Hamming code described in Example 1, construct the eight codewords in Hamming's dual code.

b. Find the minimum distance of the dual code determined in part a.

Linear Cyclic Codes

10.11 For an application that requires error detection *only*, we may use a *nonsystematic* code. In this problem, we explore the generation of such a cyclic code. Let $\mathbf{g}(X)$ denote the generator polynomial, and $\mathbf{m}(X)$ denote the message polynomial. We define the code polynomial $\mathbf{c}(X)$ simply as

$$\mathbf{c}(X) = \mathbf{m}(X)\mathbf{g}(X)$$

Hence, for a given generator polynomial, we may readily determine the codewords in the code. To illustrate this procedure, consider the generator polynomial for a (7,4) Hamming code:

$$\mathbf{g}(X) = 1 + X + X^3$$

Determine the 16 codewords in the code, and confirm the nonsystematic nature of the code.

10.12 The polynomial $1 + X^7$ has $1 + X + X^3$ and $1 + X^2 + X^3$ as primitive factors. In Example 10.2, we used $1 + X + X^3$ as the generator polynomial for a (7,4) Hamming code. In this problem, we consider the adoption of $1 + X^2 + X^3$ as the generator polynomial. This should lead to a (7,4) Hamming code that is different from the code analyzed in Example 2. Develop the encoder and syndrome calculator for the generator polynomial:

$$\mathbf{g}(X) = 1 + X^2 + X^3$$

Compare your results with those in Example 2.

10.13 Consider the (7,4) Hamming code defined by the generator polynomial

$$\mathbf{g}(X) = 1 + X + X^3$$

The codeword 0111001 is sent over a noisy channel, producing the received word 0101001 that has a single error. Determine the syndrome polynomial $\mathbf{s}(X)$ for this received word, and show that it is identical to the error polynomial $\mathbf{e}(X)$.

10.14 The generator polynomial of a (15,11) Hamming code is defined by

$$\mathbf{g}(X) = 1 + X + X^4$$

Develop the encoder and syndrome calculator for this code, using a systematic form for the code.

10.15 Consider the (15,4) maximal-length code that is the dual of the (15,11) Hamming code of Problem 10.14.

Find the generator polynomial $\mathbf{g}(X)$; hence, determine the output sequence assuming the initial state 0001. Confirm the validity of your result by cycling the initial state through the encoder.

10.16 Consider the (31,15) Reed–Solomon code.

a. How many bits are there in a symbol of the code?

b. What is the block length in bits?

c. What is the minimum distance of the code?

d. How many symbols in error can the code correct?

Convolutional Codes

10.17 A convolutional encoder has a single-shift register with two states, (i.e., constraint length $\nu = 3$), three modulo-2 adders, and an output multiplexer. The generator sequences of the encoder are as follows:

$$g^{(1)} = (1, 0, 1)$$
$$g^{(2)} = (1, 1, 0)$$
$$g^{(3)} = (1, 1, 1)$$

Draw the block diagram of the encoder.

10.18 Consider the rate $r = 1/2$, constraint length $\nu = 2$ convolutional encoder of Figure P10.18. The code is systematic. Find the encoder output produced by the message sequence 10111…

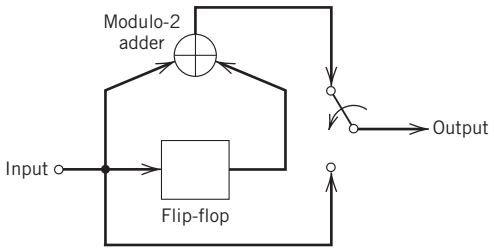


Figure P10.18

10.19 Figure P10.19 shows the encoder for a rate $r = 1/2$, constraint length $\nu = 4$ convolutional code. Determine the encoder output produced by the message sequence 10111….

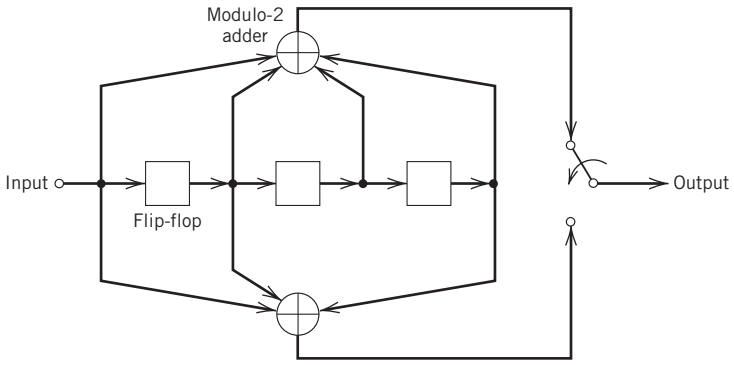


Figure P10.19

10.20 Consider the encoder of Figure P10.20 for a rate $r = 2/3$, constraint length $\nu = 2$ convolutional code. Determine the code sequence produced by the message sequence 10111….

10.21 Construct the code tree for the convolutional encoder of Figure P10.19. Trace the path through the tree that corresponds to the message sequence 10111…, and compare the encoder output with that determined in Figure P10.19.

10.22 Construct the trellis graph for the encoder of Figure P10.19, assuming a message sequence of length 5. Trace the path through the trellis corresponding to the message sequence 10111…. Compare the resulting encoder output with that found in Problem 10.19.

10.23 Construct the state graph for the encoder of Figure P10.19. Starting with the all-zero state, trace the path that corresponds to the message sequence 10111… and compare the resulting code sequence with that determined in Problem 10.19.

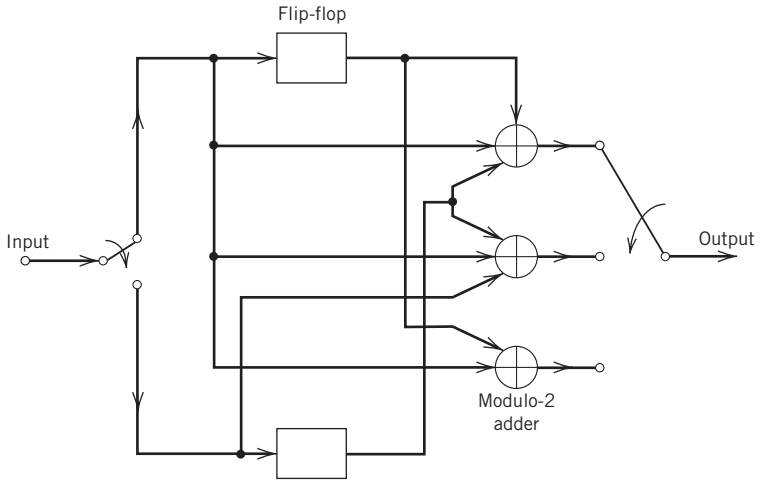


Figure P10.20

10.24 Consider the encoder of Figure 10.13.

a. Construct the state graph for this encoder.

b. Starting from the all-zero state, trace the path that corresponds to the message sequence 10111… Compare the resulting sequence with that determined in Problem 10.19.

10.25 By viewing the minimum shift keying (MSK) scheme as a finite-state machine, construct the trellis diagram for the MSK. (A description of the MSK is presented in Chapter 7).

10.26 Consider a rate-1/2, constraint length-7 convolutional code with free distance $d_{\text{free}} = 10$. Calculate the asymptotic coding gain for the following two channels:

a. Binary symmetric channel.

b. Binary input AWGN channel.

10.27 The transform-domain generator matrix $\mathbf{G}(D)$ of an RSC encoder includes ratios of polynomials in the delay variable D, whereas, in the case of a nonrecursive convolutional encoder $\mathbf{G}(D)$ is simply a polynomial in D. Justify the $\mathbf{G}(D)$ for these two cases.

10.28 Consider an eight-state RSC encoder, the generator matrix of which is given by

$$\mathbf{g}(D) = \left[1, \frac{1 + D + D^2 + D^3}{1 + D + D^2}\right]$$

where D is the delay variable.

a. Construct the block diagram of this encoder.

b. Formulate the parity-check equation that embodies all the message as well as parity-check bits in the time domain.

10.29 Describe the similarities and differences between traditional encoders and RSC encoders.

The Viterbi Algorithm

10.30 The trellis diagram of a rate-1/2, constraint length-3 convolutional code is shown in Figure P10.30. The all-zero sequence is transmitted and the received sequence is 1000100000…. Using the Viterbi decoding algorithm, compute the decoded sequence.

10.31 In Section 10.8, we described the Viterbi algorithm for maximum likelihood decoding of a convolutional code. Another application of the Viterbi algorithm is for maximum likelihood

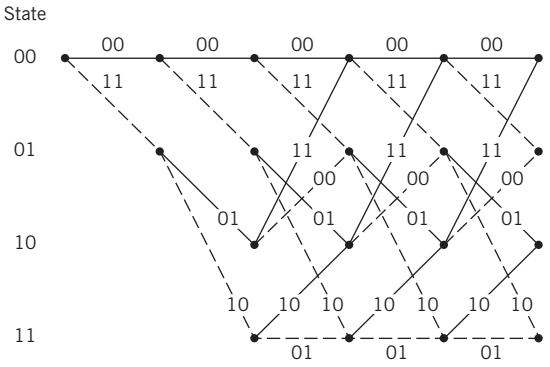


Figure P10.30

demodulation of a received sequence corrupted by ISI due to a dispersive channel. Figure P10.31 shows the trellis diagram for ISI, assuming a binary data sequence. The channel is discrete, described by the finite impulse response (1, 0, 1). The received sequence is (1.0, –0.3, –0.7, …). Use the Viterbi algorithm to determine the maximum likelihood decoded version of the sequence.

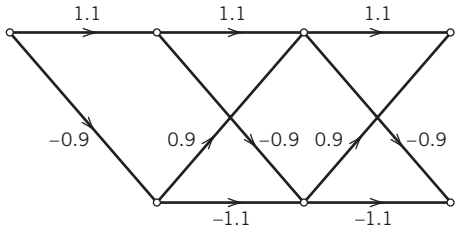


Figure P10.31

10.32 In dealing with channel equalization, a primary objective is to undo the convolution performed by a linear communication channel on the source signal. This task is well suited for the Viterbi equalizer functioning as a channel equalizer.

a. What is the underlying idea in the Viterbi algorithm that ties channel equalization and convolutional decoding together?

b. Suppose that the channel has memory defined by 2^l, where l is an integer.

What is the required length of the window for the Viterbi equalizer? Justify your answers for both parts a and b of the question.

The MAP Algorithm

10.33 Refer back to (10.92), where

$$A_j = \left(\frac{\frac{1}{2}\exp L_a(-m_j)}{1 + \exp(L_a(-m_j))} \right), \qquad j = 0, 1, 2, \ldots$$

Verify that the factor A_j is a constant, regardless of whether the message bit m_j is –1 or +1.

10.34 In Example 7, we used the max-log-MAP algorithm to decode the three message bits at the output of the RSC encoder depicted in Figure 10.23. The computations were obtained using a Matlab code. Parts a and b of the figure pertain to the block diagram of the encoder and its trellis, respectively. The five computational steps described therein apply equally well to the log-MAP algorithm.

a. Repeat Example 7, but this time develop a Matlab code for using the log-MAP algorithm to compute the decoded binary output of the encoder in Figure 10.23a.

b. Confirm the decoded binary output produced in part a by performing the five tasks involved in the log-MAP algorithm, doing all the computations in the traditional way.

c. Compare the decoded output product using the log-MAP algorithm with that reported in Example 7.

Comment on your results.

10.35 Figure P10.35 depicts two processing stages involved in the MAP decoding algorithm. The first stage is a convolutional encoder of rate, $r = k/n$, producing the code vector $\mathbf{c}$ in response to the message vector $\mathbf{m}$. The second stage is a mapper, represented by binary PSK. The signal energy per message bit at the encoder input is denoted by E_b; the noise spectral density of the AWGN channel is $N_0/2$.

Let E_s denote the signal energy per symbol transmitted by the binary PSK mapper. Show that the SNR measured at the channel output is given by

$$(\text{SNR})_{\text{out}} = \frac{E_s}{N_0} = r\left(\frac{E_b}{N_0}\right)$$

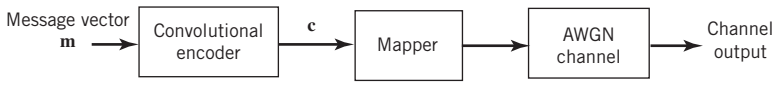


Figure P10.35

10.36 Consider an AWGN channel with unquantized output, assuming the binary code maps $0 \rightarrow -1$ and $1 \rightarrow +1$. Given a received signal $r_j^{(0)}$ at the channel output in response to a transmitted message bit m_j before decoding, the a posteriori L-value is defined by

$$L(m_j|r_j^{(0)}) = \ln\left(\frac{\mathbb{P}(m_j = +1|r_j^{(0)})}{\mathbb{P}(m_j = -1|r_j^{(0)})}\right)$$

a. Show that

$$L(m_j|r_j^{(0)}) = -\frac{E_s}{N_0}[(r_j^{(0)}-1)^2 - (r_j^{(0)}+1)^2] + \ln\left(\frac{\mathbb{P}(m_j = +1)}{\mathbb{P}(m_j = -1)}\right)$$

where E_s is the transmitted signal energy per encoded symbol.

b. The channel reliability factor is defined by the following formula, assuming that both m_j and $r_j^{(0)}$ are normalized by the factor $\sqrt{E_s}$ by

$$L_c = 4E_s/N_0$$

where E_s/N_0 is the channel output SNR. Hence, show that

$$L(m_j|r_j^{(0)}) = L_c r_j^{(0)} + L_a(m_j)$$

where $L_a(m_j)$ is the a priori probability of message bit m_j.

10.37 In this problem, we expand on Problem 10.36 by considering a binary fading wireless channel, where the channel noise is additive, white, and Gaussian. As in Problem 10.36, start with the log-

likelihood ratio of a transmitted message bit m_j conditioned on the corresponding matching filtered output r_j at time-unit j:

$$L(m_j|r_j) = \ln\left(\frac{\mathbb{P}(m_j = +1|r_j)}{\mathbb{P}(m_j = -1|r_j)}\right)$$

Let a denote the fading amplitude, which distinguishes this problem from Problem 10.36.

a. Show that

$$L(m_j|r_j) = L_c r_j + L(m_j)$$

where

$$L(m_j) = \ln\left(\frac{\mathbb{P}(m_j = +1)}{\mathbb{P}(m_j = -1)}\right)$$

and

$$L_c = \frac{4aE_s}{N_0}$$

is the modified channel reliability factor.

b. For statistically independent transmissions as in dual diversity, show that the log-likelihood ratio takes the expanded form:

$$L(m_j|r_j^{(1)}, r_j^{(2)}) = L_c^{(1)} + L_c^{(2)} + L(m_j)$$

where $L_c^{(1)}$ and $L_c^{(2)}$ denote the channel reliability factors for the two simultaneous transmissions of bit m_j as in dual diversity. Given this result, comment on the benefit gained by the use of diversity.

Turbo Codes

10.38 Let $r_c^{(1)} = p/q_1$ and $r_c^{(2)} = p/q_2$ be the code rates of RSC encoders 1 and 2 in the turbo encoder of Figure 10.26. Find the code rate of the turbo code.

10.39 The feedback nature of the constituent codes in the turbo encoder of Figure 10.26 has the following implication: a single bit error corresponds to an infinite sequence of channel errors. Illustrate this using a message sequence consisting of symbol 1 followed by an infinite number of symbols 0.

10.40 Consider the following generator matrices for rate-1/2 turbo codes:

$$\text{4-state encoder: } \mathbf{g}(D) = \left[1, \frac{1 + D + D^2}{1 + D^2}\right]$$

$$\text{8-state encoder: } \mathbf{g}(D) = \left[1, \frac{1 + D^2 + D^3}{1 + D + D^2 + D^3}\right]$$

$$\text{16-state encoder: } \mathbf{g}(D) = \left[1, \frac{1 + D^4}{1 + D + D^2 + D^3 + D^4}\right]$$

a. Construct the block diagram for each one of these RSC encoders.

b. Set up the parity-check equation associated with each encoder.

10.41 Turbo decoding relies on the feedback of extrinsic information. The fundamental principle adhered to in the turbo decoder is to avoid feeding a decoding state information that stems from the constituent decoder itself. Explain the justification for this principle in conceptual terms.

10.42 Suppose a communication receiver consists of two components: a demodulator and a decoder. The demodulator is based on a Markov model of the combined modulator and channel, and the decoder

is based on a Markov model of a forward error-correction code. Discuss how the turbo principle may be applied to construct a joint demodulator–decoder for this system.

10.43 Summarize the properties/attributes of turbo codes by expanding on the following six issues:

a. Structural composition of the turbo encoder and decoder.

b. Improvement in the speed of decoding attributed to the two constituent decoders at the expense of increased computational complexity.

c. Similarity of turbo decoding to the use of feedback in nonlinear control theory.

d. Feeding extrinsic information from constituent decoder 1 to constituent decoder 2, back and forth, thereby maintaining statistical independence between the bits from one iteration to the next.

e. Typical termination of the turbo decoding process after a relatively small number of iterations, somewhere in the range of 10 to 20.

f. Relatively small degradation in decoding performance of the Max-log-MAP algorithm in the order of 0.5 dB, compared with the MAP algorithm.

10.44 Present a comparative evaluation of convolutional codes and turbo codes in terms of the encoding and decoding strategies as well as other matters that pertain to signaling over wireless communications. Specifically, address the following issues in the comparative evaluation:

a. Encoding

b. Decoding

c. Fading wireless channels

d. Latency (i.e., delay incurred in transmission over the channel).

10.45 Referring back to the eight-state Ungerboeck 8-PSK of Figure 10.40, show that the asymptotic coding gain of this code is 3.5; see Table 10.8.

LDPC Codes

10.46 The generator polynomial of the (7,8) cyclic maximal-length code is given by

$$\mathbf{g}(X) = 1 + X + X^2 + X^4$$

Show that this code is an LDPC code by constructing its Tanner graph.

10.47 Consider the (7,4) cyclic Hamming code, whose generator polynomial is given by

$$\mathbf{g}(X) = 1 + X + X^3$$

Construct the Tanner graph of this code, demonstrating that it is another example of an LDPC code.

10.48 The expanded version of the cyclic Hamming code is obtained as follows. If $\mathbf{H}$ is parity-check matrix of the cyclic Hamming code, then the parity-check matrix of its extended version is defined by

$$\mathbf{H}' = \left[\begin{array}{ccc:c} 1 & 1 & 1 & 1 \\ \multicolumn{3}{c:}{\multirow{3}{*}{$\mathbf{H}$}} & 0 \\ & & & \vdots \\ & & & 0 \end{array}\right]$$

$$= \left[\begin{array}{ccccccc:c} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{array}\right]$$

whereby the distance between every pair of codewords in the extended code is now even.

Construct the Tanner graph of the extended cyclic Hamming code (8, 4).

10.49 In light of the linear cyclic codes considered in Problems 10.46 to 10.48, comment on the relationship between this class of codes and LDPC codes.

10.50 In Note 20, we introduced the idea of rateless codes, emphasizing the relationship that exists between the new class of codes and LDPC codes. Which features distinguish rateless codes from LDPC codes?

10.51 Develop a list comparing LDPC codes with turbo codes.

Notes

1. Feedforward error correction (FEC) relies on the controlled use of redundancy in the transmitted codeword for both the *detection and correction* of errors incurred during the course of transmission over a noisy channel. Irrespective of whether the decoding of the received codeword is successful, no further processing is performed at the receiver. Accordingly, channel coding techniques suitable for FEC require only a *one-way link* between the transmitter and receiver.

There is another approach known as *automatic-repeat request* (ARQ) for solving the error-control problem. The underlying philosophy of ARQ is quite different from that of FEC. Specifically, ARQ uses redundancy merely for the purpose of *error detection*. Upon the detection of an error in a transmitted codeword, the receiver requests a repeat transmission of the corrupted codeword, which necessitates the use of a *return path* (i.e., a feedback channel from the receiver to the transmitter).

For a comprehensive treatment of error-control coding, see Lin and Costello (2004) and Moon (2005).

2. In medicine, the term *syndrome* is used to describe a pattern of symptoms that aids in the diagnosis of a disease. In coding, the error pattern plays the role of the disease and parity-check failure that of a symptom. This use of syndrome was coined by Hagelbarger (1959).

3. The first error-correcting codes, known as Hamming codes, were invented by Hamming at about the same time as the conception of information theory by Shannon; for details, see the classic paper by Hamming (1950).

4. Maximal-length codes, also referred to as m-sequences, are discussed further in Appendix J; they provide the basis for pseudo-noise (PN) sequences, which play a key role in the study of spread spectrum signals in Chapter 9.

5. Reed–Solomon codes are so named in honor of their originators; see their classic paper (Reed and Solomon, 1960).

The book edited by Wicker and Bhargava (1994) contains an introductory chapter on Reed–Solomon codes; a historical overview of the codes written by Reed and Solomon themselves; and chapters on the applications of Reed–Solomon codes to exploration of the solar system, the compact disc, automatic repeat-request protocols, and spread-spectrum multiple-access communications.

In a historical context, Reed–Solomon codes are a subclass of the Bose–Chaudhuri and Hocquenghem (BCH) codes that represent a large class of powerful random error-correcting cyclic codes. However, it is important to recognize that the Reed–Solomon codes were discovered independently of the pioneering works by Hocquenghem (1959) and Bose and Ray-Chaudhuri (1960).

For detailed mathematical treatments of binary BCH codes and nonbinary BCH codes with emphasis on Reed–Solomon codes, see Chapters 6 and 7 of the book by Li and Costello (2004), respectively.

6. Convolutional codes were invented by Elias (1955) as an alternative to linear block codes. The aim of that classic paper was to formulate a new class of codes with as much structure as practically feasible without loss of performance in using them over binary symmetric and AWGN channels.

7. In a classic paper, Viterbi (1967) proposed a decoding algorithm for convolutional codes that has become known as the Viterbi algorithm. The algorithm was recognized by Forney (1972, 1973) to be a maximum likelihood decoder. Readable accounts of the *Viterbi algorithm* are presented in the book by Lin and Costello (2004).

The discussion presented in this chapter is confined to the classical Viterbi algorithm involving hard decisions. For iterative decoding applications with soft outputs, Hagenauer and Hoeher (1989) described the so-called *soft-output Viterbi algorithm (SOVA)*. For detailed discussion of both versions of the Viterbi algorithm, the reader is referred to Lin and Costello (2004).

8. For details of the evaluation of asymptotic coding gain for binary symmetric and binary-input AWGN channels, see Lin and Costello (2004).

9. At first sight, derivation of the MAP decoding algorithm appears to be complicated. In reality, however, the derivation is straight forward, given knowledge of probability theory. The derivation presented herein follows the book by Lin and Costello (2004).

10. For detailed mathematical description of the log-MAP algorithm, the reader is referred to the book (Lin and Costello, 2004).

11. Costello and Forney (2007) surveyed the evolution of coding on the road to channel capacity for AWGN channels over the course of 50 years, going back to the classic paper of Claude Shannon (1948). Proceeding in a stage-by-stage manner through the history of codes over band-limited channels, they came to the paper written by Berrou, *et al*., (1993) on turbo codes, which was presented at the IEEE International Communications Conference (ICC) in Geneva, Switzerland; therein, the achievement of a performance near the Shannon limit with modest decoding complexity was claimed by its three co-authors. Listening to this claim, the coding research community at the conference were stunned, with comments being whispered to the effect: "It cannot be true; they must have made a 3 dB error." However, in the course of a year, the claims reported by Berrou were confirmed by various laboratories. And, with it, the turbo revolution was launched.

12. The plots presented in Figure 10.26 follow those in the book by Frey (1998).

13. Example 9 is based on the Ph.D. thesis by Li (2011), with useful comments by Maunder (2012).

14. For the case when the interleaver's length is high, as in the simulation results plotted in the BER chart of Figure 10.31, finding the floor region can be extremely time consuming. Indeed, it is for this reason that the number of iterations in Figure 10.31 was limited to four.

15. The averaging method emanated from the Ph.D. thesis of Land (2005); this method is also described in Land *et al*. (2004). The first reference to the averaging method was made under "private communication" in Hagenauer (2004).

16. The LDPC codes, introduced by Gallager (1960, 1963), were dormant for more than three decades. Lack of interest in these codes in the 1960s and 1970s may well have been attributed to the fact that the computers of those days were not powerful enough to cope with LDPC codes of long block lengths. But, reflecting back over the 1980s, it is surprising to find that lack of interest in LDPC codes by the coding community persisted for all those years except for a single paper: Tanner (1981) proposed a graphical representation for studying the structure of Gallager's LDPC codes (as well as other codes) for the purpose of iterative decoding; such graphs are now called the *Tanner graphs*. In any event, it was not until the introduction of turbo codes and iterative decoding by Berrou *et al.* (1993) that interest in LDPC codes was rekindled. Two factors were responsible for this rekindled interest (Hanzo, 2012):

- the protection of turbo codes by a patent and unwillingness of industry to pay royalties, and
- the rediscovery of LDPC codes by MacKay and Neal (1996; MacKay, 1999).

And with it, the LDPC rediscovery was launched.

17. In a historical context, Tanner's classic paper was also forgotten for well over a decade, until its rediscovery by Wiberg (1996) in his seminal thesis.

18. For a detailed treatment of the statement that the probability distribution of the minimum distance of an LDPC code approaches a unit step function of the block length for certain values of weight-pair (t_c, t_r), see Gallager (1962, 1963).