CHAPTER 1

INTRODUCTION

1.1 WHY ERROR CONTROL?

The fundamental idea of *information theory* is that all communication is essentially digital—it is equivalent to generating, transmitting, and receiving randomly chosen *bi* nary digi*ts*, *bits*. When these bits are transmitted over a communication channel—or stored in a memory—it is likely that some of them will be corrupted by noise. In his 1948 landmark paper "A Mathematical Theory of Communication" [Sha48] Claude E. Shannon recognized that randomly chosen binary digits could (and should) be used for *measuring* the generation, transmission, and reception of *information*. Moreover, he showed that the problem of communicating information from a source over a channel to a destination can always be separated—without sacrificing optimality—into the following two subproblems: representing the source output efficiently as a sequence of binary digits (*source coding*) and transmitting binary, random, independent digits over the channel (*channel coding*). In Fig. 1.1 we show a general digital communication system. We use Shannon's *separation principle* and split the encoder and decoder into two parts each as shown in Fig. 1.2. The channel coding parts can be designed independently of the source coding parts, which simplifies the use of the same communication channel for different sources.

Fundamentals of Convolutional Coding, 2. Edition. By Rolf Johannesson, Kamil Sh. Zigangirov

To a computer specialist, "bit" and "binary digit" are entirely synonymous. In information theory, however, "bit" is Shannon's unit of information [Sha48, Mas82]. For Shannon, *information* is what we receive when uncertainty is reduced. We get exactly 1 bit of information from a binary digit when it is drawn in an experiment in which successive outcomes are independent of each other and both possible values, 0 and 1, are equiprobable; otherwise, the information is less than 1. In the sequel, the intended meaning of "bit" should be clear from the context.

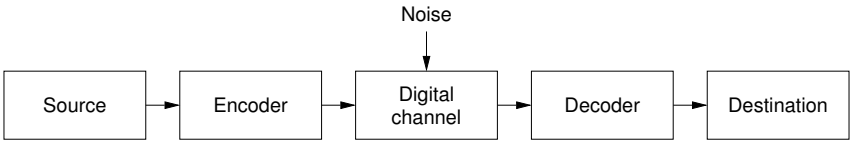

Figure 1.1 Overview of a digital communication system.

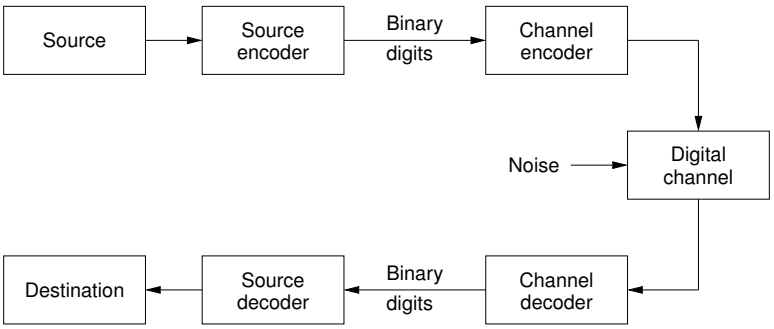

Figure 1.2 A digital communication system with separate source and channel coding.

Shannon's celebrated channel coding theorem states that every communication channel is characterized by a single parameter C_t, the *channel capacity*, such that R_t randomly chosen bits per second can be transmitted arbitrarily reliably over the channel if and only if $R_t \leq C_t$. We call R_t the *data transmission rate*. Both C_t and R_t are measured in bits per second. Shannon showed that the specific value of the *signal-to-noise ratio* is not significant as long as it is large enough, that is, so large that $R_t \leq C_t$ holds; what matters is how the information bits are encoded. The information should not be transmitted one information bit at a time, but long information sequences should be *encoded* such that each information bit has some influence on many of the bits transmitted over the channel. This radically new idea gave birth to the subject of *coding theory*.

Error control coding should protect digital data against errors that occur during transmission over a noisy communication channel or during storage in an unreliable memory. The last decades have been characterized not only by an exceptional increase in data transmission and storage but also by a rapid development in micro-electronics,

providing us with both a need for and the possibility of implementing sophisticated algorithms for error control.

Before we study the advantages of coding, we shall consider the digital communication channel in more detail. At a fundamental level, a channel is often an analog channel that transfers waveforms (Fig. 1.3). Digital data $u_0u_1u_2\ldots$, where $u_i \in \{0, 1\}$, must be *modulated* into waveforms to be sent over the channel.

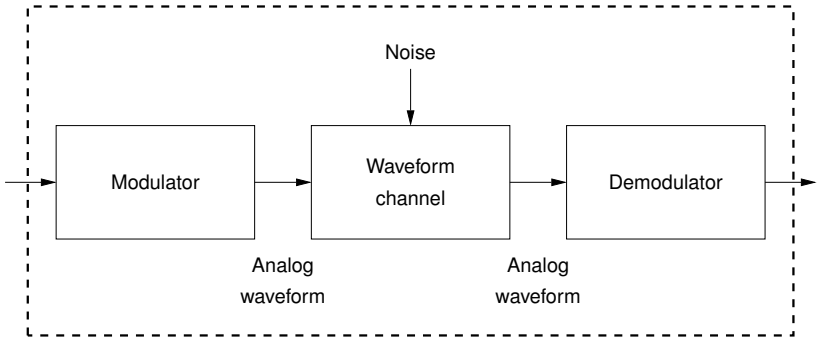

Figure 1.3 A decomposition of a digital communication channel.

In communication systems where carrier phase tracking is possible (coherent demodulation), *phase-shift keying* (PSK) is often used. Although many other modulation systems are in use, PSK systems are very common and we will use one of them to illustrate how modulations generally behave. In binary PSK (BPSK), the modulator generates the waveform

$$s_1(t) = \begin{cases} \sqrt{\frac{2E_s}{T}} \cos \omega t, & 0 \le t < T \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

for the input 1 and $s_0(t) = -s_1(t)$ for the input 0. This is an example of *antipodal signaling*. Each symbol has duration T seconds and energy $E_s = ST$, where S is the power and $\omega = \frac{2\pi}{T}$. The transmitted waveform is

$$v(t) = \sum_{i=0}^{\infty} s_{u_i}(t - iT) \tag{1.2}$$

Assume that we have a waveform channel such that additive white Gaussian noise (AWGN) $n(t)$ with zero mean and two-sided power spectral density $N_0/2$ is added to the transmitted waveform $v(t)$, that is, the received waveform $r(t)$ is given by

$$r(t) = v(t) + n(t) \tag{1.3}$$

where

$$E[n(t)] = 0 \tag{1.4}$$

and

$$E[n(t+\tau)n(t)] = \frac{N_0}{2}\delta(\tau) \tag{1.5}$$

where $E[\cdot]$ and $\delta(\cdot)$ denote the mathematical expectation and the delta function, respectively.

Based on the received waveform during a signaling interval, the demodulator produces an estimate of the transmitted symbol. The optimum receiver is a *matched filter* with impulse response

$$h(t) = \begin{cases} \sqrt{2/T}\cos\omega t, & 0 \le t < T \\ 0, & \text{else} \end{cases} \tag{1.6}$$

which is sampled each T seconds (Fig. 1.4). The matched filter output Z_i at the sample time iT,

$$Z_i = \int_{(i-1)T}^{iT} r(\tau)h(iT-\tau)d\tau \tag{1.7}$$

is a Gaussian random variable $N(\mu, \sigma^2)$ with mean

$$\mu = \pm\int_0^T \left(\sqrt{\frac{2E_\mathrm{s}}{T}}\cos\omega\tau\right)\left(\sqrt{\frac{2}{T}}\cos\omega(T-\tau)\right)d\tau = \pm\sqrt{E_\mathrm{s}} \tag{1.8}$$

where the sign is $+$ or $-$ according to whether the modulator input was 1 or 0, respectively, and variance

$$\sigma^2 = \frac{N_0}{2}\int_0^T \left(\sqrt{\frac{2}{T}}\cos\omega\tau\right)^2 d\tau = \frac{N_0}{2} \tag{1.9}$$

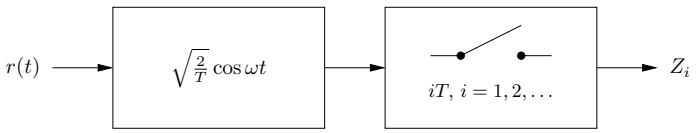

Figure 1.4 Matched filter receiver.

After the sampler we can make a *hard decision*, that is, a binary quantization with threshold zero, of the random variable Z_i. Then we obtain the simplest and most important binary-input and binary-output channel model, the *binary symmetric channel* (BSC) with *crossover probability* ϵ (Fig. 1.5). The crossover probability is of course closely related to the signal-to-noise ratio E_s/N_0. Since the channel output for a given signaling interval depends only on the transmitted waveform and noise during that interval and not on other intervals, the channel is said to be *memoryless*.

Because of symmetry, we can without loss of generality assume that a 0, that is, $-\sqrt{2E_\mathrm{s}/T}\cos\omega t$, is transmitted over the channel. Then we have a channel "error"

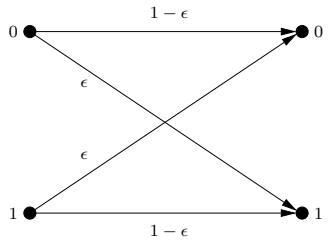

Figure 1.5 Binary symmetric channel.

if and only if the matched filter output at the sample time iT is positive. Thus, the probability that $Z_i > 0$ given that a 0 is transmitted is

$$\epsilon = P\left(Z_i > 0 \mid 0 \text{ sent}\right) \tag{1.10}$$

where Z_i is a Gaussian random variable, $Z_i \in N(-\sqrt{E_\mathrm{s}}, \sqrt{N_0/2})$, and E_s is the *energy per symbol*. Since the probability density function of Z_i is

$$f_{Z_i}(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{1.11}$$

we have

$$\begin{aligned} \epsilon &= \frac{1}{\sqrt{\pi N_0}} \int_0^\infty e^{-\frac{(z+\sqrt{E_\mathrm{s}})^2}{N_0}} dz \\ &= \frac{1}{\sqrt{2\pi}} \int_{\sqrt{2E_\mathrm{s}/N_0}}^\infty e^{-y^2/2} dy = Q\left(\sqrt{2E_\mathrm{s}/N_0}\right) \end{aligned} \tag{1.12}$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy \tag{1.13}$$

is the *complementary error function* of Gaussian statistics (often called the Q-function).

When coding is used, we prefer measuring the energy per *information* bit, E_b, rather than per symbol. For uncoded BPSK, we have $E_\mathrm{b} = E_\mathrm{s}$. Letting P_b denote the *bit error probability* (or *bit error rate*), that is, the probability that an information bit is erroneously delivered to the destination, we have for uncoded BPSK

$$P_\mathrm{b} = \epsilon = Q\left(\sqrt{2E_\mathrm{b}/N_0}\right) \tag{1.14}$$

How much better can we do with coding?

It is clear that when we use coding, it is a waste of information to make hard decisions. Since the influence of each information bit will be spread over several channel symbols, the decoder can benefit from using the *value* of Z_i (hard decisions use only the *sign* of Z_i) as an indication of how reliable the received symbol is. The demodulator can give the analog value of Z_i as its output, but it is often more

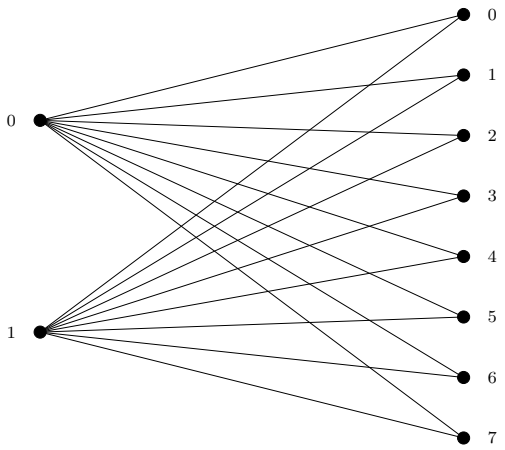

Figure 1.6 Binary input, 8-ary output, DMC.

practical to use, for example, a three-bit quantization—a *soft decision*. By introducing seven thresholds, the values of Z_i are divided into eight intervals and we obtain an eight-level soft-quantized *discrete memoryless channel* (DMC) as shown in Fig. 1.6.

Shannon [Sha48] showed that the capacity of the bandlimited AWGN channel with bandwidth W is[2]

$$C_t^W = W \log\left(1 + \frac{S}{N_0 W}\right) \text{ bits/s} \tag{1.15}$$

where $N_0/2$ and S denote the two-sided noise spectral density and the signaling power, respectively. If the bandwidth W goes to infinity, we have

$$\begin{aligned} C_t^\infty &\stackrel{\text{def}}{=} \lim_{W\to\infty} W \log\left(1 + \frac{S}{N_0 W}\right) \\ &= \frac{S}{N_0 \ln 2} \text{ bits/s} \end{aligned} \tag{1.16}$$

If we transmit K information bits during τ seconds, where τ is a multiple of bit duration T, we have

$$E_\text{b} = \frac{S\tau}{K} \tag{1.17}$$

Since the data transmission rate is $R_t = K/\tau$ bits/s, the energy per bit can be written

$$E_\text{b} = \frac{S}{R_t} \tag{1.18}$$

Combining (1.16) and (1.18) gives

$$\frac{C_t^\infty}{R_t} = \frac{E_\text{b}}{N_0 \ln 2} \tag{1.19}$$

[2]Here and hereafter we write log for $\log_2$.

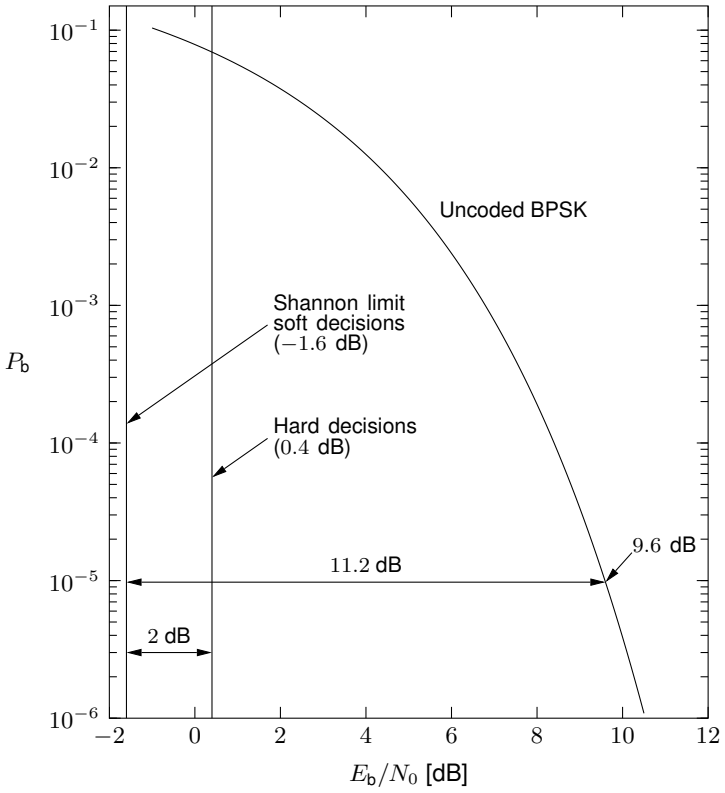

Figure 1.7 Capacity limits and regions of potential coding gain.

From Shannon's celebrated *channel coding theorem* [Sha48] it follows that for reliable communication we must have $R_t \leq C_t^{\infty}$. Hence, from this inequality and (1.19) we have

$$\frac{E_b}{N_0} \geq \ln 2 = 0.69 = -1.6 \text{ dB} \tag{1.20}$$

which is the fundamental *Shannon limit*.

In any system that provides reliable communication in the presence of additive white Gaussian noise the signal-to-noise ratio E_b/N_0 cannot be less than the Shannon limit, -1.6 dB!

On the other hand, as long as E_b/N_0 exceeds the Shannon limit, -1.6 dB, Shannon's channel coding theorem guarantees the existence of a system—perhaps very complex—for reliable communication over the channel.

In Fig. 1.7, we have plotted the fundamental limit of (1.20) together with the bit error rate for uncoded BPSK, that is, equation (1.14). At a bit error rate of 10^{-5}, the infinite-bandwidth additive white Gaussian noise channel requires an E_b/N_0 of at least 9.6 dB. Thus, at this bit error rate we have a potential *coding gain* of 11.2 dB!

For the bandlimited AWGN channel with BPSK and hard decisions, that is, a BSC with crossover probability ϵ (Fig. 1.5) Shannon [Sha48] showed that the capacity is

$$C_t^{\text{BSC}} = 2W(1 - h(\epsilon)) \text{ bits/s} \tag{1.21}$$

where

$$h(\epsilon) = -\epsilon \log \epsilon - (1 - \epsilon) \log(1 - \epsilon) \tag{1.22}$$

is the *binary entropy function*. If we restrict ourself to hard decisions, we can use (1.21) and show (Problem 1.2) that for reliable communication we must have

$$\frac{E_{\text{b}}}{N_0} \geq \frac{\pi}{2} \ln 2 = 1.09 = 0.4 \text{ dB} \tag{1.23}$$

In terms of capacity, soft decisions are about 2 dB more efficient than hard decisions.

Although it is practically impossible to obtain the entire theoretically promised 11.2 dB coding gain, communication systems that pick up 2–8 dB are routinely in use. During the last decade iterative decoding has been used to design communication systems that operate only tenths of a dB from the Shannon limit.

We conclude this section, which should have provided some motivation for the use of coding, with an adage from R. E. Blahut [Bla92]: "To build a communication channel as good as we can is a waste of money"—use coding instead!

1.2 BLOCK CODES—A PRIMER

For simplicity, we will deal only with *binary* block codes. We consider the entire sequence of information bits to be divided into *blocks* of K bits each. These blocks are called *messages* and denoted $\boldsymbol{u} = u_0\, u_1\, \ldots\, u_{K-1}$. In block coding, we let $\boldsymbol{u}$ denote a message rather than the entire information sequence as is the case in convolutional coding to be considered later.

A binary (N, K) *block code* $\mathcal{B}$ is a set of $M = 2^K$ binary N-tuples (or row vectors of length N) $\boldsymbol{v} = v_0\, v_1\, \ldots\, v_{N-1}$ called *codewords*. N is called the *block length* and the ratio

$$R = \frac{\log M}{N} = \frac{K}{N} \tag{1.24}$$

is called the *code rate* and is measured in bits per (channel) use. The data transmission rate in bits/s is obtained by multiplying the code rate (1.24) by the number of transmitted channel symbols per second:

$$R_t = R/T \tag{1.25}$$

If we measure the channel capacity for the BSC in bits/channel use (bits/c.u.), then the capacity of the BSC equals

$$C = 1 - h(\epsilon) \quad \text{(bits/c.u.)} \tag{1.26}$$

According to Shannon's channel coding theorem, for reliable communication, we must have $R \leq C$ and the block length N should be chosen sufficiently large.

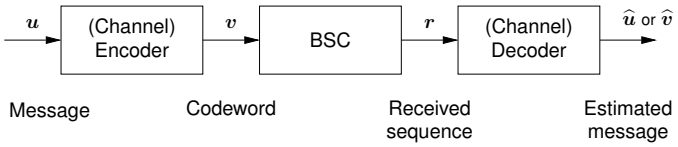

Figure 1.8 A binary symmetric channel (BSC) with (channel) encoder and decoder.

■ **EXAMPLE 1.1**

The set $\mathcal{B} = \{000, 011, 101, 110\}$ is a $(3, 2)$ code with four codewords and rate $R = 2/3$.

An *encoder* for an (N, K) block code $\mathcal{B}$ is a one-to-one mapping from the set of $M = 2^K$ binary messages to the set of codewords $\mathcal{B}$.

■ **EXAMPLE 1.2**

$u_0\, u_1$	$v_0\, v_1\, v_2$
00	000
01	011
10	101
11	110

and

$u_0\, u_1$	$v_0\, v_1\, v_2$
00	101
01	011
10	110
11	000

are two different encoders for the code $\mathcal{B}$ given in the previous example.

The rate $R = K/N$ is the fraction of the digits in the codeword that are necessary to represent the information; the remaining fraction, $1 - R = (N - K)/N$, represents the *redundancy* that can be used to detect or correct errors.

Suppose that a codeword $\boldsymbol{v}$ corresponding to message $\boldsymbol{u}$ is sent over a BSC (see Fig. 1.8). The channel output $\boldsymbol{r} = r_0\, r_1\, \ldots\, r_{N-1}$ is called the *received sequence*. The decoder transforms the received N-tuple $\boldsymbol{r}$, which is possibly corrupted by noise, into the K-tuple $\widehat{\boldsymbol{u}}$, called the estimated *message* $\boldsymbol{u}$. Ideally, $\widehat{\boldsymbol{u}}$ will be a replica of the message $\boldsymbol{u}$, but the noise may cause some *decoding errors*. Since there is a one-to-one correspondence between the message $\boldsymbol{u}$ and the codeword $\boldsymbol{v}$, we can, equivalently, consider the corresponding N-tuple $\widehat{\boldsymbol{v}}$ as the decoder output. If the codeword $\boldsymbol{v}$ was transmitted, a decoding error occurs if and only if $\widehat{\boldsymbol{v}} \neq \boldsymbol{v}$.

Let P_{E} denote the *block* (or *word*) *error probability*, that is, the probability that the decision $\widehat{\boldsymbol{v}}$ for the codeword differs from the transmitted codeword $\boldsymbol{v}$. Then we have

$$P_{\mathrm{E}} = \sum_{\boldsymbol{r}} P\left(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r}\right) P(\boldsymbol{r}) \tag{1.27}$$

where the probability that we receive $\boldsymbol{r}$, $P(\boldsymbol{r})$, is independent of the decoding rule and $P\left(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r}\right)$ is the conditional probability of decoding error given the received sequence $\boldsymbol{r}$. Hence, in order to minimize P_{E}, we should specify the decoder such

that $P(\widehat{\boldsymbol{v}} \neq \boldsymbol{v} \mid \boldsymbol{r})$ is minimized for a given $\boldsymbol{r}$ or, equivalently, such that $P(\boldsymbol{v} \mid \boldsymbol{r}) \stackrel{\text{def}}{=} P(\widehat{\boldsymbol{v}} = \boldsymbol{v} \mid \boldsymbol{r})$ is maximized for a given $\boldsymbol{r}$. Thus the block error probability P_{E} is minimized by the decoder, which as its output chooses $\widehat{\boldsymbol{u}}$ such that the corresponding $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ maximizes $P(\boldsymbol{v} \mid \boldsymbol{r})$. That is, $\boldsymbol{v}$ is chosen as the most likely codeword given that $\boldsymbol{r}$ is received. This decoder is called a *maximum a posteriori probability* (MAP) *decoder*.

Using Bayes' rule we can write

$$P(\boldsymbol{v} \mid \boldsymbol{r}) = \frac{P(\boldsymbol{r} \mid \boldsymbol{v})\, P(\boldsymbol{v})}{P(\boldsymbol{r})} \tag{1.28}$$

The code carries the most information possible with a given number of codewords when the codewords are equally likely. It is reasonable to assume that a decoder that is designed for this case also works satisfactorily—although not optimally—when the codewords are not equally likely, that is, when less information is transmitted. When the codewords are equally likely, maximizing $P(\boldsymbol{v} \mid \boldsymbol{r})$ is equivalent to maximizing $P(\boldsymbol{r} \mid \boldsymbol{v})$. The decoder that makes its decision $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ such that $P(\boldsymbol{r} \mid \boldsymbol{v})$ is maximized is called a *maximum-likelihood* (ML) *decoder*.

Notice that in an erroneous decision for the codeword some of the information digits may nevertheless be correct. The bit error probability, which we introduced in the previous section, is a better measure of quality in most applications. However, it is in general more difficult to calculate. The bit error probability depends not only on the code and on the channel, like the block error probability, but also on the encoder and on the information symbols!

The use of block error probability as a measure of quality is justified by the inequality

$$P_{\text{b}} \leq P_{\text{E}} \tag{1.29}$$

When P_E can be made very small, inequality (1.29) implies that P_{b} can also be made very small.

The *Hamming distance* between the two N-tuples $\boldsymbol{r}$ and $\boldsymbol{v}$, denoted $d_{\text{H}}(\boldsymbol{r}, \boldsymbol{v})$, is the number of positions in which their components differ.

■ **EXAMPLE 1.3**

Consider the 5-tuples 10011 and 11000. The Hamming distance between them is 3.

The Hamming distance, which is an important concept in coding theory, is a *metric*; that is,

(i) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) \geq 0$, with equality if and only if $\boldsymbol{x} = \boldsymbol{y}$ (positive definiteness)

(ii) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) = d_{\text{H}}(\boldsymbol{y}, \boldsymbol{x})$ (symmetry)

(iii) $d_{\text{H}}(\boldsymbol{x}, \boldsymbol{y}) \leq d_{\text{H}}(\boldsymbol{x}, \boldsymbol{z}) + d_{\text{H}}(\boldsymbol{z}, \boldsymbol{y})$, all $\boldsymbol{z}$ (triangle inequality)

The *Hamming weight* of an N-tuple $\boldsymbol{x} = x_0 x_1 \ldots x_{N-1}$, denoted $w_{\text{H}}(\boldsymbol{x})$, is defined as the number of nonzero components in $\boldsymbol{x}$.

For the BSC, a transmitted symbol is erroneously received with probability ϵ where ϵ is the channel crossover probability. Thus, assuming ML decoding, we must make our decision $\widehat{\boldsymbol{v}}$ for the codeword $\boldsymbol{v}$ to maximize $P(\boldsymbol{r} \mid \boldsymbol{v})$; that is,

$$\widehat{\boldsymbol{v}} = \arg\max_{\boldsymbol{v}} \{P(\boldsymbol{r} \mid \boldsymbol{v})\} \tag{1.30}$$

where

$$P(\boldsymbol{r} \mid \boldsymbol{v}) = \epsilon^{d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})}(1-\epsilon)^{N-d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})} = (1-\epsilon)^N \left(\frac{\epsilon}{1-\epsilon}\right)^{d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v})} \tag{1.31}$$

Since $0 < \epsilon < 1/2$ for the BSC, we have

$$0 < \frac{\epsilon}{1-\epsilon} < 1 \tag{1.32}$$

and, hence, maximizing $P(\boldsymbol{r} \mid \boldsymbol{v})$ is equivalent to minimizing $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})$. Clearly, ML decoding is equivalent to *minimum (Hamming) distance* (MD) *decoding*, that is, choosing as the decoder output the message $\widehat{\boldsymbol{u}}$ whose corresponding codeword $\widehat{\boldsymbol{v}}$ is (one of) the closest codeword(s) to the received sequence $\boldsymbol{r}$.

In general, the decoder must compare the received sequence $\boldsymbol{r}$ with all $M = 2^K = 2^{RN}$ codewords. The *complexity* of ML or MD decoding grows exponentially with the block length N. Thus it is infeasible to decode block codes with large block lengths. But to obtain low decoding error probability we have to use codes with relatively large block lengths. One solution of this problem is to use codes with algebraic properties that can be exploited by the decoder. Other solutions are to use codes on graphs (see Section 1.3) or convolutional codes (see Section 1.4).

In order to develop the theory further, we must introduce an algebraic structure.

A *field* is an algebraic system in which we can perform addition, subtraction, multiplication, and division (by nonzero numbers) according to the same associative, commutative, and distributive laws as we use with real numbers. Furthermore, a field is called *finite* if the set of numbers is finite. Here we will limit the discussion to block codes whose codewords have components in the simplest, but from a practical point of view also the most important, finite field, the *binary field*, $\mathbb{F}_2$, for which the rules for addition and multiplication are those of *modulo-two* arithmetic, namely

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

We notice that addition and subtraction coincide in $\mathbb{F}_2$!

The set of binary N-tuples are the *vectors* in an N-dimensional *vector space*, denoted $\mathbb{F}_2^N$, over the field $\mathbb{F}_2$. *Vector addition* is component-by-component addition in $\mathbb{F}_2$. The *scalars* are the elements in $\mathbb{F}_2$. *Scalar multiplication* of scalar $a \in \mathbb{F}_2$ and vector $x_0x_1 \ldots x_{N-1} \in \mathbb{F}_2^N$ is carried out according to the rule

$$a(x_0x_1 \ldots x_{N-1}) = ax_0ax_1 \ldots ax_{N-1} \tag{1.33}$$

Since a is either 0 or 1, scalar multiplication is trivial in $\mathbb{F}_2^N$.

Hamming weight and distance are clearly related:

$$d_{\mathrm{H}}(\boldsymbol{x},\boldsymbol{y}) = w_{\mathrm{H}}(\boldsymbol{x}-\boldsymbol{y}) = w_{\mathrm{H}}(\boldsymbol{x}+\boldsymbol{y}) \tag{1.34}$$

where the arithmetic is in the vector space $\mathbb{F}_2^N$ and where the last equality follows from the fact that subtraction and addition coincide in $\mathbb{F}_2$.

The *minimum distance*, $d_{\min}$, of a code $\mathcal{B}$ is defined as the minimum value of $d_{\mathrm{H}}(\boldsymbol{v},\boldsymbol{v}')$ over all $\boldsymbol{v}$ and $\boldsymbol{v}'$ in $\mathcal{B}$ such that $\boldsymbol{v} \neq \boldsymbol{v}'$.

■ **EXAMPLE 1.4**

The code $\mathcal{B}$ in Example 1.1 has $d_{\min} = 2$. It is a *single-error-detecting code* (see Problem 1.3).

Let $\boldsymbol{v}$ be the actual codeword and $\boldsymbol{r}$ the possibly erroneously received version of it. The *error pattern* $\boldsymbol{e} = e_0\, e_1\, \ldots\, e_{N-1}$ is the N-tuple that satisfies

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} \tag{1.35}$$

The *number of errors* is

$$w_{\mathrm{H}}(\boldsymbol{e}) = d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v}) \tag{1.36}$$

Let $\mathcal{E}_t$ denote the set of all error patterns with t or fewer errors, that is,

$$\mathcal{E}_t = \{\boldsymbol{e} \mid w_{\mathrm{H}}(\boldsymbol{e}) \leq t\} \tag{1.37}$$

We will say that a code $\mathcal{B}$ *corrects* the error pattern $\boldsymbol{e}$ if for all $\boldsymbol{v}$ the decoder maps $\boldsymbol{r} = \boldsymbol{v}+\boldsymbol{e}$ into $\widehat{\boldsymbol{v}} = \boldsymbol{v}$. If $\mathcal{B}$ corrects all error patterns in $\mathcal{E}_t$ and there is at least one error pattern in $\mathcal{E}_{t+1}$ which the code cannot correct, then t is called the *error-correcting capability* of $\mathcal{B}$.

Theorem 1.1 The code $\mathcal{B}$ has error-correcting capability t if and only if $d_{\min} > 2t$.

Proof: Suppose that $d_{\min} > 2t$. Consider the decoder which chooses $\widehat{\boldsymbol{v}}$ as (one of) the codeword(s) closest to $\boldsymbol{r}$ in Hamming distance (MD decoding). If $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$ and $\boldsymbol{e} \in \mathcal{E}_t$, then $d_{\mathrm{H}}(\boldsymbol{r},\boldsymbol{v}) \leq t$. The decoder output $\widehat{\boldsymbol{v}}$ must also satisfy $d_{\mathrm{H}}(\boldsymbol{r},\widehat{\boldsymbol{v}}) \leq t$ since $\widehat{\boldsymbol{v}}$ must be at least as close to $\boldsymbol{r}$ as $\boldsymbol{v}$ is. Thus,

$$d_{\mathrm{H}}(\boldsymbol{v},\widehat{\boldsymbol{v}}) \leq d_{\mathrm{H}}(\boldsymbol{v},\boldsymbol{r}) + d_{\mathrm{H}}(\boldsymbol{r},\widehat{\boldsymbol{v}}) \leq 2t < d_{\min} \tag{1.38}$$

which implies that $\widehat{\boldsymbol{v}} = \boldsymbol{v}$ and thus the decoding is correct.

Conversely, suppose that $d_{\min} \leq 2t$. Let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be two codewords such that $d_{\mathrm{H}}(\boldsymbol{v},\boldsymbol{v}') = d_{\min}$, and let the components of $\boldsymbol{r}$ be specified as

$$r_i = \begin{cases} v_i = v_i', & \text{all } i \text{ such that } v_i = v_i' \\ v_i', & \text{the first } t \text{ positions with } v_i \neq v_i' \text{ (if } t \leq d_{\min}\text{) or} \\ & \text{all positions with } v_i \neq v_i' \text{ (otherwise)} \\ v_i, & \text{the remaining } d_{\min} - t \text{ positions (if any)} \end{cases} \tag{1.39}$$

Thus, $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{r}) = t$ and $d_{\mathrm{H}}(\boldsymbol{v}', \boldsymbol{r}) = d_{\min} - t \leq t$ (if $t \leq d_{\min}$) and $d_{\mathrm{H}}(\boldsymbol{v}, \boldsymbol{r}) = d_{\min}$ and $d_{\mathrm{H}}(\boldsymbol{v}', \boldsymbol{r}) = 0$ (otherwise). Next we observe that both error patterns $\boldsymbol{e}$ and $\boldsymbol{e}'$ satisfying

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} = \boldsymbol{v}' + \boldsymbol{e}' \tag{1.40}$$

are in $\mathcal{E}_t$, but the decoder cannot make the correct decision for both situations, and the proof is complete. ∎

To make codes easier to analyze and to simplify the implementation of their encoders and decoders, we impose a *linear structure* on the codes.

A binary, *linear block code* $\mathcal{B}$ of rate $R = K/N$ is a K-dimensional subspace of the vector space $\mathbb{F}_2^N$; that is, each codeword can be written as a linear combination of linearly independent vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$, where $\boldsymbol{g}_i \in \mathbb{F}_2^N$, called the *basis* for the linear code $\mathcal{B}$. Then we call the $K \times N$ matrix G having $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$ as rows a *generator matrix* for $\mathcal{B}$. Clearly, since the vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_K$ are linearly independent, the matrix G has full rank. The *row space* of G is $\mathcal{B}$ itself.

■ EXAMPLE 1.5

For the code in Example 1.1, which is linear, the codewords 011 and 101 form a basis. This basis determines the generator matrix

$$G = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \tag{1.41}$$

The generator matrix offers a *linear encoding rule* for the code $\mathcal{B}$:

$$\boldsymbol{v} = \boldsymbol{u}G \tag{1.42}$$

where

$$G = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1N} \\ g_{21} & g_{22} & \cdots & g_{2N} \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ g_{K1} & g_{K2} & \cdots & g_{KN} \end{pmatrix} \tag{1.43}$$

and the *information symbols* $\boldsymbol{u} = u_0\, u_1\, \ldots\, u_{K-1}$ are encoded into the codeword $\boldsymbol{v} = v_0\, v_1\, \ldots\, v_{N-1}$.

A generator matrix is often called an *encoding matrix* and is any matrix whose rows are a basis for $\mathcal{B}$. It is called *systematic* whenever the information digits appear unchanged in the first K components of the codeword; that is, G is systematic if and only if it can be written as

$$G = (I_K \; P) \tag{1.44}$$

where I_K is the $K \times K$ identity matrix.

■ **EXAMPLE 1.6**

The generator matrix

$$G = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \tag{1.45}$$

is a systematic encoding matrix for the code in Example 1.1.

Two codes $\mathcal{B}$ and $\mathcal{B}'$ are said to be *equivalent* if the order of the digits in the codewords $\boldsymbol{v} \in \mathcal{B}$ are simply a rearrangement of that in the codewords $\boldsymbol{v}' \in \mathcal{B}'$.

Theorem 1.2 Either a linear code $\mathcal{B}$ has a systematic encoding matrix or there exists an equivalent linear code $\mathcal{B}'$ which has a systematic encoding matrix.

Proof: See Problem 1.5. ■

Let G be an encoding matrix of the (N, K) code $\mathcal{B}$. Then G is a $K \times N$ matrix of rank K. By the theory of linear equations, the solutions of the system of linear homogeneous equations

$$G\boldsymbol{x}^{\mathrm{T}} = \mathbf{0} \tag{1.46}$$

where $\boldsymbol{x} = x_1\, x_2 \ldots x_N$, form an $(N-K)$-dimensional subspace of $\mathbb{F}_2^N$. Therefore, there exists an $(N-K) \times N$ matrix H of rank $N-K$ such that

$$GH^{\mathrm{T}} = \mathbf{0} \tag{1.47}$$

We are now ready to prove a fundamental result.

Theorem 1.3 An N-tuple $\boldsymbol{v}$ is a codeword in the linear code $\mathcal{B}$ with encoding matrix G if and only if

$$\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0} \tag{1.48}$$

where H is an $(N-K) \times N$ matrix of full rank which satisfies

$$GH^{\mathrm{T}} = \mathbf{0} \tag{1.49}$$

Proof: Assume that the N-tuple $\boldsymbol{v} \in \mathcal{B}$, then $\boldsymbol{v} = \boldsymbol{u}G$ for some $\boldsymbol{u} \in \mathbb{F}_2^K$. Thus,

$$\boldsymbol{v}H^{\mathrm{T}} = \boldsymbol{u}GH^{\mathrm{T}} = \mathbf{0} \tag{1.50}$$

Conversely, suppose that $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$. Since $GH^{\mathrm{T}} = \mathbf{0}$ and both H and G have full rank, the rows of G form a basis of the solution space of $\boldsymbol{x}H^{\mathrm{T}} = \mathbf{0}$. Therefore, $\boldsymbol{v} = \boldsymbol{u}G$ for some $\boldsymbol{u} \in \mathbb{F}_2^K$, that is, $\boldsymbol{v} \in \mathcal{B}$. ■

From (1.49) it follows that each row vector of H is orthogonal to every codeword; the rows of H are *parity checks* on the codewords and we call H a *parity-check*

matrix[3] of the linear code $\mathcal{B}$. Equation (1.48) simply says that certain coordinates in each codeword must sum to zero.

It is easily verified that an (N, K) binary linear code with systematic encoding matrix $G = (I_K \ P)$ has

$$H = \left(P^{\mathrm{T}} \ I_{N-K}\right) \tag{1.51}$$

as a parity-check matrix.

■ **EXAMPLE 1.7**

The code in Example 1.1 with an encoding matrix given in Example 1.6 has

$$H = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \tag{1.52}$$

as a parity-check matrix.

Next, we will consider a member of a much celebrated class of *single-error-correcting* codes due to Hamming [Ham50].

■ **EXAMPLE 1.8**

The binary $(7, 4)$ Hamming code with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.53}$$

has

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.54}$$

as a parity-check matrix.

Note that $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$ can be written as

$$\begin{array}{ccccccccc} v_1 & + & v_2 & + & v_3 & + & v_4 & = & 0 \\ v_0 & + & v_2 & + & v_3 & + & v_5 & = & 0 \\ v_0 & + & v_1 & + & v_3 & + & v_6 & = & 0 \end{array} \tag{1.55}$$

so that each row in H determines one of the three *parity-check symbols* v_4, v_5, and v_6. The remaining four code symbols, v_0, v_1, v_2, and v_3, are the information symbols.

[3] When we consider LDPC codes in Section 1.3 and Chapter 8, we omit the full-rank requirement in the definition of parity-check matrices.

We notice that if we start with the 7-tuple 1011000, take all cyclic shifts, and form all linear combinations, we obtain a $(7, 4)$ Hamming code with a parity-check matrix

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.56}$$

which is *equivalent* to the one given in Example 1.8, that is, the two codes differ only by the ordering of the components in their codewords (permute columns 1–4). The $(7, 4)$ Hamming codes are members of a large class of important linear code families called *cyclic codes*—every cyclic shift of a codeword is a codeword. The class of cyclic codes includes, besides the Hamming codes, such famous codes as the Bose-Chaudhuri-Hocquenhem (BCH) and Reed-Solomon (RS) codes.

The cyclic behavior of these codes makes it possible to exploit a much richer algebraic structure that has resulted in the development of very efficient decoding algorithms suitable for hardware implementations.

Since the $N - K$ rows of the parity-check matrix H are linearly independent, we can use H as an encoding matrix of an $(N, N - K)$ linear code $\mathcal{B}^{\perp}$, which we call the *dual* or *orthogonal code* of $\mathcal{B}$.

Let $\boldsymbol{v}^{\perp} \in \mathcal{B}^{\perp}$ and assume that $\boldsymbol{v}^{\perp} = \boldsymbol{u}^{\perp}H$, where $\boldsymbol{u}^{\perp} \in \mathbb{F}_2^{N-K}$. Then from (1.49) it follows that

$$\boldsymbol{v}^{\perp}G^{\mathrm{T}} = \boldsymbol{u}^{\perp}HG^{\mathrm{T}} = \boldsymbol{u}^{\perp}\left(GH^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{0} \tag{1.57}$$

Conversely, assume that $\boldsymbol{v}^{\perp}G^{\mathrm{T}} = \mathbf{0}$ for $\boldsymbol{v}^{\perp} \in \mathbb{F}_2^N$. Since $HG^{\mathrm{T}} = \mathbf{0}$ and both H and G have full rank, $\boldsymbol{v}^{\perp}$ is a linear combination of the rows of H, that is, $\boldsymbol{v}^{\perp} \in \mathcal{B}^{\perp}$. Hence, G is a $K \times N$ parity-check matrix of the dual code $\mathcal{B}^{\perp}$ and we have proved the following

Theorem 1.4 An $(N - K) \times N$ parity-check matrix for the linear code $\mathcal{B}$ is an $(N - K) \times N$ encoding matrix for the dual code $\mathcal{B}^{\perp}$, and conversely.

From (1.48) and (1.57) it follows that every codeword of $\mathcal{B}$ is orthogonal to that of $\mathcal{B}^{\perp}$ and conversely.

■ **EXAMPLE 1.9**

The code $\mathcal{B}$ in Example 1.1 has the dual code $\mathcal{B}^{\perp} = \{000, 111\}$.

If $\mathcal{B} = \mathcal{B}^{\perp}$, we call $\mathcal{B}$ *self-dual*.

■ **EXAMPLE 1.10**

The $(2, 1)$ *repetition code* $\{00, 11\}$ is self-dual.

The *minimum weight*, $w_{\min}$, of a linear code $\mathcal{B}$ is the smallest Hamming weight of its nonzero codewords.

Theorem 1.5 For a linear code,

$$d_{\min} = w_{\min} \tag{1.58}$$

Proof: The theorem follows from (1.34) and the facts that for a linear code the sum of two codewords is a codeword. ∎

For the class of linear codes, the study of distance properties reduces to the study of weight properties that concern only single codewords! A most convenient consequence of this is the following.

Theorem 1.6 If H is any parity-check matrix for a linear code $\mathcal{B}$, then $d_{\min} = w_{\min}$ equals the smallest number of columns of H that form a linearly dependent set.

Proof: Follows immediately from $\boldsymbol{v}H^{\mathrm{T}} = \mathbf{0}$ for $\boldsymbol{v} \in \mathcal{B}$. ∎

■ **EXAMPLE 1.11**

Consider the (7,4) Hamming code with parity-check matrix (1.54),

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.59}$$

All pairs of columns are linearly independent. Many sets of three columns are linearly dependent, for example, columns 1, 6, and 7. It follows from the previous theorem that $d_{\min} = 3$. All single errors (that is, all error patterns of weight 1) can be corrected. This is a single-error-correcting code. This code can also detect double errors but not simultaneously with correcting single errors (see Problem 1.4).

The following theorem establishes an upper bound for the minimum distance.

Theorem 1.7 (Singleton bound) If $\mathcal{B}$ is an (N, K) linear code with minimum distance $d_{\min}$, then the number of parity-check digits is lower-bounded by

$$N - K \geq d_{\min} - 1 \tag{1.60}$$

Proof: A codeword with only one nonzero information digit has weight at most $1 + N - K$. Then, from Theorem 1.5 follows

$$w_{\min} = d_{\min} \leq N - K + 1 \tag{1.61}$$

∎

An (N, K) linear code that meets the Singleton bound with equality is called a *maximum-distance-separable* (MDS) *code*.

The only *binary* MDS codes are the trivial ones: the (N, N) code $\mathcal{B} = \mathbb{F}_2^N$, the $(N, 1)$ repetition code $\mathcal{B} = \{00\ldots0, 11\ldots1\}$, and the $(N, N-1)$ code consisting of all even-weight N-tuples. (For a nontrivial code, $2 \leq K \leq N - 1$.)

The most celebrated examples of *nonbinary* MDS codes are the Reed-Solomon codes.

To find (nontrivial) lower bounds on the minimum distance is a much harder problem. It has been proved [Gil52, Var57] that there exists a sequence of binary (N, K) linear block codes with increasing block length N having minimum distances lower-bounded by the inequality

$$d_{\min} > \rho_{\mathrm{GV}} N \tag{1.62}$$

where ρ_{GV} is the so-called *Gilbert-Varshamov parameter*. For a given rate $R = K/N$, ρ_{GV} is equal to the smallest root $\rho < 1/2$ of the equation

$$R = 1 - h(\rho) \tag{1.63}$$

where $h(\cdot)$ is the binary entropy function (1.22).

Let $\mathcal{B}$ be an (N, K) linear code. For any binary N-tuple $\boldsymbol{a}$, the set

$$\boldsymbol{a} + \mathcal{B} \stackrel{\mathrm{def}}{=} \{\boldsymbol{a} + \boldsymbol{v} \mid \boldsymbol{v} \in \mathcal{B}\} \tag{1.64}$$

is called a *coset* of $\mathcal{B}$. Every $\boldsymbol{b} \in \mathbb{F}_2^N$ is in some coset; for example, $\boldsymbol{b} + \mathcal{B}$ contains $\boldsymbol{b}$. Two binary N-tuples $\boldsymbol{a}$ and $\boldsymbol{b}$ are in the same coset if and only if their difference (sum in $\mathbb{F}_2^N$) is a codeword, or, equivalently, $(\boldsymbol{a} + \boldsymbol{b}) \in \mathcal{B}$. Every coset of $\mathcal{B}$ contains the same number of elements, 2^K, as $\mathcal{B}$ does.

Theorem 1.8 Any two cosets are either disjoint or identical.

Proof: Suppose that $\boldsymbol{c}$ belongs to both $\boldsymbol{a} + \mathcal{B}$ and $\boldsymbol{b} + \mathcal{B}$. Then $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{v} = \boldsymbol{b} + \boldsymbol{v}'$, where $\boldsymbol{v}, \boldsymbol{v}' \in \mathcal{B}$. Thus, $\boldsymbol{a} = \boldsymbol{b} + \boldsymbol{v} + \boldsymbol{v}' \in \boldsymbol{b} + \mathcal{B}$, and so $\boldsymbol{a} + \mathcal{B} \subseteq \boldsymbol{b} + \mathcal{B}$. Similarly $\boldsymbol{b} + \mathcal{B} \subseteq \boldsymbol{a} + \mathcal{B}$. Hence, $\boldsymbol{a} + \mathcal{B} = \boldsymbol{b} + \mathcal{B}$. ∎

From Theorem 1.8 two corollaries follow immediately:

Corollary 1.9 $\mathbb{F}_2^N$ is the union of all the cosets of $\mathcal{B}$.

Corollary 1.10 A binary (N, K) code $\mathcal{B}$ has 2^{N-K} cosets.

Suppose that the binary N-tuple $\boldsymbol{r}$ is received over the BSC. Then

$$\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e} \tag{1.65}$$

where $\boldsymbol{v} \in \mathcal{B}$ is a codeword and $\boldsymbol{e}$ is an error pattern. Clearly $\boldsymbol{r}$ is in the coset $\boldsymbol{r} + \mathcal{B}$. From (1.65) it follows that *the coset $\boldsymbol{r} + \mathcal{B}$ contains exactly the possible error patterns*! The N-tuple of smallest weight in a coset is called a *coset leader*. (If there is more than one N-tuple with smallest weight, any one of them can be chosen as coset leader.)

An MD decoder will select as its output the error pattern, $\widehat{\boldsymbol{e}}$ say, which is a coset leader of the coset containing $\boldsymbol{r}$, subtract (or, equivalently in $\mathbb{F}_2^N$, add) $\widehat{\boldsymbol{e}}$ from (to) $\boldsymbol{r}$, and, finally, obtain its maximum-likelihood decision $\widehat{\boldsymbol{v}}$.

We illustrate what the decoder does by showing the *standard array*. The first row consists of the code $\mathcal{B}$ with the allzero codeword on the left. The following rows are the cosets $\boldsymbol{e}_i + \mathcal{B}$ arranged in the same order with the coset leader on the left:

$\mathbf{0}$	$\boldsymbol{v}_1$	$\ldots$	$\boldsymbol{v}_{2^K-1}$
$\boldsymbol{e}_1$	$\boldsymbol{v}_1 + \boldsymbol{e}_1$	$\ldots$	$\boldsymbol{v}_{2^K-1} + \boldsymbol{e}_1$
$\vdots$	$\vdots$		$\vdots$
$\boldsymbol{e}_{2^{N-K}-1}$	$\boldsymbol{v}_1 + \boldsymbol{e}_{2^{N-K}-1}$	$\ldots$	$\boldsymbol{v}_{2^K-1} + \boldsymbol{e}_{2^{N-K}-1}$

The MD decoder decodes $\boldsymbol{r}$ to the codeword $\widehat{\boldsymbol{v}}$ at the top of the column that contains $\boldsymbol{r}$.

■ **EXAMPLE 1.12**

The $(4, 2)$ code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \tag{1.66}$$

has the following standard array:

0000	1011	0110	1101
1000	0011	1110	0101
0100	1111	0010	1001
0001	1010	0111	1100

Suppose that $\boldsymbol{r} = 1001$ is received. An MD decoder outputs $\widehat{\boldsymbol{v}} = 1101$.

Theorem 1.11 An (N, K) binary linear code $\mathcal{B}$ can correct all error patterns in a set $\mathcal{E}$ if and only if these error patterns all lie in different cosets of $\mathbb{F}_2^N$ relative to $\mathcal{B}$.

Proof: Suppose that $\boldsymbol{e}$ and $\boldsymbol{e}'$ are distinct error patterns in the same coset. Then there is a $\boldsymbol{v} \in \mathcal{B}$ such that $\boldsymbol{v} + \boldsymbol{e} = \boldsymbol{e}'$. No decoder can correct both $\boldsymbol{e}$ and $\boldsymbol{e}'$.

Conversely, suppose that all error patterns in $\mathcal{E}$ lie in different cosets. If $\boldsymbol{v}$ is the actual transmitted codeword and $\boldsymbol{e}$ the actual error pattern, then $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$ lies in $\boldsymbol{e} + \mathcal{B}$. Thus, all error patterns in $\mathcal{E}$ can be corrected by a decoder that maps $\boldsymbol{r}$ into the error pattern $\widehat{\boldsymbol{e}} \in \mathcal{E}$ (if any) that lies in the same coset $\boldsymbol{e} + \mathcal{B}$ as $\boldsymbol{r}$ does. ■

The *syndrome* of the received N-tuple $\boldsymbol{r}$, relative to the parity-check matrix H, is defined as

$$\boldsymbol{s} \stackrel{\text{def}}{=} \boldsymbol{r}H^{\mathrm{T}} \tag{1.67}$$

Assume that the transmitted codeword is $\boldsymbol{v}$ and that $\boldsymbol{r} = \boldsymbol{v} + \boldsymbol{e}$, where $\boldsymbol{e}$ is the error pattern. Both $\boldsymbol{r}$ and H are known to the receiver, which exploits (1.48) and forms

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{r}H^{\mathrm{T}} = (\boldsymbol{v} + \boldsymbol{e})H^{\mathrm{T}} \\ &= \boldsymbol{v}H^{\mathrm{T}} + \boldsymbol{e}H^{\mathrm{T}} = \mathbf{0} + \boldsymbol{e}H^{\mathrm{T}} \end{aligned} \tag{1.68}$$

so that

$$\boldsymbol{s} = \boldsymbol{e}H^{\mathrm{T}} \tag{1.69}$$

The syndrome depends only on the error pattern and not on the codeword!

In medical terminology, a syndrome is a pattern of symptoms. Here the disease is the error pattern, and a symptom is a parity-check failure.

Equation (1.69) gives $N-K$ linearly independent equations for the N components of the error pattern $\boldsymbol{e}$. Hence, there are exactly 2^K error patterns satisfying (1.69). These are precisely all the error patterns that are differences between the received N-tuple $\boldsymbol{r}$ and all 2^K different codewords $\boldsymbol{v}$. For a given syndrome, these 2^K error patterns belong to the same coset. Furthermore, if two error patterns lie in the same coset, then their difference is a codeword and it follows from (1.68) that they have the same syndrome. Hence, we have the following theorem:

Theorem 1.12 Two error patterns lie in the same coset if and only if they have the same syndrome.

From the two previous theorems a corollary follows:

Corollary 1.13 An (N, K) binary linear code $\mathcal{B}$ can correct all error patterns in a set $\mathcal{E}$ if and only if the syndromes of these error patterns are all different.

No information about the error pattern is lost by calculating the syndrome!

In Fig. 1.9 we show the structure of a general *syndrome decoder*. The syndrome former, H^{T}, is linear, but the error pattern estimator is always nonlinear in a useful decoder. Clearly, a syndrome decoder is an MD or, equivalently, an ML decoder.

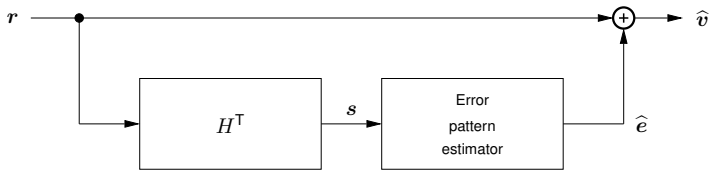

Figure 1.9 Syndrome decoder for a linear block code.

■ **EXAMPLE 1.13**

Consider the $(7, 4)$ Hamming code whose parity-check matrix is given in Example 1.11. Let $\boldsymbol{r} = 0010001$ be the received 7-tuple. The syndrome is

$$\boldsymbol{s} = (0010001)\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = 111 \tag{1.70}$$

Since $\boldsymbol{s} \neq \mathbf{0}$, $\boldsymbol{r}$ contains at least one erroneous component. For the Hamming codes there is a one-to-one correspondence between the single errors and the nonzero syndromes. Among all $2^K = 16$ possible error patterns, the MD decoder chooses the one with least Hamming weight, that is, the single error pattern corresponding to the given syndrome $\boldsymbol{s} = 111$. Since the fourth row in H^{T} is the triple 111, the MD decoder gives as its output $\widehat{\boldsymbol{e}} = 0001000$ (a single 1 in the *fourth* position). It immediately follows that

$$\begin{aligned}\widehat{\boldsymbol{v}} &= \boldsymbol{r} + \widehat{\boldsymbol{e}} \\ &= 0010001 + 0001000 \\ &= 0011001 \end{aligned} \tag{1.71}$$

If $\boldsymbol{v} = 0011001$ was sent, we have corrected the transmission error. However, if $\boldsymbol{v} = 0000000$ was sent and $\boldsymbol{e} = 0010001$, the Hamming code is not able to correct the error pattern. The syndrome decoder will in this case give as its output $\widehat{\boldsymbol{v}} = 0011001$!

Suppose that the $(7, 4)$ Hamming code is used to communicate over a BSC with channel error probability ϵ. The ML decoder can correctly identify the transmitted codeword if and only if the channel causes at most one error. The block (or word) error probability P_{E}, that is, the probability that the decision for the codeword differs from the actual codeword, is

$$\begin{aligned}P_{\mathrm{E}} &= \sum_{i=2}^{7} \binom{7}{i} \epsilon^i (1-\epsilon)^{7-i} \\ &= 21\epsilon^2 - 70\epsilon^3 + \cdots \end{aligned} \tag{1.72}$$

For the $(7, 4)$ Hamming code, it can be shown (Problem 1.21) that for all digits

$$\begin{aligned}P_{\mathrm{b}} &= 9\epsilon^2(1-\epsilon)^5 + 19\epsilon^3(1-\epsilon)^4 + 16\epsilon^4(1-\epsilon)^3 \\ &\quad +12\epsilon^5(1-\epsilon)^2 + 7\epsilon^6(1-\epsilon) + \epsilon^7 \\ &= 9\epsilon^2 - 26\epsilon^3 + \cdots \end{aligned} \tag{1.73}$$

Maximum-likelihood (ML) decoding that we discussed in this section is an important decoding method in the sense that (assuming equiprobable messages) it minimizes the block error probability. In Chapter 5 we shall show that if we use ML decoding there exist codes for which the block and bit error probabilities decrease exponentially with block length N.

A drawback with ML decoding is that its decoding complexity increases very fast—exponentially fast—with increasing block length. There is a need for low-complexity suboptimum decoding algorithms.

1.3 CODES ON GRAPHS

In this section, we introduce an important class of block codes, *codes on graphs*, which are very powerful in combination with *iterative decoding*. A well-known class

of such codes is the *low-density parity-check (LDPC)* codes invented by Gallager [Gal62, Gal63]. When these codes are decoded iteratively, they have better bit error probability/decoding complexity tradeoff than general block codes.

Tanner [Tan81] used *bipartite graphs* to describe the structure of linear block codes. These so-called *Tanner graphs* consist of two sets of nodes, the set of *symbol* (or *variable*) nodes which correspond to the symbols of the codewords and the set of *constraint* (or *factor*) nodes which correspond to the parity-check equations defining the code. In a Tanner graph, each symbol node v_n is connected by edges only with constraint nodes c_l and, similarly, each constraint node is connected only with symbol nodes. The number of edges connected to a node is called the *degree* of that node.

Each column in the parity-check matrix corresponds to a symbol node and each row corresponds to a constraint node. A 1 in the (l, n)th position of the parity-check matrix H corresponds to an edge between the symbol node v_n and the constraint node c_l.

■ **EXAMPLE 1.14**

Consider the following parity-check matrix (1.56) of a $(7, 4)$ Hamming code,

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{1.74}$$

This code has seven code symbols interrelated by three parity-check equations (*constraints*),

$$\begin{array}{ccccccccc} v_0 & + & v_1 & + & v_2 & + & v_4 & = & 0 \\ v_1 & + & v_2 & + & v_3 & + & v_5 & = & 0 \\ v_0 & + & v_1 & + & v_3 & + & v_6 & = & 0 \end{array} \tag{1.75}$$

The Tanner graph of the Hamming $(7, 4)$ code (Fig. 1.10) has seven symbol nodes and three constraint nodes. All constraint nodes have degree 4, but the degrees of symbol nodes varies from 1 to 3.

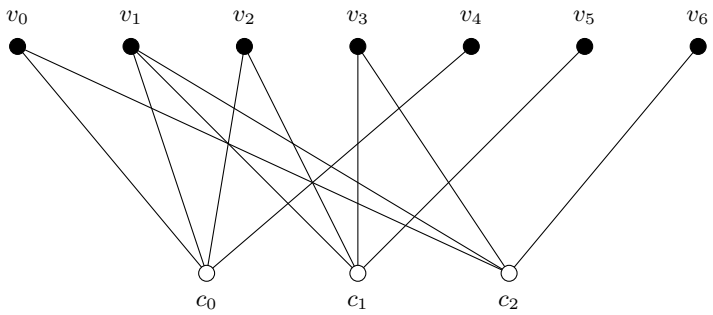

Figure 1.10 Tanner graph of a Hamming $(7, 4)$ code defined by parity-check matrix (1.74).

Note that the set of four symbols included in each of the linear equations (1.75) forms a $(4, 3)$ single-error-detecting code. Totally we have three such *constituent codes*.

An LDPC code can be defined either by a parity-check matrix or by a Tanner graph. In the first case it is determined by an $L \times N$ sparse parity-check matrix H, "containing mostly 0s and relatively few 1s" as it was formulated by Gallager [Gal62, Gal63]. A *regular* (N, J, K) LDPC code of block length N has a parity-check matrix with a fixed number K 1s in each row[4] and a fixed number J 1s in each column, where $K/J = N/L$.

The Tanner graph of a regular (N, J, K) LDPC code has symbol node degree equal to J and constraint node degree equal to K. The rows of the parity-check matrix H of a regular (N, J, K) LDPC code can be linearly dependent, then the *design rate* $R_d = 1 - J/K$ can be less than the actual code rate R.

Remark: In the sequel we will for simplicity often use short block codes as illustrations and call them LDPC codes even if the requrement "containing mostly 0s and relatively few 1s" is not fulfilled.

■ **EXAMPLE 1.15**

Consider the following parity-check matrix of the rate $R = 6/15 = 2/5$ regular $(N, J, K) = (15, 3, 5)$ LDPC block code:

$$H = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{1.76}$$

This code was found by computer search and has relatively good minimum distance, $d_{\min} = 6$, for an LDPC code of this size. Its Tanner graph is shown in Fig. 1.11.

An *irregular* LDPC code has in general a Tanner graph whose symbol node degrees as well as constraint node degrees are different.

For an LDPC code, the rows of the $L \times N$ parity-check matrix H can in general be linearly dependent and, equivalently, this matrix has not full rank. If we delete linearly dependent rows such that the remaining rows are linearly independent, then we obtain a parity-check matrix for a rate $R > R_d = 1 - L/N$ LDPC code.

[4]With a slight abuse of notations we use K to denote both the number of 1s in each row of and the number of information symbols in an (N, K) block code.

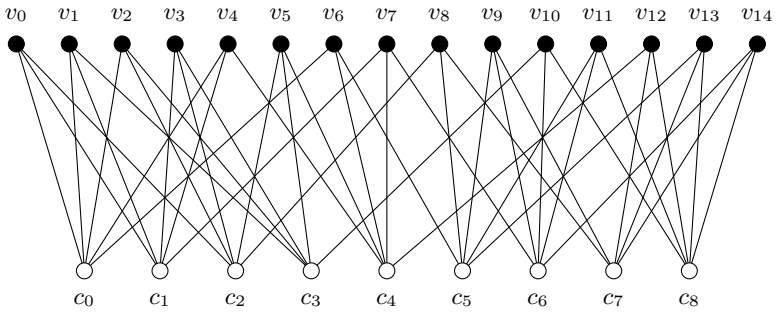

Figure 1.11 Tanner graph of the code defined by parity-check matrix (1.76).

Gallager introduced a special class of regular (N, J, K) LDPC codes. The $L \times N$ parity-check matrices H of these codes, where $N = KM$, $L = JM$, and $M > 0$ is an integer, can be represented as a composition of J submatrices H_j, $j = 1, 2, \ldots, J$,

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_J \end{pmatrix} \tag{1.77}$$

Here the $M \times N$ submatrices H_j have one 1 in each column and K 1s in each row. The other entries of the submatrices are 0s. If $J \geq 2$, then the rank of the matrix (1.77) is always strictly less than L (see Problem 1.24).

Using the ensemble of regular (N, J, K) LDPC codes with random parity-check matrices given by (1.77) Gallager[5] proved that, if $N \to \infty$, then the minimum distance $d_{\min}$ of almost all codes from this ensemble is lower-bounded by the inequality

$$d_{\min} > \rho_{J,K} N \tag{1.78}$$

where the parameter $\rho_{J,K}$ is a positive constant (if $J > 2$). In other words, the minimum distance of typical (N, J, K) codes from this ensemble, as well as of typical randomly chosen block codes, is growing linearly with N.

In Table 1.1 we show the numerical values of $\rho_{J,K}$ for some J and K. For comparison also the value of the Gilbert-Varshamov parameter ρ_{GV} is given for the corresponding design rates $R_d = 1 - J/K$. In Chapter 8 we will prove the interesting fact that if $J = 2$ then $d_{\min}$ grows only logarithmically with N.

The main advantage of LDPC codes is that these codes can be decoded *iteratively* and that the complexity of iterative decoding grows slower than exponentially with the block length N. We shall conclude this introductory section by describing a simple iterative decoding algorithm for LDPC codes.

[5]Gallager used the slightly different ensemble with the first matrix $H^{(1)}$ fixed. This has, however, no effect on the technique of his proof.

Table 1.1 Gallager's parameter $\rho_{J,K}$ compared with the Gilbert-Varshamov parameter ρ_{GV}.

J	K	R_c	$\rho_{J,K}$	ρ_{GV}	$\rho_{\text{GV}}/\rho_{J,K}$
3	4	0.25	0.112	0.215	1.920
	5	0.4	0.045	0.146	3.244
	6	0.5	0.023	0.11	4.783
4	5	0.2	0.211	0.243	1.152
	6	0.333	0.129	0.174	1.349
5	6	0.167	0.255	0.264	1.035

Gallager introduced a simple hard-decision iterative decoding algorithm for LDPC codes called the *bit-flipping (BF) algorithm* [Gal62, Gal63]. It was later modified by Zyablov and Pinsker [ZyP75]. We will describe a version of Gallager's algorithm for regular (N, J, K) LDPC codes which uses an adaptive threshold.

Suppose that the binary N-tuple (codeword) $\boldsymbol{v}$ is sent over a BSC and that the binary N-tuple $\boldsymbol{r}$ is received. On the receiver side, a bit-flipping decoder calculates the tentative syndrome

$$\boldsymbol{s}' = (s'_0\, s'_1\, \ldots\, s'_{L-1}) = \boldsymbol{r}' H^{\mathrm{T}} \tag{1.79}$$

for the tentative sequence $\boldsymbol{r}' = r'_0\, r'_1\, \ldots\, r'_{N-1}$ which initially is equal to the received sequence $\boldsymbol{r}$.

Since the code is regular, each tentative symbol r'_n, $n = 0, 1, \ldots, N-1$, is included in J parity-check equations. Let δ_n, $n = 0, 1, \ldots, N-1$, denote the number of unsatisfied parity-check equations for the tentative symbol r'_n. We introduce a threshold T which is initially set to J. The decoder searches for any r'_n such that $\delta_n = T$. If such an r'_n does not exist, we decrease the threshold by 1, that is, $T \leftarrow T - 1$, and repeat the "search/decrease threshold" procedure until we have one of the following three situations:

(i) We find a tentative symbol r'_n with $\delta_n = T > \lfloor J/2 \rfloor$.

(ii) All $\delta_n = 0$, $n = 0, 1, \ldots, N-1$, that is, $\boldsymbol{s}' = \mathbf{0}$.

(iii) $\boldsymbol{s}' \neq \mathbf{0}$ and for all symbols r'_n we have $\delta_n \leq \lfloor J/2 \rfloor$.

If we find a tentative symbol r'_n such that $\delta_n = T > \lfloor J/2 \rfloor$, then we *flip* (change) r'_n and obtain a new tentative sequence $\boldsymbol{r}'$ which yields a new tentative syndrome $\boldsymbol{s}'$ with reduced Hamming weight. Whenever such a reduction of this Hamming weight occurs, we reset the threshold to $T = J$. (This is necessary since the flipping of r'_n could result in a $\delta_{n'}$ for $n' \neq n$ being as large as J.) Then we repeat the search procedure.

If the tentative syndrome $\boldsymbol{s}' = \mathbf{0}$, then the corresponding tentative sequence $\boldsymbol{r}'$ is a codeword and $\widehat{\boldsymbol{v}} = \boldsymbol{r}'$ is the decision for $\boldsymbol{v}$. The decoding is considered successful.

If $\boldsymbol{s}' \neq \mathbf{0}$ and $\delta_n \leq \lfloor J/2 \rfloor$ for $n = 0, 1, \ldots, N-1$, then the decoding is declared as a failure. The corresponding tentative sequence $\boldsymbol{r}'$ is called a *trapping set*.

Gallager's bit-flipping algorithm is characterized by its error-correcting capability and its decoding complexity. Its error-correcting capability $t^{(\mathrm{BF})}$ is the largest integer such that all error patterns of Hamming weight $t^{(\mathrm{BF})}$ or less are correctly decoded by the algorithm. For Gallager's bit-flipping algorithm, we can show that asymptotically, that is, when $N \to \infty$, the error-correcting capability is lower-bounded by a linear function of N. For maximum-likelihood decoding we have, as shown in the previous section, the error-correcting capability $t^{(\mathrm{ML})} = \lfloor \frac{d_{\min}-1}{2} \rfloor$, where $d_{\min}$ is the minimum distance of the code.

Since the bit-flipping algorithm is essentially less powerful than an ML algorithm, we have in general $t^{(\mathrm{BF})} \leq t^{(\mathrm{ML})}$.

What can we say about the decoding complexity of the bit-flipping algorithm? We will show that the total number of operations for Gallager's bit-flipping algorithm is upper-bounded by[6] $O(N^2)$. During each iteration the decoder checks the symbols r'_n, $n = 0, 1, \ldots, N-1$, of the tentative sequence $\boldsymbol{r}'$ up to the moment when it finds a symbol which is included in δ unsatisfied parity-check equations. Then it flips this symbol which decreases the tentative syndrome weight by at least one. Since the initial syndrome weight does not exceed N, the total number of iterations is upper-bounded by $O(N)$. In each iteration, the decoder checks at most N symbols. Assuming that checking one symbol requires one computational operation, the total number of operations for one iteration is upper-bounded by $O(N)$. Then the decoding complexity of Gallager's bit-flipping algorithm is upper-bounded by $O(N^2)$.

We can also describe the bit-flipping algorithm using the Tanner graph. In this case the decoder checks how many constraint nodes connected to a given tentative symbol node correspond to unsatisfied parity-check equations. If this number equals the threshold T, then the decoder flips the corresponding tentative symbol and goes to the next phase of the decoding.

Remark: We gave one possible description of Gallager's bit-flipping algorithm, with an adaptive threshold. In principle, the decoder can flip a tentative symbol whenever the number of unsatisfied parity-check equations for this symbol $\delta > \lfloor J/2 \rfloor$. We can say in this case that the decoder uses the lowest possible threshold $\lfloor J/2 \rfloor + 1$. The error-correcting capability when $N \to \infty$ of the algorithm with lowest possible threshold is still $O(N)$, but for finite N it is less than that for the algorithm with an adaptive threshold. In Problem 1.25 we study an example when the algorithm with an adaptive threshold has error-correcting capability $t^{(\mathrm{BF})} = 1$ but the algorithm with lowest possible threshold has zero error-correcting capability.

■ **EXAMPLE 1.16**

Consider the regular $(15, 3, 5)$ LDPC code given in Example 1.15. Suppose that we use the bit-flipping algorithm with an adaptive threshold for decoding. We will

[6]Here and hereafter we write $f(x) = O(g(x))$ if $|f(x)| \leq Ag(x)$ for x sufficiently near a given limit, A is a positive constant independent of x and $g(x) > 0$. We have, e.g., $f(x) = \log x$, $x > 0$, can be written as $f(x) = O(x)$ when $x \to \infty$.

show that this algorithm corrects all single errors and that there are double errors which the algorithm does not correct.

Let the transmitted sequence be the allzero code sequence $\boldsymbol{v} = \mathbf{0} = 00\ldots0$ sent over the BSC and assume that all symbols of the received sequence $\boldsymbol{r}$ except r_7 are correctly received, that is, $\boldsymbol{r} = 000000010000000$. The decoder calculates the syndrome (cf. Fig. 1.11)

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 010010100 \tag{1.80}$$

It has Hamming weight 3. The initial value of the threshold is $T = J = 3$. Note that flipping, for example, the symbol r_4 decreases the syndrome weight by 1, but the only symbol included in $\delta = 3$ unsatisfied parity-check equations is the symbol r_7. Flipping this symbol decreases the syndrome weight by 3 and results in successful decoding.

It is easily shown that the decoder corrects all single errors. It does not, however, correct all double errors. Suppose that all symbols of the received sequence $\boldsymbol{r}$ except r_4 and r_7 are correctly received, that is, $\boldsymbol{r} = 000010010000000$. The syndrome

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 100000100 \tag{1.81}$$

has Hamming weight 2 and there are no symbols such that flipping causes a decrease of the syndrome weight. The decoding has failed and the sequence $\boldsymbol{r} = 000010010000000$ is a trapping set.

Next we consider another pattern of double errors, namely, all symbols of the received sequence $\boldsymbol{r}$ except r_0 and r_{14} are correctly received, that is, $\boldsymbol{r} = 100000000000001$. The syndrome

$$\boldsymbol{s} = \boldsymbol{r}H^{\mathrm{T}} = 111000111 \tag{1.82}$$

has Hamming weight 6. Flipping the symbol r_0 yields the tentative syndrome

$$\boldsymbol{s}' = \boldsymbol{r}'H^{\mathrm{T}} = 000000111 \tag{1.83}$$

with Hamming weight 3. Then, flipping r_{14} yields the tentative syndrome $\boldsymbol{s}' = \mathbf{0}$ and results in successful decoding of this particular double-error pattern.

Since the minimum distance $d_{\min} = 6$, we would correct all double errors if we were using a maximum-likelihood decoder.

We have described a variant of Gallager's original bit-flipping algorithm. The Zyablov-Pinsker iterative decoding algorithm finds in each step all bits of the received sequence which are included in more than $\lfloor J/2 \rfloor$ unsatisfied parity-check equations and then flips simultaneously all these bits. A theoretical analysis of the Zyablov-Pinsker algorithm [ZPZ08] gives the following asymptotical lower bound for the error-correcting capability when $N \to \infty$:

$$t^{(\mathrm{ZP})} > \rho_{J,K}^{(\mathrm{ZP})} N \tag{1.84}$$

where the coefficient $\rho_{J,K}^{(\mathrm{ZP})}$ is much smaller than both the corresponding Gilbert-Varshamov parameter ρ_{GV} and the Gallager parameter $\rho_{J,K}$. For example, for $J = 9$ and $K = 10$ we have $\rho_{J,K}^{(\mathrm{ZP})} = 1.29 \times 10^{-3}$.

It can be shown that the decoding complexity of the Zyablov-Pinsker algorithm is upper-bounded by $O(N \log N)$.

In Chapter 8 we will consider a more powerful iterative decoding algorithm for LDPC codes called the *belief propagation* (BP) algorithm. This algorithm was also invented by Gallager [Gal62, Gal63] and provides, at the cost of higher decoding complexity, better error correction than the bit-flipping algorithm.

1.4 A FIRST ENCOUNTER WITH CONVOLUTIONAL CODES

Convolutional codes are often thought of as nonblock linear codes over a finite field, but it can be an advantage to treat them as block codes over certain infinite fields. We will postpone the precise definitions until Chapter 2 and instead begin by studying a simple example of a binary convolutional encoder (Fig. 1.12).

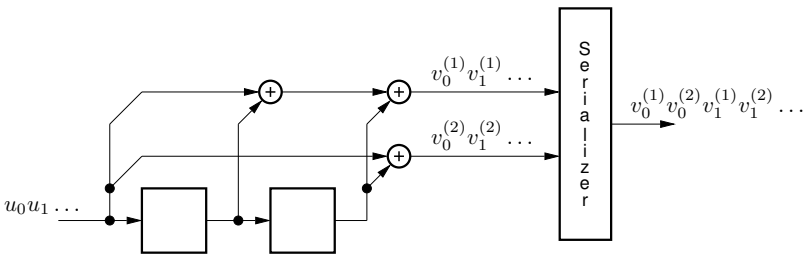

Figure 1.12 An encoder for a binary rate $R = 1/2$ convolutional code.

The information digits $\boldsymbol{u} = u_0 u_1 \ldots$ are not as in the previous section separated into blocks. Instead they form an infinite sequence that is shifted into a register, in our example, of length or *memory* $m = 2$. The encoder has two linear output functions. The two output sequences $\boldsymbol{v}^{(1)} = v_0^{(1)} v_1^{(1)} \ldots$ and $\boldsymbol{v}^{(2)} = v_0^{(2)} v_1^{(2)} \ldots$ are interleaved by a serializer to form a single-output sequence $v_0^{(1)} v_0^{(2)} v_1^{(1)} v_1^{(2)} \ldots$ that is transmitted over the channel. For each information digit that enters the encoder, two channel digits are emitted. Thus, the code rate of this encoder is $R = 1/2$ bits/channel use.

Assuming that the content of the register is zero at time $t = 0$, we notice that the two output sequences can be viewed as a convolution of the input sequence $\boldsymbol{u}$ and the two sequences $11100 \ldots$ and $10100 \ldots$, respectively. These latter sequences specify the linear output functions; that is, they specify the encoder. The fact that the output sequences can be described by convolutions is why such codes are called convolutional codes.

In a general rate $R = b/c$, where $b \leq c$, binary *convolutional encoder* (without feedback) the causal, that is, zero for time $t < 0$, information sequence

$$\boldsymbol{u} = \boldsymbol{u}_0\boldsymbol{u}_1 \ldots = u_0^{(1)}u_0^{(2)} \ldots u_0^{(b)}u_1^{(1)}u_1^{(2)} \ldots u_1^{(b)} \ldots \tag{1.85}$$

is encoded as the causal code sequence

$$\boldsymbol{v} = \boldsymbol{v}_0\boldsymbol{v}_1 \ldots = v_0^{(1)}v_0^{(2)} \ldots v_0^{(c)}v_1^{(1)}v_1^{(2)} \ldots v_1^{(c)} \ldots \tag{1.86}$$

where

$$\boldsymbol{v}_t = f\left((\boldsymbol{u}_t, \boldsymbol{u}_{t-1}, \ldots, \boldsymbol{u}_{t-m}\right) \tag{1.87}$$

The parameter m is called the encoder *memory*. The function f is required to be a *linear* function from $\mathbb{F}_2^{(m+1)b}$ to $\mathbb{F}_2^c$. It is often convenient to write such a function in matrix form:

$$\boldsymbol{v}_t = \boldsymbol{u}_t G_0 + \boldsymbol{u}_{t-1}G_1 + \cdots + \boldsymbol{u}_{t-m}G_m \tag{1.88}$$

where $G_i, 0 \leq i \leq m$, is a binary $b \times c$ matrix.

Using (1.88), we can rewrite the expression for the code sequence as

$$\boldsymbol{v}_0\boldsymbol{v}_1 \ldots = (\boldsymbol{u}_0\boldsymbol{u}_1 \ldots)\boldsymbol{G} \tag{1.89}$$

or, in shorter notation, as

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G} \tag{1.90}$$

where

$$\boldsymbol{G} = \begin{pmatrix} G_0 & G_1 & \ldots & G_m & & \\ & G_0 & G_1 & \ldots & G_m & \\ & & \ddots & \ddots & & \ddots \end{pmatrix} \tag{1.91}$$

and where here and hereafter the parts of matrices left blank are assumed to be filled in with zeros. We call $\boldsymbol{G}$ the *generator matrix* and $G_i, 0 \leq i \leq m$, the *generator submatrices*.

In Fig. 1.13, we illustrate a general convolutional encoder (without feedack).

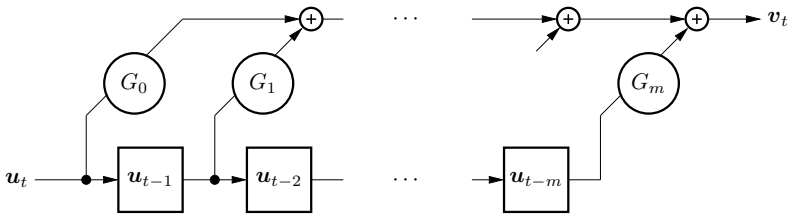

Figure 1.13 A general convolutional encoder (without feedback).

■ **EXAMPLE 1.17**

The rate $R = 1/2$ convolutional encoder shown in Fig. 1.12 has the following generator submatrices:

$$G_0 = (11) \tag{1.92}$$
$$G_1 = (10) \tag{1.93}$$
$$G_2 = (11) \tag{1.94}$$

The generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 11 & & \\ & 11 & 10 & 11 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.95}$$

■ **EXAMPLE 1.18**

The rate $R = 2/3$ convolutional encoder shown in Fig. 1.14 has generator submatrices

$$\begin{aligned} G_0 &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \\ G_1 &= \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ G_2 &= \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{aligned} \tag{1.96}$$

The generator matrix is

$$\boldsymbol{G} = \begin{pmatrix} 101 & 110 & 000 & & \\ 011 & 001 & 101 & & \\ & 101 & 110 & 000 & \\ & 011 & 001 & 101 & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \tag{1.97}$$

It is often convenient to represent the codewords of a convolutional code as paths through a *code tree*. A convolutional code is sometimes called a (linear) *tree code*. The code tree for the convolutional code generated by the encoder in Fig. 1.12 is shown in Fig. 1.15. The leftmost node is called the *root*. Since the encoder has one binary input, there are, starting at the root, two branches stemming from each node. The upper branch leaving each node corresponds to the input digit 0, and the lower branch corresponds to the input digit 1. On each branch we have two binary code digits, the two outputs from the encoder. The information sequence $1011\ldots$ is clearly seen from the tree to be encoded as the code sequence $11\,10\,00\,01\ldots$.

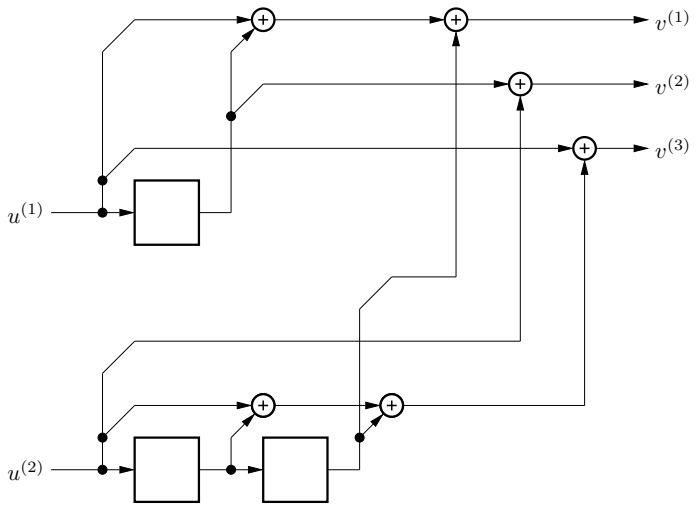

Figure 1.14 A rate $R = 2/3$ convolutional encoder.

The *state* of a system is a description of its past history which, together with a specification of the present and future inputs, suffices to determine the present and future outputs. For the encoder in Fig. 1.12, we can choose the encoder state $\boldsymbol{\sigma}$ to be the contents of its memory elements; that is, at time t we have

$$\boldsymbol{\sigma}_t = u_{t-1}u_{t-2} \tag{1.98}$$

Thus, our encoder has only four different encoder states, and two consecutive input digits are enough to drive the encoder to any specified encoder state.

For convolutional encoders, it is sometimes useful to draw the *state-transition diagram*. If we ignore the labeling, the state-transition diagram is a *de Bruijn graph* [Gol67]. In Fig. 1.16, we show the state-transition diagram for our convolutional encoder.

Let us return to the tree code in Fig. 1.15. As an example, the two input sequences 010 (node A) and 110 (node B) both drive the encoder to the same encoder state, $\boldsymbol{\sigma} = 01$. Thus, the two subtrees stemming from these two nodes are identical! Why treat them separately? We can replace them with one node corresponding to state 01 at time 3. For each time or *depth* in the tree, we can similarly replace all equivalent nodes with only one—we obtain the *trellis*-like structure shown in Fig. 1.17, where the upper and lower branches leaving the encoder states correspond to information symbols 0 and 1, respectively.

The information sequence $1011\ldots$ corresponds in the trellis to the same code sequence as in the tree, $11\,10\,00\,01\ldots$. The trellis is just a more convenient representation of the same set of encoded sequences as is specified by the tree, and it is easily constructed from the state-transition diagram. A convolutional code is often called a (linear) *trellis code*.

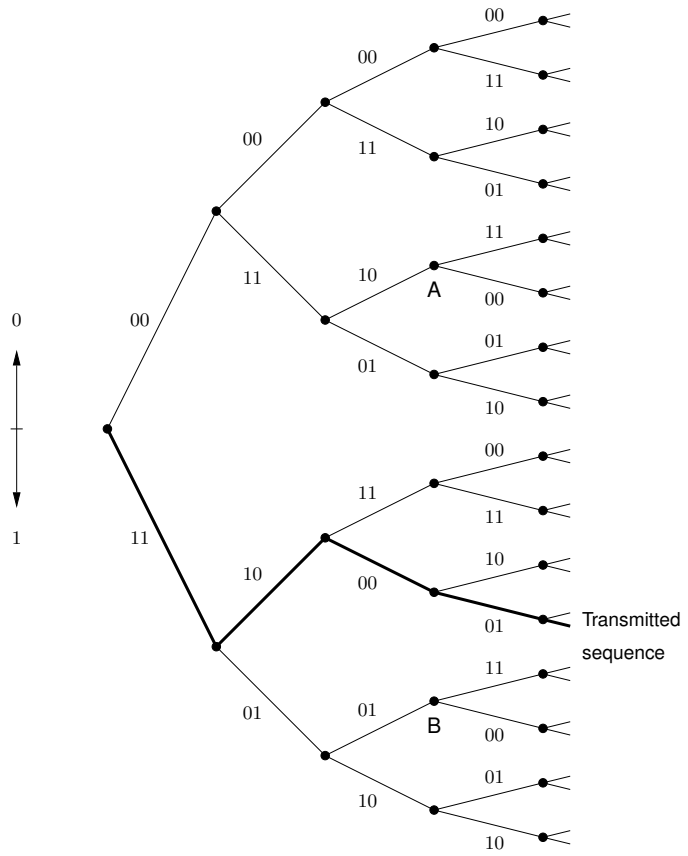

Figure 1.15 A binary rate $R = 1/2$ tree code.

We will often consider sequences of finite length; therefore, it is convenient to introduce the notations

$$\boldsymbol{x}_{[0,n)} = \boldsymbol{x}_0\boldsymbol{x}_1 \ldots \boldsymbol{x}_{n-1} \tag{1.99}$$

and

$$\boldsymbol{x}_{[0,n]} = \boldsymbol{x}_0\boldsymbol{x}_1 \ldots \boldsymbol{x}_n \tag{1.100}$$

Suppose that our trellis code in Fig. 1.17 is used to communicate over a BSC with crossover probability ϵ, where $0 < \epsilon < 1/2$. We start the encoder in encoder state $\boldsymbol{\sigma} = 00$, and feed it with the finite information sequence $\boldsymbol{u}_{[0,n)}$ followed by $m = 2$ dummy zeros in order to drive the encoder back to encoder state $\boldsymbol{\sigma} = 00$. The convolutional code is terminated and, thus, converted into a block code. The corresponding encoded sequence is the codeword $\boldsymbol{v}_{[0,n+m)}$. The received sequence is denoted $\boldsymbol{r}_{[0,n+m)}$.

To simplify the notations in the following discussion, we simply write $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{r}$ instead of $\boldsymbol{u}_{[0,n)}$, $\boldsymbol{v}_{[0,n+m)}$, and $\boldsymbol{r}_{[0,n+m)}$.

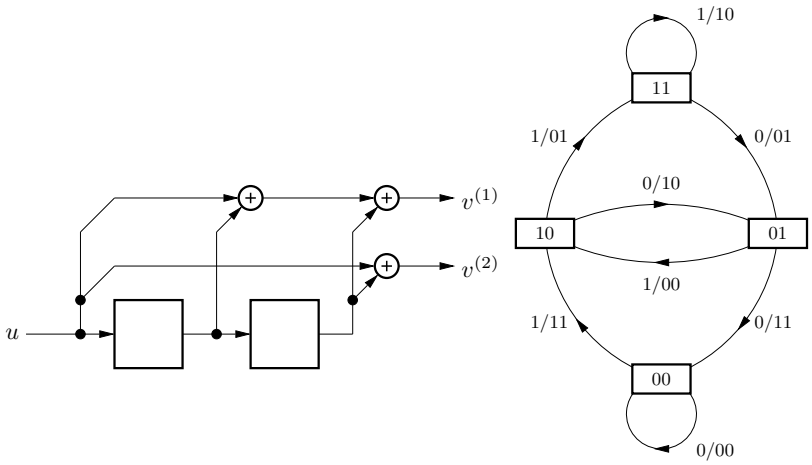

Figure 1.16 A rate $R = 1/2$ convolutional encoder and its state-transition diagram.

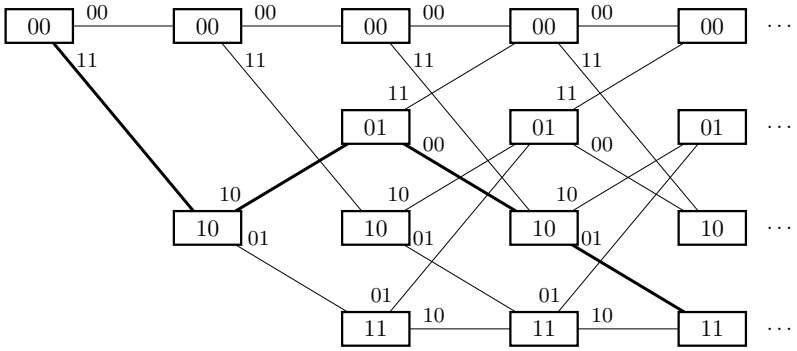

Figure 1.17 A binary rate $R = 1/2$ trellis code.

We shall now, by an example, show how the structure of the trellis can be exploited to perform maximum-likelihood (ML) decoding in a very efficient way. The memory $m = 2$ encoder in Fig. 1.12 is used to encode three information digits together with $m = 2$ dummy zeros; the trellis is terminated and our convolutional code has become a block code. A codeword consisting of 10 code digits is transmitted over a BSC. Suppose that $\boldsymbol{r} = 11\,00\,11\,00\,10$ is received. The corresponding trellis is shown in Fig. 1.18. (In practice, typically a few thousand information bits are encoded before the encoder is forced back to the allzero state by encoding m dummy zeros.)

As shown by the discussion following (1.31), the ML decoder (and the MD decoder) chooses as its decision $\widehat{\boldsymbol{v}}$ for the codeword $\boldsymbol{v}$ that minimizes the Hamming distance $d_{\mathrm{H}}(\boldsymbol{r}, \boldsymbol{v})$ between $\boldsymbol{r}$ and $\boldsymbol{v}$. That is, it minimizes the number of positions in which the codeword and the received sequence differ. In order to find the codeword that is closest to the received sequence, we move through the trellis from left to

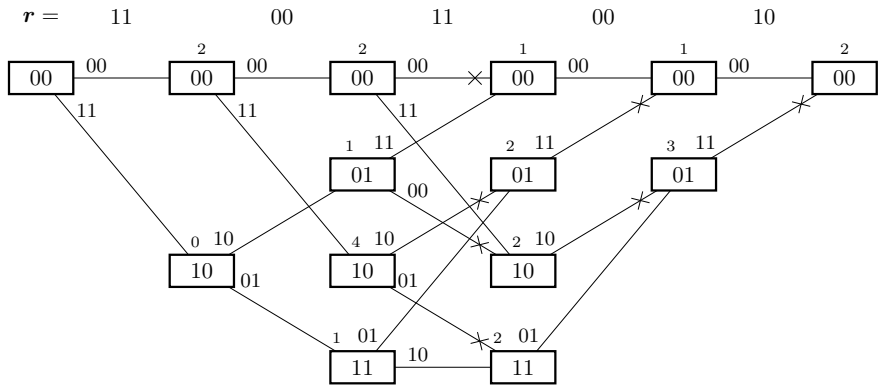

Figure 1.18 An example of Viterbi decoding for the received sequence $\boldsymbol{r} = 11\,00\,11\,00\,10$.

right, discarding *all* subpaths that could not turn out to be the prefix of the best path through the trellis. When we reach depth $m = 2$, we have four subpaths—one for each encoder state. At the next depth, however, there are eight subpaths—two per encoder state. For each encoder state at this depth, we keep only one subpath—the one that is closest to the corresponding prefix of the received sequence. We simply discard the poorer subpath into each encoder state since this poorer subpath could not possibly be the prefix of the best path through the trellis. We proceed in this manner until we reach encoder state 00 at depth 5. Because only one path through the trellis has survived, we have then found the best path through the trellis. In Fig. 1.18, the Hamming distance between the prefix of the received sequence and the best subpath leading to each encoder state is shown above the encoder state. The discarded poorer subpath is marked with the symbol $\times$ on the branch that enters the encoder state.

Two subpaths leading to an encoder state may both have the same Hamming distance to the prefix of the received sequence. In fact, this happened at state 01, depth 4. Both subpaths have distance 3 to the prefix of the received sequence! Both are equally likely to be the prefix of the best path—we can discard either subpath without eliminating all "best paths" through the trellis, in case there are more than one best path. We arbitrarily chose to discard the upper of the two subpaths entering encoder state 01 at depth 4.

The best codeword through the trellis was found to be $\widehat{\boldsymbol{v}} = 11\,10\,11\,00\,00$, which corresponds to the information sequence $\widehat{\boldsymbol{u}} = 100$. If the decision $\widehat{\boldsymbol{v}} = 11\,10\,11\,00\,00$ happened to be the transmitted codeword, we have corrected two transmission errors.

How many errors can we correct?

The most likely error event is that the transmitted codeword is changed by the BSC so that it is decoded as its closest (in Hamming distance) neighbor. It is readily seen from Fig. 1.18 that the smallest Hamming distance between any two different codewords is 5, for example, $d_{\mathrm{H}}\,(00\,00\,00\,00\,00,\ 11\,10\,11\,00\,00) = 5$. This minimum distance is called the *free distance* of the convolutional code and is denoted d_{free}. It is the single most important parameter for determining the error-correcting

capability of the code. (The free distance and several other distance measures will be discussed in detail in Chapter 3.) Since $d_{\text{free}} = 5$, we can correct all patterns of two errors.

The ML decoding algorithm described above is usually called the *Viterbi algorithm* in honor of its inventor [Vit67]. It is as simple as it is ingenious, and it is easily implementable. Viterbi decoders for memory $m = 6$ (64 states) and longer are often used in practice.

In Chapter 4, we will study Viterbi decoding in more detail and obtain tight upper bounds on the decoded bit error probability.

1.5 BLOCK CODES VERSUS CONVOLUTIONAL CODES

The system designer's choice between block and convolutional codes should depend on the application. The diehard block code supporters always advocate in favor of block codes, while their counterparts on the other side claim that in almost all situations convolutional codes outperform block codes. As always, the "truth" is not only somewhere in between, but it also depends on the application.

The theory of block codes is much richer than the theory of convolutional codes. Many sophisticated finite field concepts have been used to design block codes with a beautiful mathematical structure that has simplified the development of efficient error-correcting decoding algorithms. From a practical point of view, the Reed-Solomon (RS) codes constitute the most important family of block codes. They are extremely well suited for digital implementation. Berlekamp's bit-serial RS encoders [Ber82] have been adopted as a NASA standard for deep-space communication. The RS codes are particularly powerful when the channel errors occur in clusters—*burst errors*—which is the case in secondary memories such as magnetic tapes and disks. All compact disc players use RS codes with table-look-up decoding.

Assuming that a decoded bit error rate of 10^{-5} is satisfactory, which is the case, for example, for digitized voice, convolutional codes in combination with Viterbi decoding appear to be an extremely good combination for communication when the noise is white and Gaussian. For example, Qualcomm Inc. has on a single chip implemented a memory $m = 6$ Viterbi decoder for rates $R = 1/3,\ 1/2,\ 3/4,\ 7/8$. The rate $R = 1/2$ coding gain is 5.2 dB at $P_{\text{b}} = 10^{-5}$. This very powerful error-correcting system operates either with hard decisions or with eight-level quantized soft decisions.

The major drawback of RS codes is the difficulty of making full use of soft-decision information. As we will see in Chapter 4, the Viterbi algorithm can easily exploit the full soft-decision information provided at the decoder input and thus easily pick up the 2 dB gain over hard-decision. Furthermore, code synchronization is in general much simpler for convolutional codes than for block codes.

If a combination of a high level of data integrity, $P_{\text{b}} = 10^{-10}$ say, and a larger coding gain than a Viterbi decoder can provide is required, then we could use either an RS code or a large memory, $m = 25$ say, convolutional encoder. The complexity of the Viterbi decoder, which is essentially 2^m, will be prohibitively large in the latter

case. Instead we could use *sequential decoding* (Chapter 7) of the convolutional code whose complexity is essentially independent of the memory of the encoder.

In many applications where the noise is predominantly Gaussian, the best solution is obtained when block and convolutional codes join forces and are used in series. In Fig. 1.19 we show a *concatenated coding* system, where we use a convolutional code as the *inner code* to clean up the channel. The Viterbi decoder will correct most channel errors but will occasionally output a burst of errors. This output then becomes the input to the outer decoder. Since an RS code is very well suited to cope with bursts of errors, we use an RS code as the *outer code*. Such a concatenated coding system combines a very high level of data integrity with large coding gain and low complexity. Often a *permutor* is used between the outer and inner encoders and a corresponding *inverse permutor* between the inner and outer decoders. Then the output error burst from the inner decoder will be smeared out by the inverse permutor before the outer decoder has to cope with it.

An alternative method of decreasing the decoding complexity without decreasing the code reliability is using low-density parity-check (LDPC) codes, a class of codes on graphs, or so-called turbo codes. In Chapter 8 we discuss LDPC convolutional codes and in Chapter 9 we introduce turbo codes.

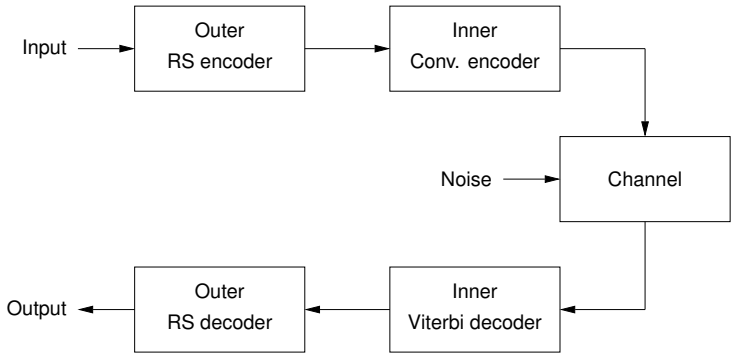

Figure 1.19 Concatenated coding system.

1.6 CAPACITY LIMITS AND POTENTIAL CODING GAIN REVISITED

We will now return to the problem of determining the regions of potential coding gain which we first encountered in Section 1.1.

Consider Shannon's formula for the capacity of the bandlimited Gaussian channel (1.15),

$$C_t^W = W \log \left(1 + \frac{S}{N_0 W}\right) \text{ bits/s}$$

where W as before denotes the bandwidth. Assume that we are transmitting at the so-called *Nyquist rate*, (that is, at a rate of $2W$ samples per second) and that we use

a rate $R = K/N$ block code. If we transmit K information bits during $\mathcal{T}$ seconds, we have

$$N = 2W\mathcal{T} \text{ samples per codeword} \tag{1.101}$$

Hence,

$$R_t = K/\mathcal{T} = 2WK/N = 2WR \text{ bits/s} \tag{1.102}$$

(Assuming a constant transmission rate R_t, the required bandwidth W is inversely proportional to the code rate R.)

By combining (1.18) and (1.102), we obtain

$$\frac{S}{WN_0} = \frac{2RE_\text{b}}{N_0} \tag{1.103}$$

For reliable communication, we must have $R_t \leq C_t^W$, that is,

$$R_t = 2WR \leq W \log \left(1 + \frac{2RE_\text{b}}{N_0}\right) \tag{1.104}$$

or, equivalently,

$$\frac{E_\text{b}}{N_0} \geq \frac{2^{2R} - 1}{2R} \tag{1.105}$$

Letting $R \to 0$, we obtain (1.20). Since the right hand side of inequality (1.105) is increasing with R, we notice that in order to communicate close to the Shannon limit, -1.6 dB, we have to use both an information rate R_t and a code rate R close to zero. Furthermore, if we use a rate $R = 1/2$ code, it follows from (1.105) that the required signal-to-noise ratio is

$$E_\text{b}/N_0 \geq 1 = 0 \text{ dB} \tag{1.106}$$

In Fig. 1.20 we show the coding limits according to (1.105) and the regions of potential coding gain for various rates R.

When we derived the coding limits determined by inequality (1.105), we assumed a required error probability P_b arbitrarily close to zero. If we are willing to tolerate a certain given value of the error probability P_b, we can of course obtain a larger coding gain. It follows from Shannon's *rate distortion theory* [McE77] that if we are transmitting the output of a binary symmetric source and can tolerate an average distortion of KP_b for a block of K information symbols, then we can represent R_t bits of information per second with only $R_t(1 - h(P_\text{b}))$ bits per second, where $h(\cdot)$ is the binary entropy function (1.22).

These $R_t(1 - h(P_\text{b}))$ bits per second should now be transmitted over the channel with an error probability arbitrarily close to zero. Hence, instead of (1.104) we have now the inequality

$$R_t(1 - h(P_\text{b})) = 2WR(1 - h(P_\text{b})) \leq W \log \left(1 + \frac{2RE_\text{b}}{N_0}\right) \tag{1.107}$$

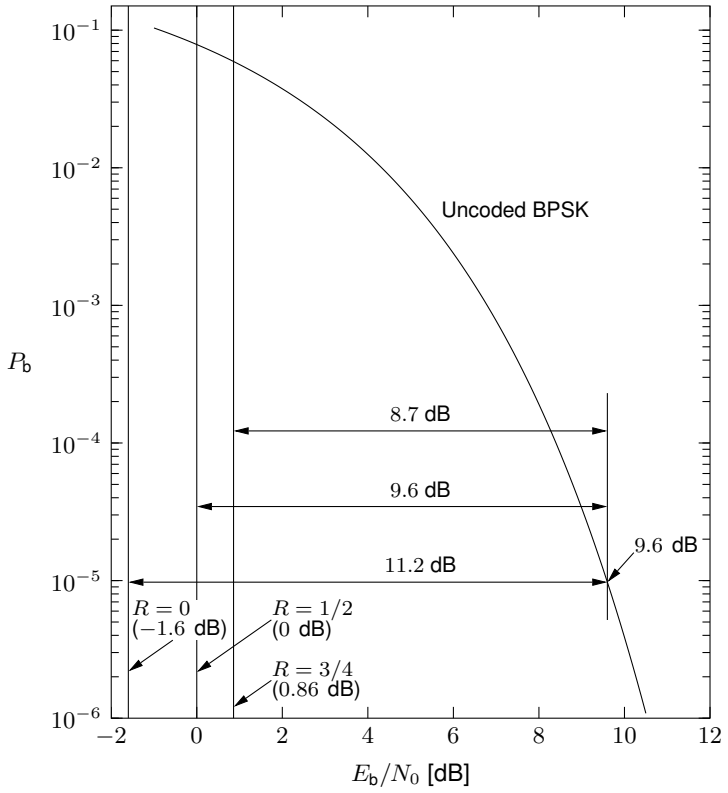

Figure 1.20 Coding limits and regions of potential coding gains for various rates R.

or, equivalently,

$$\frac{E_b}{N_0} \geq \frac{2^{2R(1-h(P_b))} - 1}{2R} \tag{1.108}$$

In Fig. 1.21 we show the coding limits according to (1.108) and the regions of potential coding gains for various rates R when we can tolerate the bit error probability P_b. We also show a comparison between these coding limits and Qualcomm's Viterbi decoder performance and that of a rate $R = 3/4$ $(256, 192)$ RS decoder.

In order to achieve the rate distortion bound we need a nonlinear source encoder. Hence, we have *not* shown that the coding limit (1.108) can be reached with *linear* codes.

Remark: In order to achieve the capacity C_t^W promised by (1.15), we have to use nonquantized inputs to the channel. If we restrict ourselves to the binary input Gaussian channel, then the formula for C_t^W, (1.15), must be replaced by a more complicated expression and the coding limits shown in Fig. 1.21 should be shifted to the right by a small fraction of a dB [BMc74].

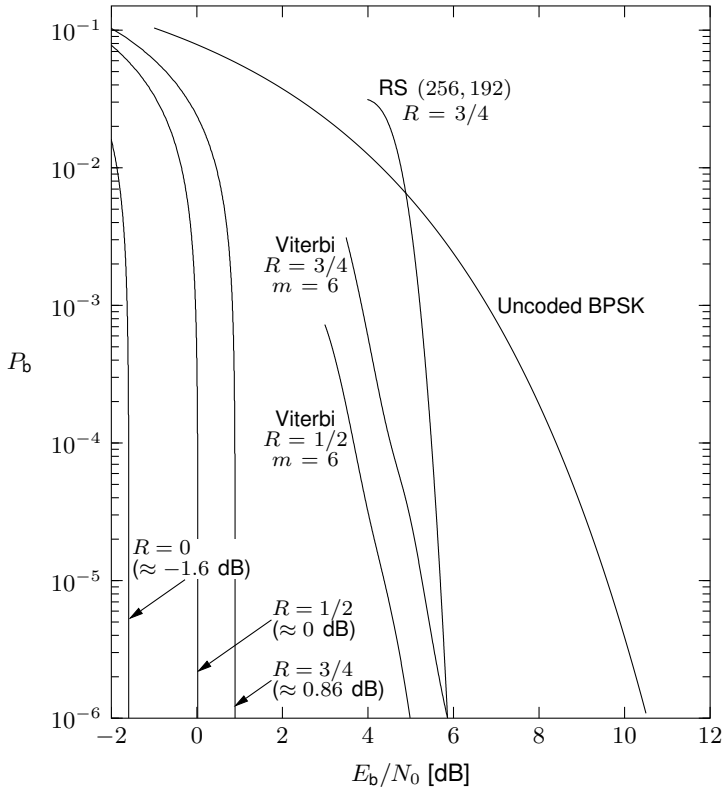

Figure 1.21 Regions of potential coding gains for various rates R when we can tolerate bit error probability P_b and a comparison with the performance of two convolutional codes and a block code.

1.7 COMMENTS

Back in 1947 when Hamming had access to a computer only on weekends, he was very frustrated over its behavior: "Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?" [Tho83]. That question inspired the development of error-correcting codes. Hamming's famous single-error-correcting $(7, 4)$ block code is mentioned by Shannon in "A Mathematical Theory of Communication" [Sha48], but Hamming's paper was not published until two years later [Ham50]. The first paper published in coding theory was that of Golay [Gol49], which in less than one page gave the generalization of Hamming codes to nonbinary fields, gave the only two multi-error-correcting perfect codes aside from the trivial binary repetition codes of odd length, and introduced the parity-check matrix (see also Problem 1.19).

Elias introduced convolutional codes in 1955 [Eli55]. The first decoding method for these codes was sequential decoding suggested by Wozencraft in 1957 [Woz57]

and further developed by Fano, who in 1963 presented a most ingenious decoding algorithm [Fan63]. The conceptually simplest algorithm for sequential decoding is the stack algorithm introduced by Zigangirov in 1966 [Zig66] and Jelinek in 1969 [Jel69]. In the meantime, Massey had suggested threshold decoding of convolutional codes [Mas63]. In Viterbi's famous paper from 1967 [Vit67], the Viterbi algorithm was invented as a proof technique and presented as "a new probabilistic nonsequential decoding algorithm". Forney [For67] was the first to draw a trellis and it was he who coined the name "trellis," which made understanding of the Viterbi algorithm easy and its maximum-likelihood nature obvious. Forney realized that the Viterbi algorithm was optimum, but it was Heller who realized that it was practical [For94]. Later, Omura [Omu69] observed that the Viterbi algorithm can be viewed as the application of dynamic programming to the problem of decoding a convolutional code.

The most important contributions promoting the use of convolutional codes were made by Jacobs and Viterbi when they founded Linkabit Corporation in 1968 and Qualcomm Inc. in 1985, completing the path "from a proof to a product" [Vit90].

LDPC block codes were invented by Gallager [Gal62, Gal63] in the early 1960s. Unfortunately, Gallager's remarkable discovery was to a large extent ignored by the coding community during almost 20 years. Two important papers by Zyablov and Pinsker [ZyP74, ZyP75] published in the middle of the 1970s were overlooked by many coding theoretists. In the beginning of the 1980s Tanner [Tan81] and Margulis [Mar82] published two important papers concerning LDPC codes. Tanner's work provided a new interpretation of LDPC codes from a graph theoretical point of view. Margulis gave explicit graph constructions of the codes. These works were also essentially ignored by the coding specialists for more than 10 years, until the beginning 1990s when Berrou, Glavieux, and Thitimajshima [BGT93] introduced the so-called *turbo codes* which inspired many coding researchers to investigate codes on graphs and iterative decoding. It has been shown that long LDPC codes with iterative decoding based on belief propagation almost achieve the Shannon limit. This rediscovery makes the LDPC codes strong competitors with other codes for error control in many communication and digital storage systems when high reliability is required.

PROBLEMS

1.1 The channel capacity for the ideal bandlimited AWGN channel of bandwidth W with two-sided noise power spectral density $N_0/2$ is given by (1.15). The signal power can be written $S = E_\mathrm{b} R_t$.

Define the *spectral bit rate* r by

$$r = R_t/W \text{ (bits/s)/Hz}$$

and show that

$$\frac{E_\mathrm{b}}{N_0} \geq \frac{2^r - 1}{r}$$

for rates R_t less than capacity. Sketch r as a function of E_b/N_0 expressed in dB.

1.2 Consider an ideal bandlimited AWGN channel with BPSK and with hard decisions. Based on transmitting $R_t = 2WR$ bits/s, where R is the code rate, the capacity is

$$C_t = 2W(1 + \epsilon \log \epsilon + (1-\epsilon)\log(1-\epsilon)) \text{ bits/s}$$

where $\epsilon = Q\left(\sqrt{rE_\mathrm{b}/N_0}\right)$ and r is the spectral bit rate $r = R_t/W$.

Show that

$$\frac{E_\mathrm{b}}{N_0} \geq \frac{\pi}{2} \ln 2$$

for reliable communication.

Hint: The Taylor series expansion of C_t is

$$C_t = 2W\left(\frac{2^2}{1\cdot 2}(\tfrac{1}{2}-\epsilon)^2 + \frac{2^4}{3\cdot 4}(\tfrac{1}{2}-\epsilon)^4 + \frac{2^6}{5\cdot 6}(\tfrac{1}{2}-\epsilon)^6 + \cdots\right)\log e$$

and

$$\epsilon = Q\left(\sqrt{rE_\mathrm{b}/N_0}\right) \geq \frac{1}{2} - \frac{1}{\sqrt{2\pi}}\sqrt{rE_\mathrm{b}/N_0}$$

1.3 Show that a block code $\mathcal{B}$ can detect all patterns of s or fewer errors if and only if $d_{\min} > s$.

1.4 Show that a block code $\mathcal{B}$ can correct all patterns of t or fewer errors and simultaneously detect all patterns of $t+1, t+2, \ldots, t+s$ errors if and only if $d_{\min} > 2t + s$.

1.5 Prove Theorem 1.2.

1.6 Consider a block code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

a) List all codewords.

b) Find a systematic encoding matrix and its parity-check matrix.

c) Determine $d_{\min}$.

1.7 Consider the following two block codes.

$\mathcal{B}_1 = \{110011, 101010, 010101, 011001, 100110, 111111, 001100, 000000\}$

$\mathcal{B}_2 = \{010101, 101010, 001100, 110110, 111111, 011001, 110011, 100110\}$

a) Are the two codes linear?

b) Determine $w_{\min}$ for each of the codes.

c) Determine $d_{\min}$ for each of the codes.

d) Determine the rate $R = K/N$.

1.8 Consider the block code $\mathcal{B} = \{000000, 110110, 011011, 101101\}$.

a) Is $\mathcal{B}$ linear?

b) Find the rate $R = K/N$.

c) Find, if it exists, a linear encoder.

d) Find, if it exists, a nonlinear encoder.

e) Determine $d_{\min}$.

1.9 Show that if $\mathcal{B}$ is a linear code and $\boldsymbol{a} \notin \mathcal{B}$, then $\mathcal{B} \cup (\boldsymbol{a} + \mathcal{B})$ is also a linear code.

1.10 Consider the binary $(6, K)$ *even-weight* code. (All codewords have even weight.)

a) Find K.

b) Give the encoding and parity-check matrices.

1.11 Consider the binary (4,3) even-weight code.

a) Construct a standard array.

b) For each coset give its syndrome.

c) How many errors can it correct?

d) Determine $d_{\min}$.

1.12 Show that a binary code can correct all single errors if and only if any parity-check matrix has distinct nonzero columns.

1.13 Consider a binary code with encoding matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

a) Find a parity-check matrix.

b) Construct a standard array.

c) List all codewords.

d) Determine from the standard array how many errors it can correct.

e) Determine $d_{\min}$.

f) For each coset give its syndrome.

g) Suppose that $\boldsymbol{r} = 110000$ is received over a BSC with $0 < \epsilon < 1/2$. Find the maximum-likelihood decision $\widehat{\boldsymbol{u}}$ for the information sequence.

1.14 Consider a block code $\mathcal{B}$ with encoding matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

a) Find a parity-check matrix.
b) List all codewords.
c) Determine $d_{\min}$.
d) Suppose that $\boldsymbol{r} = 000111$ is received over a BSC with $0 < \epsilon < 1/2$. Find the maximum-likelihood decision $\widehat{\boldsymbol{u}}$ for the information sequence.

1.15 Consider a binary (N, K) code $\mathcal{B}$ with parity-check matrix H and minimum distance d. Assume that some of its codewords have odd weight. Form a code $\widehat{\mathcal{B}}$ by concatenating a 0 at the end of every codeword of even weight and a 1 at the end of every codeword of odd weight. This technique is called *extending* a code.
a) Determine $d_{\min}$ for $\widehat{\mathcal{B}}$.
b) Give a parity-check matrix for the extended code $\widehat{\mathcal{B}}$.

1.16 Consider the (8,4) extended Hamming code.
a) Give a parity-check matrix.
b) Determine $d_{\min}$.
c) Find an encoding matrix.
d) Show how a decoder can detect that an odd number of errors has occurred.

1.17 The *Hamming sphere* of radius t with center at the N-tuple $\boldsymbol{x}$ is the set of all $\boldsymbol{y}$ in $\mathbb{F}_2^N$ such that $d_{\mathrm{H}}(\boldsymbol{x}, \boldsymbol{y}) \leq t$. Thus, this Hamming sphere contains exactly

$$V_t = \sum_{i=0}^{t} \binom{N}{i}$$

distinct N-tuples. Prove the *Hamming bound* for binary codes, that is,

$$V_{\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor} \leq 2^{N(1-R)}$$

which is an implicit upper bound on $d_{\min}$ in terms of the block length N and rate R.

1.18 The systematic parity-check matrices for the binary Hamming codes can be written recursively as

$$H_2 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

and

$$H_m = \begin{pmatrix} H_{m-1} & H_{m-1} & \mathbf{0} \\ 1 \ldots 1 & 0 \ldots 0 & 1 \end{pmatrix}, \quad m \geq 3$$

Find the parameters N, K, and $d_{\min}$ for the mth Hamming code.

1.19 A code for which the Hamming bound (see Problem 1.17) holds with equality is called a *perfect code*.

a) Show that the *repetition code*, that is, the rate $R = 1/N$ binary linear code with generator matrix $G = (1\,1\,\ldots\,1)$, is a perfect code if and only if N is odd.

b) Show that the Hamming codes of Problem 1.18 are perfect codes.

c) Show that the Hamming bound admits the possibility that an $N = 23$ perfect binary code with $d_{\rm dim} = 7$ might exist. What must K be?

Remark: The perfect code suggested in Problem 1.19(c) was found by Golay in 1949 [Gol49]. There exist no perfect binary codes other than those mentioned in this problem.

1.20 Suppose that the block code $\mathcal{B}$ with parity-check matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

is used for communication over a BSC with $0 < \epsilon < 1/2$.

a) Find $d_{\rm min}$.

b) How many errors can the code correct?

c) How many errors can the code detect?

d) For each syndrome give the error pattern $\widehat{\boldsymbol{e}}$ that corresponds to the error-correcting capability of the code.

e) For $\boldsymbol{r} = 0111011$ find $\widehat{\boldsymbol{v}}$, the maximum-likelihood decision.

f) For $\boldsymbol{r} = 0110111$ find $\widehat{\boldsymbol{v}}$, the maximum-likelihood decision.

1.21 Verify formula (1.73).

Hint: The $(7, 4)$ Hamming code has one codeword of weight 0, seven codewords of weight 3, seven codewords of weight 4, and one codeword of weight 7. The error probability is the same for all bits.

1.22 Given a Hamming code $\mathcal{B}$ with parity-check matrix H.

a) Construct an *extended* code $\mathcal{B}_{\rm ext}$ with parity-check matrix

$$H_{\rm ext} = \begin{pmatrix} 0 & & & \\ \vdots & & H & \\ 0 & & & \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

b) Determine $d_{\rm min}$ for $\mathcal{B}_{\rm ext}$.

c) Construct an *expurgated* code $\mathcal{B}_{\rm exp}$ with parity-check matrix

$$H_{\rm exp} = \begin{pmatrix} & H & \\ & & \\ 1 & \cdots & 1 \end{pmatrix}$$

d) Determine $d_{\rm min}$ for $\mathcal{B}_{\rm exp}$.

e) What is characteristic for the weights of the codewords of $\mathcal{B}_{\text{exp}}$?

1.23 Draw the Tanner graph for the extended $(8, 4)$ Hamming code defined by the parity-check matrix

$$H = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

1.24 Show that the parity-check matrix (1.77) has rank less than L.

1.25 Consider the code given in Example 1.15. Show that the bit-flipping algorithm with lowest possible threshold does not correct all single errors.

1.26 Consider the extended $(8, 4)$ Hamming code defined in Problem 1.23 and suppose that it is used to communicate over the BSC. Show that the bit-flipping algorithm with adaptive threshold corrects all single errors and that there exists a double error which the algorithm does not correct.

1.27 Consider the binary input, ternary output *binary erasure channel* (BEC) given in Fig. 1.22, where Δ denotes an erasure and δ is the probability of an erasure. Assume that we use this channel for communication together with maximum-likelihood (ML) decoding. Show that the ML decoding algorithm corrects all erasure patterns whose Hamming weights are less than the minimum distance $d_{\min}$.

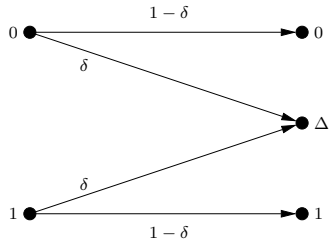

Figure 1.22 BEC used in Problem 1.27.

1.28 Consider the trellis given in Fig. 1.18.

a) List all codewords.

b) Find the ML estimate of the information sequence for the received sequence $\boldsymbol{r} = 01\,10\,01\,10\,11$ on a BSC with $0 < \epsilon < 1/2$.

1.29 Consider the convolutional encoder shown in Fig. 1.23.

a) Draw the trellis corresponding to four information digits and $m = 1$ dummy zero.

b) Find the number of codewords M represented by the trellis in Problem 1.29(a).

c) Use the Viterbi algorithm to decode when the sequence $\boldsymbol{r} = 11\,01\,10\,10\,01$ is received over a BSC with $0 < \epsilon < 1/2$.

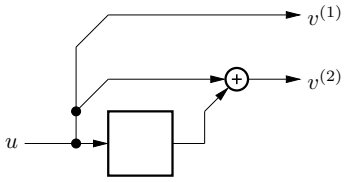

Figure 1.23 Convolutional encoder used in Problem 1.29.

1.30 Consider the convolutional encoder with generator matrix

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 01 & 11 & & \\ & 11 & 10 & 01 & 11 & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

a) Find the rate and the memory.
b) Draw the encoder.
c) Find the codeword $\boldsymbol{v}$ that corresponds to the information sequence $\boldsymbol{u} = 1100100\ldots$.

1.31 Consider the code $\mathcal{C}$ with the encoding rule

$$\boldsymbol{v} = \boldsymbol{u}\boldsymbol{G} + (11\,01\,11\,10\,11\,\ldots)$$

where

$$\boldsymbol{G} = \begin{pmatrix} 11 & 10 & 01 & 11 & & \\ & 11 & 10 & 01 & 11 & \\ & & \ddots & \ddots & \ddots & \ddots \end{pmatrix}$$

a) Is the code $\mathcal{C}$ linear?
b) Is the encoding rule linear?

1.32 Consider the rate $R = 2/3$, memory $m = 2$ convolutional encoder illustrated in Fig. 1.14.
a) Draw the trellis diagram.
b) Find the encoder matrix $\boldsymbol{G}$.
c) Let $\boldsymbol{u} = 10\,11\,01\,10\,00\,00\ldots$ be the information sequence. Find the corresponding codeword $\boldsymbol{v}$.

1.33 Plot in Fig. 1.21 the bit error probability for the $(7, 4)$ Hamming code when used to communicate over the Gaussian channel with hard decisions.
Hint: From formula (1.12), that is, $\epsilon = Q\left(\sqrt{2E_\mathrm{s}/N_0}\right)$, where $E_\mathrm{s} = RE_\mathrm{b}$, we obtain the following table:

E_s/N_0 [dB]	ϵ
0	$0.79 \cdot 10^{-1}$
2	$0.38 \cdot 10^{-1}$
4	$0.12 \cdot 10^{-1}$
6	$0.24 \cdot 10^{-2}$
8	$0.19 \cdot 10^{-3}$
10	$0.39 \cdot 10^{-5}$
12	$0.90 \cdot 10^{-8}$