NASA Reference Publication 1367

December 1996

Introduction to Forward-Error-Correcting Coding

Jon C. Freeman

National Aeronautics and Space Administration

Lewis Research Center
Cleveland, Ohio 44135

NASA Reference Publication 1367

1996

Introduction to Forward-Error-Correcting Coding

Jon C. Freeman
Lewis Research Center
Cleveland, Ohio

National Aeronautics and Space Administration

Office of Management

Scientific and Technical Information Program

Preface

The purpose of these notes is to provide a practical introduction to forward-error-correcting coding principles. The document is somewhere between a review and a how-to handbook. Emphasis is on terms, definitions, and basic calculations that should prove useful to the engineer seeking a quick look at the area. To this end, 41 example problems are completely worked out. A glossary appears at the end, as well as an appendix concerning the Q function. The motivation for this document is as follows: The basic concepts of coding can be found in textbooks devoted to communications principles or in those dealing exclusively with coding. Although each is admirable in its intent, no elementary treatment, useful for quick calculations on the job, exists. I have taken a short course on coding, given by Prof. E.J. Weldon, Jr., as well as one given in-house at NASA Lewis. These notes are for those who have not had the time either to take such courses or to study the literature in some detail.

The material included is primarily that found in basic textbooks and short courses. The reader should not anticipate developing sufficient skills to actually design a code for a specific purpose. Rather, the reader should be far enough along the learning curve to be able to read and understand the technical literature (e.g., IEEE Transactions on Information Theory). The topics I chose to discuss here were those that almost always cropped up in the references and apparently are the ones the beginner should learn. The emphasis is on definitions, concepts, and analytical measures of performance whenever possible.

The "questions of coding" from an engineer's viewpoint may be stated as, Should coding be used? And if so, which code? What performance improvement can be expected? A basic measure of performance is the coding gain, but establishing an accurate formula is not a trivial exercise. Here, I summarize the essential process to determine approximate values. Some software packages are now available to permit simulations, but they are more appropriate for true experts on coding.

Here, I consider "coding" to be only forward error correcting (FEC), as opposed to other uses of the term, which are source coding, encryption, spreading, etc. In practice, code performance is modulation dependent; thus, the code should be matched to both the channel's characteristics and the demodulator's properties. This matching is seldom, if ever, done. Usually, some standard, well-established code is used, and its appropriateness is determined by the closeness of the bit error rate to system specifications.

A goal of these notes is to present an orderly development, with enough examples to provide some intuition. Chapter 1 reviews information theory and defines the terms "self, mutual, and transmitted information." Chapter 2 reviews channel transfer concepts. Chapter 3 treats modulo-2 arithmetic and channel terminology. Chapter 4 gives an overview of block coding. Chapter 5 goes deeper into block codes, their performance, and some decoder strategies and attempts to cover finite field algebra, so that the beginner can start reading the literature. A code may be looked upon as a finite set of elements that are processed by shift registers. The properties of such registers, along with those of the code

elements are used to produce coders and decoders (codecs). The mathematics of finite fields, often referred to as "modern algebra" is the working analysis tool in the area, but most engineers are not well grounded in its concepts. Chapter 6 introduces convolutional coders, and chapter 7 covers decoding of convolutional codes. Viterbi and sequential decoding strategies are treated.

No attempt at originality is stated or implied; the examples are blends of basic problems found in the references and in course notes from various short courses. Some are solutions of chapter problems that seemed to shed light on basic points. Any and all errors and incorrect "opinions" expressed are my own, and I would appreciate the reader alerting me of them.

Contents

Chapter 1
Information Theory

Both this and the following chapter discuss information, its measure, and its transmission through a communications channel. Information theory gives a quantitative measure of the "information content" in a given message, which is defined as the ordering of letters and spaces on a page. The intuitive properties of information are as follows:

1. A message with more information should occur less often than one with less information.
2. The more "uncertainty" contained in a message, the greater the information carried by that message. For example, the phrase "we are in a hurricane" carries more information than "the wind is 10 mph from the southwest."
3. The information of unrelated events, taken as a single event, should equal the sum of the information of the unrelated events.

These intuitive concepts of "information" force the mathematical definitions in this chapter. Properties 1 and 2 imply probability concepts, and these along with the last property imply a logarithmic functional relationship. In other words, the amount of information should be proportional to the message length, and it should increase appropriately with the richness of the alphabet in which it is encoded. The more symbols in the alphabet, the greater the number of different messages of length n that can be written in it.

The notion of self-information is introduced with two examples.

EXAMPLE 1.1

Assume a 26-character alphabet and that each character occurs with the same frequency (equally likely). Assume m characters per page, and let each page comprise a single message. Then, the total number of possible messages on a given page is determined as follows: Let the position of each character be called a slot; then,

1. First slot can be filled in 26 ways.
2. Second slot can be filled in 26 ways, etc.

Because there are m slots per page, there are (26)(26)...(26), that is, m terms and 26^m possible arrangements. (In general, the number of permutations N of k alphabetic symbols, taken n at a time, is

$$N = k^n$$

and each permutation is considered a message.)

Define each arrangement as a message. The number of possible messages on two pages is 26^{2m}. Now by intuition assume that two pages will carry twice as much information as does one page. Taking logarithms of the possible arrangements yields

$$\frac{\log\left[26^{2m}\right]}{\log\left[26^{m}\right]} = 2 = \frac{\text{information on 2 pages}}{\text{information on 1 page}}$$

Thus, the log of the total number of available messages seems to make some sense. (The end of an example will henceforth be designated with a triangle ▲.)

▲

Before moving on, I must discuss pulses, binary digits, and symbols. In general, a source of information (e.g., the digital modulator output) will emit strings of pulses. These may have any number of amplitudes, but in most cases only two amplitudes are used (thus, binary pulses). The two amplitudes are represented mathematically by the digits 0 and 1 and are called binary digits. Thus, electrical pulses and binary digits become synonymous in this area. Often, groups of binary digits are processed together in a system, and these groups are called symbols.

DEFINITIONS

Binit—A binary digit, 0 or 1. Also called a bit.

Baud —The unit of signaling speed, quite often the number of symbols transmitted per second. Note that although baud is a rate, quite often the words "baud rate" are given, so that the meaning is basically vague. The speed in bauds is equal to the number of "signaling elements" sent per second. These signaling elements may or may not be groups of binary digits (someone could mean amplitudes of sine waves, etc.). Therefore, a more general definition is the number of signal events per second. Baud is also given a time interval meaning; it is the time interval between modulation envelope changes. Also, one finds the phrase "the duration of a channel symbol."

EXAMPLE 1.2

Consider a source emitting symbols at a rate of $1/T$ symbols per second. Assume that m distinct message symbols are available, denoted by $x_1,x_2,x_3,\ldots,x_m$ and together are represented by x. For simplicity, at this point, assume that each symbol can occur with the same probability. The transmission of any single symbol will represent a certain quantity of information (call it I). Because all symbols are equally likely, it seems reasonable that all carry the same amount of information. Assume that I depends on m in some way.

$$I = f(m) \tag{a}$$

where f is to be determined. If a second symbol, independent of the first one, is sent in a succeeding interval, another quantity of information I is received. Assume that the information provided by both is $I + I = 2I$. Now, if there are m alternatives in one interval, there are m^2 alternative pairs in both intervals (taken as a single event in the time $2T$). Thus,

$$2I = f(m^2) \tag{b}$$

In general, for k intervals

$$kI = f(m^k) \tag{c}$$

The simplest function to satisfy equation (c) is log; thus,

$$f(m) = A \log m$$

where A is a constant of proportionality and the base of the log is immaterial. The common convention is to define the self-information of an m-symbol, equally likely source as

$$I = \log_2 m \qquad \text{bits} \tag{d}$$

when the base is chosen as 2. Observe that the unit for information measure is bits. The value of equation (d) is the quantitative measure of information content in any one of the m symbols that may be emitted. ▲

Observe in example 1.2 that

$$I = \log_2 m = -\log_2\left(\frac{1}{m}\right) = -\log_2(p_i) \tag{1.1}$$

The probability of any symbol occurring, $p_i = 1/m$, is used to generalize to the case where each message symbol x_i has a specified probability of occurrence p_i.

DEFINITION

Let x_i occur with probability p_i; then, the self-information contained in x_i is

$$I(x_i) \triangleq -\log_2 p(x_i) \qquad i = 1, \ldots, m \tag{1.2}$$

Next, the average amount of information in any given symbol is found for the entire ensemble of m available messages.

DEFINITION

$$\langle I(x_i) \rangle = \sum_{i=1}^{m} p(x_i) I(x_i) \triangleq H(x) \tag{1.3}$$

where $H(x)$ is the average self-information or self-entropy in any given message (symbol). It is also called the entropy function. The average self-entropy in x_i can also be defined as

$$H(x_i) = p(x_i) I(x_i) \tag{1.4}$$

Finally,

$$H(x) \triangleq -\sum_{i=1}^{m} p(x_i) \log_2 p(x_i) \quad \text{bits/symbol} \tag{1.5}$$

or in briefer notation

$$H(x) = -\sum_{i=1}^{m} p(i) \log p(i) \tag{1.6}$$

Observe for the special case of equally likely events, $p(x_i) = 1/m$,

$$I(x_i) = \log_2 m$$

$$H(x) = \sum_{i=1}^{m} \frac{1}{m} \log_2 m = \log_2 m$$

or

$$H(x) = I(x_i) \tag{1.7}$$

The logarithmic variation satisfies property 3 as follows: The term "unrelated events" means independence between events. For events α and β, the joint probability is

$$p(\alpha \cap \beta) = p(\alpha, \beta) = p(\alpha\beta)$$

(these notations are found in the literature). Then,

$$p(\alpha, \beta) \triangleq p(\alpha|\beta)p(\beta) = p(\alpha)p(\beta) \tag{1.8}$$

where the second equality means $p(\alpha|\beta) = p(\alpha)$, which defines independence between α and β. Hence, if α and β are independent,

$$I(\alpha \cap \beta) = I(\alpha, \beta) = -\log p(\alpha, \beta) = -\log\left[p(\alpha)p(\beta)\right] = -\log p(\alpha) - \log p(\beta) = I(\alpha) + I(\beta)$$

or the information in both events, $I(\alpha,\beta)$, is the sum of the information contained in each.

Notation in this area is varied and one must become accustomed to the various forms. Thus, in the literature either capital P or p is used for probabilities, probability densities, or probability distributions. The meaning is clear in all cases. Recall that in probability theory the words "density" and "distribution" are used interchangeably and one must adjust accordingly. In this document, the notation is as consistent as possible. Observe carefully in the preceding discussion the interplay between self-information, average information over the ensemble, and average information about a specific symbol. Coupling this with several binary digits per symbol and noting that the units for self-information are bits gives a rich mixture for endless confusion. Also, the special case for equally likely events is often used in examples in the literature, and many of this case's results are, of course, not true in general.

ASIDE

The relationship to thermodynamics is as follows: First, recall the evolution of the entropy concept. The change in entropy in a system moving between two different equilibrium states is

$$S_2 - S_1 = \int_1^2 \frac{đQ}{T} \quad \text{reversible} \tag{1.9}$$

where $S_2 - S_1$ is the entropy change, $đQ$ is the change in heat (positive if moving into the system), and T is the temperature at which it is exchanged with the surroundings. The slash through the symbol for the change in Q alerts the reader that heat (Q) is not a perfect differential. The constraint "reversible" means that the change

from state 1 to state 2 occurs over a sequence (path) of intermediate equilibrium states. A "reversible path" means no turbulence, etc., in the gas. In general,

$$S_2 - S_1 \geq \int_1^2 \frac{dQ}{T} \tag{1.10}$$

and the equality only occurs for reversible (physically impossible, ideal situations) changes. Later, another definition arose from statistical thermodynamics, that is,

$$S = k \ln W \tag{1.11}$$

which is apparently an absolute measure (not just a change). Here, k is Boltzmann's constant and W is the "thermodynamic probability" of the state of interest. Unlike normal probabilities, W is always greater than 1. It represents the number of microscopic different arrangements of molecules that yield the same macroscopic (measurable quantities are identical) state. The calculation of W starts from first principles. From the theory, the equilibrium state of a system has the largest W and hence the maximum entropy. Another concept from statistical thermodynamics is the distribution function of a system f. It is defined by

$$dN = f\left(x, y, z, v_x, v_y, v_z, t\right)dx\, dy\, dz\, dv_x\, dv_y\, dv_z = f(\bar{r}, \bar{v}, t)d\bar{r}\, d\bar{v}$$

which means the number of particles at point (x,y,z) with velocity components (v_x, v_y, v_z) at time t. Note that f is a particle density function.

$$f = \frac{\text{number of particles} = dN}{\text{vol (real space) vol (velocity space)}} = \frac{dN}{d\bar{r}d\bar{v}} \tag{1.12}$$

Then, Boltzmann's H theorem states that

$$H \triangleq \int\int f \ln f\, d\bar{r}\, d\bar{v} \tag{1.13}$$

and he showed that

$$H = -(\text{constant})S_{\text{classical}}$$

where $S_{\text{classical}}$ is the classical entropy of thermodynamics. Basically, this says that the measured entropy is the average over the distribution functions available to the system. The reason for the log variation in equation (1.11) is as follows: Assume that the entropy of a system in a given state is some function g of the thermodynamic probability of being in that state, that is,

$$S_A = kg\left(W_A\right)$$

where the subscript A denotes the state of interest and W_A is known by some method. If a similar system is in state B,

$$S_B = kg\left(W_B\right)$$

From experiments, it was known that if the systems were mixed (combined), the resulting entropy S_{AB} was

$$S_{AB} = S_A + S_B$$

Therefore,

$$S_{AB} = kg(W_{AB})$$

But if W_{AB} is the number of arrangements of the combined system, then from counting rules

$$W_{AB} = W_A W_B$$

A little reflection shows that a possible choice for g is log:

$$S_{AB} = k\ln(W_{AB}) = k\ln(W_A W_B) = k\ln(W_A) + k\ln(W_B) = S_A + S_B$$

From this, the logarithmic variation was born. (End of aside.)

Observe that equations (1.5) and (1.13) are similar in form; Shannon (1948) mentions this in his paper. For this reason, he chose the symbol H and the name entropy for the average information. The link with thermodynamics can be established as follows: Consider a container of gas with all molecules in one corner. Because in this condition the "uncertainty" in the position of any molecule is small, let W_1 represent the thermodynamic probability of this condition ($W_1 > 1$, by definition). For this particular case $W_1 = 1$, since only one microscopic arrangement makes up this state. Recall that the gas molecules are dimensionless points, so that permutations at a specific point are not possible. Because the probability that all molecules are in one corner is very small, this is a rare event and has very low $S_{\text{classical}}$. When in equilibrium any single molecule can be anywhere in the container and the uncertainty in its position is large, the thermodynamic probability is $W_2 > W_1$. Thus, the entropy (in a thermodynamic sense) has increased. When in equilibrium any single molecule can be anywhere in the container and the uncertainty in its position is larger than in the previous case, the $S_{\text{classical}}$ is much larger and the entropy (in a thermodynamic sense) has again increased. With information, low probability of occurrence gives large self-information; the probability here is always less than 1. In other words, W and normal probability are reciprocally related, so that uncertainty is the common thread. Thus, average information, not information, and classical thermodynamic entropy vary similarly (where uncertainty is the common thread). This similarity occurs only because of the intuitive constraints imposed on I and H at the beginning of this chapter. Mathematically, both S and H are defined by density functions f and p, respectively;

$$S_{\text{classical}} = -(\text{constant})\int\int f \ln f \, d\bar{r}d\bar{v}$$

$$H(x) = -\sum_{i=0}^{m} p(x_i)\log p(x_i)$$

As a final remark, note that thermodynamic entropy increases and decreases as does f, which varies with the number of states available to the system. As the boundary conditions (pressure, volume, temperature, etc.) change, so does the number of available states. After the number of states has been determined, one must also find the distribution of particles among them. With f now found, $S_{\text{classical}}$ is found by its formula. Therefore, entropy, as we all know, is not an intuitive concept.

EXAMPLE 1.3

Consider a source that produces symbols consisting of eight pulses. Treat each symbol as a separate message. Each pulse can have one of four possible amplitudes, and each message occurs with the same frequency. Calculate the information contained in any single message.

$$\text{number of messages} = 4^8$$

The self-information is

$$I(x_i) = \log_2 (4^8) = 16 \text{ bits/message}$$

The entropy in any message is

$$H(x) = \log_2 (4^8) = 16 \text{ bits/message}$$

Here, $I(x_i)$ and $H(x)$ are equal, since all messages are equally likely.

▲

Now, I introduce some alternative units for information. If the base of the log is 2, the unit is bits. If the base is 10, the unit is hartleys. For natural logs (ln), the unit is nats or nits.

EXAMPLE 1.4

Consider the English language to consist of 27 symbols (26 letters and 1 space). If each occurs at the same frequency,

$$H = \sum_{i=1}^{N} \frac{1}{27} \log_2 (27) = 4.76 \text{ bits/symbol}$$

The actual frequency of occurrence yields $H = 4.065$ bits/symbol.

▲

A key property of $H(x)$ is

$$H(x) \text{ is a maximum when } p(x_1) = p(x_2) = \ldots = p(x_i)$$

That is, all symbols occur with the same frequency,

$$H(x)\big|_{\max} = \log_2 N$$

where N is the total number of equally likely messages.

EXAMPLE 1.5

Show that $H(x)$ is a maximum when all $p(x_i)$ are equal.

$$H = -\sum_{i=1}^{N} p_i \log p_i = -\left(p_1 \log p_1 + p_2 \log p_2 + \ldots + p_N \log p_N\right)$$

Observe that for any term

$$d(p \log p) = \left(p \frac{1}{p} + \log p\right) dp = (1 + \log p) dp$$

Then,

$$dH = -\left[dp_1(1+\log p_1) + dp_2(1+\log p_2) + \ldots + dp_N(1+\log p_N)\right]$$

Because

$$p_1 + p_2 + \ldots + p_N = 1$$

we have

$$dp_1 + dp_2 + \ldots + dp_N = 0 \tag{a}$$

By using equation (a), dp_N can be eliminated

$$\begin{aligned} dH &= -\left[dp_1 \log p_1 + dp_2 \log p_2 + \ldots + dp_N \log p_N\right] \\ &= -\left[dp_1 \log p_1 + dp_2 \log p_2 + \ldots + (-dp_1 - dp_2 - \ldots - dp_{N-1}) \log p_N\right] \end{aligned}$$

Combining terms gives

$$-dH = \left[dp_1 \log\left(\frac{p_1}{p_N}\right) + dp_2 \log\left(\frac{p_2}{p_N}\right) + \ldots + dp_{N-1} \log\left(\frac{p_{N-1}}{p_N}\right)\right] \tag{b}$$

Observe in equation (b) that dp_1, dp_2, ..., dp_{N-1} are now completely arbitrary, since the constraint in equation (a) has essentially been removed. In other words, dp_N has been removed in equation (b). Inspection shows that H is concave down $\cap$, so that at the maximum $dH = 0$ and equation (b) gives

$$\log\frac{p_1}{p_N} = \log\frac{p_2}{p_N} = \ldots = \log\frac{p_{N-1}}{p_N} = 0$$

because the dp_1, dp_2, ..., dp_{N-1} values are now arbitrary. Then,

$$\frac{p_1}{p_N} = \frac{p_2}{p_N} = \ldots = \frac{p_{N-1}}{p_N} = 1 \tag{c}$$

or

$$p_1 = p_2 = \ldots = p_{N-1} \triangleq p$$

Note that because

$$p_N = 1 - \sum_{i=1}^{N-1} p = 1 - (N-1)p$$

any term in equation (c) is

$$\frac{p}{1-(N-1)p}=1$$

or rearrange to find

$$p=\frac{1}{N} \tag{d}$$

▲

This chapter defined the term "message" and introduced the intuitive constraints applied to the measure of information. Then, it showed the utility of the log of the number of permutations, and covered the blending of pulse, binary digit, and symbol used in information theory. Bit and baud were discussed, the term "self-information" was introduced, and the term "average information" (or entropy) was defined. After alluding to notational variations, the chapter discussed the links between information theory and classical thermodynamic entropy. The last example showed $H(x)$ to be a maximum for equally likely outcomes.

Chapter 2
Channel Transfer

This chapter considers a discrete memoryless source (DMS) transmitting symbols (groups of binary digits) over a memoryless channel (fig. 2.1). The source emits symbols $\underline{x}$ that are impressed onto the transmitted waveform u, which then traverses the channel medium. The received waveform v is then demodulated, and the received sequence is denoted by $\underline{y}$. How closely $\underline{y}$ matches $\underline{x}$ yields a measure of the fidelity of the channel. The word "channel" is loosely defined, in that it may include portions of modulators, demodulators, decoders, etc. In general, it means some portion between the source and sink of the communicating parties. The fidelity of the channel is represented as either a channel transition probability matrix or a channel transition diagram (fig. 2.2). In this figure, the term $p(y_i|x_i)$ means the conditional probability that y_i is received in the ith time slot, given that x_i was transmitted in that slot (with the delay in the system appropriately considered). In principle, these entries are determined by measurement on a given channel. Because of the property of probabilities for exhaustive events, the sum over any row must be unity. It follows that a particular output, say y_n, is obtained with probability

$$p(y_n) = \sum_{m=1}^{M} p(y_n|x_m)p(x_m) \tag{2.1}$$

where $p(x_m)$ is the probability that x_m was input to the channel. The entropy of the channel output is

$$H(y) = -\sum_{n=1}^{N} p(y_n)\log_2 p(y_n) \qquad \text{bits/symbol} \tag{2.2}$$

and the entropy of the output, given that a particular input, say x_m, was present, is

$$H(y|x_m) = -\sum_{n=1}^{N} p(y_n|x_m)\log_2 p(y_n|x_m)$$

When averaged over all possible inputs, the conditional entropy of the output given the input $H(y|x)$ is

$$H(y|x) = -\sum_{m=1}^{M}\sum_{n=1}^{N} p(x_m, y_n)\log_2 p(y_n|x_m) \qquad \text{bits/symbol} \tag{2.3}$$

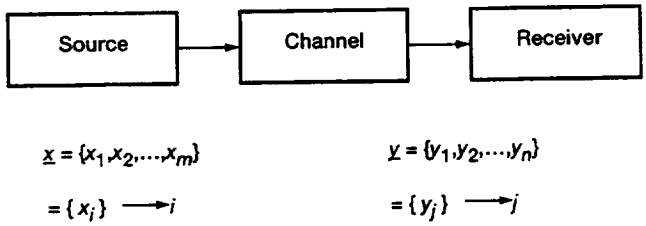


Figure 2.1.—Basic communications channel. (The source emits symbols x_i (with shortened notation i), and at the receiver the symbol y_j appears. The difference between x_i and y_j is the corruption added by the channel.)

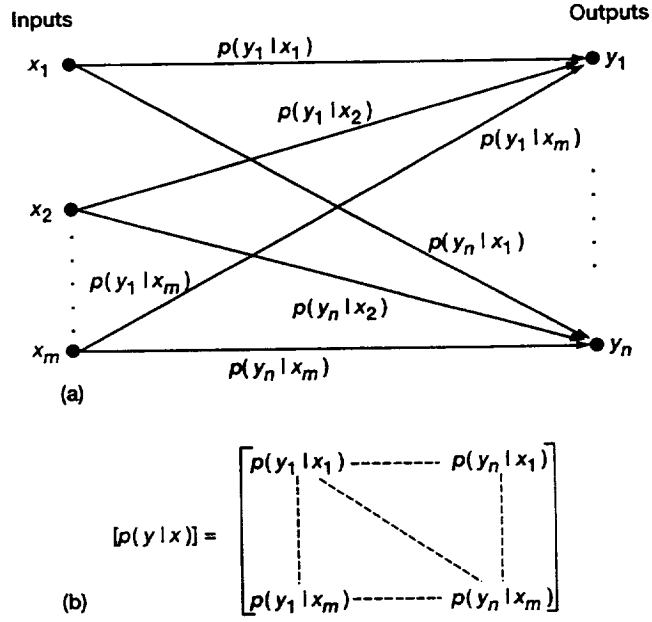


Figure 2.2.—Channel transition diagram (a) and alternative representation of channel transition matrix (b).

where the relationship

$$p(x_m, y_n) = p(y_n|x_m)p(x_m)$$

has been used for the probability that the joint event that the input was x_m and the output was y_n has occurred. In a similar fashion, the conditional entropy $H(x|y)$ can be defined by replacing $p(y_n|x_m)$ by $p(x_m|y_n)$ in equation (2.3).

Recall that entropy is the average amount of information; therefore, $H(x|y)$ is the average information about x (channel input) given the observation of y at the receiver. This knowledge is arrived at after averaging over all possible inputs and outputs. Because $H(x)$ is the entropy for the input symbol with no side information (not knowing the channel output), it follows that the average information transferred through the channel is

$$I(x;y) = H(x) - H(x|y) \quad \text{bits/symbol} \tag{2.4}$$

where $I(x;y)$ is defined as the mutual information. By Bayes' theorem

$$I(x;y) = H(y) - H(y|x) \quad \text{bits/symbol} \tag{2.5}$$

In either case, $I(x;y)$ can be written as

$$I(x;y) = \sum_{m=1}^{M} \sum_{n=1}^{N} p(x_m, y_n) \log_2 \frac{p(x_m, y_n)}{p(x_m)p(y_n)} \quad \text{bits/symbol} \tag{2.6}$$

where $p(x_m, y_n) = p(y_n|x_m)p(x_m) = p(x_m|y_n)p(y_n)$ are the joint probabilities of the event that the channel input is x_m and its output is y_n.

By the theorem of total probability, the mutual information can be expressed as a function of the channel input probabilities $p(x_m)$ and the channel transition probabilities $p(y_n|x_m)$. For a specified channel, the transition terms are fixed and are presumed to be determined by experiment. With mutual information defined, the maximum, which Shannon (1948) defined as the capacity of a channel, is defined as

$$C = \max_{p(x_m)} I(x;y)$$

The channel capacity C is the maximum amount of information that can be conveyed through the channel without error if the source is matched to the channel in the sense that its output symbols occur with the proper probabilities such that the maximum mutual information is achieved. The $p(x_m)$ under the "max" in the preceding equation means that the source is appropriately adjusted to achieve the maximum. The alteration of the probabilities of the source's output symbols $p(x_m)$ to maximize the probability of successful (error free) transmission is assumed to occur by appropriate coding of the raw source output symbols. The early thrust in coding theory was to search for such optimum codes.

Although developed for a discrete channel (finite number of inputs and outputs), $I(x;y)$ can be generalized to channels where the inputs and outputs take on a continuum of values (the extreme of "soft" modulators and demodulators).

An alternative approach to redeveloping equations (2.1) to (2.6) is to start with the reasonable definition of joint entropy $H(x,y)$ (let $N = M = n$ for simplicity):

$$H(x,y) \triangleq -\sum_{i=1}^{n} \sum_{j=1}^{n} p(x_i, y_j) \log p(x_i, y_j) = -\sum_{i} \sum_{j} p(i,j) \log p(i,j)$$

If x_i and y_j are independent

$$p(x_i, y_j) = p(x_i)p(y_j) = p(i)p(j)$$

Then,

$$\begin{aligned} H(x,y) &= -\sum_{i} \sum_{j} p(i)p(j) \log[p(i)p(j)] \\ &= -\sum_{i} p(i) \log p(i) \sum_{j} p(j) - \sum_{j} p(j) \log p(j) \sum_{i} p(i) = H(x) + H(y) \end{aligned}$$

If there is some dependence

$$p(i,j) = p(i|j)p(j) = p(j|i)p(i)$$

then,

$$H(x, y) = -\sum_i p(i) \log p(i) \sum_j p(j|i) = -\sum_i \sum_j p(i) p(j|i) \log p(j|i) = H(x) + H(y|x)$$

where

$$H(y|x) \triangleq -\sum_i \sum_j p(i, j) \log p(j|i)$$

is the conditional entropy. It is also called the equivocation of x about y or the equivocation of y given x. It can be shown that

$$H(x, y) = H(x) + H(y|x) = H(y) + H(x|y)$$

Then, the mutual information is defined by

$$I(x;y) \triangleq H(x) - H(x|y) = H(y) - H(y|x)$$
$$= \sum_i \sum_j p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(x_i) p(y_j)} = I(y;x)$$

In my opinion, the key to enabling the subtraction of the equivocation from the self-entropy is just the additive property of entropy by its basic definition. Many variations on this theme are found in the literature; mutual information is sometimes called delivered entropy. Also, there are more axiomatic and perhaps more mathematically rigorous presentations, but I think that the above essentially covers the basic idea. The Venn type of diagram shown in figure 2.3 is sometimes used, and it can be helpful when following certain presentations.

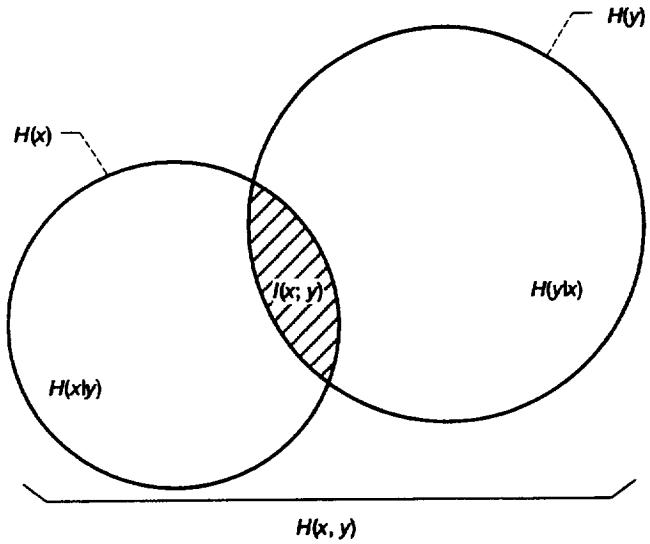


Figure 2.3.—Venn diagram for various entropy terms and their relationship with mutual term $I(x; y)$.

For computational purposes, the relationships between logs are

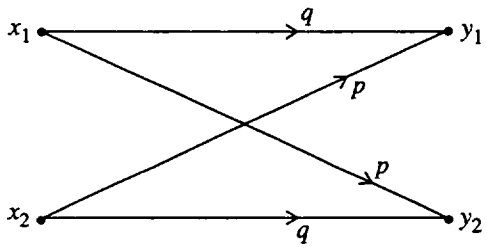
$$\log_2 x = \frac{\log_{10} x}{\log_{10} 2} = 3.321928 \log_{10} x = 1.442695 \ln x$$

EXAMPLE 2.1

The classic binary symmetric channel (BSC) with raw error (or crossover) probability p serves as an easy demonstration of the procedures discussed above. The two symbols x_1 and x_2 have frequencies of occurrence such that

$$p(x_1) = \alpha$$
$$p(x_2) = 1 - \alpha = \beta$$

the channel diagram is



$q = 1 - p$

and the channel transition matrix is

$$P = \begin{bmatrix} q & p \\ p & q \end{bmatrix} = \begin{bmatrix} p(y_1|x_1) & p(y_2|x_1) \\ p(y_1|x_2) & p(y_2|x_2) \end{bmatrix}$$

The final objective is to determine the capacity, and the sequence of steps to find it are as follows: First, the entropy of the source symbols x_1, x_2 is

$$H(x) = -\alpha \log \alpha - (1-\alpha) \log (1-\alpha)$$

Then, from the definition of conditional entropy,

$$H(y|x) = -\sum_{i=1}^{m} \sum_{j=1}^{n} p(x_i, y_j) \log \left[p(y_j|x_i) \right]$$

and using $p(x_i, y_j) = p(y_j|x_i) p(x_i)$,

$$H(y|x) = -\sum_{i=1}^{2} \sum_{j=1}^{2} p(x_i) p(y_j|x_i) \log p(y_j|x_i)$$

$$= -\sum_{i=1}^{2} p(x_i) p(y_1|x_i) \log p(y_1|x_i) + p(x_i) p(y_2|x_i) \log p(y_2|x_i)$$

$$
\begin{aligned}
&= -\Big\{p(x_1)p(y_1|x_1)\log p(y_1|x_1) + p(x_1)p(y_2|x_1)\log p(y_2|x_1) \\
&\quad + p(x_2)p(y_1|x_2)\log p(y_1|x_2) + p(x_2)p(y_2|x_2)\log p(y_2|x_2)\Big\} \\
&= -\{\alpha q \log q + \alpha p \log p + \beta p \log p + \beta q \log q\} \\
&= -\Big\{(\alpha+\beta)q \log q + (\alpha+\beta)p \log p\Big\} \\
&= -p \log p - q \log q \\
&\triangleq H_2(p)
\end{aligned}
$$

Next, find $H(y)$:

$$H(y) = -\sum_{j=1}^{2} p(y_j)\log p(y_j)$$

Now,

$$
\begin{aligned}
p(y_1) &= p(y_1|x_1)p(x_1) + p(y_1|x_2)p(x_2) = q\alpha + p\beta \\
p(y_2) &= p(y_2|x_1)p(x_1) + p(y_2|x_2)p(x_2) = p\alpha + q\beta
\end{aligned}
$$

Then,

$$H(y) = -(q\alpha + p\beta)\log(q\alpha + p\beta) - (p\alpha + q\beta)\log(p\alpha + q\beta)$$

Then, the mutual information is

$$I(x;y) = H(y) - H(y|x) = H_2(q\alpha + p\beta) - H_2(p)$$

where $H_2(u)$ is the entropy function for a binary source,

$$H_2(u) \triangleq -u \log u - (1-u)\log(1-u)$$

Figure 2.4, a sketch of $H_2(u)$, shows that $H_2(0.5) = H_{2_{\max}}$ has a maximum of unity; thus, the channel capacity is

$$C = 1 - H_2(p) = 1 + p \log p + (1-p)\ \log(1-p) \qquad \text{bits/symbol}$$

where $\alpha = \beta = 1/2$ by observation of the plot. Then finally,

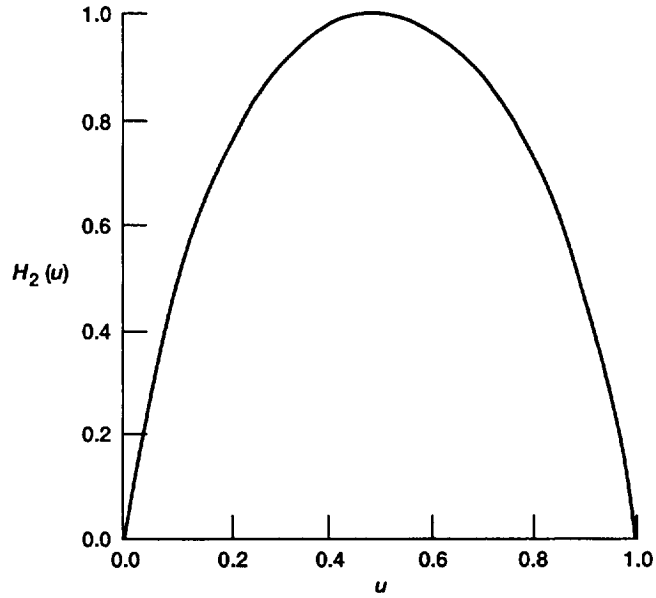


Figure 2.4.—Entropy function for binary source.

$$C \triangleq C_{\mathrm{BSC}}$$

$$C_{\mathrm{BSC}} = 1 + p\log p + (1-p)\log(1-p) \qquad \text{bits/symbol}$$

▲

Figure 2.5 gives channel capacity C versus the crossover probability p. The capacity can be given in various units; namely bits per symbol, bits per binit (binit means binary digit), or bits per second. For example, if $p = 0.3$, then $C = 0.278$ bit/symbol, which is the maximum entropy each symbol can carry. If the channel were perfect, each symbol could carry the self-entropy $H(x)$, which is calculated by the size of the source's alphabet and the probability of each symbol occurring. The 30-percent chance of error induced by the channel can be corrected by some suitable code, but the redundancy of the code forces each symbol to carry only 0.278 bit. Another interpretation of C follows by assuming that each transmitted symbol carries 1 bit. Then, C is the remaining information per symbol at the receiver. When $p = 0.3$, each received symbol carries only 0.278 bit. This rather drastic loss of information (72.2 percent) for $p = 0.3$ occurs because, although only 30 percent are in error, the receiver has no clue as to which ones. Thus, the code to tell the receiver which symbols are in error takes up a large amount of overhead. In the original development of the theory, the symbols, which are composed of binary digits, were assumed to be mapped by the modulator into some specific analog waveform to be transmitted over the channel. If the received waveform were demodulated in error, the number of actual binary digits in error could not be determined. Thus, errors are basically message errors, and the conversion from message error to binary digit error is always vague.

Finally, consider the case for a continuous source (one that emits analog waveforms). The definition for the entropy is as before, with the summation going to the integral:

$$H = -\int_{-\infty}^{\infty} p(x)\log[p(x)]\,dx$$

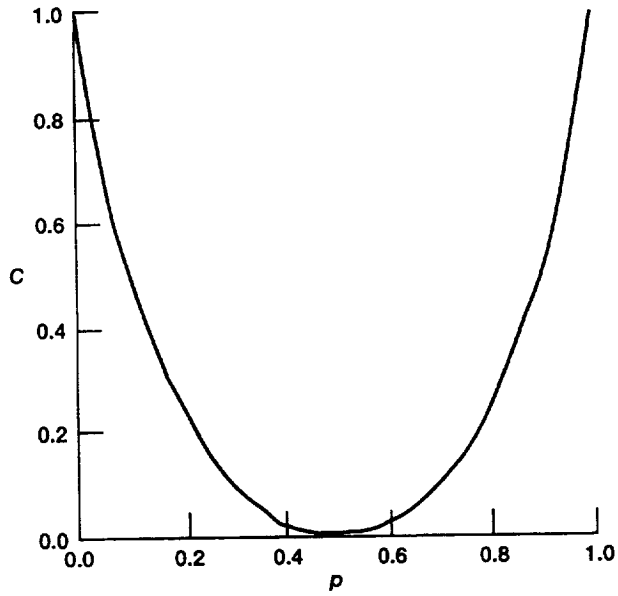


Figure 2.5.—Channel capacity C versus crossover probability p for binary symmetric channel.

The problem is to determine the form for $p(x)$ that maximizes the entropy, under the constraint that the average power is fixed (i.e., the variance is fixed for voltage waveforms). This constraint is

$$\int_{-\infty}^{\infty} x^2 p(x)\,dx = \sigma^2$$

The basic constraint for probability densities is

$$\int_{-\infty}^{\infty} p(x)\,dx = 1$$

which forces $p(x)$ to be Gaussian:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-x^2/2\sigma^2}$$

where σ^2 is the average signal power. Evaluating the integral for H gives the maximum entropy for an analog source,

$$H(x) = \log\left(\sigma\sqrt{2\pi e}\right)$$

The classic formula for channel capacity is arrived at by considering a theoretical code with signal power S. The entropy of the code is

$$H_S = \frac{1}{2}\log(2\pi eS)$$

If the channel is the classical additive white Gaussian noise (AWGN), no fading or intersymbol interference (ISI) allowed, the noise entropy is

$$H_N = \frac{1}{2}\log(2\pi eN)$$

where N is the average noise power. The capacity is thus

$$C \triangleq H_{S+N} - H_N = \frac{1}{2}\log\left(1+\frac{S}{N}\right)$$

Now, the maximum information rate $R_{\max}$ is

$$R_{\max} = \frac{C}{T}$$

where

$$T = \frac{1}{2W}$$

(W = bandwidth) is the Nyquist rate for no ISI. Then,

$$R_{\max} = \frac{\frac{1}{2}\log\left(1+\frac{S}{N}\right)}{\frac{1}{2W}} = W\ \log\left(1+\frac{S}{N}\right)$$

where $N = N_o W$. Here, the noise power spectral density N_o is in watts per hertz (single sided), and again the constant average signal power is S. If

$$\frac{S}{N} = 2R\frac{E_b}{N_o}$$

where $R = k/n$ is the code rate, then

$$C = \frac{1}{2}\log_2\left(1+2R\frac{E_b}{N_o}\right) \qquad \text{bits/sec}$$

Here, k is the number of actual information symbols emitted by the source, and the encoder takes them and outputs n total symbols (adds the appropriate coding baggage).

This classic capacity formula differed from the general opinions of the day in the following ways: Apparently, it was thought that the noise level of the channel limited the maximum information that could be transmitted. Shannon's (1948) formula shows that the rate of information transfer is actually bounded and is related to the basic parameters of signal power S, bandwidth W, and noise power spectral density N_o. Another measure of the upper limit for information transmission is called the cutoff rate. It is less than C and arises as follows.

The capacity as defined earlier is the absolute upper limit for error-free transmission. However, the length of the code and the time to decode the symbols to extract the desired message may be prohibitively long. The cutoff rate serves as more of an implementation limit for practical decoders. It turns out that the computations required to decode one information bit for a sequential decoder has asymptotically a Pareto distribution, that is,

$$P(\text{comp} \geq N) < \beta N^{-\alpha} \qquad N >> 1$$

where "comp" means the number of computations and N is some large chosen number. The coefficient α is the Pareto exponent, and it along with β (another constant) depend on the channel transition probabilities and the code rate R. This relationship was found by Gallager (1968) and verified through simulation. The code rate and the exponent are related by

$$R = \frac{E_0(\alpha)}{\alpha}$$

where

$$E_0(\alpha) = \alpha - \log_2 \left\{ \left[(1-p)^{\frac{1}{1+\alpha}} + p^{\frac{1}{1+\alpha}} \right]^{1+\alpha} \right\}$$

is the Gallager function for the BSC. The solution when $\alpha = 1$ yields $R \triangleq R_0$, the computational cutoff rate. In general, systems use $1 < \alpha < 2$. The value R_0 sets the upper limit on the code rate. For the binary input/continuous output (very soft) case,

$$R_0 = 1 - \log_2 \left(1 + e^{RE_b / N_o} \right)$$

and for the discrete memoryless channel/binary symmetric channel case (DMC/BSC)

$$R_0 = 1 - \log_2 \left[1 + 2\sqrt{p(1-p)} \right]$$

Then, the probability of a bit error is

$$p_{\text{bit}} \leq \frac{2^{-KR_0/R}}{\left\{ 1 - 2^{-[R_0/R - 1]} \right\}^2} \qquad R < R_0$$

where K is the constraint length of the encoder. The terms "sequential code" and "constraint length" will be defined in chapters 6 and 7.

Other variations on cutoff rate can be found. However, they often are involved with channel coding theorems, etc., and most likely the discussion deals with upper bounds on message error rates. Two such expressions are

$$P(\text{message error}) < C_R \, 2^{-nR_0} \qquad R < R_0$$

$$P(\text{message error}) < C_R \, 2^{-KR_0} \qquad R < R_0$$

The leading coefficients C_R are determined experimentally and depend on the channel and the code rate. The exponent n is the block length for a block code, whereas K is the constraint length for a convolutional code.

EXAMPLE 2.2

What are the self-information and average information values in a coin-flipping experiment? Here, the symbols are heads and tails. Then, self-information is

$$I(\text{head}) = -\log_2\left(\frac{1}{2}\right) = 1 \text{ bit}$$

and the average information is

$$H(\text{symbol}) = \underbrace{\frac{1}{2}\log_2\left(\frac{1}{2}\right)}_{\text{head}} \underbrace{-\log_2\left(\frac{1}{2}\right)}_{\text{tail}} \begin{cases} = 1 \text{ bit / flip} \\ = 1 \text{ bit / symbol} \end{cases}$$

so the entropy is 1 bit/symbol or 1 bit/flip. Other units, such as nits per flip or hartleys per flip, could also be used by changing the base of the log.

▲

EXAMPLE 2.3

This is an interesting exercise on "units." Consider transmitting the base 10 digits 0,1,2,...,9 by using a code consisting of four binary digits. The code table is

0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
.	.
.	.
.	.
15	1 1 1 1

Note that decimals 10 to 15 (corresponding to 1010 to 1111) never appear. The total number of symbols N is 10 for this channel. Now, the self-information per symbol (assuming that all are equally likely) is

$$I(x_i) = \log_2 10 \quad \text{bits}$$

Then, forming the ratio of the number of information bits transmitted per binary digit gives

$$\frac{\text{bits}}{\text{binary digit}} = \frac{\log_2 10}{4} = 0.83 \frac{\text{bit}}{\text{binit}}, \text{ or } 0.83 \frac{\text{bit}}{\text{bit}}$$

Here, binit stands for binary digit. Quite often, binit is shortened to "bit," which gives the rather confusing unit of "bit per bit." Here, each binary digit carries only 0.83 "information bit" (or self-information) because only 10 of the possible 16 sequences are used. The value 0.83 is further reduced by propagation over the channel after being acted upon by the channel transition probabilities.

Similarly, the capacity for the binary symmetric channel can be written as

$$C = 1 + p\log_2 p + (1-p)\log_2(1-p) \quad \text{bits / binit or bits / symbol}$$

where the latter units are information bits per symbol. The capacity can also be given as a rate as was done for the AWGN channel:

$$C = W \log_2\left(1+\frac{S}{N}\right) \quad \text{bits/sec}$$

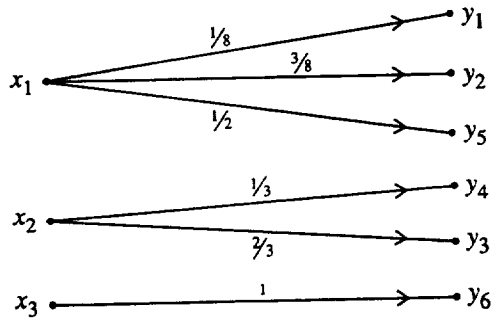
Thus, one must be aware of the possible confusion about what "bit" means. If one is just talking about the baud of a channel (the number of symbols transmitted per second), information content is not considered. The term "bits per second" is then a measure of system speed, and information is not the issue. The bits in this case are just binary symbols that the modem can handle, and any pseudorandom bit stream can pass and carry absolutely no information. ▲

EXAMPLE 2.4

Approximately eight basic channel models are used in the literature:

1. Lossless
2. Deterministic
3. Ideal
4. Uniform
5. Binary symmetric (BSC)
6. Binary erasure (BEC)
7. General binary (GBC)
8. M-ary symmetric

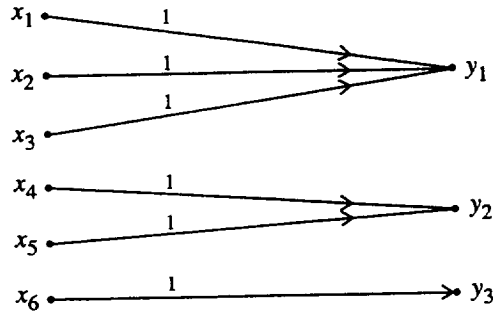
For the lossless channel the probability matrix contains only one nonzero element in each column:



$$p(y|x) = \begin{bmatrix} \frac{1}{8} & \frac{3}{8} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{2}{3} & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, $C = \log Q$, where Q is the number of source symbols.

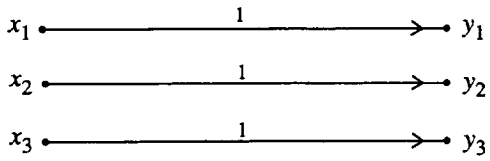
The deterministic channel has only one nonzero element in each row



$$p(y|x) = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, $C = \log Z$, where Z is the number of output symbols.

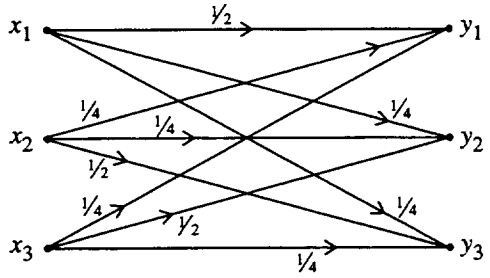
The ideal channel is



$$p(y|x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here, $C = \log Q = \log Z$, where Q and Z are the number of source and output symbols, respectively.

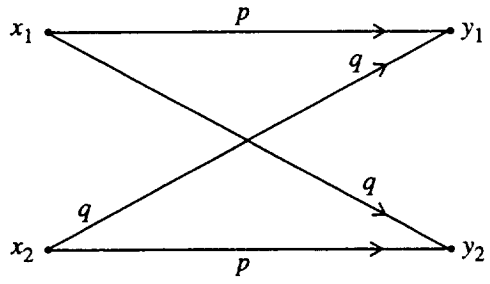
In the uniform channel model, every row and column is an arbitrary permutation of the probabilities in the first row:



$$p(y|x) = \begin{bmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Here, $C = \log Q + \sum_{n=1}^{Q} p(y_n|x_m) \log p(y_n|x_m)$. Here, Q is the number of input symbols.

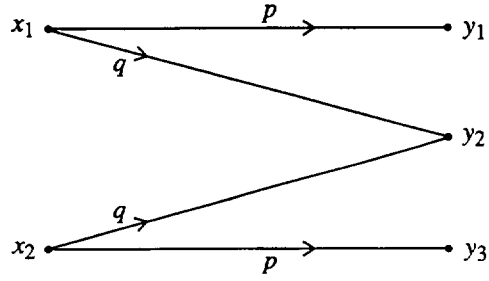
The BSC model uses the formula for the uniform channel model:



$$p(y|x) = \begin{bmatrix} p & q \\ q & p \end{bmatrix}$$

Here, $C = \log_2 2 + p \log p + q \log q$.

For the BEC model, the middle output y_2 is the erasure. An erasure is a demodulator output that informs the user that the demodulator could not guess at the symbol.

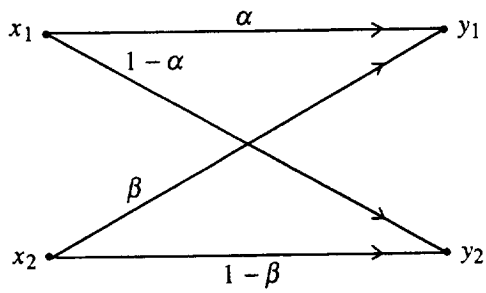


Erasure

$$p(y|x) = \begin{bmatrix} p & q & 0 \\ 0 & q & p \end{bmatrix}$$

Here, $C = p$.

For the GBC model



$$p(y|x) = \begin{bmatrix} \alpha & 1-\alpha \\ \beta & 1-\beta \end{bmatrix}$$

Here, $C = \log\left[\sum_{i=1}^{m} 2^{x_i}\right]$. One must find the x_i by solving the following set:

$$p_{11}x_1 + \cdots + p_{1m}x_m = \sum_{i=1}^{m} p_{1j} \log p_{1j}$$

.

.

.

$$p_{m1}x_1 + \cdots + p_{mm}x_m = \sum_{j=1}^{m} p_{mj} \log p_{mj}$$

Solve for $x = x_i$, $i = 1,\ldots,m$. Alternatively,

$$C = \frac{-\beta H_2(\alpha) + \alpha H_2(\beta)}{\beta - \alpha} + \log\left\{1 + 2^{\left[H_2(\alpha) - H_2(\beta)\right]/(\beta-\alpha)}\right\}$$

The M-ary symmetric channel sends not binary symbols but M-ary ones, where M is an integer.

$$P(y|x) = \begin{bmatrix} p & \frac{1-p}{M-1} & \frac{1-p}{M-1} & \cdots & \frac{1-p}{M-1} \\ \frac{1-p}{M-1} & p & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \frac{1-p}{M-1} \\ \frac{1-p}{M-1} & \cdots & \cdots & \frac{1-p}{M-1} & p \end{bmatrix}$$

Here, $C = \log M - (1-p)\log(M-1) - H_2(p)$.

▲

EXAMPLE 2.5

Assume a well-shuffled deck of 52 cards. How much entropy is revealed by drawing a card? Since any card is equally likely,

$$H = \log_2 52 = 5.7 \text{ bits} = \ln 52 = 3.95 \text{ nats} = \log_{10} 52 = 1.716 \text{ hartleys}$$

Chapter 3
Mathematical Preliminaries

3.1 Modulo-2 Arithmetic

Modulo-2 arithmetic is defined in the two tables below.

Addition

$\oplus$	0	1
0	0	1
1	1	0

Multiplication

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

The dot (or inner) product of two sequences is defined as

$$(10110)\cdot(11100) \triangleq 1\cdot 1 \oplus 0\cdot 1 \oplus 1\cdot 1 \oplus 1\cdot 0 \oplus 0\cdot 0 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 = 0$$

A sequence of modulo-2 digits has no numerical significance in most coding discussions. However, a polynomial representation for a string of binary digits is used universally; for example,

$$11010011 \leftrightarrow 1 \oplus x \oplus x^3 \oplus x^6 \oplus x^7$$

Here, a one in the ith position means the term x^i is in the polynomial. Here, i starts at zero on the left, but just the opposite notation is used in many treatments in the general literature. The polynomials (like integers) form a ring and factoring is an important property; for example,

$$\left(x^7 \oplus 1\right) = (x \oplus 1)\left(x^3 \oplus x \oplus 1\right)\left(x^3 \oplus x^2 \oplus 1\right)$$

Because the factoring is not readily apparent, tables of such factorizations are often needed. Multiplication of two sequences is best done by just multiplying the polynomials and then transforming back to binary digits as the following example shows.

Example 3.1
Multiply 101101 by 1101.

$$101101 \leftrightarrow 1 \oplus x^2 \oplus x^3 \oplus x^5, \qquad 1101 \leftrightarrow 1 \oplus x \oplus x^3$$

$$\begin{aligned}\left(1 \oplus x^2 \oplus x^3 \oplus x^5\right)\left(1 \oplus x \oplus x^3\right) &= 1 \oplus x^2 \oplus x^3 \oplus x^5 \oplus x \oplus x^3 \oplus x^4 \oplus x^5 \oplus x^3 \oplus x^5 \oplus x^6 \oplus x^8 \\ &= 1 \oplus x \oplus x^2 \oplus x^3 \oplus x^4 \oplus x^8 \leftrightarrow 111110001\end{aligned}$$

Note that

$$x^3 \oplus x^3 = 0$$
$$x^5 \oplus x^5 = 0$$

etc. ▲

Modulo-2 addition and multiplication are associative and commutative. A product of $n_1 \cdot n_2$ has $(n_1 \oplus n_2 - 1)$ digits.

Modulo-2 division is just like ordinary algebra; for example, $\left(x^3 \oplus 1\right) \div \left(x^2 \oplus x\right)$.

$$\begin{array}{l} \underline{x^2 \oplus x} \;\; x^3 \oplus 1 \quad \underline{\;x \oplus 1} \leftarrow \text{quotient} \\ \quad \underline{x^3 \oplus x^2} \\ \qquad x^2 \oplus 1 \\ \qquad \underline{x^2 \oplus x} \\ \qquad\quad x \oplus 1 \leftarrow \text{remainder} \end{array}$$

Convolution of two sequences is as follows: (1101)*(10011)

Step 1

	1	0	0	1	1
1011					

alignment (here, 1101 is reversed)

Step 2

	1	0	0	1	1
101	1				

$1 \cdot 1 = 1$ (first term in result)

Step 3

	1	0	0	1	1
10	1	1			

$0 \cdot 1 \oplus 1 \cdot 1 = 0 \oplus 1 = 1$

Step 4

	1	0	0	1	1
1	0	1	1		

$0 \cdot 1 \oplus 0 \cdot 1 \oplus 1 \cdot 0 = 0$

Step 5

1	0	0	1	1
1	0	1	1	

$1 \cdot 1 \oplus 0 \cdot 1 \oplus 0 \cdot 0 \oplus 1 \cdot 1 = 1 \oplus 0 \oplus 0 \oplus 1 = 0$

Step 6

1	0	0	1	1
	1	0	1	1

$1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \oplus 0 \cdot 1 = 1 \oplus 1 \oplus 0 \oplus 0 = 0$

Step 7

1	0	0	1	1	
		1	0	1	1

$1 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 1 = 1$

Step 8

1	0	0	1	1	
			1	0	11

$1 \cdot 0 \oplus 1 \cdot 1 = 1$

Step 9

1	0	0	1	1	
				1	011

$1 \cdot 1 = 1$

$$\therefore (1101) * (10011) = (11000111)$$

3.2 Channel Terminology

The terminology is not always clear in that different portions of the communications system may be included in the "channel." A basic block diagram for a block-coded channel is given in figure 3.1. Here, the channel symbols are just n-bit binary sequences (i.e., 1011...). Figure 3.2 represents a system wherein the symbols are strings of binary digits and all strings are of a specified length. This figure then shows the additional boxes that convert from binary digits to strings of such digits.

The "channel" is often just the transmission medium, but "channel symbols" are emitted from the demodulator, so that the channel boundaries are fuzzy. The inputs to the encoder are k-bit messages, source bits, information bits, etc. The encoder emits n-bit code words, code bits, channel bits, bauds, channel symbols, etc. A more detailed model is given in figure 3.3 for convolutional codes.

Next, the message energy is defined to be

$$E_m = \int_0^T S^2(t)\, dt$$

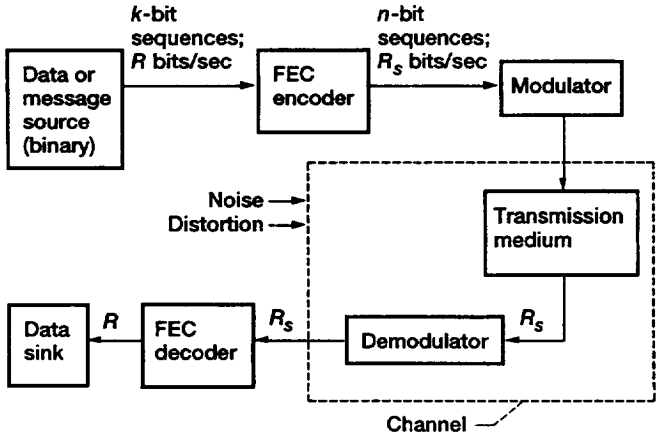


Figure 3.1.—Basic communications system using block coding.

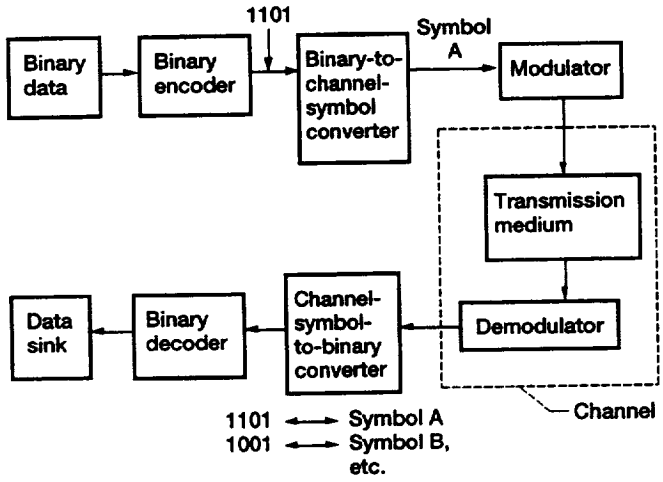


Figure 3.2.—More detailed block diagram for basic block-coded channel.

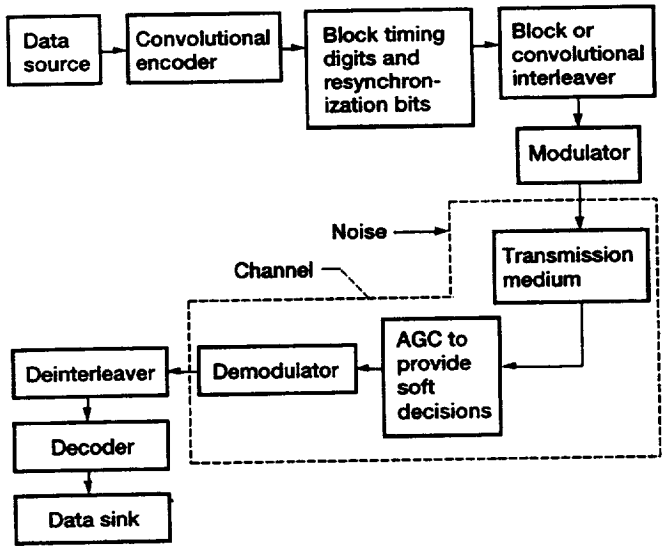


Figure 3.3.—Communications channel using convolutional coding. (The source and sink are separated by the convolutional codec, interleaver/deinterleaver, synchronization units, and modulator/demodulator pair.)

where $S(t)$ is shown in figure 3.4. The received energy in a binit (called a bit) is

$$E_b = \frac{E_m}{k} \qquad \text{energy/data bit or energy/bit}$$

In block coding, the source data are segmented in k-bit blocks and passed to the encoder. The encoder calculates some parity check bits (by modulo-2 addition of some of the k bits) and outputs the original k bits along with the check bits. The number of binits (bits) from the encoder is n, thus, an (n,k) encoder. In most cases, the n bits are constrained to the same time interval as are the original k bits. Thus, the channel bits contain less energy than do the source bits. In other words, the message energy in either a k-bit source sequence or an n-bit channel sequence is the same.

$$\therefore E_m = kE_b = nE_s$$

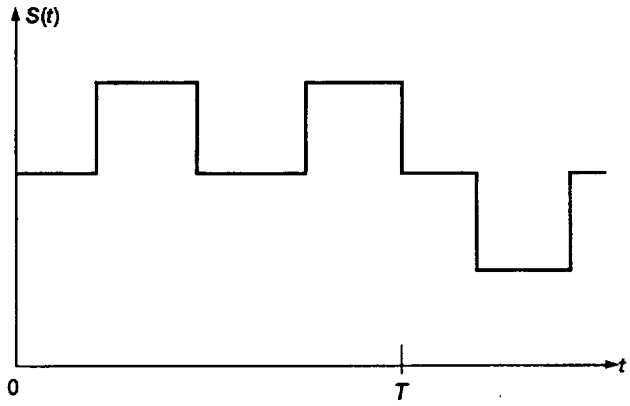


Figure 3.4.—Arbitrary message waveform.

where E_s is the received energy in the channel symbol. The quantities R, r, R_s, n, and k are related by

$$\frac{k}{n} \triangleq r = \frac{R}{R_s} < 1$$

where

r code rate or code efficiency
R data rate or information symbol rate, bits/sec
R_s symbol rate, channel symbol rate, chip rate, baud, etc.

Thus, coding increases the bandwidth as well as the number of errors emitted from the demodulator. The increase in errors is due to the reduced energy per pulse now available to the demodulator. When the coding is turned off, the demodulator makes decisions on energy values E_b, whereas with coding the decisions are made on E_s and $E_s < E_b$, where

$$E_s = \frac{k}{n} E_b = r E_b$$

The correction capability of the code overcomes the extra demodulator errors. At the receiver, let

$$P = \text{received power in modulated signal} = E_s R_s$$

Then the signal-to-noise ratio (SNR) is

$$\frac{P}{N_o} = \frac{E_s R_s}{N_o} = \frac{E_b R}{N_o}$$

or

$$\frac{E_b}{N_o} = \frac{P}{N_o \mathrm{R}}$$

From a coding point of view, the system appears as shown in figure 3.5. The message sequence $\underline{m}$ enters the encoder and is mapped into the code vector sequence $\underline{u}$. After propagation through the channel, the decoder acts on the sequence $\underline{z}$ and outputs $\hat{\underline{m}}$; and one assumes that $\hat{\underline{m}} = \underline{m}$ with high probability. Systematic encoders, which produce code words of the form indicated in figure 3.5, are considered in most cases.

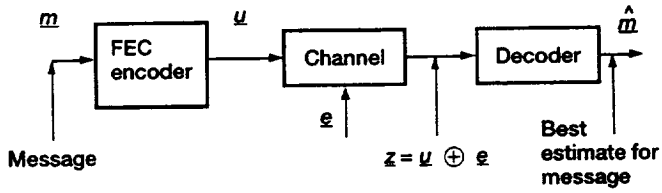


Figure 3.5.—Basic block coder/decoder system. (The code vector $\underline{u}$ is corrupted by the channel's noise vector $\underline{e}$. The decoder attempts to remove $\underline{e}$ and recover the message.)

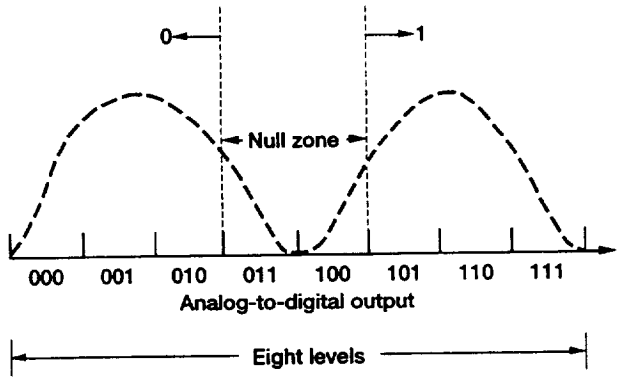


Figure 3.6.—Soft-decision decoder, here quantized to standard three bits or eight levels. (The null zone is used by the demodulator to alert the decoder that the particular bit is completely uncertain or has been "erased.")

Systematic means that the original message bits are preserved (kept in order) and the parity bits are appended to the end of the string.

Figure 3.5 summarizes the basic steps used in error correction. The message vector, say $\underline{m} = 1011$, is mapped into the code vector $\underline{u} = 1011001$ by the encoder circuit. The received vector $\underline{z}$ is the modulo-2 addition of the transmitted sequence $\underline{u}$ and the error vector $\underline{e}$ added by the channel. The task of the decoder may be listed as follows:

1. Is $\underline{e} = 0$?
2. If $\underline{e} \neq \underline{0}$, determine $\underline{e}$.
3. Develop, or reconstruct, $\hat{\underline{e}}$ by some decoding algorithm. Hope that $\hat{\underline{e}} = \underline{e}$.
4. Remove the effect of the corruption due to $\hat{\underline{e}}$ by just adding it to the received vector $\underline{z}$; $\underline{z} + \hat{\underline{e}} = \underline{u} + \underline{e} + \hat{\underline{e}}$.
5. If $\hat{\underline{e}} = \underline{e}$, the decoding is successful and the error is corrected.

Obviously, step 3 is the key one in the procedure. How to perform it is essentially the basic task of decoding techniques.

When the demodulator outputs binary (hard) decisions, it gives the decoder the minimal amount of information available to decide which bits might be in error. On the other hand, a soft decision gives the decoder information as to the confidence the demodulator has in the bit. In other words, a hard-decision demodulator outputs just two voltage levels corresponding to one or zero. A soft-decision demodulator on the other hand, generally outputs three-bit words that give the location of the best estimate of the signal (fig. 3.6). In other words, the output 000 corresponds to a strong zero, whereas 011 corresponds to the weakest zero. Similarly, 111 corresponds to a strong chance that a one was transmitted. Another demodulator output is the null zone, or erasure output. When the signal is about equidistant from either a one or a zero, the demodulator sends a special character to alert the decoder that the bit's value is essentially uncertain.

Chapter 4
Block Codes

This chapter covers the basic concepts of block codes; chapter 5, a "second pass," adds much of the detail needed for in-depth understanding.

The forward-error-correcting (FEC) encoder accepts k information bits and outputs n bits. The $n - k$ added bits are formed by modulo-2 sums of a particular set of the k input bits. The output blocks of n bits are the code words. For n-tuples consisting of binary digits, there are 2^n distinct n-tuples. Of these, only 2^k are chosen as permissible code words. Let $\underline{u}_i$ and $\underline{u}_j$ be code vectors. The code is linear if $\underline{u}_i \oplus \underline{u}_j$ is also a code word. A linear block code is a set of 2^k n-tuples (a vector subspace; i.e., a subset of the possible 2^n n-tuples). Figure 4.1 illustrates the concept of selecting code words from the entire vector space. The large dots represent the code words, and the small dots represent possible received vectors, which are code words corrupted by the channel (i.e., noise vectors added to code vectors). The code words should be widely separated (i.e., the sequences of ones and zeros should be as different looking as possible) to minimize decoder errors. It would be preferable if $k \sim n$, but generally $2^k << 2^n$ for good codes.

EXAMPLE 4.1

Assume that the code word $\underline{u}$ was sent and that the channel creates two errors; that is, let

$$\underline{u} = 1011001$$
$$\underline{e} = 0010001$$

Then, $\underline{z} = \underline{u} \oplus \underline{e} = 1001000$. Somehow the decoder must recognize that $\underline{z}$ is not a possible code vector and then determine $\underline{e}$.

▲

The basic idea is to generate a code that permits the decoder to perform its function. Two matrices are developed that keep track of the digital strings that make up the code; these are the code generator $\underline{G}$ and the parity check $\underline{H}$. Although they are generally developed in parallel, $\underline{G}$ is discussed first.

Let the set $\{ \underline{v}_1, \underline{v}_2, \ldots, \underline{v}_k \}$ form a basis in the subspace; then define the code generator $\underline{G}$ by

$$\underline{G} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The generated code word is $\underline{u} = \underline{m}\,\underline{G}$, where $\underline{m}$ is the message vector that defines the operation of the encoder.

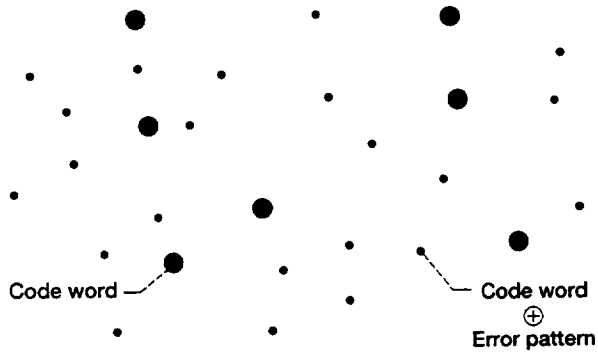


Figure 4.1.—Schematic for n-dimensional vector space. (The large dots are the set of n-tuples that form the code. The unused vectors may appear at the decoder if the demodulator makes an error.)

EXAMPLE 4.2

For a (6,3) code, choose

$$\underset{\sim}{G} = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Note that the rank of $\underset{\sim}{G}$ is k. Also note that the last three columns form the identity matrix. The code is systematic if

$$\underset{\sim}{G} = \left[\underset{\sim}{P} \mid \underset{\sim}{I}_k\right]$$

where $\underset{\sim}{P}$ is the parity array portion. The code word is

$$\underset{\sim}{u} = (n-k) \text{ parity bits}, \underbrace{m_1, \ldots, m_k}_{k \text{ message bits}}$$

Note that here the "block" is turned around (i.e., the parity check bits are first and the message bits follow). Both forms of $\underset{\sim}{G}$ are used in the literature:

$$\underset{\sim}{G} = \left[\underset{\sim}{P} \mid \underset{\sim}{I}_k\right]$$

or

$$\underset{\sim}{G} = \left[\underset{\sim}{I}_k \mid \underset{\sim}{P}\right]$$

The code word set is the row space of $\underset{\sim}{G}$. The all-zero word is always a code word.

▲

EXAMPLE 4.3

For a linear (5,3) code, choose

$$\underset{\sim}{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

The number of code words is $2^k = 2^3 = 8$. All of the code words are 00000 $\boxed{10011}$ $\boxed{00101}$ 10110 $\boxed{01010}$ 11001 01111 11100, where the boxed ones form the basis. The code has k (here three) dimensions. Arrange these basis code words as

$$\underset{\sim}{G} = \left[\begin{array}{ccc|cc} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array}\right] = \left[\underset{\sim}{I}_3 \,\vdots\, \underset{\sim}{P}\right] \qquad \therefore \underset{\sim}{P} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Only the $\underset{\sim}{P}$ matrix distinguishes one (n,k) code from another. Encode via $\underset{\sim}{G}$ as follows:

$$\underset{\sim}{C} = \underset{\sim}{v}_m \, \underset{\sim}{G}$$

where $\underset{\sim}{v}_m$ is the message vector and $\underset{\sim}{C}$ is the code word. Let $\underset{\sim}{v}_m = (101)$. Then,

$$(101)\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} = 10110 = \underset{\sim}{C}$$

Observe that $\underset{\sim}{C} = \left[\underset{\sim}{v}_m \; \underset{\sim}{v}_m \; \underset{\sim}{P}\right]$, which means that the message is transparent or that the code is systematic.

▲

4.1 Standard Array

The "standard array" is a table that describes the partitioning of received sequences such that a decoding strategy can be applied. The table is constructed as follows: The first row starts with the all-zero code word on the left, and all remaining code words, arranged in any order, fill out the row. Next, choose an error pattern and place it under the all-zero code word. For example, consider a (6,3) code generated by

$$\underset{\sim}{G} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The first row of the standard array is then

000000 011100 101010 110001 110110 101101 011011 000111

where the code words are found by using the $2^k = 2^3 = 8$ message vectors. That is, for $\underset{\sim}{m} = 101$

$$[101]\begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} = 101101 = \text{code word}$$

which is the sixth entry in the row. Note that the error pattern chosen for the next row cannot be any of the entries in the first row. Choose 100000 and add it to all the entries in the first row and place these sums under the code word used; that is, the first code word is 011100 and adding it to 100000 gives 111100, which is placed under 011100. The table is

000000	011100	101010	110001	110110	101101	011011	000111
100000	111100	001010	010001	010110	001101	111011	100111

Choose another error pattern (which does not appear anywhere in the table yet) and form the next row. For 010000, the table is

000000	011100	101010	110001	110110	101101	011011	000111
100000	111100	001010	010001	010110	001101	111011	100111
010000	001100	111010	100001	100110	111101	001011	010111

Continuing in this manner, the table becomes

000000	011100	101010	110001	110110	101101	011011	000111
100000	111100	001010	010001	010110	001101	111011	100111
010000	001100	111010	100001	100110	111101	001011	010111
001000	010100	100010	111001	111110	100101	010011	001111
000100	011000	101110	110101	110010	101001	011111	000011
000010	011110	101000	110011	110100	101111	011001	000101
000001	011101	101011	110000	110111	101100	011010	000110
100100	111000	001110	010101	010010	001001	111111	100011

Observe that the table has $2^n = 2^6 = 64$ entries, which are all of the possible 6-tuples. The code words are on the first row, and there are $2^k = 2^3 = 8$ of them. The error patterns with the fewest number of ones (hence, fewest errors) form the first column. The last entry was found by inspecting the table and choosing a vector with the fewest ones that was not in the table. The rows of the table are called cosets. The entries in the first column are called coset leaders. The entry in any row is the sum of that row's coset leader and the code word at the top of the column. All entries to the right of the vertical line and below the horizontal one represent all possible received vectors. A decoding scheme would choose the code word at the top of the column as the most likely one sent. Recall that the coset leaders are chosen to be the most likely error patterns. There are 2^{n-k} cosets and each coset contains 2^k n-tuples. Suppose the received vector is 101100 (which is the sixth entry in row 6); then, a maximum-likelihood decoder (MLD) would choose 101101 (the column header) as the probable code word.

In summary, the table as described would operate as follows: The decoder would recognize the first row as valid code words and pass them on. If any of the vectors in the fourth quadrant of the table (49 entries) are received, the decoder can process them and determine the coset leader (error pattern). Adding the error pattern $\underline{e}$ to the received vector will generate the code word at the top of the column. The last coset leader (100100) is the only double-error pattern discernible. Thus, for this special case the decoder can detect and correct all single-error patterns and one double-error pattern (the last coset leader). If any other double-error pattern occurs, the decoder will make a mistake. In other words, the decoder formed from this table is able to recognize just the errors that form the first column. The array gives a good intuitive understanding of decoding strategy and the ways errors can pass undetected. Note that the code is not just single-error correcting but can correct the given double-error pattern in the last row. In other words, the correctable patterns do not always fall into easily quantified limits.

EXAMPLE 4.4

Choose an $(n,k) = (6,3)$ code and let $\underset{\sim}{G}$ be

$$\underset{\sim}{G} = \left[\begin{array}{ccc:ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

which is slightly different from the $\underset{\sim}{G}$ used in the previous discussion. Here,

$$\underset{\sim}{P} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

The 2^k code vectors are the three rows of $\underset{\sim}{G}$ and their $\oplus$ sums. Thus, the code words are

101100 111010 011001 010110

100011 110101 001111 000000

The table is

000000	001111	110101	100011	010110	011001	111010	101100
000001	001110	110100	100010	010111	011000	111011	101101
000010	001101						
000100	001011			etc.			
001000	000111						
etc.	etc.						

▲

DEFINITIONS

The following definitions are needed for further discussion: Hamming weight is the number of ones in a code word $w(u)$; for example,

$$w(001101) = 3$$

Hamming distance $d(\underline{u}, \underline{v})$ is the number of places by which the code vectors $\underline{u}$ and $\underline{v}$ differ, or the number of bit changes to map $\underline{u}$ into $\underline{v}$. Let

$$\underline{u} = 110110$$

$$\underline{v} = 100101$$

$$\therefore d(\underline{u}, \underline{v}) = 3$$

It turns out that

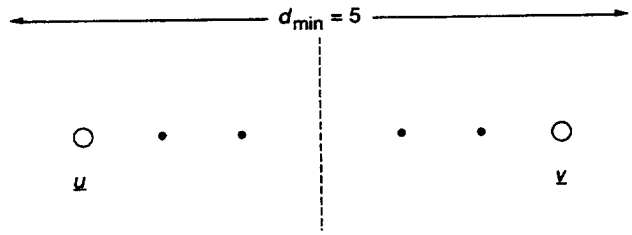


Figure 4.2.—Hamming distance. (If $\underline{u}$ is transmitted and if either of the vectors to the left of the dashed line are decoded, $\underline{u}$ is chosen. If either of the vectors to the right of the dashed line are decoded, $\underline{v}$ is chosen and an error occurs.)

$$d(\underline{u}, \underline{v}) = w(\underline{u} \oplus \underline{v}) = w(010011) = 3$$

The minimum Hamming distance d_{min} is the distance between the two closest code words; it is also the weight of the "lightest" code word. The error correction power of a given code is determined by d_{min}. The number of correctable errors t in a received word is

$$t = \frac{d_{min} - 1}{2}$$

This equality follows from "nearest neighbor decoding," which says that the received word is decoded into the code word "nearest" in Hamming distance (fig. 4.2).

EXAMPLE 4.5

Assume that the transmitted code word is $\underline{u} = 10001$ and that the received word is $\underline{z} = 10010$. Then, since $\underline{z} = \underline{u} \oplus \underline{e}$,

$$\underline{e} = \underline{z} \oplus \underline{u} = 00011$$

The ones in $\underline{e}$ correspond to the bits in error. Define t to be the weight of $\underline{e}$. Here, $t = 2$; thus,

$$t = \frac{d_{min} - 1}{2}$$

implies that d_{min} should be 5 or 6 to correct all possible double-error combinations in any code word.

▲

In an erasure, the error location is known, but no hint is given as to the bit value; for example,

$$\underline{z} = 1101\_101$$
$$\uparrow$$
erasure (a glitch such that digit is erased)

Then, define

e_c	number of errors corrected
e_d	number of errors detected
p	number of erasures corrected
x	number of erasures

It follows that

$$d_{\min} \geq e_c + e_d + 1 = x + 2e_c + 1 \geq p + 1 \qquad e_d \geq e_c$$

In the design phase, choose $e_c + e_d$ for the available $d_{\min}$, which freezes the decoder design. It can be shown that

$$d_{\min} \leq n - k + 1$$

4.2 Parity Check Matrix

The parity check matrix helps describe the code structure and starts the decoding operations. For a given generator

$$\underset{\sim}{G} = \left[\underset{\sim}{P} \,\vdots\, \underset{\sim}{I}_k\right]$$

the parity check is given by

$$\underset{\sim}{H} \triangleq \left[\underset{\sim}{I}_{n-k} \,\vdots\, \underset{\sim}{P}^T\right]$$

For example,

$$\underset{\sim}{G} = \left[\begin{array}{ccc:ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \qquad \begin{array}{l} n = 6 \\ k = 3 \end{array}$$

Then,

$$\underset{\sim}{H} = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

The rank of $\underset{\sim}{H}$ is $(n - k)$, and its row space is the null space of the code words developed by $\underset{\sim}{G}$. Then,

$$\underset{\sim}{G}\,\underset{\sim}{H}^T = \underset{\sim}{0}$$

Thus,

$$\underset{\sim}{u}\,\underset{\sim}{H}^T = \underset{\sim}{0}$$

The parity check generation scheme can be determined by inspecting the rows of $\underset{\sim}{H}$. In the preceding equation, let a_i represent the ith digit in the message vector; then (in the right partition),

1. First row means that $a_1 \oplus a_3$ is the first check digit.

2. Second row means that $a_1 \oplus a_2$ is the second check digit.
3. Third row means that $a_2 \oplus a_3$ is the third check digit.

Thus, H is useful in describing the encoding process. The equation

$$\underline{u}\underset{\sim}{H}^T = \underline{0}$$

is the key to detecting errors. Here, $\underline{u}$ is a valid code word. If

$$\underline{r}\,\underset{\sim}{H}^T \neq \underline{0}$$

then $\underline{r}$ is not a code word and the error pattern $\underline{e}$ must be found. From the standard array, $\underline{r}$ is somewhere in the table, and the header code word would be the decoded word. Consider a code word $\underline{v} = (\underline{v}_m, \underline{v}_c)$, where m means message portion and c means check portion. Form the syndrome defined by

$$\underset{\sim}{H} = \left[\begin{array}{c|c} \underset{\sim}{P}^T & \underset{\sim}{I}_{n-k} \end{array}\right]$$

$$\underline{S} = \underline{v}\,\underset{\sim}{H}^T = \underline{v}_m\,\underset{\sim}{P} \oplus \underline{v}_c$$

Thus, $\underline{S}$ is an $(n-k)$ vector, where $\underline{v}_m\,\underset{\sim}{P}$ are the locally generated checks and $\underline{v}_c$ are the received checks. If $\underline{S} = \underline{0}$, no errors are detected. If $\underline{S} \neq \underline{0}$, errors are present. Thus, $\underline{S}$ is determined solely by the error pattern $\underline{e}$. Observe that if $\underline{r} = \underline{u} \oplus \underline{e}$,

$$\underline{S} = \underline{r}\,\underset{\sim}{H}^T = (\underline{u} \oplus \underline{e})\,\underset{\sim}{H}^T = \underline{u}\,\underset{\sim}{H}^T \oplus \underline{e}\,\underset{\sim}{H}^T = \underline{0} \oplus \underline{e}\,\underset{\sim}{H}^T = \underline{e}\,\underset{\sim}{H}^T$$

That is, each error has a specific syndrome.

The properties of H are as follows:

1. No columns are all zero.
2. All columns are unique.
3. The dual code of an (n,k) code is generated by $\underset{\sim}{H}$. That is, $u_{\text{dual}} = \underline{m}\,\underset{\sim}{H}$.
4. The rank of $\underset{\sim}{H}$ is the degree of $\underset{\sim}{G}$ (row rank is the number of linearly independent rows).
5. The number of checks equals the row rank of $\underset{\sim}{H}^T$.

4.3 Syndrome Decoding

Syndrome decoding is the basic decoding scheme used in block codes. Basically, it relies on the fact that each error pattern generates a specific syndrome. Essentially, the decoder takes the message bits and regenerates the parity checks. It then compares them with the transmitted checks (by modulo-2 addition). If the sum is zero, no error is assumed. If the sum is not zero, at least one of the received digits is in error. The decoder must then determine which bits are in error. The error correction procedure is as follows:

1. Calculate the syndrome $\underline{S} = \underline{r}\,\underset{\sim}{H}^T = \underline{e}\,\underset{\sim}{H}^T + \underline{u}\,\underset{\sim}{H}^T = \underline{e}\,\underset{\sim}{H}^T$ (there are 2^{n-k} syndromes).
2. From $\underline{S}$ determine the error pattern (the tough step).
3. Let $\hat{\underline{e}}$ be the error pattern determined from step 2. Note that it may not be the true error pattern shown in step 1.
4. Form $\hat{\underline{u}} = \underline{r} + \hat{\underline{e}}$

Note that if $\hat{\underline{e}} = \underline{e}$, then $\hat{\underline{u}} = \underline{u}$ and correct decoding is achieved. It will be shown, however, that the estimate $\hat{\underline{e}}$ is not always correct and a decoding error occurs. The probability of such an error is the measure of the code's strength. Since 2^{n-k} syndromes are possible for an (n,k) code, 2^{n-k} error patterns are correctable. There are $2^n - 2^{n-k}$ uncorrectable patterns, and if $\underline{e}$ is one of them, a decoding error occurs. Some complications associated with syndrome decoding are as follows:

1. Several $\underline{e}$ patterns yield the same syndrome $\underline{S}$.
2. Some $\underline{e}$ patterns are code words and thus undetectable errors.
3. Since a maximum-likelihood decoder (MLD) always assumes an $\underline{e}$ with the lowest weight (fewest errors), decoding errors occur.

EXAMPLE 4.6

Consider a (6,3) code, and decode using the standard array. Let

$$\underset{\sim}{G} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \underline{v}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Therefore,

$$\underset{\sim}{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

Then, the number of code words is $2^k = 2^3 = 8$ (table 4.1).

TABLE 4.1. — CODE WORDS

Symbol	Code word	Weight
$\underline{v}_1$	110100	3
$\underline{v}_2$	011010	3
$\underline{v}_3$	101110	4
$\underline{v}_4$	101001	3
$\underline{v}_5$	011101	4
$\underline{v}_6$	110011	4
$\underline{v}_7$	000111	3
$\underline{v}_8$	000000	0

The weight column shows that $d_{\min} = 3$, so $t = 1$; or single-error correcting is guaranteed. The array is

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110

Observe that the last coset has two errors and was chosen arbitrarily. Thus, a double-error pattern is correctable, which is in addition to the guaranteed single-error patterns. The syndromes are

$$\underline{S}_j = \underline{e}_j \underline{H}^T$$

Then,

TABLE 4.2. — VALUES OF $\underline{e}_j$ AND $\underline{S}_j$

$\underline{e}_j$	$\underline{S}_j$
000000	000
000001	101
000010	011
000100	110
001000	001
010000	010
100000	100
010001	111

Then, each $\underline{e}_j$ has a unique $\underline{S}_j$ (table 4.2). Suppose that the channel adds the error

$$\underline{e} = 100100$$

Then,

$$\underline{S} = [100100]\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = 100 \oplus 110 = 010$$

and the decoder would choose $\underline{e} = 010000$ (from the previous $\underline{e}_j$-$\underline{S}_j$ table); thus, a decoder error has occurred. ▲

4.4 Classes of Code

Because the classes or types of code are extensive, only some of the more common ones are discussed here. The classes or types are not mutually exclusive, as a subset of one class may be a subset of another class or classes. The most useful codes are linear group block codes or linear convolutional codes. Some block codes are listed below:

1. Cyclic codes—Codes where a cyclic shift in a code word generates another code word (i.e., if 10110110 is a code word, an end-around shift gives 01011011, which is also a code word).
2. Bose-Chaudhuri-Hocquenghem (BCH) codes—A cyclic code with the property

$$n = 2^m - 1 \qquad m = 3,4,5,...$$

To correct t errors, one needs

$$n - k \leq mt$$

or

$$k \geq n - mt, \qquad d_{\min} \geq 2t + 1$$

For example, let $m = 4$, $t = 2$, and $k = 7$. Thus, a (15,7) code results, and $d_{\min} = 5$.

3. Golay codes—One of the three types of "perfect" code (i.e., a t-error-correcting code whose standard array has all the error patterns of t (or fewer) errors and no others as coset leaders). The two binary forms are (23,12) and (24,12). For these, $t = 1$.
4. Hamming codes —Hamming codes have the properties

$$n = 2^m - 1, \qquad n - k = m \qquad m = 1,2,3,...$$
$$d_{\min} = 3, \qquad t = 1$$

Note that there are 2^{n-k} different binary sequences of length $n - k$ (delete the all-zero sequence); then,

$$n = 2^m - 1$$

which defines these codes.

EXAMPLE 4.7

For the (7,4) Hamming code there are seven possible sequences of length three to choose from: 001, 010, 011, 100, 101, 110, 111. Choose four out of the seven; $\binom{7}{4} = 35$ choices. If the code is to be systematic (two or more binary ones are needed), choose four out of four (hence, only one choice). However, the number of permutations of the four is 4! = 24, which means 24 distinct choices for $\underset{\sim}{H}$. Choose the following pair:

$$\underset{\sim}{H}^T = \begin{bmatrix} 011 \\ 101 \\ 110 \\ \hline 111 \\ \hline 100 \\ 010 \\ 001 \end{bmatrix}, \qquad \underset{\sim}{G} = \left[\begin{array}{c:c} 1000 & 011 \\ 0100 & 101 \\ 0010 & 110 \\ 0001 & 111 \\ \hline \left(\underset{\sim}{I}_k\right) & \left(\underset{\sim}{P}\right) \end{array}\right]$$

The encoder is designed from $\underset{\sim}{H}$ (fig. 4.3). In the figure, m_1, m_2, m_3, and m_4 are the message bits and C_1, C_2, and C_3 are the three checks. The checks are read from each column of $\underset{\sim}{H}$. Here,

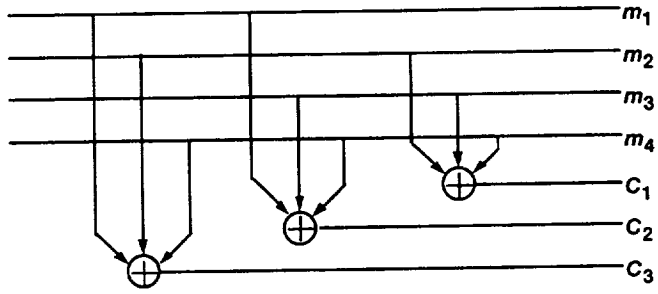


Figure 4.3.—Encoder for (7,4) Hamming code. (The three checks are developed from the four message bits, m_1, m_2, m_3, and m_4.)

$$C_1 = m_2 \oplus m_3 \oplus m_4$$
$$C_2 = m_1 \oplus m_3 \oplus m_4$$
$$C_3 = m_1 \oplus m_2 \oplus m_4$$

For example, let the code word $\underline{y} = \underline{x}\,\underline{G}$

$$\underline{x} = 1011$$
$$\underline{y} = 1011010$$

Assume an error in the fifth digit (counting from the left); then,

$$\underline{e} = 0000100$$

and

$$\underline{z} = \underline{y} \oplus \underline{e} = 1011110$$

At the decoder, calculate $\underline{S}$

$$\underline{S} = \underline{z}\,\underline{H}^T = 100$$

Because 100 is the fifth row of $\underline{H}$, the fifth digit is in error. The decoder generates $\underline{e}$ and adds this to $\underline{z}$ to correct the error. This association of fifth with fifth is a special case and should not be considered typical. A decoder for the (7,4) Hamming code appears in figure 4.4. ▲

Hamming codes have the following miscellaneous properties:

1. The total number of distinct Hamming codes with $n = 2^m - 1$ is given by

$$\text{number} = \frac{\left(2^m - 1\right)!}{\prod_{i=0}^{m-1}\left(2^m - 2^i\right)}$$

For the (7,4) Hamming code, $m = 3$ and

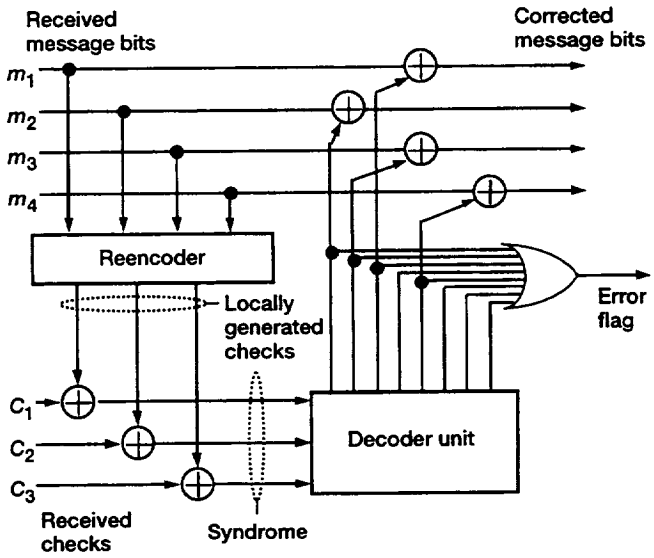


Figure 4.4.—Decoder for (7,4) Hamming code generated in figure 4.3.

$$\prod_{i=0}^{2} = \left(2^3 - 1\right)\left(2^3 - 2\right)\left(2^3 - 2^2\right) = (7)(6)(4)$$

$$\therefore \text{number} = \frac{7!}{(7)(6)(4)} = 30$$

2. Dual Hamming codes are known as maximal length codes:

$$n = 2^m - 1, \quad d = 2^m - 1, \quad k = m$$

In the following codes, all nonzero code words have the same weight; hence, all distances between code words are the same (referred to as "a simplex"):

1. Reed-Muller codes—Cyclic codes with an overall parity check digit added

$$n = 2^m, \quad k = \sum_{i=0}^{r} \binom{m}{i}, \quad d_{\min} = 2^{m-r}$$

2. Goppa codes—A general noncyclic group that includes the BCH (which are cyclic); mainly of theoretical interest

3. Fire codes—Codes for correcting bursts of errors. A burst of length b is defined as a string of b bits, the first and last of which are ones already there. Here,

$$b = \frac{n - k + 1}{3}$$

and

$$n = \mathrm{LCM}\left[2^m - 1, 2b - 1\right]$$

4. Reed-Solomon codes—Often used nonbinary codes with the following properties:

$$n = m\left(2^m - 1\right) \text{ bits}, \quad k = n - 2t \text{ bits}, \quad d = m(2t+1) \text{ bits}$$

4.5 Decoders

The standard array partitions all the possible 2^n n-tuples that may be received into rows and columns. The decoder receives $\underline{r}$ and finds $\underline{S}$. It determines $\underline{e}$ by either a lookup table, or other means, and adds this to $\underline{r}$ to recover the transmitted code word. This scheme is known as maximum-likelihood decoding (MLD). Block decoders are generally classified as algebraic or nonalgebraic. Algebraic types solve sets of equations to determine $\underline{e}$; the others use special algorithms. A class of nonalgebraic decoders, called information set decoders, includes Meggit and threshold types. The decoding processes are discussed in chapter 5. In general, hard decisions are used, as soft decisions cause algorithm and circuit complexity problems. Some decoders handle erasures as well as errors. Error-trapping decoders are discussed in Lin and Costello (1983).

4.6 Counting Errors and Coding Gain

For simplicity, only binary coding and decoding are assumed. Then, the energy between an uncoded and coded bit is straightforward,

$$E_c = \frac{k}{n} E_b = r\, E_b \tag{4.1}$$

where E_c is the energy for a coded bit (one leaving the encoder), E_b is the energy for an information bit (one entering the encoder), and r is the code rate. For the many digital modulation schemes used, the modems generate and make decisions on symbols (groups of bits), so that the counting of bit errors is more involved. If the codec is turned off, $r = 1$ and $E_c = E_b$. A given modulation scheme has a bit-error-rate-versus-E_b/N_o plot, which is the probability of received bit error p_b versus the ratio of energy per bit to noise power. For binary phase shift keying (BPSK) the relationship is

$$p_b = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \tag{4.2}$$

and is plotted in figure 4.5. Without coding, the theoretical probability of error is given by equation (4.2). However, in a real system, the curve (fig. 4.5) would be pushed to the right somewhat to account for implementation losses. When coding is applied, the probability of a bit error is (subscript c means coded)

$$p_c = Q\left(\sqrt{\frac{2E_c}{N_o}}\right) = Q\left(\sqrt{\frac{2rE_b}{N_o}}\right) \tag{4.3}$$

Note that because

$$p_c > p_b$$

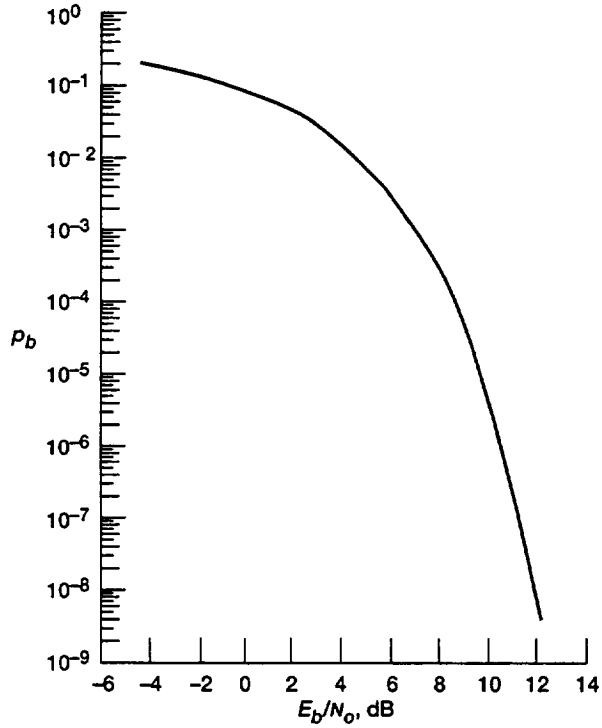


Figure 4.5.—Probability of error per bit in BPSK signaling system versus ratio of energy per bit to noise power spectral density E_b/N_o. (For BPSK, QPSK, MSK, and OKQPSK (gray coded), $p_u = 1/2 \text{ erfc} \sqrt{2E_b/N_o} = Q(\sqrt{2E_b/N_o})$.)

more errors are emerging from the demodulator. The decoder only works on blocks of bits (code words); therefore, the block error rate must be determined for blocks emerging from the decoder, given the channel bits with error probability entering the demodulator. Once this block error rate is found, the resulting bit error rate must be somehow calculated into the data sink. This last step is difficult, and many approximations are used in the literature.

The probability that a block is decoded incorrectly may be called p_B. In the literature,

$$\text{prob (block decoded in error)} = p_m \text{ (message error)} = p_w \text{ (word error)} = p_E \text{ (decoder error)} = p_B$$

Once p_B has been found, the probability of binit (bit) errors emerging from the decoder can be approximated. Then, $(p_b)_s$ (here subscript s means error going into the data sink) can be plotted versus E_b/N_o to see how the code performs. Figure 4.6 shows the uncoded BPSK curve along with those for two (n,k) codes. Note that the vertical axis is both p_b and $(p_b)_s$. Observe that the shapes of the two $(p_b)_s$-versus-E_b/N_o curves are not the same and that neither is representable by some standard $Q(\bullet)$ curve. Each has been calculated point by point. The "threshold points" for both codes are near $E_b/N_o \approx 6$ dB (where they intersect the uncoded curve). If $E_b/N_o < 6$ dB, coding degrades performance because the number of errors is so great that in each received word the number of errors is larger than the error patterns the code has been designed for. Also, the demodulator makes more errors than in the uncoded case, since now decisions are made on pulses with less signal energy while coded. For $E_b/N_o > 6$ dB, the correcting power kicks in and improves performance. In this range, the correction capability overtakes the extra demodulator errors that occur due to the lower pulse energy in coded conditions.

The coding gain is the difference in E_b/N_o between the coded and uncoded plots for the same $p_b = (p_b)_s$. For example, the gain for the (n_2, k_2) code at $p_b = 10^{-5}$ is about 1.5 dB. It can be shown that the asymptotic gain is roughly

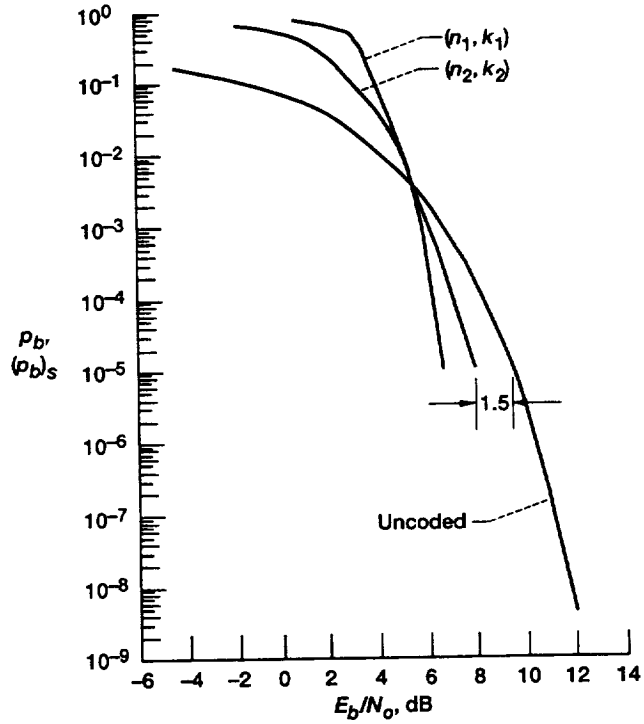


Figure 4.6.—Bit error rate of two (n, k) codes along with basic curve for BPSK. (At $p = 10^{-5}$ the (n_2, k_2) code has a 1.5-dB coding gain.

$$G = \text{gain (asymptotic)} \simeq 10 \log[r(t+1)] \qquad \text{for hard decisions}$$

$$\simeq 10 \log[rd_{\min}] \qquad \text{for soft decisions}$$

Here, G is in decibels.

EXAMPLE 4.8

Calculate the change in bit error rate between an uncoded and coded situation. Assume BPSK in Gaussian noise, and assume that the (15,11) BCH ($t = 1$) code is used. Also assume that hard decisions are made. This problem illustrates the nature of approximations needed to determine the coding gain. The decoder operates only on blocks of digits; therefore, if a block is decoded incorrectly, the bit error rate cannot be determined.

Let p_u and p_c represent the uncoded and coded channel bit (more generally, symbol) error probabilities.

$$p_u = Q\left(\sqrt{\frac{2E_b}{N_o}}\right), \quad p_c = Q\left(\sqrt{\frac{2E_c}{N_o}}\right)$$

Here, E_b and E_c are the bit energies in the uncoded and coded cases. Let $E_b/N_o = 8.0$ dB for purposes of calculation and assume the data rate $R = 4800$ bits/sec. Then, without coding,

$$\frac{E_b}{N_o} = 6.3096, \quad \frac{S}{N_o} = R\left(\frac{E_b}{N_o}\right) = 30\ 286\ (44.8\ \text{dB})$$

$$p_u = Q\left(\sqrt{12.62}\right) = 2.0425 \times 10^{-4}$$

where the following approximation for $Q(x)$ was used:

$$Q(x) \cong \frac{1}{x\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \qquad x > 3$$

The probability that the uncoded message block will be received in error $(p_m)_u$ is calculated as follows: Each block contains 11 digits (slots). The probability of no error in any slot is $(1 - p_u)$. For 11 consecutive slots, the probability of no error is $(1 - p_u)^{11}$. Then, the probability of some errors in a block is $1 - (1 - p_u)^{11}$. Thus,

$$(p_m)_u = 1 - (1 - p_u)^k = 1 - (1 - p_u)^{11} = 2.245 \times 10^{-3}$$

is the probability that at least one bit is in error out of all 11 in the message block.

With coding,

$$\frac{E_c}{N_o} = R\frac{E_b}{N_o} = \frac{11}{15}\frac{E_b}{N_o}$$

so that

$$p_c = Q\left(\sqrt{\frac{11}{15}(12.62)}\right) = 1.283 \times 10^{-3}$$

Note that

$$p_c > p_u$$

as stated earlier. The code performance is not yet apparent, but it will be shown later that $(p_m)_c$, the block error rate for a t-error-correcting code, is

$$(p_m)_c = \sum_{j=t+1}^{n} \binom{n}{j} (p_c)^j (1 - p_c)^{n-j}$$

and here $t = 1$ and $n = 15$. A good approximation is just the first term; then,

$$(p_m)_c = \binom{15}{2} (p_c)^2 (1 - p_c)^{13} = 1.7 \times 10^{-4}$$

Observe that block error rate for coding $(p_m)_c$ is less than that for uncoded blocks $(p_m)_u$; that is,

$$(p_m)_c = 1.7\times10^{-4} < (p_m)_u = 2.245\times10^{-3}$$

even though more bit errors are present at the demodulator output with coding. Note that

$$\frac{(p_m)_u}{(p_m)_c} = 13.2$$

or the code has improved the message error rate by a factor of 13.2. Now, from the block error rate calculate the resulting bit error rate. A commonly used approximation is

$$(p_b)_s \simeq \frac{1}{n}\sum_{i=t+1}^{n} i\binom{n}{i} p_c^i (1-p_c)^{n-i}$$

and when $t = 1$, this can be shown to reduce to (Sklar (1988), appendix D)

$$(p_b)_s = p_c\left[1-(1-p_c)^{n-1}\right] = 2.285\times10^{-5}$$

Table 4.3 determines the message or block error rates for a range of E_b/N_o; they are plotted in figure 4.7 along with the standard BPSK curve. Note that the coded case is worse than the uncoded one at 4 dB and crosses at about 4.7 dB.

TABLE 4.3. — BLOCK ERROR RATES

E_b/N_o, dB	p_u	p_c	$(p_m)_u$	$(p_m)_c$
4	0.012576	0.1237	0.12996	0.2887
6	0.00241	0.00787	2.619×10^{-2}	5.874×10^{-3}
7	8.39×10^{-4}	3.368×10^{-3}	9.19×10^{-3}	1.14×10^{-3}
8	2.043×10^{-4}	1.283×10^{-3}	2.245×10^{-3}	1.7×10^{-4}
8.5	8.929×10^{-5}	6.89×10^{-4}	9.82×10^{-4}	4.94×10^{-4}
9.0	3.554×10^{-5}	3.45×10^{-4}	3.91×10^{-4}	1.24×10^{-5}
9.5	1.273×10^{-5}	1.6×10^{-4}	1.4×10^{-4}	2.69×10^{-6}
9.6	1.022×10^{-5}	1.36×10^{-4}	1.13×10^{-4}	1.94×10^{-6}
10.0	4.05×10^{-6}	6.8×10^{-5}	4.46×10^{-5}	4.86×10^{-7}

Table 4.4 gives the $(p_b)_s$ or the bit error rate into the sink; this is plotted in figure 4.8. It crosses the BPSK curve at about 5.5 dB. At $(p_b)_s = 1.0\times10^{-7}$, the gain is about 1.3 dB. The approximate gain given earlier is

$$G(\text{asym})_{\text{dB}} \simeq 10\log_{10}\left[\frac{11}{15}(2)\right] = 1.66 \text{ dB}$$

which agrees within the normal limits in such problems.

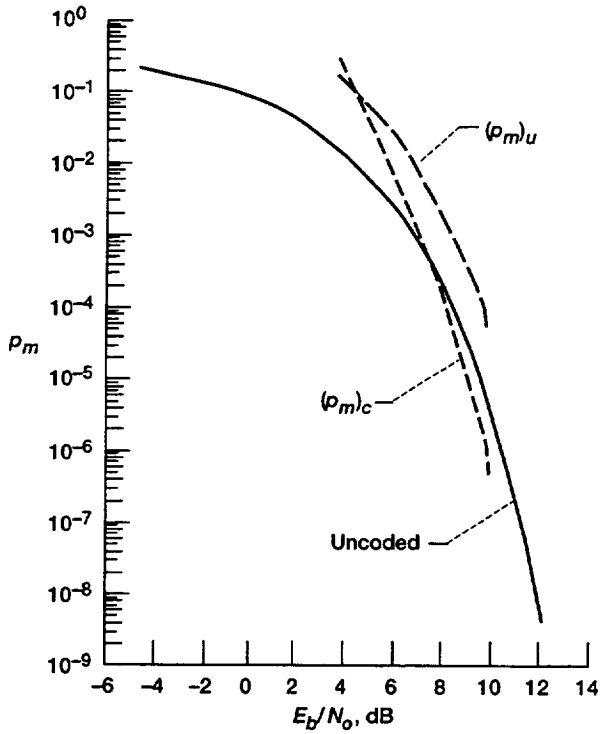


Figure 4.7.—Coded $(p_m)_c$ and uncoded $(p_m)_u$ block error rates (dashed lines) for (15, 11), $t = 1$ code.

TABLE 4.4. — BIT ERROR RATE INTO SINK

E_b/N_o, dB	p_c	$(p_b)_s$
4	0.1237	0.1042
6	7.87×10^{-3}	8.249×10^{-4}
7	3.368×10^{-3}	1.554×10^{-4}
8	1.283×10^{-3}	2.285×10^{-5}
8.5	6.8872×10^{-4}	6.611×10^{-6}
9.0	3.45×10^{-4}	1.664×10^{-6}
9.5	1.6×10^{-4}	3.59×10^{-7}
9.6	1.36×10^{-4}	2.583×10^{-7}
10.0	6.81×10^{-5}	6.48×10^{-8}

▲

The calculation of the probability of a bit error from a decoder is necessarily vague, since the causes for errors are found in signaling technique, signal-to-noise ratio, interference, demodulator type, decoder implementation, code, etc. Essentially, the decoder emits blocks that should be code words. However, the blocks can be erroneous for two basic reasons. First, the error pattern could be a code word; thus, an

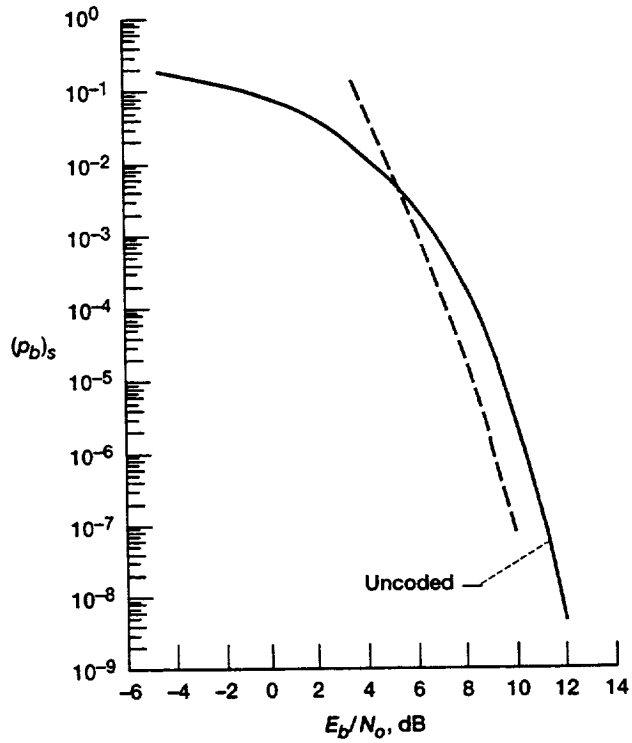


Figure 4.8.—Bit error rate of (15, 11), $t = 1$ code (dashed line).

undetected error event occurs. The number of incorrect bits is indeterminant; all we know is that the block is in error. Second, the error pattern could have more errors than the code can handle; this is sometimes called algorithm or code shortcomings. Summarizing the determination of code gain again,

1. The uncoded bit error rate is known from basic modulation theory; for example, (n,k)(BPSK)

$$p_u = Q\left(\sqrt{\frac{2E_b}{N_o}}\right)$$

2. The coded bit error rate is then calculated for an (n,k) code as

$$p_c = Q\left(\sqrt{r\frac{2E_b}{N_o}}\right) \qquad r = \frac{k}{n}$$

3. The uncoded message, or block, error rate can be found by

$$(p_m)_u = 1-(1-p_u)^k$$

but it is not necessary in the final analysis.

4. The coded message, or block, error rate must be found. Many expressions are available, and a commonly used one is

$$(p_m)_c = \sum_{i=t+1}^{n} \binom{n}{i} p_c^i (1-p_c)^{n-i}$$

5. Once this is found, the number of bit errors into the sink $(p_b)_s$ is calculated. A commonly used expression is

$$(p_b)_s = \frac{1}{n}\sum_{i=t+1}^{n} i\binom{n}{i} p_c^i (1-p_c)^{n-1}$$

which is written in terms of the coded probability p_c. The form of $(p_b)_s$ is nearly the same as $(p_m)_c$ except that each term in $(p_b)_s$ is weighted by the factor i/n.

6. Plotting p_u and $(p_b)_s$ on a common scale permits the graphical determination of the gain.

The interplay between p_u, $(p_m)_c$, and $(p_b)_s$ depends on the code structure, the algorithm implementation, and the form chosen for $\underset{\sim}{G}$. Different equations are found for $(p_m)_c$ and $(p_b)_s$ because various assumptions are used and special formulas are valid for specific codes. Thus, the literature is rich in formulas, many of which are summarized here.

4.6.1 Formulas for Message or Block Errors

The following notation is used:

$$(p_m)_c \equiv p_B$$

In the concept of the weight distribution of a code, the number of code words with the specific weight i is represented by A_i. The complete set of $\{A_i\}$ represents the complete weight distribution for the specific code. The weight distributions for some codes are known and published, but many codes have unknown distributions. The distribution is denoted by the enumerating polynomial

$$A(x) = \sum_{i=1}^{n} A_i x^i$$

where A_i is the number of code words with weight i. For the dual $(n, n-k)$ code, the enumerator is known to be

$$B(x) = 2^{-k}\left(1+x^n\right) A\left(\frac{1-x}{1+x}\right)$$

For Hamming codes,

$$A(x) = \frac{1}{n+1}\left[(1+x)^n + n(1+x)^{\frac{n-1}{2}}(1-x)^{\frac{n+1}{2}}\right]$$

For their duals, which are maximal length codes $(2^m - 1, m)$,

$$A(x) = 1 + \left(2^m - 1\right)x^{2^{m-1}}$$

For the Golay (23,12) code,

$$A(x) = 1 + 253\left(x^7 + 2x^8 + 2x^{15} + x^{16}\right) + 1288\left(x^{11} + x^{12}\right) + x^{23}$$

For the extended Golay (24,12) code,

$$A(x) = 1 + 759\left(x^8 + x^{16}\right) + 2576\, x^{12} + x^{24}$$

Note that for the extended code, the odd-weight code words (weights 7, 15, 11, and 23 of the basic code) have been eliminated. For Reed-Solomon codes,

$$A_i = \binom{N}{i}(q-1)\sum_{j=0}^{i-D}(-1)^j\binom{i-1}{j}q^{i-j-D} \qquad i \geq D = D_{\min},\ q = 2^k$$

An (n,k) code can detect $2^n - 2^k$ error patterns. Of these, 2^{n-k} are correctable. The number of undetectable error patterns is $2^k - 1$. The most commonly used formula for p_B is

$$p_B = (p_m)_c = \sum_{i=t+1}^{n}\binom{n}{i}p_c^i\,(1-p_c)^{n-i}$$

which is due to algorithm shortcomings (i.e., more errors than the code can handle). The block errors due to undetected errors may have the following forms:

$$p_B(\text{undetected}) = \sum_{i=d_{\min}}^{n} A_i p_c^i (1-p_c)^{n-i}$$

(note that $A_i = 0$ for $i < d_{\min}$) or

$$p_B(\text{undetected}) = 1 - \sum_{i=0}^{t}\binom{n}{i}p_c^i(1-p_c)^{n-i} = \sum_{j=d_{\min}-1}^{n}\binom{n}{j}p_c^j(1-p_c)^{n-j}$$

For the special case of codes with $n - k = 1$,

$$p_B(\text{undetected}) = \frac{n}{2} \qquad n \text{ even}$$

$$= \sum_{j=1}^{\frac{n-1}{2}}\binom{n}{2j}p_c^{2j}(1-p_c)^{n-2j} \qquad n \text{ odd}$$

In general, the complete expression for p_B is the sum of both; that is,

$$p_B\,(\text{total}) = p_B + p_B\,(\text{undetected})$$

However, in the literature it is seldom clear what approximations are made (i.e., if the undetected portion is omitted or not).

Many bounds for p_B have been developed to help in the general case, and the most commonly discussed ones are

1. Sphere packing (upper bound case)

$$p_B \le \sum_{j=\frac{d_{\min}+1}{2}}^{n} \binom{n}{j} p_c^j (1-p_c)^{n-j} \qquad n \text{ odd}$$

$$\le \sum_{j=\frac{d_{\min}}{2}}^{n} \binom{n}{j} p_c^j (1-p_c)^{n-j} \qquad n \text{ even}$$

2. Union bound

$$p_B \le \sum_{j=1}^{n} A_j \begin{cases} \sum_{i=\frac{j+1}{2}}^{j} \binom{j}{i} p_c^i (1-p_c)^{j-i} & j \text{ odd} \\ \frac{1}{2}\binom{j}{j/2} p_c^{j/2} (1-p_c)^{j/2} + \sum_{i=\frac{j}{2}+1}^{j} \binom{j}{i} p_c^i (1-p_c)^{j-i} & j \text{ even} \end{cases}$$

3. Sphere packing (lower bound case). Let t be the largest integer such that

$$2^{n-k} \ge \sum_{i=0}^{t} \binom{n}{i}$$

and

$$N_{t+1} = 2^{n-k} - \sum_{i=0}^{t} \binom{n}{i}$$

Then,

$$p_B > 1 - \sum_{i=0}^{t} \binom{n}{i} p_c^i (1-p_c)^{n-i} - N_{t+1}\, p_c^{t+1} (1-p_c)^{n-t-1}$$

4. Plotkin (a lower bound). This is a bound on the minimum distance available. The effect on p_B is therefore indirect.

$$n - k > 2d_{\min} - 2 - \log_2 d_{\min}$$

In these formulas, the inequalities become exact only for the "perfect codes" (i.e., Hamming, Golay, and repetition).

4.6.2 Formulas for Bit Errors Into Sink

The most common formula for the bit error rate from the decoder, which goes to the sink, is

$$(p_b)_s = \frac{1}{n}\sum_{i=t+1}^{n} \beta_i \binom{n}{i} p_c^i (1-p_c)^{n-i}$$

where β_i is the number of remaining errors in the decoded block. Obviously, this number is vague and the following limits are generally imposed:

$$i - t \le \beta_i \le i + t$$

Here, i is the number of errors entering the decoder. Other forms are

$$(p_b)_s = \frac{1.5\,t+1}{n} p_B(\text{total})$$

$$(p_b)_s = \sum_{i=1}^{n} \frac{i}{n}\binom{n}{i}\frac{(p_m)_u}{2^n-1} = \frac{2^{n-1}}{2^n-1} p_B$$

$$(p_b)_s = \frac{d_{\min}}{n} p_B(\text{undetected});\quad p_B(\text{undetected}) = \frac{p_c^{d_{\min}}}{2^{n-k}}$$

$$(p_b)_s = \frac{2^{k-1}}{2^k-1} p_B$$

The reasoning behind these formulas is as follows: Under the pessimistic assumption that a pattern of i bit errors ($i > t$) will cause the decoded word to differ from the correct word in $(i + t)$ positions, a fraction $(i + t)/n$ of the k information symbols is decoded erroneously. Alternatively, a block error will contain at least $t + 1$ errors (if it is detectable) or $2t + 1$ bit errors (if it is not). Thus, on the average the factor $1.5t + 1$ results.

A result published by Torrieri (1984) is perhaps most accurate:

$$(p_b)_s = \frac{d_{\min}}{n}\sum_{i=t+1}^{d_{\min}} \binom{n}{i} p_c^i (1-p_c)^{n-i} + \frac{1}{n}\sum_{i=d_{\min}+1}^{n} i\binom{n}{i} p_c^i (1-p_c)^{n-i}$$

or

$$= \sum_{i=\frac{n+1}{2}}^{n} \binom{n}{i} p_c^i (1-p_c)^{n-i}$$

The first equation is exact for the odd-n repetition code, $d = n$, $k = 1$.

Some simple bounds on $(p_b)_s$ can be developed as follows: Consider 1 sec of transmission; the number of code words transmitted during this interval is $1/T_w$, where T_w is the duration for a code word. Since each code word contains k information symbols, the total number of information symbols transmitted is k/T_w. The number of word errors is p_B/T_w. If α denotes the number of information symbol errors per word error, the bit error probability is

$$(p_b)_s = \frac{\alpha\, p_B / T_w}{k / T_w} = \alpha \frac{p_B}{k}$$

which is simply the ratio of the number of information symbols in error to the total number of information symbols transmitted. The problem, however, is to determine α, which varies from case to case. As a worst case, assume that each word error results in k information symbol errors; then,

$$(p_b)_s < p_B$$

The lower bound is obtained by considering the most favorable situation in which each word error results in only one information symbol error. For this case $\alpha = 1$ and

$$(p_b)_s > \frac{p_B}{k}$$

For small values of k, the bounds are tight and $(p_b)_s \sim p_B$.

A simple approximation for the high E_b/N_o cases is as follows: Here the symbol error probability is quite small, and word errors are probably due to $t + 1$ symbol errors. Of these $t + 1$ symbol errors, $(t + 1)(k/n)$ are, on the average, information symbol errors; thus,

$$\alpha = (t+1)\frac{k}{n}$$

and the approximation

$$(p_b)_s \simeq \frac{t+1}{n} p_B$$

follows. Another upper bound is

$$(p_b)_s \leq 2^{-n(r_0 - r)}$$

where

$$r_0 = 1 - \log_2\left[1 + \sqrt{4p_c(1-p_c)}\right]$$

is the cutoff rate.

The following bounds on $d_{\min}$ indirectly affect $(p_b)_s$:

1. Varsharmov-Gilbert-Sacks bound

$$\sum_{i=0}^{d_{\min}-2} \binom{n-1}{i} < 2^{n-k}$$

2. Elias bound

$$\frac{d_{\min}}{n} \leq 2A(1-A)$$

where $0 \leq A \leq 1$ and A satisfies the equation

$$r = \frac{k}{n} = 1 + A \log_2 A + (1-A)\log_2(1-A)$$

All BCH codes (which are used often) are known for $n \leq 1023$, and the relationships between n, $d_{\min}$, and t are

$$t = \begin{cases} \dfrac{d_{\min}}{2} - 1 & d_{\min} \text{ even} \\ \dfrac{d_{\min} - 1}{2} & d_{\min} \text{ odd} \end{cases}$$

$$n - k > b - 1 + \log_2 n$$

where b is the burst length.

For Hamming codes a special formula exists:

$$(p_b)_s = 1 - \gamma_0(1-p_c)^n - \gamma_1 p_c(1-p_c)^{n-1} - \gamma_2 p_c^2(1-p_c)^{n-2}$$

where γ_i is the number of coset leaders of weight i and

$$\gamma_i \leq \binom{n}{i}, \qquad \sum_{i=0}^{n} \gamma_i = 2^{n-k}$$

4.7 Formula Development

The extensive compilation of formulas for p_B and $(p_b)_s$ was necessary, since $(p_b)_s$ is needed to calculate the coding gain. Coding gain is the main figure of merit for a communications system application. The computed gain for a given code is at best rather approximate, and the uncertainty at $(p_b)_s = 10^{-5}$ is about 0.9 dB (difference between bounds). At $(p_b)_s = 10^{-6}$, this reduces to about 0.5 dB. Since the realizable gain for most practical situations is about 3.5 to 4.5 dB, the uncertainty is about 25 percent. This fact is part of the reason why bit-error-rate testers (BERT's) are often used to evaluate a codec pair on a simulated channel.

The columns of the standard array divide the n-tuples into subsets of words "close" to the column header. The number of n-tuples N_e in each set obeys the following (for a t-error-correcting code):

$$N_e \geq 1 + n + \binom{n}{2} + \ldots + \binom{n}{t}$$

Note that there are exactly n patterns that differ from the column header in one position, $\binom{n}{2}$ patterns that differ in two positions, etc. Previous examples show that almost always some patterns are left over after assigning all those that differ in t or fewer places (thus, the inequality). Since there are 2^n possible sequences, the number of code words N_c obeys

$$N_c = 2^k \leq \frac{2^n}{\left[1 + n + \binom{n}{2} + \ldots + \binom{n}{t}\right]}$$

which is known as the Hamming or sphere packing bound.

Several developments for the block error rate p_B are presented here. Note that

$$\text{prob(any one bit received correctly)} = (1 - p_c)$$

$$\text{prob (all } n \text{ received correctly)} = (1 - p_c)^n$$

$$\text{prob (received block has some error)} = 1 - (1 - p_c)^n$$

$$\text{prob (first bit in error; others correct)} = p_c(1 - p_c)^{n-1}$$

$$\text{prob (just one bit in error)} = np_c(1 - p_c)^{n-1}$$

The last expression follows, since the bit in error can be in any of the n possible slots in the block and all others are correct.

$$\text{prob(two or more errors)} = \left[1 - (1 - p_c)^n\right] - np_c(1 - p_c)^{n-1}$$

Here, the first term is the probability of some error; the second is the probability of one error. This last expression is the probability for a single-error-correcting (and only single) code. Sometimes, this is called the undetected incorrect block error probability, but the same terminology also applies to the case when the error pattern is itself a code word. Thus, some confusion is possible. Rewrite this as

$$\begin{aligned} \text{prob(two or more errors)} &= p_B(\text{undetected if Hamming}) \\ &= p_c^2\, n(n-1) \qquad p_c \text{ small} \\ &= (p_c n)^2 \qquad p_c \text{ small, } n \text{ large} \end{aligned}$$

The calculation for two errors is as follows: For a particular pattern of two errors, the probability of error is

$$p_c^2(1 - p_c)^{n-2}$$

That is, two in error and $n - 2$ correct. The total number of different patterns that contain two errors is

$$\binom{n}{2} = \frac{n!}{2!\,(n-2)!}$$

or the number of combinations formed by choosing from a pool of n distinct objects, grabbing them two at a time. The distinctness of n stems from each slot carrying a label. Then,

$$\text{prob(two errors)} = \binom{n}{2} p_c^2(1 - p_c)^{n-2}$$

Generalizing to ℓ errors gives

$$\text{prob}(\ell \text{ errors}) = \binom{n}{\ell} p_c^{\ell}(1 - p_c)^{n-\ell}$$

Note that

$$\sum_{\ell=0}^{n}\binom{n}{\ell}p_c^{\ell}(1-p_c)^{n-\ell}=(1-p_c)^n=1$$

Alternatively, the coefficient for two errors can be viewed as follows: Observe that

$$\binom{n}{2}=\frac{n!}{2!\,(n-2)!}$$

is also the number of permutations of n objects, two of which are alike (the errors) and $n-2$ of which are alike (the undamaged bits).

To end this section, refer to table 4.5, which catalogs the various expressions for p_u for many digital modulation schemes. These equations may be plotted when needed to perform a gain calculation.

TABLE 4.5—MODULATION ERROR RATES

$$\left[\text{Let } A=Q\left(\sqrt{\frac{2E_b}{N_o}}\right),\ B=Q\left(\sqrt{\frac{E_b}{N_o}}\right),\ C=\frac{1}{2}\exp\left(-\frac{E_b}{2N_o}\right);\ R=\text{bit rate.}\right]$$

Type of signaling	Required bandwidth	p_u
Baseband unipolar	$R/2$	B
Baseband polar	$R/2$	A
Bandpass binary phase shift keying (BPSK)	R	A } coherent detection; matched filter; hard decision
Bandpass quadraphase shift keying (QPSK, gray coded)	$R/2$	A }
Minimum shift keying (MSK)	$3R/2$	A coherent C noncoherent
On-off keying (OOK)	R	B coherent C noncoherent ($E_b/N_o > 1/4$)
Frequency shift keying (FSK)	$R+2\Delta f$ ($\Delta f = f_2 - f_1$)	B coherent C noncoherent
Differential phase shift keying (DPSK)	R	C noncoherent
Differentially encoded quadrature phase shift keying (DEQPSK)	--------	$2B$
M-ary	--------	$P_{\text{symbol}}=\frac{1}{M}\exp\left(-\frac{E_{\text{symbol}}}{N_o}\right)\sum_{j=2}^{M}(-1)^j\binom{M}{j}\exp\left(\frac{E_{\text{symbol}}}{jN_o}\right)$

4.8 Modification of Codes

Often, there is a need to modify a specific code to conform to system constraints. In other words, the values of n and k must be changed so that the code "fits" into the overall signaling scheme. The block length can be increased or decreased by changing the number of information and check bits. The block length can be kept constant while changing the number of code words. The changes that are possible will be illustrated for the Hamming (7,4) code. The basic (7,4) code is cyclic and the defining matrices are

$$\underset{\sim}{G} = \left[\begin{array}{cccc:cccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

$$\underset{\sim}{H} = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

For cyclic codes, another notation is used for the generator, namely the generator polynomial. This polynomial and what it means are discussed in chapter 5. For the above $\underset{\sim}{G}$, it is

$$g(x) = (1 + x + x^3)$$

The changes to the code are illustrated in figure 4.9, which is the example in Clark and Cain (1981).

A code may be extended by annexing additional parity checks. The added checks are carefully chosen to improve code weight structure (i.e., to modify the set $\{A_i\}$). For a single overall parity check addition, the check is equal to the remainder obtained by dividing the original code word by the polynomial $x + 1$. With the additional check the weight of all code words is an even number. Thus, the (7,4), $d = 3$ (the subscript min is dropped for convenience) Hamming code becomes an (8,4), $d = 4$ code. Because the new code is no longer cyclic, no generator polynomial is given. All codes with an odd minimum distance will have it increased by one by the addition of an overall parity check. A code may be punctured by deleting parity check digits. Puncturing is the inverse of extending. The deleted check is carefully chosen to keep the minimum distance the same as that before puncturing. A code may be expurgated by discarding some of the code words. For cyclic codes, this can be accomplished by multiplying $g(x)$ by $x + 1$. For the case $(x + 1)$, the new generator is $g(x)\ (x + 1)$, and the code words are just the even ones from the original code. A code may be augmented by

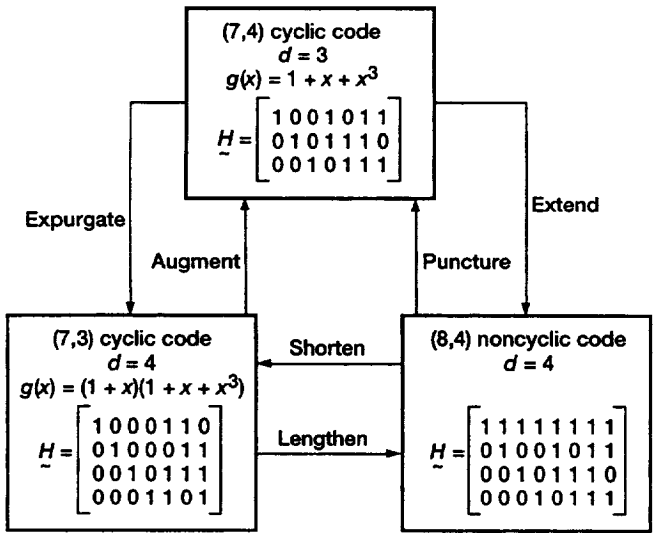


Figure 4.9.—Changes that specific (7,4) code can assume for specific applications.

adding new code words. Augmentation is the inverse of expurgation. Any cyclic code can be augmented by dividing out one of its factors. For example, if $g(x)$ has the factor $x + 1$, then $g(x)/(x + 1)$ generates another code with the same code word length. A code may be lengthened by adding additional information symbols. For a binary cyclic code that has a factor $x + 1$, the lengthening is done in two steps. First, augment by dividing by $x + 1$; then, extend by adding an overall parity check. A code may be shortened by deleting information bits. For cyclic codes, this can be done by making a segment of the information symbols identically zero at the beginning of each code word. A shortened cyclic code is no longer cyclic. In summary,

$$(n,k) \rightarrow (n+1,k) \qquad \text{extended by 1}$$
$$(n,k) \rightarrow (n-i,k-i) \qquad 0 \le i \le k \quad \text{shortened by } i$$

EXAMPLE 4.9

This example follows the discussion in Sweeney (1991). To shorten the code with matrices

$$\underset{\sim}{G} = \left[\begin{array}{cccc:ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right]$$

$$\underset{\sim}{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

first set one of the information bits permanently to zero and then remove that bit from the code. Let us set the third information bit to zero and thus remove the third row from $\underset{\sim}{G}$:

$$\underset{\sim}{G}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Next, to delete that bit, remove the third column:

$$\underset{\sim}{G}'' = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The parity check matrix changes as follows: The checks at the end of the deleted row in $\underset{\sim}{G}$ appear as the third column of $\underset{\sim}{H}$, so that the third column should be deleted:

$$\underset{\sim}{H}'' = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

which is a (6,3) code.

A second example of shortening uses the (15,11) code with H:

$$\underset{\sim}{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Removing all the odd-weight code words by deleting all the even-weight columns gives

$$\underset{\sim}{H}' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

which is a (8,4) code with $d = 4$.

▲

Chapter 5
Block Coding (Detailed)

5.1 Finite Fields

An (n,k) code comprises a finite number of code words, and if certain properties are incorporated, the code words can be treated as elements of a finite field. A finite field is the set $\{0,1,2,3,\ldots, p-1\}$, which is a field of order p (p is a prime number) under modulo-p addition and multiplication. It can be shown that the order of any finite field is a prime, and such fields are called prime or Galois fields. They are denoted as GF(p).

EXAMPLE 5.1

In modulo-p addition, take two elements in the field and add them (ordinary addition); the modulo-p sum is the remainder obtained by dividing the result by p. For $p = 5$, the table below summarizes the procedure.

$\oplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

In modulo-p multiplication, take two elements and multiply (ordinary); the remainder after division by p is the result. The table below summarizes the operation for $p = 5$.

$\cdot$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

▲

It is possible to extend the field GF(p) to a field of p^m (where m is a positive integer) elements, called an extension field of GF(p), denoted by GF(p^m).

EXAMPLE 5.2

GF(2) is the set $\{0,1\}$ with modulo-2 addition and multiplication

$\oplus$	0	1
0	0	1
1	1	0

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

▲

From here on, only a + is used for modulo-2 addition, for convenience.

5.1.1 Properties of GF(2^m)

The notation GF(q) is used often; here, $q = 2$ (in general, $q = 2^m$). A polynomial $f(x)$ with coefficients from GF(2) is

$$f(x) = f_0 + f_1 x + f_2 x^2 + \ldots + f_n x^n$$

where $f_i = 0$ or 1 is a polynomial over GF(2). There are 2^n polynomials of degree n. Division of polynomials is crucial. Let

$$f(x) = 1 + x + x^4 + x^5 + x^6$$
$$g(x) = 1 + x + x^3$$

Then,

$f(x)/g(x)$:

$$\begin{array}{r} x^3 + x^2 \leftarrow q(x) \\ x^3 + x + 1 \;\big|\; x^6 + x^5 + x^4 + x + 1 \\ \underline{x^6 \quad + x^4 + x^3} \\ x^5 + x^3 + x + 1 \\ \underline{x^5 + x^3 + x^2} \\ x^2 + x + 1 \leftarrow r(x) \end{array}$$

or

$$f(x) = q(x)g(x) + r(x)$$

where $q(x)$ is the quotient and $r(x)$ is the remainder. When $r(x) = 0$, f is divisible by g and g is a factor of f. If $f(x)$ has an even number of terms, it is divisible by $x + 1$. A root of $f(x)$, x_r, means $f(x_r) = 0$. A polynomial $p(x)$ over GF(2) of degree m is said to be irreducible over GF(2) if $p(x)$ is not divisible by any polynomial over GF(2) of degree less than m but greater than zero. Any irreducible polynomial over GF(2) of degree m divides $x^{2^m-1} + 1$.

EXAMPLE 5.3

Note that $p(x) = x^3 + x + 1$ divides $x^{2^3-1} + 1 = x^7 + 1$, so that $p(x)$ is irreducible.

▲

An irreducible polynomial $p(x)$ of degree m is primitive if the smallest positive integer n for which $p(x)$ divides $x^n + 1$ is $n = 2^m - 1$. A list of primitive polynomials is given in table 5.1. For each degree m, only a polynomial with the fewest number of terms is listed; others exist but are not given.

TABLE 5.1. — PRIMITIVE POLYNOMIALS

m	Polynomial
3	$1+x+x^3$
4	$1+x+x^4$
5	$1+x+x^5$
6	$1+x+x^6$
7	$1+x^3+x^7$
8	$1+x^2+x^3+x^4+x^8$
9	$1+x^4+x^9$
10	$1+x^3+x^{10}$
11	$1+x^2+x^{11}$
12	$1+x+x^4+x^6+x^{12}$
13	$1+x+x^3+x^4+x^{13}$

A useful property of polynomials is

$$[f(x)]^{2^\ell} = f\left(x^{2^\ell}\right)$$

5.1.2 Construction of GF(2^m)

To construct a field, first introduce a symbol α and then construct the set

$$F = \left\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^j, \ldots\right\} \qquad \alpha^0 \triangleq 1$$

Because the set is infinite, truncate it in the following way: Since

$$x^{2^m-1} + 1 = q(x)p(x) \qquad p(x) \text{ primitive}$$

replace x by α

$$\alpha^{2^m-1} + 1 = q(\alpha)p(\alpha)$$

Set $p(\alpha) = 0$; then,

$$\alpha^{2^m-1} + 1 = 0$$

or

$$\alpha^{2^m-1} = 1$$

which truncates the set to

$$F = \left\{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\right\}$$

EXAMPLE 5.4

Construct the field GF(2^4) by using $p(x) = 1 + x + x^4$. Note that $p(x)$ is given in table 5.1. Set $p(\alpha) = 0$:

$$1+\alpha+\alpha^4 = 0$$

Then,

$$\alpha^4 = 1+\alpha$$

This last identity is used repeatedly to represent the elements of this field. For example,

$$\alpha^5 = \alpha\alpha^4 = \alpha(1+\alpha) = \alpha+\alpha^2$$
$$\alpha^6 = \alpha\alpha^5 = \alpha\left(\alpha+\alpha^2\right) = \alpha^2+\alpha^3$$
$$\alpha^7 = \alpha\alpha^6 = \alpha\left(\alpha^2+\alpha^3\right) = \alpha^3+\alpha^4 = \alpha^3+1+\alpha = 1+\alpha+\alpha^3$$

etc. Note that $\alpha^{15} = 1$. Three representations of the field are given in table 5.2.

TABLE 5.2. — THREE REPRESENTATIONS FOR ELEMENTS OF GF(2^4) GENERATED BY $p(x) = 1+x+x^4$

Power					4-tuple
0	0				(0000)
1	1				(1000)
α		α			(0100)
α^2			α^2		(0010)
α^3				α^3	(0001)
α^4	$1+\alpha$				(1100)
α^5		$\alpha+\alpha^2$			(0110)
α^6			$\alpha^2 + \alpha^3$		(0011)
α^7	$1+\alpha$			$+\alpha^3$	(1101)
α^8	$1 + \alpha^2$				(1010)
α^9		α		$+\alpha^3$	(0101)
α^{10}	$1+$	α	$+ \alpha^2$		(1110)
α^{11}		α	$+ \alpha^2$	$+\alpha^3$	(0111)
α^{12}	$1 +$	α	$+ \alpha^2$	$+\alpha^3$	(1111)
α^{13}	1		α^2	$+\alpha^3$	(1011)
α^{14}	1			$+\alpha^3$	(1001)

Observe that the "elements" of the field are 4-tuples formed from ones and zeroes. Each element has three representations, and each is used in different steps in subsequent discussions. A general element is given the symbol β. For example,

$$\beta \leftrightarrow \alpha^{12} \leftrightarrow 1+\alpha+\alpha^2+\alpha^3 \leftrightarrow (1\,1\,1\,1)$$

Let β be a root of a polynomial of degree less than m in GF(2^m). Let $\phi(x)$ be the smallest degree polynomial such that $\phi(\beta) = 0$. Then, $\phi(x)$ (it is unique) is the minimal polynomial of β. Minimal polynomials derived from the GF(2^4) field are given in table 5.3.

TABLE 5.3. — MINIMAL POLYNOMIALS OF ELEMENTS IN GF(2^4)

[Generated by $p(x) = 1 + x + x^4$.]

Conjugate roots	Minimal polynomial
0	x
$\alpha\ \alpha^2\ \alpha^4\ \alpha^8$	$x+1$
$\alpha^3\ \alpha^6\ \alpha^9\ \alpha^{12}$	x^4+x+1
$\alpha^5\ \alpha^{10}$	$x^4+x^3+x^2+x+1$
$\alpha^7\ \alpha^{11}\ \alpha^{13}\ \alpha^{14}$	x^4+x^3+1

▲

5.2 Encoding and Decoding

The simplest encoding/decoding scheme is best explained by a specific example; the one chosen is the example in Lin and Costello (1983).

EXAMPLE 5.5

For a (7,4) Hamming code, choose the generator as

$$\underset{\sim}{G} = \left[\underset{\sim}{P} \vdots \underset{\sim}{I}_k\right] = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \underline{v} = \underline{u}\,\underset{\sim}{G}$$

Here, the parity check digits are at the beginning of the code word. The circuit to encode a message vector $\underline{u} = (u_0, u_1, u_2, u_3)$ is given in figure 5.1. The message register is filled by clocking in u_3, u_2, u_1, u_0 and simultaneously passing them to the output. Next, the modulo-2 adders form the outputs $\underline{v} = (v_0, v_1, v_2)$ in the parity register. The switch at the right moves to extract $\underline{v}$. The parity check matrix is

$$\underset{\sim}{H}^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The coset leaders and corresponding syndromes are

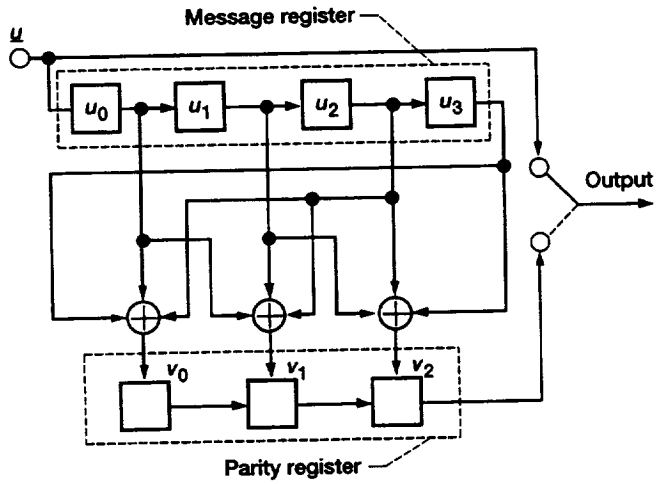


Figure 5.1.—Encoder for (7,4) Hamming code.

Syndrome	Coset leader
1 0 0	1 0 0 0 0 0 0
0 1 0	0 1 0 0 0 0 0
0 0 1	0 0 1 0 0 0 0
1 1 0	0 0 0 1 0 0 0
0 1 1	0 0 0 0 1 0 0
1 1 1	0 0 0 0 0 1 0
1 0 1	0 0 0 0 0 0 1

The circuit to perform the correction is shown in figure 5.2. The received bits are entered as r_0, ..., r_6. The modulo-2 adders form the syndrome (s_0, s_1, s_2). A combinatorial logic network calculates the appropriate error pattern, and the last row of adders serves to add e_i to r_i and correct the word, which is placed in the "corrected output" buffer. If only a single error is present, only one e_i is present, and the corresponding r_i is corrected.

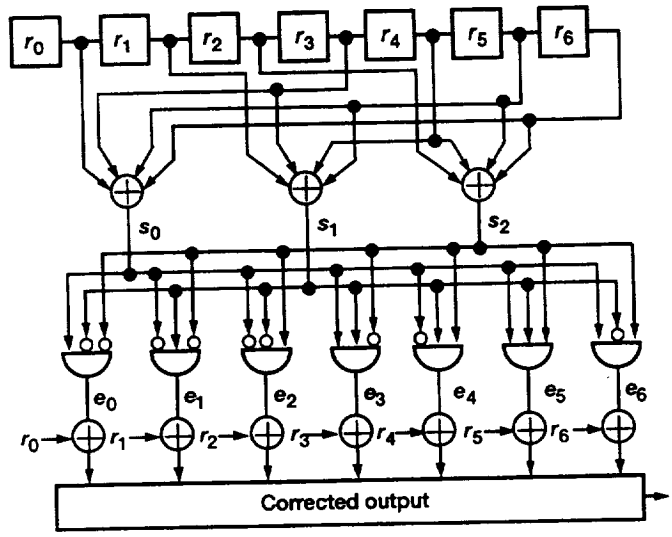


Figure 5.2.—Decoder for (7,4) Hamming code.

5.2.1 Cyclic Codes and Encoders

Many codes are cyclic (an end-around shift of a code word is also a code word; i.e., if 1001011 is a code word, then 1100101 is the first end-around shift and is also a code word). Such codes can be represented by a generator polynomial $g(x)$. The recipe for an (n,k) code is

$$\underline{v}(x) = \underline{u}(x)g(x) \tag{5.1}$$

where

$$\underline{v}(x) = v_0 + v_1x + v_2x^2 + \ldots + v_{k-1}x^{k-1}$$

$$\underline{u}(x) = u_0 + u_1x + u_2x^2 + \ldots + u_{k-1}x^{k-1}$$

$$g(x) = 1 + g_1x + g_2x^2 + \ldots + g_{n-k-1}x^{n-k-1} + x^{n-k}$$

A property of $g(x)$ is that $g(x)$ for an (n,k) code divides $x^n + 1$; that is,

$$\left(x^n + 1\right) = g(x)h(x)$$

or

$$x^7 + 1 = \left(1 + x + x^3\right)\left(1 + x + x^2 + x^4\right)$$

This factoring is not obvious and must be found by table look-up in general. Further factoring is also possible in this case:

$$x^7 + 1 = (1 + x)\left(1 + x + x^3\right)\left(1 + x^2 + x^3\right)$$

where there are two $g(x)$ factors, both of degree 3. Therefore, each generates a (7,4) code. Observe that the code word $\underline{v}(x)$ in equation (5.1) is not in systematic form but can be put in that form with the following procedure:

1. Premultiply the message $\underline{u}(x)$ by x^{n-k}.
2. Divide $x^{n-k}\underline{u}(x)$ by $g(x)$ to obtain a remainder $\underline{b}(x)$.
3. Form $\underline{v}(x) = x^{n-k}\underline{u}(x) + \underline{b}(x)$.

Example 5.6

Encode $\underline{u}(x) = 1101 \rightarrow 1 + x + x^3$ with $g(x) = 1 + x + x^3$ in a (7,4) code. Form

$$x^3\left(1 + x + x^3\right) = x^3 + x^4 + x^6$$

Form

$$\frac{x^3 + x^4 + x^6}{x^3 + x + 1}$$

The quotient is $q(x) = x^3$ with remainder $\underline{b}(x) = 0$. Then,

$$v(x) = x^3 + x^4 + x^6 = 0001101$$

Note that the last four digits (1101) are the message and the first three (000) are the parity check digits. ▲

EXAMPLE 5.7

This problem shows the correspondence between the generator matrix and the generator polynomial for cyclic codes. Consider a (7,4) cyclic code generated by

$$g(x) = 1 + x + x^3$$

Determine its generator matrix G in systematic form. The procedure is to divide x^{n-k+i} by $g(x)$ for $i = 0,1,2,\ldots,k-1$. For $i = 0$, $x^{n-k} = x^3$.

$$\begin{array}{lll} \underline{x^3 + x + 1} & x^3 & \underline{\vert 1 \leftarrow q(x)} \\ & \underline{x^3 + x + 1} & \\ & x + 1 & \leftarrow r(x) \end{array}$$

so that $x^3 = q(x)g(x) + r(x) \Rightarrow x^3 = 1 \times g(x) + (1 + x)$. For $i = 1$,

$$x^{n-k+1} = x^4$$

After division,

$$x^4 = xg(x) + \left(x + x^2\right)$$

Continuing,

$$x^5 = \left(1 + x^2\right)g(x) + \left(x^2 + x + 1\right)$$
$$x^6 = \left(x^3 + x + 1\right)g(x) + \left(1 + x^2\right)$$

Rearrange the above to obtain four code polynomials

$$v_0(x) = 1 + x + x^3$$
$$v_1(x) = x + x^2 + x^4$$
$$v_2(x) = 1 + x + x^2 + x^5$$
$$v_3(x) = 1 + x^2 + x^6$$

which are found by adding together the single term $x^{(\bullet)}$ on the left with the remainder. That is, x^3 is added to $(1 + x)$ to form $v_0(x)$. Use these as rows of a (7×4) matrix; thus,

$$\underset{\sim}{G} = \left[\begin{array}{cccc:cccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}\right] = \left[\underset{\sim}{P} \vdots \underset{\sim}{I}_{n-k}\right]$$

Note that $g(x)$ can be read from $\underset{\sim}{G}$ by observing the first row of $\underset{\sim}{G}$. The row 1101000 corresponds to x^0, x^1, and x^3 so that

$$g(x) = x^0 + x^1 + x^3 = 1 + x + x^3$$

▲

5.2.2 Encoder Circuits Using Feedback

Most encoders use feedback shift registers. Recall that the code word can be found in two ways,

$$v(x) = u(x)g(x)$$

or

$$v(x) = x^{n-k}u(x) + b(x)$$

where the generator polynomial has the form

$$g(x) = 1 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1}x^{n-k-1} + x^{n-k}$$

Figure 5.3 gives the basic encoder scheme. The multiplication of the message vector by x^{n-k} basically adds zeros onto the left of the message vector, which gives enough bits for n complete shifts of the register. The operation proceeds with switch I closed and switch II down. The machine shifts k times, loading the message into the registers. At this time, the message vector has moved out and comprises $\underset{\sim}{v}(x)$, and at the same time the parity checks have been formed and reside in the registers. Next, switch I is opened, switch II moves up, and the remaining $n - k$ shifts move the parity checks into $\underset{\sim}{v}(x)$. During these shifts the leading zeros appended to $\underset{\sim}{u}(x)$ earlier are shifted into the register, clearing it.

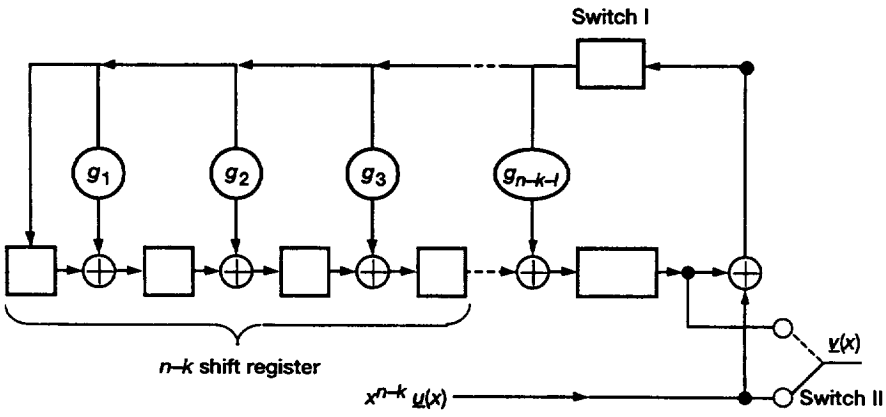


Figure 5.3.—Basic encoder for cyclic codes.

EXAMPLE 5.8

Encode the message vector $\underline{u}(x) = 1011$ into a (7,4) code by using the generator polynomial $g(x) = 1 + x + x^3$:

$$\underline{u}(x) = 1011 = 1 + x^2 + x^3$$

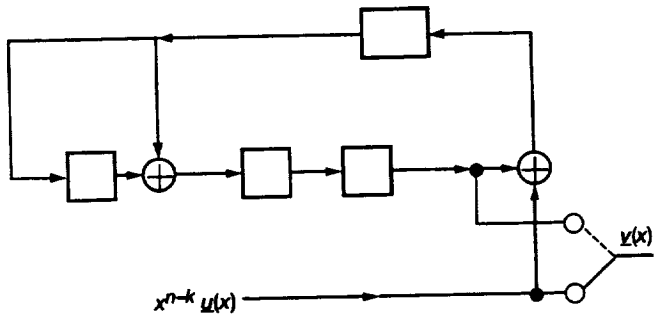
$$x^{n-k}\underline{u}(x) = x^3 + x^5 + x^6$$

$$x^{n-k}\underline{u}(x) = \underline{v}(x) + b(x) = \underline{u}(x)g(x) + b(x)$$

$$\therefore b(x) = \text{remainder mod } g(x) \text{ of } x^{n-k}\underline{u}(x) = x^3 + x^5 + x^6$$

For the $(n - k)$, three-stage encoding shift register shown in figure 5.4, the steps are as shown. After the fourth shift, switch I is opened, switch II is moved up, and the parity bits contained in the register are shifted to the output. The output code vector is $\underline{v} = 1001011$, or in polynomial form, $\underline{v}(x) = 1 + x^3 + x^5 + x^6$. ▲

Next, consider the syndrome calculation using a shift register. Recall that the syndrome was calculated by using modulo-2 adders in figure 5.2; a different method using registers is given in figure 5.5. Here, the received vector is shifted in; and after it has been loaded, the syndrome occupies the register. The lower portion gives the syndrome calculator for the (7,4) code used in previous examples. Note that the generator matrix used for the case in figure 5.2 yields the same generator polynomial as shown in figure 5.5; thus, different implementations of the same decoding scheme can be compared.



Input queue	Shift number	Register contents	Output
0001011	0	000	–
000101	1	110	1
00010	2	101	1
0001	3	100	0
000	4	100	1
00	5	010	0
0	6	001	0
–	7	000	1

Figure 5.4.—Cyclic encoder steps while encoding message vector $\underline{u}(x) = 1011$.

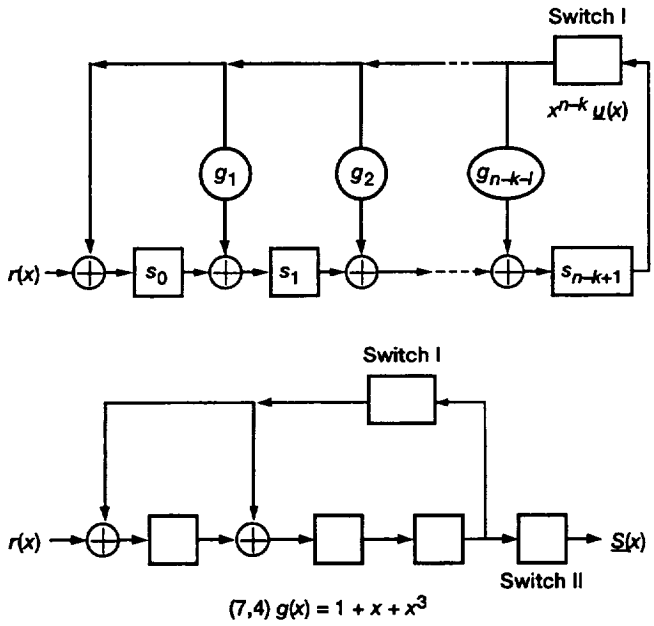


Figure 5.5.—Decoder using shift register. (a) General syndrome calculator. (b) Calculator for specific (7,4) code given by generator $g(x) = 1 + x + x^3$.

5.3 Decoders

In syndrome decoding for general block codes and for the special case of cyclic codes, the difficult step of determining the error pattern e commences once the syndrome is known. Many algorithms have been developed for this stage of decoding; and their evolution and implementation form a large body of material in the journals. Each has its good/bad, cost/complexity tradeoffs, etc. According to Clark and Cain (1981) decoders are algebraic or nonalgebraic. Algebraic decoders solve simultaneous equations to find e; also, finite-field Fourier transforms are sometimes used. Only hard-decision decoders are discussed here, since they find the most use. Soft-decision decoders (nonalgebraic, such as Massey's APP (a posteriori probability), Hartmann-Rudolph, Weldon, partial syndrome, etc.) are omitted. The nonalgebraic decoders use properties of codes to find e, and in many instances a code and decoder are "made for each other." Some schemes discussed here are also used with convolutional codes, as covered in chapters 6 and 7.

The delineation of decoding algorithms is not crisp. For example, some authors use Meggit decoders as a classification with feedback decoding being a subset. Others, however, include Meggit decoders as a special form of feedback decoding. Following the lead of both Clark and Cain (1981) and of Lin and Costello (1983), the discussion of decoders begins with cyclic codes.

5.3.1 Meggit Decoders

The algorithm for Meggit decoders depends on the following properties of cyclic codes:

1. There is a unique one-to-one correspondence between each member in the set of all correctable errors and each member in the set of all syndromes.
2. If the error pattern is shifted cyclically one place to the right, the new syndrome is obtained by advancing the feedback shift register containing $S(x)$ one shift to the right.

These properties imply that the set of error patterns can be divided into equivalence classes, where each class contains all cyclic shifts of a particular pattern. For a cyclic code of block length n, each class can be identified by advancing the syndrome register no more than n times and testing for a specific pattern after each shift. Figure 5.6 shows a basic form for a Meggit decoder that uses feedback (some forms do not use feedback). The

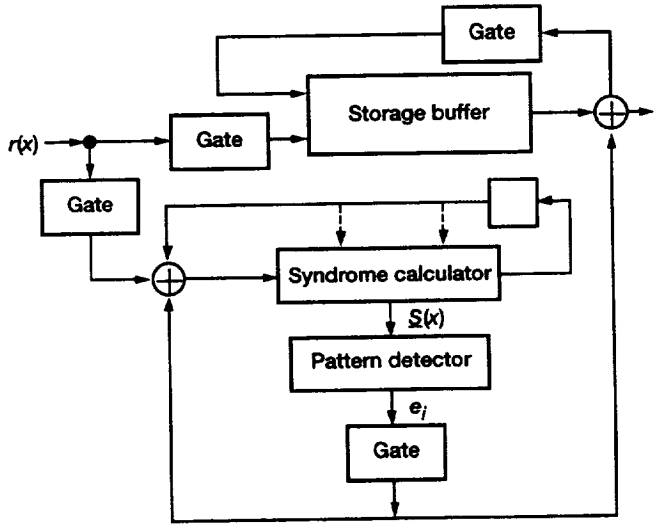


Figure 5.6.—Feedback Meggitt decoder.

received vector is shifted into the storage buffer and syndrome calculator simultaneously. At the completion of the load step, a syndrome resides in the syndrome calculator. Next, the pattern detector tests the syndrome to see if it is one of the correctable error patterns with an error at the highest order position. If a correctable pattern is detected, a one appears at the pattern detector's output; the received symbol in the rightmost stage of the storage buffer is assumed to be in error and is corrected by adding the one to it. If a zero appears at the pattern detector's output, the received symbol at the rightmost stage is assumed to be correct, and no correction is needed (adding a zero does not change it). As the first received bit is read from the storage buffer (corrected if needed), the syndrome calculator is shifted once. The output of the pattern detector is also fed back to the syndrome calculator to modify the syndrome. This effectively "removes" the effect of this error on the syndrome and results in a new syndrome corresponding to the altered received vector shifted one place to the right. This process repeats, with each received symbol being corrected sequentially. This basic idea has many variations and many differences in the number of times the received vector is shifted versus the number of times the syndrome calculator can change. Also, the phase of shifts can vary. In this manner, bursts of errors are handled as well as shortened cyclic codes. The Meggit decoder for the (7,4) code is shown in figure 5.7.

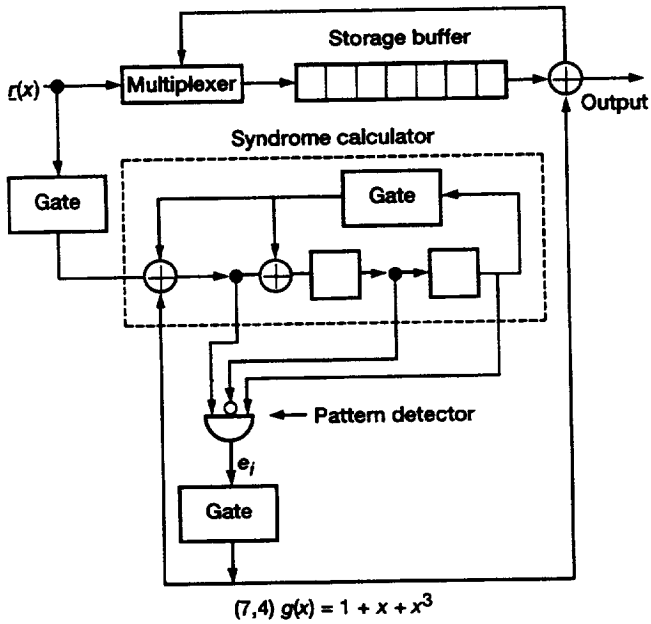


Figure 5.7.—Meggitt decoder for specific (7,4) cyclic code.

5.3.2 Error-Trapping Decoders

Error-trapping decoders are a subset of Meggit decoders, and several forms and enhancements on the basic concept exist (e.g., Kasami's method). They work because of the following property: If errors are confined to the $n-k$ high-order positions of the received polynomial $\underline{r}(x)$, the error pattern $\underline{e}(x)$ is identical to $x^k\underline{S}^{(n-k)}(x)$, where $\underline{S}^{(n-k)}(x)$ is the syndrome of $\underline{r}^{(n-k)}(x)$, the $(n-k)$th cyclic shift of $\underline{r}(x)$. When this event occurs, it computes $\underline{S}^{(n-k)}(x)$ and adds $x^k\underline{S}^{(n-k)}(x)$ to $\underline{r}(x)$. In other words, the scheme searches segments of $\underline{r}(x)$ in hopes of finding a segment that contains all the errors (error trapping). If the number of errors in $\underline{r}(x)$ is t or less and if they are confined to $n-k$ consecutive positions, the errors are trapped in the syndrome calculator only when the weight of the syndrome in the calculator is t or less. The weight of $\underline{S}(x)$ is tested by a $(n-k)$-input threshold gate whose output is one when t or fewer of its inputs are one. Its inputs come from the syndrome calculator.

5.3.3 Information Set Decoders

Information set decoders work on a large class of codes (hard or soft decision). In an (n,k) group code, an information set is defined to be any set of k positions in the code word that can be specified independently. The remaining $n-k$ positions are referred to as the "parity set." If the generator matrix for the code can be written in echelon canonical form, the first k positions form an information set. Any other set of positions can form an information set if it is possible to make the corresponding columns of the generator matrix into unit weight columns through elementary row operations. For example, consider the (7,4) Hamming code whose generator is

$$\underset{\sim}{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

By adding the first row to the third and fourth rows, this matrix can be transformed to

$$\underset{\sim}{G}' = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

This has the effect of "interchanging" columns 1 and 5. Positions 2, 3, 4, and 5 now form an information set (have only a single one in their columns). This example shows that a necessary and sufficient condition for being able to "interchange" any arbitrary column with one of the unit weight columns is that they both have a one in the same row. By this criterion, column 1 can be interchanged with column 5 or 6 but not with column 7, column 2 can be interchanged with column 6 or 7 but not with column 5, etc. Since the symbols contained in the information set can be specified independently, they uniquely define a code word. If there are no errors in these positions, the remaining symbols in the transmitted code word can be reconstructed. This property provides the basis for all information set algorithms. A general algorithm is as follows:

1. Select several different information sets according to some rule.
2. Construct a code word for each set by assuming that the symbols in the information set are correct.
3. Compare each hypothesized code word with the actual received sequence and select the code word that is closest (smallest metric, closest in Hamming distance).

5.3.4 Threshold Decoders

Threshold decoders are similar to Meggit decoders but need certain code structures. Majority-logic decoding is a form of threshold decoding for hard-decision cases and has been used often. (It is seldom used now.)

Threshold decoding uses circuitry to work on the syndrome to produce a likely estimate of some selected error digit. The main point is that any syndrome digit, being a linear combination of error digits, represents a known sum of error digits. Further, any linear combination of syndrome digits is thus also a known sum of error digits. Hence, all 2^{n-k} such possible combinations of syndrome digits are all of the known sums of error digits available at the receiver. Such a sum is called a parity check equation and denoted by A_i (the ith parity check equation). Thus, each A_i is a syndrome digit or a known sum of syndrome digits. A parity check equation A_i is said to check an error digit e_j if e_j appears in A_i. A set $\{A_i\}$ of parity check equations is said to be orthogonal on e_m if each A_i checks e_m but no other error digits are checked by more than one A_i. For example, the following set is orthogonal on e_3 (all additions are modulo-2):

$$\begin{aligned} A_1 &= e_1 \oplus e_2 \oplus e_3 \\ A_2 &= e_3 \oplus e_4 \oplus e_5 \\ A_3 &= e_3 \oplus e_6 \oplus e_7 \end{aligned}$$

Although e_3 appears in each A_i, each of the other error digits appears in only a single A_i. Majority-logic decoding is a technique of solving for a specific error digit given an orthogonal set of parity check equations for that error digit and is characterized by the following: Given a set of $J = 2t + S$ parity checks orthogonal on e_m, any pattern of t or fewer errors in the digits checked by the set $\{A_i\}$ will cause no decoding error (i.e., is correctable) and patterns of $t + 1,\ldots,t + s$ errors are detectable if e_m is decoded by the rule

$$\begin{aligned} \hat{e}_m = 1 &\text{ if more than } (J+S)/2 \text{ of the } A_i \text{ have value} = 1 \\ \hat{e}_m = 0 &\text{ if } (J-S)/2 \text{ or fewer have values} = 1 \\ &\text{error detection only if otherwise} \end{aligned}$$

Here, $\hat{e}_m$ denotes the estimate of e_m. Thus, $J + 1$ corresponds to the effective minimum distance for majority-logic decoding. Further, it can be shown that the code must have a minimum distance of at least $J + 1$. A code is completely orthogonalized if $d_{\min} - 1$ orthogonal parity check equations can be found for each error digit.

5.3.5 Algebraic Decoders

Algebraic decoders are used on "algebraically defined" codes, such as BCH codes. The algebraic structure imposed on the codes permits computationally efficient decoding algorithms. First, the underlying structure of these BCH codes must be studied. A primitive BCH code has

$$n = 2^m - 1, \qquad n - k \le mt, \qquad t < 2^{m-1} \qquad m \ge 3$$
$$d_{\min} \ge 2t + 1$$

The generator polynomial is of the form

$$g(x) = m_1(x) \cdot m_3(x) \cdot m_5(x) \cdot \cdots \cdot m_{2t-1}(x)$$

(i.e., t factors).

Write the parity check matrix in the form (for $n = 15$)

$$\underset{\sim}{H} = \begin{bmatrix} a_1 & a_2 \cdots a_{15} \\ a_1^3 & a_2^3 \cdots a_{15}^3 \end{bmatrix} \qquad n, k = (15,11)$$

The $\{a_i\}$, $i = 1,...,15$ are distinct nonzero elements of GF(2^4). If errors occur in positions i and j of the received word, the syndrome

$$\underline{S} = \underline{e}\,\underset{\sim}{H} = (s_1, s_2, s_3, s_4)$$

produces two equations in two unknowns

$$a_i + a_j = s_1$$

and

$$a_i^3 + a_j^3 = s_3$$

If these equations could be solved for a_i and a_j, the error locations i and j would be known. Error correction would then consist of inverting the received symbols in these locations. Because the equations are nonlinear, any method of solving them directly is not obvious. However, it is possible to begin by eliminating one of the variables. Thus, solving the first equation for a_i and substituting into the second equation yields

$$a_j^2 + s_1 a_j + s_1^2 + \frac{s_3}{s_1} = 0$$

Had the first equation been solved for a_j, the resulting equation would be the same, with a_i replacing a_j. Consequently, both a_i and a_j are solutions (or roots) of the same polynomial:

$$\sigma = z^2 + s_1 z + s_1^2 + \frac{s_3}{s_1} = 0$$

This polynomial is called an error locator polynomial. One method of finding its roots is simple trial and error. Substituting each of the nonzero elements from GF(2^4) into this equation guarantees that the location of both errors will be found. The complete recipe for decoding is as follows:

1. From $r(x)$ calculate remainders modulo m_1, m_3, and m_5; these result in partial syndromes s_i. For a t-error-correcting code, there are $2t$ such m-bit syndromes.
2. From the s_i, find the coefficients for an e-degree error locator polynomial ($e \le t$), where e is the number of errors. The technique for doing this is called the Berlekamp iterative algorithm. This polynomial $\sigma(x)$ has the significance that its roots give the location of the errors in the block. The roots are the error location numbers α^i, $i = 0,...,14$ (if $n = 15$).
3. Find the roots, generally by using the Chien search, which involves checking each of the n code symbol locations to see if that location corresponds to a root.
4. Correct the errors. For binary codes, this entails just complementing the erroneous bit. For Reed-Solomon codes (nonbinary), a formula for correcting the symbol exists.

5.4 Miscellaneous Block Code Results

5.4.1 Reed-Solomon Codes

Reed-Solomon (R–S) codes use the following procedure:

1. Choose nonbinary symbols from GF(2^m). Each symbol has m bits (i.e., let $m = 8$, a symbol is (10101010) or eight bits).
2. Define $q = 2^m$. Then,

$$N = q - 1 \quad \text{symbols / word}$$
$$N - K = 2t \quad \text{to correct } t \text{ symbols}$$
$$d_{\min} = 2t + 1$$

since $d_{\min} = N - K + 1$, it is maximum-distance separable (largest possible $d_{\min}$). On a bit basis,

$$N \rightarrow n = m\left(2^m - 1\right) \quad \text{bits}$$

$$N - K \rightarrow m(N - K) \quad \text{check bits}$$

which is cyclic (subset of BCH) and good for bursts.

3. Use Berlekamp-Massey or Euclidean decoders, which can correct

$$\begin{array}{ll} 1 & \text{burst of length } b_1(t-1)m+1 \text{ bits} \\ 2 & \text{bursts of length } b_2(t-3)m+3 \text{ bits} \\ \vdots & \vdots \\ i & \text{bursts of length } b_i(t-2i+1)m+2i-1 \text{ bits} \end{array}$$

4. Let b be the maximum correctable burst length (guaranteed), and let ℓ be the length of the shortest burst in a code word (1xxxxx1):

$$b \le \frac{\ell - 1}{2}$$

For example, for (N,K) = (15,9)

$$t = 3, \quad m = 4, \quad d = 7$$

If the code is viewed as a binary (60,36), R–S codes can correct any burst of three four-bit symbols, where α is in GF(2^m):

$$g(x) = (x+\alpha)\left(x+\alpha^2\right)\left(x+\alpha^3\right)\cdots\left(x+\alpha^{d-1}\right)$$

$$g(x) = (x+\alpha)\cdots\left(x+\alpha^6\right) \qquad \text{for above example}$$

$$\therefore g(x) = x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4 x^3 + \alpha^6 x^2 + \alpha^9 x + \alpha^6$$

5. To calculate the error probability, let

$$R = \frac{K}{N}, \qquad (E_b)_c = R \cdot (E_b)_u$$

From p_c, determine the channel symbol error rate p_{symbol}

$$p_{\text{symbol}} = 1 - (1 - p_c)^m$$

Let $p_u(E)$ be the probability of undetected error (symbol error):

$$p_u(E) = \sum_{i=1}^{N} A_i p_{\text{symbol}}^i \left(1 - p_{\text{symbol}}\right)^{N-i}$$

$$A_0 = 1, \quad A_j = 0 \qquad 1 \le j \le N - K$$

$$A_j = \binom{N}{j} \sum_{h=0}^{j-1-(N-K)} (-1)^h \binom{j}{h} \left[q^{j-h-(N-K)} - 1\right] \qquad \text{for } (N-K)+1 \le j \le N$$

The probability of decoding error (symbol error) is

$$p(E) \le \sum_{i=t+1}^{N} \binom{N}{i} p_{\text{symbol}}^i \left(1 - p_{\text{symbol}}\right)^{N-i}$$

The total symbol error probability is

$$p_{\text{tot}} = p_u(E) + p(E) \simeq p(E) \doteq \frac{1}{N} \sum_{j=t+1}^{N} j \binom{N}{j} p_{\text{symbol}}^j \left(1 - p_{\text{symbol}}\right)^{N-j}$$

Now, to find bit error rate,

$$\frac{(p_b)_s}{(p_E)_{\text{tot}}} = \frac{1}{2}\frac{M}{M-1} \text{ for M-ary multiple-frequency shift keying(MFSK)}$$

or

$$(p_b)_s \doteq \frac{1.5t+1}{N} p_{\text{tot}}$$

or

$$(p_b)_s \le \frac{1}{2}\frac{M}{M-1}\sum_{j=t+1}^{N}\frac{j+t}{N}\binom{N}{j}p_{\text{symbol}}^{i}\left(1-p_{\text{symbol}}\right)^{N-j}$$

5.4.2 Burst-Error Correcting Codes

Burst-error-correcting codes include the following types:

1. Burst detecting and efficiency correcting,

$$\eta = \frac{2b}{n-k}$$

2. Fire codes, $g(x) = \left(x^c - 1\right)p(x)$, where p has degree m.

$$c \ge d_{\min} + b - 1$$

$$m > b$$

where b is burst length and the code corrects all bursts $\le b$ and detects all bursts $\le d_{\min}$ bits long. In general,

$$b \le \frac{n-k}{2}$$

$$n-k > b-1+\log_2 n$$

$$n-k > 2(b-1)+\log_2(n-2b+2)$$

Detecting a burst of length b requires b parity bits, and correcting a burst of length b requires $2b$ parity bits.

A common application of cyclic codes is for error detection. Such a code is called a cyclic redundancy check (CRC) code. Since virtually all error-detecting codes in practice are of the CRC type, only this class of code is discussed. A CRC error burst of length b in the n-bit received code word is defined as a contiguous sequence or an end-around-shifted version of a contiguous sequence of b bits, in which the first and last bits and any number of intermediate bits are received in error. The binary (n,k) CRC codes can detect the following n-bit channel-error patterns:

1. All CRC error bursts of length $n - k$ or less
2. A fraction $1 - 2^{-(n-k-1)}$ of the CRC error burst of length $b = n - k + 1$
3. A fraction $1 - 2^{-(n-k)}$ of the CRC error bursts of length $b > n - k + 1$

4. All combinations of $d_{\min} - 1$ or fewer errors
5. All error patterns with an odd number of errors if the generator polynomial has an even number of nonzero coefficients

Usually, the basic cyclic codes used for error detection are selected to have a very large block length n. Then, this basic code, in a systematic form, is shortened and is no longer cyclic. All standard CRC codes use this approach, so that the same generator polynomial applies to all the block lengths of interest. Three standard CRC codes are commonly used:

1. CRC–12 code with $g(x) = 1 + x + x^2 + x^3 + x^{11} + x^{12}$

2. CRC–16 code with $g(x) = 1 + x^2 + x^{15} + x^{16}$

3. International Telegraph and Telephone Consultative Committee (CCITT) CRC code with

$$g(x) = 1 + x^5 + x^{12} + x^{16}$$

4. A more powerful code with

$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

has been proposed where extra detection capability is needed.

5.4.3 Golay Code

The weight enumerator for Golay code (23,12) is

$$A(z) = 1 + 253z^7 + 506z^8 + 1288z^{11} + 1288z^{12} + 506z^{15} + 253z^{16} + z^{23}$$

Code (23,12) has $d = 7$ and $t = 3$ and corrects up to three errors.

$$\left(x^{23} + 1\right) = (1 + x)g_1(x)g_2(x)$$

$$g_1(x) = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$$

$$g_2(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11} \triangleq m_{89}(x)$$

Recall that

$$2^k \le \frac{2^n}{1 + n + \binom{n}{2} + \cdots + \binom{n}{t}}$$

For $n = 23$, and $t = 3$,

$$\binom{23}{3} + \binom{23}{2} + \binom{23}{1} + 1 = 2048$$

but $2048 = 2^{11}$,

$$\therefore 2^k = \frac{2^{23}}{2^{11}}, \qquad \therefore k = 12$$

Thus, 2^{12} code words equals 4096 spheres of Hamming radius 3, closely packed. Each sphere contains 2^{11} vectors. There are $2^{n-k} = 2^{11}$ syndromes, which correspond one to one to all error patterns. Adding the overall check bit gives code (24,12) (then $r = 1/2$), which detects all patterns of up to four errors. The extended code (24,12) has $d_{\min} = 8$. Using the decoding table concept shows that exactly n patterns differ from the correct pattern in one position, $\binom{n}{2}$ patterns differ in two positions, etc. Since there are almost always some patterns left over (after assigning all those that differ in t or fewer places),

$$N_e \geq 1 + n + \binom{n}{2} + \cdots + \binom{n}{t}$$

where N_e is the number of n-tuples in a column. Since there are 2^n possible sequences, the number of code words N_c obeys

$$N_c \leq \frac{2^n}{1 + n + \binom{n}{2} + \cdots + \binom{n}{t}}$$

(sphere packing bound). For an (n,k) code, $N_c = 2^k$; thus,

$$2^k \leq \frac{2^n}{1 + n + \binom{n}{2} + \cdots + \binom{n}{t}}$$

Golay noted that $n = 23$, $k = 12$, and $t = 3$ provide the equality in the above—thus, the "perfect" (23,12) code.

5.4.4 Other Codes

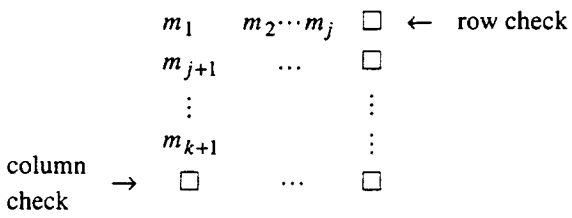
The following is some miscellaneous information about codes:

1. Hamming codes are single-error-correcting BCH (cyclic).
2. For $t = 1$ codes,

$$k \leq n - \log_2(n+1)$$

$$r = \frac{k}{n} \leq 1 - \frac{1}{n}\log_2(n+1), \qquad 2^n - 2^k(n+1)$$

3. A multidimensional code uses a matrix for a code word.
4. An alternative to the (n,k) notation uses $M(n,d)$, where d is $d_{\min}$.
5. A rectangular or product code produces checks on both the columns and rows of a matrix that is loaded with the message. That is,



6. Hardware cost is proportional to $(n)\cdot(t)$.
7. If searching for a code to apply to a system, see page 124 of Peterson and Weldon (1972) (i.e., given required n, k, and $d_{\min}$, is a code available?).

5.4.5 Examples

EXAMPLE 5.9

The probability of one code word being transformed to another code word is

$$\left(\frac{1}{2}\right)^{n-k}$$

▲

EXAMPLE 5.10

Reed-Muller codes are specified by $n = 2^m$.

$$k = 1 + \binom{m}{1} + \cdots + \binom{m}{r}, \qquad n - k = 1 + \binom{m}{1} + \cdots + \binom{m}{m-r-1}$$

$$d = 2^{m-r}$$

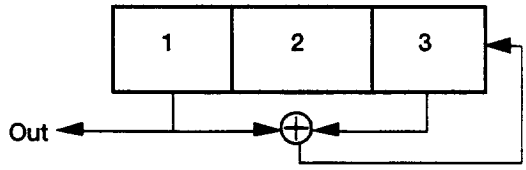
▲

EXAMPLE 5.11

Maximum-length shift register codes (MLSR) are defined by

$$(n, k) = \left(2^m - 1, m\right) \qquad m = 1, 2, 3, \cdots$$

They are duals of Hamming $(2^m - 1, 2^m - 1 - m)$ codes. All code words have same weight of 2^{m-1} (except the all-zero word). The distance is $d_{\min} = 2^{m-1}$. To encode, load the message and shift the register to the left $2^m - 1$ times.



▲

EXAMPLE 5.12

Soft-decision decoders use the Euclidean distance between the received vector and permissible code vectors. For example, suppose three successive waveform voltages from the demodulator are –0.1, 0.2, and 0.99 (a hard decision about zero would yield (011) as the word). Let each of these voltages be denoted by y_i, and assume that some predetermined voltage levels in the decoder have been assigned x_i. The Euclidean distance between signal levels is defined as

$$\sum_{i=1}^{n}(y_i - x_i)^2$$

In soft-decision decoding, this distance measure is used to find the closest code word. ▲

EXAMPLE 5.13

In general, $d_{\min} \le n - k + 1$ (Singleton bound). The equality implies a maximum-distance separable code; R–S codes are such codes. Some upper and lower bounds on $d_{\min}$ exist (fig. 5.8). Some formulas are

1. Gilbert-Varsharmov—For a q-ary code

$$\sum_{i=0}^{d-2}\binom{n}{i}(q-1)^i < q^{n-k} \le \sum_{i=0}^{d-1}\binom{n}{i}(q-1)^i$$

2. Plotkin

$$k \le n - 2d_{\min} + 2 + \log_2 d_{\min}$$

3. Griesmer—Let $\lceil d \rceil$ represent the integer that is not less than $d/2$.

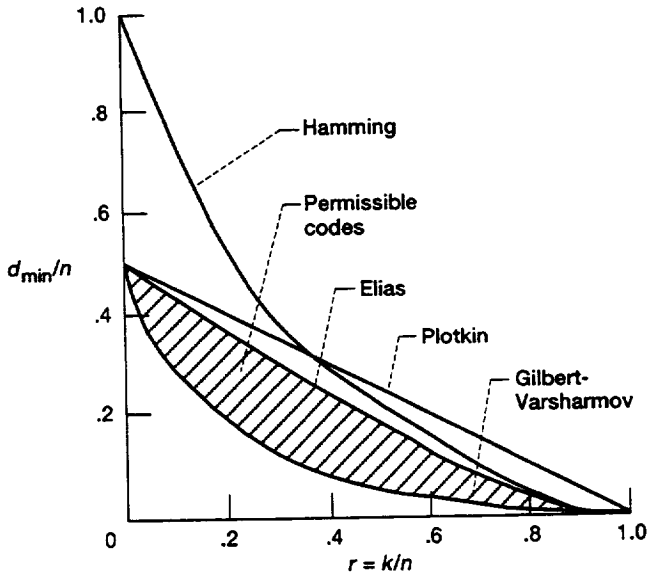


Figure 5.8.—Some classic upper and lower bounds on $d_{\min}$ for (n, k) block codes.

$$n \geq \sum_{i=0}^{k-1} \frac{\lceil d \rceil}{q^i}$$

4. Hamming

$$q^{n-k} \geq \sum_{i=0}^{t} \binom{n}{i} (q-1)^i$$

▲

EXAMPLE 5.14

The distance between three words obeys the triangle equality

$$d(x, y) + d(y, z) \geq d(x, z) \tag{a}$$

Observe that

$$W(x \oplus z) \leq W(x) + W(z) \tag{b}$$

which follows from the definition of weight and modulo-2 addition. Assume that

$$x = z \oplus y = y \oplus z$$

Then,

$$z = y \oplus x = x \oplus z$$

Use these in equation (b) to give

$$W(x \oplus z) \leq W(y \oplus z) + W(x \oplus y)$$

or

$$d(x, z) \leq d(y, z) + d(x, y)$$

since

$$d(A, B) \triangleq W(A \oplus B)$$

▲

EXAMPLE 5.15

The structure for codes developed over GF(2^m) is as follows: For example, let $m = 4$ and GF(16). The elements are

$$
\begin{array}{ll}
0 & (0000) \\
1 & (1000) \\
\alpha & \vdots \\
\vdots & \vdots \\
\alpha^{14} & \vdots
\end{array}
$$

Let the string of input information bits be represented by x's

$$
\underbrace{xxxx}_{\alpha_i} \quad \underbrace{xxxx}_{\alpha_j} \quad \underbrace{xxxx}_{\alpha_k} \cdots
$$

First, divide the string into four-bit blocks where each block is a symbol or element from GF(16), as shown above. Next, clock the symbols α_ℓ into the encoder and output coded symbols.

▲

EXAMPLE 5.16

To find a code, use appendixes in Clark and Cain (1981), Lin and Costello (1983), and Peterson and Weldon (1972). The tables are given in octal.

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

For example, octal 3525 means 011 101 010 101, which corresponds to the generator polynomial

$$
g(x) = x^{10} + x^9 + x^8 + x^6 + x^4 + x^2 + 1
$$

Also, 23 corresponds to 010 011 $\rightarrow x^4 + x + 1$, or

$$
g(x) = 1 + x + x^4
$$

which is an $(n,k) = (15,11)$ code.

▲

Chapter 6
Convolutional Coding

Convolutional encoding is more complex than block coding. Its explanation is somewhat involved, since notation and terminology are not standard in the literature. Convolutional codes are "tree" or "recurrent" in that some checks depend on previous checks. Following Lin and Costello (1983), a code is denoted by (n,k,m), where k inputs produce n outputs and m is the memory order of the encoder. If the encoder has a single shift register, m is its number of delay elements. For the encoder in figure 6.1, $m = 3$. For each bit entering, the commutator rotates and outputs two bits; thus, the code is denoted as (2,1,3). First, the impulse responses of the encoder are defined to be the two output sequences $v^{(1)}$ and $v^{(2)}$ when $\bar{u}$ = (1 0 0 0 ...), that is, a one followed by an infinite string of zeros. The shift register is loaded with zeros before applying the input. Observe that four nodes feed the output modulo-2 adders, and thus the impulse response contains four bits. By placing a one at the input node (the three delay elements are still loaded with zeros), $v^{(1)} = 1$ and $v^{(2)} = 1$.

After moving the one through the register,

$$v^{(1)} = 1011 \triangleq g^{(1)} \tag{6.1}$$

$$v^{(2)} = 1111 \triangleq g^{(2)} \tag{6.2}$$

where $g^{(1)}$ and $g^{(2)}$ are the impulse responses for this encoder. They are also called generator sequences, connection vectors, or connection pictorials. The encoding equations become

$$\bar{v}^{(1)} = \bar{u} * \bar{g}^{(1)} \tag{6.3}$$

$$\bar{v}^{(2)} = \bar{u} * \bar{g}^{(2)} \tag{6.4}$$

where * represents convolution in discrete modulo-2 fashion. For the general case, let

$$\bar{g}^{(1)} = \left(g_0^{(1)}, g_1^{(1)}, g_2^{(1)}, \ldots, g_m^{(1)}\right) \tag{6.5}$$

$$\bar{g}^{(2)} = \left(g_0^{(2)}, g_1^{(2)}, g_2^{(2)}, \ldots, g_m^{(2)}\right) \tag{6.6}$$

$$\bar{v} = \left(v_0^{(1)}, v_0^{(2)}, v_1^{(1)}, v_1^{(2)}, \ldots\right) \tag{6.7}$$

where $\bar{v}$ is the interlacing of $v^{(1)}$ and $v^{(2)}$; then, a compact encoding equation is

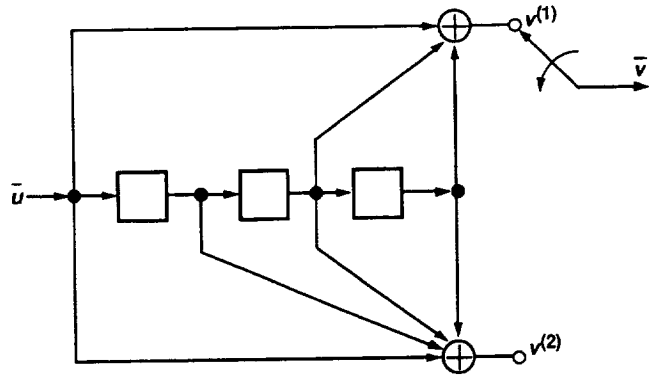


Figure 6.1.—One implementation of (2,1,3) convolutional encoder.

$$\bar{v} = \bar{u}\,\underset{\sim}{G} \tag{6.8}$$

where

$$\underset{\sim}{G} = \begin{bmatrix} g_0^{(1)}g_0^{(2)} & g_1^{(1)}g_1^{(2)} & g_2^{(1)}g_2^{(2)} & \cdots & g_m^{(1)}g_m^{(2)} & 0\,0 & \cdots \\ 0\,0 & g_0^{(1)}g_0^{(2)} & g_1^{(1)}g_1^{(2)} & & & & \\ 0\,0 & 0\,0 & g_0^{(1)}g_0^{(2)} & & & & \\ \vdots & \vdots & \vdots & & & & \\ & & & & & & \end{bmatrix} \tag{6.9}$$

is the generator matrix (of infinite extent).

EXAMPLE 6.1

For

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

$$\underset{\sim}{G} = \begin{bmatrix} 11 & 01 & 11 & 11 & 00 & 00 & 00 & \cdots \\ 00 & 11 & 01 & 11 & 11 & 00 & 00 & \\ 00 & 00 & 11 & 01 & 11 & 11 & 00 & \\ 00 & 00 & 00 & 11 & 01 & 11 & 11 & \cdots \\ \vdots & & & & & & & \end{bmatrix}$$

For $\bar{u}$ to five places (i.e., $\bar{u} = (10111)$),

$$\underset{\sim}{G} = \begin{bmatrix} 11 & 01 & 11 & 11 & 00 & 00 & 00 & 00 \\ 00 & 11 & 01 & 11 & 11 & 00 & 00 & 00 \\ 00 & 00 & 11 & 01 & 11 & 11 & 00 & 00 \\ 00 & 00 & 00 & 11 & 01 & 11 & 11 & 00 \\ 00 & 00 & 00 & 00 & 11 & 01 & 11 & 11 \end{bmatrix}$$

▲

The previous encoder can be redrawn in other ways, and this allows different means of describing the encoding procedure. In figure 6.2, the encoder has been redrawn by using a four-stage shift register; but observe that the first cell receives the first digit of $\bar{u}$ on the first shift. In the previous representation, the first output occurred when the first bit was at node 1 (to the left of the first cell). Another set of connection vectors G_j can be defined for this encoder:

$$G_1 = 1\,1, \qquad G_2 = 0\,1, \qquad G_3 = 1\,1, \qquad G_4 = 1\,1 \tag{6.10}$$

where the subscripts refer to the register delay cells. The number of digits in each vector is equal to the number of modulo-2 adders. Let $\underset{\sim}{G}^+$ be a generator matrix and again let $\bar{u}$ have five places; then,

$$\underset{\sim}{G}^+ = \left(G_1\ G_2\ G_3\ G_4\right)$$

or

$$\underset{\sim}{G}^+ = \begin{bmatrix} 11 & 01 & 11 & 11 & & & & \\ & 11 & 01 & 11 & 11 & & 0 & \\ & & 11 & 01 & 11 & 11 & & \\ 0 & & & 11 & 01 & 11 & 11 & \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix} \tag{6.11}$$

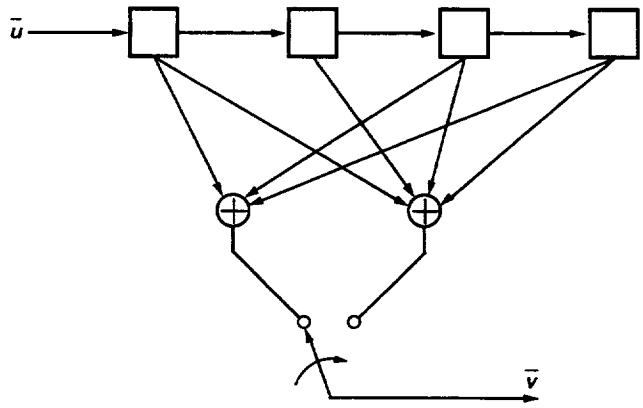


Figure 6.2.—Alternative encoder circuit of (2,1,3) convolutional encoder of figure 6.1.

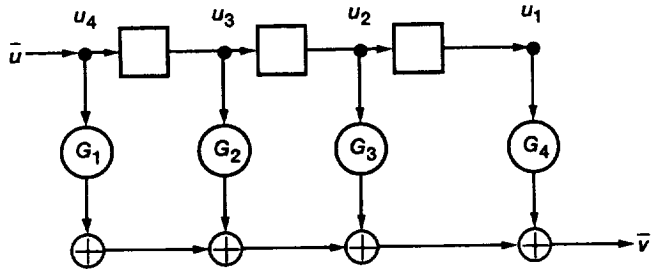


Figure 6.3.—Third representation of (2,1,3) convolutional encoder of figure 6.1.

which is just G in the previous example. Another representation of the encoder is given in figure 6.3. Here, the machine is at its fourth shift, so that from equation (6.7) the output is $v_3^{(1)}v_3^{(2)}$. From either the example or equation (6.11) for $\bar{u} = (10111)$,

$$\bar{v} = (11,01,00,01,01,01,00,11) \tag{6.12}$$

In figure 6.3 (as u_4 enters)

$$\begin{aligned} v_3^{(1)}v_3^{(2)} &= u_4G_1 \oplus u_3G_2 \oplus u_2G_3 \oplus u_1G_4 = 1\cdot(11)\oplus 1\cdot(01)\oplus 0\cdot(11)\oplus 1\cdot(11) \\ &= 11\oplus 01\oplus 00\oplus 11 = 10\oplus 11 = 01 \end{aligned}$$

which is indeed the value in equation (6.12) (the fourth pair). Thus, the fourth pair of outputs depends on u_4, u_3, u_2, and u_1, the memory (u_3, u_2, u_1), or the "convolution." Note that the last representation does not use a commutator.

Here, the same encoder has been described with three different circuit representations and two different sets of "connection vectors." This multiplicity of representations and terminology can cause some confusion if the reader is not careful when scanning the literature.

6.1 Constraint Length

Several definitions for the term "constraint length" can be found in the literature. The reasons for this confusing state of affairs will become evident as the discussion progresses. One reason is the variability in encoder design. For the simple case of a one-bit-in, three-bit-out encoder (fig. 6.4), the output commutator representation means that three output bits are generated for each input bit. Therefore, the code has $r = 1/3$ or $(n,k) = (3,1)$. Each block of n output bits depends on the present input bit (which resides in the first cell of the shift register), as well as on two previous input bits. Loosely, the encoder's memory is 2, which is both the number of previous input bits and the number of shifts by which a given bit can influence the output (do not count the shift when the bit first enters the shift register). The number of modulo-2 adders is three; in general, let v represent the number of such adders. Thus, here $v = n$. Each input bit affects three consecutive three-bit output blocks. So what is the "memory" of such an encoder? The various definitions of constraint length are variations on the notion of memory. The previous circuit can be redrawn as shown in figure 6.5 (upper part). Unfortunately, this encoder can also be drawn as shown in the lower part of the figure. The difference is the decision of placing the present input bit into a shift register stage or not. Therefore, how many shift register stages are needed for this particular (n,k) coding scheme?

Another encoder (fig. 6.6) has two input bits and three output bits per cycle; thus, $(n,k) = (3,2)$. Finally, in the case shown in figure 6.7, where $k = 3$ and $n = 4$, if the three input rails are considered to be inputs to shift registers; there is a zero-, a one-, and a two-stage register. In the case shown in figure 6.8, where $k = 2$ and $n = 3$, the output commutator rotates after two input bits enter. Two other variations (fig. 6.9) show some modulo-2 adders that deliver outputs to other adders.

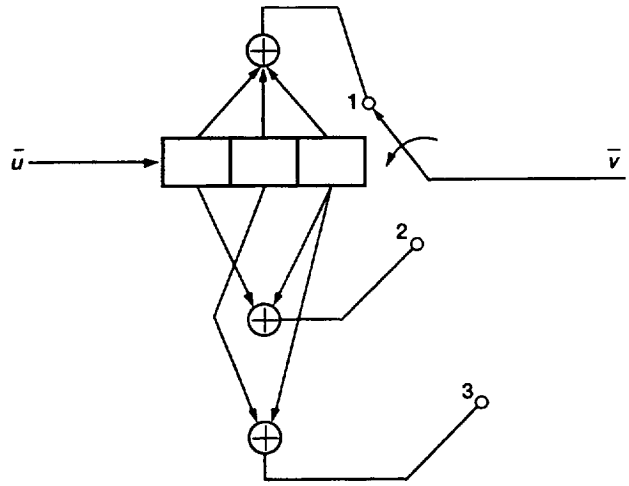


Figure 6.4.—General one-bit-in, three-bit-out convolutional encoder.

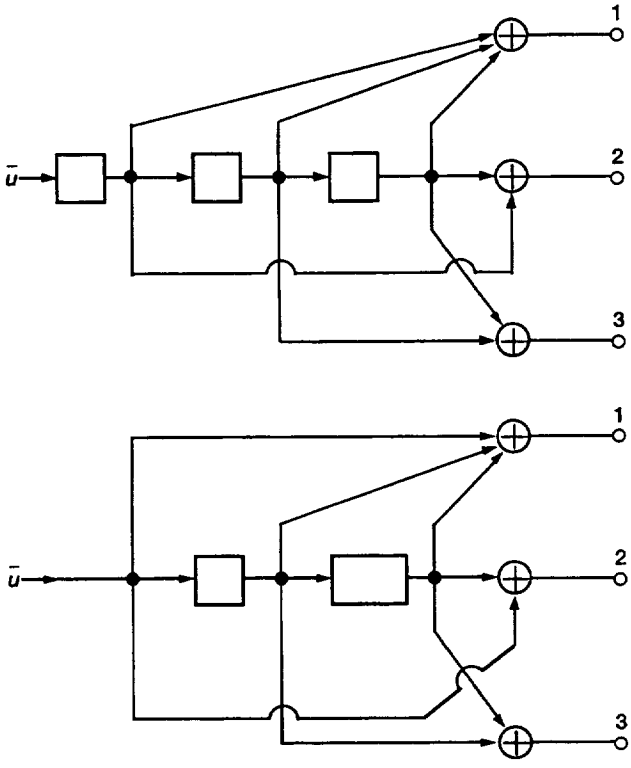


Figure 6.5.—Two alternative but equivalent representations of encoder circuit given in figure 6.4.

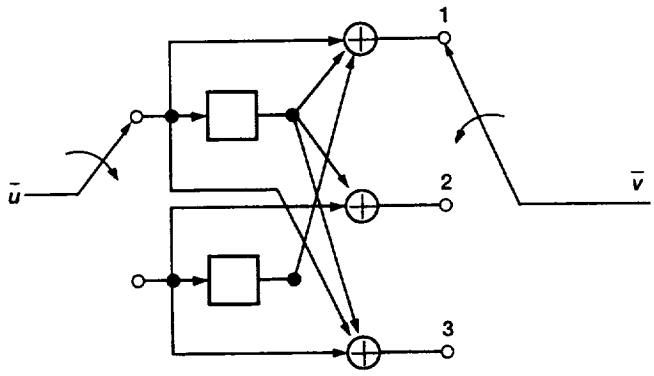


Figure 6.6.—General $k = 2$, $n = 3$ encoder.

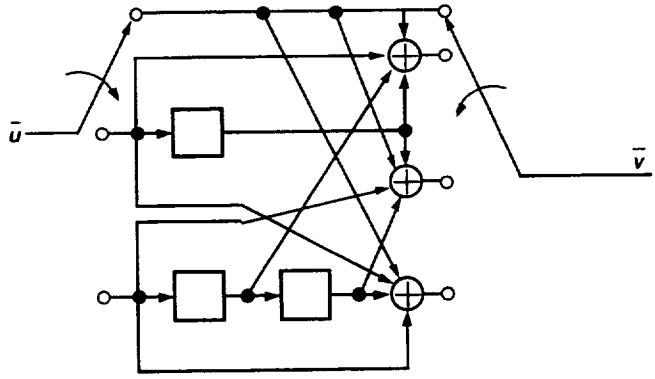


Figure 6.7.—General $k = 3$, $n = 4$ convolutional encoder.

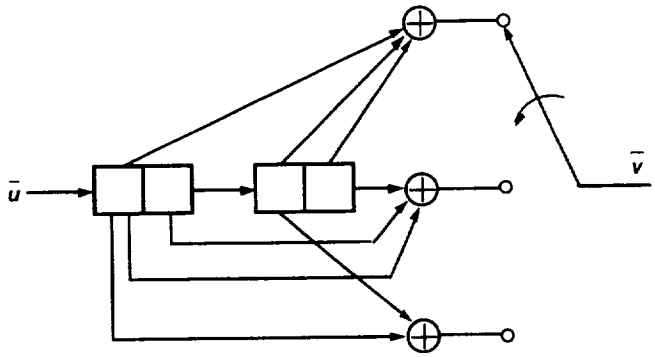


Figure 6.8.—Encoder where each register holds two input digits.

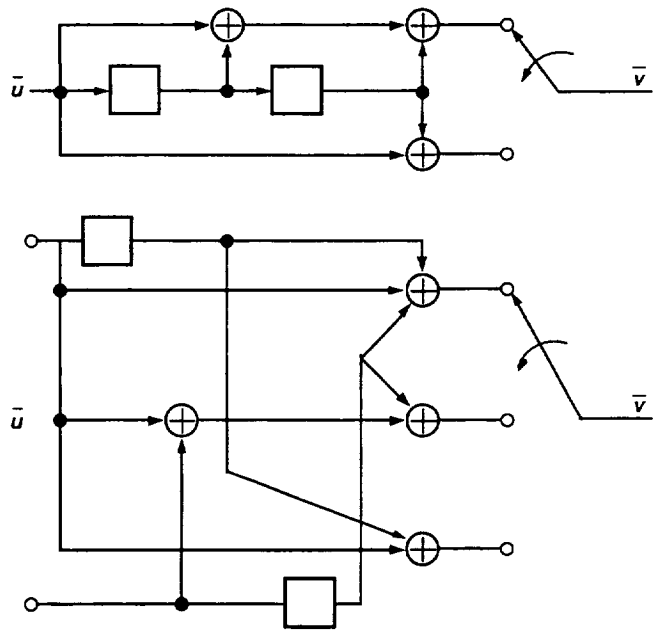


Figure 6.9.—Alternative encoder schemes wherein some modulo-2 adders feed other adders.

With these variations for encoder construction a "memory" is somewhat hard to define. Consider a variation on figure 6.5 depicted in figure 6.10. Here, each "delay element" consists of k stages and the input commutator would wait at each tap until k bits entered the machine. After loading the third, or last, tap the output commutator would sweep the remaining three outputs. For simplicity, assume that each "delay element" holds only one bit; then, each shift register consists of K_i single-bit elements. Here $K_0 = 0$, $K_1 = 1$, and $K_2 = 2$. The fact that the subscript equals the number of delay elements in this case is just an accident. (Figure 6.11 gives some situations where the notation can be confusing.)

With this background, the following definitions can be stated:

1. Let K_i be the length (in one-bit stages) of the ith shift register. Let k be the number of input taps; then,

$$m \triangleq \max_{1 \le i \le k} K_i \qquad \text{memory order}$$

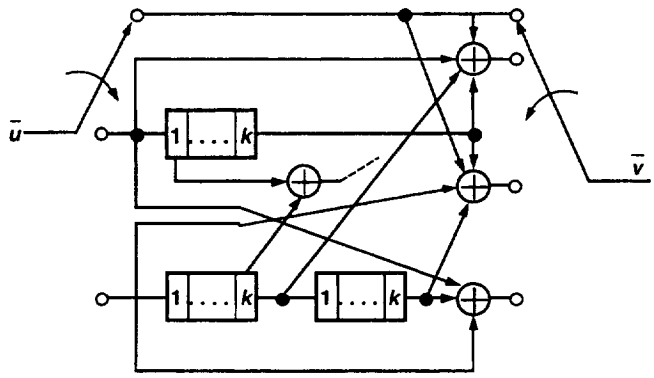


Figure 6.10.—Encoder wherein k-bit registers are employed (variation on circuit in fig. 6.5).

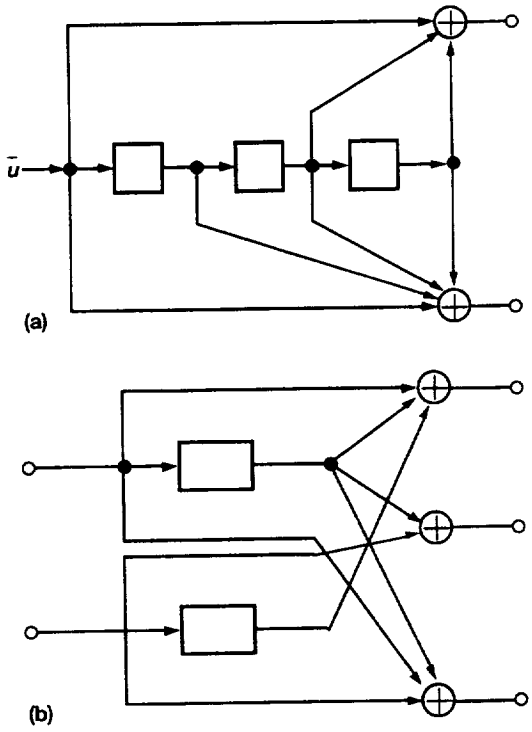


Figure 6.11.—Examples of encoders where constraint length, memory order, and number of shift registers can be confused. (a) (2,1,3); $m = K_1 = K = 3$. (b) (3,2,1); $K_1 = 1$, $K_2 = 1$, $m = K_1 = K_2 = 1$, $K = 2$.

$$K \triangleq \sum_{i=1}^{k} K_i \qquad \text{total encoder memory}$$

2. Following Lin and Costello (1983), k is the number of input taps and n is the number of output taps. Specify a code by (n,k,m). Then,

$$\text{CL} \triangleq n_A = n(m+1)$$

which says the constraint length (CL) is the maximum number of output bits that can be affected by a single input bit. This word definition is most often what is meant by constraint length. However, a slew of other terms is used. Sometimes, m is called the number of state bits; then,

$$\text{memory span} \triangleq m + k$$

Often, the memory span is called the CL. Sometimes, m is called the CL. Sometimes, n_A above is called the constraint span. In many situations, the CL is associated with the shift registers in different ways. For example, in figure 6.12, the $K = 4$ means the total encoder memory; whereas $K = 2$ is the number of k-bit shift registers.

3. Finally, the code rate needs to be clarified. A convolutional encoder generates n encoded bits for each k information bits, and $r = k/n$ is called the code rate. Note, however, that for an information sequence of finite length $k \cdot L$, the corresponding code word has length $n(L + m)$, where the final $n \cdot m$ outputs are generated after the last nonzero information block has entered the encoder. In other words, an information sequence is terminated with all-zero blocks in order to allow the encoder memory to clear. The block code rate is given by $kL/n(L + m)$, the ratio of the number of information bits to the length of the code word. If $L >> m$, then $L/(L + m) \approx 1$, and the block code rate and the convolutional rate are approximately equal. If L were small, the ratio $kL/n(L + m)$, which is the effective rate of information transmission, would be reduced below the code rate by a fractional amount

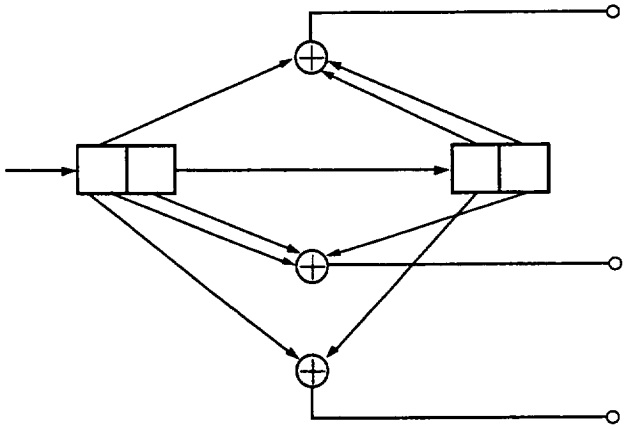


Figure 6.12.—Encoder where two two-bit registers are used and corresponding notational ambiguity. $k = 2$, $m = 3$, CL $\triangleq K$, $K = 4$ or $K = 2$.

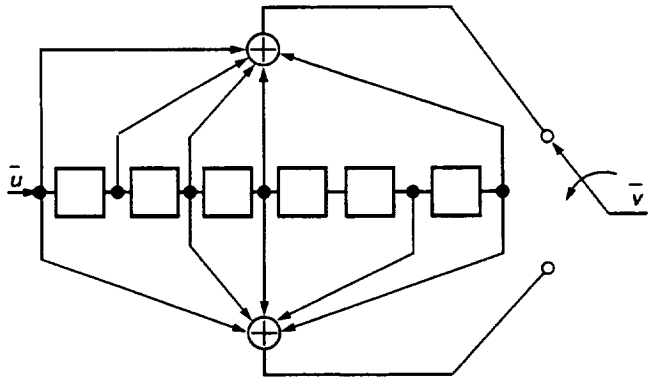


Figure 6.13.—Popular $r = 1/2$, $K = 7 = (m + k)$ convolutional encoder.

$$\frac{m}{L+m}$$

called the fractional rate loss. The nm blocks of zeros following the last information block are called the tail or flush bits.

4. Quite often, the memory span $(m + k)$ is designated as K, the constraint length. For example, a very popular $r = 1/2$, $K = 7$ encoder (fig. 6.13) means $(n = 2, k = 1)$. Here, the CL refers to the number of input bits $(m + k = 6 + 1)$, or the memory span.

6.2 Other Representations

With the convolutional term, constraint length, and other ideas covered, the many alternative representations for encoders can now be discussed. The example below summarizes the previous notions.

Example 6.2

Consider an encoder with delay cells R_i consisting of k subcells each (fig. 6.14). Often, the constraint length is the number of delay cells i. Here, every n (equal to v) outputs depends on the present k (those in cell 1) and $(K-1)$ previous k-tuples. Then, the code can be described as (n,k,K), and the constraint span is (K/k) v.

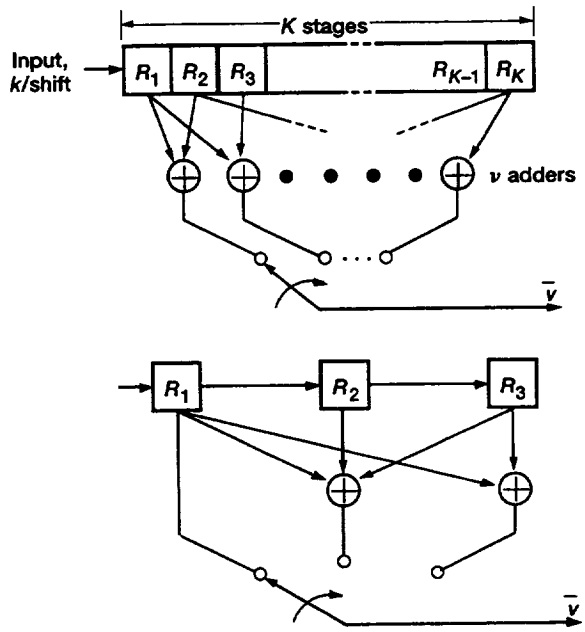


Figure 6.14.—General convolutional encoder (upper) and specific $K = 3$, $v = 2$ example.

For simplicity, assume one-bit cells; then, an encoder could be as shown in the lower portion of figure 6.14. Write the output 3-tuple at a particular shift as (v_1, v_2, v_3). Then,

$$v_1 = R_1$$

$$v_2 = R_1 \oplus R_2 \oplus R_3$$

$$v_3 = R_1 \oplus R_3$$

Let the input stream be $u = u_0,\ldots,u_A, u_B, u_C,\ldots$ and assume that u_C is shifted into R_1. Then, R_2 contains u_B and R_3 contains u_A and

$$v_1 = u_C$$

$$v_2 = u_C \oplus u_B \oplus u_A$$

$$v_3 = u_C \oplus u_A$$

▲

The next representation uses a delay operator D as follows: Define

$$g^{(1)}(D) = g_0^{(1)} + g_1^{(1)}D + g_2^{(1)}D^2 + \cdots + g_m^{(1)}D^m$$

$$g^{(2)}(D) = g_0^{(2)} + g_1^{(2)}D + g_2^{(2)}D^2 + \cdots + g_m^{(2)}D^m$$

For the connection vectors of the previous sections, this means that

$$\bar{g}^{(1)} = (1011) \Rightarrow g^{(1)}(D) = 1 + D^2 + D^3$$

$$\bar{g}^{(2)} = (1111) \Rightarrow g^{(2)}(D) = 1 + D + D^2 + D^3$$

Define

$$g(D) = g^{(1)}\left(D^2\right) + Dg^{(2)}\left(D^2\right)$$

Then,

$$\bar{v}(D) \triangleq u\left(D^m\right)g(D)$$

for

$$u = (10111) \Rightarrow 1 + D^2 + D^3 + D^4$$

Then,

$$\begin{aligned}\bar{v}(D) &= u\left(D^m\right)\left[\left(1 + D^4 + D^6\right) + D\left(1 + D^2 + D^4 + D^6\right)\right] \\ &= 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15}\end{aligned}$$

after the multiplication and modulo-2 additions, where the exponents refer to the position in the sequence. Recall from equation (6.12) that

$$\bar{v} = (11, 01, 00, 01, 01, 01, 00, 11)$$

Therefore, the above expression in D notation gives

$$\bar{v} = (1101000101010011)$$

This is again just the polynomial representation for a bit stream.

The next representation is the "tree" for the encoder (fig. 6.15). The particular path taken through the tree is determined by the input sequence. If a zero appears, an upward branch is taken. If a one appears, a downward branch is taken. The output sequence for a given input sequence is read from the diagram as shown. For the input sequence 1011, the output is 11 01 00 10.

The next representation is the state diagram. The state of the encoder is defined to be the bits in the shift register with the following association. For the (2,1,3) code developed earlier (fig. 6.1), the states are labeled by using the following rule: The set of states is denoted by

$$S_0, S_1, S_2, \ldots, S_{2^K - 1}$$

where the subscripts are the coefficients

$$S_i \leftrightarrow b_0\, b_1\, b_2, \ldots, b_{K-1}$$

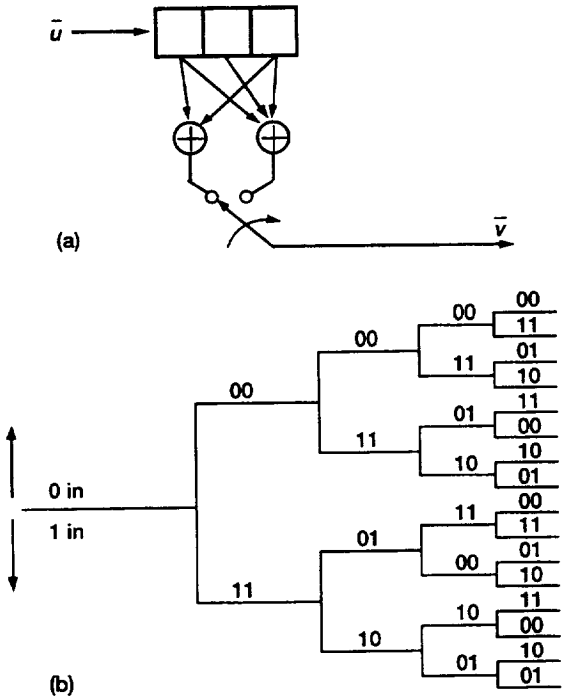


Figure 6.15.—Encoder (a) and tree for encoder (b).

where the integer i is expanded as

$$i = b_0 2^0 + b_1 2^1 + b_2 2^2 + \cdots$$

For this example, $K = 3$ (total encoder memory). Then, eight states are possible S_0, S_1, S_2, S_3, S_4, S_5, S_6, and S_7. The state notation and register contents correspond as follows:

Binary K-tuple	State
000	S_0
001	S_4
010	S_2
011	S_6
100	S_1
101	S_5
110	S_3
111	S_7

The corresponding state diagram (fig. 6.16) has the following interpretation: If the encoder is in state S_4, for example, a zero input causes an output of 11 and movement to state S_0, and a one input causes an output of 00 and movement to state S_1.

The trellis is a state diagram with a time axis, and that for the above state diagram is given in figure 6.17. Each column of nodes in the trellis represents the state of the register before any input. If a register has K stages, the first $K - k$ bits in the register determine its state. Only $K - k$ bits are needed, since the end k bits are dumped out as the next k input bits occur. The trellis has 2^{K-k} nodes in a column, and successive columns refer to successive commutation times. Branches connecting nodes indicate the change of register state as a particular input of k bits is shifted in and a commutation is performed. A branch must exist at each node for

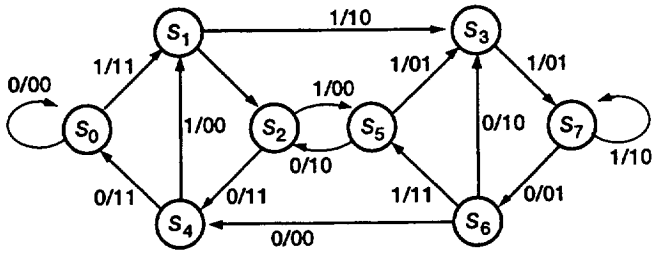


Figure 6.16.—State diagram for encoder given in figure 6.1.

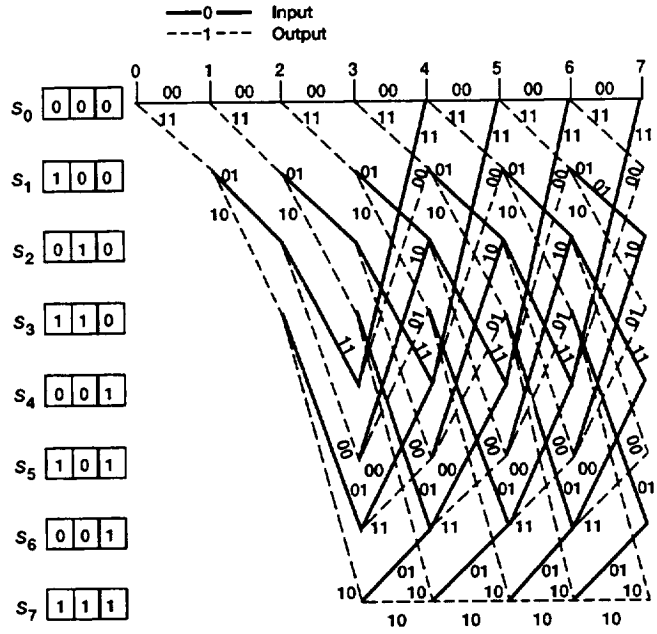


Figure 6.17.—Trellis diagram for state diagram of figure 6.16.

each possible k-bit input. If the register is in some initial state, not all nodes are possible until $K - k$ bits have been shifted in and the register is free of its initial condition. After Lk input bits, a distance of L columns has been progressed into the trellis, producing Ln output symbols. The trellis for the (2,1,3) code under consideration has eight rows of nodes, corresponding to the eight states $S_0,...,S_7$. Each column of nodes represents a time shift (when a commutation occurs). The dashed and solid lines represent paths taken for a one or zero input. For example, the input sequence $u = 111$ (assuming that the register is in S_0 state) takes three dashed paths and winds up at state S_7. The outputs are labeled so that the output sequence is 111001. A block of zeros will sooner or later move the register back to state S_0; this is called flushing. For this code, three zeros are needed to flush (clear) the encoder (to return to state S_0 from any other state).

6.3 Properties and Structure of Convolutional Codes

Recall that the codes have several modes of representation. The "algebraic" forms include connection pictorials, vectors, and polynomials; as well as generator matrices. The tree, state, and trellis diagrams are geometrical formulations.

A rather academic point is that of a catastrophic encoder. Such an encoder can get hung up so that a long string of ones produces, for example, three output ones followed by all zeros. If the three leading ones are corrupted by the channel, the decoder can only assume that all zeros constitute the message; thus, a theoretically arbitrary long sequence of errors results.

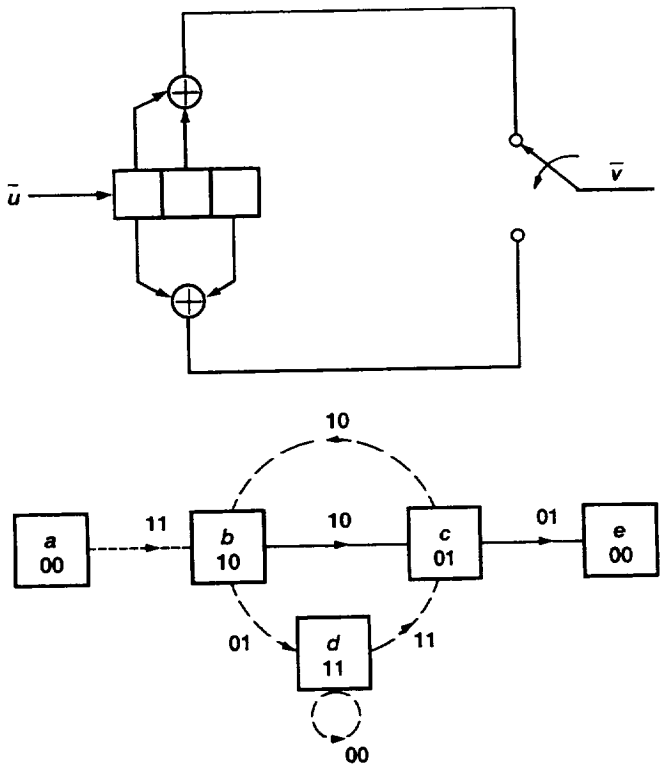


Figure 6.18.—Example of catastrophic encoder.

EXAMPLE 6.3

The encoder in figure 6.18 is a catastrophic encoder. Such machines can be easily recognized by noting the connection vectors. They will have a common multiple. Here,

$$g^{(1)} = 110 \rightarrow 1 + D$$

$$g^{(2)} = 101 \rightarrow 1 + D^2$$

but

$$1 + D^2 = (1 + D)(1 + D) = g^{(1)}(1 + D)$$

Thus,

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

and their common multiple is $1 + D$. Next, consider a code where

$$g^{(1)} = 1 + D^2 + D^3$$

$$g^{(2)} = 1 + D + D^2 + D^3$$

Now, $g^{(2)}/g^{(1)} = 1$ with a remainder of D. Since a remainder exists, no common factor is present; hence, the encoder is not catastrophic. In general, if the ratio equals D^{ℓ} for $\ell \geq 0$, the code is not catastrophic. The state diagram reveals catastrophic behavior when a self-loop of zero weight (that from state d in fig. 6.18) exists. This zero-weight self-loop cannot be in either "end" state in the diagram (here, a or e). In this diagram, a and e represent the same state. Systematic codes are never catastrophic.

▲

6.4 Distance Properties

Let A and B be two code words of length i branches in a trellis. The Hamming distance is as before

$$d_H(A, B) = w(A \oplus B)$$

Define the ith-order column distance function $d_c(i)$ as the minimum d_H between all pairs of code words of length i branches that differ in their first branch of the code tree. Another way of saying this is that $d_c(i)$ is the minimum-weight code word over the first $(i + 1)$ time units whose initial information block is nonzero. It depends on the first $n(i + 1)$ columns of $\underline{G}$ (for (n,k) code); hence, the word "column" in the definition. Two special distances are defined in terms of the column distance function as follows:

$$d_{\min} = d_c(i = m)$$
$$d_{\text{free}} = d_c(i \to \infty)$$

The minimum distance $d_{\min}$ occurs when $i = m$, the memory order; whereas d_{free} is for arbitrarily long paths. Quite often, $d_{\min} = d_{\text{free}}$. The distance profile is the set of distances

$$\underline{d} = [d_c(1), d_c(2), d_c(3), \ldots]$$

In general, these distances are found by searching the trellis. An optimum distance code has a $d_{\min}$ that is greater than or equal to the $d_{\min}$ of any other code with the same constraint length (memory order). An optimum free distance code has a similar property with d_{free}.

The next measure is the determination of the weight distribution function A_i. Here, A_i is the number of code words of weight i (the number of branches is not important here). This set $\{A_i\}$ is found from the state diagram as shown next.

The error correction power in a block code sense would say

$$t = \left(\frac{d_{\text{free}} - 1}{2} \right)$$

but this is a rather coarse measure. Observe for future reference that a tree repeats itself after K branchings. In the trellis, there are 2^{K-1} nodes for 2^{K-1} states. For a given register, the code structure depends on the taps. Nonsystematic codes have larger d_{free}, but systematic ones are less prone to the accumulation of errors.

The final topic for this chapter is the generating function $T(x)$ for a code. It is defined as

$$T(x) = \sum_i A_i x^i$$

where A_i is the number of code words of length i. The function is derived by studying the state diagram for a specific code. Problem 10.5 of Lin and Costello (1983) is used to describe the procedure. The code is described as (3,1,2) with encoder diagram shown in figure 6.19. The connection vectors are

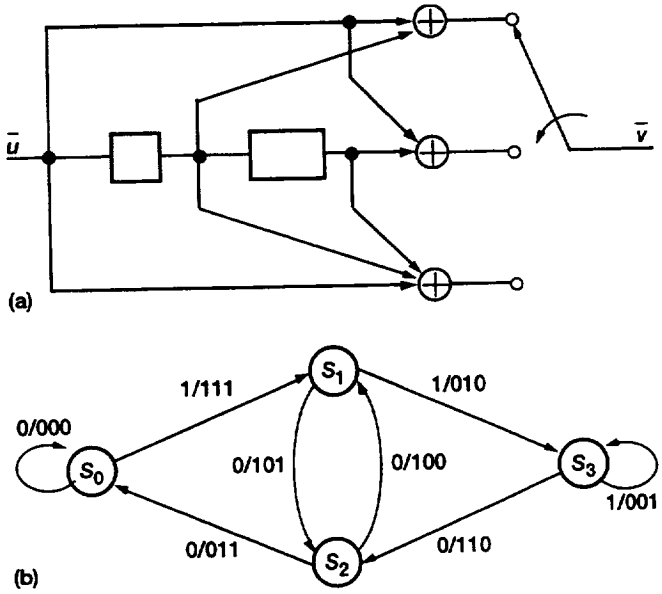


Figure 6.19.—Encoder (a) and state diagram (b) for (3,1,2) code.

$$g^{(1)} = (1\,1\,0), \qquad g^{(2)} = (1\,0\,1), \qquad g^{(3)} = (1\,1\,1)$$

From the encoder diagram the state diagram can be drawn as shown in the lower part of the figure. Next, the S_0 state is split into two parts as shown in figure 6.20, which constitutes the modified state diagram. Added to the branches are branch gain measures x^i, where i is the weight of the n encoded bits on that branch. The S_0 state is separated to delineate paths that "reemerge" to that state after passing through several intermediate states. If a self-loop is attached to the S_0 state, it is dropped at this step. From the modified state diagram, the generating function can be determined by using signal flow graph procedures. The S_0 states on the left and right are called the initial and final states of the graph, respectively. The terms needed are defined as

1. Forward path—A path connecting the initial and final states that does not go through any state more than once
2. Path gain—The product of the branch gains along a path F_i
3. Loop—A closed path starting at any state and returning to that state without going through any other state twice. A set of loops is "nontouching" if no state belongs to more than one loop in the set. Define

$$\Delta = 1 - \sum_i C_i + \sum_{j,k} C_j C_k - \sum_{\ell,o,p} C_\ell C_o C_p + \cdots$$

where $\sum_i C_i$ is the sum of the loop gains, $\sum_{j,k} C_j C_k$ is the product of the loop gains of two nontouching loops summed over all pairs of nontouching loops, $\sum_{\ell,o,p} C_\ell C_o C_p$ is the product of the loop gains of three nontouching loops summed over all triples of nontouching loops, etc. Next, define Δ_i, which is exactly like Δ but only for that portion of the graph not touching the ith forward path. That is, all states along the ith forward path, together with all branches connected to these states, are removed from the graph when computing Δ_i. Mason's formula for graphs gives the generating function as

$$T(x) = \frac{\sum_i F_i \Delta_i}{\Delta}$$

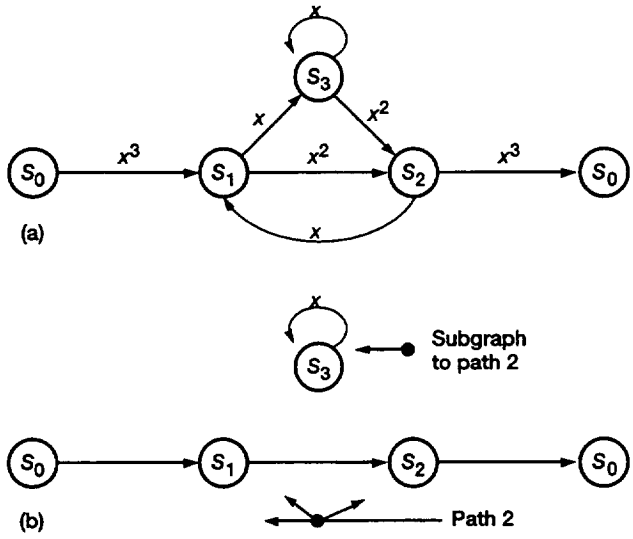


Figure 6.20.—Modified state diagram (a) of figure 6.19(b) and path 2 and its subgraph (b).

where the sum in the numerator is over all forward paths. First, calculate Δ, and this requires counting all loops. Figure 6.20(a) shows three loops, which are listed here along with their path gains.

Loop		
1	S_3S_3	$C_1 = x$
2	$S_1S_3S_2S_1$	$C_2 = x^4$
3	$S_1S_2S_3$	$C_3 = x^3$

There is only one set of nontouching loops, {loop 1, loop 3}, and the product of their gains is $C_1C_3 = x^4$. Thus, Δ is found as

$$\Delta = 1 - \left(x + x^4 + x^3\right) + x^4 = 1 - x - x^3$$

Now, to find Δ_1, there are two forward paths

Forward path		
1	$S_0S_1S_3S_2S_0$	$F_1 = x^8$
2	$S_0S_1S_2S_0$	$F_2 = x^7$

where the gains are also found. Because path 1 touches all states, its subgraph contains no states; thus,

$$\Delta_1 = 1$$

Because path 2 does not touch state S_3, its subgraph is that shown in figure 6.20(b). Only one loop exists here, with gain = x; thus,

$$\Delta_2 = 1 - x$$

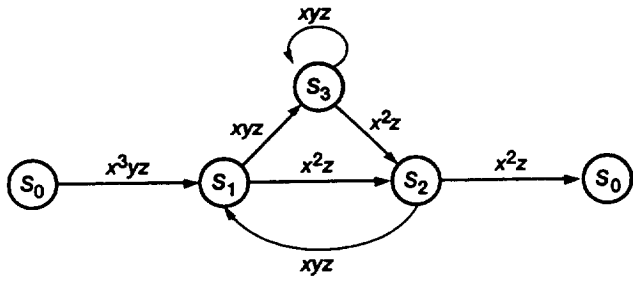


Figure 6.21.—Augmented state diagram of figure 6.20.

Finally, the transfer function is

$$T(x)=\frac{x^8\cdot 1+x^7(1-x)}{1-x-x^3}=\frac{x^7}{1-x-x^3}=x^7+x^8+x^9+2x^{10}+3x^{11}+4x^{12}+\cdots$$

with the following interpretation: The coefficients of x^7, x^8, and x^9 are all unity and have one code word each with weights 7, 8, and 9. Continuing in this manner, two words with weight 10, three with weight 11, etc.

Next, the augmented state diagram is made (fig. 6.21). Here, the branches are given added weighting. The exponent of y is the weight of the output code word, and each branch is given the factor z. Repeating the previous calculation gives

Loop		
1	S_3S_3	$C_1=xyz$
2	$S_1S_3S_2S_1$	$C_2=x^4y^2z^3$
3	$S_1S_2S_1$	$C_3=x^3yz^2$

The pair of loops has gain $C_1\,C_3=x^4\,y^2\,z^3$; thus,

$$\Delta=1-\left(xyz+x^4y^2z^3+x^3yz^2\right)+x^4y^2z^3=1-xyz-x^3yz^2$$

The forward path 1 is $F_1=x^8\,y^2\,z^4$: then, $\Delta_1=1$. The forward path 2 is $F_2=x^7\,y^1\,z^3$; then, $\Delta_2=1-xyz$. The generating function is therefore

$$T(x,y,z)=\frac{\sum_i F_i\Delta_i}{\Delta}=\frac{x^7yz^3}{1-xyz-x^3yz^2}=x^7yz^3+x^8y^2z^4+x^9y^3z^5+x^{10}y^2z^5+x^{10}y^4z^6+2x^{11}y^3z^6$$

$$+x^{11}y^5z^7+3x^{12}y^4z^7+x^{12}y^5z^8+\cdots$$

with the following interpretation: The first term means that the code word with weight 7 has output sequence with weight 1 (y exponent) and length of 3 branches (z exponent). The other terms have similar interpretations. This completes the discussion of convolutional encoders.

Chapter 7
Decoding of Convolutional Codes

The decoding of convolutional codes can be divided into three basic types: Viterbi, sequential, and threshold. Viterbi and sequential are similar in that they search through the trellis or tree; whereas threshold can be used for both block and convolutional codes. Threshold is also associated with the terms "majority logic," "feedback," and "definite decoding." Historically, sequential came first, but it is simpler to discuss Viterbi's algorithm first.

7.1 Viterbi's Algorithm

The idea in Viterbi's algorithm is to select a string of received bits and compare them with all possible strings obtained by tracing all possible paths through the trellis. For a sufficiently long string and not many errors, it seems reasonable to assume that one path through the trellis should agree rather closely with the received string. In other words, the decoder has properly reproduced the sequence of states that the encoder performed. The few bits of disagreement are the channel-induced errors. Experience has shown that the correct, or most likely path, becomes evident after about five constraint lengths through the trellis. The scheme is therefore to compare and store all possible paths for a set number of steps through the trellis and then select the "survivor," the most likely path. Some storage can be saved by closely studying the properties of paths through the trellis. To study these effects, a metric is defined as follows: Let $\bar{v}$ be the transmitted code word and $\bar{r}$ be the received sequence. For the DMC with channel transition probability $p(r_i \mid v_i)$,

$$p(\bar{r} \mid \bar{v}) = \prod_{i=0}^{N-1} p(r_i \mid v_i)$$

$$\bar{v} = (v_0, v_1, \ldots, v_{N-1}), \qquad \bar{r} = (r_0, r_1, \ldots, r_{N-1})$$

Then, taking the log (to reduce to sums) gives

$$\log\ p(\bar{r} \mid \bar{v}) = \sum_{i=0}^{N-1} \log p(r_i \mid v_i)$$

This is the log-likelihood function, and it is the "metric" associated with the path $\bar{v}$. The notation of Lin and Costello (1983) uses

$$M(\bar{r} \mid \bar{v}) \triangleq \log p(\bar{r} \mid \bar{v})$$

whereas others use $\Gamma(\bar{r} \mid \bar{v}) = \log p(\bar{r} \mid \bar{v})$. The metric for each segment along the path is

$$M(r_i \mid v_i) = \log p(r_i \mid v_i)$$

and is called a "branch metric." Thus, the path metric is the sum of all branches:

$$M(\bar{r} \mid \bar{v}) = \sum_{i=0}^{N-1} M(r_i \mid v_i)$$

A partial path metric for the first j branches of a path is

$$M\left([\bar{r} \mid \bar{v}]_j\right) = \sum_{i=0}^{j-1} M(r_i \mid v_i)$$

For the binary symmetric channel with transition probability p,

$$\log p(\bar{r} \mid \bar{v}) = \log \left[p^z (1-p)^{N-z}\right]$$

where $z = d(\bar{r}, \bar{v})$ is the Hamming distance between $\bar{r}$ and $\bar{v}$. Thus,

$$M(\bar{r} \mid \bar{v}) = N \log(1-p) - z \log\left(\frac{1-p}{p}\right) = -A - Bz$$

where A and B are positive constants ($p < 0.5$). Therefore, minimizing the Hamming distance maximizes the metric.

The basis of Viterbi decoding is the following observation: If any two paths in the trellis merge to a single state, one of them can always be eliminated in the search for the optimum path. The path with the smaller net metric at this point can be dropped because of the Markov nature of the encoder states. That is, the present state summarizes the encoder history in the sense that previous states cannot affect future states or future output branches. If both paths have the same metric at the merging state, either one can be eliminated arbitrarily without altering the outcome. Thus, "ties" cause no problems. In Lin and Costello (1983), the metric is chosen as

$$\text{metric} = \sum_{i=0}^{N-1} C_2\left[\log p(r_i \mid v_i) + C_1\right]$$

to bias the metrics for ease of computation. The constants C_1 and C_2 are chosen appropriately.

The storage required at each step in the trellis is straightforward, although the notation is not. Essentially, one of the two paths entering a node is stored as the "survivor." The notation variation between authors is the indexing from either zero or one in the counting. In Lin and Costello (1983), there are 2^K states at a step in the trellis; others use 2^{K-1}. Thus, the number of survivors is either 2^K or 2^{K-1} per level, or step, within the trellis. If L is the constraint length, 2^{kL} metrics are computed at each node, so that $2^{k(L-1)}$ metrics and surviving sequences must be stored. Each sequence is about $5kL$ bits long before the "final survivor" is selected. Thus, Viterbi requires $L < 10$. The complexity goes as 2^K while the cost goes as 2^ν, where ν is the number of modulo-2 adders in the encoder. The scheme is good for hard- or soft-decision demodulators. If one starts and stops at the zero, or topmost, node of the trellis, the transient in getting into or out of the trellis proper is called

the input transient and output flush. A truncated algorithm does not use the tail or flush bits. Tail-biting preloads its trailing bits from the zero start node to enter the full trellis. It then starts decoding after the tail is loaded.

7.2 Error Performance Bounds

As for block codes, the probabilities of sequence (string or path) errors is first found for error performance bounds. Then, the bit errors are bounded by terms involving the raw channel transition probability. Recall for blocks that a syndrome indicated that a block was in error and then the bits were processed. Here, no flag (syndrome) is found; but the sequence closest in Hamming distance consistent with the possible paths through the trellis is chosen. Thus, the error counting is again not extremely crisp. If the generator function is computed (impractical in many cases), then for hard decisions (binary symmetric channel),

$$p_{\text{block}} = p_{\text{string}} \le T(x)\Big|_{x=\sqrt{4p(1-p)}}$$

$$p_{\text{bit}} \le \frac{1}{k}\frac{\partial T(x,y)}{\partial y}\Big|_{\substack{y=1 \\ x=\sqrt{4p(1-p)}}}$$

For soft decisions,

$$p_{\text{block}} \le T(x)\Big|_{x=\sqrt{4p(1-p)}}$$

$$p_{\text{bit}} \le \frac{1}{2k}\frac{\partial T(x,y)}{\partial y}\Big|_{\substack{y=1 \\ x=e^{-rE_b/N_o}}}$$

In general, the coding gain is bounded by

$$\frac{r\,d_{\text{free}}}{2} \le \text{gain} \le r\,d_{\text{free}}$$

The Viterbi algorithm is used for raw bit error rates in the range 10^{-4} to 10^{-5} or better. The decision depth is the number of steps into the trellis before a decision is made. When $T(x,y)$ is not feasible, use

$$p_{\text{bit}} \approx \frac{1}{k} B_{d_{\text{free}}} 2^{d_{\text{free}}} p^{d_{\text{free}}/2}$$

where $B_{d_{\text{free}}}$ is the total number of ones on all paths of weight d_{free}, that is, the number of dotted branches on all these paths.

The Viterbi algorithm may be summarized as follows:

1. At time t, find the path metric for each path entering each state by adding the path metric of the survivor at time $t-1$ to the most recent branch metric.

2. For each state of time t, choose the path with the maximum metric as the survivor.
3. Store the state sequence and the path metric for the survivor at each state.
4. Increment the time t and repeat steps 1 to 3.

It is necessary to truncate the survivors to some reasonable length, called the decision depth δ. At time t, a decision is forced at time $t - \delta$ by using some criterion, which may be found by trial and error in some cases. As time progresses, it may be necessary to renormalize the metrics.

EXAMPLE 7.1

Consider the trellis diagram for a particular (3,1,2) code (fig. 7.1). The bold line is a possible input path, and the corresponding input and output sequences u and v to the encoder are

$$u = 10011010$$

$$v = 111\,101\,011\,111\,010\,110\,100\,101$$

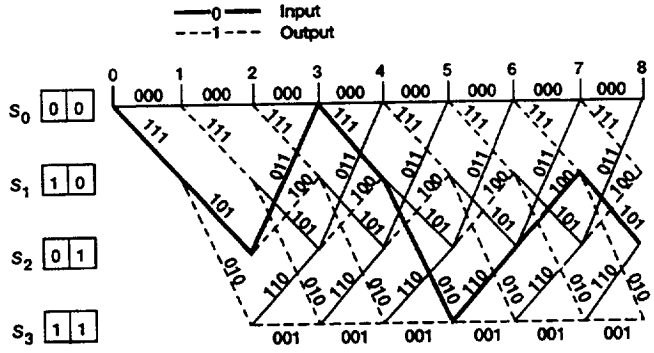


Figure 7.1.—Trellis diagram and arbitrary path through it.

Suppose the received sequence is

$$R = 111\,101\,111\,111\,000\,110\,101\,101$$
$$\uparrow \qquad \uparrow \qquad \uparrow$$

where the errors are denoted by arrows. The decoding steps can be summarized as follows: To simplify matters, the Hamming distance between possible paths is used to select survivors. As stated earlier, real decoders accumulate a metric, but using the closest path in Hamming distance is equivalent (and easier) for example purposes.

Step 1: Figure 7.2 shows the paths that are needed to get past the transient and enter the complete trellis. These four paths terminate at step 2 in the trellis, and their Hamming distances from the received path are

Path 1 input 00, output 000 000, $H = 5$

Path 2 input 01, output 000 111, $H = 4$

Path 3 input 10, output 111 101, $H = 0$

Path 4 input 11, output 111 010, $H = 3$

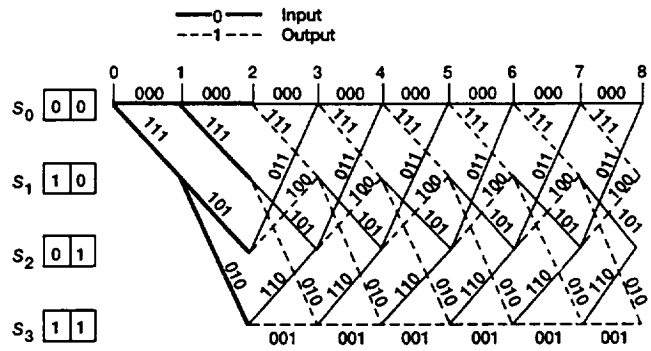


Figure 7.2.—Set of paths required in input transient phase (before complete trellis is available).

Step 2: Next, each path is extended from its node at step 2. For example, path 1 is extended to the column 3 nodes with inputs 0 or 1 and with outputs 000 and 111, respectively. Call these paths 1a and 1b. Their Hamming distances from the received sequence are

$$\text{Path 1a} \quad H = 5 + 3 = 8$$

$$\text{Path 1b} \quad H = 5 + 0 = 5$$

Since path 1b is closer in Hamming distance, it is the "survivor." The extensions of the other paths are as follows, where the "a" path is the uppermost one from a particular node:

$$\text{Path 2a} \quad H = 4 + 1 = 5$$

$$\text{Path 2b} \quad H = 4 + 2 = 6$$

$$\text{Path 3a} \quad H = 0 + 1 = 1$$

$$\text{Path 3b} \quad H = 0 + 2 = 2$$

$$\text{Path 4a} \quad H = 3 + 1 = 4$$

$$\text{Path 4b} \quad H = 3 + 2 = 5$$

Therefore, the survivors are paths 1b, 2a, 3a, and 4a (fig. 7.3). To simplify notation at this point, drop the letter designation on the paths and call them just 1, 2, 3, and 4. Now, extending the paths to nodes in column 4, where again the "a" and "b" extensions are used, gives

$$\text{Path 1a} \quad H = 5 + 1 = 6$$

$$\text{Path 1b} \quad H = 5 + 2 = 7$$

$$\text{Path 2a} \quad H = 5 + 1 = 6$$

$$\text{Path 2b} \quad H = 5 + 2 = 7$$

$$\text{Path 3a} \quad H = 1 + 3 = 4$$

Path 3b $H = 1 + 0 = 1$

Path 4a $H = 4 + 1 = 5$

Path 4b $H = 4 + 2 = 6$

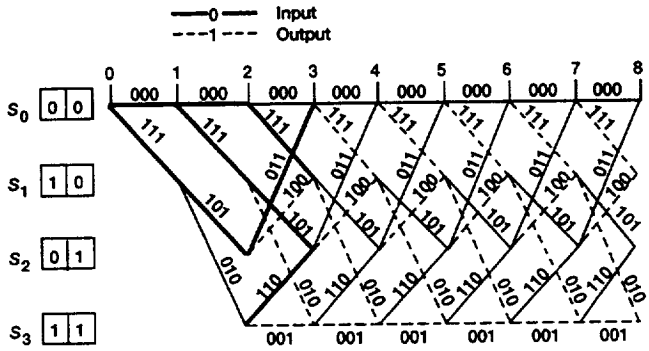


Figure 7.3.—Survivors at column 3 in trellis.

Then, the survivors are 1a, 2a, 3b, and 4a, with corresponding Hamming distances (fig. 7.4):

Path 1a $H = 6$

Path 2a $H = 6$

Path 3b $H = 1$

Path 4a $H = 5$

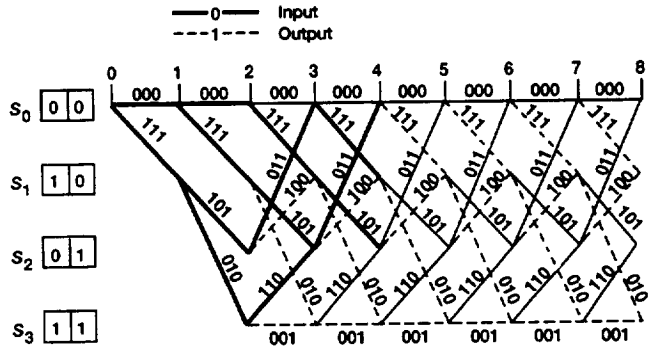


Figure 7.4.—Survivors at column 4.

The next extension yields the survivors at column 5 (fig. 7.5):

Path 1b $H = 7$

Path 2a $H = 6$

Path 3b $H = 2$

Path 4a $H = 5$

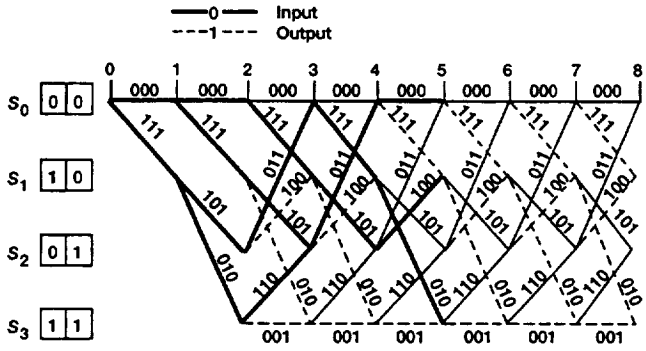


Figure 7.5.—Survivors at column 5.

The next extension gives the survivors at column 6 (fig. 7.6):

Path 1b $H = 8$

Path 2b $H = 7$

Path 3a $H = 2$

Path 4b $H = 6$

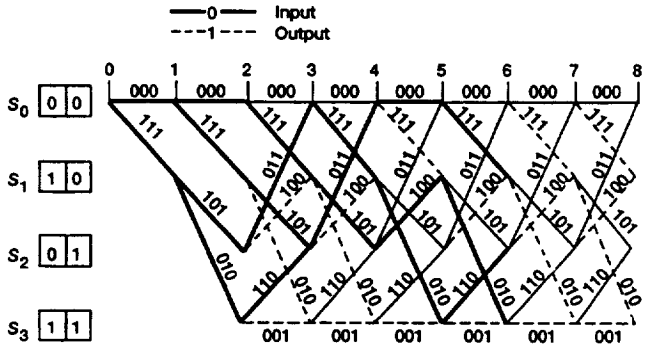


Figure 7.6.—Survivors at column 6.

The next step gives the survivors at column 7 (fig. 7.7):

Path 1b $H = 9$

Path 2a $H = 7$

Path 3b $H = 3$

Path 4a $H = 6$

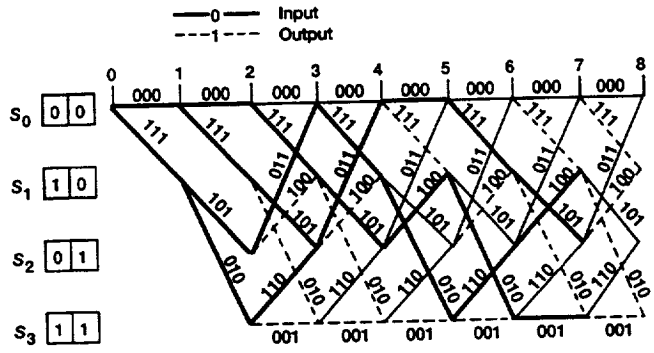


Figure 7.7.—Survivors at column 7.

Note that path 3b differs from the received sequence by only three places and is the best choice for the decoder to make. The next closest path has weight 6, which is much farther away. If the decoder were to make the decision to now drop all contenders, correct decoding has occurred. ▲

Michelson and Levesque (1985) gives some tables of good codes for use with Viterbi's algorithm. Their notation corresponds to that used earlier as

$$(n, k, m) \leftrightarrow (v, b, k)$$

Their notation is (bk) stages, b bits/shift between registers, b bits/shift input to the encoder, v bits/shift out, rate b/v, and constraint length k. Table 7.1 (our notation used) gives the constraint length (number of shift registers in encoder) and the tap connection vectors. Here $b = 1$, and binary signaling is assumed. Table 7.2 gives the very important derivative expressions for calculating the bit-error-rate curve. General derivative expressions were given earlier while discussing error calculations, and those given in the figure are quite useful for practical calculations.

TABLE 7.1.—GOOD CODES FOR VITERBI DECODING

Constraint length, K	Connection vectors
$r = 1/2; b = 1$	
3	111, 101
4	1111, 1101
5	11101, 10011
6	111101, 101011
7	1111001, 10100111
8	11111001, 10100111
9	111101011, 101110001
$r = 1/3; b = 1$	
3	111, 111, 101
4	1111, 1101, 1011
5	11111, 11011, 10101
6	111101, 101011, 100111
7	1111001, 1110101, 1011011
8	11110111, 11011001, 10010101

TABLE 7.2.—DERIVATIVE FUNCTIONS FOR CODES OF TABLE 7.1

K	$\frac{d}{dy} T(x,y)\vert\, y = 1, \quad x = \sqrt{4p(1-p)}$
$r = 1/2$	
3	$x^5 + 4x^6 + 12x^7 + 32x^8 + 80x^9 + 192x^{10} + 448x^{11} + 1024x^{12} + 2304x^{13} + \ldots$
4	$2x^6 + 7x^7 + 18x^8 + 49x^9 + 130x^{10} + 333x^{11} + 836x^{12} + 2069x^{13} + 5060x^{14} + \ldots$
5	$4x^7 + 12x^8 + 20x^9 + 72x^{10} + 225x^{11} + 500x^{12} + 1324x^{13} + 3680x^{14} + 8967x^{15} + \ldots$
7	$36x^{10} + 211x^{12} + 1404x^{14} + 11\,633x^{16} + 76\,628x^{18} + 469\,991x^{20} + \ldots$
$r = 1/3$	
3	$3x^8 + 15x^{10} + 58x^{12} + 201x^{14} + 655x^{16} + 2052x^{18} + \ldots$
4	$6x^{10} + 6x^{12} + 58x^{14} + 118x^{16} + 507x^{18} + 1284x^{20} + 4323x^{22} + \ldots$
5	$12x^{12} + 12x^{14} + 56x^{16} + 320x^{18} + 693x^{20} + 2324x^{22} + 8380x^{24} + \ldots$
6	$x^{13} + 8x^{14} + 26x^{15} + 20x^{16} + 19x^{17} + 62x^{18} + 86x^{19} + 204x^{20} + 420x^{21} + 710x^{22} + 1345x^{23} + \ldots$

7.3 Sequential Decoding

Sequential decoding decodes by searching through the tree to make the best guess. Unlike Viterbi, it does not necessarily have to perform the 2^K operations per decoded block. Sequential decoding steps quickly through the tree when $\bar{r} = \bar{v}$ or nearly $\bar{v}$. However, during noisy intervals it moves up and down the tree searching for the best candidate paths.

7.3.1 ZJ or Stack Algorithm

In the ZJ or stack algorithm (fig. 7.8), metrics for each path are computed, stored, and bubble sorted in the stack at each step. Then, the most likely one is extended in all possible paths, and the metric is stored and recomputed. If the decoder is on the right (low error) path, the accumulated metric should grow steadily. If the path is incorrect, its metric will drop, and one of the lower paths in the stack will have its metric bubbled to the top. The decoder goes back to the next candidate and starts a new search. In noisy conditions, the decoder can spend so much time moving back and forth that the input storage buffer overflows. This buffer overflow due to random search times is the practical limitation.

Since paths of different length must be compared, a reasonable method of metric computation must be made. The most commonly used metric was introduced by Fano, thus the name Fano metric. It is

$$M(r_i \mid v_i) = \log_2 \frac{p(r_i \mid v_i)}{p(r_i)} - R \tag{7.1}$$

where $R = r$ is the code rate (note the problem in notation between code rate (R or r) and the received vector $\bar{r}$ and its components r_i). The partial metric for the first ℓ branches of a path $\bar{v}$ is

$$M\left([\bar{r} \mid \bar{v}]_{\ell-1}\right) = \sum_{i=0}^{n\ell-1} \log_2 p(r_i \mid v_i) + \sum_{i=0}^{n\ell-1} \left[\log_2 \frac{1}{p(r_i)} - R\right]$$

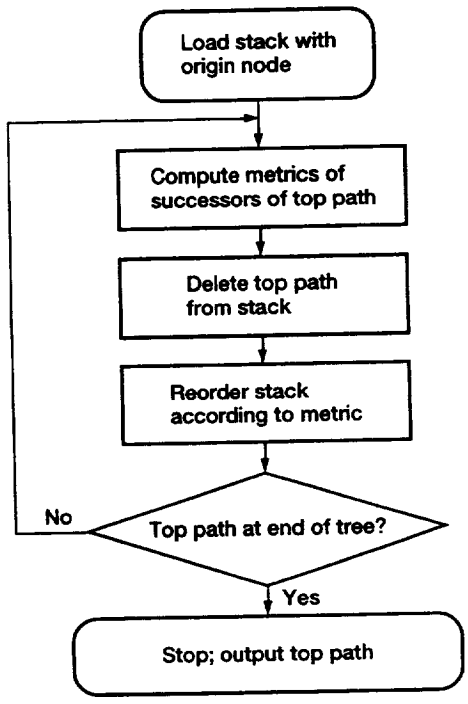


Figure 7.8.—ZJ or stack algorithm.

The first term is the metric in Viterbi's algorithm; the second term represents a positive bias that increases linearly with path length. Hence, longer paths have a larger bias than shorter ones, reflecting the fact that they are closer to the end of the tree and more likely to be a part of the most likely path.

For the binary symmetric channel with transition probability p, the bit metrics are, from equation (7.1),

$$M(r_i \mid v_i) = \log_2 \frac{p}{(1/2)} - R = \log_2 2p - R \qquad r_i \neq v_i$$

$$M(r_i \mid v_i) = \log_2 \frac{1-p}{(1/2)} - R = \log_2 2(1-p) - R \qquad r_i = v_i$$

In the stack of the ZJ algorithm, an ordered list or stack of previously examined paths of different lengths is kept in storage. Each stack entry contains a path along with its metric values. The path with the largest metric is placed on top, and the others are listed in order of decreasing metric. Each decoding step consists of extending the top path in the stack by computing the branch metrics of its 2^k succeeding branches and then adding these to the metric of the top path to form 2^k new paths, called the successors of the top path. The top path is then deleted from the stack, its 2^k successors are inserted, and the stack is rearranged in order of decreasing metric values. When the top path in the stack is at the end of the tree, the algorithm terminates. Figure 7.8 summarizes the idea.

7.3.2 Fano Algorithm

Before discussing the famous Fano algorithm, let us first consider the concept of breakout nodes. The plot in figure 7.9 is the accumulated metric versus depth into the tree. Nodes on the most likely path, wherein the metric never falls below this metric value, are called breakout nodes. The significance of this is that the decoder never goes back farther than the last such node. Simulation to show the distribution of breakout nodes at the same metric value gives a qualitative measure of noise bursts on the channel, with accompanying delays.

The stack algorithm can require large storage to remember all terminal nodes of the paths examined during the decoding process. The fact that the next path to be searched farther is always readily available in the stack is the main reason for the simplicity of the algorithm. However, this advantage is paid for by the large storage needed. The Fano algorithm, on the other hand, uses little storage and does not remember the metrics of previously searched paths. Therefore, in order to know whether the current path under consideration has the highest metric and should be explored farther, the decoder uses a set of threshold values to test the acceptability of the paths. As long as the current threshold value is satisfied (i.e., the currently explored path has a higher metric than the current threshold value), the decoder is assumed to be moving along an acceptable path and proceeds forward. However, if the current threshold is violated, the algorithm stops proceeding and goes into

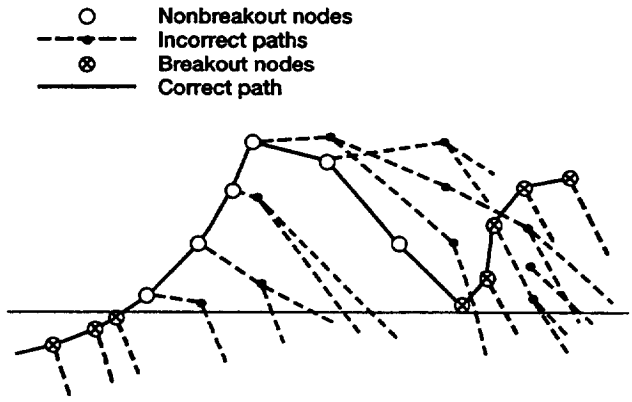


Figure 7.9.—Schematic path showing accumulation of metric and indicating breakout nodes.

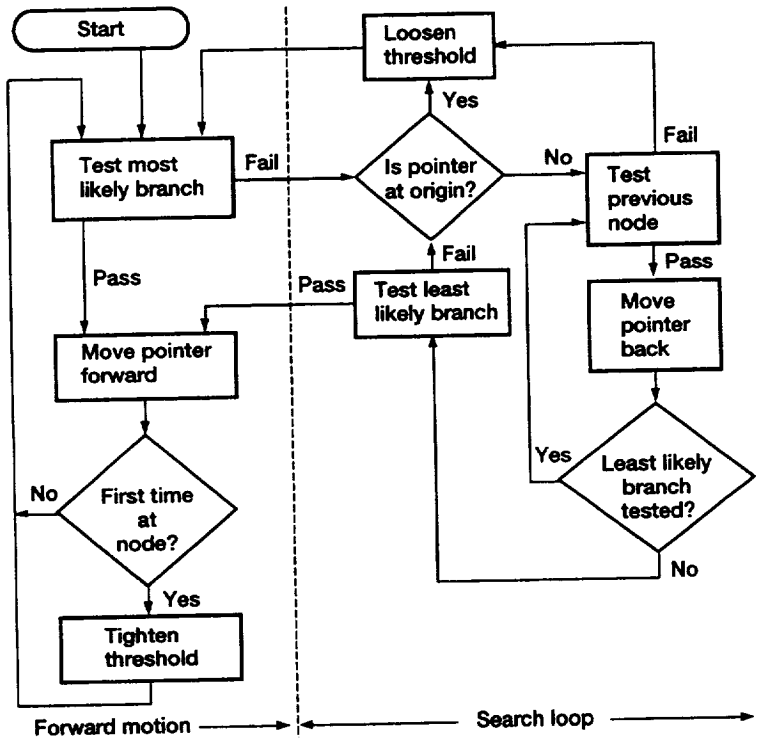


Figure 7.10.—Fano algorithm.

a search mode for a better path. Since no storage is used, the search for a better path is performed on a branch-per-branch basis. From its current node, the decoder retraces its way back and attempts to find another path that does not violate the current threshold value. The new path is then searched until trouble again occurs. The details of the rules that govern the motion of the decoder are best explained by the flowchart in figure 7.10. Note that the decoder has both a search loop and a forward loop. The rules are as follows:

1. A particular node is said to satisfy any threshold smaller than or equal to its metric value and to violate any threshold larger than its metric value.
2. Starting at value zero, the threshold T changes its value throughout the algorithm by multiples of increments Δ, a preselected constant.
3. The threshold is said to have been tightened when its value is increased by as many Δ increments as possible without being violated by the current node's metric.
4. The node being currently examined by the decoder is indicated by a search pointer.
5. In the flow diagram, when a node is tested, the branch metric is computed and the total accumulated metric of that node is evaluated and compared with the threshold. A node may be tested in both a forward and a backward move.
6. The decoder never moves its search pointer to a node that violates the current threshold.
7. The threshold is tightened only when the search pointer moves to a node never before visited.

Figure 7.11 is a more detailed flowchart. Finally, the received distance tree searched is shown schematically in figure 7.12.

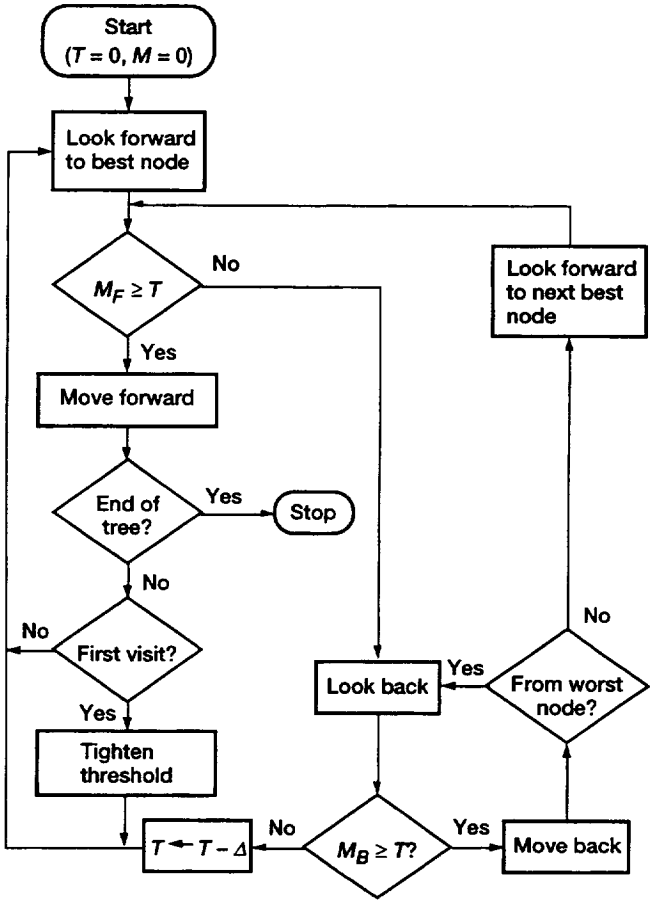


Figure 7.11—Slightly more detailed flowchart for Fano algorithm.

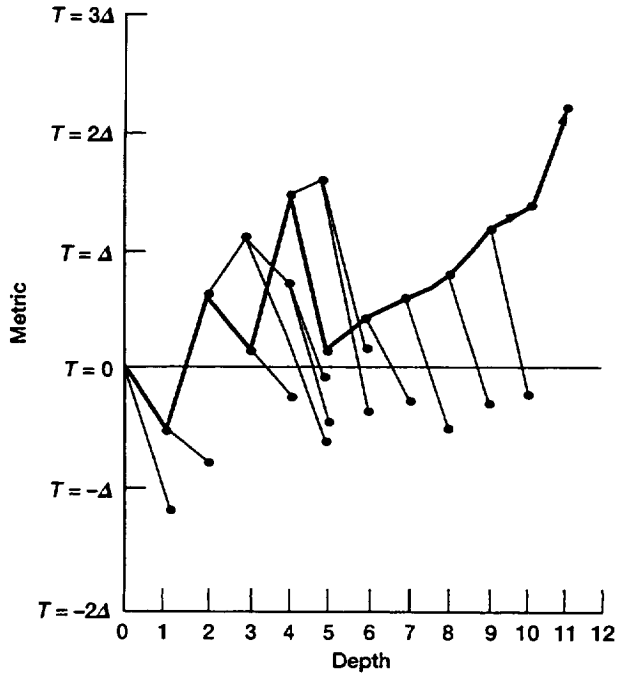


Figure 7.12.—Schematic of how tree may be searched in Fano algorithm.

7.4 Performance Characteristics for Sequential Decoding

The number of computations required to decode one information bit has asymptotically a Pareto distribution, that is,

$$P(C \geq N) < \beta N^{-\alpha} \qquad N >> 1$$

where C is the number of computations and α is the Pareto exponent. This relationship was found by Gallager (1968) and verified through simulation. Here, α and β are functions of the channel transition probabilities and the code rate r. The code rate and exponent are related by

$$r = \frac{E_0(\alpha)}{\alpha}$$

where

$$E_0(\alpha) = \alpha - \log_2\left[\left((1-p)^{\frac{1}{1+\alpha}} + p^{\frac{1}{1+\alpha}}\right)^{1+\alpha}\right]$$

is the Gallager function for the binary symmetric channel. The solution when $\alpha = 1$ yields $r \triangleq R_0$, the computational cutoff rate. In general, systems use $1 < \alpha < 2$. The value of R_0 sets the upper limit on the code rate. For binary input and continuous output (very soft) case,

$$R_0 = 1 - \log_2\left(1 + e^{-rE_b/N_o}\right)$$

and for the DMC/BSC

$$R_0 = 1 - \log_2\left[1 + 2\sqrt{p(1-p)}\right]$$

Then,

$$P_{\text{bit}} \leq \frac{2^{-KR_0/r}}{\left\{1 - 2^{-[R_0/r - 1]}\right\}^2} \qquad r < R_0$$

where K is the constraint length.

Chapter 8
Summary

Chapters 4 and 7 provide the formulas needed to plot bit error rate versus signal-to-noise ratio E_b/N_o for either block or convolutional codes. From these plots, the code gain at a prescribed bit error rate can be inferred. The real system issues of cost/complexity, robustness to bursts, etc., cannot be so neatly handled. Most block codes are decoded by using hard decisions; convolutional codes are often soft implementations. Since block decoders work on code words (blocks), the calculation of bit error rates is always approximate; chapter 4 covers this in detail. In many applications, block codes are used only for error detection. The calculation of bit error rates for convolutional codes is also somewhat vague, but for certain cases the results are rather crisp.

The theoretical and practical limits of code rate are channel capacity C and computational cutoff rate R_0, respectively. For the binary symmetric channel, they are

$$C = 1 + p\log_2 p + (1-p)\log_2(1-p)$$

$$R_0 = 1 - \log_2\left[\frac{1}{2} + \sqrt{p(1-p)}\right]$$

For binary or quadrature phase shift keying with an analog decoder (infinitely soft) on the AWGN channel, they are

$$C = \frac{1}{2}\log_2\left(1 + 2r\frac{E_b}{N_o}\right)$$

$$R_0 = 1 - \log_2\left[1 + \exp\left(\frac{-rE_b}{N_o}\right)\right]$$

These require

$$\frac{E_b}{N_o} > \frac{2^{2r}-1}{2r} \qquad \text{for } r < C$$

$$\frac{E_b}{N_o} \geq -\frac{1}{r}\ln\left(2^{1-r}-1\right) \qquad \text{for } r < R_0$$

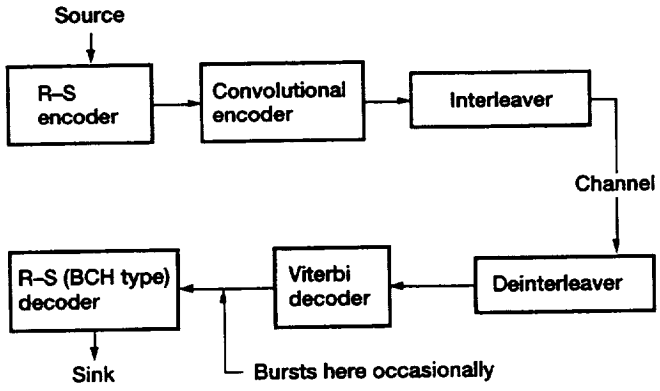


Figure 8.1.—Concatenated coding/decoding scheme along with interleaver/deinterleaver pair.

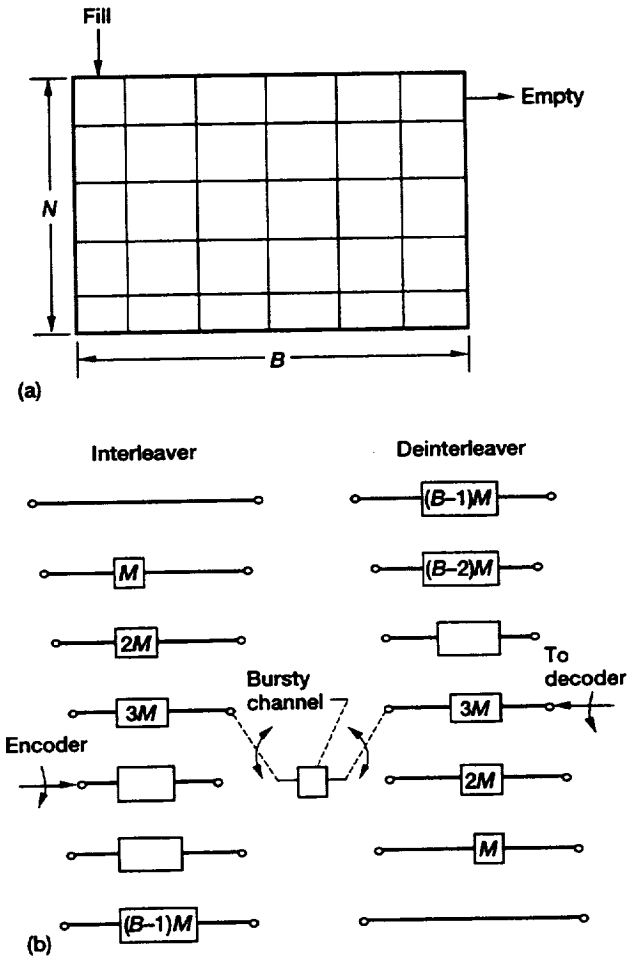


Figure 8.2.—Block interleaver (a), fill registers by column read out by rows; and convolutional interleaver (b).

For convolutional codes

$$\text{BER} \sim 2^{-KR_0/r}$$

Interleaving and concatenated codes are useful in that they break up large bursts as well as maximize the power and minimize the shortcomings of particular codes. Figure 8.1 shows an outer Reed-Solomon code with an inner convolutional one, as well as an interleaver for generality. The interleaver breaks up bursts into smaller ones that can be handled by the inner convolutional code. This inner decoder tends to make errors in bursts; then, the Reed-Solomon decoder can clean them up.

Recall that a burst may be defined as follows: Let the burst length be $\ell = mb$ and assume an $m \times n$ (row times column) interleaver. The interleaver breaks the burst into m smaller ones, each of length b. Recall that an (n,k) block code can correct a burst with length

$$b \leq \frac{n-k}{2}$$

Interleavers are used when a bursty channel exists (i.e., fading due to multipath and grain defects in magnetic storage). Viterbi decoders are better than sequential decoders on bursty channels, even though both are poor.

EXAMPLE 8.1

Figure 8.2 shows two interleavers, both a block and a convolutional type. For block interleaving, the transmitter reads encoder output symbols into a memory by columns until it is full. Then, the memory is read out to the modulator by rows. While one memory is filling, another is being emptied, so that two are needed. At the receiver, the inverse operation is effected by reading the demodulator output into a memory by rows and reading the decoder input from the memory by columns; two memories are also needed. For the convolutional case, all multiplexers in the transmitter and receiver are operated synchronously. The multiplexer switches change position after each symbol time, so that successive encoder outputs enter different rows of the interleaver memory. Each interleaver and deinterleaver row is a shift register, which makes the implementation straightforward.

▲

Appendix A
Q Function

The Q function is defined as

$$Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-u^2/2} du$$

with the property

$$Q(-x) = 1 - Q(x), \qquad Q(0) = \frac{1}{2}$$

It is related to the error function by

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-u^2} du$$

$$Q(x) = \frac{1}{2}\left[1 - \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right] = \frac{1}{2}\operatorname{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

where

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$$

Observe that

$$\operatorname{erfc}(x) = 2Q\left(x\sqrt{2}\right)$$

$$\operatorname{erf}(x) = 1 - 2Q\left(x\sqrt{2}\right)$$

The function is bounded as follows, where the bounds are close for $x \geq 3$:

$$\left(1 - \frac{1}{x^2}\right)\frac{e^{-x^2/2}}{x\sqrt{2\pi}} \leq Q(x) \leq \frac{1}{x\sqrt{2\pi}} e^{-x^2/2}$$

Appendix B
Glossary

A

Adder, modulo-2: Same as exclusive-OR

	0	1
0	0	1
1	1	0

ADM: Adaptive delta modulation

AGC: Automatic gain control

Algorithm: Berlekamp, Euclid, Fano, Hartmann-Rudolph, Omura, stack, threshold, Viterbi

ARQ: Automatic repeat request (used extensively in computers); a reverse channel is needed to alert the sender that the message was received in error, and a retransmission is needed.

Asymptotic coding gain: Block, $G \sim r(t+1)$; convolutional, $\frac{rd_{\text{free}}}{2} < G < rd_{\text{free}}$

Augmented code: Adding new code words to an existing code

AWGN: Additive white Gaussian noise

B

BCH: Bose-Chaudhuri-Hocquenghem

BCH bound: Lower bound for minimum distance for such codes, $d_{\min} \geq 2t+1$, where $2t + 1$ is the "design distance"

BEC: Binary erasure channel

BER: Bit error rate

Bit: Measure of information; a "unit" used to measure information

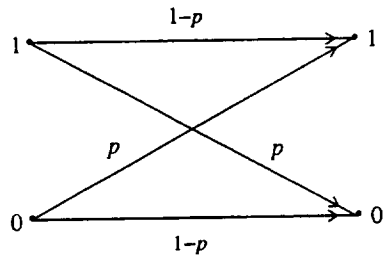
Binit: Binary digit 0 or 1; also called a "bit"

Bounded distance decoding: Algorithm wherein decoder corrects all errors of weight t or less but no others

Bounds on distance for block codes: Upper and lower bounds for $d_{\min}$

BPSK: Binary phase shift keying

BSC: Binary symmetric channel



Burst: A sequence wherein the first and last binit are in error but those in between may be correct (50/50); bursts often occur in groups (i.e., bursts of bursts).

C

Catastrophic error propagation: Some convolutional codes can generate infinite errors from a few in the "wrong place"; of academic interest only.

CCIT: International Telegraph and Telephone Consultative Committee

CDF: Column distance function; a measure used in convolutional codes

Check bit: One of $n - k$ redundant binits

Chien search: Searching error locator polynomial $\sigma(x)$ to find its roots. The error location numbers are the reciprocals of these roots.

Code rate: For blocks, k/n; for convolutional codes, $kL/n(L + m)$

Code word: One of 2^k words out of total 2^n n-tuples

Coding gain: The difference in E_b/N_o between a coded and uncoded channel at a specified BER

Computational cutoff rate: Maximum rate for sequential decoder to handle, R_0

Concatenated codes: Two codes in tandem, "outer" and "inner" codes

Constraint length (CL): Number of output bits that single input bit affects; there are many uses for this term.

Coset: Row of code vectors in standard array

CRC: Cyclic redundancy check

D

DEQPSK: Differentially encoded quadrature phase shift keying

Detection: Demodulator step to recover signal; coherent means the carrier and its phase are required, whereas incoherent does not use carrier phase information.

Distance (designed): Minimum guaranteed distance, $2t + 1$

Distance (Euclidean): Normal vector length between points, $d^2 = (\rho_i - S_i)^2$, where ρ_i and S_i are voltage levels in a demodulator

Distance (free): Minimum-weight code word in a convolutional code, d_{free}; it can be of any length and is generated by a nonzero information sequence.

Distance (Hamming): See Hamming distance.

DM: Delta modulation

DMC: Discrete memoryless channel

DMS: Discrete memoryless source

DPCM: Differential pulse code modulation

DPSK: Differential phase shift keying

E

E_b/N_o: Ratio of energy per bit to noise power level

Erasure: An output of a demodulator that means no decision; neither zero or one can be chosen.

Ergodic process: Each choice is independent of all previous ones.

Error pattern: Vector added by channel noise, received vector = transmitted + error

Expurgated code: Code with some code words discarded, often leaving remaining words with even weights

Extended code: Code with more check bits added, chosen such that the weight structure is improved

F

Fano algorithm: Original tree search (sequential) algorithm; it moves quickly through the tree when errors are few; it slows proportionally with error rate and thus adapts to noise level, unlike Viterbi's algorithm, which calculates 2^K values in each step.

FEC: Forward-error correcting (only case considered herein)

Field (Galois): See Galois field.

Fire codes: Burst-error-correcting codes

Fractional rate loss: $m/(L + m)$, for convolutional codes, where L denotes information bit length and m denotes memory order (the maximum length of all shift registers)

G

Galois field: Finite field of elements where number of elements is prime number or a power of prime number; codes are developed so that the code words become elements of such fields, GF(q).

GBC: General binary channel

GCD: Greatest common divisor

Golay code: Famous (23,12) perfect code whose properties have been studied exhaustively

H

Hamming codes: Defined by $n = 2^m - 1$, $n - k = m$, where $m = 1,2,3, \ldots$, $d_{\min} = 3$, and $t = 1$

Hamming distance: Number of positions by which two code words differ

Hamming weight: Number of ones in a code word

I

Interleaving: Block or convolutional; interleaving breaks up code blocks before transmission. After reception, the inverse reconstruction process tends to break up noise bursts added by the channel. Smaller bursts are more easily decoded.

ISI: Intersymbol interference

L

LCM: Lowest common multiple

Lengthened code: A code where parity check symbols have been added

Linear code: Sum of two code words is also a code word. Most practical codes are linear.

LPC: Linear product code

M

M-ary signaling: Modulator output is segmented into blocks of j bits. Then, each sequence is mapped into a waveform. There are $M = 2^j$ such waveforms. For example,

$$1001 \rightarrow S_1(\text{waveform})$$
$$0111 \rightarrow S_2(\text{waveform})$$

Here, $j = 4$, so that $M = 2^4 = 16$ waveforms.

Markov process: Each choice depends on previous ones but no farther back in the sequence of choices.

Message: Ordering of letters and spaces on a page

MFSK: Multiple-frequency shift keying

MLD: Maximum-likelihood decoder

MLSR: Maximum-length shift register codes

MSK: Minimum shift keying

N

Noise averaging: A method of understanding how codes work. The redundancy increases the uniqueness of words to help decoding, and noise averaging allows the receiver to average out the noise (by matched filtering) over long time spans, where T (word length) becomes very large.

NASA code use: The Mariner probes (1969–76) used a Reed-Muller $(2^m, m + 1)$ code (32,6) and decoded by a finite-field transform algorithm (over GF(2^m)). Then from 1977 on, NASA switched to a convolutional code with $K = 7$ constraint length and $m = 6$ memory.

O

ODP: Optimum distance profile, a code whose distance profile is best for a given constraint length

OKQPSK: Offset-keyed quadrature phase shift keying

OOK: On-off keying

P

Pareto distribution: Number of calculations C exceeding N in sequential decoder is given by this distribution:

$$p(C \geq N) = aN^{-\alpha} \qquad N >> 1$$

Perfect code: Code that corrects all patterns of t (or fewer) errors but no more. The Golay (23,12), Hamming ($t = 1$), and repetition (n odd) codes are the only known perfect binary codes.

Punctured code: Code where some parity check symbols have been deleted

Q

QPSK: Quadrature phase shift keying

Quick look-in:A convolutional code wherein two sequences can be added to get information; there is thus no decoding step in the classical sense.

R

R_0: Computational cutoff rate; replaces channel capacity C in a practical sense. The decoder cannot operate properly if $r > R_0$ ($r = k/n$).

Reed-Muller codes: Cyclic codes with overall parity check digit added

Reed-Solomon (R–S) code: Very popular nonbinary block code that is good for bursts. Its structure is known and thus its error probabilities are calculable with good accuracy.

Repetition: Code formed by repeating a given symbol a fixed number of times

S

SEC: Single-error correcting

SECDED: Single-error-correcting, double-error-detecting code, an extended Hamming code $(n,k) \rightarrow (n+1,k)$ (i.e., one added overall parity check is an example)

SNR: Often, ratio of signal to noise energy per bit (symbol). SNR = $\gamma_b = \alpha^2 E_b/N_o$, where E_b is waveform energy and α is amplitude of received signal $r(t)$

Source coding: Includes PCM, DPCM, DM (delta modulation), ADM (adaptive delta modulation), and LPC. In contrast, channel coding increases alphabet and cost, adds bandwidth, and needs a decoder.

Source rate: $R = H/T$ bits per message per second

Syndrome: Sum of transmitted and locally generated checks at receiver

T

Tail biting: A convolutional encoding scheme wherein a block of bits $L+m$ long is processed as follows: The last m bits are fed into the encoder to initialize it, but the output is ignored. Then, $L+m$ bits are encoded and transmitted. This eliminates the normal zero tail flush bits and gives the last m bits the same amount of coding protection that the L bits possess.

Tree codes: Codes wherein encoder has memory. Convolutional codes are a subset of tree codes.

U

Undetected error: The case wherein the error vector is itself a code word, so that its sum with the message vector creates a valid code word.

$$\text{Prob}(\text{Block error undetected}) = \sum_{j=1}^{n} A_j p^j (1-p)^{n-j}$$

where A_j denotes weight distribution value.

V

Viterbi: Very popular decoding algorithm for short-constraint-length ($K < 10$) convolutional codes; often $K = 7$, $r = 1/2$. Works well when uncoded bit error rate is about 10^{-4} or 10^{-5}.

W

Weight distribution: Knowing the sequence A_j, where A_j is the number of code words with weight j, means knowing the code structure very well.

Weight (Hamming): See Hamming weight.

Z

ZJ algorithm: Stack algorithm in sequential decoding. The most promising path in the tree has its accumulated metric bubbled to the top of the stack.

Appendix C
Symbols

A_i	number of code words with specific weight i
a_i	ith digit in message vector
b	burst length
C	channel capacity
$\underline{C}$	code word, $\underline{v}_m \underline{G}$
C_R	leading coefficients
C_1, C_2, C_3	checks
D	distance in Reed-Solomon codes
$D_{\min}$	minimum distance in Reed-Solomon codes
$\underline{d}$	distance profile
d_{free}	minimum-weight code word in convolutional code
d_H	Hamming distance
$d_{\min}$	distance between two closest code words
E_b	energy per information bit
E_c	energy per coded bit
E_m	message energy
E_s	energy in channel symbol
$E_0(\alpha)$	Gallager function
$\underline{e}$	noise vector; error vector; error pattern

$\underline{\hat{e}}$	assumed error vector
e_c	number of errors corrected
e_d	number of errors detected
f	distribution function of system (particle density function)
G	asymptotic gain
$\underline{G}$	code generator matrix
GF(p)	Galois fields
g(x)	generator polynomial
$\underline{H}$	parity check matrix
$H(x)$	average self-information or self-entropy in any message symbol
$H(y)$	entropy of output channel
$\underline{H}^T$	transpose of parity check matrix
H_S	entropy of code
I	quantity of information
K	constraint length
k	number of input bits; number of alphabetic symbols: Boltzmann's constant
L	information bit length
ℓ	number of errors; length of shortest burst in code word
M	path metric; movement parameter
M_B	backward-movement parameter in Fano algorithm
M_F	forward-movement parameter in Fano algorithm
m	memory order; number of slots per page; number of terms; number of message symbols
$\underline{m}$	message vector
$\underline{\hat{m}}$	output
N	average noise power; number of permutations; number of particles; number of equally likely messages
N_c	number of code words
N_e	number of n-tuples

N_o	noise power spectral density
n	number of output bits
n_A	constraint span
P	received power in modulated signal
p	probability; number of erasures corrected; transition probability
$p(E)$	probability of decoding error
p_B	probability that block is decoded incorrectly
p_b	probability of received bit error
$(p_b)_s$	resulting bit error rate into data sink
p_{bit}	probability of bit error
p_c	probability of bit error when coding is applied; coded channel bit
p_E	probability of decoder error
p_i	probability of any symbol occurring, $1/m$
p_m	probability of message error
$(p_m)_c$	block error rate for coding
$(p_m)_u$	probability that uncoded block message will be received in error
p_{symbol}	channel symbol error rate
p_u	uncoded channel bit
$p_u(E)$	probability of undetected error
p_w	probability of word error
Q	heat; number of source symbols
R	bit rate; code rate; data rate or information symbol rate
R_{max}	maximum information symbol rate
R_s	symbol rate; channel symbol rate; chip rate; baud; etc.
R_0	computational cutoff rate
r	code rate
$\bar{r}$	received vector; volume of real space
r_i	components of received vector

r_0	computational cutoff rate (in sequential decoding)
S	entropy; signal power
$\underline{S}$	n-k vector; syndrome
S_i	partial syndrome; voltage level in demodulator
T	threshold; time; Nyquist rate for no ISI, $1/2W$
$T(x)$	generating function
T_w	duration of code word
t	time; number of correctable errors in received word
u	transmitted waveform
$\hat{u}$	assumed input vector
$\underline{u}$	code vector; code vector sequence
$\bar{u}$	input vector in convolutional code
v	received waveform
$\underline{v}$	code vector
$\underline{v}_m$	message vector
$\bar{v}$	output vector in convolutional code; volume of velocity space; transmitted code word
v_x, v_y, v_z	velocity components
W	thermodynamic probability of state of interest; bandwidth
x	number of erasures
$\underline{x}$	symbol emitted by source
x_i	message symbol
x,y,z	coordinates
$\underline{y}$	received sequence
z	Hamming distance between $\bar{r}$ and $\bar{v}$
$\underline{z}$	received vector
α	event; number of information symbol errors per word error; symbol; Pareto exponent; amplitude of received signal

α^i	error number locations
β	event; general element
β_i	number of remaining errors in decoded block
γ_i	number of coset leaders of weight i
ρ_i	voltage level in demodulator
$\phi(x)$	smallest degree polynomial

References

Clark, G., and Cain, J., 1981, *Error-Correction Coding for Digital Communications*. Plenum Press, New York.

Gallager, R.G., 1968, *Information Theory and Reliable Communication*. John Wiley & Sons, New York.

Lin, S., and Costello, D., 1983, *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey.

Michelson, A.M., and Levesque, A.H., 1985, *Error-Control Techniques for Digital Communication*. Wiley-Interscience, New York.

Peterson, W.W., and Weldon, E., Jr., 1972, *Error-Correcting Codes*. Second ed., The MIT Press, Cambridge, Massachusetts.

Shannon, C.E., 1948, "Mathematical Theory of Communication." *Bell Systems Technical Journal*, vol. 27, no. 3, 1948, pp. 379–423 (Part I), pp. 623–656 (Part II).

Sklar, B., 1988, *Digital Communications Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey.

Sweeney, P., 1991, *Error Control Coding: An Introduction*. Prentice-Hall International, Englewood Cliffs, New Jersey.

Torrieri, D.J., 1984, "The Information-Bit Error Rate for Block Codes." *IEEE Transactions Communications*, vol. COM–32, no. 4, pp. 474–476.

Bibliography

Bhargava, V.K., et al., 1981, *Digital Communications by Satellite*. Wiley-Interscience, New York.

Couch, L.W., 1987, *Digital and Analog Communication Systems*. Second ed., Macmillan.

Fano, R.M., 1961, *Transmission of Information*. The MIT Press, Cambridge, Massachusetts, and John Wiley & Sons, Inc., New York.

Feher, K., 1981, *Digital Communications: Satellite/Earth Station Engineering*. Prentice-Hall, Englewood Cliffs, New Jersey.

Haber, F., 1974, *An Introduction to Information and Communication Theory*. Addison-Wesley Publishing Co.

Hancock, J.C., 1961, *An Introduction to the Principles of Communication Theory*. McGraw-Hill, New York.

Lucky, R.W., Salz, J., and Weldon, E., 1968, *Principles of Data Communication*. McGraw-Hill, New York.

McEliece, R.J., 1977, *The Theory of Information and Coding: A Mathematical Framework for Communication*. Addison-Wesley Publishing Co.

Odenwalder, J.P., 1985, "Error Control." *Data Communications Networks, and Systems*, Chap. 10., T. Bartee, ed., Howard W. Sams & Co., Indianapolis, Indiana.

Peterson, W.W., 1961, *Error-Correcting Codes*. The MIT Press, Cambridge, Massachusetts.

Proakis, J.G., 1983, *Digital Communications*. McGraw-Hill, New York.

Richards, R.K., 1971, *Digital Design*. Wiley-Interscience, New York.

Schwartz, M., 1970, *Information Transmission, Modulation, and Noise*. Second ed., McGraw-Hill, New York.

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

Wiggert, D., 1978, *Error-Control Coding and Applications*. Artech House, Dedham, Massachusetts.

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

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1996	Reference Publication

4. TITLE AND SUBTITLE

Introduction to Forward-Error-Correcting Coding

5. FUNDING NUMBERS

WU-235-04-0D

6. AUTHOR(S)

Jon C. Freeman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135-3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E-7780

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA RP-1367

11. SUPPLEMENTARY NOTES

Responsible person, Jon C. Freeman, organization code 6120, (216) 433-3380.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 32

This publication is available from the NASA Center for Aerospace Information, (301) 621-0390.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This reference publication introduces forward error correcting (FEC) and stresses definitions and basic calculations for use by engineers. The seven chapters include 41 example problems, worked in detail to illustrate points. A glossary of terms is included, as well as an appendix on the Q function. Block and convolutional codes are covered.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Coding	150
	16. PRICE CODE: A07

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	