5

Coding Theory

Author: Kenneth H. Rosen, AT&T Laboratories.

Prerequisites: The prerequisites for this chapter are the basics of logic, set theory, number theory, matrices, and probability. See Sections 1.1, 2.1, 2.2, 3.4–3.7, and 6.1 of *Discrete Mathematics and Its Applications*.

Introduction

The usual way to represent, manipulate, and transmit information is to use bit strings, that is, sequences of zeros and ones. It is extremely difficult, and often impossible, to prevent errors when data are stored, retrieved, operated on, or transmitted. Errors may occur from noisy communication channels, electrical interference, human error, or equipment error. Similarly, errors are introduced into data stored over a long period of time on magnetic tape as the tape deteriorates.

It is particularly important to ensure reliable transmission when large computer files are rapidly transmitted or when data are sent over long distances, such as data transmitted from space probes billions of miles away. Similarly, it is often important to recover data that have degraded while stored on a tape. To guarantee reliable transmission or to recover degraded data, techniques from *coding theory* are used. Messages, in the form of bit strings, are encoded by translating them into longer bit strings, called *codewords*. A set of codewords is called a *code*. As we will see, we can detect errors when we use certain codes. That is, as long as not too many errors have been made, we can determine whether one or more errors have been introduced when a bit string was transmitted. Furthermore, when codes with more redundancy are used, we can correct errors. That is, as long as not too many errors were introduced in transmission, we can recover the codeword from the bit string received when this codeword was sent.

Coding theory, the study of codes, including error detecting and error correcting codes, has been studied extensively for the past forty years. It has become increasingly important with the development of new technologies for data communications and data storage. In this chapter we will study both error detecting and error correcting codes. We will introduce an important family of error correcting codes. To go beyond the coverage in this chapter and to learn about the latest applications of coding theory and the latest technical developments, the reader is referred to the references listed at the end of the chapter.

Error Detecting Codes

A simple way to detect errors when a bit string is transmitted is to add a **parity check bit** at the end of the string. If the bit string contains an even number of 1s we put a 0 at the end of the string. If the bit string contains an odd number of 1s we put a 1 at the end of the string. In other words, we encode the message $x_1x_2\ldots x_n$ as $x_1x_2\ldots x_nx_{n+1}$, where the parity check bit x_{n+1} is given by

$$x_{n+1} = (x_1 + x_2 + \ldots + x_n) \textbf{ mod } 2.$$

Adding the parity check bit guarantees that the number of 1s in the extended string must be even. It is easy to see that the codewords in this code are bit strings with an even number of 1s.

Note that when a parity check bit is added to a bit string, if a single error is made in the transmission of a codeword, the total number of 1s will be odd. Consequently, this error can be detected. However, if two errors are made, these errors cannot be detected, since the total number of 1s in the extended string with two errors will still be even. In general, any odd number of errors can be detected, while an even number of errors cannot be detected.

Example 1 Suppose that a parity check bit is added to a bit string before it is transmitted. What can you conclude if you receive the bit strings 1110011 and 10111101 as messages?

Solution: Since the string 1110011 contains an odd number of 1s, it cannot be a valid codeword (and must, therefore, contain an odd number of errors).

On the other hand, the string 10111101 contains an even number of 1s. Hence it is either a valid codeword or contains an even number of errors. □

Another simple way to detect errors is to repeat each bit in a message twice, as is done in the following example.

Example 2 Encode the bit string 011001 by repeating each bit twice.

Solution: Repeating each bit twice produces the codeword 001111000011. □

What errors can be detected when we repeat each bit of a codeword twice? Since the codewords are those bit strings that contain pairs of matching bits, that is, where the first two bits agree, the third and fourth bits agree, and so on, we can detect errors that change no more than one bit of each pair of these matching bits. For example, we can detect errors in the second bit, the third bit, and the eighth bit of when codewords have eight bits (such as detecting that 01101110, received when the codeword 00001111 was sent, has errors). On the other hand, we cannot detect an error when the third and fourth bit are both changed (such as detecting that 00111111, received when the codeword 00001111 was sent, has errors).

So far we have discussed codes that can be used to detect errors. When errors are detected, all we can do to obtain the correct codeword is to ask for retransmission and hope that no errors will occur when this is done. However, there are more powerful codes that can not only detect but can also correct errors. We now turn our attention to these codes, called error correcting codes.

Error Correcting Codes

We have seen that when redundancy is included in codewords, such as when a parity check bit is added to a bit string, we can detect transmission errors. We can do even better if we include more redundancy. We will not only be able to detect errors, but we will also be able to correct errors. More precisely, if sufficiently few errors have been made in the transmission of a codeword, we can determine which codeword was sent. This is illustrated by the following example.

Example 3 To encode a message we can use the *triple repetition code*. We repeat a message three times. That is, if the message is $x_1x_2x_3$, we encode it as $x_1x_2x_3x_4x_5x_6x_7x_8x_9$ where $x_1 = x_4 = x_7$, $x_2 = x_5 = x_8$, and $x_3 = x_6 = x_9$. The valid codewords are 000000000, 001001001, 010010010, 011011011, 100100100, 101101101, 110110110, and 111111111.

We decode a bit string, which may contain errors, using the *simple majority rule*. For example, to determine x_1, we look at x_1, x_4, and x_7. If two or three of these bits are 1, we conclude that $x_1 = 1$. Otherwise, if two or three of these bits are 0, we conclude that $x_1 = 0$. In general, we look at the three bits corresponding to each bit in the original message. We decide that a bit in the message is 1 if a majority of bits in the string received in positions corresponding to this bit are 1s and we decide this bit is a 0 otherwise. Using this procedure, we correctly decode the message as long as at most one error has been made in the bits corresponding to each bit of the original message.

For example, when a triple repetition code is used, if we receive 011111010, we conclude that the message sent was 011. (For instance, we decided that the first bit of the message was 0 since the first bit is 0, the fourth bit is 1, and the seventh bit is 0, leading us to conclude that the fourth bit is wrong.) $\square$

To make the ideas introduced by the triple repetition code more precise we need to introduce some ideas about the *distance* between codewords and the probability of errors. We will develop several important concepts before returning to error correcting codes.

Hamming Distance

There is a simple way to measure the distance between two bit strings. We look at the number of positions in which these bit strings differ. This approach was used by Richard Hamming* in his fundamental work in coding theory.

Definition 1 The *Hamming distance* $d(\mathbf{x},\mathbf{y})$ between the bit strings $\mathbf{x} = x_1x_2\ldots x_n$ and $\mathbf{y} = y_1y_2\ldots y_n$ is the number of positions in which these strings differ, that is, the number of i $(i = 1, 2, \ldots, n)$ for which $x_i \neq y_i$. $\square$

Note that the Hamming distance between two bit strings equals the number of changes in individual bits needed to change one of the strings into the other.

* Richard Hamming (1915–1998) was one of the founders of modern coding theory. He was born in Chicago and received his B.S. from the University of Chicago, his M.A. from the University of Nebraska, and his Ph.D. from the University of Illinois in 1942. He was employed by the University of Illinois from 1942 to 1944 and the University of Louisville from 1944 to 1945. From 1945 until 1946 he was on the staff of the Manhattan Project in Los Alamos. He joined Bell Telephone Laboratories in 1946, where he worked until 1976. His research included work in the areas of coding theory, numerical methods, statistics, and digital filtering. Hamming joined the faculty of the Naval Postgraduate School in 1976. Among the awards he won are the Turing Prize from the ACM and the IEEE Hamming Medal (named after him).

We will find this observation useful later.

Example 4 Find the Hamming distance between the bit strings 01110 and 11011 and the Hamming distance between the bit strings 00000 and 11111.

Solution: Since 01110 and 11011 differ in their first, third, and fifth bits, $d(01110, 11011) = 3$. Since 00000 and 11111 differ in all five bits, we conclude that $d(00000, 11111) = 5$. □

The Hamming distance satisfies all the properties of a *distance function* (or *metric*), as the following theorem demonstrates.

Theorem 1 Let $d(\mathbf{x}, \mathbf{y})$ represent the Hamming distance between the bit strings $\mathbf{x}$ and $\mathbf{y}$ of length n. Then

(i) $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}$, $\mathbf{y}$

(ii) $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$

(iii) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}$, $\mathbf{y}$

(iv) $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$ for all $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$.

Proof: Properties (i), (ii), and (iii) follow immediately from the definition of the Hamming distance. To prove (iv), we use the fact that $d(\mathbf{x}, \mathbf{y})$ is the number of changes of bits required to change $\mathbf{x}$ into $\mathbf{y}$. Note that for every string $\mathbf{z}$ of length n the number of changes needed to change $\mathbf{x}$ into $\mathbf{y}$ does not exceed the number of changes required to change $\mathbf{x}$ into $\mathbf{z}$ and to then change $\mathbf{z}$ into $\mathbf{y}$. ■

How can the Hamming distance be used in decoding? In particular, suppose that when a codeword $\mathbf{x}$ from a code C is sent, the bit string $\mathbf{y}$ is received. If the transmission was error-free, then $\mathbf{y}$ would be the same as $\mathbf{x}$. But if errors were introduced by the transmission, for instance by a noisy line, then $\mathbf{y}$ is not the same as $\mathbf{x}$. How can we correct errors, that is, how can we recover $\mathbf{x}$?

One approach would be to compute the Hamming distance between $\mathbf{y}$ and each of the codewords in C. Then to decode $\mathbf{y}$, we take the codeword of minimum Hamming distance from $\mathbf{y}$, if such a codeword is unique. If the distance between the closest codewords in C is large enough and if sufficiently few errors were made in transmission, this codeword should be $\mathbf{x}$, the codeword sent. This type of decoding is called *nearest neighbor decoding*.

Example 5 Use nearest neighbor decoding to determine which code word was sent from the code $C = \{0000, 1110, 1011\}$ if 0110 is received.

Solution: We first find the distance between 0110 and each of the codewords. We find that

$$d(0000, 0110) = 2,$$

$$d(1110, 0110) = 1,$$

$$d(1011, 0110) = 3.$$

Since the closest codeword to 0110 is 1110, we conclude that 1110 was the codeword sent. □

Will nearest neighbor decoding produce the most likely codeword that was sent from a binary string that was received? It is not hard to see that it will if each bit sent has the same probability p of being received incorrectly and $p < 1/2$. We call a transmission channel with this property a *binary symmetric channel*. Such a channel is displayed in Figure 1.

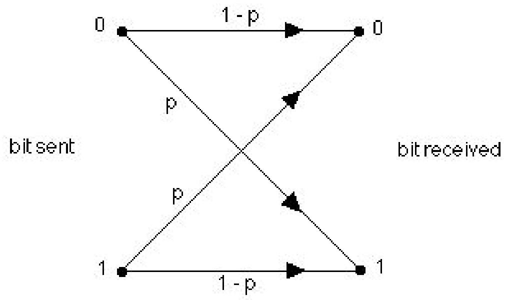

Figure 1. A binary symmetric channel.

Example 6 Suppose that when a bit is sent over a binary symmetric channel the probability it is received incorrectly is 0.01. What is the probability that the bit string 100010 is received when the bit string 000000 is sent?

Solution: Since the probability a bit is received incorrectly is 0.01, the probability that a bit is received correctly is $1 - 0.01 = 0.99$. For 100010 to be received, when 000000 is sent, it is necessary for the first and fifth bits to be received incorrectly and the other four bits to be received correctly. The probability that this occurs is

$$(0.99)^4(0.01)^2 = 0.000096059601.$$ □

We will now show that nearest neighbor decoding gives us the most likely codeword sent, so that it is also *maximum likelihood decoding*.

Theorem 2 Suppose codewords of a binary code C are transmitted using a binary symmetric channel. Then, nearest neighbor decoding of a bit string received produces the most likely codeword sent.

Proof: To prove this theorem we first need to find the probability that when a codeword of length n is sent, a bit string with k errors in specified positions is received. Since the probability each bit is received correctly is $1-p$ and the probability each bit is received in error is p, it follows that the probability of k errors in specified positions is $p^k(1-p)^{n-k}$. Since $p < 1/2$ and $1-p > 1/2$, it follows that

$$p^i(1-p)^{n-i} > p^j(1-p)^{n-j}$$

whenever $i < j$. Hence, if $i < j$, the probability that a bit string with i specified errors is received is greater than the probability that a bit string with j specified errors is received. Since is more likely that errors were made in fewer specified positions when a codeword was transmitted, nearest neighbor decoding produces the most likely codeword. ■

The Hamming distance between codewords in a binary code determines its ability to detect and/or correct errors. We need to make the following definition before introducing two key theorems relating to this ability.

Definition 2 The *minimum distance* of a binary code C is the smallest distance between two distinct codewords, that is,

$$d(C) = \min\{d(\mathbf{x}, \mathbf{y}) | \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\}.$$ □

Example 7 Find the minimum distance of the code

$$C = \{00000, 01110, 10011, 11111\}.$$

Solution: To compute the minimum distance of this code we will find the distance between each pair of codewords and then find the smallest such distance. We have $d(00000, 01110) = 3$, $d(00000, 10011) = 3$, $d(00000, 11111) = 5$, $d(01110, 10011) = 4$, $d(01110, 11111) = 2$, and $d(10011, 11111) = 2$. We see that the minimum distance of C is 2. □

Example 8 Find the minimum distance of the code

$$C = \{000000, 111111\}.$$

Solution: Since there are only two codewords and $d(000000, 111111) = 6$, the minimum distance of this code is 6. □

The minimum distance of a code tells us how many errors it can detect and how many errors it can correct, as the following two theorems show.

Theorem 3 A binary code C can detect up to k errors in any codeword if and only if $d(C) \geq k + 1$.

Proof: Suppose that C is a binary code with $d(C) \geq k + 1$. Suppose that a codeword $\mathbf{x}$ is transmitted and is received with k or fewer errors. Since the minimum distance between codewords is at least $k + 1$, the bit string received cannot be another codeword. Hence, the receiver can detect these errors.

Now suppose that C can detect up to k errors and that $d(C) \leq k$. Then there are two codewords in C that differ in no more than k bits. It is then possible for k errors to be introduced when one of these codewords is transmitted so that the other codeword is received, contradicting the fact that C can detect up to k errors. ■

Theorem 4 A binary code C can correct up to k errors in any codeword if and only if $d(C) \geq 2k + 1$.

Proof: Suppose that C is a binary code with $d(C) \geq 2k + 1$. Suppose that a codeword $\mathbf{x}$ is transmitted and received with k or fewer errors as the bit string $\mathbf{z}$, so that $d(\mathbf{x}, \mathbf{z}) \leq k$. To see that C can correct these errors, note that if $\mathbf{y}$ is a codeword other than $\mathbf{x}$, then $d(\mathbf{z}, \mathbf{y}) \geq k + 1$. To see this note that if $d(\mathbf{z}, \mathbf{y}) \leq k$, then by the triangle inequality $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y}) \leq k + k = 2k$, contradicting the assumption that $d(C) \geq 2k + 1$.

Conversely, suppose that C can correct up to k errors. If $d(C) \leq 2k$, then there are two codewords that differ in $2k$ bits. Changing k of the bits in one of these codewords produces a bit string that differs from each of these two codewords in exactly k positions, thus making it impossible to correct these k errors. ■

Example 9 Let C be the code $\{00000000, 11111000, 01010111, 10101111\}$. How many errors can C detect and how many can it correct?

Solution: Computing the distance between codewords shows that the minimum distance of C is 5. By Theorem 3, it follows that C can detect up to $5 - 1 = 4$ errors. For example, when we use C to detect errors, we can detect the four errors made in transmission when we receive 11110000 when the codeword 00000000 was sent.

By Theorem 4, it follows that C can correct up to $\lfloor (5-1)/2 \rfloor = 2$ errors. For example, when we use C to correct errors, we can correct the two errors introduced in transmission when we receive 11100000 when the codeword 11111000 was sent. □

Perfect Codes

To allow error correction we want to make the minimum distance between codewords large. But doing so limits how many codewords are available. Here we will develop a bound on the number of codewords in a binary code with a given minimum distance.

Lemma 1 Suppose $\mathbf{x}$ is a bit string of length n and that k is a nonnegative integer not exceeding n. Then there are

$$C(n,0) + C(n,1) + \cdots + C(n,k).$$

bit strings $\mathbf{y}$ of length n such that $d(\mathbf{x},\mathbf{y}) \le k$ (where d is the Hamming distance).

Proof: Let i be a nonnegative integer. The number of bit strings $\mathbf{y}$ with $d(\mathbf{x},\mathbf{y}) = i$ equals the number of ways to select the i locations where $\mathbf{x}$ and $\mathbf{y}$ differ. This can be done in $C(n,i)$ ways. It follows that there are

$$C(n,0) + C(n,1) + \cdots + C(n,k)$$

bit strings such that $d(\mathbf{x},\mathbf{y}) \le k$. □

We can describe the statement in Lemma 1 in geometric terms. By the **sphere of radius** k centered at $\mathbf{x}$ we mean the set of all bit strings $\mathbf{y}$ such that $d(\mathbf{x},\mathbf{y}) \le k$. Lemma 1 says that there are exactly $\sum_{i=0}^{k} C(n,i)$ bit stings in the sphere of radius k centered at $\mathbf{x}$.

Lemma 2 Let C be a binary code containing codewords of length n and let $d(C) = 2k+1$. Then given a bit string $\mathbf{y}$ of length n, there is at most one codeword $\mathbf{x}$ such that $\mathbf{y}$ is in the sphere of radius k centered at $\mathbf{x}$.

Proof: Suppose that $\mathbf{y}$ is in the sphere of radius k centered at two different codewords $\mathbf{x}_1$ and $\mathbf{x}_2$. Then $d(\mathbf{x}_1,\mathbf{y}) \le k$ and $d(\mathbf{x}_2,\mathbf{y}) \le k$. By the triangle inequality for the Hamming distance this implies that

$$d(\mathbf{x}_1,\mathbf{x}_2) \le d(\mathbf{x}_1,\mathbf{y}) + d(\mathbf{x}_2,\mathbf{y}) \le k + k = 2k,$$

contradicting the fact that the minimum distance between codewords is $2k+1$. □

We can now give a useful bound on how many codewords can be in a code consisting of n-tuples that can correct a specified number of errors.

Theorem 5 The Sphere Packing or (Hamming) Bound Suppose that C is a code of bit strings of length n with $d(C) = 2k+1$. Then $|C|$, the number of codewords in C, cannot exceed

$$\frac{2^n}{C(n,0) + C(n,1) + \cdots + C(n,k).}$$

Proof: There are 2^n bit strings of length n. By Lemma 1 the sphere of radius k centered at a codeword $\mathbf{x}$ contains

$$C(n,0) + C(n,1) + \cdots + C(n,k)$$

bit strings. Since no bit string can be in two such spheres (by Lemma 2), it follows that the number of bit strings of length n is at least as large as the number of codewords times the number of bit strings in each such sphere. Hence,

$$2^n \geq |C|[C(n,0) + C(n,1) + \cdots + C(n,k)].$$

We obtain the inequality we want by dividing by the second factor on the right-hand side of this inequality (and writing the inequality with the smaller term first). ■

Example 10 Find an upper bound for the number of codewords in a code C where codewords are bit strings of length seven and the minimum distance between codewords is three.

Solution: The minimum distance between codewords is $3 = 2k+1$, so that $k = 1$. Hence the sphere packing bound shows that there are no more than

$$2^7/[C(7,0) + C(7,1)] = 128/8 = 16$$

codewords in such a code. □

The sphere packing bound gives us an upper bound for the number of codewords in a binary code with a given minimum distance where codewords are bit strings of length n. The codes that actually achieve this upper bound, that is, that have the most codewords possible, are of special interest because they are the most efficient error correcting codes. Such codes are known as *perfect codes*.

Example 11 Show that the code consisting of just two codewords 00000 and 11111 is a perfect binary code.

Solution: The minimum distance between codewords in this code is 5. The sphere packing bound states that there are at most

$$2^5/[C(5,0)+C(5,1)+C(5,2)] = 32/16 = 2$$

codewords in a code consisting of 5-tuples with minimum distance 5. Since there are 2 codewords in this code, it is a perfect binary code. □

The code in Example 11 is called a *trivial perfect code* since it only consists of the two codewords, one containing only 0s and the other containing only 1s. As Exercise 8 demonstrates, when n is an odd positive integer there are trivial perfect codes consisting of the two codewords which are bit strings of length n consisting of all 0s and of all 1s. Finding perfect binary codes different from the trivial codes has been one of the most important problems in coding theory. In the next section we will introduce a class of perfect binary codes known as Hamming codes.

Generator Matrices

Before describing Hamming codes, we need to generalize the concept of a parity check bit. When we use a parity check bit, we encode a message $x_1x_2\ldots x_k$ as $x_1x_2\ldots x_kx_{k+1}$ where $x_{k+1} = (x_1 + x_2 + \cdots + x_k) \textbf{ mod } 2$. To generalize this notion, we add more than one check bit. More precisely, we encode a message $x_1x_2\ldots x_k$ as $x_1x_2\ldots x_kx_{k+1}\ldots x_n$, where the last $n-k$ bits x_{k+1},...,x_n, are *parity check bits*, obtained from the k bits in the message. We will describe how these parity check bits are specified.

Consider a k-bit message $x_1x_2\cdots x_k$ as a $1\times k$ matrix $\mathbf{x}$. Let $\mathbf{G}$ be a $k\times n$ matrix that begins with the $k\times k$ identity matrix $\mathbf{I}_k$. That is, $\mathbf{G} = (\mathbf{I}_k|\mathbf{A})$, where $\mathbf{A}$ is a $k\times(n-k)$ matrix, known as a *generator matrix*. We encode $\mathbf{x}$ as $E(\mathbf{x}) = \mathbf{xG}$, where we do arithmetic modulo 2. Coding using a parity check bit and using the triple repetition code are special cases of this technique, as illustrated in Examples 12 and 13.

Example 12 We can represent encoding by adding a parity check bit to a three-bit message as $E(\mathbf{x}) = \mathbf{xG}$, where

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

Note that to obtain G we add a column of 1s to $\mathbf{I}_3$, the 3×3 identity matrix. That is, $G = (\mathbf{I}_3|\mathbf{A})$, where

$$\mathbf{A} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \qquad \square$$

Example 13 We can represent encoding using the triple repetition code for three-bit messages as $E(\mathbf{x}) = \mathbf{xG}$, where

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

Note that $\mathbf{G}$ is formed by repeating the identity matrix of order three, $\mathbf{I}_3$, three times, that is,

$$\mathbf{G} = (\mathbf{I}_3|\mathbf{I}_3|\mathbf{I}_3). \qquad \square$$

We now consider an example which we will use to develop some important ideas.

Example 14 Suppose that

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

that is, $\mathbf{G} = (\mathbf{I}_3|\mathbf{A})$, where

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

What are the codewords in the code generated by this generator matrix?

Solution: We encode each of the eight three-bit messages $\mathbf{x} = x_1x_2x_3$ as $E(\mathbf{x}) = \mathbf{x}\mathbf{G}$. This produces the codewords 000000, 001101, 010110, 011011, 100111, 101010, 110001, and 111100. For example, we get the third of these by computing

$$E(010) = (0\ 1\ 0)\mathbf{G} = (0\ 1\ 0)\begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} = (0\ 1\ 0\ 1\ 1\ 0). \qquad \square$$

It is easy to see that we can find the codewords in a binary code generated by a generator matrix G by taking all possible linear combinations of the rows of G (since arithmetic is modulo 2, this means all sums of subsets of the set of rows of G). The reader should verify this for codewords in the code in Example 14.

It is easy to see that the binary codes formed using generator matrices have the property that the sum of any two codewords is again a codeword. That is, they are **linear codes**. To see this, suppose that $\mathbf{y}_1$ and $\mathbf{y}_2$ are codewords generated by the generator matrix $\mathbf{G}$. Then there are bit strings $\mathbf{x}_1$ and $\mathbf{x}_2$ such that $E(\mathbf{x}_1) = \mathbf{y}_1$ and $E(\mathbf{x}_2) = \mathbf{y}_2$, where $E(\mathbf{x}) = \mathbf{x}\mathbf{G}$. It follows that $\mathbf{y}_1 + \mathbf{y}_2$ is also a codeword since $E(\mathbf{x}_1 + \mathbf{x}_2) = \mathbf{y}_1 + \mathbf{y}_2$. (Here we add bit strings by adding their components in the same positions using arithmetic modulo 2.)

We will see that there is an easy way to find the minimum distance of a linear code. Before we see this, we need to make the following definition.

Definition 3 The *weight* of a codeword $\mathbf{x}$, denoted by $w(\mathbf{x})$, in a binary code is the number of 1s in this codeword. $\square$

Example 15 Find the weights of the codewords 00000, 10111, and 11111.

Solution: Counting the number of 1s in each of these codewords we find that $w(00000) = 0$, $w(10111) = 4$, and $w(11111) = 5$. $\square$

Lemma 3 Suppose that $\mathbf{x}$ and $\mathbf{y}$ are codewords in a linear code C. Then $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} + \mathbf{y})$.

Proof: The positions with 1s in them in $\mathbf{x}+\mathbf{y}$ are the positions where $\mathbf{x}$ and $\mathbf{y}$ differ. Hence $d(\mathbf{x}, \mathbf{y}) = w(\mathbf{x} + \mathbf{y})$. $\blacksquare$

We also will need the fact that $\mathbf{0}$, the bit string with all 0s, belongs to a linear code.

Lemma 4 Suppose that C is a nonempty linear code. Then $\mathbf{0}$ is a codeword in C.

Proof: Let $\mathbf{x}$ be a codeword in C. Since C is linear $\mathbf{x}+\mathbf{x}=\mathbf{0}$ belongs to C. ■

Theorem 6 The minimum distance of a linear code C equals the minimum weight of a nonzero codeword in C.

Proof: Suppose that the minimum distance of C is d. Then there are codewords $\mathbf{x}$ and $\mathbf{y}$ such that $d(\mathbf{x},\mathbf{y}) = d$. By Lemma 3 it follows that $w(\mathbf{x}+\mathbf{y}) = d$. Hence w, the minimum weight of a codeword, is no larger than d.

Conversely, suppose that the codeword $\mathbf{x}$ is a nonzero codeword of minimum weight. Then using Lemma 4 we see that $w = w(\mathbf{x}) = w(\mathbf{x}+\mathbf{0}) = d(\mathbf{x},\mathbf{0}) \geq d$. It follows that $w = d$, establishing the theorem. ■

Parity Check Matrices

Note that in Example 14 the bit string $x_1x_2x_3$ is encoded as $x_1x_2x_3x_4x_5x_6$ where $x_4 = x_1+x_2+x_3$, $x_5 = x_1+x_2$, and $x_6 = x_1+x_3$ (here, arithmetic is carried out modulo 2). Because we are doing arithmetic modulo 2, we see that

$$\begin{aligned} x_1+x_2+x_3+x_4 &= 0\\ x_1+x_2+x_5 &= 0\\ x_1+x_3+x_6 &= 0. \end{aligned}$$

Furthermore, it is easy to see that $x_1x_2x_3x_4x_5x_6$ is a codeword if and only if it satisfies this system of equations.

We can express this system of equations as

$$\begin{pmatrix} 1&1&1&1&0&0\\ 1&1&0&0&1&0\\ 1&0&1&0&0&1 \end{pmatrix} \begin{pmatrix} x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6 \end{pmatrix} = \begin{pmatrix} 0\\ 0\\ 0 \end{pmatrix},$$

that is,

$$\mathbf{HE}(\mathbf{x})^t = \mathbf{0},$$

where $\mathbf{E}(\mathbf{x})^t$ is the transpose of $\mathbf{E}(\mathbf{x})$ and $\mathbf{H}$, the parity check matrix, is given by

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

Note that $\mathbf{H} = (\mathbf{A}^t|\mathbf{I}_{n-k})$. With this notation we see that $\mathbf{x} = x_1x_2x_3x_4x_5x_6$ is a codeword if and only if $\mathbf{Hx}^t = \mathbf{0}$, since checking this equation is the same as checking whether the parity check equations hold.

In general, suppose that $\mathbf{G}$ is a $k \times n$ generator matrix with

$$\mathbf{G} = (\mathbf{I}_k|\mathbf{A}),$$

where $\mathbf{A}$ is a $k \times (n-k)$ matrix. To G we associate the *parity check matrix* $\mathbf{H}$, where

$$\mathbf{H} = (\mathbf{A}^t|\mathbf{I}_{n-k}).$$

Then $\mathbf{x}$ is a codeword if and only if $\mathbf{Hx}^t = \mathbf{0}$. Note that from a generator matrix $\mathbf{G}$ we can find the associated parity check matrix $\mathbf{H}$, and conversely, given a parity check matrix $\mathbf{H}$, we can find the associated generator matrix $\mathbf{G}$. More precisely, note that if $\mathbf{H} = (\mathbf{B}|\mathbf{I}_r)$, then $\mathbf{G} = (\mathbf{I}_{n-r}|\mathbf{B}^t)$.

We have seen that the parity check matrix can be used to detect errors. That is, to determine whether $\mathbf{x}$ is a codeword we check whether

$$\mathbf{Hx}^t = \mathbf{0}.$$

Not only can the parity check matrix be used to detect errors, but when the columns of this matrix are distinct and are all nonzero, it also can be used to correct errors. Under these assumptions, suppose that the codeword $\mathbf{x}$ is sent and that $\mathbf{y}$ is received, which may or may not be the same as $\mathbf{x}$. Write $\mathbf{y} = \mathbf{x} + \mathbf{e}$, where $\mathbf{e}$ is an *error string*. (We have $\mathbf{e} = 0$ if no errors arose in the transmission). In general, the error string $\mathbf{e}$ has 1s in the positions where $\mathbf{y}$ differs from $\mathbf{x}$ and 0s in all other positions. Now suppose that only one error has been introduced when $\mathbf{x}$ was transmitted. Then $\mathbf{e}$ is a bit string that has only one nonzero bit which is in the position where $\mathbf{x}$ and $\mathbf{y}$ differ, say position j. Since $\mathbf{Hx}^t = \mathbf{0}$, it follows that

$$\begin{aligned}\mathbf{Hy}^t &= \mathbf{H}(\mathbf{x}^t + \mathbf{e}) \\ &= \mathbf{Hx}^t + \mathbf{e}^t \\ &= \mathbf{e}^t \\ &= \mathbf{c}_j\end{aligned}$$

where $\mathbf{c}_j$ is the jth column of $\mathbf{H}$.

Hence, if we receive $\mathbf{y}$ and assume that no more than one error is present, we can find the codeword $\mathbf{x}$ that was sent by computing $\mathbf{Hy}^t$. If this is zero, we know that $\mathbf{y}$ is the codeword sent. Otherwise, it will equal the jth column of $\mathbf{H}$ for some integer j. This means that the jth bit of $\mathbf{y}$ should be changed to produce $\mathbf{x}$.

Example 16 Use the parity check matrix to determine which codeword from the code in Example 14 was sent if 001111 was received. Assume that at most one error was made.

Solution: We find that

$$\mathbf{H}\mathbf{y}^t = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

This is the fifth column of **H**. If follows that the fifth bit of 001111 is incorrect. Hence the code word sent was 001101. □

Hamming Codes

We can now define the Hamming codes. We define them using parity check matrices.

Definition 4 A *Hamming code of order* r where r is a positive integer, is a code generated when we take as parity check matrix **H** an $r \times (2^r - 1)$ matrix with columns that are all the $2^r - 1$ nonzero bit strings of length r in any order such that the last r columns form the identity matrix. □

Interchanging the order of the columns leads to what is known as an *equivalent code*. For details on equivalence of codes the reader is referred to the references at the end of this chapter.

Example 17 Find the codewords in a Hamming code of order 2.

Solution: The parity check matrix of this code is

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

We have $\mathbf{H} = (\mathbf{B}|\mathbf{I}_2)$ where $\mathbf{B} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Hence, the generator matrix **G** of this code equals $\mathbf{G} = (\mathbf{I}_{3-2}|\mathbf{B}^t) = (1\ 1\ 1)$. Since the codewords are linear combinations of the rows of G, we see that this code has two codewords, 000 and 111. This is the linear repetition code of order 3. □

Example 18 Find the codewords in a Hamming code of order 3.

Solution: For the parity check matrix of this code we use

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

We have $\mathbf{H} = (\mathbf{B}|\mathbf{I}_3)$ where

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}.$$

Hence the generator matrix $\mathbf{G}$ of this code equals

$$\mathbf{G} = (\mathbf{I}_{7-3}|\mathbf{B}^t) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The 16 codewords in this code C can be found by taking all possible sums of the rows of $\mathbf{G}$. We leave this as an exercise at the end of the chapter. □

To show that the Hamming codes are perfect codes we first need to establish two lemmas.

Lemma 5 A Hamming code of order r contains 2^{n-r} codewords where $n = 2^r - 1$.

Proof: The parity check matrix of the Hamming code is an $r \times n$ matrix. It follows that the generator matrix for this code is a $(n-r) \times n$ matrix. Recall that the codewords are the linear combinations of the rows. As the reader can show, no two linear combinations of the rows are the same. Since there are 2^{n-r} different linear combinations of row, there are 2^{n-r} different codewords in a Hamming code of order r. ■

Lemma 6 The minimum distance of a Hamming code of order r is 3 whenever r is a positive integer.

Proof: The parity check matrix $\mathbf{H}_r$ has columns which are all nonzero and no two of which are the same. Hence, from our earlier discussion, a Hamming code of order r can correct single errors. By Theorem 3 we conclude that the minimum distance of this code is at least 3. Among the columns of $\mathbf{H}_r$ are the following three columns:

$$\mathbf{c}_1 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \mathbf{c}_2 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad \mathbf{c}_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Note that $\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3 = \mathbf{0}$. Let $\mathbf{x}$ be the bit string with 1 in the positions of these columns and zero elsewhere. Then $\mathbf{H}\mathbf{x}^t = \mathbf{0}$, since it is $\mathbf{c}_1 + \mathbf{c}_2 + \mathbf{c}_3$. It follows that $\mathbf{x}$ is a codeword. Since $w(\mathbf{x}) = 3$, by Theorem 6 it follows that the minimum distance is no more than 3. We conclude that the minimum distance of a Hamming code of order r is 3. ■

Theorem 7 The Hamming code of order r is a perfect code.

Proof: Let $n = 2^r - 1$. By Lemma 5 a Hamming code of order r contains 2^{n-r} codewords, each of which is an n-tuple. By Lemma 6 the minimum distance of the Hamming code of order r is 3. We see that this code achieves the maximum number of codewords allowed by the sphere packing bound. To see this, note that

$$2^{n-r}(1 + C(n,1)) = 2^{n-r}(1 + n) = 2^{n-r}(1 + 2^r - 1) = 2^n.$$

This is the upper bound of the sphere-packing bound. hence a Hamming code of order r is perfect. ■

By Theorem 7 we see that the Hamming codes are examples of perfect codes. The study of perfect codes has been one of the most important areas of research in coding theory and has lead to the development of many important results. See the references at the end of the chapter to learn about what is known about perfect codes.

Summary

In this chapter we have studied how codes can be used for error detection and error correction. We have introduced an important class of codes known as the Hamming codes. However, we have only touched the surface of a fascinating and important subject that has become extremely important for modern computing and communications. The interested reader can consult the references listed at the end of this chapter to learn about many other classes of codes that have practical applications.

For example, pictures of the planets taken by space probes have been encoded using powerful codes, such as a code known as the *Reed-Muller* code (see [5] for details). This code has been used to encode the bit string of length 6 representing the brightness of each pixel of an image by a bit string of length 32. This Reed-Muller code consists of 64 codewords, each a bit string of length 32, with minimum distance 16. Another interesting example is the use of a family of codes known as *Reed-Solomon* codes used in digital audio recording (see [5] for details). Finally, many concepts and techniques from both linear algebra and abstract algebra are used in coding theory. Studying coding theory may convince the skeptical reader about the applicability of some of the more abstract areas of mathematics.

Suggested Readings

1. E. Berlekamp, editor, *Key Papers in the Development of Coding Theory*, IEEE Press, New York, 1974.
2. R. Hamming, *Coding and Information Theory*, 2nd Edition, Prentice Hall, Upper Saddle River, N.J., 1986.
3. R. Hill, *A First Course in Coding Theory*, Oxford University Press, Oxford, 1990.
4. V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 3rd Edition, John Wiley & Sons, Hoboken, N.J., 1998.
5. S. Vanstone and P. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Springer, New York. 1989.

Exercises

1. Could the following bit strings have been received correctly if the last bit is a parity bit?
a) 1000011
b) 111111000
c) 10101010101
d) 110111011100

2. Find the Hamming distance between each of the following pairs of bit strings.

a) 00000,11111
b) 1010101,0011100
c) 000000001,111000000
d) 1111111111,0100100011

3. Suppose the bit string 01011 is sent using a binary symmetric channel where the probability a bit is received incorrectly is 0.01. What is the probability that
a) 01011, the bit string sent, is received?
b) 11011 is received?
c) 01101 is received?
d) 10111 is received?
e) no more than one error is present in the bit string received?

4. How many errors can each of the following binary codes detect and how many can it correct?
a) $\{0000000, 1111111\}$
b) $\{00000, 00111, 10101, 11011\}$
c) $\{00000000, 11111000, 01100111, 100101101\}$

5. Suppose that the probability of a bit error in transmission over a binary symmetric channel is 0.001. What is the probability that when a codeword with eight bits is sent from a code with minimum distance five, the bit string received is decoded as the codeword sent (when nearest neighbor decoding is used)?

⋆6. Show that if the minimum distance between codewords is four it is possible to correct an error in a single bit and to detect two bit errors without correction.

7. Use the sphere packing bound to give an upper bound on the number of codewords in a binary code where codewords are bit strings of length nine and the minimum distance between codewords is five.

8. Show that whenever n is an odd positive integer, the binary code consisting of the two bit strings of length n containing all 0s or all 1s is a perfect code.

9. Suppose that $\mathbf{x}$ and $\mathbf{y}$ are bit strings of length n and m is the number of positions where both $\mathbf{x}$ and $\mathbf{y}$ have 1s. Show that $w(\mathbf{x}+\mathbf{y}) = w(\mathbf{x})+w(\mathbf{y})-2m$.

10. Find the parity check matrix associated with the code formed by adding a parity check bit to a bit string of length 4.

11. Find the parity check matrix associated with the triple repetition code for bit strings of length 3.

12. Suppose that the generator matrix for a binary code is

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

What is the parity check matrix **H** for this code?

13. Suppose that the parity check matrix for a binary code is

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

What is the generator matrix **G** for this code?

14. Find the 16 codewords in the Hamming code of order 3 described in Example 18.

$\star$**15.** Sometimes, instead of errors, bits are erased during the transmission of a message or from a tape or other storage medium. The position, but not the value, of an erased bit is known. We say that a code C can correct r erasures if a message received with no errors and with no more than r erasures can be corrected to a unique codeword that agrees with the message received in all the positions that were not erased.

a) Show that a binary code of minimum distance d can correct $d - 1$ erasures.

b) Show that a binary code of minimum distance d can correct t errors and r erasures if $d = 2t + r + 1$.

Computer Projects

1. Given a binary code, determine the number of errors that it can detect and the number of errors that it can correct.

2. Given a binary code with minimum distance k, where k is a positive integer, write a program that will detect errors in codewords in as many as $k - 1$ positions and correct errors in as many as $\lfloor (k-1)/2 \rfloor$ positions.