3

Repetition and Hamming codes

The theory of error-correcting codes comes from the need to protect information from corruption during transmission or storage. Take your CD or DVD as an example. Usually, you might convert your music into MP3 files[1] for storage. The reason for such a conversion is that MP3 files are more compact and take less storage space, i.e. they use fewer binary digits (bits) compared with the original format on CD. Certainly, the price to pay for a smaller file size is that you will suffer some kind of distortion, or, equivalently, losses in audio quality or fidelity. However, such loss is in general indiscernible to human audio perception, and you can hardly notice the subtle differences between the uncompressed and compressed audio signals. The compression of digital data streams such as audio music streams is commonly referred to as *source coding*. We will consider it in more detail in Chapters 4 and 5.

What we are going to discuss in this chapter is the opposite of compression. After converting the music into MP3 files, you might want to store these files on a CD or a DVD for later use. While burning the digital data onto a CD, there is a special mechanism called *error control coding* behind the CD burning process. Why do we need it? Well, the reason is simple. Storing CDs and DVDs inevitably causes small scratches on the disk surface. These scratches impair the disk surface and create some kind of lens effect so that the laser reader might not be able to retrieve the original information correctly. When this happens, the stored files are corrupted and can no longer be used. Since the scratches are inevitable, it makes no sense to ask the users to keep the disks in perfect condition, or discard them once a perfect read-out from the disk becomes impossible. Therefore, it would be better to have some kind of engineering mechanism to protect the data from being compromised by minor scratches.

[1] MP3 stands for *MPEG-2 audio layer 3*, where MPEG is the abbreviation for *moving picture experts group*.

We use error-correcting codes to accomplish this task. Error-correcting codes are also referred to as *channel coding* in general.

First of all, you should note that it is impossible to protect the stored MP3 files from impairment without increasing the file size. To see this, say you have a binary data stream $\mathbf{s}$ of length k bits. If the protection mechanism were not allowed to increase the length, after endowing $\mathbf{s}$ with some protection capability, the resulting stream $\mathbf{x}$ is at best still of length k bits. Then the whole protection process is nothing but a mapping from a k-bit stream to another k-bit stream. Such mapping is, at its best, one-to-one and onto, i.e. a bijection, since if it were not a bijection, it would not be possible to recover the original data. On the other hand, because of the bijection, when the stored data stream $\mathbf{x}$ is corrupted, it is impossible to recover the original $\mathbf{s}$. Therefore, we see that the protection process (henceforth we will refer to it as an *encoding process*) must be an injection, meaning $\mathbf{x}$ must have length larger than k, say n, so that when $\mathbf{x}$ is corrupted, there is a chance that $\mathbf{s}$ may be recovered by using the extra $(n-k)$ bits we have used for storing extra information.

How to encode efficiently a binary stream of length k with minimum $(n-k)$ extra bits added so that the length k stream $\mathbf{s}$ is well protected from corruption is the major concern of error-correcting codes. In this chapter, we will briefly introduce two kinds of error-correcting codes: the *repetition code* and the *Hamming code*. The repetition code, as its name suggests, simply repeats information and is the simplest error-protecting/correcting scheme. The Hamming code, developed by Richard Hamming when he worked at Bell Labs in the late 1940s (we will come back to this story in Section 3.3.1), on the other hand, is a bit more sophisticated than the repetition code. While the original Hamming code is actually not that much more complicated than the repetition code, it turns out to be optimal in terms of sphere packing in some high-dimensional space. Specifically, this means that for certain code length and error-correction capability, the Hamming code actually achieves the maximal possible rate, or, equivalently, it requires the fewest possible extra bits.

Besides error correction and data protection, the Hamming code is also good in many other areas. Readers who wish to know more about these subjects are referred to Chapter 8, where we will briefly discuss two other uses of the Hamming code. We will show in Section 8.1 how the Hamming code relates to a geometric subject called *projective geometry*, and in Section 8.2 how the Hamming code can be used in some mathematical games.

3.1 Arithmetics in the binary field

Prior to introducing the codes, let us first study the arithmetics of binary operations (see also Section 2.1). These are very important because the digital data is binary, i.e. each binary digit is either of value 0 or 1, and the data will be processed in a binary fashion. By binary operations we mean binary addition, subtraction, multiplication, and division. The binary addition is a modulo-2 addition, i.e.

$$\begin{aligned} 0+0&=0,\\ 1+0&=1,\\ 0+1&=1,\\ 1+1&=0. \end{aligned} \tag{3.1}$$

The only difference between binary and usual additions is the case of $1+1$. Usual addition would say $1+1=2$. But since we are working with modulo-2 addition, meaning the sum is taken as the remainder when divided by 2, the remainder of 2 divided by 2 equals 0, hence we have $1+1=0$ in binary arithmetics.

By moving the second operand to the right of these equations, we obtain subtractions:

$$\begin{aligned} 0&=0-0,\\ 1&=1-0,\\ 0&=1-1,\\ 1&=0-1. \end{aligned} \tag{3.2}$$

Further, it is interesting to note that the above equalities also hold if we replace "$-$" by "$+$". Then we realize that, in binary, subtraction is the same as addition. This is because the remainder of -1 divided by 2 equals 1, meaning -1 is considered the same as 1 in binary. In other words,

$$a-b=a+(-1)\times b=a+(1)\times b=a+b. \tag{3.3}$$

Also, it should be noted that the above implies

$$a-b=b-a=a+b \tag{3.4}$$

in binary, while this is certainly false for real numbers.

Multiplication in binary is the same as usual, and we have

$$\begin{aligned} 0\times 0&=0,\\ 1\times 0&=0,\\ 0\times 1&=0,\\ 1\times 1&=1. \end{aligned} \tag{3.5}$$

The same holds also for division.

Exercise 3.1 *Show that the laws of association and distribution hold for binary arithmetics. That is, show that for any $a,b,c\in\{0,1\}$ we have*

$$\begin{aligned} a+b+c&=(a+b)+c=a+(b+c) && \textit{(additive associative law),}\\ a\times b\times c&=(a\times b)\times c=a\times(b\times c) && \textit{(multiplicative associative law),}\\ a\times(b+c)&=(a\times b)+(a\times c) && \textit{(distributive law).} \end{aligned}$$ ◊

Exercise 3.2 *In this chapter, we will use the notation $\stackrel{?}{=}$ to denote a conditional equality, by which we mean that we are unsure whether the equality holds. Show that the condition of $a\stackrel{?}{=}b$ in binary is the same as $a+b\stackrel{?}{=}0$.* ◊

3.2 Three-times repetition code

A binary digit (or *bit* in short) s is to be stored on CD, but it could be corrupted for some reason during read-out. To recover the corrupted data, a straightforward means of protection is to store as many copies of s as possible. For simplicity, say we store three copies. Such a scheme is called the *three-times repetition code*. Thus, instead of simply storing s, we store (s,s,s). To distinguish them, let us denote the first s as x_1 and the others as x_2 and x_3. In other words, we have

$$\begin{cases} x_2=x_3=0 & \text{if } x_1=0,\\ x_2=x_3=1 & \text{if } x_1=1, \end{cases} \tag{3.6}$$

and the possible values of (x_1,x_2,x_3) are (000) and (111).

When you read out the stream (x_1,x_2,x_3) from a CD, you must check whether $x_1=x_2$ and $x_1=x_3$ in order to detect if there was a data corruption. From Exercise 3.2, this can be achieved by the following computation:

$$\begin{cases} \text{data clean} & \text{if } x_1+x_2=0 \text{ and } x_1+x_3=0,\\ \text{data corrupted} & \text{otherwise.} \end{cases} \tag{3.7}$$

For example, if the read-out is $(x_1,x_2,x_3)=(000)$, then you might say the data

is clean. Otherwise, if the read-out shows $(x_1, x_2, x_3) = (0\,0\,1)$ you immediately find $x_1 + x_3 = 1$ and the data is corrupted.

Now say that the probability of writing in 0 and reading out 1 is p, and the same for writing in 1 and reading out 0. You see that a bit is corrupted with probability p and remains clean with probability $(1-p)$. Usually we can assume $p < 1/2$, meaning the data is more likely to be clean than corrupted. In the case of $p > 1/2$, a simple bit-flipping technique of treating the read-out of 1 as 0 and 0 as 1 would do the trick.

Thus, when $p < 1/2$, the only possibilities for data corruption going undetected are the cases when the read-out shows $(1\,1\,1)$ given writing in was $(0\,0\,0)$ and when the read-out shows $(0\,0\,0)$ given writing in was $(1\,1\,1)$. Each occurs with probability[2] $p^3 < 1/8$. Compared with the case when the data is unprotected, the probability of undetectable corruption drops from p to p^3. It means that when the read-out shows either $(0\,0\,0)$ or $(1\,1\,1)$, we are more confident that such a read-out is clean.

The above scheme is commonly referred to as *error detection* (see also Chapter 2), by which we mean we only detect whether the data is corrupted, but we do not attempt to correct the errors. However, our goal was to correct the corrupted data, not just detect it. This can be easily achieved with the repetition code. Consider the case of a read-out $(0\,0\,1)$: you would immediately guess that the original data is more likely to be $(0\,0\,0)$, which corresponds to the binary bit $s = 0$. On the other hand, if the read-out shows $(1\,0\,1)$, you would guess the second bit is corrupted and the data is likely to be $(1\,1\,1)$ and hence determine the original $s = 1$.

There is a good reason for such a guess. Again let us denote by p the probability of a read-out bit being corrupted, and let us assume[3] that the probability of s being 0 is $1/2$ (and of course the same probability for s being 1). Then, given the read-out $(0\,0\,1)$, the probability of the original data being $(0\,0\,0)$ can be computed as follows. Here again we assume that the read-out bits are corrupted independently. Assuming $\Pr[s=0] = \Pr[s=1] = 1/2$, it is clear that

$$\Pr(\text{Writing in } (0\,0\,0)) = \Pr(\text{Writing in } (1\,1\,1)) = \frac{1}{2}. \tag{3.8}$$

It is also easy to see that

[2] Here we assume each read-out bit is corrupted independently, meaning whether one bit is corrupted or not has no effect on the other bits being corrupted or not. With this independence assumption, the probability of having three corrupted bits is $p \cdot p \cdot p = p^3$.

[3] Why do we need this assumption? Would the situation be different without this assumption? Take the case of $\Pr[s=0] = 0$ and $\Pr[s=1] = 1$ as an example.

$$\begin{aligned}
&\Pr(\text{Writing in } (000) \text{ and reading out } (001)) \\
&\quad = \Pr(\text{Writing in } (000)) \cdot \Pr(\text{Reading out } (001) \,|\, \text{Writing in } (000)) && (3.9)\\
&\quad = \Pr(\text{Writing in } (000)) \cdot \Pr(0 \to 0) \cdot \Pr(0 \to 0) \cdot \Pr(0 \to 1) && (3.10)\\
&\quad = \frac{1}{2} \cdot (1-p) \cdot (1-p) \cdot p && (3.11)\\
&\quad = \frac{(1-p)^2 p}{2}. && (3.12)
\end{aligned}$$

Similarly, we have

$$\Pr(\text{Writing in } (111) \text{ and reading out } (001)) = \frac{(1-p)p^2}{2}. \tag{3.13}$$

These together show that

$$\begin{aligned}
&\Pr(\text{Reading out } (001)) \\
&\quad = \Pr(\text{Writing in } (000) \text{ and reading out } (001)) \\
&\qquad + \Pr(\text{Writing in } (111) \text{ and reading out } (001)) && (3.14)\\
&\quad = \frac{(1-p)p}{2}. && (3.15)
\end{aligned}$$

Thus

$$\begin{aligned}
&\Pr(\text{Writing in } (000) \,|\, \text{Reading out } (001)) \\
&\quad = \frac{\Pr(\text{Writing in } (000) \text{ and reading out } (001))}{\Pr(\text{Reading out } (001))} && (3.16)\\
&\quad = 1-p. && (3.17)
\end{aligned}$$

Similarly, it can be shown that

$$\Pr(\text{Writing in } (111) \,|\, \text{Reading out } (001)) = p. \tag{3.18}$$

As $p < 1/2$ by assumption, we immediately see that

$$1-p > p, \tag{3.19}$$

and, given that the read-out is (001), the case of writing in (111) is less likely. Hence we would guess the original data is more likely to be (000) due to its higher probability. Arguing in a similar manner, we can construct a table for decoding, shown in Table 3.1.

From Table 3.1, we see that given the original data being (000), the correctable error events are the ones when the read-outs are (100), (010), and (001), i.e. the ones when only one bit is in error. The same holds for the other write-in of (111). Thus we say that the three-times repetition code is a *single-error-correcting code*, meaning the code is able to correct all possible one-bit errors. If there are at least two out of the three bits in error during read-out,

Table 3.1 *Decoding table for the repetition code based on probability*

Read-outs	Likely original	Decoded output
(000), (100), (010), (001)	(000)	$s=0$
(111), (011), (101), (110)	(111)	$s=1$

then this code is bound to make an erroneous decision as shown in Table 3.1. The probability of having an erroneous decoded output is given by

$$\Pr(\text{Uncorrectable error}) = 3p^2(1-p)+p^3 \tag{3.20}$$

that is smaller than the original p.

Exercise 3.3 *Prove* $3p^2(1-p)+p^3 < p$ *for* $p \in (0,1/2)$. ◊

Exercise 3.4 *It should be noted that Table 3.1 is obtained under the assumption of* $\Pr[s=0]=\Pr[s=1]=1/2$. *What if* $\Pr[s=0]=0$ *and* $\Pr[s=1]=1$*? Reconstruct the table for this case and conclude that* $\Pr(\text{Uncorrectable error})=0$. *Then rethink whether you need error protection and correction in this case.* ◊

To summarize this section, below we give a formal definition of an error-correcting code.

Definition 3.5 A code $\mathscr{C}$ is said to be an (n,k) *error-correcting code* if it is a scheme of mapping k bits into n bits, and we say $\mathscr{C}$ has code rate $\mathrm{R}=k/n$. We say $\mathscr{C}$ is a t*-error-correcting code* if $\mathscr{C}$ is able to correct any t or fewer errors in the received n-vector. Similarly, we say $\mathscr{C}$ is an e*-error-detecting code* if $\mathscr{C}$ is able to detect any e or fewer errors in the received n-vector.

With the above definition, we see that the three-times repetition code is a $(3,1)$ error-correcting code with code rate $\mathrm{R}=1/3$, and it is a 1-error-correcting code. When being used purely for error detection, it is also a 2-error-detecting code. Moreover, in terms of the error correction or error detection capability, we have the following two theorems. The proofs are left as an exercise.

Theorem 3.6 *Let* $\mathscr{C}$ *be an* (n,k) *binary error-correcting code that is* t*-error-correcting. Then assuming a raw bit error probability of* p*, we have*

$$\begin{aligned}\Pr(\text{Uncorrectable error}) \le &\binom{n}{t+1}p^{t+1}(1-p)^{n-t-1}\\ &+\binom{n}{t+2}p^{t+2}(1-p)^{n-t-2}+\cdots+\binom{n}{n}p^n,\end{aligned} \tag{3.21}$$

where $\binom{n}{\ell}$ *is the binomial coefficient defined as*

$$\binom{n}{\ell} \triangleq \frac{n!}{\ell!(n-\ell)!}. \tag{3.22}$$

Theorem 3.7 *Let* $\mathscr{C}$ *be an* (n,k) *binary error-correcting code that is e-error-detecting. Then assuming a raw bit error probability of p, we have*

$$\begin{aligned}\Pr(\text{Undetectable error}) \leq &\binom{n}{e+1} p^{e+1}(1-p)^{n-e-1} \\ &+ \binom{n}{e+2} p^{e+2}(1-p)^{n-e-2} + \cdots + \binom{n}{n} p^n. \end{aligned} \tag{3.23}$$

Exercise 3.8 *Prove Theorems 3.6 and 3.7.*

Hint: For the situation of Theorem 3.6, if a code is t-error-correcting, we know that it can correctly deal with all error patterns of t or fewer errors. For error patterns with more than t errors, we do not know: some of them might be corrected; some not. Hence, as an upper bound to the error probability, assume that any error pattern with more than t errors cannot be corrected. The same type of thinking also works for Theorem 3.7. ◊

Recall that in (3.6) and (3.7), given the binary bit s, we use $x_1 = x_2 = x_3 = s$ to generate the length-3 binary stream (x_1, x_2, x_3), and use $x_1 + x_2 \stackrel{?}{=} 0$ and $x_1 + x_3 \stackrel{?}{=} 0$ to determine whether the read-out has been corrupted. In general, it is easier to rewrite the two processes using matrices; i.e., we have the following:

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} = s \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \qquad \text{(generating equation)}, \tag{3.24}$$

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \text{(check equations)}. \tag{3.25}$$

The two matrix equations above mean that we use the matrix

$$\mathsf{G} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \tag{3.26}$$

to generate the length-3 binary stream and use the matrix

$$\mathsf{H} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{3.27}$$

to check whether the data is corrupted. Thus, the matrix G is often called the *generator matrix* and H is called the *parity-check matrix*. We have the following definition.

Definition 3.9 Let $\mathscr{C}$ be an (n,k) error-correcting code that maps length-k binary streams $\mathbf{s}$ into length-n binary streams $\mathbf{x}$. We say $\mathscr{C}$ is a *linear code* if there exist a binary matrix G of size $(k \times n)$ and a binary matrix H of size $((n-k) \times n)$ such that the mapping from $\mathbf{s}$ to $\mathbf{x}$ is given by

$$\mathbf{x} = (x_1 \ \cdots \ x_n) = \underbrace{(s_1 \ \cdots \ s_k)}_{=\mathbf{s}} \mathsf{G} \tag{3.28}$$

and the check equations are formed by

$$\mathsf{H}\mathbf{x}^{\mathsf{T}} \stackrel{?}{=} \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{3.29}$$

where by $\mathbf{x}^{\mathsf{T}}$ we mean the transpose of vector $\mathbf{x}$ (rewriting horizontal rows as vertical columns and vice versa). The vector $\mathbf{x}$ is called a *codeword* associated with the binary *message* $\mathbf{s}$.

Exercise 3.10 *With linear codes, the detection of corrupted read-outs is extremely easy. Let $\mathscr{C}$ be an (n,k) binary linear code with a parity-check matrix H of size $((n-k) \times n)$. Given any read-out $\mathbf{y} = (y_1, \ldots, y_n)$, show that $\mathbf{y}$ is corrupted if $\mathsf{H}\mathbf{y}^{\mathsf{T}} \neq \mathbf{0}^{\mathsf{T}}$, i.e. if at least one parity-check equation is unsatisfied. It should be noted that the converse is false in general.*[4] ◊

Exercise 3.11 *Let $\mathscr{C}$ be an (n,k) binary linear code with generator matrix G of size $(k \times n)$ and parity-check matrix H of size $((n-k) \times n)$. Show that the product matrix $\mathsf{H}\mathsf{G}^{\mathsf{T}}$ must be a matrix whose entries are either 0 or multiples of 2; hence, after taking modulo reduction by 2, we have $\mathsf{H}\mathsf{G}^{\mathsf{T}} = \mathsf{0}$, an all-zero matrix.* ◊

Exercise 3.12 (Dual code) *In Definition 3.9, we used matrix G to generate the codeword $\mathbf{x}$ given the binary message $\mathbf{s}$ and used the matrix H to check the integrity of read-outs of $\mathbf{x}$ for an (n,k) linear error-correcting code $\mathscr{C}$. On the other hand, it is possible to reverse the roles of G and H. The process is detailed as follows. Given $\mathscr{C}$, G, and H, we define the* dual code *$\mathscr{C}^{\perp}$ of $\mathscr{C}$ by encoding the length-$(n-k)$ binary message $\mathbf{s}'$ as $\mathbf{x}' = \mathbf{s}'\mathsf{H}$ and check the integrity of $\mathbf{x}'$ using $\mathsf{G}\mathbf{x}'^{\mathsf{T}} \stackrel{?}{=} \mathbf{0}^{\mathsf{T}}$. Based on the above, verify the following:*

(a) The dual code $\mathscr{C}^{\perp}$ of the three-times repetition code is a $(3,2)$ linear code with rate $\mathsf{R}' = 2/3$, and

(b) $\mathscr{C}^{\perp}$ is a 1-error-detecting code.

[4] For the *correction* of corrupted read-outs of linear codewords, see the discussion around (3.34).

This code is called the single parity-check code*; see Chapter 2.* ◊

In the above exercise, we have introduced the concept of a dual code. The dual code is useful in the sense that once you have an (n,k) linear code with generator matrix G and parity-check matrix H, you immediately get another $(n,n-k)$ linear code for free, simply by reversing the roles of G and H. However, readers should be warned that this is very often not for the purpose of error correction. Specifically, throughout the studies of various kinds of linear codes, it is often found that if the linear code $\mathscr{C}$ has a very strong error correction capability, then its dual code $\mathscr{C}^\perp$ is highly likely to be weak. Conversely, if $\mathscr{C}$ is very weak, then its dual $\mathscr{C}^\perp$ might be strong. Nevertheless, the duality between $\mathscr{C}$ and $\mathscr{C}^\perp$ can be extremely useful when studying the combinatorial properties of a code, such as packing (see Section 3.3.3), covering (see p. 179), weight enumerations, etc.

3.3 Hamming code

In Section 3.2, we discussed the $(3,1)$ three-times repetition code that is capable of correcting all 1-bit errors or detecting all 2-bit errors. The price for such a capability is that we have to increase the file size by a factor of 3. For example, if you have a file of size 700 MB to be stored on CD and, in order to keep the file from corruption, you use a $(3,1)$ three-times repetition code, a space of 2100 MB, i.e. 2.1 GB, is needed to store the encoded data. This corresponds to almost half of the storage capacity of a DVD!

Therefore, we see that while the three-times repetition code is able to provide some error protection, it is highly inefficient in terms of rate. In general, we would like the rate $\mathsf{R} = k/n$ to be as close to 1 as possible so that wastage of storage space is kept to a minimum.

3.3.1 Some historical background

The problem of finding efficient error-correcting schemes, but of a much smaller scale, bothered Richard Wesley Hamming (1915–1998) while he was employed by Bell Telephone Laboratory (Bell Labs) in the late 1940s. Hamming was a mathematician with a Ph.D. degree from the University of Illinois at Urbana-Champaign in 1942. He was a professor at the University of Louisville during World War II, and left to work on the Manhattan Project in 1945, programming a computer to solve the problem of whether the detonation of an atomic bomb would ignite the atmosphere. In 1946, Hamming went to Bell

Labs and worked on the Bell Model V computer, an electromechanical relay-based machine. At that time, inputs to computers were fed in on punch cards, which would invariably have read errors (the same as CDs or DVDs in computers nowadays). Prior to executing the program on the punch cards, a special device in the Bell Model V computer would check and detect errors. During weekdays, when errors were found, the computer would flash lights so the operators could correct the problem. During after-hours periods and at weekends, when there were no operators, the machine simply terminated the program and moved on to the next job. Thus, during the weekends, Hamming grew increasingly frustrated with having to restart his programs from scratch due to the unreliable card reader.

Over the next few years he worked on the problem of error correction, developing an increasingly powerful array of algorithms. In 1950 he published what is now known as the *Hamming code*, which remains in use in some applications today.

Hamming is best known for the Hamming code he developed in 1950, as well as the Hamming window[5] used in designing digital filters, the Hamming bound related to sphere packing theory (see Section 3.3.3), and the Hamming distance as a measure of distortion in digital signals (see Exercise 3.16). Hamming received the Turing award in 1968 and was elected to the National Academy of Engineering in 1980.

Exercise 3.13 *The error-correcting mechanism used in CDs*[6] *for data protection is another type of error-correcting code called* Reed–Solomon (R-S) code*, developed by Irving S. Reed and Gustave Solomon in 1960. The use of R-S codes as a means of error correction for CDs was suggested by Jack van Lint (1932–2004) while he was employed at Philips Labs in 1979. Two consecutive R-S codes are used in serial in a CD. These two R-S codes operate in bytes (B) instead of bits (1 B = 8 bits). The first R-S code takes in 24 B of raw data and encodes them into a codeword of length 28 B. After this, another mechanism called interleaver would take 28 such encoded codewords, each 28 B long, and then permute the overall* $28^2 = 784$ *B of data symbols. Finally, the second R-S code will take blocks of 28 B and encode them into blocks of*

[5] The Hamming window was actually not due to Hamming, but to John Tukey (1915–2000), who also rediscovered with James Cooley the famous algorithm of the fast Fourier transform (FFT) that was originally invented by Carl Friedrich Gauss in 1805, but whose importance to modern engineering was not realized by the researchers until 160 years later. The wonderful Cooley–Tukey FFT algorithm is one of the key ingredients of your MP3 players, DVD players, and mobile phones. So, you actually have Gauss to thank for it. Amazing, isn't it?

[6] A similar mechanism is also used in DVDs. We encourage you to visit the webpage of Professor Tom Høholdt at http://www2.mat.dtu.dk/people/T.Hoeholdt/DVD/index.html for an extremely stimulating demonstration.

32 B. Thus, the first R-S code can be regarded as a (28 B, 24 B) linear code and the second as a (32 B, 28 B) linear code. Based on the above, determine the actual size (in megabytes (MB)) of digital information that is stored on a CD if a storage capacity of 720 MB is claimed. Also, what is the overall code rate used on a CD? ◊

3.3.2 Encoding and error correction of the $(7,4)$ Hamming code

The original Hamming code is a $(7,4)$ binary linear code with the following generator and parity-check matrices:

$$\mathsf{G} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathsf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{3.30}$$

Specifically, the encoder of the $(7,4)$ Hamming code takes in a message of four bits, say $\mathbf{s} = (s_1, s_2, s_3, s_4)$ and encodes them as a codeword of seven bits, say $\mathbf{x} = (p_1, p_2, p_3, s_1, s_2, s_3, s_4)$, using the following generating equations:

$$\begin{cases} p_1 = s_1 + s_3 + s_4, \\ p_2 = s_1 + s_2 + s_3, \\ p_3 = s_2 + s_3 + s_4. \end{cases} \tag{3.31}$$

Mappings from $\mathbf{s}$ to $\mathbf{x}$ are tabulated in Table 3.2.

Table 3.2 *Codewords of the* $(7,4)$ *Hamming code*

Message	Codeword	Message	Codeword
0 0 0 0	0 0 0 0 0 0 0	1 0 0 0	1 1 0 1 0 0 0
0 0 0 1	1 0 1 0 0 0 1	1 0 0 1	0 1 1 1 0 0 1
0 0 1 0	1 1 1 0 0 1 0	1 0 1 0	0 0 1 1 0 1 0
0 0 1 1	0 1 0 0 0 1 1	1 0 1 1	1 0 0 1 0 1 1
0 1 0 0	0 1 1 0 1 0 0	1 1 0 0	1 0 1 1 1 0 0
0 1 0 1	1 1 0 0 1 0 1	1 1 0 1	0 0 0 1 1 0 1
0 1 1 0	1 0 0 0 1 1 0	1 1 1 0	0 1 0 1 1 1 0
0 1 1 1	0 0 1 0 1 1 1	1 1 1 1	1 1 1 1 1 1 1

There are several ways to memorize the $(7,4)$ Hamming code. The simplest

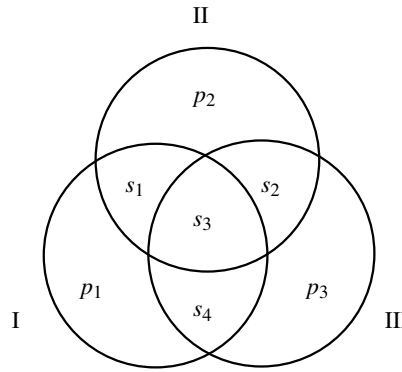

Figure 3.1 Venn diagram of the $(7,4)$ Hamming code.

way is perhaps to use the Venn diagram pointed out by Robert J. McEliece [McE85] and shown in Figure 3.1. There are three overlapping circles: circles I, II, and III. Each circle represents one generating equation as well as one parity-check equation of the $(7,4)$ Hamming code (see (3.31)). For example, circle I corresponds to the first generating equation of $p_1 = s_1 + s_3 + s_4$ and the parity-check equation $p_1 + s_1 + s_3 + s_4 = 0$ since bits s_1, s_3, s_4, and p_1 are included in this circle. Similarly, the second circle, circle II, is for the check equation of $p_2 + s_1 + s_2 + s_3 = 0$, and the third circle III is for the check equation of $p_3 + s_2 + s_3 + s_4 = 0$. Note that each check equation is satisfied if, and only if, there is an even number of 1s in the corresponding circle. Hence, the p_1, p_2, and p_3 are also known as *even parities*.

Using the Venn diagram, error correction of the Hamming code is easy. For example, assume codeword $\mathbf{x} = (0\,1\,1\,0\,1\,0\,0)$ was written onto the CD, but due to an unknown one-bit error the read-out shows $(0\,1\,1\,1\,1\,0\,0)$. We put the read-out into the Venn diagram shown in Figure 3.2. Because of the unknown one-bit error, we see that

- the number of 1s in circle I is 1, an odd number, and a warning (bold circle) is shown;
- the number of 1s in circle II is 3, an odd number, and a warning (bold circle) is shown;
- the number of 1s in circle III is 2, an even number, so no warning (normal circle) is given.

From these three circles we can conclude that the error must not lie in circle III, but lies in both circles I and II. This leaves s_1 as the only possibility, since s_1

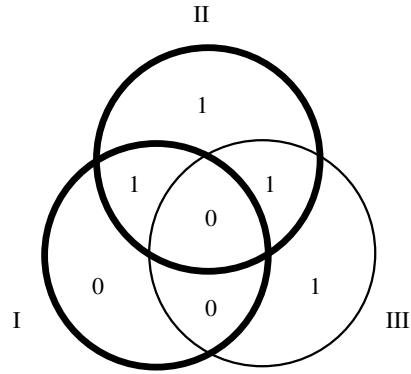

Figure 3.2 Venn diagram used for decoding (0111100).

is the only point lying in circles I and II but not in III. Hence s_1 must be wrong and should be corrected to a 0 so that both warnings are cleared and all three circles show no warning. This then corrects the erroneous bit as expected.

Let us try another example. What if the read-out is (1110100)? The corresponding Venn diagram is shown in Figure 3.3. Following the same reasoning

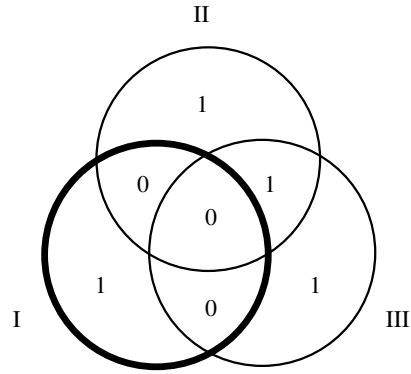

Figure 3.3 Venn diagram used for decoding (1110100).

as before, we see that the error must lie in circle I, but cannot be in circles II and III. Hence the only possible erroneous bit is p_1. Changing the read-out of p_1 from 1 to 0 will correct the one-bit error.

Now let us revisit the above two error-correcting examples and try to formulate a more systematic approach. In the first example, the read-out was $\mathbf{y} = (0111100)$. Using the parity-check matrix H defined in (3.30) we see

the following connection:

$$\mathsf{H}\mathbf{y}^{\mathsf{T}} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \mathbf{I} \\ \mathbf{II} \\ \mathrm{III} \end{pmatrix}. \tag{3.32}$$

This corresponds exactly to the Venn diagram of Figure 3.2. Note that the first entry of $(1\,1\,0)$ corresponds to the first parity-check equation of $p_1 + s_1 + s_3 + s_4 \stackrel{?}{=} 0$ as well as to circle I in the Venn diagram. Since it is unsatisfied, a warning is shown. Similarly for the second and third entries of $(1\,1\,0)$. Now we ask the question: which column of H has the value $(1\,1\,0)^{\mathsf{T}}$? It is the fourth column, which corresponds to the fourth bit of $\mathbf{y}$, i.e. s_1 in $\mathbf{x}$. Then we conclude that s_1 is in error and should be corrected to a 0. The corrected read-out is therefore $(0\,1\,1\,0\,1\,0\,0)$.

For the second example of a read-out being $\mathbf{y} = (1\,1\,1\,0\,1\,0\,0)$, carrying out the same operations gives

$$\mathsf{H}\mathbf{y}^{\mathsf{T}} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \Longleftrightarrow \quad \begin{pmatrix} \mathbf{I} \\ \mathrm{II} \\ \mathrm{III} \end{pmatrix} \tag{3.33}$$

corresponding to the Venn diagram of Figure 3.3. Since $(1\,0\,0)^{\mathsf{T}}$ is the first column of H, it means the first entry of $\mathbf{y}$ is in error, and the corrected read-out should be $(0\,1\,1\,0\,1\,0\,0)$.

There is a simple reason why the above error correction technique works. For brevity, let us focus on the first example of $\mathbf{y} = (0\,1\,1\,1\,1\,0\,0)$. This is the case when the fourth bit of $\mathbf{y}$, i.e. s_1, is in error. We can write $\mathbf{y}$ as follows:

$$\mathbf{y} = (0\,1\,1\,1\,1\,0\,0) = \underbrace{(0\,1\,1\,0\,1\,0\,0)}_{=\mathbf{x}} + \underbrace{(0\,0\,0\,1\,0\,0\,0)}_{=\mathbf{e}}, \tag{3.34}$$

where $\mathbf{x}$ is the original codeword written into a CD and $\mathbf{e}$ is the *error pattern*.

Recall that $\mathbf{x}$ is a codeword of the $(7,4)$ Hamming code; we must have $\mathsf{H}\mathbf{x}^{\mathsf{T}} = \mathbf{0}^{\mathsf{T}}$ from Definition 3.9. Thus from the distributive law verified in Exercise 3.1 we see that

$$\mathsf{H}\mathbf{y}^{\mathsf{T}} = \mathsf{H}(\mathbf{x}^{\mathsf{T}} + \mathbf{e}^{\mathsf{T}}) = \mathsf{H}\mathbf{x}^{\mathsf{T}} + \mathsf{H}\mathbf{e}^{\mathsf{T}} = \mathbf{0}^{\mathsf{T}} + \mathsf{H}\mathbf{e}^{\mathsf{T}} = \mathsf{H}\mathbf{e}^{\mathsf{T}}. \tag{3.35}$$

Since the only nonzero entry of $\mathbf{e}$ is the fourth entry, left-multiplying $\mathbf{e}^{\mathsf{T}}$ by H gives the fourth column of H. Thus, $\mathsf{H}\mathbf{e}^{\mathsf{T}} = (1\,1\,0)^{\mathsf{T}}$. In other words, we have the following logic deductions: without knowing the error pattern $\mathbf{e}$ in the first place, to correct the one-bit error,

$$\mathsf{H}\mathbf{y}^{\mathsf{T}} = (1\,1\,0)^{\mathsf{T}} \tag{3.36}$$

$$\Longrightarrow \quad \mathsf{H}\mathbf{e}^{\mathsf{T}} = (1\,1\,0)^{\mathsf{T}} \tag{3.37}$$

$$\Longrightarrow \quad \mathbf{e} = (0\,0\,0\,1\,0\,0\,0). \tag{3.38}$$

The last logic deduction relies on the following two facts:

(1) we assume there is only one bit in error, and
(2) all columns of H are distinct.

With the above arguments, we get the following result.

Theorem 3.14 *The $(7,4)$ Hamming code can be classified as one of the following:*

(1) a single-error-correcting code,
(2) a double-error-detecting code.

Proof The proof of the Hamming code being able to correct all one-bit errors follows from the same logic deduction given above. To establish the second claim, simply note that when two errors occur, there are two 1s in the error pattern $\mathbf{e}$, for example $\mathbf{e} = (1\,0\,1\,0\,0\,0\,0)$. Calculating the parity-check equations shows $\mathsf{H}\mathbf{y}^{\mathsf{T}} = \mathsf{H}\mathbf{e}^{\mathsf{T}}$. Note that no two distinct columns of H can be summed to yield $(0\,0\,0)^{\mathsf{T}}$. This means any double-error will give $\mathsf{H}\mathbf{y}^{\mathsf{T}} \neq \mathbf{0}^{\mathsf{T}}$ and hence can be detected. □

From Theorem 3.14 we see that the $(7,4)$ Hamming code is able to correct all one-bit errors. Thus, assuming each bit is in error with probability p, the probability of erroneous correction is given by

$$\Pr(\text{Uncorrectable error}) \leq \binom{7}{2} p^2 (1-p)^5 + \binom{7}{3} p^3 (1-p)^4 + \cdots + \binom{7}{7} p^7. \tag{3.39}$$

It should be noted that in (3.39) we actually have an equality. This follows from

the fact that the Hamming code cannot correct any read-outs having more than one bit in error.

Exercise 3.15 *For error detection of the* $(7,4)$ *Hamming code, recall the check equation* $\mathsf{H}\mathbf{y}^{\mathsf{T}} = \mathsf{H}\mathbf{e}^{\mathsf{T}} \stackrel{?}{=} \mathbf{0}^{\mathsf{T}}$. *Using this relation, first show that an error pattern* $\mathbf{e}$ *is undetectable if, and only if,* $\mathbf{e}$ *is a nonzero codeword. Thus the* $(7,4)$ *Hamming code can detect some error patterns that have more than two errors. Use this fact to show that the probability of a detection error of the* $(7,4)$ *Hamming code is*

$$\Pr(\text{Undetectable error}) = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7, \tag{3.40}$$

which is better than what has been claimed by Theorem 3.7.

Hint: Note from Table 3.2 that, apart from the all-zero codeword, there are seven codewords containing three 1s, another seven codewords containing four 1s, and one codeword consisting of seven 1s. ◊

Next we compare the performance of the three-times repetition code with the performance of the $(7,4)$ Hamming code. First, note that both codes are able to correct all one-bit errors or detect all double-bit errors. Yet, the repetition code requires to triple the size of the original file for storage, while the $(7,4)$ Hamming code only needs $7/4 = 1.75$ times the original space. Therefore, the $(7,4)$ Hamming code is more efficient than the three-times repetition code in terms of required storage space.

Before concluding this section, we use the following exercise problem as a quick introduction to another contribution of Hamming. It is related to the topic of *sphere packing*, which will be discussed in Section 3.3.3.

Exercise 3.16 (Hamming distance) *Another contribution of Richard Hamming is the notion of* Hamming distance. *Given any two codewords* $\mathbf{x} = (x_1, x_2, \ldots, x_7)$ *and* $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_7)$, *the Hamming distance between* $\mathbf{x}$ *and* $\mathbf{x}'$ *is the number of places* $\mathbf{x}$ *differs from* $\mathbf{x}'$. *For example, the Hamming distance between* (1011100) *and* (0111001) *is 4 since* $\mathbf{x}$ *differs from* $\mathbf{x}'$ *in the first, second, fifth, and seventh positions. Equivalently, you can compute* $(1011100) + (0111001) = (1100101)$. *Then the condition given in Exercise 3.2, namely,* $x_\ell \stackrel{?}{=} x'_\ell$ *is equivalent to* $x_\ell + x'_\ell \stackrel{?}{=} 0$ *for* $\ell = 1, 2, \ldots, 7$, *shows the distance is 4 since there are four 1s appearing in the sum* (1100101). *Note that the number of ones in a binary vector is called the* Hamming weight *of the vector.*

Now, using Table 3.2, show that every two distinct codewords of a $(7,4)$ *Hamming code are separated by Hamming distance* ≥ 3. ◊

Some of you might wonder why not simply stick to the general definition of Euclidean distance and try to avoid the need for this new definition of Hamming distance. There is a good reason for this. Recall that the Euclidean distance between two distinct points (x,y) and (x',y') is defined as follows:

$$d \triangleq \sqrt{(x-x')^2+(y-y')^2}. \tag{3.41}$$

However, this definition will not work in the binary space. To see this, consider the example of $(x,y)=(0\,0)$ and $(x',y')=(1\,1)$. The Euclidean distance between these two points is given by

$$d=\sqrt{(0-1)^2+(0-1)^2}=\sqrt{1+1}=\sqrt{0}=0, \tag{3.42}$$

where you should note that $1+1=0$ in binary. Thus, the Euclidean distance fails in the binary space.

3.3.3 Hamming bound: sphere packing

In Exercise 3.16 we have seen that every distinct pair of codewords of the $(7,4)$ Hamming code is separated by Hamming distance at least $d=3$. Thus in terms of geometry we have a picture as shown in Figure 3.4(a). Now if we draw two spheres as shown in Figure 3.4(b) (you might want to think of them as high-dimensional balls), each with radius $\mathrm{R}=1$, centered at $\mathbf{x}$ and $\mathbf{x}'$, respectively, these two spheres would not overlap and must be well-separated. Points within the $\mathbf{x}$-sphere represent the read-outs that are at a distance of at most 1 from $\mathbf{x}$. In other words, the points within the $\mathbf{x}$-sphere are either $\mathbf{x}$ or $\mathbf{x}$ with a one-bit error.

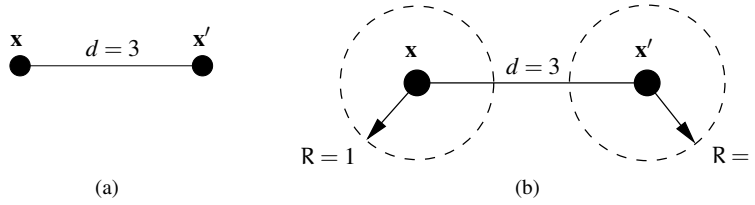

Figure 3.4 Geometry of $\mathbf{x}\neq\mathbf{x}'$ in the $(7,4)$ Hamming code with Hamming distance 3.

If we draw a sphere with radius $\mathrm{R}=1$ centered at each codeword of the $(7,4)$ Hamming code, there will be 16 nonoverlapping spheres since there are 16 codewords and every pair of distinct codewords is separated by a distance of at least 3. Given that a codeword $\mathbf{x}$ was written into the CD, for example,

the one-bit error read-out must be at distance 1 from $\mathbf{x}$ and therefore must lie within the radius-1 $\mathbf{x}$-sphere centered at $\mathbf{x}$. It cannot lie in other spheres since the spheres are well-separated. This shows that this one-bit error read-out is closer to $\mathbf{x}$ than to any other codewords of the $(7,4)$ Hamming code.

Thus, correcting a one-bit error is always possible for the $(7,4)$ Hamming code. This can be seen as a geometrical explanation of the single-error-correction capability of the $(7,4)$ Hamming code. We may generalize the above argument slightly and give the following theorem.

Theorem 3.17 *Let $\mathscr{C}$ be an (n,k) error-correcting code (not necessarily linear, i.e. it does not necessarily have the generator and parity-check matrices G and H). Assume that every distinct pair of codewords in $\mathscr{C}$ is separated by Hamming distance at least d; then $\mathscr{C}$ is a t-error-correcting code with*

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor, \tag{3.43}$$

where by $\lfloor \xi \rfloor$ we mean[7] the largest integer not larger than ξ. Also, if $\mathscr{C}$ is used only for error detection, then $\mathscr{C}$ is an $(d-1)$-error-detecting code.

Proof Given d, we can draw spheres with radius t centered at the codewords of $\mathscr{C}$. Since $2t < d$, the spheres must be nonoverlapping. Extending the proof to error detection is obvious. □

The above theorem says that in order to correct more errors, the codewords should be placed as far apart as possible. But this is not what we are interested in here. Instead, we are interested in the reverse direction. We ask the following question.

> *Consider a t-error-correcting code $\mathscr{C}$ that maps input messages to a binary stream of length n. So, we draw spheres of radius t centered at the codewords of $\mathscr{C}$. The spheres do not overlap with each other. What is the maximal number of codewords $\mathscr{C}$ can have? In other words, we are interested in knowing how many nonoverlapping radius-t spheres can be packed into an n-dimensional binary space.*

This is the *sphere packing* problem in discrete mathematics. Let us work through some examples in order to understand the question better.

Example 3.18 The $(7,4)$ Hamming code has 16 codewords, hence 16 spheres with radius 1, since the code is 1-error-correcting.

- The codewords have length 7, with a binary value in each coordinate. So,

[7] For example, $\lfloor 1.1 \rfloor = \lfloor 1.9 \rfloor = 1$.

the number of possible length-7 binary tuples is $2^7 = 128$, meaning there are 128 points in this 7-dimensional binary space.

- Each codeword of the Hamming code is surrounded by a sphere with radius 1. There are $1 + \binom{7}{1} = 8$ points in each sphere. This first "1" corresponds to the center, i.e. distance 0. The remaining $\binom{7}{1}$ points are the ones at distance 1 from the center, i.e. one-bit errors from the codeword.

Thus, the 16 nonoverlapping spheres actually cover $16 \times 8 = 128$ points, which are all the points in the 7-dimensional binary space. We see that the $(7,4)$ Hamming code has the tightest possible packing of radius-1 spheres in the 7-dimensional binary space. ◊

Example 3.19 Let us consider the dual of the $(7,4)$ Hamming code $\mathscr{C}$ in this example. Recall from Exercise 3.12 that the dual code $\mathscr{C}^\perp$ is obtained by reversing the roles of the generating matrix G and the parity-check matrix H of $\mathscr{C}$. That is, we use the parity-check matrix H for encoding and the generator matrix G for checking. Thus the generator matrix $\mathsf{G}^\perp$ of $\mathscr{C}^\perp$ is given by

$$\mathsf{G}^\perp = \mathsf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix} \tag{3.44}$$

and it maps binary messages of length 3 to codewords of length 7. All the eight possible codewords are tabulated in Table 3.3.

Table 3.3 *Codewords of the dual code of the* $(7,4)$ *Hamming code*

Message	Codeword	Message	Codeword
0 0 0	0 0 0 0 0 0 0	1 0 0	1 0 0 1 0 1 1
0 0 1	0 0 1 0 1 1 1	1 0 1	1 0 1 1 1 0 0
0 1 0	0 1 0 1 1 1 0	1 1 0	1 1 0 0 1 0 1
0 1 1	0 1 1 1 0 0 1	1 1 1	1 1 1 0 0 1 0

You can check that the codewords are separated by Hamming distance 4 exactly. Hence $\mathscr{C}^\perp$ is able to correct errors up to $t = \lfloor \frac{4-1}{2} \rfloor = 1$, which is the same as $\mathscr{C}$ does. Thus, it is clear that $\mathscr{C}^\perp$ is not a good packing of radius-1 spheres in the 7-dimensional binary space since it packs only eight spheres, while $\mathscr{C}$ can pack 16 spheres into the same space. ◊

Why are we interested in the packing of radius-t spheres in an n-dimensional

space? The reason is simple. Without knowing the parameter k in the first place, i.e. without knowing how many distinct 2^k binary messages can be encoded, by fixing t we make sure that the codewords are immune to errors with at most t error bits. Choosing n means the codewords will be stored in n bits. Being able to pack more radius-t spheres into the n-dimensional spaces means we can have more codewords, hence larger k. This gives a general bound on k, known as the *sphere bound*, and stated in the following theorem.

Theorem 3.20 (Sphere bound) *Let n, k, and t be defined as above. Then we have*

$$2^k \leq \frac{2^n}{\binom{n}{0}+\binom{n}{1}+\cdots+\binom{n}{t}}. \tag{3.45}$$

Codes with parameter n, k, and t that achieve equality in (3.45) are called perfect*, meaning a perfect packing.*

Proof Note that 2^k is the number of codewords, while 2^n is the number of points in an n-dimensional binary space, i.e. the number of distinct binary n-tuples. The denominator shows the number of points within a radius-t sphere. The inequality follows from the fact that for t-error-correcting codes the spheres must be nonoverlapping. □

Finally we conclude this section with the following very deep result.

Theorem 3.21 *The only parameters satisfying the bound (3.45) with equality are*

$$\begin{cases} n = 2^u - 1, & k = 2^u - u - 1, & t = 1, & \text{for any positive integer } u; \\ n = 23, & k = 12, & t = 3; & \\ n = 2u+1, & k = 1, & t = u, & \text{for any positive integer } u. \end{cases} \tag{3.46}$$

This theorem was proven by Aimo Tietäväinen [Tie73] in 1973 after much work by Jack van Lint. One code satisfying the second case of $n = 23$, $k = 12$, and $t = 3$ is the *Golay code*, hand-constructed by Marcel J. E. Golay in 1949 [Gol49].[8] Vera Pless [Ple68] later proved that the Golay code is the only code with these parameters that satisfies (3.45) with equality. The first case is a general Hamming code of order u (see Exercise 3.22 below), and the last case is the $(2u+1)$-times repetition code, i.e. repeating the message $(2u+1)$ times.

[8] This paper is only half a page long, but belongs to the most important paper in information theory ever written! Not only did it present the perfect Golay code, but it also gave the generalization of the Hamming code and the first publication of a parity-check matrix. And even though it took over 20 years to prove it, Golay already claimed in that paper that there were no other perfect codes. For more details on the life of Marcel Golay, see http://www.isiweb.ee.ethz.ch/archive/massey_pub/pdf/BI953.pdf.

Exercise 3.22 (Hamming code of order u) *Recall that the* $(7,4)$ *Hamming code is defined by its parity-check matrix*

$$\mathsf{H} = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}. \tag{3.47}$$

Note that the columns of the above (3×7) *matrix consist of all possible nonzero length-3 binary vectors. From this, we can easily define a general Hamming code* $\mathscr{C}_u$ *of order u. Let* H_u *be the matrix whose columns consist of all possible nonzero length-u binary vectors;* H_u *is of size* $(u \times (2^u - 1))$*. Then a general Hamming code* $\mathscr{C}_u$ *is the code defined by the parity-check matrix* H_u *with* $n = 2^u - 1$ *and* $k = 2^u - u - 1$*. Show that*

(a) $\mathscr{C}_u$ *is 2-error-detecting (Hint: Show that* $\mathsf{H}_u \mathbf{e}^\mathsf{T} \neq \mathbf{0}^\mathsf{T}$ *for all nonzero vectors* $\mathbf{e}$ *that have at most two nonzero entries);*

(b) $\mathscr{C}_u$ *is 1-error-correcting (Hint: Show that* $\mathsf{H}_u \mathbf{y}^\mathsf{T} = \mathbf{0}^\mathsf{T}$ *for some nonzero vector* $\mathbf{y}$ *if, and only if,* $\mathbf{y}$ *has at least three nonzero entries. Then use this to conclude that every pair of distinct codewords is separated by Hamming distance at least 3).* ◊

3.4 Further reading

In this chapter we have briefly introduced the theory of error-correcting codes and have carefully studied two example codes: the three-times repetition code and the $(7,4)$ Hamming code. Besides their error correction capabilities, we have also briefly studied the connections of these codes to sphere packing in high-dimensional spaces.

For readers who are interested in learning more about other kinds of error-correcting codes and their practical uses, [Wic94] is an easy place to start, where you can learn about a more general treatment of the Hamming codes. Another book, by Shu Lin and Daniel Costello [LC04], is a comprehensive collection of all modern coding schemes. An old book by Jessy MacWilliams and Neil Sloane [MS77] is the most authentic source for learning the theory of error-correcting codes, but it requires a solid background in mathematics at graduate level.

The Hamming codes are closely related to combinatorial designs, difference sets, and Steiner systems. All are extremely fascinating objects in combinatorics. The interested readers are referred to [vLW01] by Jack van Lint and Richard Wilson for further reading on these subjects. These combinatorial

objects are also used in the designs of radar systems, spread spectrum-based cellular communications, and optical fiber communication systems.

The topic of sphere packing is always hard, yet fascinating. Problems therein have been investigated for more than 2000 years, and many remain open. A general discussion of this topic can be found in [CS99]. As already seen in Theorem 3.21, the sphere packing bound is not achievable in almost all cases. Some bounds that are tighter than the sphere packing bound, such as the Gilbert–Varshamov bound, the Plotkin bound, etc., can be found in [MS77] and [Wic94]. In [MS77] a table is provided that lists all known best packings in various dimensions. An updated version can be found in [HP98]. So far, the tightest lower bound on the existence of the densest possible packings is the Tsfasman–Vlăduţ–Zink (TVZ) bound, and there are algebraic geometry codes constructed from function fields defined by the Garcia–Stichtentoth curve that perform better than the TVZ bound, i.e. much denser sphere packings. A good overview of this subject can be found in [HP98].

References

[CS99] John Conway and Neil J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3rd edn. Springer Verlag, New York, 1999.

[Gol49] Marcel J. E. Golay, "Notes on digital coding," *Proceedings of the IRE*, vol. 37, p. 657, June 1949.

[HP98] W. Cary Huffman and Vera Pless, eds., *Handbook of Coding Theory*. North-Holland, Amsterdam, 1998.

[LC04] Shu Lin and Daniel J. Costello, Jr., *Error Control Coding*, 2nd edn. Prentice Hall, Upper Saddle River, NJ, 2004.

[McE85] Robert J. McEliece, "The reliability of computer memories," *Scientific American*, vol. 252, no. 1, pp. 68–73, 1985.

[MS77] F. Jessy MacWilliams and Neil J. A. Sloane, *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, 1977.

[Ple68] Vera Pless, "On the uniqueness of the Golay codes," *Journal on Combination Theory*, vol. 5, pp. 215–228, 1968.

[Tie73] Aimo Tietäväinen, "On the nonexistence of perfect codes over finite fields," *SIAM Journal on Applied Mathematics*, vol. 24, no. 1, pp. 88–96, January 1973.

[vLW01] Jacobus H. van Lint and Richard M. Wilson, *A Course in Combinatorics*, 2nd edn. Cambridge University Press, Cambridge, 2001.

[Wic94] Stephen B. Wicker, *Error Control Systems for Digital Communication and Storage*. Prentice Hall, Englewood Cliffs, NJ, 1994.