2

Error-detecting codes

When a message is transmitted, the inevitable noise disturbance usually degrades the quality of communication. Whenever repetition is possible, it is sufficient to detect the occurrence of an error. When an error is detected, we simply repeat the message, and it may be correct the second time or even possibly the third time.

It is not possible to detect an error if every possible symbol, or set of symbols, that can be received is a legitimate message. It is only possible to catch errors if there are some restrictions on what a proper message is. The problem is to keep these restrictions on the possible messages down to ones that are simple. In practice, "simple" means "easily computable." In this chapter, we will mainly investigate the problem of designing codes such that any single error can be detected at the receiver. In Chapter 3, we will then consider *correcting* the errors that occur during the transmission.

2.1 Review of modular arithmetic

We first give a quick review of the basic arithmetic which is extensively used in the following sections. For binary digits, which take values of only 0 and 1, the rules for addition and multiplication are defined by

$$\begin{array}{lll} 0+0=0 & & 0\times 0=0 \\ 0+1=1 & \text{and} & 0\times 1=0 \\ 1+0=1 & & 1\times 0=0 \\ 1+1=0 & & 1\times 1=1, \end{array} \tag{2.1}$$

respectively. For example, by (2.1), we have

$$1+1\times 0+0+1\times 1=1+0+0+1=0. \tag{2.2}$$

If we choose to work in the decimal arithmetic, the binary arithmetic in (2.1) can be obtained by dividing the result in decimal by 2 and taking the remainder. For example, (2.2) yields

$$1+0+0+1=2\equiv 0 \bmod 2. \tag{2.3}$$

Occasionally, we may work modulo some number other than 2 for the case of a nonbinary source. Given a positive integer m, for the addition and multiplication mod m ("mod" is an abbreviation for "modulo"), we merely divide the result in decimal by m and take the nonnegative remainder. For instance, consider an information source with five distinct outputs 0, 1, 2, 3, 4. It follows that

$$2+4=1\times 5+\text{“1”} \quad\Longleftrightarrow\quad 2+4\equiv 1 \bmod 5, \tag{2.4}$$

$$3\times 4=2\times 5+\text{“2”} \quad\Longleftrightarrow\quad 3\times 4\equiv 2 \bmod 5. \tag{2.5}$$

Other cases for the modulo 5 addition and multiplication can be referred to in Table 2.1.

Table 2.1 *Addition and multiplication modulo 5*

+ mod 5	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

× mod 5	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

For multiplication mod m, we have to be more careful if m is not a prime. Suppose that we have the numbers a and b congruent to a' and b' modulo the modulus m. This means that

$$a\equiv a' \bmod m \quad\text{and}\quad b\equiv b' \bmod m \tag{2.6}$$

or

$$a=a'+k_1m \quad\text{and}\quad b=b'+k_2m \tag{2.7}$$

for some integers k_1 and k_2. For the product ab, we have

$$ab=a'b'+a'k_1m+b'k_2m+k_1k_2m^2 \tag{2.8}$$

and hence

$$ab\equiv a'b' \bmod m. \tag{2.9}$$

Now consider the particular case

$$a = 15, \quad b = 12, \quad m = 10. \tag{2.10}$$

We have $a' = 5$ and $b' = 2$ by (2.7) and $ab \equiv a'b' \equiv 0 \bmod 10$ by (2.9). But neither a nor b is zero! Only for a *prime* modulus do we have the important property that if a product is zero, then at least one factor must be zero.

Exercise 2.1 *In order to become more familiar with the modular operation check out the following problems:*

$$3 \times 6 + 7 \equiv ? \bmod 11 \tag{2.11}$$

and

$$5 - 4 \times 2 \equiv ? \bmod 7. \tag{2.12}$$

◊

More on modular arithmetic can be found in Section 3.1.

2.2 Independent errors – white noise

To simplify the analysis of noise behavior, we assume that errors in a message satisfy the following constraints:

(1) the probability of an error in any binary position is assumed to be a fixed number p, and
(2) errors in different positions are assumed to be independent.[1]

Such noise is called "white noise" in analogy with white light, which is supposed to contain uniformly all the frequencies detected by the human eye. However, in practice, there are often reasons for errors to be more common in some positions in the message than in others, and it is often true that errors tend to occur in bursts and not to be independent.We assume white noise in the very beginning because this is the simplest case, and it is better to start from the simplest case and move on to more complex situations after we have built up a solid knowledge on the simple case.

Consider a message consisting of n digits for transmission. For white noise, the probability of no error in any position is given by

$$(1 - p)^n. \tag{2.13}$$

[1] Given events $\mathcal{A}_\ell$, they are said to be independent if $\Pr(\bigcap_{\ell=1}^n \mathcal{A}_\ell) = \prod_{\ell=1}^n \Pr(\mathcal{A}_\ell)$. Here "$\bigcap_\ell$" denotes set-intersection, i.e. $\bigcap_\ell \mathcal{A}_\ell$ is the set of elements that are members of *all* sets $\mathcal{A}_\ell$. Hence, $\Pr(\bigcap_{\ell=1}^n \mathcal{A}_\ell)$ is the event that *all* events $\mathcal{A}_\ell$ occur at the same time. The notation $\prod_\ell$ is a shorthand for multiplication: $\prod_{\ell=1}^n a_\ell \triangleq a_1 \cdot a_2 \cdots a_n$.

The probability of a single error in the message is given by

$$np(1-p)^{n-1}. \tag{2.14}$$

The probability of ℓ errors is given by the $(\ell+1)$th term in the binomial expansion:

$$\begin{aligned} 1 &= \big((1-p)+p\big)^n && (2.15)\\ &= \binom{n}{0}(1-p)^n + \binom{n}{1}p(1-p)^{n-1} + \binom{n}{2}p^2(1-p)^{n-2} \\ &\quad + \cdots + \binom{n}{n}p^n && (2.16)\\ &= (1-p)^n + np(1-p)^{n-1} + \frac{n(n-1)}{2}p^2(1-p)^{n-2} + \cdots + p^n. && (2.17) \end{aligned}$$

For example, the probability of exactly two errors is given by

$$\frac{n(n-1)}{2}p^2(1-p)^{n-2}. \tag{2.18}$$

We can obtain the probability of an even number of errors $(0,2,4,\ldots)$ by adding the following two binomial expansions and dividing by 2:

$$1 = \big((1-p)+p\big)^n = \sum_{\ell=0}^{n}\binom{n}{\ell}p^\ell(1-p)^{n-\ell}, \tag{2.19}$$

$$(1-2p)^n = \big((1-p)-p\big)^n = \sum_{\ell=0}^{n}(-1)^\ell\binom{n}{\ell}p^\ell(1-p)^{n-\ell}. \tag{2.20}$$

Denote by $\lfloor \xi \rfloor$ the greatest integer not larger than ξ. We have[2]

$$\begin{aligned} \Pr(\text{An even number of errors}) &= \sum_{\ell=0}^{\lfloor n/2\rfloor}\binom{n}{2\ell}p^{2\ell}(1-p)^{n-2\ell} && (2.21)\\ &= \frac{1+(1-2p)^n}{2}. && (2.22) \end{aligned}$$

The probability of an odd number of errors is 1 minus this number.

Exercise 2.2 *Actually, this is a good chance to practice your basic skills on the method of induction: can you show that*

$$\sum_{\ell=0}^{\lfloor n/2\rfloor}\binom{n}{2\ell}p^{2\ell}(1-p)^{n-2\ell} = \frac{1+(1-2p)^n}{2} \tag{2.23}$$

and

$$\sum_{\ell=0}^{\lfloor (n-1)/2\rfloor}\binom{n}{2\ell+1}p^{2\ell+1}(1-p)^{n-2\ell-1} = \frac{1-(1-2p)^n}{2} \tag{2.24}$$

[2] Note that zero errors also counts as an even number of errors here.

by induction on n?

Hint: Note that

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \tag{2.25}$$

for $n, k \geq 1$. ◊

2.3 Single parity-check code

The simplest way of encoding a binary message to make it error-detectable is to count the number of 1s in the message, and then append a final binary digit chosen so that the entire message has an even number of 1s in it. The entire message is therefore of *even parity*. Thus to $(n-1)$ message positions we append an nth parity-check position. Denote by x_ℓ the original bit in the ℓth message position, $\forall\, 1 \leq \ell \leq n-1$, and let x_n be the parity-check bit. The constraint of even parity implies that

$$x_n = \sum_{\ell=1}^{n-1} x_\ell \tag{2.26}$$

by (2.1). Note that here (and for the remainder of this book) we omit "mod 2" and implicitly assume it everywhere. Let y_ℓ be the channel output corresponding to x_ℓ, $\forall\, 1 \leq \ell \leq n$. At the receiver, we firstly count the number of 1s in the received sequence $\mathbf{y}$. If the even-parity constraint is violated for the received vector, i.e.

$$\sum_{\ell=1}^{n} y_\ell \neq 0, \tag{2.27}$$

this indicates that at least one error has occurred.

For example, given a message $(x_1, x_2, x_3, x_4) = (0\,1\,1\,1)$, the parity-check bit is obtained by

$$x_5 = 0+1+1+1 = 1, \tag{2.28}$$

and hence the resulting even-parity codeword $(x_1, x_2, x_3, x_4, x_5)$ is $(0\,1\,1\,1\,1)$. Suppose the codeword is transmitted, but a vector $\mathbf{y} = (0\,0\,1\,1\,1)$ is received. In this case, an error in the second position is met. We have

$$y_1 + y_2 + y_3 + y_4 + y_5 = 0+0+1+1+1 = 1\,(\neq 0); \tag{2.29}$$

thereby the error is detected. However, if another vector of $(0\,0\,1\,1\,0)$ is received, where two errors (in the second and the last position) have occurred,

no error will be detected since

$$y_1 + y_2 + y_3 + y_4 + y_5 = 0 + 0 + 1 + 1 + 0 = 0. \tag{2.30}$$

Evidently in this code any odd number of errors can be detected. But any even number of errors cannot be detected.

For channels with white noise, (2.22) gives the probability of any even number of errors in the message. Dropping the first term of (2.21), which corresponds to the probability of no error, we have the following probability of undetectable errors for the single parity-check code introduced here:

$$\Pr(\text{Undetectable errors}) = \sum_{\ell=1}^{\lfloor n/2 \rfloor} \binom{n}{2\ell} p^{2\ell} (1-p)^{n-2\ell} \tag{2.31}$$

$$= \frac{1 + (1-2p)^n}{2} - (1-p)^n. \tag{2.32}$$

The probability of detectable errors, i.e. all the odd-number errors, is then obtained by

$$\Pr(\text{Detectable errors}) = 1 - \frac{1 + (1-2p)^n}{2} = \frac{1 - (1-2p)^n}{2}. \tag{2.33}$$

Obviously, we should have that

$$\Pr(\text{Detectable errors}) \gg \Pr(\text{Undetectable errors}). \tag{2.34}$$

For p very small, we have

$$\Pr(\text{Undetectable errors}) = \frac{1 + (1-2p)^n}{2} - (1-p)^n \tag{2.35}$$

$$= \frac{1}{2} + \frac{1}{2}\left[\binom{n}{0} - \binom{n}{1}(2p) + \binom{n}{2}(2p)^2 - \cdots\right] - \left[\binom{n}{0} - \binom{n}{1}p + \binom{n}{2}p^2 - \cdots\right] \tag{2.36}$$

$$= \frac{1}{2} + \frac{1}{2}\left[1 - 2np + \frac{n(n-1)}{2}4p^2 - \cdots\right] - \left[1 - np + \frac{n(n-1)}{2}p^2 - \cdots\right] \tag{2.37}$$

$$\simeq \frac{n(n-1)}{2}p^2 \tag{2.38}$$

and

$$\Pr(\text{Detectable errors}) = \frac{1 - (1-2p)^n}{2} \tag{2.39}$$

$$= \frac{1}{2} - \frac{1}{2}\left[\binom{n}{0} - \binom{n}{1}(2p) + \cdots\right] \tag{2.40}$$

$$= \frac{1}{2} - \frac{1}{2}[1 - 2np + \cdots] \tag{2.41}$$

$$\simeq np. \tag{2.42}$$

In the above approximations, we only retain the leading term that dominates the sum.

Hence, (2.34) requires

$$np \gg \frac{n(n-1)}{2} p^2, \tag{2.43}$$

and implies that the shorter the message, the better the detecting performance.

In practice, it is common to break up a long message in the binary alphabet into blocks of $(n-1)$ digits and to append one binary digit to each block. This produces the *redundancy* of

$$\frac{n}{n-1} = 1 + \frac{1}{n-1}, \tag{2.44}$$

where the redundancy is defined as the total number of binary digits divided by the minimum necessary. The *excess redundancy* is $1/(n-1)$. Clearly, for low redundancy we want to use long messages, but for high reliability short messages are better. Thus the choice of the length n for the blocks to be sent is a compromise between the two opposing forces.

2.4 The ASCII code

Here we introduce an example of a single parity-check code, called the *American Standard Code for Information Interchange (ASCII)*, which was the first code developed specifically for computer communications. Each character in ASCII is represented by seven data bits constituting a unique binary sequence. Thus a total of 128 $(= 2^7)$ different characters can be represented in ASCII. The characters are various commonly used letters, numbers, special control symbols, and punctuation symbols, e.g. \$, %, and @. Some of the special control symbols, e.g. ENQ (enquiry) and ETB (end of transmission block), are used for communication purposes. Other symbols, e.g. BS (back space) and CR (carriage return), are used to control the printing of characters on a page. A complete listing of ASCII characters is given in Table 2.2.

Since computers work in bytes which are blocks of 8 bits, a single ASCII symbol often uses 8 bits. The eighth bit is set so that the total number of 1s in the eight positions is an even number. For example, consider "K" in Table 2.2 encoded as $(113)_8$, which can be transformed into binary form as follows:

$$(113)_8 = 1001011 \tag{2.45}$$

Table 2.2 *Seven-bit ASCII code*

Octal code	Char.	Octal code	Char.	Octal code	Char.	Octal code	Char.
000	NUL	040	SP	100	@	140	‘
001	SOH	041	!	101	A	141	a
002	STX	042	”	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046	&	106	F	146	f
007	BEL	047	’	107	G	147	g
010	BS	050	(	110	H	150	h
011	HT	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FF	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	[	173	{
034	FS	074	<	134	\	174	|
035	GS	075	=	135	]	175	}
036	RS	076	>	136	^	176	~
037	US	077	?	137	_	177	DEL

(where we have dropped the first 2 bits of the first octal symbol). In this case, the parity-check bit is 0; "K" is thus encoded as 10010110 for even parity. You are encouraged to encode the remaining characters in Table 2.2.

By the constraint of even parity, any single error, a 0 changed into a 1 or a 1 changed into a 0, will be detected[3] since after the change there will be an odd number of 1s in the eight positions. Thus, we have an error-detecting code that helps to combat channel noise. Perhaps more importantly, the code makes it much easier to maintain the communication quality since the machine can detect the occurrence of errors by itself.

2.5 Simple burst error-detecting code

In some situations, errors occur in bursts rather than in isolated positions in the received message. For instance, lightning strikes, power-supply fluctuations, loose flakes on a magnetic surface are all typical causes of a burst of noise. Suppose that the maximum length of any error burst[4] that we are to detect is L. To protect data against the burst errors, we first divide the original message into a sequence of words consisting of L bits. Aided with a pre-selected error-detecting code, parity checks are then computed over the corresponding word positions, instead of the bit positions.

Based on the above scenario, if an error burst occurs within one word, in effect only a single word error is observed. If an error burst covers the end of one word and the beginning of another, still no two errors corresponding to the same position of words will be met, since we assumed that any burst length l satisfies $0 \leq l \leq \mathrm{L}$. Consider the following example for illustration.

Example 2.3 If the message is

Hello NCTU

and the maximum burst error length L is 8, we can use the 7-bit ASCII code in Table 2.2 and protect the message against burst errors as shown in Table 2.3. (Here no parity check is used within the ASCII symbols.) The encoded message is therefore

Hello NCTUn

[3] Actually, to be precise, every odd number of errors is detected.

[4] An error burst is said to have length L if errors are confined to L consecutive positions. By this definition, the *error patterns* 0111110, 0101010, and 0100010 are all classified as bursts of length 5. Note that a 0 in an error pattern denotes that no error has happened in that position, while a 1 denotes an error. See also (3.34) in Section 3.3.2.

Table 2.3 *Special type of parity check to protect against burst errors of maximum length* $\mathrm{L} = 8$

$\mathrm{H} = (110)_8 =$	01001000
$\mathrm{e} = (145)_8 =$	01100101
$\mathrm{l} = (154)_8 =$	01101100
$\mathrm{l} = (154)_8 =$	01101100
$\mathrm{o} = (157)_8 =$	01101111
$\mathrm{SP} = (040)_8 =$	00100000
$\mathrm{N} = (116)_8 =$	01001110
$\mathrm{C} = (103)_8 =$	01000011
$\mathrm{T} = (124)_8 =$	01010100
$\mathrm{U} = (125)_8 =$	01010101
Check sum =	01101110 = n

where n is the parity-check symbol.

Suppose a burst error of length 5, as shown in Table 2.4, is met during the transmission of the above message, where the bold-face positions are in error. In this case, the burst error is successfully detected since the check sum is not 00000000. However, if the burst error of length 16 shown in Table 2.5 occurs, the error will not be detected due to the all-zero check sum. ◊

Exercise 2.4 *Could you repeat the above process of encoding for the case of* $\mathrm{L} = 16$*? Also, show that the resulting code can detect all the bursts of length at most 16.* ◊

Exercise 2.5 *Can you show that the error might not be detected if there is more than one burst, even if each burst is of length no larger than* L*?* ◊

2.6 Alphabet plus number codes – weighted codes

The codes we have discussed so far were all designed with respect to a simple form of "white noise" that causes some bits to be flipped. This is very suitable for many types of machines. However, in some systems, where people are involved, other types of noise are more appropriate. The first common human error is to interchange adjacent digits of numbers; for example, 38 becomes 83. A second common error is to double the wrong one of a triple of digits, where two adjacent digits are the same; for example, 338 becomes 388. In addition, the confusion of O ("oh") and 0 ("zero") is also very common.

Table 2.4 *A burst error of length 5 has occurred during transmission and is detected because the check sum is not 0000000; bold-face positions denote positions in error*

H ⇒ K	0	1	0	0	1	0	**1**	**1**
e ⇒ ENQ	0	**0**	**0**	0	0	1	0	1
l	0	1	1	0	1	1	0	0
l	0	1	1	0	1	1	0	0
o	0	1	1	0	1	1	1	1
SP	0	0	1	0	0	0	0	0
N	0	1	0	0	1	1	1	0
C	0	1	0	0	0	0	1	1
T	0	1	0	1	0	1	0	0
U	0	1	0	1	0	1	0	1
n	0	1	1	0	1	1	1	0
Check sum =	0	1	1	0	0	0	1	1

Table 2.5 *A burst error of length 16 has occurred during transmission, but it is not detected; bold-face positions denote positions in error*

H ⇒ K	0	1	0	0	1	0	**1**	**1**
e ⇒ J	0	1	**0**	0	**1**	**0**	**1**	**0**
l ⇒ @	0	1	**0**	0	**0**	**0**	0	0
l	0	1	1	0	1	1	0	0
o	0	1	1	0	1	1	1	1
SP	0	0	1	0	0	0	0	0
N	0	1	0	0	1	1	1	0
C	0	1	0	0	0	0	1	1
T	0	1	0	1	0	1	0	0
U	0	1	0	1	0	1	0	1
n	0	1	1	0	1	1	1	0
Check sum =	0	0	0	0	0	0	0	0

Table 2.6 *Weighted sum: progressive digiting*

Message	Sum	Sum of sum
w	w	w
x	$w+x$	$2w+x$
y	$w+x+y$	$3w+2x+y$
z	$w+x+y+z$	$4w+3x+2y+z$

In English text-based systems, it is quite common to have a source alphabet consisting of the 26 letters, space, and the 10 decimal digits. Since the size of this source alphabet, 37 ($= 26+1+10$), is a prime number, we can use the following method to detect the presence of the above described typical errors. Firstly, each symbol in the source alphabet is mapped to a distinct number in $\{0,1,2,\ldots,36\}$. Given a message for encoding, we weight the symbols with weights $1,2,3,\ldots$, beginning with the check digit of the message. Then, the weighted digits are summed together and reduced to the remainder after dividing by 37. Finally, a check symbol is selected such that the sum of the check symbol and the remainder obtained above is congruent to 0 modulo 37.

To calculate this sum of weighted digits easily, a technique called *progressive digiting*, illustrated in Table 2.6, has been developed. In Table 2.6, it is supposed that we want to compute the weighted sum for a message $wxyz$, i.e. $4w+3x+2y+1z$. For each symbol in the message, we first compute the running sum from w to the symbol in question, thereby obtaining the second column in Table 2.6. We can sum these sums again in the same way to obtain the desired weighted sum.

Example 2.6 We assign a distinct number from $\{0,1,2,\ldots,36\}$ to each symbol in the combined alphabet/number set in the following way: "0" $= 0$, "1" $= 1$, "2" $= 2$, …, "9" $= 9$, "A" $= 10$, "B" $= 11$, "C" $= 12$, …, "Z" $= 35$, and "space" $= 36$. Then we encode

3B 8.

We proceed with the progressive digiting as shown in Table 2.7 and obtain a weighted sum of 183. Since $183 \bmod 37 = 35$ and $35+2$ is divisible by 37, it follows that the appended check digit should be

"2" $= 2$.

Table 2.7 *Progressive digiting for the example of "3B 8": we need to add "2" = 2 as a check-digit to make sure that the weighted sum divides 37*

		Sum	Sum of sum
"3"	= 3	3	3
"B"	= 11	14	17
"space"	= 36	50	67
"8"	= 8	58	125
Check-digit	= ??	58	183

$$37 \overline{)\,183} \quad 4,\ 148,\ 35$$

Table 2.8 *Checking the encoded message "3B 82"*

3	3×5	=	15	
B	11×4	=	44	
"space"	36×3	=	108	
8	8×2	=	16	
2	2×1	=	2	
	Sum	=	185	$= 37\times5\equiv0 \mod 37$

The encoded message is therefore given by

3B 82.

To check whether this is a legitimate message at the receiver, we proceed as shown in Table 2.8.

Now suppose "space" is lost during the transmission such that only "3B82" is received. Such an error can be detected since the weighted sum is now not congruent to $0 \mod 37$; see Table 2.9. Similarly, the interchange from "82" to

Table 2.9 *Checking the corrupted message "3B82"*

3	3×4	=	12	
B	11×3	=	33	
8	8×2	=	16	
2	2×1	=	2	
	Sum	=	63	$\not\equiv 0 \mod 37$

Table 2.10 *Checking the corrupted message "3B 28"*

3	3×5	$=$	15		
B	11×4	$=$	44		
"space"	36×3	$=$	108		
2	2×2	$=$	4		
8	8×1	$=$	8		
	Sum	$=$	179	$\not\equiv 0$	mod 37

"28" can also be detected; see Table 2.10. ◊

In the following we give another two examples of error-detecting codes that are based on modular arithmetic and are widely used in daily commerce.

Example 2.7 The *International Standard Book Number (ISBN)* is usually a 10-digit code used to identify a book uniquely. A typical example of the ISBN is as follows:

$$\underbrace{0}_{\substack{\text{country}\\ \text{ID}}} - \underbrace{52}_{\substack{\text{publisher}\\ \text{ID}}} \underbrace{18-4868}_{\substack{\text{book}\\ \text{number}}} - \underbrace{7}_{\substack{\text{check}\\ \text{digit}}}$$

where the hyphens do not matter and may appear in different positions. The first digit stands for the country, with 0 meaning the United States and some other English-speaking countries. The next two digits are the publisher ID; here 52 means Cambridge University Press. The next six digits, 18 – 4868, are the publisher-assigned book number. The last digit is the weighted check sum modulo 11 and is represented by "X" if the required check digit is 10.

To confirm that this number is a legitimate ISBN number we proceed as shown in Table 2.11. It checks! ◊

Exercise 2.8 *Check whether 0 – 8044 – 2957 – X is a valid ISBN number.* ◊

Example 2.9 The *Universal Product Code (UPC)* is a 12-digit single parity-check code employed on the bar codes of most merchandise to ensure reliability in scanning. A typical example of UPC takes the form

$$\underbrace{0\,36000}_{\substack{\text{manufacturer}\\ \text{ID}}} \underbrace{29145}_{\substack{\text{item}\\ \text{number}}} \underbrace{2}_{\substack{\text{parity}\\ \text{check}}}$$

Table 2.11 *Checking the ISBN number 0 – 5218 – 4868 – 7*

	Sum	Sum of sum
0	0	0
5	5	5
2	7	12
1	8	20
8	16	36
4	20	56
8	28	84
6	34	118
8	42	160
7	49	$209 = 11 \times 19 \equiv 0 \bmod 11$

where the last digit is the parity-check digit. Denote the digits as $x_1, x_2, \ldots, x_{12}$. The parity digit x_{12} is determined such that

$$3(x_1 + x_3 + x_5 + x_7 + x_9 + x_{11}) + (x_2 + x_4 + x_6 + x_8 + x_{10} + x_{12}) \tag{2.46}$$

is a multiple[5] of 10. In this case,

$$3(0+6+0+2+1+5) + (3+0+0+9+4+2) = 60. \tag{2.47}$$

$\Diamond$

2.7 Trade-off between redundancy and error-detecting capability

As discussed in the previous sections, a single parity check to make the whole message even-parity can help the detection of any single error (or even any odd number of errors). However, if we want to detect the occurrence of more errors in a noisy channel, what can we do for the design of error-detecting codes? Can such a goal be achieved by increasing the number of parity checks, i.e. at the cost of extra redundancy? Fortunately, the answer is positive. Let us consider the following illustrative example.

[5] Note that in this example the modulus 10 is used although this is not a prime. The slightly unusual summation (2.46), however, makes sure that every single error can still be detected. The reason why UPC chooses 10 as the modulus is that the check digit should also range from 0 to 9 so that it can easily be encoded by the bar code.

Example 2.10 For an information source of eight possible outputs, obviously each output can be represented by a binary 3-tuple, say (x_1, x_2, x_3). Suppose three parity checks x_4, x_5, x_6 are now appended to the original message by the following equations:

$$\begin{cases} x_4 = x_1 + x_2, \\ x_5 = x_1 + x_3, \\ x_6 = x_2 + x_3, \end{cases} \tag{2.48}$$

to form a legitimate codeword $(x_1, x_2, x_3, x_4, x_5, x_6)$. Compared with the single parity-check code, this code increases the excess redundancy from $1/3$ to $3/3$. Let $(y_1, y_2, y_3, y_4, y_5, y_6)$ be the received vector as $(x_1, x_2, x_3, x_4, x_5, x_6)$ is transmitted. If at least one of the following parity-check equations is violated:

$$\begin{cases} y_4 = y_1 + y_2, \\ y_5 = y_1 + y_3, \\ y_6 = y_2 + y_3, \end{cases} \tag{2.49}$$

the occurrence of an error is detected.

For instance, consider the case of a single error in the ith position such that

$$y_i = x_i + 1 \quad \text{and} \quad y_\ell = x_\ell, \quad \forall \ell \in \{1, 2, \ldots, 6\} \setminus \{i\}. \tag{2.50}$$

It follows that

$$\begin{cases} y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3 & \text{if } i = 1, \\ y_4 \neq y_1 + y_2,\ y_6 \neq y_2 + y_3 & \text{if } i = 2, \\ y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } i = 3, \\ y_4 \neq y_1 + y_2 & \text{if } i = 4, \\ y_5 \neq y_1 + y_3 & \text{if } i = 5, \\ y_6 \neq y_2 + y_3 & \text{if } i = 6. \end{cases} \tag{2.51}$$

Therefore, all single errors can be successfully detected. In addition, consider the case of a double error in the ith and jth positions, respectively, such that

$$y_i = x_i + 1, \quad y_j = x_j + 1, \quad \text{and} \quad y_\ell = x_\ell, \quad \forall \ell \in \{1, 2, \ldots, 6\} \setminus \{i, j\}. \tag{2.52}$$

We then have

$$\begin{cases} y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (1,2), \\ y_4 \neq y_1 + y_2,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (1,3), \\ y_5 \neq y_1 + y_3 & \text{if } (i,j) = (1,4), \\ y_4 \neq y_1 + y_2 & \text{if } (i,j) = (1,5), \\ y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (1,6), \\ y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3 & \text{if } (i,j) = (2,3), \\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (2,4), \\ y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (2,5), \\ y_4 \neq y_1 + y_2 & \text{if } (i,j) = (2,6), \\ y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (3,4), \\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (3,5), \\ y_5 \neq y_1 + y_3 & \text{if } (i,j) = (3,6), \\ y_4 \neq y_1 + y_2,\ y_5 \neq y_1 + y_3 & \text{if } (i,j) = (4,5), \\ y_4 \neq y_1 + y_2,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (4,6), \\ y_5 \neq y_1 + y_3,\ y_6 \neq y_2 + y_3 & \text{if } (i,j) = (5,6). \end{cases} \tag{2.53}$$

Hence, this code can detect any pattern of double errors. ◊

Exercise 2.11 *Unfortunately, not all triple errors may be caught by the code of Example 2.10. Can you give an example for verification?* ◊

Without a proper design, however, increasing the number of parity checks may not always improve the error-detecting capability. For example, consider another code which appends the parity checks by

$$\begin{cases} x_4 = x_1 + x_2 + x_3, \\ x_5 = x_1 + x_2 + x_3, \\ x_6 = x_1 + x_2 + x_3. \end{cases} \tag{2.54}$$

In this case, x_5 and x_6 are simply repetitions of x_4. Following a similar discussion as in Example 2.10, we can show that all single errors are still detectable. But if the following double error occurs during the transmission:

$$y_1 = x_1 + 1, \quad y_2 = x_2 + 1, \quad \text{and} \quad y_\ell = x_\ell, \quad \forall 3 \leq \ell \leq 6, \tag{2.55}$$

none of the three parity-check equations corresponding to (2.54) will be violated. This code thus is not double-error-detecting even though the same amount of redundancy is required as in the code (2.48).

2.8 Further reading

In this chapter simple coding schemes, e.g. single parity-check codes, burst error-detecting codes, and weighted codes, have been introduced to detect the presence of channel errors. However, there exists a class of linear block codes, called *cyclic codes*, which are probably the most widely used form of error-detecting codes. The popularity of cyclic codes arises primarily from the fact that these codes can be implemented with extremely cost-effective electronic circuits. The codes themselves also possess a high degree of structure and regularity (which gives rise to the promising advantage mentioned above), and there is a certain beauty and elegance in the corresponding theory. Interested readers are referred to [MS77], [Wic94], and [LC04] for more details of cyclic codes.

References

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

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

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