1

Introduction to Information Theory

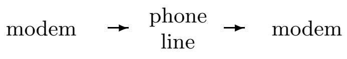
> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.
>
> *(Claude Shannon, 1948)*

In the first half of this book we study how to measure information content; we learn how to compress data; and we learn how to communicate perfectly over imperfect communication channels.

We start by getting a feeling for this last problem.

1.1 How can we achieve perfect communication over an imperfect, noisy communication channel?

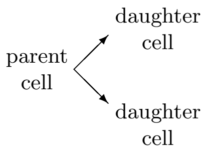
Some examples of noisy communication channels are:



- an analogue telephone line, over which two modems communicate digital information;

Galileo
radio waves
Earth

- the radio communication link from Galileo, the Jupiter-orbiting spacecraft, to earth;



- reproducing cells, in which the daughter cells' DNA contains information from the parent cells;

computer memory
disk drive
computer memory

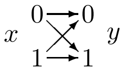
- a disk drive.

The last example shows that communication doesn't have to involve information going from one *place* to another. When we write a file on a disk drive, we'll read it off in the same location – but at a later *time*.

These channels are noisy. A telephone line suffers from cross-talk with other lines; the hardware in the line distorts and adds noise to the transmitted signal. The deep space network that listens to Galileo's puny transmitter receives background radiation from terrestrial and cosmic sources. DNA is subject to mutations and damage. A disk drive, which writes a binary digit (a one or zero, also known as a *bit*) by aligning a patch of magnetic material in one of two orientations, may later fail to read out the stored binary digit: the patch of material might spontaneously flip magnetization, or a glitch of background noise might cause the reading circuit to report the wrong value for the binary digit, or the writing head might not induce the magnetization in the first place because of interference from neighbouring bits.

In all these cases, if we transmit data, e.g., a string of bits, over the channel, there is some probability that the received message will not be identical to the transmitted message. We would prefer to have a communication channel for which this probability was zero – or so close to zero that for practical purposes it is indistinguishable from zero.

Let's consider a noisy disk drive that transmits each bit correctly with probability $(1-f)$ and incorrectly with probability f. This model communication channel is known as the *binary symmetric channel* (figure 1.4).



$$\begin{array}{lcllcl} P(y\!=\!0\,|\,x\!=\!0) & = & 1-f; & P(y\!=\!0\,|\,x\!=\!1) & = & f; \\ P(y\!=\!1\,|\,x\!=\!0) & = & f; & P(y\!=\!1\,|\,x\!=\!1) & = & 1-f. \end{array}$$

Figure 1.4. The binary symmetric channel. The transmitted symbol is x and the received symbol y. The noise level, the probability that a bit is flipped, is f.

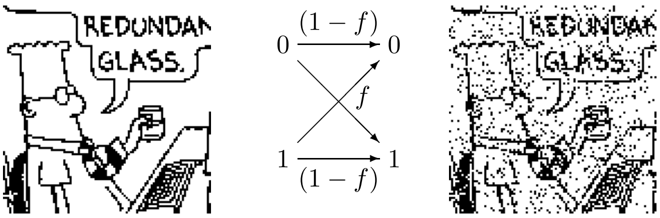


Figure 1.5. A binary data sequence of length 10 000 transmitted over a binary symmetric channel with noise level $f = 0.1$. [Dilbert image Copyright©1997 United Feature Syndicate, Inc., used with permission.]

As an example, let's imagine that $f = 0.1$, that is, ten per cent of the bits are flipped (figure 1.5). A useful disk drive would flip no bits at all in its entire lifetime. If we expect to read and write a gigabyte per day for ten years, we require a bit error probability of the order of 10^{-15}, or smaller. There are two approaches to this goal.

The physical solution

The physical solution is to improve the physical characteristics of the communication channel to reduce its error probability. We could improve our disk drive by

1. using more reliable components in its circuitry;
2. evacuating the air from the disk enclosure so as to eliminate the turbulence that perturbs the reading head from the track;
3. using a larger magnetic patch to represent each bit; or
4. using higher-power signals or cooling the circuitry in order to reduce thermal noise.

These physical modifications typically increase the cost of the communication channel.

The 'system' solution

Information theory and coding theory offer an alternative (and much more exciting) approach: we accept the given noisy channel as it is and add communication *systems* to it so that we can detect and correct the errors introduced by the channel. As shown in figure 1.6, we add an *encoder* before the channel and a *decoder* after it. The encoder encodes the source message $\mathbf{s}$ into a *transmitted* message $\mathbf{t}$, adding *redundancy* to the original message in some way. The channel adds noise to the transmitted message, yielding a received message $\mathbf{r}$. The decoder uses the known redundancy introduced by the encoding system to infer both the original signal $\mathbf{s}$ and the added noise.

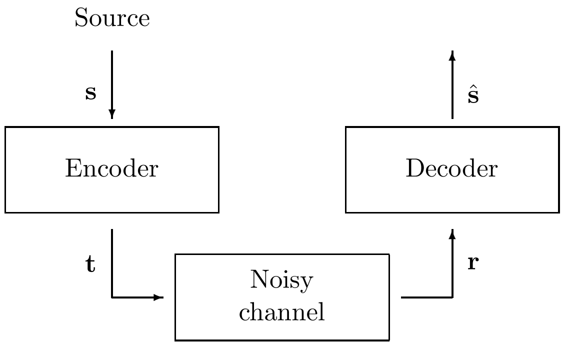


Figure 1.6. The 'system' solution for achieving reliable communication over a noisy channel. The encoding system introduces systematic redundancy into the transmitted vector **t**. The decoding system uses this known redundancy to deduce from the received vector **r** *both* the original source vector *and* the noise introduced by the channel.

Whereas physical solutions give incremental channel improvements only at an ever-increasing cost, system solutions can turn noisy channels into reliable communication channels with the only cost being a *computational* requirement at the encoder and decoder.

Information theory is concerned with the theoretical limitations and potentials of such systems. 'What is the best error-correcting performance we could achieve?'

Coding theory is concerned with the creation of practical encoding and decoding systems.

▶ 1.2 Error-correcting codes for the binary symmetric channel

We now consider examples of encoding and decoding systems. What is the simplest way to add useful redundancy to a transmission? [To make the rules of the game clear: we want to be able to detect *and* correct errors; and retransmission is not an option. We get only one chance to encode, transmit, and decode.]

Repetition codes

A straightforward idea is to repeat every bit of the message a prearranged number of times – for example, three times, as shown in table 1.7. We call this *repetition code* 'R_3'.

Source sequence **s**	Transmitted sequence **t**
0	000
1	111

Table 1.7. The repetition code R_3.

Imagine that we transmit the source message

$$\mathbf{s} = 0\ 0\ 1\ 0\ 1\ 1\ 0$$

over a binary symmetric channel with noise level $f = 0.1$ using this repetition code. We can describe the channel as 'adding' a sparse noise vector **n** to the transmitted vector – adding in modulo 2 arithmetic, i.e., the binary algebra in which 1+1=0. A possible noise vector **n** and received vector $\mathbf{r} = \mathbf{t} + \mathbf{n}$ are shown in figure 1.8.

$$\begin{array}{rccccccc}
\mathbf{s} & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\mathbf{t} & \overbrace{0\,0\,0} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} & \overbrace{1\,1\,1} & \overbrace{1\,1\,1} & \overbrace{0\,0\,0} \\
\mathbf{n} & 0\,0\,0 & 0\,0\,1 & 0\,0\,0 & 0\,0\,0 & 1\,0\,1 & 0\,0\,0 & 0\,0\,0 \\
\hline
\mathbf{r} & 0\,0\,0 & 0\,0\,1 & 1\,1\,1 & 0\,0\,0 & 0\,1\,0 & 1\,1\,1 & 0\,0\,0
\end{array}$$

Figure 1.8. An example transmission using R_3.

How should we decode this received vector? The optimal algorithm looks at the received bits three at a time and takes a majority vote (algorithm 1.9).

Received sequence $\mathbf{r}$	Likelihood ratio $\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)}$	Decoded sequence $\hat{\mathbf{s}}$
000	γ^{-3}	0
001	γ^{-1}	0
010	γ^{-1}	0
100	γ^{-1}	0
101	γ^{1}	1
110	γ^{1}	1
011	γ^{1}	1
111	γ^{3}	1

Algorithm 1.9. Majority-vote decoding algorithm for R_3. Also shown are the likelihood ratios (1.23), assuming the channel is a binary symmetric channel; $\gamma \equiv (1-f)/f$.

At the risk of explaining the obvious, let's prove this result. The optimal decoding decision (optimal in the sense of having the smallest probability of being wrong) is to find which value of $\mathbf{s}$ is most probable, given $\mathbf{r}$. Consider the decoding of a single bit s, which was encoded as $\mathbf{t}(s)$ and gave rise to three received bits $\mathbf{r} = r_1r_2r_3$. By Bayes' theorem, the *posterior probability* of s is

$$P(s\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s)P(s)}{P(r_1r_2r_3)}. \tag{1.18}$$

We can spell out the posterior probability of the two alternatives thus:

$$P(s=1\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=1)P(s=1)}{P(r_1r_2r_3)}; \tag{1.19}$$

$$P(s=0\,|\,r_1r_2r_3) = \frac{P(r_1r_2r_3\,|\,s=0)P(s=0)}{P(r_1r_2r_3)}. \tag{1.20}$$

This posterior probability is determined by two factors: the *prior probability* $P(s)$, and the data-dependent term $P(r_1r_2r_3\,|\,s)$, which is called the *likelihood* of s. The normalizing constant $P(r_1r_2r_3)$ needn't be computed when finding the optimal decoding decision, which is to guess $\hat{s}=0$ if $P(s=0\,|\,\mathbf{r}) > P(s=1\,|\,\mathbf{r})$, and $\hat{s}=1$ otherwise.

To find $P(s=0\,|\,\mathbf{r})$ and $P(s=1\,|\,\mathbf{r})$, we must make an assumption about the prior probabilities of the two hypotheses $s=0$ and $s=1$, and we must make an assumption about the probability of $\mathbf{r}$ given s. We assume that the prior probabilities are equal: $P(s=0) = P(s=1) = 0.5$; then maximizing the posterior probability $P(s\,|\,\mathbf{r})$ is equivalent to maximizing the likelihood $P(\mathbf{r}\,|\,s)$. And we assume that the channel is a binary symmetric channel with noise level $f < 0.5$, so that the likelihood is

$$P(\mathbf{r}\,|\,s) = P(\mathbf{r}\,|\,\mathbf{t}(s)) = \prod_{n=1}^{N} P(r_n\,|\,t_n(s)), \tag{1.21}$$

where $N = 3$ is the number of transmitted bits in the block we are considering, and

$$P(r_n\,|\,t_n) = \begin{cases} (1-f) & \text{if} \quad r_n = t_n \\ f & \text{if} \quad r_n \neq t_n. \end{cases} \tag{1.22}$$

Thus the likelihood ratio for the two hypotheses is

$$\frac{P(\mathbf{r}\,|\,s=1)}{P(\mathbf{r}\,|\,s=0)} = \prod_{n=1}^{N} \frac{P(r_n\,|\,t_n(1))}{P(r_n\,|\,t_n(0))}; \tag{1.23}$$

each factor $\frac{P(r_n|t_n(1))}{P(r_n|t_n(0))}$ equals $\frac{(1-f)}{f}$ if $r_n = 1$ and $\frac{f}{(1-f)}$ if $r_n = 0$. The ratio $\gamma \equiv \frac{(1-f)}{f}$ is greater than 1, since $f < 0.5$, so the winning hypothesis is the one with the most 'votes', each vote counting for a factor of γ in the likelihood ratio.

> Thus the majority-vote decoder shown in algorithm 1.9 is the optimal decoder if we assume that the channel is a binary symmetric channel and that the two possible source messages `0` and `1` have equal prior probability.

We now apply the majority vote decoder to the received vector of figure 1.8. The first three received bits are all `0`, so we decode this triplet as a `0`. In the second triplet of figure 1.8, there are two `0`s and one `1`, so we decode this triplet as a `0` – which in this case corrects the error. Not all errors are corrected, however. If we are unlucky and two errors fall in a single block, as in the fifth triplet of figure 1.8, then the decoding rule gets the wrong answer, as shown in figure 1.10.

$\mathbf{s}$	0	0	1	0	1	1	0
$\mathbf{t}$	000	000	111	000	111	111	000
$\mathbf{n}$	000	001	000	000	101	000	000
$\mathbf{r}$	000	001	111	000	010	111	000
$\hat{\mathbf{s}}$	0	0	1	0	0	1	0
corrected errors		$\star$					
undetected errors					$\star$		

Figure 1.10. Decoding the received vector from figure 1.8.

Exercise 1.2.[*2*, p.16] Show that the error probability is reduced by the use of R_3 by computing the error probability of this code for a binary symmetric channel with noise level f.

The exercise's rating, e.g.'[*2*]', indicates its difficulty: '1' exercises are the easiest. Exercises that are accompanied by a marginal rat are especially recommended. If a solution or partial solution is provided, the page is indicated after the difficulty rating; for example, this exercise's solution is on page 16.

The error probability is dominated by the probability that two bits in a block of three are flipped, which scales as f^2. In the case of the binary symmetric channel with $f = 0.1$, the R_3 code has a probability of error, after decoding, of $p_b \simeq 0.03$ per bit. Figure 1.11 shows the result of transmitting a binary image over a binary symmetric channel using the repetition code.

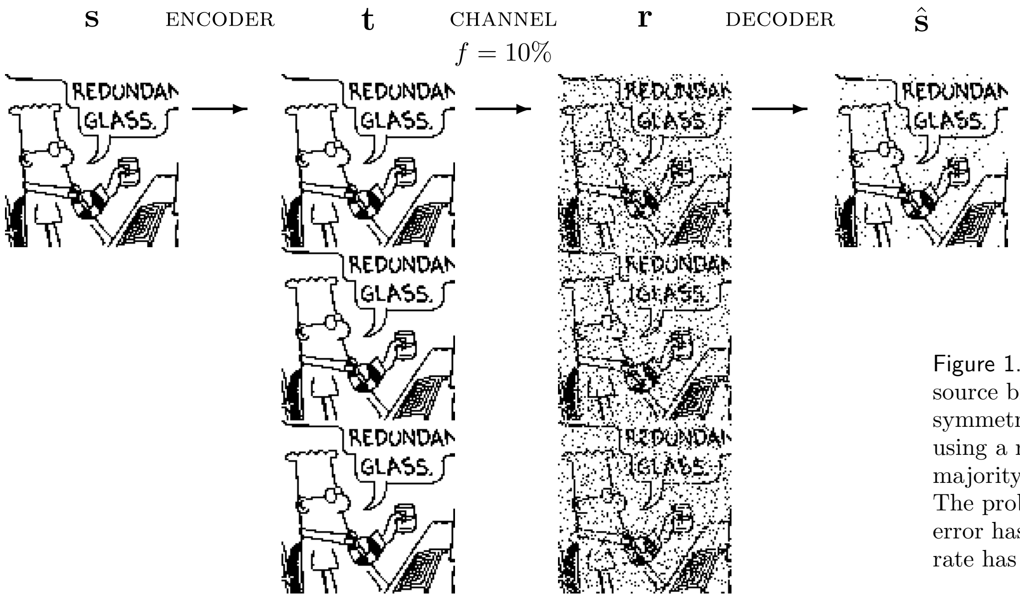


Figure 1.11. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a repetition code and the majority vote decoding algorithm. The probability of decoded bit error has fallen to about 3%; the rate has fallen to 1/3.

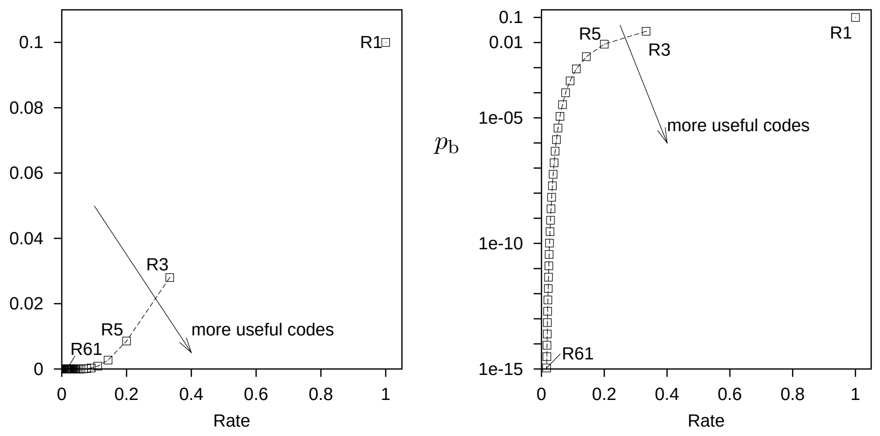


Figure 1.12. Error probability p_b versus rate for repetition codes over a binary symmetric channel with $f = 0.1$. The right-hand figure shows p_b on a logarithmic scale. We would like the rate to be large and p_b to be small.

The repetition code R_3 has therefore reduced the probability of error, as desired. Yet we have lost something: our *rate* of information transfer has fallen by a factor of three. So if we use a repetition code to communicate data over a telephone line, it will reduce the error frequency, but it will also reduce our communication rate. We will have to pay three times as much for each phone call. Similarly, we would need three of the original noisy gigabyte disk drives in order to create a one-gigabyte disk drive with $p_b = 0.03$.

Can we push the error probability lower, to the values required for a sellable disk drive – 10^{-15}? We could achieve lower error probabilities by using repetition codes with more repetitions.

Exercise 1.3.[*3*, p.16] (a) Show that the probability of error of R_N, the repetition code with N repetitions, is

$$p_b = \sum_{n=(N+1)/2}^{N} \binom{N}{n} f^n (1-f)^{N-n}, \qquad (1.24)$$

for odd N.

(b) Assuming $f = 0.1$, which of the terms in this sum is the biggest? How much bigger is it than the second-biggest term?

(c) Use Stirling's approximation (p.2) to approximate the $\binom{N}{n}$ in the largest term, and find, approximately, the probability of error of the repetition code with N repetitions.

(d) Assuming $f = 0.1$, find how many repetitions are required to get the probability of error down to 10^{-15}. [Answer: about 60.]

So to build a *single* gigabyte disk drive with the required reliability from noisy gigabyte drives with $f = 0.1$, we would need *sixty* of the noisy disk drives. The tradeoff between error probability and rate for repetition codes is shown in figure 1.12.

Block codes – the (7, 4) *Hamming code*

We would like to communicate with tiny probability of error *and* at a substantial rate. Can we improve on repetition codes? What if we add redundancy to *blocks* of data instead of encoding one bit at a time? We now study a simple *block code.*

A *block code* is a rule for converting a sequence of source bits $\mathbf{s}$, of length K, say, into a transmitted sequence $\mathbf{t}$ of length N bits. To add redundancy, we make N greater than K. In a *linear* block code, the extra $N - K$ bits are linear functions of the original K bits; these extra bits are called *parity-check bits*. An example of a linear block code is the $(7, 4)$ *Hamming code*, which transmits $N = 7$ bits for every $K = 4$ source bits.

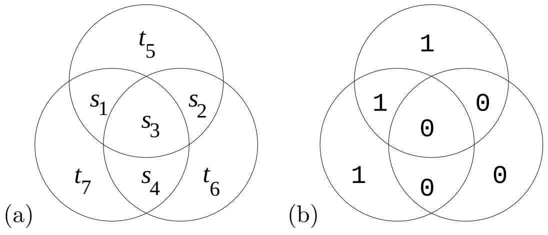


Figure 1.13. Pictorial representation of encoding for the $(7, 4)$ Hamming code.

The encoding operation for the code is shown pictorially in figure 1.13. We arrange the seven transmitted bits in three intersecting circles. The first four transmitted bits, $t_1t_2t_3t_4$, are set equal to the four source bits, $s_1s_2s_3s_4$. The parity-check bits $t_5t_6t_7$ are set so that the *parity* within each circle is even: the first parity-check bit is the parity of the first three source bits (that is, it is 0 if the sum of those bits is even, and 1 if the sum is odd); the second is the parity of the last three; and the third parity bit is the parity of source bits one, three and four.

As an example, figure 1.13b shows the transmitted codeword for the case $\mathbf{s} = 1000$. Table 1.14 shows the codewords generated by each of the 2^4 = sixteen settings of the four source bits. These codewords have the special property that any pair differ from each other in at least three bits.

s	**t**	**s**	**t**	**s**	**t**	**s**	**t**
0000	0000000	0100	0100110	1000	1000101	1100	1100011
0001	0001011	0101	0101101	1001	1001110	1101	1101000
0010	0010111	0110	0110001	1010	1010010	1110	1110100
0011	0011100	0111	0111010	1011	1011001	1111	1111111

Table 1.14. The sixteen codewords $\{\mathbf{t}\}$ of the $(7, 4)$ Hamming code. Any pair of codewords differ from each other in at least three bits.

Because the Hamming code is a linear code, it can be written compactly in terms of matrices as follows. The transmitted codeword $\mathbf{t}$ is obtained from the source sequence $\mathbf{s}$ by a linear operation,

$$\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}, \tag{1.25}$$

where $\mathbf{G}$ is the *generator matrix* of the code,

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}, \tag{1.26}$$

and the encoding operation (1.25) uses modulo-2 arithmetic ($1+1 = 0$, $0+1 = 1$, etc.).

In the encoding operation (1.25) I have assumed that $\mathbf{s}$ and $\mathbf{t}$ are column vectors. If instead they are row vectors, then this equation is replaced by

$$\mathbf{t} = \mathbf{s}\mathbf{G}, \tag{1.27}$$

where

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}. \tag{1.28}$$

I find it easier to relate to the right-multiplication (1.25) than the left-multiplication (1.27). Many coding theory texts use the left-multiplying conventions (1.27–1.28), however.

The rows of the generator matrix (1.28) can be viewed as defining four basis vectors lying in a seven-dimensional binary space. The sixteen codewords are obtained by making all possible linear combinations of these vectors.

Decoding the $(7,4)$ *Hamming code*

When we invent a more complex encoder $\mathbf{s} \rightarrow \mathbf{t}$, the task of decoding the received vector $\mathbf{r}$ becomes less straightforward. Remember that *any* of the bits may have been flipped, including the parity bits.

If we assume that the channel is a binary symmetric channel and that all source vectors are equiprobable, then the optimal decoder identifies the source vector $\mathbf{s}$ whose encoding $\mathbf{t}(\mathbf{s})$ differs from the received vector $\mathbf{r}$ in the fewest bits. [Refer to the likelihood function (1.23) to see why this is so.] We could solve the decoding problem by measuring how far $\mathbf{r}$ is from each of the sixteen codewords in table 1.14, then picking the closest. Is there a more efficient way of finding the most probable source vector?

Syndrome decoding for the Hamming code

For the $(7,4)$ Hamming code there is a pictorial solution to the decoding problem, based on the encoding picture, figure 1.13.

As a first example, let's assume the transmission was $\mathbf{t} = \texttt{1000101}$ and the noise flips the second bit, so the received vector is $\mathbf{r} = \texttt{1000101} \oplus \texttt{0100000} = \texttt{1100101}$. We write the received vector into the three circles as shown in figure 1.15a, and look at each of the three circles to see whether its parity is even. The circles whose parity is *not* even are shown by dashed lines in figure 1.15b. The decoding task is to find the smallest set of flipped bits that can account for these violations of the parity rules. [The pattern of violations of the parity checks is called the *syndrome*, and can be written as a binary vector – for example, in figure 1.15b, the syndrome is $\mathbf{z} = (1,1,0)$, because the first two circles are 'unhappy' (parity 1) and the third circle is 'happy' (parity 0).]

To solve the decoding task, we ask the question: can we find a unique bit that lies *inside* all the 'unhappy' circles and *outside* all the 'happy' circles? If so, the flipping of that bit would account for the observed syndrome. In the case shown in figure 1.15b, the bit r_2 lies inside the two unhappy circles and outside the happy circle; no other single bit has this property, so r_2 is the only single bit capable of explaining the syndrome.

Let's work through a couple more examples. Figure 1.15c shows what happens if one of the parity bits, t_5, is flipped by the noise. Just one of the checks is violated. Only r_5 lies inside this unhappy circle and outside the other two happy circles, so r_5 is identified as the only single bit capable of explaining the syndrome.

If the central bit r_3 is received flipped, figure 1.15d shows that all three checks are violated; only r_3 lies inside all three circles, so r_3 is identified as the suspect bit.

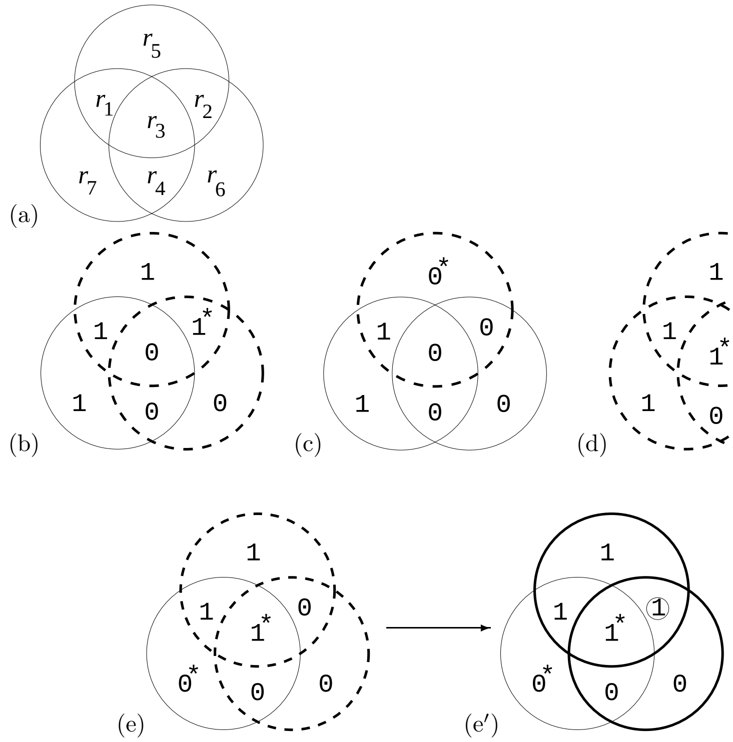


Figure 1.15. Pictorial representation of decoding of the Hamming $(7,4)$ code. The received vector is written into the diagram as shown in (a). In (b,c,d,e), the received vector is shown, assuming that the transmitted vector was as in figure 1.13b and the bits labelled by $\star$ were flipped. The violated parity checks are highlighted by dashed circles. One of the seven bits is the most probable suspect to account for each 'syndrome', i.e., each pattern of violated and satisfied parity checks.
In examples (b), (c), and (d), the most probable suspect is the one bit that was flipped.
In example (e), two bits have been flipped, s_3 and t_7. The most probable suspect is r_2, marked by a circle in (e′), which shows the output of the decoding algorithm.

Syndrome **z**	000	001	010	011	100	101	110	111
Unflip this bit	*none*	r_7	r_6	r_4	r_5	r_1	r_2	r_3

Algorithm 1.16. Actions taken by the optimal decoder for the $(7,4)$ Hamming code, assuming a binary symmetric channel with small noise level f. The syndrome vector **z** lists whether each parity check is violated (1) or satisfied (0), going through the checks in the order of the bits r_5, r_6, and r_7.

If you try flipping any one of the seven bits, you'll find that a different syndrome is obtained in each case – seven non-zero syndromes, one for each bit. There is only one other syndrome, the all-zero syndrome. So if the channel is a binary symmetric channel with a small noise level f, the optimal decoder unflips at most one bit, depending on the syndrome, as shown in algorithm 1.16. Each syndrome could have been caused by other noise patterns too, but any other noise pattern that has the same syndrome must be less probable because it involves a larger number of noise events.

What happens if the noise actually flips more than one bit? Figure 1.15e shows the situation when two bits, r_3 and r_7, are received flipped. The syndrome, 110, makes us suspect the single bit r_2; so our optimal decoding algorithm flips this bit, giving a decoded pattern with three errors as shown in figure 1.15e′. If we use the optimal decoding algorithm, any two-bit error pattern will lead to a decoded seven-bit vector that contains three errors.

General view of decoding for linear codes: syndrome decoding

We can also describe the decoding problem for a linear code in terms of matrices. The first four received bits, $r_1r_2r_3r_4$, purport to be the four source bits; and the received bits $r_5r_6r_7$ purport to be the parities of the source bits, as defined by the generator matrix $\mathbf{G}$. We evaluate the three parity-check bits for the received bits, $r_1r_2r_3r_4$, and see whether they match the three received bits, $r_5r_6r_7$. The differences (modulo 2) between these two triplets are called the *syndrome* of the received vector. If the syndrome is zero – if all three parity checks are happy – then the received vector is a codeword, and the most probable decoding is

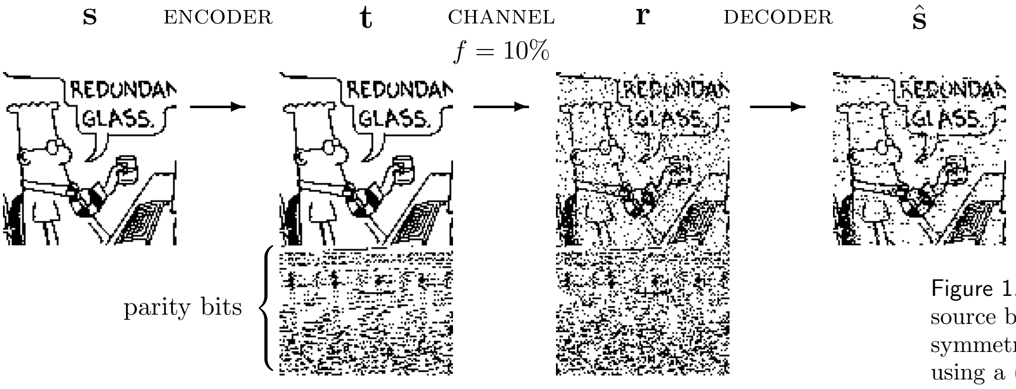


Figure 1.17. Transmitting 10 000 source bits over a binary symmetric channel with $f = 10\%$ using a $(7, 4)$ Hamming code. The probability of decoded bit error is about 7%.

given by reading out its first four bits. If the syndrome is non-zero, then the noise sequence for this block was non-zero, and the syndrome is our pointer to the most probable error pattern.

The computation of the syndrome vector is a linear operation. If we define the 3×4 matrix $\mathbf{P}$ such that the matrix of equation (1.26) is

$$\mathbf{G}^{\mathsf{T}} = \begin{bmatrix} \mathbf{I}_4 \\ \mathbf{P} \end{bmatrix}, \tag{1.29}$$

where $\mathbf{I}_4$ is the 4×4 identity matrix, then the syndrome vector is $\mathbf{z} = \mathbf{H}\mathbf{r}$, where the *parity-check matrix* $\mathbf{H}$ is given by $\mathbf{H} = \begin{bmatrix} -\mathbf{P} & \mathbf{I}_3 \end{bmatrix}$; in modulo 2 arithmetic, $-1 \equiv 1$, so

$$\mathbf{H} = \begin{bmatrix} \mathbf{P} & \mathbf{I}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.30}$$

All the codewords $\mathbf{t} = \mathbf{G}^{\mathsf{T}}\mathbf{s}$ of the code satisfy

$$\mathbf{H}\mathbf{t} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{1.31}$$

▷ Exercise 1.4.[1] Prove that this is so by evaluating the 3×4 matrix $\mathbf{H}\mathbf{G}^{\mathsf{T}}$.

Since the received vector $\mathbf{r}$ is given by $\mathbf{r} = \mathbf{G}^{\mathsf{T}}\mathbf{s} + \mathbf{n}$, the syndrome-decoding problem is to find the most probable noise vector $\mathbf{n}$ satisfying the equation

$$\mathbf{H}\mathbf{n} = \mathbf{z}. \tag{1.32}$$

A decoding algorithm that solves this problem is called a *maximum-likelihood decoder.* We will discuss decoding problems like this in later chapters.

Summary of the $(7, 4)$ Hamming code's properties

Every possible received vector of length 7 bits is either a codeword, or it's one flip away from a codeword.

Since there are three parity constraints, each of which might or might not be violated, there are $2 \times 2 \times 2 = 8$ distinct syndromes. They can be divided into seven non-zero syndromes – one for each of the one-bit error patterns – and the all-zero syndrome, corresponding to the zero-noise case.

The optimal decoder takes no action if the syndrome is zero, otherwise it uses this mapping of non-zero syndromes onto one-bit error patterns to unflip the suspect bit.

There is a *decoding error* if the four decoded bits $\hat{s}_1, \hat{s}_2, \hat{s}_3, \hat{s}_4$ do not all match the source bits s_1, s_2, s_3, s_4. The *probability of block error* $p_{\rm B}$ is the probability that one or more of the decoded bits in one block fail to match the corresponding source bits,

$$p_{\rm B} = P(\hat{\mathbf{s}} \neq \mathbf{s}). \tag{1.33}$$

The *probability of bit error* $p_{\rm b}$ is the average probability that a decoded bit fails to match the corresponding source bit,

$$p_{\rm b} = \frac{1}{K} \sum_{k=1}^{K} P(\hat{s}_k \neq s_k). \tag{1.34}$$

In the case of the Hamming code, a decoding error will occur whenever the noise has flipped more than one bit in a block of seven. The probability of block error is thus the probability that two or more bits are flipped in a block. This probability scales as $O(f^2)$, as did the probability of error for the repetition code R_3. But notice that the Hamming code communicates at a greater rate, $R = 4/7$.

Figure 1.17 shows a binary image transmitted over a binary symmetric channel using the $(7,4)$ Hamming code. About 7% of the decoded bits are in error. Notice that the errors are correlated: often two or three successive decoded bits are flipped.

Exercise 1.5.[1] This exercise and the next three refer to the $(7,4)$ Hamming code. Decode the received strings:

(a) $\mathbf{r} =$ 1101011

(b) $\mathbf{r} =$ 0110110

(c) $\mathbf{r} =$ 0100111

(d) $\mathbf{r} =$ 1111111.

Exercise 1.6.[2, p.17] (a) Calculate the probability of block error $p_{\rm B}$ of the $(7,4)$ Hamming code as a function of the noise level f and show that to leading order it goes as $21f^2$.

(b) [3] Show that to leading order the probability of bit error $p_{\rm b}$ goes as $9f^2$.

Exercise 1.7.[2, p.19] Find some noise vectors that give the all-zero syndrome (that is, noise vectors that leave all the parity checks unviolated). How many such noise vectors are there?

▷ Exercise 1.8.[2] I asserted above that a block decoding error will result whenever two or more bits are flipped in a single block. Show that this is indeed so. [In principle, there might be error patterns that, after decoding, led only to the corruption of the parity bits, with no source bits incorrectly decoded.]

Summary of codes' performances

Figure 1.18 shows the performance of repetition codes and the Hamming code. It also shows the performance of a family of linear block codes that are generalizations of Hamming codes, called BCH codes.

This figure shows that we can, using linear block codes, achieve better performance than repetition codes; but the asymptotic situation still looks grim.

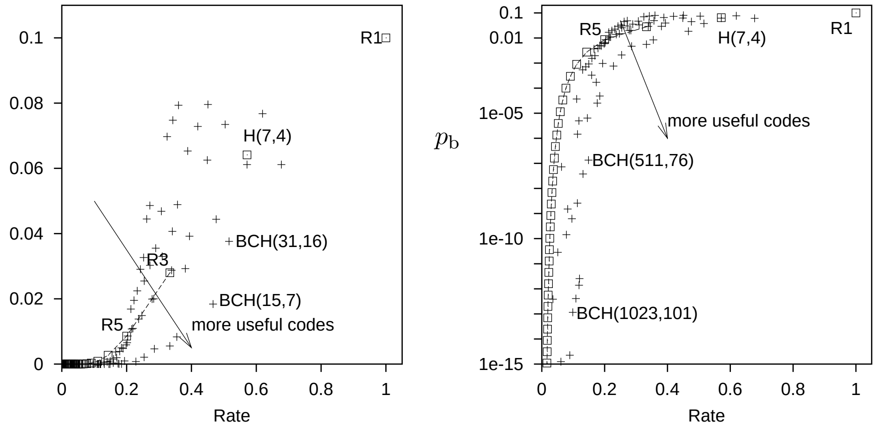


Figure 1.18. Error probability p_b versus rate R for repetition codes, the $(7, 4)$ Hamming code and BCH codes with blocklengths up to 1023 over a binary symmetric channel with $f = 0.1$. The righthand figure shows p_b on a logarithmic scale.

Exercise 1.9.[*4*, p.19] Design an error-correcting code and a decoding algorithm for it, estimate its probability of error, and add it to figure 1.18. [Don't worry if you find it difficult to make a code better than the Hamming code, or if you find it difficult to find a good decoder for your code; that's the point of this exercise.]

Exercise 1.10.[*3*, p.20] A $(7, 4)$ Hamming code can correct any *one* error; might there be a $(14, 8)$ code that can correct any two errors?

Optional extra: Does the answer to this question depend on whether the code is linear or nonlinear?

Exercise 1.11.[*4*, p.21] Design an error-correcting code, other than a repetition code, that can correct any *two* errors in a block of size N.

▶ 1.3 What performance can the best codes achieve?

There seems to be a trade-off between the decoded bit-error probability p_b (which we would like to reduce) and the rate R (which we would like to keep large). How can this trade-off be characterized? What points in the (R, p_b) plane are achievable? This question was addressed by Claude Shannon in his pioneering paper of 1948, in which he both created the field of information theory and solved most of its fundamental problems.

At that time there was a widespread belief that the boundary between achievable and nonachievable points in the (R, p_b) plane was a curve passing through the origin $(R, p_\mathrm{b}) = (0, 0)$; if this were so, then, in order to achieve a vanishingly small error probability p_b, one would have to reduce the rate correspondingly close to zero. 'No pain, no gain.'

However, Shannon proved the remarkable result that the boundary between achievable and nonachievable points meets the R axis at a *non-zero* value $R = C$, as shown in figure 1.19. For any channel, there exist codes that make it possible to communicate with *arbitrarily small* probability of error p_b at non-zero rates. The first half of this book (Parts I–III) will be devoted to understanding this remarkable result, which is called the *noisy-channel coding theorem*.

Example: $f = 0.1$

The maximum rate at which communication is possible with arbitrarily small p_b is called the *capacity* of the channel. The formula for the capacity of a

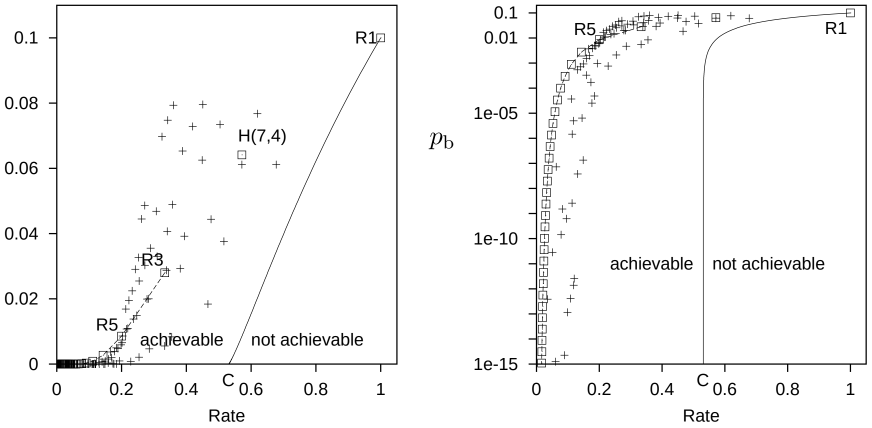


Figure 1.19. Shannon's noisy-channel coding theorem. The solid curve shows the Shannon limit on achievable values of (R, p_b) for the binary symmetric channel with $f = 0.1$. Rates up to $R = C$ are achievable with arbitrarily small p_b. The points show the performance of some textbook codes, as in figure 1.18.

The equation defining the Shannon limit (the solid curve) is $R = C/(1 - H_2(p_b))$, where C and H_2 are defined in equation (1.35).

binary symmetric channel with noise level f is

$$C(f) = 1 - H_2(f) = 1 - \left[f \log_2 \frac{1}{f} + (1-f) \log_2 \frac{1}{1-f} \right]; \qquad (1.35)$$

the channel we were discussing earlier with noise level $f = 0.1$ has capacity $C \simeq 0.53$. Let us consider what this means in terms of noisy disk drives. The repetition code R_3 could communicate over this channel with $p_b = 0.03$ at a rate $R = 1/3$. Thus we know how to build a single gigabyte disk drive with $p_b = 0.03$ from three noisy gigabyte disk drives. We also know how to make a single gigabyte disk drive with $p_b \simeq 10^{-15}$ from sixty noisy one-gigabyte drives (exercise 1.3, p.8). And now Shannon passes by, notices us juggling with disk drives and codes and says:

> 'What performance are you trying to achieve? 10^{-15}? You don't need *sixty* disk drives – you can get that performance with just *two* disk drives (since 1/2 is less than 0.53). And if you want $p_b = 10^{-18}$ or 10^{-24} or anything, you can get there with two disk drives too!'

[Strictly, the above statements might not be quite right, since, as we shall see, Shannon proved his noisy-channel coding theorem by studying sequences of block codes with ever-increasing blocklengths, and the required blocklength might be bigger than a gigabyte (the size of our disk drive), in which case, Shannon might say 'well, you can't do it with those *tiny* disk drives, but if you had two noisy *terabyte* drives, you could make a single high-quality terabyte drive from them'.]

▶ 1.4 Summary

The $(7, 4)$ *Hamming Code*

By including three parity-check bits in a block of 7 bits it is possible to detect and correct any single bit error in each block.

Shannon's noisy-channel coding theorem

Information can be communicated over a noisy channel at a non-zero rate with arbitrarily small error probability.

Information theory addresses both the *limitations* and the *possibilities* of communication. The noisy-channel coding theorem, which we will prove in Chapter 10, asserts both that reliable communication at any rate beyond the capacity is impossible, and that reliable communication at all rates up to capacity is possible.

The next few chapters lay the foundations for this result by discussing *how to measure information content* and the intimately related topic of *data compression*.

▶ 1.5 Further exercises

▷ Exercise 1.12.[*2*, p.21] Consider the repetition code R_9. One way of viewing this code is as a *concatenation* of R_3 with R_3. We first encode the source stream with R_3, then encode the resulting output with R_3. We could call this code 'R_3^2'. This idea motivates an alternative decoding algorithm, in which we decode the bits three at a time using the decoder for R_3; then decode the decoded bits from that first decoder using the decoder for R_3.

Evaluate the probability of error for this decoder and compare it with the probability of error for the optimal decoder for R_9.

Do the concatenated encoder and decoder for R_3^2 have advantages over those for R_9?

▶ 1.6 Solutions

Solution to exercise 1.2 (p.7). An error is made by R_3 if two or more bits are flipped in a block of three. So the error probability of R_3 is a sum of two terms: the probability that all three bits are flipped, f^3; and the probability that exactly two bits are flipped, $3f^2(1-f)$. [If these expressions are not obvious, see example 1.1 (p.1): the expressions are $P(r=3 \mid f, N=3)$ and $P(r=2 \mid f, N=3)$.]

$$p_{\rm b} = p_{\rm B} = 3f^2(1-f) + f^3 = 3f^2 - 2f^3. \tag{1.36}$$

This probability is dominated for small f by the term $3f^2$.

See exercise 2.38 (p.39) for further discussion of this problem.

Solution to exercise 1.3 (p.8). The probability of error for the repetition code R_N is dominated by the probability that $\lceil N/2 \rceil$ bits are flipped, which goes (for odd N) as

$$\binom{N}{\lceil N/2 \rceil} f^{(N+1)/2}(1-f)^{(N-1)/2}. \tag{1.37}$$

Notation: $\lceil N/2 \rceil$ denotes the smallest integer greater than or equal to $N/2$.

The term $\binom{N}{K}$ can be approximated using the binary entropy function:

$$\frac{1}{N+1} 2^{NH_2(K/N)} \leq \binom{N}{K} \leq 2^{NH_2(K/N)} \Rightarrow \binom{N}{K} \simeq 2^{NH_2(K/N)}, \tag{1.38}$$

where this approximation introduces an error of order $\sqrt{N}$ – as shown in equation (1.17). So

$$p_{\rm b} = p_{\rm B} \simeq 2^N (f(1-f))^{N/2} = (4f(1-f))^{N/2}. \tag{1.39}$$

Setting this equal to the required value of 10^{-15} we find $N \simeq 2\frac{\log 10^{-15}}{\log 4f(1-f)} = 68$. This answer is a little out because the approximation we used overestimated $\binom{N}{K}$ and we did not distinguish between $\lceil N/2 \rceil$ and $N/2$.

A slightly more careful answer (short of explicit computation) goes as follows. Taking the approximation for $\binom{N}{K}$ to the next order, we find:

$$\binom{N}{N/2} \simeq 2^N \frac{1}{\sqrt{2\pi N/4}}. \tag{1.40}$$

This approximation can be proved from an accurate version of Stirling's approximation (1.12), or by considering the binomial distribution with $p = 1/2$ and noting

$$1 = \sum_K \binom{N}{K} 2^{-N} \simeq 2^{-N} \binom{N}{N/2} \sum_{r=-N/2}^{N/2} e^{-r^2/2\sigma^2} \simeq 2^{-N} \binom{N}{N/2} \sqrt{2\pi}\sigma, \tag{1.41}$$

where $\sigma = \sqrt{N/4}$, from which equation (1.40) follows. The distinction between $\lceil N/2 \rceil$ and $N/2$ is not important in this term since $\binom{N}{K}$ has a maximum at $K = N/2$.

Then the probability of error (for odd N) is to leading order

$$p_{\rm b} \simeq \binom{N}{(N+1)/2} f^{(N+1)/2}(1-f)^{(N-1)/2} \tag{1.42}$$

$$\simeq 2^N \frac{1}{\sqrt{\pi N/2}} f[f(1-f)]^{(N-1)/2} \simeq \frac{1}{\sqrt{\pi N/8}} f[4f(1-f)]^{(N-1)/2}. \tag{1.43}$$

The equation $p_{\rm b} = 10^{-15}$ can be written

In equation (1.44), the logarithms can be taken to any base, as long as it's the same base throughout. In equation (1.45), I use base 10.

$$(N-1)/2 \simeq \frac{\log 10^{-15} + \log \frac{\sqrt{\pi N/8}}{f}}{\log 4f(1-f)} \tag{1.44}$$

which may be solved for N iteratively, the first iteration starting from $\hat{N}_1 = 68$:

$$(\hat{N}_2 - 1)/2 \simeq \frac{-15 + 1.7}{-0.44} = 29.9 \;\Rightarrow\; \hat{N}_2 \simeq 60.9. \tag{1.45}$$

This answer is found to be stable, so $N \simeq 61$ is the blocklength at which $p_{\rm b} \simeq 10^{-15}$.

Solution to exercise 1.6 (p.13).

(a) The probability of block error of the Hamming code is a sum of six terms – the probabilities that 2, 3, 4, 5, 6, or 7 errors occur in one block.

$$p_{\rm B} = \sum_{r=2}^{7} \binom{7}{r} f^r (1-f)^{7-r}. \tag{1.46}$$

To leading order, this goes as

$$p_{\rm B} \simeq \binom{7}{2} f^2 = 21 f^2. \tag{1.47}$$

(b) The probability of bit error of the Hamming code is smaller than the probability of block error because a block error rarely corrupts all bits in the decoded block. The leading-order behaviour is found by considering the outcome in the most probable case where the noise vector has weight two. The decoder will erroneously flip a *third* bit, so that the modified received vector (of length 7) differs in three bits from the transmitted vector. That means, if we average over all seven bits, the probability that a randomly chosen bit is flipped is 3/7 times the block error probability, to leading order. Now, what we really care about is the probability that a source bit is flipped. Are parity bits or source bits more likely to be among these three flipped bits, or are all seven bits equally likely to be corrupted when the noise vector has weight two? The Hamming code is in fact completely symmetric in the protection it affords to the seven bits (assuming a binary symmetric channel). [This symmetry can be proved by showing that the role of a parity bit can be exchanged with a source bit and the resulting code is still a $(7,4)$ Hamming code; see below.] The probability that any one bit ends up corrupted is the same for all seven bits. So the probability of bit error (for the source bits) is simply three sevenths of the probability of block error.

$$p_{\rm b} \simeq \frac{3}{7} p_{\rm B} \simeq 9f^2. \tag{1.48}$$

Symmetry of the Hamming $(7,4)$ *code*

To prove that the $(7,4)$ code protects all bits equally, we start from the parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.49}$$

The symmetry among the seven transmitted bits will be easiest to see if we reorder the seven bits using the permutation $(t_1 t_2 t_3 t_4 t_5 t_6 t_7) \rightarrow (t_5 t_2 t_3 t_4 t_1 t_6 t_7)$. Then we can rewrite $\mathbf{H}$ thus:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}. \tag{1.50}$$

Now, if we take any two parity constraints that $\mathbf{t}$ satisfies and add them together, we get another parity constraint. For example, row 1 asserts $t_5 + t_2 + t_3 + t_1 = \text{even}$, and row 2 asserts $t_2 + t_3 + t_4 + t_6 = \text{even}$, and the sum of these two constraints is

$$t_5 + 2t_2 + 2t_3 + t_1 + t_4 + t_6 = \text{even}; \tag{1.51}$$

we can drop the terms $2t_2$ and $2t_3$, since they are even whatever t_2 and t_3 are; thus we have derived the parity constraint $t_5 + t_1 + t_4 + t_6 = \text{even}$, which we can if we wish add into the parity-check matrix as a fourth row. [The set of vectors satisfying $\mathbf{Ht} = \mathbf{0}$ will not be changed.] We thus define

$$\mathbf{H}' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{1.52}$$

The fourth row is the sum (modulo two) of the top two rows. Notice that *the second, third, and fourth rows are all cyclic shifts of the top row.* If, having added the fourth redundant constraint, we drop the first constraint, we obtain a new parity-check matrix $\mathbf{H}''$,

$$\mathbf{H}'' = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \tag{1.53}$$

which still satisfies $\mathbf{H}''\mathbf{t} = 0$ for all codewords, and which looks just like the starting $\mathbf{H}$ in (1.50), except that all the columns have shifted along one to the right, and the rightmost column has reappeared at the left (a cyclic permutation of the columns).

This establishes the symmetry among the seven bits. Iterating the above procedure five more times, we can make a total of seven different **H** matrices for the same original code, each of which assigns each bit to a different role.

We may also construct the super-redundant seven-row parity-check matrix for the code,

$$\mathbf{H}''' = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{1.54}$$

This matrix is 'redundant' in the sense that the space spanned by its rows is only three-dimensional, not seven.

This matrix is also a *cyclic* matrix. Every row is a cyclic permutation of the top row.

Cyclic codes: if there is an ordering of the bits $t_1 \ldots t_N$ such that a linear code has a *cyclic* parity-check matrix, then the code is called a *cyclic code*.

The codewords of such a code also have cyclic properties: any cyclic permutation of a codeword is a codeword.

For example, the Hamming $(7,4)$ code, with its bits ordered as above, consists of all seven cyclic shifts of the codewords 1110100 and 1011000, and the codewords 0000000 and 1111111.

Cyclic codes are a cornerstone of the algebraic approach to error-correcting codes. We won't use them again in this book, however, as they have been superceded by sparse-graph codes (Part VI).

Solution to exercise 1.7 (p.13). There are fifteen non-zero noise vectors which give the all-zero syndrome; these are precisely the fifteen non-zero codewords of the Hamming code. Notice that because the Hamming code is *linear*, the sum of any two codewords is a codeword.

Graphs corresponding to codes

Solution to exercise 1.9 (p.14). When answering this question, you will probably find that it is easier to invent new codes than to find optimal decoders for them. There are many ways to design codes, and what follows is just one possible train of thought. We make a linear block code that is similar to the $(7,4)$ Hamming code, but bigger.

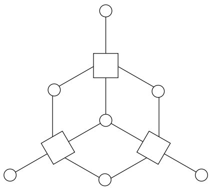


Figure 1.20. The graph of the $(7,4)$ Hamming code. The 7 circles are the bit nodes and the 3 squares are the parity-check nodes.

Many codes can be conveniently expressed in terms of graphs. In figure 1.13, we introduced a pictorial representation of the $(7,4)$ Hamming code. If we replace that figure's big circles, each of which shows that the parity of four particular bits is even, by a 'parity-check node' that is connected to the four bits, then we obtain the representation of the $(7,4)$ Hamming code by a *bipartite graph* as shown in figure 1.20. The 7 circles are the 7 transmitted bits. The 3 squares are the parity-check nodes (not to be confused with the 3 parity-check *bits*, which are the three most peripheral circles). The graph is a 'bipartite' graph because its nodes fall into two classes – bits and checks

– and there are edges only between nodes in different classes. The graph and the code's parity-check matrix (1.30) are simply related to each other: each parity-check node corresponds to a row of $\mathbf{H}$ and each bit node corresponds to a column of $\mathbf{H}$; for every 1 in $\mathbf{H}$, there is an edge between the corresponding pair of nodes.

Having noticed this connection between linear codes and graphs, one way to invent linear codes is simply to think of a bipartite graph. For example, a pretty bipartite graph can be obtained from a dodecahedron by calling the vertices of the dodecahedron the parity-check nodes, and putting a transmitted bit on each edge in the dodecahedron. This construction defines a parity-check matrix in which every column has weight 2 and every row has weight 3. [The weight of a binary vector is the number of 1s it contains.]

This code has $N = 30$ bits, and it appears to have $M_{\text{apparent}} = 20$ parity-check constraints. Actually, there are only $M = 19$ *independent* constraints; the 20th constraint is redundant (that is, if 19 constraints are satisfied, then the 20th is automatically satisfied); so the number of source bits is $K = N - M = 11$. The code is a $(30, 11)$ code.

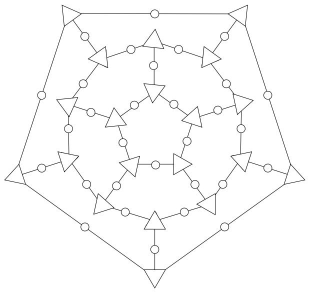


Figure 1.21. The graph defining the $(30, 11)$ dodecahedron code. The circles are the 30 transmitted bits and the triangles are the 20 parity checks. One parity check is redundant.

It is hard to find a decoding algorithm for this code, but we can estimate its probability of error by finding its lowest-weight codewords. If we flip all the bits surrounding one face of the original dodecahedron, then all the parity checks will be satisfied; so the code has 12 codewords of weight 5, one for each face. Since the lowest-weight codewords have weight 5, we say that the code has distance $d = 5$; the $(7, 4)$ Hamming code had distance 3 and could correct all single bit-flip errors. A code with distance 5 can correct all double bit-flip errors, but there are some triple bit-flip errors that it cannot correct. So the error probability of this code, assuming a binary symmetric channel, will be dominated, at least for low noise levels f, by a term of order f^3, perhaps something like

$$12\binom{5}{3} f^3(1-f)^{27}. \tag{1.55}$$

Of course, there is no obligation to make codes whose graphs can be represented on a plane, as this one can; the best linear codes, which have simple graphical descriptions, have graphs that are more tangled, as illustrated by the tiny $(16, 4)$ code of figure 1.22.

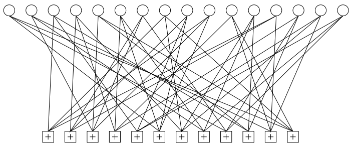


Figure 1.22. Graph of a rate-$1/4$ low-density parity-check code (Gallager code) with blocklength $N = 16$, and $M = 12$ parity-check constraints. Each white circle represents a transmitted bit. Each bit participates in $j = 3$ constraints, represented by ⊞ squares. The edges between nodes were placed at random. (See Chapter 47 for more.)

Furthermore, there is no reason for sticking to linear codes; indeed some nonlinear codes – codes whose codewords cannot be defined by a linear equation like $\mathbf{Ht} = \mathbf{0}$ – have very good properties. But the encoding and decoding of a nonlinear code are even trickier tasks.

Solution to exercise 1.10 (p.14). First let's assume we are making a linear code and decoding it with syndrome decoding. If there are N transmitted bits, then the number of possible error patterns of weight up to two is

$$\binom{N}{2} + \binom{N}{1} + \binom{N}{0}. \tag{1.56}$$

For $N = 14$, that's $91 + 14 + 1 = 106$ patterns. Now, every distinguishable error pattern must give rise to a distinct syndrome; and the syndrome is a list of M bits, so the maximum possible number of syndromes is 2^M. For a $(14, 8)$ code, $M = 6$, so there are at most $2^6 = 64$ syndromes. The number of possible error patterns of weight up to two, 106, is bigger than the number of syndromes, 64, so we can immediately rule out the possibility that there is a $(14, 8)$ code that is 2-error-correcting.

The same counting argument works fine for nonlinear codes too. When the decoder receives $\mathbf{r} = \mathbf{t} + \mathbf{n}$, his aim is to deduce both $\mathbf{t}$ and $\mathbf{n}$ from $\mathbf{r}$. If it is the case that the sender can select any transmission $\mathbf{t}$ from a code of size $S_{\mathbf{t}}$, and the channel can select any noise vector from a set of size $S_{\mathbf{n}}$, and those two selections can be recovered from the received bit string $\mathbf{r}$, which is one of at most 2^N possible strings, then it must be the case that

$$S_{\mathbf{t}} S_{\mathbf{n}} \leq 2^N. \tag{1.57}$$

So, for a (N, K) two-error-correcting code, whether linear or nonlinear,

$$2^K \left[\binom{N}{2} + \binom{N}{1} + \binom{N}{0} \right] \leq 2^N. \tag{1.58}$$

Solution to exercise 1.11 (p.14). There are various strategies for making codes that can correct multiple errors, and I strongly recommend you think out one or two of them for yourself.

If your approach uses a linear code, e.g., one with a collection of M parity checks, it is helpful to bear in mind the counting argument given in the previous exercise, in order to anticipate how many parity checks, M, you might need.

Examples of codes that can correct any two errors are the $(30, 11)$ dodecahedron code on page 20, and the $(15, 6)$ pentagonful code to be introduced on p. 221. Further simple ideas for making codes that can correct multiple errors from codes that can correct only one error are discussed in section 13.7.

Solution to exercise 1.12 (p.16). The probability of error of R_3^2 is, to leading order,

$$p_{\mathrm{b}}(\mathrm{R}_3^2) \simeq 3\,[p_{\mathrm{b}}(\mathrm{R}_3)]^2 = 3(3f^2)^2 + \cdots = 27f^4 + \cdots, \tag{1.59}$$

whereas the probability of error of R_9 is dominated by the probability of five flips,

$$p_{\mathrm{b}}(\mathrm{R}_9) \simeq \binom{9}{5} f^5(1-f)^4 \simeq 126f^5 + \cdots. \tag{1.60}$$

The R_3^2 decoding procedure is therefore suboptimal, since there are noise vectors of weight four that cause it to make a decoding error.

It has the advantage, however, of requiring smaller computational resources: only memorization of three bits, and counting up to three, rather than counting up to nine.

This simple code illustrates an important concept. Concatenated codes are widely used in practice because concatenation allows large codes to be implemented using simple encoding and decoding hardware. Some of the best known practical codes are concatenated codes.