Introduction

In 1948, in the introduction to his classic paper, "A mathematical theory of communication," Claude Shannon[1,*] wrote:

> *"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point."*

To solve that problem he created, in the pages that followed, a completely new branch of applied mathematics, which is today called *information theory* and/or *coding theory*. This book's object is the presentation of the main results of this theory as they stand 30 years later.

In this introductory chapter we illustrate the central ideas of information theory by means of a specific pair of mathematical models, the *binary symmetric source* and the *binary symmetric channel*.

The binary symmetric source (the source, for short) is an object which emits one of two possible symbols, which we take to be "0" and "1," at a rate of R symbols per unit of time. We shall call these symbols *bits*, an abbreviation of *binary digits*. The bits emitted by the source are random, and a "0" is as likely to be emitted as a "1." We imagine that the source rate R is continuously variable, that is, R can assume any nonnegative value.

The binary symmetric channel (the *BSC*[2] for short) is an object through which it is possible to transmit one bit per unit of time. However, the channel is not completely reliable: there is a fixed probability p (called the *raw bit error probability*[3]), $0 \leqslant p \leqslant \frac{1}{2}$, that the output bit will not be the same as the input bit.

We now imagine two individuals, the sender and the receiver. The sender must try to convey to the receiver as accurately as possible the source output,

* Notes, denoted by superior numerals, appear at the end of most chapters.

and the only communication link allowed between the two is the BSC described above. (However, we will allow the sender and receiver to get together before the source is turned on, so that each will know the nature of the data-processing strategies the other will be using.) We assume that both the sender and receiver have access to unlimited amounts of computing power, storage capacity, government funds, and other resources.

We now ask, For a given source rate R, how accurately can the sender communicate with the receiver over the BSC? We shall eventually give a very precise general answer to this question, but let's begin by considering some special cases.

Suppose $R = 1/3$. This means that the channel can transmit bits three times as fast as the source produces them, so the source output can be *encoded* before transmission by repeating each bit three times. For example, if the source's first five bits were 10100, the encoded stream would be 111000111000000. The receiver will get three versions of each source bit, but because of the channel "noise" these versions may not all be the same. If the channel garbled the second, fifth, sixth, twelfth, and thirteenth transmitted bits, the receiver would receive 101011111001100. A little thought should convince you that in this situation the receiver's best strategy for *decoding* a given source bit is to take the majority vote of the three versions of it. In our example he would decode the received message as 11100, and would make an error in the second bit. In general, a source bit will be received in error if either two or three of its three copies are garbled by the channel. Thus, if P_e denotes the *bit error probability*,

$$\begin{aligned} P_e &= P\{2 \text{ channel errors}\} + P\{3 \text{ channel errors}\} \\ &= 3p^2(1-p) \qquad\quad + p^3 \\ &= 3p^2 - 2p^3. \end{aligned} \tag{0.1}$$

Since $p \leqslant \frac{1}{2}$, this is less than the raw bit error probability p; our simple coding scheme has improved the channel's reliability, and for very small p the relative improvement is dramatic.

It is now easy to see that even higher reliability can be achieved by repeating each bit more times. Thus, if $R = 1/(2n+1)$ for some integer n, we could repeat each bit $2n+1$ times before transmission (see Prob. 0.2) and use majority-vote decoding as before. It is simple to obtain a formula for the resulting bit error probability $P_e^{(2n+1)}$:

$$P_e^{(2n+1)} = \sum_{k=n+1}^{2n+1} P\{k \text{ channel errors out of } 2n+1 \text{ transmitted bits}\}$$

$$= \sum_{k=n+1}^{2n+1} \binom{2n+1}{k} p^k (1-p)^{2n+1-k}$$

$$= \binom{2n+1}{n+1} p^{n+1} + \text{terms of higher degree in } p. \tag{0.2}$$

If $n > 1$, this approaches 0 much more rapidly as $p \to 0$ than the special case $n = 1$ considered above.[4] So in this rather weak sense the longer repetition schemes are more powerful than the shorter ones. However, we would like to make the stronger assertion that, for a fixed BSC with a fixed raw error probability $p < \frac{1}{2}$, $P_e^{(2n+1)} \to 0$ as $n \to \infty$, that is, by means of these repetition schemes the channel can be made as reliable as desired. It is possible but not easy to do this by studying formula (0.2) for $P_e^{(2n+1)}$. We shall use another approach and invoke the *weak law of large numbers*,* which implies that, if N bits are transmitted over the channel, then for any $\varepsilon > 0$

$$\lim_{N\to\infty} P\left\{\left|\frac{\text{number of channel errors}}{N} - p\right| > \varepsilon\right\} = 0. \tag{0.3}$$

In other words, for large N, the fraction of bits received in error is unlikely to differ substantially from p. Thus we can make the following estimate of $P_e^{(2n+1)}$:

$$P_e^{(2n+1)} = P\Big\{\text{fraction of transmitted bits received in error}$$

$$\geqslant \frac{n+1}{2n+1} = \frac{1}{2} + \frac{1}{4n+2}\Big\}$$

$$\leqslant P\{\text{fraction} > \tfrac{1}{2}\}$$

$$\leqslant P\{|\text{fraction} - p| > \tfrac{1}{2} - p\},$$

and so by (0.3) $P_e^{(2n+1)}$ does approach 0 as $n \to \infty$. We have thus reached the conclusion that if R is very small, it is possible to make the overall error probability very small as well, even though the channel itself is quite noisy. This is of course not particularly surprising.

* Discussed in Appendix A.

So much, temporarily, for rates less than 1. What about rates larger than 1? How accurately can we communicate under those circumstances?

If $R > 1$, we could, for example, merely transmit the fraction $1/R$ of the source bits and require the receiver to guess the rest of the bits, say by flipping an unbiased coin. For this not-very-bright scheme it is easy to calculate that the resulting bit error probability would be

$$\begin{aligned} P_e &= \frac{1}{R} \times p + \frac{R-1}{R} \times \frac{1}{2} \\ &= \tfrac{1}{2} - \left(\tfrac{1}{2} - p\right)/R. \end{aligned} \tag{0.4}$$

Another, less uninspired method which works for some values of $R > 1$ will be illustrated for $R = 3$. If $R = 3$ there is time to transmit only one third of the bits emitted by the source over the channel. So the sender divides the source bits into blocks of three and transmits only the majority-vote of the three. For example if the source emits 101110101000101, the sender will transmit 11101 over the channel. The receiver merely triples each received bit. In the present case if the channel garbled the second transmitted bit he would receive 10101, which he would expand to 111000111000111, thereby making five bit errors. In general, the resulting bit error probability turns out to be

$$\begin{aligned} P_e &= \tfrac{1}{4} \times (1 - p) + \tfrac{3}{4} \times p \\ &= \tfrac{1}{4} + p/2. \end{aligned} \tag{0.5}$$

Notice that this is less than $\frac{1}{3} + p/3$, which is what our primitive "coin-flipping" strategy gives for $R = 3$. The generalization of this strategy to other integral values of R is left as an exercise (see Prob. 0.4).

The schemes we have considered so far have been trivial, though perhaps not completely uninteresting. Let us now give an example which is much less trivial and in fact was unknown before 1948.

We assume now that $R = 4/7$, so that for every four bits emitted by the source there is just time to send three extra bits over the channel. We choose these extra bits very carefully: if the four source bits are denoted by x_0, x_1, x_2, x_3, then the extra or *redundant* or *parity-check* bits, labeled x_4, x_5, x_6, are determined by the equations

$$\begin{aligned} x_4 &\equiv x_1 + x_2 + x_3 \pmod 2, \\ x_5 &\equiv x_0 + x_2 + x_3 \pmod 2, \\ x_6 &\equiv x_0 + x_1 + x_3 \pmod 2. \end{aligned} \tag{0.6}$$

Thus, for example, if $(x_0, x_1, x_2, x_3) = (0110)$, then $(x_4, x_5, x_6) = (011)$, and the complete seven-bit *codeword* which would be sent over the channel is 0110011.

To describe how the receiver makes his estimate of the four source bits from a garbled seven-bit codeword, that is, to describe his *decoding algorithm*, let us rewrite the parity-check equations (0.6) in the following way:

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 \qquad\quad &= 0, \\ x_0 \quad + x_2 + x_3 \quad + x_5 \quad &= 0, \\ x_0 + x_1 \quad + x_3 \quad\quad + x_6 &= 0. \end{aligned} \tag{0.7}$$

(In (0.7) it is to be understood that the arithmetic is modulo 2.) Stated in a slightly different way, if the binary matrix H is defined by

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

we see that each of the 16 possible codewords $\mathbf{x} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6)$ satisfies the matrix-vector equation

$$H\mathbf{x}^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}. \tag{0.8}$$

(In (0.8) the superscript T means "transpose.")

It turns out to be fruitful to imagine that the BSC adds (mod 2) either a 0 or a 1 to each transmitted bit, 0 if the bit is not received in error and 1 if it is. Thus if $\mathbf{x} = (x_0, x_1, \ldots, x_6)$ is transmitted, the received vector is $\mathbf{y} = (x_0 + z_0,\ x_1 + z_1, \ldots, x_6 + z_6)$, where $z_i = 1$ if the channel caused an error in the ith coordinate and $z_i = 0$ if not. Thus, if $\mathbf{z} = (z_0, \ldots, z_6)$ denotes the *error pattern*, then $\mathbf{y} = \mathbf{x} + \mathbf{z}$.

The receiver, who knows only $\mathbf{y}$ but wants to know $\mathbf{x}$, now does a very clever thing: he computes the following vector $\mathbf{s} = (s_0, s_1, s_2)$:

$$\begin{aligned} \mathbf{s}^T &= H\mathbf{y}^T \\ &= H(\mathbf{x} + \mathbf{z})^T \\ &= H\mathbf{x}^T + H\mathbf{z}^T \\ &= H\mathbf{z}^T \qquad \text{(see (0.8))}. \end{aligned} \tag{0.9}$$

Here $\mathbf{s}$ is called the *syndrome*[5] of $\mathbf{y}$; a 0 component in the syndrome indicates

that the corresponding parity-check equation is satisfied by $\mathbf{y}$, a 1 indicates that it is not. According to (0.9), the syndrome does not depend on which codeword was sent, but only on the error pattern $\mathbf{z}$. However, since $\mathbf{x} = \mathbf{y} + \mathbf{z}$, if the receiver can find $\mathbf{z}$ he will know $\mathbf{x}$ as well, and so he focuses on the problem of finding $\mathbf{z}$. The equation $\mathbf{s}^T = \mathrm{H}\mathbf{z}^T$ shows that $\mathbf{s}^T$ is the (binary) sum of those columns of H corresponding to 1's in $\mathbf{z}$, that is, corresponding to the bits of the codeword that were garbled by the channel:

$$\mathbf{s}^T = z_0 \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + z_1 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \cdots + z_6 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{0.10}$$

The receiver's task, once he has computed $\mathbf{s}$, is to "solve" the equation $\mathbf{s}^T = H\mathbf{z}^T$ for $\mathbf{z}$. Unfortunately, this is only three equations in seven unknowns, and for any $\mathbf{s}$ there will always be 16 possibilities for $\mathbf{z}$. This is clearly progress, since there were a priori 128 possibilities for $\mathbf{z}$, but how can the receiver choose among the remaining 16? For example, suppose $\mathbf{y} = (0111001)$ was received. Then $\mathbf{s} = (101)$, and the 16 candidate $\mathbf{z}$'s turn out to be:

0	1	0	0	0	0	0		0	0	1	0	0	1	1
1	1	0	0	0	1	1		0	0	0	1	0	1	0
0	0	0	0	1	0	1		0	1	1	1	0	0	1
0	1	1	0	1	1	0		1	0	1	0	0	0	0
0	1	0	1	1	1	1		1	0	0	1	0	0	1
1	0	0	0	1	1	0		1	1	1	1	0	1	0
1	1	1	0	1	0	1		0	0	1	1	1	0	0
1	1	0	1	1	0	0		1	0	1	1	1	1	1

Faced with this set of possible error patterns, it is fairly obvious what to do: since the raw bit error probability p is $< \frac{1}{2}$, the fewer 1's (errors) in an error pattern, the more likely it is to have been the actual error pattern. In the current example, we're lucky: there is a unique error pattern (0100000) of least weight, the weight being the number of 1's. So in this case the receiver's best estimate of $\mathbf{z}$ (based both on the syndrome and on the channel statistics) is $\mathbf{z} = (0100000)$; the estimate of the transmitted codeword is $\mathbf{x} = \mathbf{y} + \mathbf{z} = (0011001)$; and finally, the estimate of the four source bits is (0011).

Of course we weren't really lucky in the above example, since we can show that for any syndrome $\mathbf{s}$ there will always be a unique solution to $H\mathbf{z}^T = \mathbf{s}^T$ of weight 0 or 1. To see this, notice that if $\mathbf{s} = (000)$, then $\mathbf{z} = (0000000)$ is the desired solution. But if $\mathbf{s} \neq (000)$, then $\mathbf{s}^T$ must occur as one of the columns

of H; if $\mathbf{s}^T =$ the ith column of H, then the error pattern $\mathbf{z}$, which has one 1 in the ith position and 0's elsewhere, is the unique minimum-weight solution to $H\mathbf{z}^T = \mathbf{s}^T$.

We can now formally describe a *decoding algorithm* for this scheme, which is called the (7, 4) *Hamming code*. Given the received vector $\mathbf{y}$, the receiver executes the following steps:

1. Compute the syndrome $\mathbf{s}^T = H\mathbf{y}^T$.
2. If $\mathbf{s} = \mathbf{0}$, set $\hat{\mathbf{z}} = 0$; go to 4.
3. Locate the unique column of H which is equal to $\mathbf{s}$; call it column i; set $\hat{\mathbf{z}} =$ all 0's except for a single 1 in the ith coordinate.
4. Set $\hat{\mathbf{x}} = \mathbf{y} + \hat{\mathbf{z}}$. (This is the decoder's estimate of the transmitted codeword.)
5. Output $(\hat{x}_0, \hat{x}_1, \hat{x}_2, \hat{x}_3)$, the first four components of $\mathbf{x}$. (This is the decoder's estimate of the original source bits.)

It is of course possible that the vector $\hat{\mathbf{z}}$ produced by this algorithm will not be equal to the actual error pattern $\mathbf{z}$. However, if the channel causes at most one error, that is, if the weight of $\mathbf{z}$ is 0 or 1, then it follows from the above discussion that $\hat{\mathbf{z}} = \mathbf{z}$. Thus the Hamming code is a *single-error-correcting code*. In fact it is easy to see that the above decoding algorithm will fail to correctly identify the original codeword $\mathbf{x}$ iff the channel causes two or more errors. Thus, if P_E denotes the *block* error probability $P\{\hat{\mathbf{x}} \neq \mathbf{x}\}$,

$$P_E = \sum_{k=2}^{7} \binom{7}{k} p^k (1-p)^{7-k}$$

$$= 21p^2 - 70p^3 + \text{etc.}$$

Of course the block error probability P_E doesn't tell the whole story, for even if $\hat{\mathbf{x}} \neq \mathbf{x}$, some of the components of $\hat{\mathbf{x}}$ may nevertheless be right. If we denote the bit error probability $P\{\hat{x}_i \neq x_i\}$ by $P_e^{(i)}$, it is possible to show that, for all $0 \leqslant i \leqslant 6$,

$$P_e^{(i)} = 9p^2(1-p)^5 + 19p^3(1-p)^4 + 16p^4(1-p)^3$$

$$+ 12p^5(1-p)^2 + 7p^6(1-p) + p^7$$

$$= 9p^2 - 26p^3 + \text{etc.} \tag{0.11}$$

Comparing this to (0.1), we see that for BSC's with very small raw error probabilities the Hamming code performs at rate $4/7 = 0.571$ about as well as the crude repetition scheme at rate $1/3 = 0.333$.

We could also use the (7, 4) Hamming code to communicate at $R = 7/4$ by reversing the roles of sender and receiver. Here the sender would partition the sequence of source bits into blocks of seven, reduce each block of seven to only four via the above decoding algorithm (which in this context would become an "encoding algorithm"), and transmit these four bits over the channel. The receiver would decode the four received bits by adding three extra bits, computed by the parity-check rules (0.6). For this scheme the resulting bit error probability $P_e^{(i)} = P\{\hat{x}_i \neq x_i\}$ is not independent of i, but the average $P_e = \left(\sum_{i=0}^{6} P_e^{(i)}\right)/7$ is given by

$$P_e = \frac{1}{8}(1-p)^4 + \frac{53}{28}(1-p)^3 p + 3(1-p)^2 p^2 + \frac{59}{28}(1-p)p^3 + \frac{7}{8}p^4$$

$$= \frac{1}{8} + \frac{39}{28}p + \text{etc.} \tag{0.12}$$

For a noiseless ($p = 0$) BSC, this is much superior, for example, to the "coin-flipping" technique for $R = 7/4$, which from (0.4) gives $P_e = \frac{3}{14} = .214$.

Let us summarize what we know so far by specializing to a particular BSC, say $p = .1$, and for each of the communication schemes discussed so far placing a point on the (x, y) plane, with $x = R$, the rate, and $y = P_e$, the overall bit error probability, as shown in Fig. 0.1. Given sufficient patience and ingenuity, we could continue inventing ad hoc schemes and putting points on Fig. 0.1. Our eventual goal would be, of course, to learn which points are achievable and which are not. Incredibly, this goal has already been reached by Shannon. But before giving Shannon's result, let us formalize somewhat the concept of a rate R coding scheme with associated bit error probability P_e.

As suggested by Fig. 0.2, an (n, k) code is a scheme in which the source sequence is partitioned into blocks of k bits, and in which each k-bit source $\mathbf{u}$ block is mapped ("encoded") into an n-bit codeword $\mathbf{x}$, which is transmitted over the channel and received, possibly garbled, as $\mathbf{y}$. The decoder maps the n-bit noisy codeword $\mathbf{y}$ into a k-bit block $\mathbf{v}$, which is an estimate of the original source sequence $\mathbf{u}$. The *rate* of this communication system is $R = k/n$; the bit error probability is defined as

$$P_e = \frac{1}{k}\sum_{i=1}^{k} P_e^{(i)},$$

where

$$P_e^{(i)} = P\{v_i \neq u_i\}, \qquad i = 1, 2, \ldots, k.$$

(You should be able to see immediately how each of the schemes described so

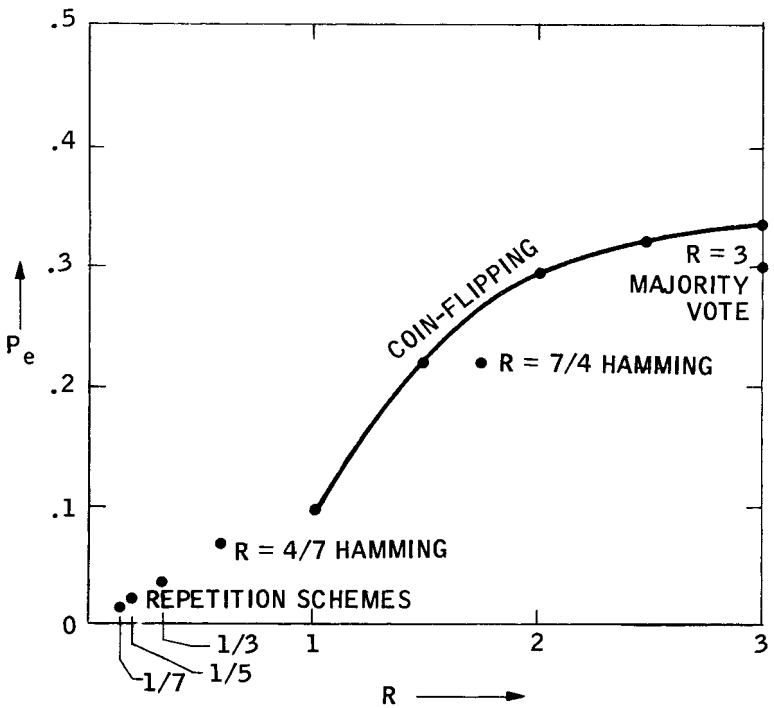

Figure 0.1 Some achievable (R, P_e) pairs for a BSC with $p = .1$.

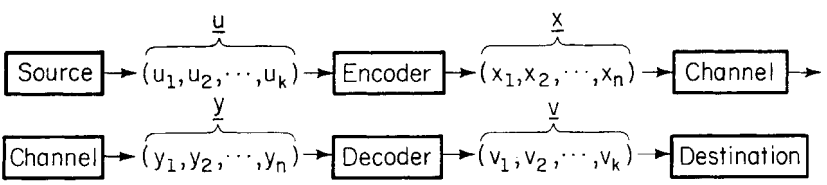

Figure 0.2 An (n, k) code for the binary symmetric source and BSC.

far, with the possible exception of the "coin-flipping" strategy for $R \geqslant 1$, fits this description; see Prob. 0.5.) We say that a point (x, y) in Fig. 0.1 is "achievable" if there exists such an (n, k) code with $k/n \geqslant x$, $P_e \leqslant y$. Not to prolong the suspense, Fig. 0.3 shows the set of achievable points for our special BSC ($p = .1$). Of course the crucial thing to know about Fig. 0.3 is the description of the boundary between the achievable and nonachievable regions. In order to give the description, we need to introduce the important *binary entropy function*:

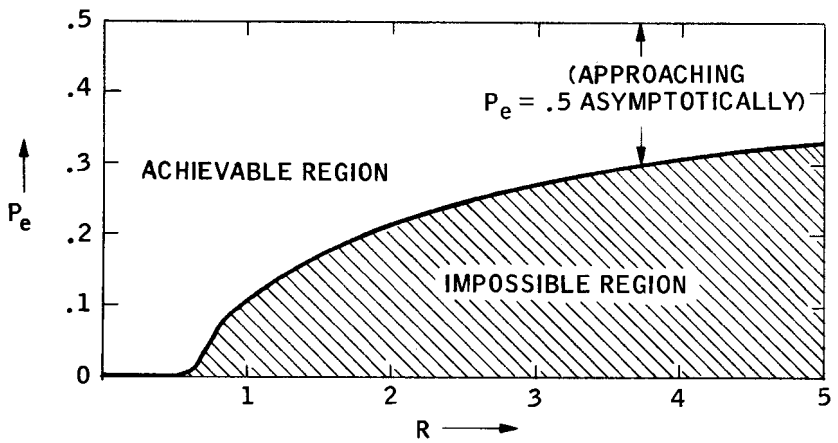

Figure 0.3 The achievable $(R,\ P_e)$ pairs for a binary symmetric source and a BSC $(p = .1)$.

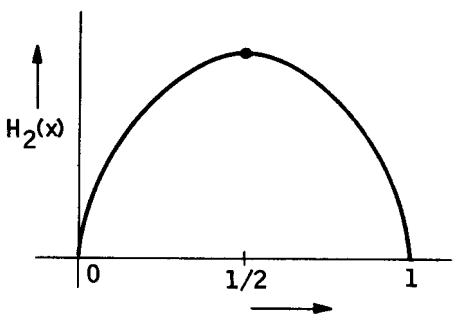

Figure 0.4 The binary entropy function.

$$H_2(x) = -x\log_2 x - (1-x)\log_2(1-x), \qquad 0 < x < 1,$$

$$H_2(0) = H_2(1) = 0. \tag{0.13}$$

A graph of $y = H_2(x)$ is shown in Fig. 0.4. (Some important properties of $H_2(x)$ are described in Prob. 0.10.) We can now describe the boundary between the achievable and nonachieveable regions in Fig. 0.3. The curved part of the boundary is the set of points $(R,\ P_e)$ satisfying

$$R = \frac{1 - H_2(.1)}{1 - H_2(P_e)}, \qquad 0 \leqslant P_e < \tfrac{1}{2}. \tag{0.14}$$

The remainder of the boundary is a segment of the R axis, from $R = 0$ to