Parabola Volume 38, Issue 1 (2002)

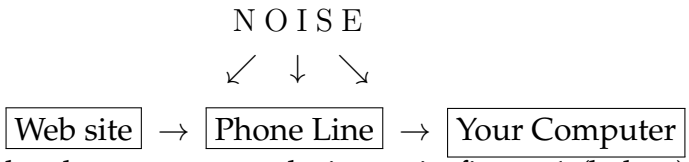
CORRECTING ERRORS

Rodney K. James[1]

Thiz iz a nartical on a voy ding miss takes.

Before you complain about the spelling in the above sentence, read it aloud and you will see that it makes sense (I hope!). Similarly, when you are talking on the telephone, it is often possible to make sense of what the other person is saying even when there is a noise on the line. At worst, even if you cannot understand it, you can know that what you hear does not make sense and so ask them to repeat what they said.

This is all very well for us intelligent human beings, but what about when you want to download something from the internet into your computer? Surely you cannot rely on there being no error introduced along the way. So the picture is



which we will replace by the more general picture in figure 1 (below) of a **message** being sent from a **source** to a **receiver**.

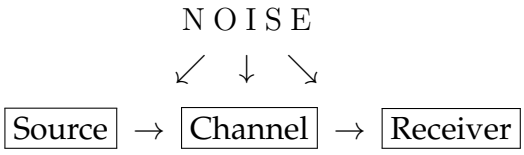


Figure 1

Also, since the example of a computer being the receiver is the most common, we will suppose that the message consists of a string of **bits** (binary digits) 0 and 1.

If we consider the case of human communication, we can see that figure 1 is too simple. In order to convey my thoughts to you, I am using a typewriter to convert (or **encode**) those thoughts into written English. When they reach you through the channel of this magazine, your eyes (and brain) convert (or **decode**) this back into thoughts. Thus figure 1 becomes

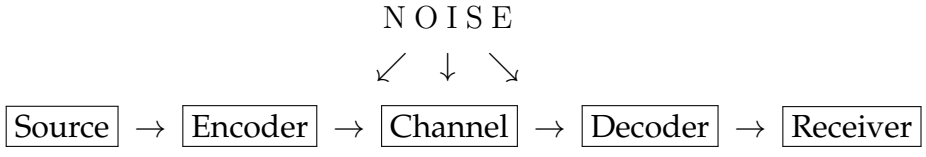


Figure 2

[1]Rod is a Lecturer in Pure Mathematics at UNSW and is currently editor of Parabola.

where the aim of the encoder is to use a **code** to change the original message into something in which errors can be **corrected** (or at least **detected**).

Some Error-Correcting Codes

Duplication consists of sending every bit twice. For example, the "message" 010 would be encoded as 00 11 00. Thus if the string 01 11 01 is received, it is obvious that there are errors in the first and last pairs and so this would be decoded as ?1? where ? means that an error was detected. This code cannot *correct* errors.

Triplication means that each bit is sent three times. For example, the message 010 would be encoded as 000 111 000. Now if the string 010 111 001 was received and it was assumed *that there was at most one error in each triple*, then the decoder would correct the second and last bits to 0. Note that if we allowed for the possibility of more than one error then the best we would be able to do is ?1? as in duplication.

Repetition a (large) number of *odd* times eventually gives us confidence of correcting errors. However, if we want to correct t errors, this requires an encoded message of length $2t + 1$, which is much larger than the original message.

In general, we can define the **information rate** of a code as

$$\text{information rate} = \frac{\text{length of original message}}{\text{length of encoded message}}$$

For example, the information rate of duplication is $\frac{1}{2}$ and of triplication is only $\frac{1}{3}$. Clearly the larger the information rate of a code is, the better it is.

Parity is a code with a large information rate. In this code, the message is broken up into **blocks** of a set length and a **checkbit** is placed at the end of each block. This consists of a 1 or 0 depending on whether the number of 1's in the block is odd or even. For example if the message had a block 1010100 (of length 7) then this block would be encoded as 10101001, making the total number of 1's even. If *one* of these bits was changed in the channel, this would be noticed by the fact that an odd number of 1's was received.

A parity code with blocks of length n has the (high) information rate of $\frac{n}{n+1}$, but it can only *detect* (and not *correct*) one error.

Rectangular codes are a variation on parity codes. In this case, the blocks are written in a column (each with a checkbit at the end). These blocks are themselves separated into blocks (of blocks) and a check bit is placed below each column of bits.

For example, if we use blocks of length 7 (with an eighth checkbit on the end of each block) and place a block of checkbits below each fourth block, then

$$\begin{array}{lcl} 1\,0\,0\,1\,1\,0\,1 & & 1\,0\,0\,1\,1\,0\,1\;\;0 \\ 1\,0\,0\,0\,0\,0\,1 & & 1\,0\,0\,0\,0\,0\,1\;\;0 \\ 1\,0\,1\,0\,1\,0\,0 & \text{would be encoded as} & 1\,0\,1\,0\,1\,0\,0\;\;1 \\ 1\,0\,0\,1\,0\,0\,0 & & 1\,0\,0\,1\,0\,0\,0\;\;0 \\ & & \\ & & 0\,0\,1\,0\,0\,0\,0\;\;1 \end{array}$$

Now suppose that we receive

$$\begin{array}{c} 1\,0\,1\,1\,1\,0\,1\,0 \\ 1\,0\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1 \end{array}$$

Since the first row has an odd number of 1's, there is an error in the first row. Also, since the third column has an odd number of 1's, there is an error in that column. Thus the error must have occurred in the third bit of the first row. Since there are only two choices for this bit (0 and 1) we can correct it to a 0.

One nice feature of a rectangular code is that it can correct several errors caused by **burst** noise, i.e. noise which goes for a short period of time, thus affecting several consecutive bits. For example, suppose that we know that burst noise has affected the above message and we receive

$$\begin{array}{c} 1\,0\,1\,1\,1\,1\,1\,1 \\ 1\,0\,0\,0\,0\,0\,1\,0 \\ 1\,0\,1\,0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0\,1 \end{array}$$

As before, the first row has an odd number of 1's, and so there are errors in the first block. Now, by looking at the columns, we can see that these errors occurred in the third, sixth and eighth bits (at least) and correct these. (In fact this is an example of burst noise making a string of bits all into 1's).

The information rate of a rectangular code with m blocks of length n is $\frac{mn}{(m+1)(n+1)}$, which you should be able to show (by Calculus) has a maximum when $m = n$. Thus **square** codes are the best rectangular codes.

Triangular codes are similar to rectangular codes, except that the blocks decrease in size and a checkbit counts whether the total number of 1's in both its row and column is odd or even. For example, the six bits 001101 can be rearranged

$$\begin{array}{l} 001 \\ 10 \\ 1 \end{array} \qquad \text{which is encoded as} \qquad \begin{array}{l} 001\ 1 \\ 10\ 0 \\ 1\ 1 \\ 0 \end{array}$$

where the last 0 in the second row counts the two 1's in the second row and third column.

The information rate of an $n \times n$ triangular code is

$$\frac{n(n-1)/2}{(n+1)n/2} = \frac{n-1}{n+1} = 1 - \frac{2}{n+1}.$$

Parity Checks

In each of the above examples, encoding consisted of expanding the original message to a string $x_1x_2\ldots x_n$ so that the number of 1's in a certain collection of the x_i was even. For example, in the parity code, we ensured that the total number of *all* 1's was even.

To make it easier to analyze codes mathematically, we will invent a new addition for bits where

$$0+0=0, \quad 0+1=1+0=1 \text{ (as usual)} \quad \text{and} \quad 1+1=0.$$

Thus, since each bit is either a 0 or 1, saying that the total number of 1's is even is equivalent to saying

$$x_1 + x_2 + \cdots + x_n = 0.$$

Similarly, in the duplication code (where we send $x_1x_2\ldots x_{2m}$), we ensured that

$$x_1 + x_2 = x_3 + x_4 = \cdots = x_{2m-1} + x_{2m} = 0$$

and in the triangulation code, where we send

$$\begin{array}{l} x_{11}\ x_{12}\ x_{13}\ x_{14} \\ x_{21}\ x_{22}\ x_{23} \\ x_{31}\ x_{32} \\ x_{41}, \end{array}$$

we ensure that

$$x_{11}+x_{12}+x_{13}+x_{14} = x_{21}+x_{22}+x_{13}+x_{23} = x_{31}+x_{12}+x_{22}+x_{32} = x_{11}+x_{21}+x_{31}+x_{41} = 0.$$

So a code can be thought of as a set of **parity checks**

$$\sum_{j=1}^{n} a_{ij}x_j = 0 \quad (i = 1, 2, \ldots, m) \quad \text{where} \quad a_{ij} = 0 \text{ or } 1.$$

Efficient Codes

Our aim now is to find error-correcting codes which will correct one error and whose information rates are as large as possible, i.e. ones which replace a message $x_1x_2 \ldots x_k$ by $x_1x_2 \ldots x_kx_{k+1} \ldots x_n$ such that k/n is as large as possible. For convenience, we will write $m = n - k$ for the number of checkbits.

If you consider the first example in the case of the rectangular codes, you will see that the error was discovered (and corrected) by looking at the values of all the checkbits. On the other hand, it was impossible to correct an error in the case of the parity code because there were not enough checkbits (there was only one!). In general, an error can only be corrected if its position is determined by the sequence $x_{k+1}x_{k+2} \ldots x_{k+m}$ of m checkbits. Since each checkbit is either 0 or 1, there are 2^m sequences of checkbits possible and so, since there are n possible places for an error to occur,

$$\text{number of possible checkbit sequences } = 2^m \geq n + 1 \tag{1}$$

(since the checkbit sequence $00 \ldots 0$ indicates NO errors).

Thus the information rate of a *most efficient* code (if one exists) is

$$\frac{k}{n} = \frac{n-m}{n} = 1 - \frac{m}{2^m - 1}.$$

Examples:

1. Suppose we have a message of length $k = 3$ to encode.
   If we use m checkbits, then the code will have length $n = m + 3$ where

   $$2^m \geq n + 1 = m + 4.$$

   After trial and error, the smallest such code has $m = 3$ and $n = 6$.

2. This time we will try to get the most out of the number of checkbits and so we construct the largest code with $m = 3$. Equation (1) becomes

   $$2^m = 2^3 = 8 \geq n + 1.$$

   So $n = 7$ and the encoded message is $x_1x_2x_3x_4x_5x_6x_7$ with 3 checkbits and $k = 7 - 3 = 4$ information bits.

Now it turns out in practice that it is better to place the 3 checkbits in the encoded message at positions 1, 2 and 4 (powers of 2). Also if we write

$$\begin{aligned} x_1 &= x_3 + x_5 + x_7 \\ x_2 &= x_3 \quad\quad + x_6 + x_7 \\ x_4 &= \quad\quad x_5 + x_6 + x_7 \end{aligned} \tag{2}$$

(and remembering the new definition for addition) then you can check that each of the 8 possible sequences $x_1x_2x_4$ occurs at least once as we run through all possible values of x_3, x_5, x_6, x_7. For example, to encode the message 0101,

$$\begin{aligned} x_3 = x_6 = 0, \quad\quad & x_5 = x_7 = 1, \\ x_1 &= 0+1+1=0 \\ x_2 &= 0+0+1=1 \\ x_4 &= 1+0+1=0 \end{aligned}$$

and so the encoded message is $\underline{0}\,\underline{1}\,0\,\underline{0}\,1\,0\,1$.

To correct an error, note that equations (2) can be written (using the new definition for addition) as the **parity checks**

$$\begin{array}{cccccccc} x_1 & & +x_3 & & +x_5 & & +x_7 & = 0 \\ & x_2 & +x_3 & & & +x_6 & +x_7 & = 0 \\ & & & x_4 & +x_5 & +x_6 & +x_7 & = 0 \end{array}$$

Now suppose we receive 0 1 0 0 1 1 1.

Substituting these into the left hand side of the above parity checks yields

$$\begin{array}{cccccccc} x_1 & & +x_3 & & +x_5 & & +x_7 & = 0 \\ & x_2 & +x_3 & & & +x_6 & +x_7 & = 1 \\ & & & x_4 & +x_5 & +x_6 & +x_7 & = 1. \end{array}$$

Since these are not all 0, there is an error. In fact, these values (reading upwards) are 110 which is binary for 6, showing that the error is in 6th place. Thus the encoded message sent was $0\,1\,0\,0\,1\,\underline{0}\,1$, i.e. the original message was 0 1 0 1.

This code is called the **Hamming (7,4) code** because it has (encoded) length 7 with an original message of length 4.

Using this idea, Hamming (in 1950) constructed codes of length $2^m - 1$ for every value

of m. Their information rates R are given in the following table:

m	$n = 2m - 1$	$k = n - m$	$R = \frac{k}{n}$
3	7	4	0.57
4	15	11	0.73
5	31	26	0.84
6	63	57	0.90
7	127	120	0.94
8	255	247	0.97
9	511	502	0.98
10	1023	1013	0.99

Correcting More Errors

The Hamming codes are examples of the most efficient codes which will correct one error. If we want to correct t errors, then equation (1) becomes

$$2^m \geq 1 + n + \binom{n}{2} + \cdots + \binom{n}{t}$$

where $\binom{n}{r}$ means the number of ways of choosing r objects from n objects. Very few codes are known where $2^m = 1 + n + \binom{n}{2} + \cdots + \binom{n}{t}$ and $t > 1$. However one such code (discovered by Reed and Muller) was used when Mariner 9 transmitted colour pictures from Mars in 1979 and an extension of another one (discovered by Golay) was used when Voyager transmitted similar pictures from Jupiter and Saturn in the following two years. The details of these codes are:

	m	n	k	R	t
Reed-Muller	26	32	6	$\frac{3}{16}$	7
Golay	11	23	12	$\frac{12}{23}$	3
Extended Golay	11	24	12	$\frac{1}{2}$	3

It is to be hoped that in the near future more codes like these will be found, making it possible to transmit similar information efficiently.

Further Reading

R. W. Hamming, *Coding and information theory*, Prentice-Hall (1986).
R. Hill, *A first course in coding theory*, Clarendon (1986).
V. Pless, *Introduction to the theory of error correcting codes*, Wiley (1982/89).
O. Pretzel, *Error-Correcting Codes and Finite Fields*, Clarendon (1992).
S. Roman, *Coding and information theory*, Springer (1992).