The Bell System Technical Journal

Vol. XXIX *April, 1950* *No. 2*

Error Detecting and Error Correcting Codes

By R. W. HAMMING

1. Introduction

THE author was led to the study given in this paper from a consideration of large scale computing machines in which a large number of operations must be performed without a single error in the end result. This problem of "doing things right" on a large scale is not essentially new; in a telephone central office, for example, a very large number of operations are performed while the errors leading to wrong numbers are kept well under control, though they have not been completely eliminated. This has been achieved, in part, through the use of self-checking circuits. The occasional failure that escapes routine checking is still detected by the customer and will, if it persists, result in customer complaint, while if it is transient it will produce only occasional wrong numbers. At the same time the rest of the central office functions satisfactorily. In a digital computer, on the other hand, a single failure usually means the complete failure, in the sense that if it is detected no more computing can be done until the failure is located and corrected, while if it escapes detection then it invalidates all subsequent operations of the machine. Put in other words, in a telephone central office there are a number of parallel paths which are more or less independent of each other; in a digital machine there is usually a single long path which passes through the same piece of equipment many, many times before the answer is obtained.

In transmitting information from one place to another digital machines use codes which are simply sets of symbols to which meanings or values are attached. Examples of codes which were designed to detect isolated errors are numerous; among them are the highly developed 2 out of 5 codes used extensively in common control switching systems and in the Bell Relay

"Error Detecting and Error Correcting Codes" by R.W. Hamming
from *Bell System Technical Journal*, Volume 29, Number 2, April
1950, pages 147-160.

Computers,[1] the 3 out of 7 code used for radio telegraphy,[2] and the word count sent at the end of telegrams.

In some situations self checking is not enough. For example, in the Model 5 Relay Computers built by Bell Telephone Laboratories for the Aberdeen Proving Grounds,[1] observations in the early period indicated about two or three relay failures per day in the 8900 relays of the two computers, representing about one failure per two to three million relay operations. The self-checking feature meant that these failures did not introduce undetected errors. Since the machines were run on an unattended basis over nights and week-ends, however, the errors meant that frequently the computations came to a halt although often the machines took up new problems. The present trend is toward electronic speeds in digital computers where the basic elements are somewhat more reliable per operation than relays. However, the incidence of isolated failures, even when detected, may seriously interfere with the normal use of such machines. Thus it appears desirable to examine the next step beyond error detection, namely error correction.

We shall assume that the transmitting equipment handles information in the binary form of a sequence of 0's and 1's. This assumption is made both for mathematical convenience and because the binary system is the natural form for representing the open and closed relays, flip-flop circuits, dots and dashes, and perforated tapes that are used in many forms of communication. Thus each code symbol will be represented by a sequence of 0's and 1's.

The codes used in this paper are called *systematic* codes. Systematic codes may be defined[3] as codes in which each code symbol has exactly n binary digits, where m digits are associated with the information while the other $k = n - m$ digits are used for error detection and correction. This produces a *redundancy* R defined as the ratio of the number of binary digits used to the minimum number necessary to convey the same information, that is,

$$R = n/m.$$

This serves to measure the efficiency of the code as far as the transmission of information is concerned, and is the only aspect of the problem discussed in any detail here. The redundancy may be said to lower the effective channel capacity for sending information.

The need for error correction having assumed importance only recently, very little is known about the economics of the matter. It is clear that in

[1] Franz Alt, "A Bell Telephone Laboratories' Computing Machine"—I, II. Mathematical Tables and Other Aids to Computation, Vol. 3, pp. 1–13 and 60–84, Jan. and Apr. 1948.

[2] S. Sparks, and R. G. Kreer, "Tape Relay System for Radio Telegraph Operation," *R.C.A. Review*, Vol. 8, pp. 393–426, (especially p. 417), 1947.

[3] In Section 7 this is shown to be equivalent to a much weaker appearing definition.

using such codes there will be extra equipment for encoding and correcting errors as well as the lowered effective channel capacity referred to above. Because of these considerations applications of these codes may be expected to occur first only under extreme conditions. Some typical situations seem to be:

a. unattended operation over long periods of time with the minimum of standby equipment.

b. extremely large and tightly interrelated systems where a single failure incapacitates the entire installation.

c. signaling in the presence of noise where it is either impossible or uneconomical to reduce the effect of the noise on the signal.

These situations are occurring more and more often. The first two are particularly true of large scale digital computing machines, while the third occurs, among other places, in "jamming" situations.

The principles for designing error detecting and correcting codes in the cases most likely to be applied first are given in this paper. Circuits for implementing these principles may be designed by the application of well-known techniques, but the problem is not discussed here. Part I of the paper shows how to construct special minimum redundancy codes in the following cases:

a. single error detecting codes

b. single error correcting codes

c. single error correcting plus double error detecting codes.

Part II discusses the general theory of such codes and proves that under the assumptions made the codes of Part I are the "best" possible.

PART I

SPECIAL CODES

2. Single Error Detecting Codes

We may construct a single error detecting code having n binary digits in the following manner: In the first $n-1$ positions we put $n-1$ digits of information. In the n-th position we place either 0 or 1, so that the entire n positions have an even number of 1's. This is clearly a single error detecting code since any single error in transmission would leave an odd number of 1's in a code symbol.

The redundancy of these codes is, since $m = n - 1$,

$$R = \frac{n}{n-1} = 1 + \frac{1}{n-1}.$$

It might appear that to gain a low redundancy we should let n become very large. However, by increasing n, the probability of at least one error in a

symbol increases; and the risk of a double error, which would pass undetected, also increases. For example, if $p \ll 1$ is the probability of any error, then for n so large as $1/p$, the probability of a correct symbol is approximately $1/e = 0.3679\ldots$, while a double error has probability $1/2e = 0.1839\ldots$.

The type of check used above to determine whether or not the symbol has any single error will be used throughout the paper and will be called a *parity check*. The above was an *even* parity check; had we used an odd number of 1's to determine the setting of the check position it would have been an odd parity check. Furthermore, a parity check need not always involve all the positions of the symbol but may be a check over selected positions only.

3. Single Error Correcting Codes

To construct a single error correcting code we first assign m of the n available positions as information positions. We shall regard the m as fixed, but the specific positions are left to a later determination. We next assign the k remaining positions as check positions. The values in these k positions are to be determined in the encoding process by even parity checks over selected information positions.

Let us imagine for the moment that we have received a code symbol, with or without an error. Let us apply the k parity checks, in order, and for each time the parity check assigns the value observed in its check position we write a 0, while for each time the assigned and observed values disagree we write a 1. When written from right to left in a line this sequence of k 0's and 1's (to be distinguished from the values assigned by the parity checks) may be regarded as a binary number and will be called the *checking number*. We shall require that this checking number give the position of any single error, with the zero value meaning no error in the symbol. Thus the check number must describe $m + k + 1$ different things, so that

$$2^k \geq m + k + 1$$

is a condition on k. Writing $n = m + k$ we find

$$2^m \leq \frac{2^n}{n+1}.$$

Using this inequality we may calculate Table I, which gives the maximum m for a given n, or, what is the same thing, the minimum n for a given m.

We now determine the positions over which each of the various parity checks is to be applied. The checking number is obtained digit by digit, from right to left, by applying the parity checks in order and writing down the corresponding 0 or 1 as the case may be. Since the checking number is

TABLE I

n	m	Corresponding k
1	0	1
2	0	2
3	1	2
4	1	3
5	2	3
6	3	3
7	4	3
8	4	4
9	5	4
10	6	4
11	7	4
12	8	4
13	9	4
14	10	4
15	11	4
16	11	5
	Etc.	

to give the position of any error in a code symbol, any position which has a 1 on the right of its binary representation must cause the first check to fail. Examining the binary form of the various integers we find

$$\begin{aligned} 1 &= 1 \\ 3 &= 11 \\ 5 &= 101 \\ 7 &= 111 \\ 9 &= 1001 \end{aligned}$$

Etc.

have a 1 on the extreme right. Thus the first parity check must use positions

$$1, 3, 5, 7, 9, \cdots .$$

In an exactly similar fashion we find that the second parity check must use those positions which have 1's for the second digit from the right of their binary representation,

$$\begin{aligned} 2 &= 10 \\ 3 &= 11 \\ 6 &= 110 \\ 7 &= 111 \\ 10 &= 1010 \\ 11 &= 1011 \end{aligned}$$

Etc.,

the third parity check

$$
\begin{aligned}
4 &= 100 \\
5 &= 101 \\
6 &= 110 \\
7 &= 111 \\
12 &= 1100 \\
13 &= 1101 \\
14 &= 1110 \\
15 &= 1111 \\
20 &= 10100 \\
&\text{Etc.}
\end{aligned}
$$

It remains to decide for each parity check which positions are to contain information and which the check. The choice of the positions 1, 2, 4, 8, $\cdots$ for check positions, as given in the following table, has the advantage of making the setting of the check positions independent of each other. All other positions are information positions. Thus we obtain Table II.

TABLE II

Check Number	Check Positions	Positions Checked
1	1	1, 3, 5, 7, 9, 11, 13, 15, 17, $\cdots$
2	2	2, 3, 6, 7, 10, 11, 14, 15, 18, $\cdots$
3	4	4, 5, 6, 7, 12, 13, 14, 15, 20, $\cdots$
4	8	8, 9, 10, 11, 12, 13, 14, 15, 24, $\cdots$
.	.	.
.	.	.
.	.	.

As an illustration of the above theory we apply it to the case of a seven-position code. From Table I we find for $n = 7$, $m = 4$ and $k = 3$. From Table II we find that the first parity check involves positions 1, 3, 5, 7 and is used to determine the value in the first position; the second parity check, positions 2, 3, 6, 7, and determines the value in the second position; and the third parity check, positions 4, 5, 6, 7, and determines the value in position four. This leaves positions 3, 5, 6, 7 as information positions. The results of writing down all possible binary numbers using positions 3, 5, 6, 7, and then calculating the values in the check positions 1, 2, 4, are shown in Table III.

Thus a seven-position single error correcting code admits of 16 code symbols. There are, of course, $2^7 - 16 = 112$ meaningless symbols. In some applications it may be desirable to drop the first symbol from the code to avoid the all zero combination as either a code symbol or a code symbol plus a single error, since this might be confused with no message. This would still leave 15 useful code symbols.

TABLE III

Position							Decimal Value of Symbol
1	2	3	4	5	6	7	
0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1
0	1	0	1	0	1	0	2
1	0	0	0	0	1	1	3
1	0	0	1	1	0	0	4
0	1	0	0	1	0	1	5
1	1	0	0	1	1	0	6
0	0	0	1	1	1	1	7
1	1	1	0	0	0	0	8
0	0	1	1	0	0	1	9
1	0	1	1	0	1	0	10
0	1	1	0	0	1	1	11
0	1	1	1	1	0	0	12
1	0	1	0	1	0	1	13
0	0	1	0	1	1	0	14
1	1	1	1	1	1	1	15

As an illustration of how this code "works" let us take the symbol 0 1 1 1 1 0 0 corresponding to the decimal value 12 and change the 1 in the fifth position to a 0. We now examine the new symbol

0 1 1 1 0 0 0

by the methods of this section to see how the error is located. From Table II the first parity check is over positions 1, 3, 5, 7 and predicts a 1 for the first position while we find a 0 there; hence we write a

1 .

The second parity check is over positions 2, 3, 6, 7, and predicts the second position correctly; hence we write a 0 to the left of the 1, obtaining

0 1 .

The third parity check is over positions 4, 5, 6, 7 and predicts wrongly; hence we write a 1 to the left of the 0 1, obtaining

1 0 1 .

This sequence of 0's and 1's regarded as a binary number is the number 5; hence the error is in the fifth position. The correct symbol is therefore obtained by changing the 0 in the fifth position to a 1.

4. SINGLE ERROR CORRECTING PLUS DOUBLE ERROR DETECTING CODES

To construct a single error correcting plus double error detecting code we begin with a single error correcting code. To this code we add one more posi-

tion for checking all the previous positions, using an even parity check. To see the operation of this code we have to examine a number of cases:

1. No errors. All parity checks, including the last, are satisfied.
2. Single error. The last parity check fails in all such situations whether the error be in the information, the original check positions, or the last check position. The original checking number gives the position of the error, where now the zero value means the last check position.
3. Two errors. In all such situations the last parity check is satisfied, and the checking number indicates some kind of error.

As an illustration let us construct an eight-position code from the previous seven-position code. To do this we add an eighth position which is chosen so that there are an even number of 1's in the eight positions. Thus we add an eighth column to Table III which has:

TABLE IV

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

PART II

GENERAL THEORY

5. A GEOMETRICAL MODEL

When examining various problems connected with error detecting and correcting codes it is often convenient to introduce a geometric model. The model used here consists in identifying the various sequences of 0's and 1's which are the symbols of a code with vertices of a unit n-dimensional cube. The code points, labelled x, y, z, $\cdots$, form a subset of the set of all vertices of the cube.

Into this space of 2^n points we introduce a *distance*, or, as it is usually called, a *metric*, $D(x, y)$. The definition of the metric is based on the observation that a single error in a code point changes one coordinate, two errors, two coordinates, and in general d errors produce a difference in d coordinates.

Thus we define the distance $D(x, y)$ between two points x and y as the number of coordinates for which x and y are different. This is the same as the least number of edges which must be traversed in going from x to y. This distance function satisfies the usual three conditions for a metric, namely,

$$D(x, y) = 0 \quad \text{if and only if } x = y$$

$$D(x, y) = D(y, x) > 0 \quad \text{if } x \neq y$$

$$D(z, y) + D(y, z) \geq D(x, z) \text{ (triangle inequality).}$$

As an example we note that each of the following code points in the three-dimensional cube is two units away from the others,

$$0\ 0\ 1$$
$$0\ 1\ 0$$
$$1\ 0\ 0$$
$$1\ 1\ 1\ .$$

To continue the geometric language, a sphere of radius r about a point x is defined as all points which are at a distance r from the point x. Thus, in the above example, the first three code points are on a sphere of radius 2 about the point (1, 1, 1). In fact, in this example any one code point may be chosen as the center and the other three will lie on the surface of a sphere of radius 2.

If all the code points are at a distance of at least 2 from each other, then it follows that any single error will carry a code point over to a point that is *not* a code point, and hence is a meaningless symbol. This in turn means that any single error is detectable. If the minimum distance between code points is at least three units then any single error will leave the point nearer to the correct code point than to any other code point, and this means that any single error will be correctable. This type of information is summarized in the following table:

TABLE V

Minimum Distance	Meaning
1	uniqueness
2	single error detection
3	single error correction
4	single error correction plus double error detection
5	double error correction
	Etc.

Conversely, it is evident that, if we are to effect the detection and correction listed, then all the distances between code points must equal or exceed the minimum distance listed. Thus the problem of finding suitable codes is

the same as that of finding subsets of points in the space which maintain at least the minimum distance condition. The special codes in sections 2, 3, and 4 were merely descriptions of how to choose a particular subset of points for minimum distances 2, 3, and 4 respectively.

It should perhaps be noted that, at a given minimum distance, some of the correctability may be exchanged for more detectability. For example, a subset with minimum distance 5 may be used for:

a. double error correction, (with, of course, double error detection).

b. single error correction plus triple error detection.

c. quadruple error detection.

Returning for the moment to the particular codes constructed in Part I we note that any interchanges of positions in a code do not change the code in any essential way. Neither does interchanging the 0's and 1's in any position, a process usually called complementing. This idea is made more precise in the following definition:

Definition. Two codes are said to be *equivalent* to each other if, by a finite number of the following operations, one can be transformed into the other:

1. The interchange of any two positions in the code symbols.

2. The complementing of the values in any position in the code symbols.

This is a formal equivalence relation ($\sim$) since $A \sim A$; $A \sim B$ implies $B \sim A$; and $A \sim B$, $B \sim C$ implies $A \sim C$. Thus we can reduce the study of a class of codes to the study of typical members of each equivalence class.

In terms of the geometric model, equivalence transformations amount to rotations and reflections of the unit cube.

6. Single Error Detecting Codes

The problem studied in this section is that of packing the maximum number of points in a unit n-dimensional cube such that no two points are closer than 2 units from each other. We shall show that, as in section 2, 2^{n-1} points can be so packed, and, further, that any such optimal packing is equivalent to that used in section 2.

To prove these statements we first observe that the vertices of the n-dimensional cube are composed of those of two $(n-1)$-dimensional cubes. Let A be the maximum number of points packed in the original cube. Then one of the two $(n-1)$-dimensional cubes has at least $A/2$ points. This cube being again decomposed into two lower dimensional cubes, we find that one of them has at least $A/2^2$ points. Continuing in this way we come to a two-dimensional cube having $A/2^{n-2}$ points. We now observe that a square can have at most two points separated by at least two units; hence the original n-dimensional cube had at most 2^{n-1} points not less than two units apart.

To prove the equivalence of any two optimal packings we note that, if the packing is optimal, then each of the two sub-cubes has half the points. Calling this the first coordinate we see that half the points have a 0 and half have a 1. The next subdivision will again divide these into two equal groups having 0's and 1's respectively. After $(n - 1)$ such stages we have, upon reordering the assigned values if there be any, exactly the first $n - 1$ positions of the code devised in section 2. To each sequence of the first $n - 1$ coordinates there exist $n - 1$ other sequences which differ from it by one coordinate. Once we fix the n-th coordinate of some one point, say the origin which has all 0's, then to maintain the known minimum distance of two units between code points the n-th coordinate is uniquely determined for all other code points. Thus the last coordinate is determined within a complementation so that any optimal code is equivalent to that given in section 2.

It is interesting to note that in these two proofs we have used only the assumption that the code symbols are all of length n.

7. Single Error Correcting Codes

It has probably been noted by the reader that, in the particular codes of Part I, a distinction was made between information and check positions, while, in the geometric model, there is no real distinction between the various coordinates. To bring the two treatments more in line with each other we redefine a *systematic* code as a code whose symbol lengths are all equal and

1. The positions checked are independent of the information contained in the symbol.
2. The checks are independent of each other.
3. We use parity checks.

This is equivalent to the earlier definition. To show this we form a matrix whose i-th row has 1's in the positions of the i-th parity check and 0's elsewhere. By assumption 1 the matrix is fixed and does not change from code symbol to code symbol. From 2 the rank of the matrix is k. This in turn means that the system can be solved for k of the positions expressed in terms of the other $n - k$ positions. Assumption 3 indicates that in this solving we use the arithmetic in which $1 + 1 = 0$.

There exist non-systematic codes, but so far none have been found which for a given n and minimum distance d have more code symbols than a systematic code. Section 9 gives an example of a non-systematic code.

Turning to the main problem of this section we find from Table V that a single error correcting code has code points at least three units from each other. Thus each point may be surrounded by a sphere of radius 1 with no two spheres having a point in common. Each sphere has a center point and

n points on its surface, a total of $n + 1$ points. Thus the space of 2^n points can have at most:

$$\frac{2^n}{n+1}$$

spheres. This is exactly the bound we found before in section 3.

While we have shown that the special single error correcting code constructed in section 3 is of minimum redundancy, we cannot show that all optimal codes are equivalent, since the following trivial example shows that this is not so. For $n = 4$ we find from Table I that $m = 1$ and $k = 3$. Thus there are at most two code symbols in a four-position code. The following two optimal codes are clearly not equivalent:

$$\begin{matrix} 0\ 0\ 0\ 0 \\ 1\ 1\ 1\ 1 \end{matrix} \quad \text{and} \quad \begin{matrix} 0\ 0\ 0\ 0 \\ 0\ 1\ 1\ 1 \end{matrix}$$

8. Single Error Correcting Plus Double Error Detecting Codes

In this section we shall prove that the codes constructed in section 4 are of minimum redundancy. We have already shown in section 4 how, for a minimum redundancy code of $n - 1$ dimensions with a minimum distance of 3, we can construct an n dimensional code having the same number of code symbols but with a minimum distance of 4. If this were not of minimum redundancy there would exist a code having more code symbols but with the same n and the same minimum distance 4 between them. Taking this code we remove the last coordinate. This reduces the dimension from n to $n - 1$ and the minimum distance between code symbols by, at most, one unit, while leaving the number of code symbols the same. This contradicts the assumption that the code we began our construction with was of minimum reduncancy. Thus the codes of section 4 are of minimum redundancy.

This is a special case of the following general theorem: To any minimum redundancy code of N points in $n - 1$ dimensions and having a minimum distance of $2k - 1$ there corresponds a minimum redundancy code of N points in n dimensions having a minimum distance of $2k$, and conversely. To construct the n dimensional code from the $n - 1$ dimensional code we simply add a single n-th coordinate which is fixed by an even parity check over the n positions. This also increases the minimum distance by 1 for the following reason: Any two points which, in the $n - 1$ dimensional code, were at a distance $2k - 1$ from each other had an odd number of differences between their coordinates. Thus the parity check was set oppositely for the two points, increasing the distance between them to $2k$. The additional coordinate could not decrease any distances, so that all points in the code are now at a minimum distance of $2k$. To go in the reverse direction we simply

drop one coordinate from the n dimensional code. This reduces the minimum distance of $2k$ to $2k - 1$ while leaving N the same. It is clear that if one code is of minimum redundancy then the other is, too.

9. Miscellaneous Observations

For the next case, minimum distance of five units, one can surround each code point by a sphere of radius 2. Each sphere will contain

$$1 + C(n, 1) + C(n, 2)$$

points, where $C(n, k)$ is the binomial coefficient, so that an upper bound on the number of code points in a systematic code is

$$\frac{2^n}{1 + C(n, 1) + C(n, 2)} = \frac{2^{n+1}}{n^2 + n + 2} \geq 2^m.$$

This bound is too high. For example, in the case of $n = 7$, we find that $m = 2$ so that there should be a code with four code points. The maximum possible, as can be easily found by trial and error, is two.

In a similar fashion a bound on the number of code points may be found whenever the minimum distance between code points is an odd number. A bound on the even cases can then be found by use of the general theorem of the preceding section. These bounds are, in general, too high, as the above example shows.

If we write the bound on the number of code points in a unit cube of dimension n and with minimum distance d between them as $B(n, d)$, then the information of this type in the present paper may be summarized as follows:

$$B(n, 1) = 2^n$$

$$B(n, 2) = 2^{n-1}$$

$$B(n, 3) = 2^m \leq \frac{2^n}{n + 1}$$

$$B(n, 4) = 2^m \leq \frac{2^{n-1}}{n}$$

$$B(n - 1, 2k - 1) = B(n, 2k)$$

$$B(n, 2k - 1) = 2^m \leq \frac{2^n}{1 + C(n, 1) + \cdots + C(n, k - 1)}.$$

While these bounds have been attained for certain cases, no general methods have yet been found for contructing optimal codes when the minimum distance between code points exceeds four units, nor is it known whether the bound is or is not attainable by systematic codes.

We have dealt mainly with systematic codes. The existence of non-systematic codes is proved by the following example of a single error correcting code with $n = 6$.

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

The all 0 symbol indicates that any parity check must be an even one. The all 1 symbol indicates that each parity check must involve an even number of positions. A direct comparison indicates that since no two columns are the same the even parity checks must involve four or six positions. An examination of the second symbol, which has three 1's in it, indicates that no six-position parity check can exist. Trying now the four-position parity checks we find that

$$
\begin{array}{c}
1\,2\quad\;\;5\,6 \\
\;\;2\,3\,4\,5
\end{array}
$$

are two independent parity checks and that no third one is independent of these two. Two parity checks can at most locate four positions, and, since there are six positions in the code, these two parity checks are not enough to locate any single error. The code is, however, single error correcting since it satisfies the minimum distance condition of three units.

The only previous work in the field of error correction that has appeared in print, so far as the author is aware, is that of M. J. E. Golay.[4]

[4] M. J. E. Golay, Correspondence, Notes on Digital Coding, *Proceedings of the I.R.E.*, Vol. 37, p. 657, June 1949.