Error-Correcting Codes: An Application of Geometry

Information is stored and exchanged in the form of streams of characters from some alphabet. An *alphabet* is a finite set of symbols, such as the *lower-case Roman alphabet* {a,b,c,…,z}. Larger digits may be formed by including the upper-case Roman letters, punctuation marks, the digits 0 through 9, and possibly other symbols such as '$', '%', etc. In the other extreme, we may choose the simplest possible alphabet, the *binary alphabet* {0,1}. We prefer the binary alphabet for several reasons:

1. its ease of implementation as the on/off state of an electric circuit, the North/South polarization of positions on a magnetic tape or disk, etc.;
2. the fact that letters of any alphabet can be easily represented as strings of 0's and 1's; and
3. access to powerful algebraic tools for encoding and decoding.

Strings of letters from our chosen alphabet are called *words* or, when the binary alphabet is in use, *bitstrings*.

All such information is subject to corruption due to imperfect storage media (dust, scratches or manufacturing defects on optical CD's; demagnetizing influences on magnetic tape or disks) or noisy transmission channels (electromagnetic static in telephone lines or atmosphere). A bit error occurs when a 0 is changed to a 1, or a 1 to a 0, due to such influences. The goal of error-correcting codes is to protect information against such errors. Thanks to error-correcting codes, we can expect that a copy of a copy of a copy of a copy of an audio CD will sound exactly like the original when played back (at least 99.999% of the time); or that a large binary file downloaded off the internet will be a perfect copy of the original (again, 99.999% of the time).

What makes such recovery of the original binary file possible? Roughly, an error-correcting code adds redundancy to a message. Without this added redundancy, no error-correction would be possible. The trick, however, is to add as little redundancy as necessary, since longer messages are more costly to transmit. Finding the optimal balance between achieving a high error-correction and keeping the size of the encoded message as small as possible (i.e. achieving a high

information rate, to be defined later) is one of the prime concerns of coding theory. (Note that our codes are used to protect information from alteration due to faulty transmission or storage/retrieval; not to secure information from unauthorized access.) These concepts are best explained through examples.

Suppose we wish to send a bitstring of length 4, i.e. one of the sixteen possible message words 0000, 0001, 0010, 0011, …, 1111. We refer to these sixteen strings as *message words*, and the (If a message were more than 4 bits in length, it could be divided into blocks of 4 bits each, which could then be encoded and sent individually.) Note that these bitstrings are the binary representations of the integers 0, 1, 2, …, 15; they also correspond to the hexadecimal digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Scheme 1: "As is"

One possibility is that we send each bitstring "as is": for example, the message 1011 would be sent as 1011. This scheme does not allow for any error correction, and so it is only practical for a noiseless channel.

Scheme 2: Parity Check Bit

As a second example, consider appending a parity check bit to the end of each message word. This last bit is chosen so that each codeword has an even number of 1's. For example, the message 1011 would be encoded as 10111; the message 0110 would be encoded as 01100; see Table A. This parity check bit allows for the detection of a single bit error during transmission; however this error cannot be corrected.
For example, if the word 10111 is received, this would be accepted as a valid codeword and would be decoded as 1011. If the word 11010 is received, no decoding is possible and this word would be rejected. In practice the receiver would ask the sender to resend the message if possible.

Scheme 3: A 3-Repetition Code

In order to allow for correction of up to one bit error, we consider the possibility of sending each bit three times. Under this scheme, the message word 0100 would be encoded as the codeword 000111000000 of length 12. Suppose this word is transmitted and that, due to a single bit error during transmission, the word 000111001000 is received. Each triple of bits is decoded according to a 'majority rules' principle, thus 000 yields 0; 111 yields 1; 001 yields 0; and 000 yields 0, so

the original message word 0100 is recovered despite the bit error introduced during transmission.

Table A: Four Schemes for Encoding of 4-bit Message Words

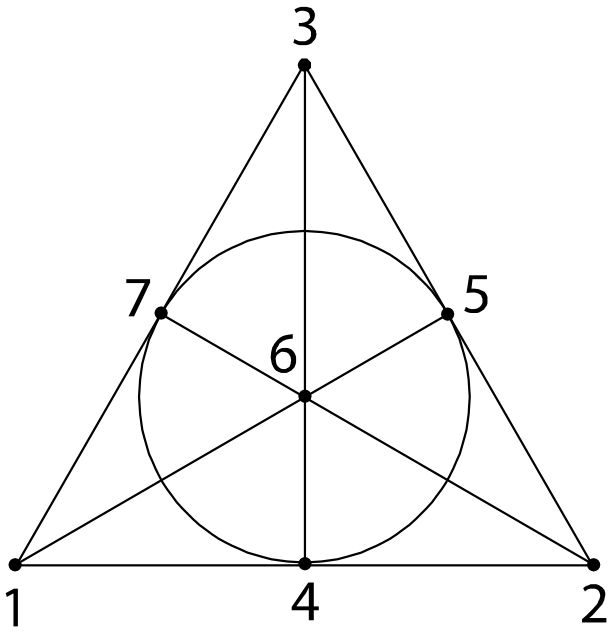
Msg. No.	Message Text	Scheme 1 ("As Is") Codeword	Scheme 2 (Parity Check) Codeword	Scheme 3 (3-Repetition) Codeword	Scheme 4 (Hamming) Codeword
0	**0000**	0000	00000	000000000000	**0000000**
1	**0001**	0001	00011	000000000111	**0001101**
2	**0010**	0010	00101	000000111000	**0010111**
3	**0011**	0011	00110	000000111111	**0011010**
4	**0100**	0100	01001	000111000000	**0100011**
5	**0101**	0101	01010	000111000111	**0101110**
6	**0110**	0110	01100	000111111000	**0110100**
7	**0111**	0111	01111	000111111111	**0111001**
8	**1000**	1000	10001	111000000000	**1000110**
9	**1001**	1001	10010	111000000111	**1001011**
10	**1010**	1010	10100	111000111000	**1010001**
11	**1011**	1011	10111	111000111111	**1011100**
12	**1100**	1100	11000	111111000000	**1100101**
13	**1101**	1101	11011	111111000111	**1101000**
14	**1110**	1110	11101	111111111000	**1110010**
15	**1111**	1111	11110	111111111111	**1111111**

Scheme 4: The Hamming Code

Finally we consider a scheme that corrects errors, yet is more efficient than the repetition code. In this scheme every 4-bit message word is encoded as a 7-bit codeword according to Table A. Note that we have simply appended three bits to every message word; the rule for choosing these bits is more complicated than the

rule used in Scheme 2 but will be revealed later. Under this scheme, the codeword for message 1011 is 1011100. Suppose this word suffers from a single bit error transmission, and is received as 1011110, say. This word will be safely recognized as 1011100 (and decoded as 1011) since all other Hamming codewords differ from 1011110 in at least two positions. The property of the Hamming code which guarantees unique decodability with at most one bit error, is that any two Hamming codewords differ from in each other in at least three positions. This code was discovered by Richard Hamming, a pioneer in the theory of error-correcting codes who was primarily interested in their application to early electronic computers for achieving fault-tolerant computation.

Richard Hamming
1915–1998



The Projective Plane of Order Two:

7 points, 7 lines

3 points on each line

3 lines through each point

This code, like the 3-repetition code, allows for single bit errors to be corrected; yet it is superior in that it requires the transmission of only 7 bits (rather than 12 bits) for every 4 bits of actual information.

The Hamming code is constructed from the projective plane of order two, as follows. The seven lines of the plane give rise to fourteen of the Hamming codewords; for example the line {1,2,4} yields the codeword 1101000 with 1's in the 1st, 2nd and 4th positions; also the codeword 0010111, with 0's in the 1st, 2nd and 4th positions. The remaining two codewords are simply 0000000 and 1111111. This makes up all sixteen codewords.

It is apparent that the construction of efficient codes is a geometric problem: one seeks a large number of points ('codewords') in a higher-dimensional space, such

that any two of these points are far apart. In particular the construction of good codes is related to the problem of constructing efficient packings of spheres in higher-dimensional space.

Sphere Packing

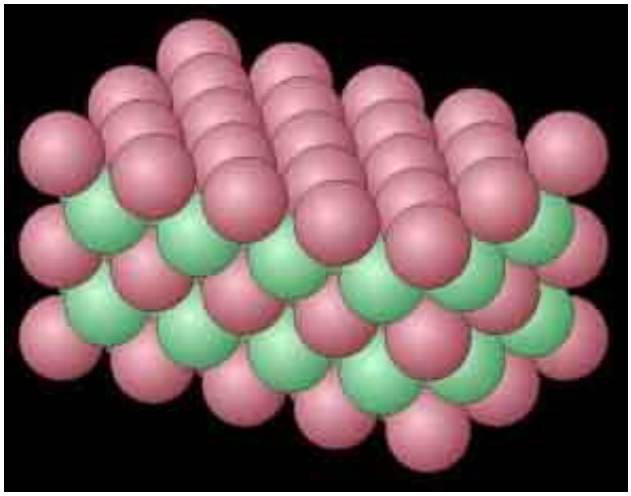
It has long been recognized that the densest possible packing of disks of equal area in the plane, is the packing seen in Figure B:

Figure A:
Loosely packed pennies

Figure B:
Densely packed pennies

We require that the disks do not overlap, and we want to fill as many as possible into a given large plane region. There is a similar familiar lattice packing of equal-sized balls in 3 dimensions, shown in Figure C:

Figure C:
Lattice packing
in 3 dimensions



It was shown only recently (by Thomas Hales, in 1997) that this packing is in fact the densest possible packing of space by equal-sized balls. For every $n = 1, 2, 3, \ldots$,

Thomas Hales

we may ask what is the densest possible packing of equal-sized balls in Euclidean space of n dimensions. For $n > 3$ this problem is open, but the problem is intimately related to the problem of constructing good error-correcting codes.

We explain the connection between sphere-packing and construction of good codes, using the Hamming code of length 7 as an example. In this case we may view the codewords as points in a 7-dimensional space, albeit a *discrete* space with coordinates 0,1 rather than real number coordinates (so that this space has only 2^7=128 points in all). Note that points in this space are the same as vectors, or bitstrings, of length 7. Any two distinct Hamming codewords differ in at least three positions (in fact, the number of coordinates in which they differ is always 3, 4 or 7). It is this property of the Hamming code that guarantees that single bit errors are correctible. Heuristically, the fact that the codewords are far apart means that they are not easily confused with each other.

The Hamming code described above is the consequence of a surprisingly dense packing of balls in 8-dimensional Euclidean space. To construct this packing, first add a parity check to each of the Hamming codewords from Table A, to obtain bitstrings of length 8, each having an even number of 1's. This gives the list of bitstrings in the table at the right. Now consider all the points $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ having *integer* coordinates x_i, such that the pattern of even and odd coordinates conforms to the table at the right (0 = even, 1 = odd). For example, the point $(2, -1, 0, 6, -8, 3, -5, 7)$ is such a vector (with pattern of even and odd given by 01000111). We then take these points as the centers of spheres in Euclidean 8-space, of radius 1. This gives the densest packing of spheres known in eight dimensional Euclidean space. In this arrangement, every sphere touches exactly 240 others! (Compare this with the packings in 2 and 3 dimensions, in which each sphere touches exactly 6 or 12 others, respectively.)

00000000
00011011
00101110
00110101
01000111
01011100
01101001
01110010
10001101
10010110
10100011
10111000
11001010
11010001
11100100
11111111